Copyright 2026 Synthiam Inc.

Using ARCx

Getting Started

Learn how to build a robot with ARCx, Synthiam's robot programming platform. This beginner's guide takes you from choosing a robot controller, motors and a camera to driving your robot, adding voice commands and controlling it from your phone. You can get movement and live video working through robot skills and their settings before learning to write scripts.

Your first goal is simple: **connect the robot, move it deliberately, stop it, and see through its camera**. Add speech, remote control and autonomous features one working step at a time.

Your path to a working robot

1. Choose your robot hardware.
2. Install ARCx and create a project.
3. Connect your EZB controller.
4. Use the robot designer to map your ports.
5. Choose a movement panel and drive.
6. Add a camera and view live video.
7. Add a speech listener and phrase scripts.
8. Add joystick or phone remote control.
9. Grow into AI, navigation and talking robots.
10. Start and stop your robot each session.

How ARCx works

ARCx runs on a **robot computer**: the computer that runs the software and communicates with your hardware. That computer can sit on the robot or nearby. You use ARCx through a web browser on that computer, or through a phone, tablet or another computer configured to reach it.

Term	What it means
EZB controller	The hardware interface between ARCx and the robot's servos, motors and sensors. Use a controller supported by ARCx, with the appropriate firmware and connection method.
Port	A numbered connection on a controller. Settings must refer to the port where the part is actually wired.
Robot skill	An add-on that provides a capability, such as a camera source, movement panel, speech recognition or joystick control.
Project	Your saved robot setup: skills, their settings, scripts, robot port map and custom screens. ARCx project files use the `.arcx` extension.
Workspace	A page where you arrange and use robot skill windows.
Script	A sequence of instructions, such as stopping the wheels or playing a greeting. Buttons,

	spoken phrases and other skills can run scripts.

Robot skills work together. A joystick, a voice command and a navigation skill can all send movement commands through the robot's movement panel. A camera source supplies video to viewers and vision skills. A speech listener converts sound into text; a phrase recognition skill decides which script that text should run.

```
Phone or computer browser --> ARCx running on the robot computer
                                        |
                        Robot skills and your project
                                        |
                           EZB controller --> Robot hardware

Microphone --> Speech listener --> Phrase recognition --> Script
Joystick / phone controls / script --> Movement panel --> Wheels or legs
Camera --> Camera source --> Live view / vision skills
```

1. Choose your robot hardware

For a first project, a small wheeled robot is easier to set up than a walking robot. Choose compatible parts before wiring them together; the movement panel and camera skill you use later depend on these choices.

Part	What to decide
Robot computer	Choose a computer covered by the installation guides. It runs ARCx throughout your session.
EZB controller	Check that ARCx supports the board, its firmware, the connection method and the ports your parts need.
Frame or chassis	This holds the controller, battery, wheels and other parts. An existing robot kit can provide this structure.
Motors and wheels	DC gear motors need a suitable motor driver, often called an **H-bridge**. Continuous rotation servos are another way to drive wheels.
Position servos	These move to a chosen position, useful for a head, arm or camera mount. They behave differently from continuous rotation servos, which spin for wheel movement.
Camera	Choose a USB camera connected to the ARCx computer, or a camera supported by a video-capable EZB.
Power	Match the controller, motors, servos and driver to an appropriate battery or supply, including their voltage and current requirements.
Microphone and speaker	Add these for speech recognition and spoken responses. Start with audio devices attached to the ARCx computer.

Remote control	Use a compatible game controller, or build a touchscreen interface for your phone.

Follow each hardware manufacturer's wiring instructions. A motor driver handles the power needed by DC motors; do not wire a DC motor directly to a controller's signal pin. Power off before changing wiring, and check polarity and any required shared ground connections. Write down which controller and port each part uses. Keep the robot supported with its wheels clear of the floor for the first movement tests, and keep its power switch within reach.
Ready to continue: You know how the robot will move, how its controller connects, and which camera you will use.
[opt:hardware]

2. Install ARCx and create a project

1. Follow the guide for Windows, Raspberry Pi or Linux x64.
2. Start ARCx. On Windows, use the **ARCx** shortcut. On Linux, use the launch instructions in your installation guide.
3. Open `http://127.0.0.1:5000` in a browser on that computer if the interface does not open automatically. Leave the ARCx console running.
4. Complete the initial setup and sign in to your Synthiam account when prompted.
5. Use **File > New** to create your robot project. Save any existing work before replacing an open project.
6. Use **File > Save** and give the project a recognizable name, such as `My First Robot`.

To add capabilities, select **Robot Skills > Add**, find the skill you need, and add it to your workspace. Open a skill's configuration using its settings control. Read its help for hardware-specific requirements, then save your project after configuring it.
You will use this same **add, configure, test, save** sequence throughout the guide. Skills may need downloads, models or service credentials; their instructions explain those requirements.
Ready to continue: ARCx opens in your browser and you have a saved project.

3. Connect your EZB controller

1. Prepare the controller using its setup instructions, including any required firmware and USB or network setup.
2. Power the robot controller and connect it to the ARCx computer using the controller's supported method.
3. Find **EZB Connection** in the workspace. If it is missing, add it through **Robot Skills > Add**.
4. For your first controller, use **EZB 0**. The number identifies the controller within your project; it is separate from the port number on that board.
5. Select or enter the controller's **Address / port**. Use the address or serial port provided by your controller's setup, rather than copying an address from a different robot.
6. Press the connect button and confirm the controller shows as connected.

Serial controllers may need connection settings such as a baud rate in the skill's configuration. Match these to the controller's instructions. Add further EZB entries only when your robot needs more controllers.
Connecting establishes communication. The movement, servo and camera skills still need their own configuration.
Ready to continue: EZB Connection reports that your controller is connected.

4. Use the robot designer to map your ports

ARCx's robot designer is called **Robot Configuration** in the interface. It turns your wiring notes into a visual map so you can select a named part instead of remembering its port number.

1. Open **Project > Robot Configuration**.

2. Give the robot a name.
3. Select **Upload a picture**. Use a clear photograph or a drawing saved as an image. A simple labeled sketch is enough.
4. Under **Add a port**, choose the type of connection you want to describe, then click where the part belongs on the picture.
5. Enter a helpful **Name**, such as `Head tilt`, and a **Description** with wiring notes.
6. Set **EZB Index** and **Port** to match the physical wiring. Repeat for the ports your robot uses.
7. For position servos, configure appropriate **Hard limits** and enable **Enforce hard limits on this servo** to keep requested positions within the robot's mechanical range.
8. Use **File > Save** to keep the picture and port definitions.

For example, if your head-tilt servo is physically connected to D2 on EZB 0, map that servo to EZB index `0`, port `D2`, and name it `Head tilt`. This is an example, not a required wiring arrangement.

Adjusting a servo's hard-limit sliders can move the connected servo. Support the mechanism and make small adjustments within a known safe range. Continuous rotation wheel servos need their stop and direction settings configured in their movement panel.

Port names appear in ARCx's port selectors, and the picture helps you select parts visually. You must still choose the correct ports in each robot skill's settings; drawing a marker does not wire or configure a motor driver for you.

Ready to continue: Your port map matches your wiring, with clear names and suitable position-servo limits.

5. Choose a movement panel and drive

A **movement panel** translates commands such as Forward, Reverse, Left, Right and Stop into the actions your particular robot needs.

How your robot moves	Movement panel to look for
Two driven wheels using continuous rotation servos	**Continuous Rotation Movement Panel**
DC motors controlled through a compatible H-bridge	**Hbridge Movement Panel**
Legs driven by servo pose sequences	**Auto Position (movement Panel)**; this also requires configured walking animations

1. Add the movement panel that matches your hardware. Start with one movement panel for the robot's drive system.
2. Open its configuration and choose the controller and ports from your wiring map.
3. Configure the panel's direction, stop and speed settings as appropriate. An H-bridge may require separate direction and speed-control ports; follow its wiring instructions.
4. With the wheels raised, test **Stop**, then briefly test each direction at a low usable speed. Press **Stop** after each test.
5. Correct any reversed direction or wheel assignment before placing the robot on the floor.
6. Test on a clear, level surface, then save the project.

For an arm, head or other individual joint, add a **Servo** skill and select the appropriate position-servo port. A movement panel controls travel; a servo skill lets you test a particular joint.

Later, your joystick, phone controls and voice scripts can use the same movement system. Get the panel working directly before adding those controls.

Ready to continue: The robot moves in the expected directions and stops reliably from the panel.

6. Add a camera and view live video

1. Add **Camera Source (usb)** for a USB camera attached to the ARCx computer, or **Camera Source (ezb)** for a supported EZB camera.
2. Select the USB camera in the skill's configuration, or provide the EZB camera address in its camera controls.
3. Start the camera using the skill's start control.
4. Confirm that the live image changes when you move something in front of the lens.
5. Save the project.

The camera source provides the live view and makes its video available to other skills. When you later add camera widgets, recording or vision skills, select this source in their settings. A USB camera is connected to the computer running ARCx. Opening ARCx on a phone does not automatically make the phone's camera the robot's camera.

Ready to continue: You can see a live image from the robot's camera.

7. Add a speech listener and phrase scripts

Robot voice control has two parts:

Part	What it does
Speech-to-text engine, or listener	Listens to microphone audio and turns speech into recognized text.
Speech Phrase Recognition	Matches that text to phrases you define and executes the associated scripts. This is your phrase engine.

Set up the listener

1. Connect and check the microphone on the ARCx computer.
2. Add a speech-to-text skill suitable for your system. Examples include **Windows Speech Engine** for Windows or **Sherpa-Onnx Speech To Text** for local model-based recognition.
3. Follow the engine's configuration instructions for language, audio input and any required model downloads or credentials.
4. Configure its listening mode, including a wake word if the selected engine supports one. A wake word tells the listener when you want to address the robot.

Connect phrases to actions

1.
Add **Speech Phrase Recognition** and open its configuration.

2.
In **Speech To Text Engine**, select the listener you added.

3.
Use the **+** control in the phrase table to add a phrase such as `robot stop`.

4.
Open that row's script editor. For a first JavaScript example, select JavaScript and enter:
`Movement.Stop();`

5.
Save the script and skill settings, and turn on **Enabled** in the Speech Phrase Recognition window. Make sure the listener is not paused.

