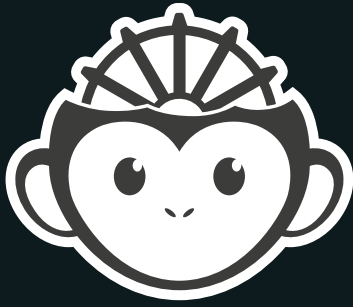




Documentation

Stackmill 1.2.1

Issued 2026-10-04

Contents

Stackmill	5
Your screen explained	6
Starting the cell	8
Choosing a recipe and a target	10
Pallet stations	11
When it pauses	13
When it faults	14
Emergency and safety stops	16
Moving the robot by hand	17
Calibration	18
Speed and payload	20
Operator card	21
What you need	23
Install on a desktop	24
Installing a Stackmill server	25
Adding Stackmill to a server you already run	28
Setting up the router 1.2.1	31
Activate a licence	33
First launch	34
Updates	35
Units and frames	37
Robot	38
Tool	39
Position	40

Reference	41
Volume	42
Work zone	43
Container	44
Imported object	45
Programs	46
Recipes	48
Variables	49
Cells, devices and flows	50
Licences	51
The 3D view	52
Objects panel	53
Programs panel	54
Variables panel	55
Code view	56
Menus	57
Settings	58
Live mode	59
Building an operator HMI	60
A full palletisation	61
Depalletising	63
Bundles and interlayers	64
Two stations	65
A calibration program	66
A standard program	67
FANUC controller setup	68

Exporting to the robot	70
Backup and restore	71
FAQ	72
Glossary	74
Troubleshooting	76
Safety	78
Remote support	80
What's new	81
Roadmap	82
Release notes	84

Stackmill

Build, simulate and run a palletising cell — without writing robot code.



I run a cell

Starting it, choosing a recipe, and what to do when it stops.

[Operating a cell](#)



I build programs

Objects, programs and recipes, then a full palletisation start to finish.

[Concepts](#) · [Guides](#)



I install Stackmill

On a desktop machine, on a server, and how licences work.

[Get started](#) · [Licences](#)

Getting the software

Download the demo from apemill.com to try it: it edits, saves and simulates, but cannot export a program. A university, a teacher or a student is usually eligible for a free education licence.

What the software is, what it needs and what it costs lives on the product site: [what Stackmill does](#) · [pricing](#) · [frequently asked questions](#).

Talking to us

Questions, training, or a feature you need — write to contact@apemill.com.

Your screen explained

The operator screen never changes shape. The same four areas are always in the same place, whatever the cell is doing.

The status bar

Across the top, one coloured band gives the state of the cell in a single glance.

Colour	State	Meaning
Grey	Offline	No link to the robot. Nothing will move.
Blue, pulsing	Initialising	Starting up. Wait for it to finish.
Green	Ready	Armed, waiting for you to press Start.
Teal	Running	Producing.
Amber	Paused	Stopped on purpose. It can carry on.
Red	Fault	Stopped by a problem.
Orange	Aborted	The cycle was abandoned. It must be rearmed.

Beside the state you get the running recipe, the current phase — picking, moving, dropping, going home — and the speed the cell is running at.

The pallet view

The middle of the screen is the cell seen from above: each pallet station, what is on it, and how far through its recipe it is. A station shows its own state and its count, so you can see at a glance which one will finish first. Details are in [Pallet stations](#).

The controls

The buttons live on the right. **Only the buttons that can be used right now are enabled** — if Start is greyed out, the cell is telling you it is not ready, and the message area says why. That is by design: an operator should never have to remember which button is legal in which state.

The messages

Under the controls, one line says what the cell is waiting for or what went wrong, and what to do about it. It is the most important text on the screen. Every message an operator can see is listed in [When it faults](#) and [Pallet stations](#), with the action it asks for.

Starting the cell

Before anything moves

Check that nobody is inside the fence and that every guard is closed. The software cannot see people. Starting a cell is a decision you make about a machine that is about to move on its own.

Before you start

- The fence is closed and nobody is inside.
- A pallet is present on at least one station, and it is the right one.
- The robot controller is in **AUTO** with remote mode enabled. If it is not, the screen says Robot not in remote mode and Start stays disabled.