6.
Say the phrase, using the wake word first if configured. Check the phrase log and confirm the action runs.

7.
Save the project, then add more phrases one at a time.

A script is simply the action attached to the phrase. Use the script editor's command and API tools to build actions for your installed skills. Start with a stop command or a small, already-tested gesture before adding spoken driving commands.
Use distinct phrases and test them in the room where the robot will operate. Keep a manual Stop control available; microphone noise or a missed phrase can prevent a voice command from being recognized.
Ready to continue: Your listener recognizes a phrase and Speech Phrase Recognition runs its script.

8. Add joystick or phone remote control

Option A: Drive with a joystick or game controller

1. Connect or pair the controller with the **computer running ARCx** using USB or Bluetooth.
2. Add the **Joystick** robot skill.
3. Select your controller in its configuration and check that its live inputs respond.
4. Configure the driving controls to send movement commands. You can also assign buttons to scripts and sticks to servos.
5. Test direction, stopping and pause behavior with the wheels raised before driving normally.
6. Save the project.

The Joystick skill supplies the driver's input; your movement panel still handles the motors.

Option B: Build a phone control panel with Custom UI

Custom UI means a custom user interface: your own screen of buttons, joysticks, sliders and camera views.

1.
Open **Workspaces > Custom UI > Manage Custom UI**.

2.
Select **Add screen**, name it `Robot Remote`, and choose **Portrait 390 x 780 (phone)** or another suitable layout.

3.
Select **Design** and add widgets from the palette: movement buttons or a movement joystick, a camera view, and any servo or script buttons you need.

4.
Configure each widget's properties. Select your camera source for the camera widget, the correct ports for servo controls, and scripts for action buttons.

5.
Include an easily reached Stop control. Use **Preview** or **View** to test the screen, then save the project.

6.
Enable access from another device using your installation guide. ARCx listens only on its own computer by default; the launch option for local-network access is:
`-b http://0.0.0.0:5000`

7.

Restart ARCx with that option, reopen your project and reconnect the robot. Put the phone on the same trusted network and browse to `http://<robot-computer-ip>:5000`, replacing the placeholder with the ARCx computer's actual address.

8.
Open your custom screen from **Workspaces > Custom UI**. Bookmark its viewer page for the next session.

`127.0.0.1` on your phone refers to the phone itself, so use the robot computer's network address. Follow the platform guide's firewall instructions. Keep remote access on a trusted network: the browser interface can control the robot, and HTTPS alone does not add a login. **Ready to continue:** You can drive, stop and view the camera from your chosen remote control.

9. Grow into AI, navigation and talking robots

Once manual control works, add one capability at a time. Reuse the movement panel, camera source and named ports you have already tested.

Next capability	What to explore	What to prepare
Text-to-speech (TTS)	A speech synthesis skill so the robot can say greetings, status messages or responses	A working speaker and the chosen engine's requirements. **Basic Speech Synthesis** uses Synthiam's cloud service and requires a Synthiam login and internet connection.
Robot vision	**Face Tracking** or **Object Training And Tracking**	A working camera source; configure any tracking servos and their limits before enabling motion.
Servo animations	**Auto Position** for gestures and coordinated poses	Tested servo ports and limits; build a small motion before combining many joints.
AI robot skills	**Autonomous Ai** for an AI-driven robot personality and behaviors	Configure the chosen AI service or local endpoint, then connect the camera, speech and actions you want it to use. Provider requirements and costs vary.
Robot navigation	**Visual Waypoint Navigation** for learning camera-based locations and routes	A forward-facing camera, working movement panel and a clear test area. Follow the skill's route-training instructions.
Custom robot behaviors	**Workspaces > Blockly**, scripts and script collections	Begin with a short sequence of actions you have already tested individually.

Speech recognition lets the robot **hear**; text-to-speech lets it **talk**. Configure audio output to the computer's speaker or a supported EZB speaker as appropriate. Keep the microphone from repeatedly recognizing the robot's own speech.

Navigation and AI can issue movement commands. First supervise short tests in an open area with manual stopping available. Add and test any required sensing separately; a camera or AI skill alone does not establish that the robot will avoid obstacles.
When you are ready to build your own add-on, see Create Your First Robot Skill.

10. Start and stop your robot each session

After the initial setup, use this routine:

1. Check the robot's battery, wiring and space around it.
2. Start the robot computer and ARCx, then open the browser interface.
3. Open your saved `.arcx` project through **File > Open**. The installation guides also explain the `-f` option for opening a project at launch.
4. Power and connect the EZB controller. Check any connection scripts you have configured, because they may run as soon as the controller connects.
5. Start or verify the camera, listener and other skills you intend to use. Loading a project and connecting the controller are separate steps; startup behavior depends on each skill's settings.
6. Check manual Stop and basic movement before enabling voice-driven motion, navigation or autonomous AI.
7. Open your joystick controls or Custom UI and use the robot.

When finished, stop movement and disable autonomous behaviors, save your project, and power down the robot hardware according to its instructions. Press **Ctrl+C** in ARCx's console to shut down the application; allow time for it to finish.
Closing the browser or locking your phone does not stop ARCx or guarantee that the robot stops. Stop the robot explicitly before leaving it.

Troubleshooting your first robot

Problem	What to check first
The browser cannot open ARCx	Is the ARCx process running? On its own computer, try `http://127.0.0.1:5000`.
EZB will not connect	Check controller power, firmware, address or serial port, network connection and controller-specific settings.
A servo or motor does not move	Check the EZB connection, power, controller index, physical port and skill configuration. A named port marker alone does not configure the skill.
The robot turns when asked to go forward	Check left/right assignments and direction settings in the movement panel. Retest with the wheels raised.
The camera has no live picture	Check the selected USB device or EZB camera address, then verify that the camera source is started.
Speech is heard but no action runs	Check the selected engine in Speech Phrase Recognition, its Enabled switch, phrase text, confidence setting and script. Test the script separately.
The listener hears nothing	Check the microphone, audio input settings, model or service setup, pause state and wake-word configuration.

The joystick appears but cannot drive	Check its driving configuration and confirm that the movement panel works directly.
The phone cannot connect	Check the bind option, computer IP address, shared network and firewall. Use the computer's IP, not `127.0.0.1`.
Changes are missing after reopening	Use **File > Save** after editing skills, the robot map or Custom UI screens. Confirm that you reopened the same project.

Use **Workspaces > Debug Log** and the affected skill's own log for error details. Fix the first failing step before adding another capability.

What Is ARCx

The next generation of robot programming software

ARCx is a cross-platform, browser-based robot programming and control platform developed by Synthiam. It enables hobbyists, educators, researchers, and commercial developers to build intelligent robots without having to program every component from scratch.

ARCx builds on Synthiam's original ARC software, but introduces a fundamentally different architecture designed for modern robotics, embedded computers, AI, and remote operation.

The biggest difference is that ARCx runs directly on the robot while its programming interface runs in a web browser. This allows you to configure, program, monitor, and control a robot from a laptop, tablet, or phone without interrupting its operation.

What makes ARCx different?

Browser-based programming

Access the robot through its IP address using any modern browser. The browser is only the interface; robot processing continues on the host computer even after you disconnect.

Cross-platform operation

Designed for Windows x64, Linux x64, and Linux ARM64, including Raspberry Pi and compatible single-board computers. ARCx can run without a monitor or desktop environment.

Modular Robot Skills

Add capabilities such as computer vision, servo control, speech recognition, navigation, and AI through independent plugins. Skills can be loaded and unloaded while ARCx is running.

ARCx's robot skill framework has been enhanced to enable advanced robot skills using machine learning with ONNX, CUDA, and other frameworks.

Integrated AI assistance

Athena, Synthiam's AI assistant, helps users understand, configure, and program robots. It can work with project settings, scripts, variables, and Robot Skills, with approval required for modifications.

Multiple simultaneous connections

Several browsers can interact with the same running robot, making it useful for classrooms, development teams, and remote robotics applications.

How ARCx works

Browsers provide the user interface. ARCx executes robot logic and manages hardware independently.

For example, a Raspberry Pi running ARCx can control a robot's servos, cameras, and sensors. You connect to the Raspberry Pi from your laptop using a browser, configure its behaviors, and then close the laptop. The robot continues operating.

That architecture is particularly valuable for autonomous robots and industrial deployments.

Programming and capabilities

ARCx supports three main programming approaches:

- Blockly: Visual drag-and-drop programming for beginners and education.
- JavaScript: Scripting for robot behaviors, automation, and advanced control.
- Python: Programming for developers who prefer Python.

Robot Skills communicate through `ControlCommand()`, including commands that return values directly to scripts.
Its broader capabilities include OpenCV-based computer vision, face and color tracking, depth sensing, audio input/output, speech recognition and synthesis, servo control, gait programming, and support for multiple robot controllers.

ARC vs. ARCx

Feature	ARC	ARCx
Operating systems	Windows	Windows, Linux, ARM64
Interface	Windows desktop	Web browser
Headless operation	Not the primary design	Native
Multiple browser clients	No	Yes
Robot Skills	Modular plugins	Hot-loadable plugins
AI assistant	Yes	Project-integrated Athena
Embedded deployment	More limited	Designed for SBCs
UI technology	Windows desktop	Blazor

Technical architecture

ARCx is built around modern .NET technology, with the following architecture:

Component	Technology
Runtime	.NET
Web interface	Blazor
Live communication	SignalR
Plugins	.NET AssemblyLoadContext
Computer vision	OpenCV
Project security	RSA-based encryption
Distribution	Cross-platform executables
Default web port	5000

An important architectural decision is separating each Robot Skill's background worker, user interface, and configuration. This allows robot behavior to continue independently of browser interaction.

Development status

ARCx had its first public release in September 2026, and development has continued through October with workspace improvements, depth sensing, and reliability enhancements.

Why ARCx matters

Traditional robot software often requires developers to build separate systems for motor control, sensors, vision, speech, AI, and remote interfaces.

ARCx brings these technologies together in a single extensible environment. Instead of spending weeks integrating libraries, developers can combine existing Robot Skills and focus on the robot's actual behavior.
The long-term value of ARCx is that it makes the computer inside a robot into a remotely programmable, extensible robotics platform. That is useful for everything from a Raspberry Pi-powered hobby robot to a sophisticated humanoid or commercial service robot.
You can learn more at synthiam.com/Products/ARCx.

Robot Skills

What Is A Robot Skill

A robot skill is an add-on for ARCx that gives your robot a particular ability. It might connect a camera, understand spoken words, control a motor, read a sensor, or provide a way for someone to interact with the robot.

You build a robot project by choosing skills and making them work together. Each skill handles a part of the job, so you can concentrate on what you want the robot to do without creating every piece yourself. A simple project might use just a few skills; a more ambitious robot might combine vision, speech, movement, and automation.

This guide explains what skills are for and the parts you will encounter when using or creating one. You do not need programming experience to follow it.

Giving a robot abilities

Physical hardware gives a robot its possibilities: cameras can capture images, microphones can capture sound, and motors can move things. Software turns those possibilities into useful behavior. Robot skills provide pieces of that software inside ARCx.

For example, a camera skill can make a camera's video available to a project. A vision skill could examine those images for an object. A movement skill could then help the robot turn toward it. Connecting those abilities creates a behavior that is more useful than any individual part.

Skills can also work entirely with information or user interaction. A skill might display a dashboard, connect to an online service, organize a sequence of actions, or let someone control the robot with a gamepad. The word "skill" covers both visible actions and the supporting abilities that make them possible.

The exact features depend on the skill. Some need particular hardware, an internet connection, an account with another service, or a compatible operating system. A skill's description and instructions explain what it needs.

Why use robot skills?

Robot skills make capabilities reusable. Once someone has created a skill for a device or behavior, other people can add it to their projects and configure it for their own robots. This helps in several ways:

- **Start with something that already works.** An existing skill can provide the controls and device communication you need to begin experimenting.
- **Choose the abilities your robot needs.** You can combine skills around a particular purpose, such as a talking companion, a mobile robot, or an interactive exhibit.
- **Keep complicated work manageable.** Camera handling, speech, movement, and other jobs can each have their own settings and controls.
- **Improve one part of a project.** A skill can gain features or fixes that benefit projects using it.
- **Reuse ideas.** The same kind of skill can be useful on many different robots, with different settings in each project.

ARCx provides the surrounding workspace and shared facilities. Skills bring the individual capabilities into that environment.

Why do people create them?

People often create a robot skill because they have an idea that existing skills do not quite cover. They might have a new piece of hardware, a particular way of controlling their robot, or a service they want to connect to.

A hobbyist might make a skill for a sensor they just bought. A teacher might create an easy control panel for a classroom activity. A hardware maker might provide a skill so customers can use a product in ARCx. Someone building an exhibit might package a repeated routine

into a few clear controls for its operators.
Creating a skill also turns a solution into something easier to use again. A capability that previously required repeated setup can have a name, a settings page, useful controls, and instructions. Its creator can keep developing it for their own projects or share it through the Robot Skill Store.
You do not have to publish a skill to make creating one worthwhile. A skill built for one robot can still make that robot easier to configure, operate, and maintain.

The main parts of a skill

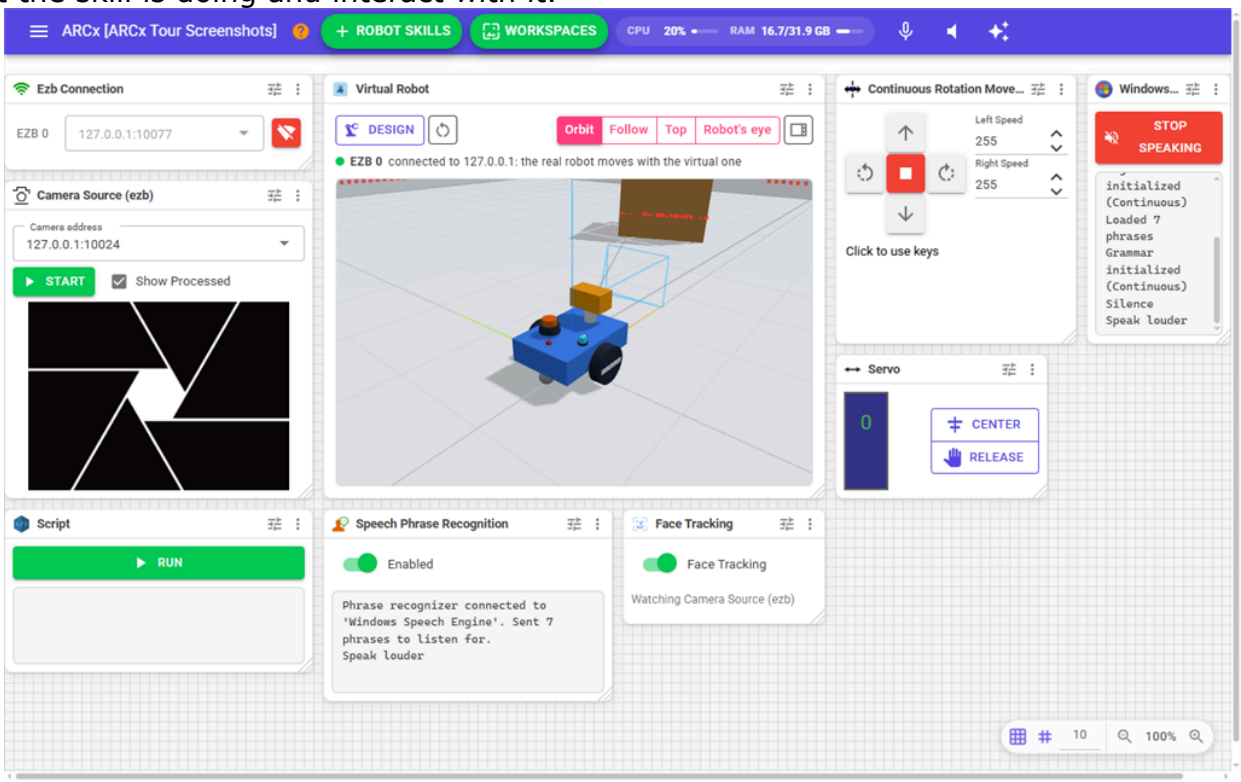
A skill combines the screens people interact with and a **worker** that carries out its behavior. The screens have different purposes, but they belong to the same capability.

Part	What it is for	Example for a distance-sensor skill
Main window	Everyday controls and live information in the ARCx workspace	Show the current distance and whether the sensor is connected
Configuration screen	Set up how the skill should work with your robot	Choose the sensor connection, units, and alert distance
Custom UI widget	Present a focused view on a custom control panel, when provided	Show a large distance reading and a nearby-object indicator
Worker	Perform the skill's work and maintain its current state	Read the sensor, check the distance, and make the result available

The amount of interface varies. Some skills need a rich display with many controls; others mostly work in the background and have very little to show.

The main window: use and observe

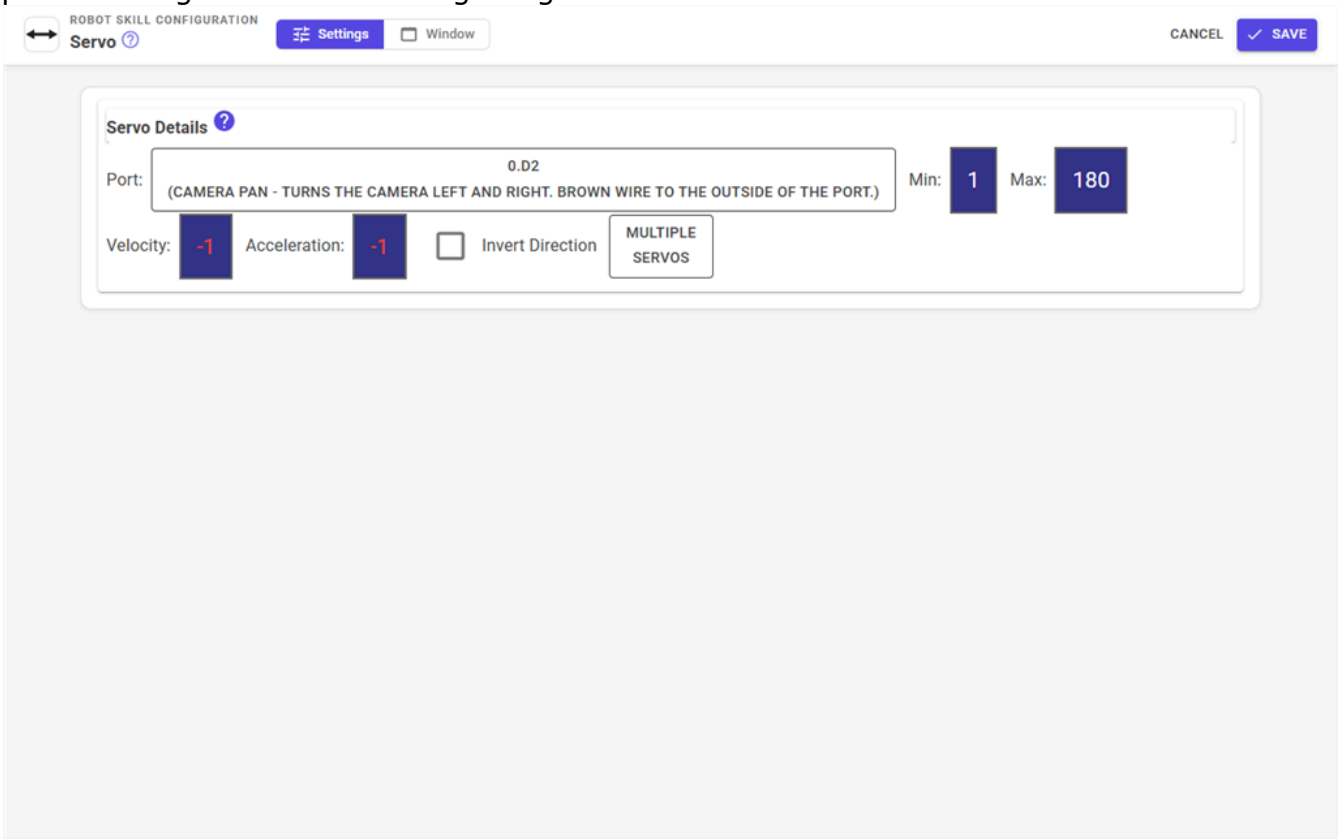
The main window is the skill's everyday presence on the ARCx workspace. It helps you see what the skill is doing and interact with it.