Initialising

A cell that has just been powered on, or that was aborted, must be initialised before it can run. The screen walks you through it step by step; you cannot skip a step and you cannot get them out of order.

Confirming the payload

Initialisation asks one question you must answer honestly: **is the tool empty, or is it holding a part?**

The robot has no way to know. Your answer decides how it will move — a robot that thinks it is empty while holding a box will move as if it were light, and a robot that thinks it is loaded while empty will move slowly for nothing. If you are not sure, look at the tool.

Going to HOME

Once the payload is confirmed, the robot returns to its HOME position. It moves slowly and on a checked route. If it is already at HOME, this step passes immediately.

If the robot cannot reach HOME — it is against something, or in a position the trajectory does not cover — the screen offers the manual options described in [Moving the robot by hand](#).

Starting production

With the cell **Ready**, choose the recipe (see [Choosing a recipe and a target](#)) and press **Start**. The cell picks up where the recipe says, not necessarily at layer one: if a pallet was half full when the cell stopped, it carries on filling it.

Choosing a recipe and a target

A **recipe** is one product on one kind of pallet: the box size, how many per layer, how the layers stack. Changing product means changing recipe.

Choosing the recipe

Pick it from the recipe list before starting. The cell will not start without one, and the message area says the recipe was never started if you try.

You cannot change recipe while the cell is running. Stop it first, and let it finish the box it is holding.

Palletising or depalletising

The same recipe runs in both directions. **Palletise** builds the stack; **depalletise** takes it apart, in reverse order. Choose the direction next to the recipe.

Setting a target

By default the cell fills the pallet and stops. You can also tell it to stop earlier:

- **By layer** — stop when a given number of layers is complete. Use this to finish a part pallet at the end of a shift.
- **By part** — stop after a given number of boxes.
- **The whole pallet** — the default.

The target counts what is placed from now on, and the screen shows both the target and how far along it is.

Changing it while running

The target can be changed while the cell runs; the recipe cannot. A new target takes effect at the end of the box in progress — nothing is dropped or abandoned to obey it.

Pallet stations

A station is a place where a pallet sits. A cell has one or two.

Reading a station

The station says	It means	What to do
No pallet	Nothing is on it	Put an empty pallet on
To validate	A pallet is there but nobody has confirmed it	Check it, then validate it
Complete	The recipe finished this pallet	Take it away, put an empty one on
Removed mid-cycle	A pallet was taken while the cell was using it	Put one back, then press Reset

Validating a new pallet

Validating is you telling the cell this pallet is empty, square, and the right kind. The cell will not place a box on a station that has not been validated: the message area says Pallet ... waiting for validation and waits.

Look before you validate. A pallet that is broken, badly placed or already carrying something will produce a stack that leans — and the cell has no way to see any of that.

Removing a full pallet

When a pallet is **Complete**, take it away and put an empty one on. The station goes back to To validate, and the cell carries on as soon as you validate it.

Working with two stations

With two stations the cell fills one while you empty the other, and production never stops for a pallet change. The pallet view shows which one is filling and which is waiting. If both are unavailable, the cell waits and says Waiting for a pallet — it does not fault.

Part tare 1.2.1

When the robot can weigh what it carries, the gear menu of a station has one more entry: **Tare on next part** weighs the next part again. A recipe change does it by itself; use the entry when the product changes within the same recipe.

When it pauses

A pause is not a fault. The cell stopped cleanly, on purpose, and everything it knows is still true.

Why it paused

Either somebody pressed **Pause**, or the cell reached a point where it needs something — a pallet, a part, a validation. The message area says which.

Resuming

Press **Resume**. The robot carries on from where it stopped, in the same cycle, with the same box. Nothing is repeated and nothing is skipped.

Aborting

Abort abandons the cycle instead of continuing it. Use it when resuming is not what you want — the wrong product is loaded, or something in the cell needs to be cleared.

After an abort the cell is in **Aborted** and must be rearmed before it can run again, which means going through [initialisation](#) once more. **A part held by the tool stays held** — rearming will ask you about it.