Depending on the skill, you might see a camera image, a connection button, a microphone level, a position control, a map, or a list of recent events. The window can help answer questions such as "Is it connected?", "What is it detecting?", and "What is happening now?" This is also where a creator can make a complicated capability approachable. A useful display and a few clearly named controls can make the skill much easier to understand and try.

The configuration screen: adapt it to your robot

The configuration screen holds choices that determine how the skill behaves. You normally open it through the skill's settings or gear control.

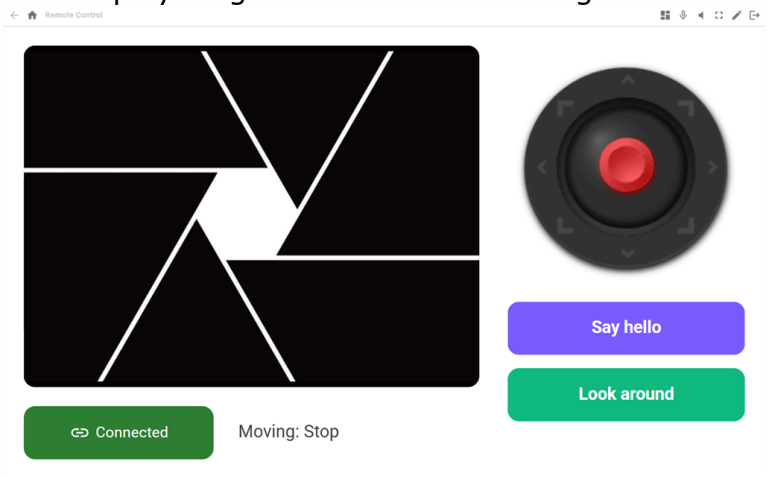


For a hardware skill, you might choose a connection or a port. For a speech skill, you might choose a voice or phrases to recognize. For an automation skill, you might select the conditions that trigger an action.

These settings let the same skill serve different projects. Two people can use the same sensor skill while choosing different connections, units, or thresholds. The skill's project settings are saved with the project so they can be restored when it is opened again.

The Custom UI widget: a view for the operator

A Custom UI is a control panel you arrange for the person using the robot. It can bring the most useful controls and displays together on a screen designed around a task.



A skill can provide a widget for that screen: a compact presentation of some part of its capability. A speech skill might offer a listening indicator; a sensor skill might offer a gauge; a camera skill might offer a video view. A skill can provide more than one widget for different purposes.
This allows a project builder to work with detailed skill windows while an operator uses a simpler panel. Both views connect to the same underlying skill. If a skill does not provide a dedicated widget, its main view can be used on a Custom UI screen.

The worker: carry out the behavior

The worker is the part of the skill that runs inside ARCx and does the work. It can communicate with hardware, process information, respond to commands, keep track of the current state, and coordinate actions.
For a distance sensor, the worker might read measurements and check whether an object is close. The main window displays those measurements, the configuration screen lets you choose what "close" means, and a widget shows the result on your control panel.
Keeping the worker separate from its screens lets a skill operate without requiring someone to keep its window visible. Depending on how the skill is designed, it can continue its work while you view another page or while no browser is connected, as long as ARCx and the project remain running.
It also means that opening the same project from another browser gives you another view of the running skill. It does not create a second worker for that skill instance. The connected screens interact with the same underlying state. Closing a browser is therefore not a general way to stop the robot's behavior; use the controls provided by the skill or project.

How skills work together

A project becomes more capable when its skills exchange information and respond to one another. Depending on what they support, skills can provide readings, share video, respond to commands, or take part in scripts and automated routines.
Imagine a robot that greets someone who approaches it:

1. A sensor skill detects that someone is nearby.
2. A project routine checks that it is time to greet a visitor.
3. A speech skill says hello.
4. A movement skill performs a wave.
5. A display shows a welcome message.

Each part has a clear job. The project supplies the coordination that turns those jobs into a complete interaction. The skills need to offer the appropriate connections or commands, and the project builder sets up how they work together.
Some skills allow you to add more than one copy to a project. For example, you might use separate copies for two sensors, each with its own settings and worker. Other skills allow only one copy because of the role they perform.

From finding a skill to using it

You can browse the Robot Skill Store from **Robot Skills > Add** in ARCx. Read a skill's description to understand its purpose, requirements, and available instructions, then add it to your project.
A typical next step is to configure it for your hardware or intended behavior, try its controls, and check the information it displays. Once the skill works on its own, you can connect it to other parts of the project and include its controls in a Custom UI if useful. Save the project to keep your setup.
Skills can receive improvements after you start using them. ARCx can update installed skills at startup and show a notice of what changed. See Updating Robot Skills for how that works.

If you have an idea for a skill

Start by describing the ability you want to give your robot. Think about what the person

using it should be able to see, what they should be able to change, and what the skill should do on its own.
For example: "I want to connect my distance sensor, see its reading, choose an alert distance, and let my project react when something gets too close." That short description already suggests the main window, configuration screen, and worker. A large reading for an operator's panel could become an optional widget.
You can begin with a small, useful capability and expand it as you learn what your robot needs. For a walkthrough of creating a skill with help from an AI coding assistant, see Create Your First Robot Skill.

Updating

ARCx automatically checks for newer versions of installed robot skills when it starts. After updates finish, the first browser to open the main ARCx interface shows a popup listing the updated skills. Once the popup is dismissed, later connections do not show the same notice again.

When updates happen

Automatic updates are enabled by default. In **Options > Preferences**, use **Update robot skills when ARCx starts** to enable or disable them. The underlying setting is `ARCxAutoUpdateRobotSkillsIfAvailable`.
Updates run during application startup, before the web server accepts connections and before projects load, including projects opened with the `-f` command-line option. ARCx waits for the update attempts to finish before continuing. This lets it replace skill files before a project loads their assemblies and potentially locks them.
No browser needs to be connected: updates also work when ARCx starts headlessly. Opening or refreshing a browser page does not trigger another automatic update pass. To check again through this startup process, restart ARCx.

How ARCx finds updates

1. **Check the Robot Skill Store list.** ARCx checks the ARCx store's latest update timestamp and skill count against its cached `AvailablePluginsList.json`. It downloads a new list if the cache is missing, the store timestamp is newer, or the number of skills has changed. The count check also detects skills becoming public or being withdrawn without a version change.
2. **Read installed versions.** ARCx reads the installed skills' `plugin.json` files from its Plugins folder.
3. **Compare matching skills.** A skill is selected when its store GUID matches an installed skill and the store version number is greater than the installed version. This covers installed skills even if the current project does not use them. Equal or older store versions are not installed by this update pass, and it does not install every uninstalled skill in the store.
4. **Check whether installation can proceed.** Automatic installation requires ARCx to be signed in to a Synthiam account. It is also skipped if ARCx detects another running instance, which could be using the same skill files. These reasons are written to the log.

The store list is checked even when automatic skill updates are disabled. Refreshing the list on the Add Robot Skill page refreshes the catalog; it does not run the startup bulk updater.

How an update is installed

ARCx updates the selected skills one at a time:

1. Read the existing version and log the skill name and old/new versions.
2. Download the skill's package using the signed-in account's authentication GUID. Store access and subscription requirements still apply.
3. Buffer the download in memory and check its ZIP signature before removing the installed files. If the store returns an error message instead of a package, ARCx reports that message.

4. Remove the existing skill folder, recreate it, and extract the package into `Plugins/<skill GUID>`. Archive extraction checks that file paths stay inside that destination.
5. Record a successful update, including the skill GUID, name, previous version, new version, and UTC update time.

This replaces the installed skill package. All projects using that installed skill subsequently load the updated files.

If an individual update fails, ARCx logs the error and continues with the other skills. Only successful updates are added to the update notice. A failed download leaves the existing installation untouched. Replacement is not transactional: a failure after folder removal can leave an incomplete installation, and an extraction failure removes `plugin.json` so ARCx does not treat the partial package as installed.

The updated-skills popup

The notice appears on the first render of the main ARCx layout, after sign-in, terms, and subscription checks have passed. It lists:

- The skill's thumbnail and name.
- The previous and new versions, such as `v3 -> v5`.
- The update date, displayed in local time.
- A **What's new** button that opens the skill's Synthiam page in a new tab to read its changes.

The most recently updated skills appear first. The custom UI viewer used as a robot control panel does not use this layout, so opening that viewer does not display or acknowledge the notice.

Shown once after dismissal

The notice is shared by browsers connected to that ARCx instance; it is not tracked separately per browser, tab, project, or connecting user. Closing it with **Got it**, the close button, Escape, or another dialog dismissal marks the saved list as viewed. Subsequent connections and restarts do not show that list again. A later successful startup update creates a new notice.

The acknowledgement happens when the dialog closes, not when a browser first connects. If a browser disconnects while the popup is still open, the list remains unviewed and can appear on the next connection. If multiple browsers open the main interface before anyone dismisses it, each can see the notice. There is no exclusive reservation for the first connection.

Remembering updates across restarts

ARCx stores the notice in `UpdatedSkills.json` in its user data folder (`Documents/ARCx Data` on Windows, `~/ARCx Data` on Linux and macOS). The file contains a `Viewed` flag and a `Skills` list. It is server-side state, so clearing browser storage does not reset the acknowledgement.

Updates accumulate across restarts until someone dismisses the notice. This is useful when a robot runs without a browser. Each skill appears only once: if it updates from v3 to v4, then from v4 to v5 before anyone views the notice, its row shows `v3 -> v5` with the latest update time.

After dismissal, ARCx retains the list with `Viewed` set to `true`. The next successful startup update replaces the viewed list with a fresh one. If no skills update, ARCx does not create a new notice, although an older unviewed notice can still appear.

If a skill does not update

- Confirm **Update robot skills when ARCx starts** is enabled, then restart ARCx.
- Confirm ARCx is signed in to Synthiam and the account can download the package. Signing in after startup does not rerun the startup update pass; restart after signing in.
- Close other ARCx instances before restarting so they cannot hold the skill files open.
- Check the internet connection and ARCx logs. If the store-list refresh fails, startup

continues with the cache it has, which may not include the newest release. Downloads still require access to the store.
- For file-lock or permission failures, check the logged folder and error. An incomplete installation may need to be downloaded again.

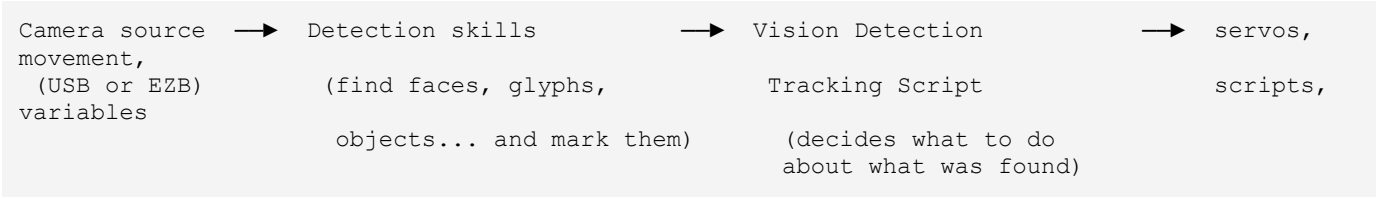
Robot skills are updated when ARCx is loaded and there is an internet connection.

Computer Vision

How To

Camera vision lets a robot see something and act on it: keep a face in the middle of the picture with pan and tilt servos, drive up to a docking marker, or say hello when it recognizes a toy. This page explains how to set it up and what happens between the camera taking a picture and the robot moving.

In ARCx this is never one robot skill. It is three pieces, and each one does a single job:



1. **A camera source** robot skill turns a camera into a named video source.
2. **Detection skills** search each frame for what they know about (faces, glyphs, trained objects) and mark what they find. Detection skills only detect. They do not move servos or drive the robot, and they set no tracking variables.
3. **Vision Detection Tracking Script** watches everything the detection skills find on one camera. It turns that into servo tracking, movement tracking, scripts, variables and control commands, in the same way for every kind of detection.

Keeping detection separate from tracking means you set up tracking once. If you add Face Tracking and Glyph Recognition to the same camera, one Vision Detection Tracking Script can follow faces with the servos, run a script when a glyph appears, and count both.

Contents

1. Quick start
2. Step 1: Choose a video source
3. Step 2: Add detection skills
4. Step 3: Add Vision Detection Tracking Script
5. How it works
6. Example setups
7. Performance
8. Troubleshooting

Quick start

To have a pan/tilt camera follow a face:

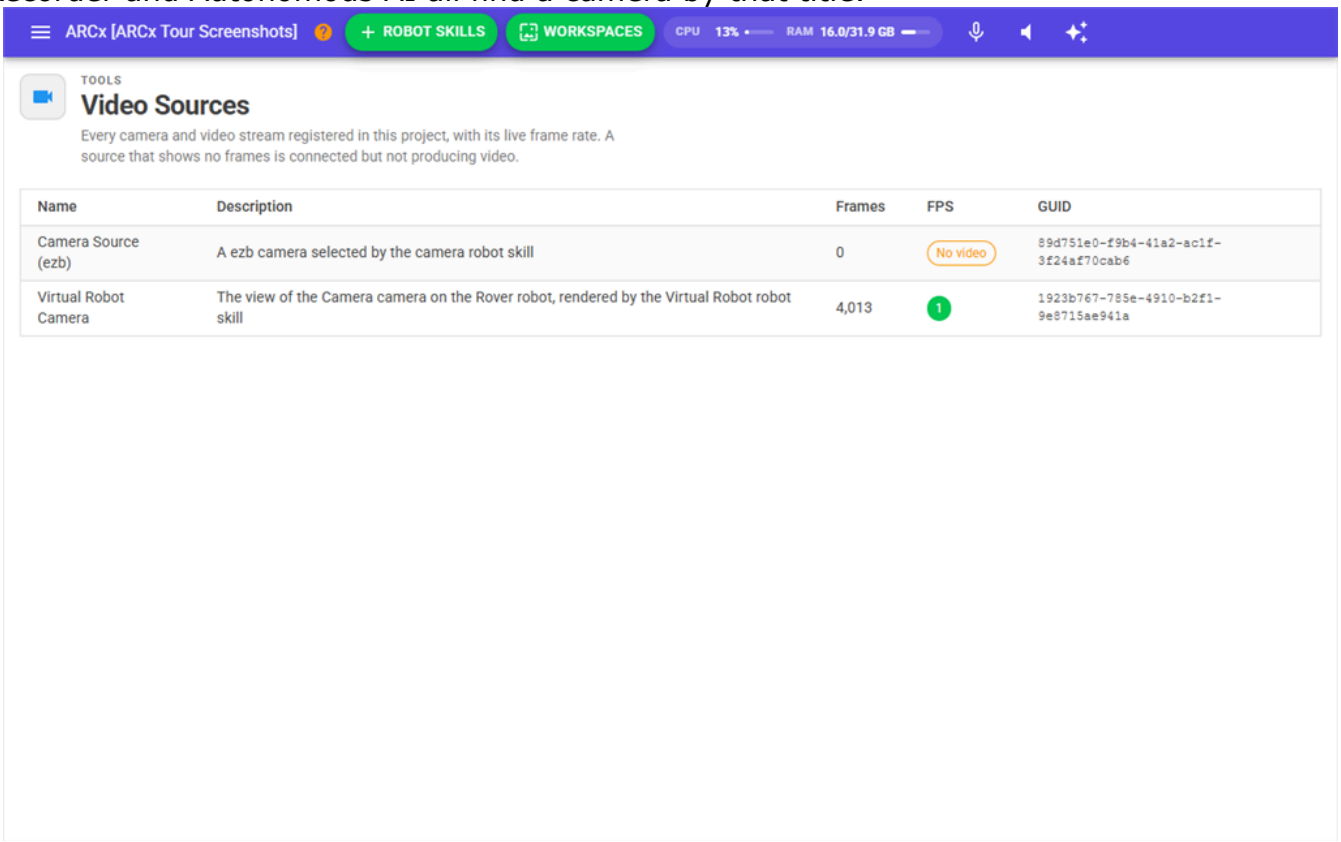
1. **Add a robot skill → Camera → Camera Source (usb)** (or **Camera Source (ezb)**). Choose the camera and press **Start**.
2. **Add a robot skill → Camera → Face Tracking.** It picks the camera by itself. Turn on its **Face Tracking** switch. Faces are now outlined on the video.
3. **Add a robot skill → Camera → Vision Detection Tracking Script.** Open its configuration, choose the same camera on the **Detections** tab, and choose the pan and tilt servos on the **Servos** tab. Save.
4. Turn on **Servos** in the Vision Detection Tracking Script window. Look at the camera and move. The camera follows you.

The rest of this page explains each step and the choices in it.

Step 1: Choose a video source

Every camera in ARCx is a **video source**: a robot skill that captures frames and publishes them under its **window title**. Detection skills, Vision Detection Tracking Script, Video

Recorder and Autonomous AI all find a camera by that title.



Robot skill	Use it for
Camera Source (usb)	Any camera the operating system recognizes: a USB webcam, a capture card, a laptop's built in camera. Pick the **Video Device** and a **Resolution** in its configuration.
Camera Source (ezb)	The camera on an EZB, such as the EZ-B v4 camera. Pick the camera's address in the box at the top of its window. It connects directly to the camera over the network.

Add it from **Add a robot skill → Camera**, then press **Start**. The button shows the frame rate while the camera runs.

Points to know:

- **Give the camera a meaningful window title** such as "Head Camera", and do it before adding the skills that use it. Other skills refer to the camera by title. If you rename it later, update them.
- **Several cameras are allowed**, one camera source skill each. Each detection skill and each Vision Detection Tracking Script watches one camera, chosen in its configuration.
- **Pick a modest resolution.** Detection runs on a smaller copy of each frame (see Performance), so a high camera resolution makes the video sharper but does not make detection better. 320x240 or 640x480 is plenty for tracking.
- **Show Processed** in the camera window switches between the picture straight from the camera and the **processed** picture, which has the boxes and labels that the detection skills draw. Turn it on while setting up, so you can see what is being detected.

Step 2: Add detection skills

A detection skill searches each frame for one kind of thing and marks what it finds on the video. Add one for each kind of thing your robot needs to see:

Detection skill	Finds	Kind	Name	Good for
Face Tracking	Front facing human faces. No training needed.	`face`	`face`	Looking at people, noticing someone is there.
Glyph Recognition	Printed square glyph markers. The skill prints them.	`glyph`	The glyph's name, such as `Glyph 1` or `Dock`	Docking stations, room markers, robot courses. Also reports which way up the glyph is and how far away it is.
Object Training And Tracking	Objects you train by showing them to the camera, or from a picture.	`object`	The name you trained it with	Toys, boxes, book covers: anything with printed or textured detail.

Any other detection skill works the same way. Each one reports its own kind, for example `color` for a colour detector.

To add one:

1. **Add a robot skill → Camera →** the detection skill.
2. Open its configuration and choose the **Video Source**. If you do not, the first camera in the project is used.
3. Turn on its detection switch. What it finds is outlined on the processed video.

You may add several detection skills to the same camera, and Vision Detection Tracking Script sees what all of them find. Every one you add costs processor time on every frame, so add only the ones the robot needs.

What a detection is

Every detection skill reports what it finds in the same form. This is what Vision Detection Tracking Script works with:

Detail	Example
Kind	`face`, `glyph`, `object`, `color`
Name	`Glyph 1`, `Teddy`. A detection with no name of its own, such as an unrecognized face, uses its kind as its name.
Position and size	In camera pixels.
Confidence	How sure the detector is, 0 to 1, when it has a measure of that.
Primary	The detector's main detection, usually its largest.
Details	Anything only that kind has, such as a glyph's `Rotation` or a distance the detector measured.

What detection skills still do themselves

Some detection skills keep features that only make sense for them. For example, Glyph

Recognition can run a script per glyph and draw an overlay picture on it. Object Training And Tracking still has its own scripts and variables for the object it is most sure of.
For anything that moves the robot, use Vision Detection Tracking Script. If you use the variables of Object Training And Tracking **and** Vision Detection Tracking Script, note that both default to the same names (`ObjectIsDetected`, `ObjectName` and so on). Give one of them different names in its configuration, or they overwrite each other.

Step 3: Add Vision Detection Tracking Script

Vision Detection Tracking Script does not detect anything itself. It watches what the detection skills report for one camera and decides what to do about it.

1. **Add a robot skill → Camera → Vision Detection Tracking Script.**
2. Open its configuration. On the **Detections** tab, choose the **Video Source**: the same camera as the detection skills.
3. Optionally narrow what it reacts to with **Kinds** (for example `face`) and **Names** (for example `Dock, Glyph 1`). Leave them empty to react to everything. The kinds and names already seen on that camera are listed below each field. Click one to add it.
4. Set up the reactions you need, described below, and save.
5. Show the camera something. It appears on the map in the skill window, with the **target** outlined in yellow.

The target

Several things can be in view at once. Vision Detection Tracking Script chooses one of them as the **target**: the one the servos and the robot follow, and the one the variables describe. **Choose By** decides which: largest, closest to the centre, most confident, the detector's primary, leftmost, rightmost, or longest in view. **Target Names** gives preferred names in order: `Dock, Glyph 1` follows the dock when it is in view, and Glyph 1 otherwise. **Sticky** stays with the current target while it is in view, so the servos do not jump between two people.

Servo tracking

Choose a **pan** and a **tilt** servo on the **Servos** tab, and turn on **Servos** in the window. The servos turn to keep the target in the middle of the picture. If a servo turns the camera away from the target, tick **Invert Direction** for that servo. When the target is lost, the camera can **scan** from side to side looking for it, or **centre** itself.

Movement tracking

Turn on **Drive** in the window. The robot turns toward the target using the project's movement panel. With **Drive Forward** on, it also drives toward the target until it fills **Stop At Size** percent of the picture, or until it is **Stop At Distance** away when the distance is known. **Back Up When Close** and **Search When Lost** make a robot that keeps its distance from a person and follows them.
The robot stops when the target is lost, when the skill stops, and while any configuration window is open. The panic button turns servo and movement tracking off.

Scripts

Script	Runs
Object Detected	When anything comes into view.
All Lost	When the last thing in view has gone.
Triggers	Your own list: each trigger matches kinds and names, and has a script for when a match comes into view and another for when it goes.

```
// Trigger: kind "face", script when it comes into view
Audio.Say("Hello there");
```

A detector that misses the odd frame would make an object flicker in and out of view, and its scripts would run over and over. **Consecutive Frames** (how many frames in a row before something counts as in view) and **Lost After** (how long it must be gone) prevent that.

Variables and control commands

The target is described by global variables that any script can read: `ObjectIsDetected`, `ObjectName`, `ObjectType`, `ObjectCenterX`, `ObjectHorizontalLocation`, `ObjectHorizontalAngle`, `ObjectDistance`, `ObjectCount` and more. Angles are in radians (positive is left and up), and distances are in metres, like every other position a script sees in ARCx.

```
if (getVar("ObjectIsDetected", false))
  print("I can see " + getVar("ObjectName") + " on the " +
getVar("ObjectHorizontalLocation"));
```

To ask about **anything** in view, not only the target, use its control commands:

```
// Drive forward until the dock glyph is seen, or 10 seconds pass
Movement.forward();
ControlCommand("Vision Detection Tracking Script", "WaitForObject", "Dock", 10000);
Movement.stop();
```

The Vision Detection Tracking Script manual lists every setting, variable and control command.

More than one

You can use one Vision Detection Tracking Script per job. On the same camera, one can follow faces with the pan/tilt servos (Kinds `face`) while another runs scripts for glyphs (Kinds `glyph`). Give each a different window title, because control commands find a skill by its title.

How it works

For every frame:

1. **The camera source** captures a frame and hands it to ARCx's video routing.
2. **ARCx makes a computer vision copy**: a smaller version of the frame, at the width set in **Options → Preferences → Video → Computer vision image width**. Detection skills share this copy, so adding a detection skill does not change the camera's resolution or frame rate, and the copy is only made once however many skills use it.
3. **Each detection skill** watching that camera searches the frame (or skips it, according to its **Frames To Skip**) and adds what it found: kind, name, position, confidence and details.
4. **ARCx collects every detection on the frame** from every detection skill on that camera, and hands the whole list to each Vision Detection Tracking Script watching it. This happens before anything is drawn, so a detection that only lasts one frame is never missed.
5. **ARCx draws the boxes and labels** on the processed picture, which the camera window shows when **Show Processed** is on.
6. **Vision Detection Tracking Script**, for that list:

   filters it to the kinds, names, confidence, size and region you asked for,
   keeps track of what is in view: something counts as in view after **Consecutive Frames** frames in a row, and as lost after **Lost After** milliseconds unseen,
   chooses the target, smooths its position, and works out its angle and distance from the **Camera Field Of View**,

moves the servos and the robot, if tracking is on and the target is outside the **Dead Zone** in the middle of the picture,
runs scripts for things that have just come into view or gone, and updates the variables.

Objects are told apart by kind and name only. Detectors do not give each object an identity, so two people are both `face`: both are counted, but they come into and go out of view together.

Example setups

A head that looks at people. Camera Source (usb) → Face Tracking → Vision Detection Tracking Script with Kinds `face`, pan and tilt servos, **Servos** on, **Scan When Lost** on. Add a trigger for `face` that says hello, with a **Cooldown** so it does not greet the same person every few seconds.
Drive to a charging dock. Camera Source (ezb) → Glyph Recognition, with a glyph named `Dock` printed and stuck on the dock → Vision Detection Tracking Script with Names `Dock`, **Drive** and **Drive Forward** on, and **Stop At Distance** set to the distance where the robot should stop. Glyph Recognition measures the distance itself, so it is accurate.
React to toys. Camera Source (usb) → Object Training And Tracking, with each toy trained under its own name → Vision Detection Tracking Script with Kinds `object` and one trigger per toy name, each with its own script.
One camera, two jobs. Face Tracking and Glyph Recognition on the same camera. One Vision Detection Tracking Script with Kinds `face` keeps the head pointed at people. A second, with Kinds `glyph` and its own title, runs a script whenever a room marker comes into view.
Follow a person around the room. A detector that finds people or faces → Vision Detection Tracking Script with **Drive**, **Drive Forward**, **Back Up When Close** and **Search When Lost** on. The robot follows when the person walks away and backs off when they come closer.

Performance

Vision is the most processor-hungry thing most robots do. In order of how much difference each makes:

- **Only add the detection skills you need**, and turn detection off when it is not needed. A detection skill that is switched off costs nothing.
- **Frames To Skip** in each detection skill. `0` searches every frame. `2` searches one frame in three. Movement and servos seldom need more than ten detections a second.
- **Computer vision image width** in **Options → Preferences → Video**. Smaller is faster. Larger finds smaller and more distant things.
- **Camera resolution** matters least for detection, because detection uses the smaller copy. It still costs processor time to capture and display, so keep it modest.
- **Object Training And Tracking** gets slower with every enabled object. Disable the objects you are not looking for right now.

Troubleshooting

Nothing is outlined on the video. Turn on **Show Processed** in the camera window. Check that the detection skill's switch is on and that its **Video Source** is this camera. Check that the camera is running.
Things are outlined, but Vision Detection Tracking Script shows nothing. Its **Video Source** is a different camera, or its **Kinds**, **Names**, **Min Confidence**, **Min Size** or **Region** filters exclude what is detected. Clear the filters, then add them back one at a time. The **Details** window lists exactly what is being reported.
The servos turn the wrong way. Tick **Invert Direction** for that servo on the **Servos**

tab.
The servos twitch or overshoot. Raise **Smoothing**, lower **Max Step**, or widen the **Dead Zone**.
The target jumps between two objects. Turn on **Sticky**, or name the one you want in **Target Names**.
Scripts run over and over. The object flickers in and out of view. Raise **Lost After** in Vision Detection Tracking Script, or **Consecutive Frames**, before lowering the detector's own thresholds. A trigger's **Cooldown** also helps.
The robot does not drive. Drive is off, there is no movement panel in the project, or a configuration window is open (the robot stops while one is open).
The distance is wrong or -1. Distance is only known when the detector measures it (Glyph Recognition does) or when **Object Width** is set in Vision Detection Tracking Script. Set **Camera Field Of View** to your camera's field of view: most webcams are 55 to 78 degrees, and the EZ-B camera is about 60.
Another skill cannot find the camera. Skills find a camera by its window title. If you renamed the camera skill, choose it again in each skill's configuration.
Variables change unexpectedly. Object Training And Tracking and Vision Detection Tracking Script both use `ObjectName` and the other `Object...` variables by default. Rename one set.
Everything is slow. See Performance.
Each skill's own manual has a troubleshooting section for problems specific to it: glyphs that are not read, objects that will not train, cameras that will not start.

Speech Recognition

How To

Speech recognition lets a robot act on what it hears: "robot move forward", "go to sleep", "what time is it". This page explains how to set it up and what happens between someone speaking and a script running.
In ARCx this is never one robot skill. It is three pieces, and each one does a single job:

```
Microphone  ⟶  Speech to text engine  ⟶  Speech Phrase Recognition  ⟶  your scripts
 (Preferences)   (turns sound into text)    (matches text to phrases)     (do something)
```

1. **The microphone** is chosen once for the whole of ARCx, in Options → Preferences. It can be a sound card, a USB microphone, or the microphone of a web browser.
2. **A speech to text engine** robot skill turns that sound into text. Engines have no scripts of their own. On its own an engine only fills its log.
3. **Speech Phrase Recognition** watches the text that the engine produces. When the text matches a phrase in its list, it runs that phrase's script.

Contents

1. Quick start
2. Step 1: Choose the microphone
3. Step 2: Add a speech to text engine
4. Step 3: Add Speech Phrase Recognition
5. When the robot listens
6. How it works
7. Listening from a script
8. Example setups
9. Troubleshooting

Quick start

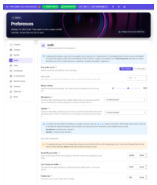
For a robot that responds to spoken commands, with nothing to sign up for:

1. **Options → Preferences → Audio → Microphone**: leave it on **System default**, or pick your microphone.
2. **Add a robot skill → Audio → SherpaOnnx Speech To Text** (any platform), or **Windows Speech Engine** (Windows only).
3. **Add a robot skill → Audio → Speech Phrase Recognition**. It selects the engine by itself.
4. Open the configuration of Speech Phrase Recognition. Edit the phrase table so each row is something you will say and the script to run. Save.
5. Say one of the phrases. It appears in the log of Speech Phrase Recognition and its script runs.

The rest of this page explains each step and the choices in it.

Step 1: Choose the microphone

ARCx has **one microphone for the whole project**. Every robot skill that listens gets its sound from the same place: speech engines, Voice Activity Detection, recorders. Robot skills do not choose their own microphone. You choose it once, here:



Options → Preferences → Audio → Microphone

Choice	What it means
System default	The microphone the operating system is set to use on the computer ARCx runs on. If you change the default in Windows or Linux, ARCx follows it.
A web browser	The microphone of a browser you are using to control ARCx, such as a phone or a laptop. See below.
A named device	A specific sound card, USB microphone or microphone array on the computer ARCx runs on.

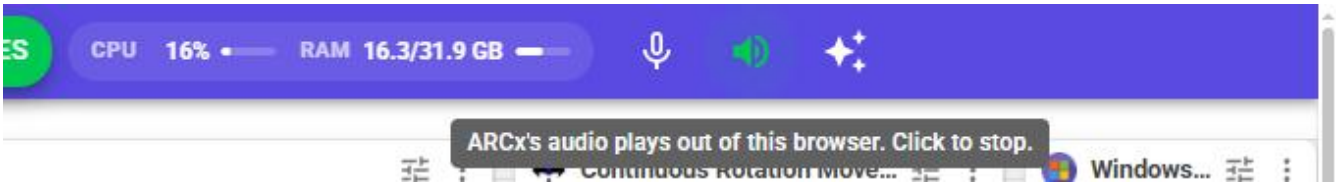
A named device is saved by its name, not its position in the list, so the choice still works after other devices are plugged in or removed. If the device is unplugged, it stays selected and is marked **(not available)**. ARCx uses the next best microphone until the device comes back.

A microphone on the computer ARCx runs on

This is a sound card input, a USB microphone or a microphone array plugged into the computer that runs ARCx. On a robot, that computer is usually the robot itself. Choose **System default** or the device by name.
Speech engines work best with a microphone close to the speaker. A USB microphone array, of the kind sold for conference calls, works much better across a room than a laptop's built in microphone.

A web browser's microphone

ARCx is controlled from a web browser, and that browser can be ARCx's microphone. For example, you can hold up a phone and talk to the robot from across the house. Any number of browsers can have ARCx open, so you choose **which one** by pressing a button in that browser:



1. Set **Microphone** to **A web browser** in Preferences.
2. A microphone button appears in the top bar of every browser that has ARCx open. In the browser you want to talk into, press it. The browser asks for permission to use its microphone; allow it.
3. The button turns green. That browser is now ARCx's microphone.

Points to know:

- **The last browser to press the button has the microphone.** If another browser presses it, that browser takes over and the first one lets go.

- Press the button again to release the microphone. **Until a browser has claimed it, or after it is released, ARCx uses the system default microphone of its own computer.**

- The browser only records while something in ARCx is listening, so the browser's recording indicator is not on all the time.

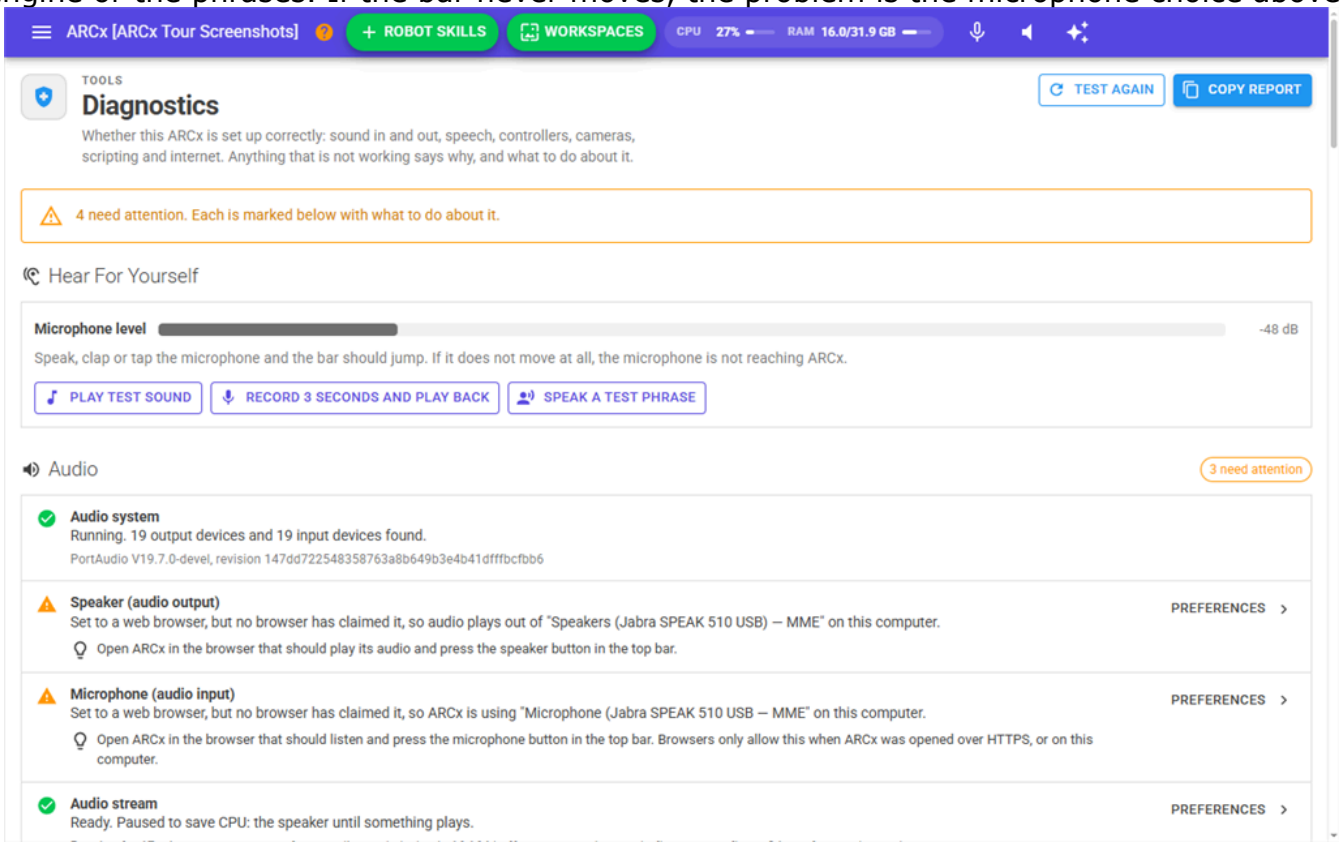
- **Browsers only allow the microphone over HTTPS**, or when the browser runs on the same computer as ARCx. To use the microphone of a phone or another computer, start ARCx on an `https://` address with a certificate:
  `ARCx -b https://0.0.0.0:5001 --cert robot.pfx --cert-password secret`

  See the command line manual for certificates. Preferences shows a warning when the page you are looking at cannot use its microphone for this reason.

The microphone button in the top bar chooses **which microphone** ARCx uses. It is not a button that makes the robot listen. For that, see When the robot listens.

Checking that sound arrives

Every speech engine has a **level bar** in its window that shows the sound reaching it. If the bar moves when you speak, the microphone is working, and any remaining problem is in the engine or the phrases. If the bar never moves, the problem is the microphone choice above.



Tools → Diagnostics also checks sound in and speech, and says what to do about anything that is not working.

Step 2: Add a speech to text engine

A speech to text engine turns sound into text and does nothing else. **It has no scripts and no phrase list of its own.** It provides recognition to whatever in the project asks for it:

- the **Speech Phrase Recognition** robot skill (Step 3),
- the listen option of an AI robot skill such as Autonomous AI,
- the `Audio.WaitForSpeech()` script command and the Wait For Speech popup,
- the wake word of another engine.

Add one from **Add a robot skill → Audio**. Choose by where ARCx runs and what you are willing to pay:

Engine	Runs on	Cost	Internet	Strengths

SherpaOnnx Speech To Text	Windows, Linux (including Raspberry Pi), macOS on Apple silicon	Free	Only to download the model the first time	Offline neural models in many languages. Good to leave listening all day.
Windows Speech Engine	Windows only	Free	No	Built into Windows. Uses the phrase list as a strict grammar, which is very reliable for commands. Also provides a text to speech voice.
Azure Speech To Text Engine	Any	Your Synthiam account, with a daily limit on requests	Yes	Accurate cloud recognition, with no Azure key to set up. Records **one phrase per request** and never listens on its own (see below).
Azure Speech To Text Engine (Custom)	Any	Billed to your own Azure key	Yes	Listens continuously, can detect which of several languages is spoken.

Each engine's own manual covers its settings in detail.
You may add more than one engine to a project. The usual reason is a wake word: a free engine listens all day for "hey robot", and only then does a cloud engine record what follows. Each engine can only be added once per project.

Engines that only listen when triggered

Azure Speech To Text Engine costs a request from a daily allowance every time it records. To avoid spending that allowance on an empty room, it **never listens just because something asked it to listen continuously**. It records only when:

- its **wake word** is heard (set in its own configuration),
- its **Listen** button or Custom UI microphone widget is pressed,
- a script asks for one phrase (`GetText` control command, `Audio.WaitForSpeech()`),
- Speech Phrase Recognition is sent its **Wake** control command.

If you pair it with Speech Phrase Recognition and set none of these, nothing is ever heard. The log says so.

Step 3: Add Speech Phrase Recognition

Speech Phrase Recognition is the robot skill that turns recognized speech into actions. It has a table of phrases, each with a script. It asks the engine to listen for those phrases, and runs a phrase's script when the phrase is heard.



1. **Add a robot skill → Audio → Speech Phrase Recognition.** Only one is allowed per project.
2. Open its configuration. **Speech To Text Engine** is already set to the first engine in the project. Change it if you have more than one, for example to choose the cloud engine instead of the wake word engine.
3. Edit the **phrase table** at the bottom. Each row is one phrase and the script that runs when it is heard. Add rows with **+** and remove them with the bin icon. The skill ships with movement commands as examples.
4. Save. Listening starts immediately.

A phrase row's script can do anything a script can do:

```
// Phrase: "robot move forward"
Audio.Say("Moving forward");
Movement.forward();
```

Writing good phrases

- **Short, distinct, multi-word phrases** work best. "robot stop" is better than "stop".
- **Avoid phrases that sound alike.** "robot move forward" and "robot move backward" differ by one word, and that is the usual cause of the wrong script running.
- **Start every command with the robot's name.** This makes ordinary conversation in the room much less likely to trigger a command.
- Matching ignores case and surrounding spaces, and compares the whole phrase.

Other settings worth knowing

Setting	Use it for
Min Confidence	Rejects phrases the engine is unsure of. Default 0.70. Lower it for a poor microphone, raise it if commands trigger by mistake.
Low Confidence Script	Runs when something was heard but rejected. A good place for `Audio.Say("Sorry, say that again?")`.
Any Detected Phrase Script	Runs before every matched phrase's own script, and is waited for. Use it for a common acknowledgement such as a beep.
Enable Phrase / **Disable Phrase**	Spoken phrases that switch acting on commands on and off. They work even while the skill is disabled, so you can switch it back on by voice.
Engine Woke Script / **Engine Slept Script**	Run when the engine wakes on its wake word and when it goes back to sleep. For example, light an LED while the robot is listening.

It also sets three global variables that any script can read:

Variable	Contents
`SpeechPhrase`	The text of the last phrase heard.
`SpeechConfidence`	How sure the engine was, 0 to 1.
`SpeechAwake`	`true` while the engine is listening, `false` while it waits for a wake word.

The Speech Phrase Recognition manual describes every setting.

When the robot listens

A robot that listens all the time is easy to set up, but it can be triggered by a television, and it can spend a cloud engine's allowance. ARCx gives you four ways to decide when the robot listens. All of them are set on the **engine**, or sent to Speech Phrase Recognition from a script. None needs a different microphone.

Always listening

With **SherpaOnnx Speech To Text**, **Windows Speech Engine** or **Azure Speech To Text Engine (Custom)**, Speech Phrase Recognition listens continuously from the moment the project opens. Nothing more to set up. Use the **Paused** switch in the engine's window to stop it for a while, or the **Enable Phrase** / **Disable Phrase** to stop acting on commands by voice.

After a wake word

The engine stays asleep until a phrase such as "hey robot" is heard, then listens for a while.

1. Open the configuration of the engine that Speech Phrase Recognition uses, and find **Wake Word**.
2. Turn on **Only listen after a wake word**.
3. Choose the **Wake Word Engine**: the engine that listens all day for the phrase. Choose a free, local one, such as SherpaOnnx or Windows Speech Engine. It may be the same engine.
4. Enter the **Wake Phrases**, separated by commas.
5. Set **Stay Awake (ms)**: how long the robot keeps listening after the wake phrase. Every phrase heard restarts this timer. 0 means one phrase per wake.

The wake word is a setting of the **engine**, not of Speech Phrase Recognition. ARCx does the waking, so any robot skill that uses the engine gets the same behaviour. Speech Phrase Recognition runs its **Engine Woke Script** and **Engine Slept Script** when this happens.

When anybody talks

Add the **Voice Activity Detection** robot skill and choose it as the **Wake Word Engine** in step 3 above. It detects that someone is talking without recognizing words, so there is no phrase to enter. The engine wakes as soon as anybody speaks. This is a good fit for a cloud engine, as long as the room is not noisy. Set its sensitivity first. Its manual explains how.

When a button is pressed (push to talk)

Have a script tell the engine to listen. This works from anything that can run a script: a button on a Custom UI screen, a joystick or gamepad button, or another robot skill.

```
// Listen for one command
ControlCommand("Speech Phrase Recognition", "Wake");
```

Wake starts the engine listening as if its wake word had been heard. It only works for an engine that sleeps: one with a wake word configured, or one that only listens when triggered, such as **Azure Speech To Text Engine**. An engine that is always listening is never asleep, so **Wake** does nothing for it. **Sleep** puts the engine back to sleep early, and **IsAwake** returns whether it is listening.

Azure Speech To Text Engine, with no wake word set, is the most natural push to talk engine. Each **Wake** records exactly one phrase and costs exactly one request. Its own **Listen** button, and the **Listen** microphone widget it offers for Custom UI screens, also record one phrase. That phrase only goes to the log, though, which makes them a way to test the engine rather than to run commands.

How it works

From someone speaking to a script running:

1. **The microphone.** ARCx records the selected microphone (or the browser that claimed it) as mono sound at 16,000 samples per second, 64 milliseconds at a time, and hands it to every robot skill that is listening. The microphone only records while something is listening. When nothing is, it is switched off.
2. **The request.** When the project opens, Speech Phrase Recognition asks ARCx's speech recognition system to listen continuously on its chosen engine, and gives it the phrase list.
3. **The engine.** The engine converts the sound into text. How the phrase list is used depends on the engine:

Windows Speech Engine uses the list as a grammar. It only tries to hear those phrases, which is why it is so reliable for commands.
SherpaOnnx and **Azure (Custom)** recognize everything, then compare it with the list. Small mistakes are forgiven, and how close the match was is reported as the confidence.
Azure Speech To Text Engine recognizes free text. Speech Phrase Recognition does the matching itself.

4. **The speech recognition system** sits between engines and the skills that use them. It does three things so that neither side has to:

Restarting. An engine that records one phrase at a time, such as Azure Speech To Text Engine, is started again after each phrase. Speech Phrase Recognition sees a continuous stream of phrases either way.
Wake words. It keeps a sleeping engine quiet while the wake word engine listens, wakes it when the phrase is heard, and puts it back to sleep when the awake time runs out.
Sharing. There is one microphone. When a script calls `Audio.WaitForSpeech()` on the engine that Speech Phrase Recognition uses, the phrase skill is paused, the script gets its answer, and the phrase skill resumes.

5. **The robot's own voice.** Engines stop listening while the robot is talking (and for a short time after), so the robot does not hear its own text to speech and react to it. This is the **Ignore the robot's own voice** setting in each engine.
6. **The result.** The engine reports the text and a confidence. Speech Phrase Recognition then, in this order:

sets `SpeechPhrase` and `SpeechConfidence`,
runs the **Low Confidence Script** and stops, if the confidence is below **Min Confidence**,
handles the **Enable Phrase** and **Disable Phrase**,
stops, if it is disabled,
if the text matches a row, runs the **Any Detected Phrase Script** and waits for it to finish, then starts that row's script in the background.

Only one robot skill can listen continuously through an engine at a time. If an AI skill and Speech Phrase Recognition both want the same engine continuously, one of them reports "Could not listen through ...". Give each its own engine.

Listening from a script

A script can also ask for one phrase and wait for the answer. It does not need Speech Phrase Recognition for this:

```
Audio.Say("Should I move forward?");
var answer = Audio.WaitForSpeech(10, "yes", "no");   // wait up to 10 seconds for yes or no
```

```
if (answer == "yes")
  Movement.forward();
```

The engine used for this is set in **Project → Properties → Wait For Speech To Text Engine**. Leave it empty to use any engine in the project. Engines that only listen when triggered, such as Azure Speech To Text Engine, record straight away for this, because the request is itself the trigger.

Example setups

Offline voice commands on a Raspberry Pi. System default microphone (a USB microphone array) → SherpaOnnx Speech To Text with the default streaming model → Speech Phrase Recognition. Free, no internet after the first download, always listening.
A Windows robot with a strict command list. System default microphone → Windows Speech Engine → Speech Phrase Recognition. Install the speech language pack for your language in Windows Settings first.
Cloud accuracy without spending requests on an empty room. SherpaOnnx Speech To Text (free, always listening) as the **Wake Word Engine** in the configuration of Azure Speech To Text Engine, with the wake phrase `hey robot` → Speech Phrase Recognition set to use Azure Speech To Text Engine. A request is only made after the robot hears its name.
Talk to the robot from your phone. ARCx started on `https://` with a certificate → Microphone set to **A web browser** → press the microphone button in the phone's browser → any engine → Speech Phrase Recognition.
Push to talk. Azure Speech To Text Engine with no wake word → Speech Phrase Recognition set to use it → a Custom UI button whose script is `ControlCommand("Speech Phrase Recognition", "Wake");`. One press, one phrase, one request.

Troubleshooting

The level bar in the engine's window never moves. No sound is reaching ARCx. Check **Options → Preferences → Audio → Microphone**. If it is set to **A web browser**, check that a browser has claimed it (its top bar button is green). If no browser has claimed it, check the computer's own microphone. Tools → Diagnostics reports the microphone's state.
The browser asks for no permission, or Preferences warns that this browser cannot use its microphone. The page was opened over plain `http://` from another computer. Start ARCx on an `https://` address with a certificate.
"There is no default speech to text engine selected." There is no speech to text engine in the project. Add one (Step 2).
The level bar moves, but Speech Phrase Recognition logs nothing. The engine is asleep waiting for its wake word, or it is an engine that only listens when triggered and nothing has triggered it. The log of Speech Phrase Recognition says which. Say the wake word, set one, or send **Wake**.
Phrases are logged as low confidence. Lower **Min Confidence**, move the microphone closer, or speak more clearly.
The wrong script runs. Two phrases sound too alike. Make them more different, or start each with the robot's name.
A phrase is heard but no script runs. Speech Phrase Recognition is disabled (its **Enabled** switch is off, or the disable phrase was heard). Phrases are still logged while it is disabled.
The robot reacts to its own voice. Turn on **Ignore the robot's own voice** in the engine's configuration, and raise **Keep ignoring for (ms)** until the end of each sentence is no longer heard. A robot that speaks through an EZB needs most of a second.
"Could not listen through ..." in the log. Another robot skill, such as an AI skill, is already listening continuously through that engine. Give one of them a different engine.
Commands trigger from the television or conversation. Start every phrase with the

robot's name, raise **Min Confidence**, or use a wake word so the robot only listens when called.
Each engine's own manual has a troubleshooting section for problems specific to it: models that will not download, language packs, Azure accounts and keys.