What is kept

A pause changes nothing on the pallet. The count, the layer in progress and the target all survive it, and they survive an abort too. Resuming or restarting continues the same pallet rather than starting a new one.

When it faults

A fault means the cell stopped because something was wrong. It stays stopped until you deal with it — it will never restart on its own.

Reading the message

The screen gives the fault in two parts: **what happened**, and **what to do about it**. Read the second part first; it is written for the situation you are actually in.

Faults you can resume

Some faults stop the cycle without losing it. The message says cycle can resume, and once the cause is gone you can **Acquit & Resume** and carry on from the same box.

Message	What to do
Robot fault — cycle can resume	Clear the cause if there is one, then Acquit & Resume.
Robot not in remote mode	Put the controller back in AUTO with remote enabled, then Reset.
Waiting for a pallet, a part or a validation	Nothing is broken. Give it what it asks for.

Faults you cannot resume

Others end the cycle. Clear the cause, press **Reset**, and the cell goes back to a state you can rearm from.

Message	What to do
Robot fault	Read the detail on screen, clear the cause, then Reset.
Program stopped: ...	The reason is named in the message. Correct it, then Reset. The code stays visible until the cell is rearmed.
Contact stop — the robot touched something	Free the contact, then Reset. See Emergency and safety stops .
Tool payload differs from the declared one	Look at what the tool is holding, then correct the payload during initialisation.

The recovery steps

When a fault leaves the robot somewhere it should not be, the screen walks you out of it rather than leaving you to guess: it will tell you whether the tool is holding something, offer to free it, and bring the robot back to HOME on a checked route. Follow the steps in order.

Calling for help

Stop and call for help when the message is not one of the above, when the same fault comes back after a reset, or when the cell asks you to move the robot by hand and you have not been trained to. **Do not open the fence to look.**

Emergency and safety stops

 The red button is not on this screen

The emergency stop is the physical button on the cell. Use it whenever anything is wrong. Nothing in the software replaces it, and nothing in the software can override it.

Emergency stop

While the emergency stop is engaged, the robot has no power to move and the screen says Emergency stop active. Release the button, then **rearm** the cell — releasing it alone does not put the cell back in service, and that is deliberate.

Collaborative stop

A collaborative robot stops by itself when it touches something. The screen says Contact stop — the robot touched something.

Free the contact first: clear whatever the robot is against, and make sure it is not resting on a person or a part. Then press **Reset**. If the robot is wedged and cannot be freed by hand, use the escape option offered on screen rather than forcing it.

Soft stop

The screen also has a **Soft stop**, which stops motion through the software. It is convenient and it is immediate, but it is **not** a safety device: it depends on the software and the network being alive. Use the red button for anything that matters.

Rearming

After an emergency stop, a collaborative stop or an abort, the cell must be rearmed. Rearming runs [initialisation](#) again, which is where the cell asks whether the tool is holding a part — answer that from what you can see, not from what you remember.

Moving the robot by hand

 **Trained staff only**

Everything on this page moves an industrial robot under your control, with the guards in a state that lets it move. If you have not been trained for it, stop here and call someone who has.

Who may do this

Manual movement is for setting up, for clearing a jam and for recovering after a fault. It is not part of running production, and the buttons are hidden from a normal operator session.

Jogging

Jogging drives one axis, or the tool, a little at a time while you hold the control. Two modes:

- **Joint** — one axis at a time. Predictable, and the safest way out of an awkward position.
- **Linear** — the tool moves in a straight line. Easier to aim, but several axes move at once.

Jog speed is limited, and the movement stops the moment you let go.

Free motion

Free motion releases the brakes so the arm can be moved by hand. The screen asks you to confirm before it does — **the arm can drop under its own weight when the brakes release** if it is extended or loaded. Support it before confirming, and take the load out of the tool first if you can.

Escaping a collision

If the robot is against something and normal motion is refused, the screen offers a dedicated escape: a short, slow move away from the contact. Use it rather than jogging blindly, and rather than forcing the arm.

Returning to HOME

When the robot is clear, send it back to HOME. The cell checks the route before it moves. Once it is home, the cell can be rearmed and production restarted from [Starting the cell](#).

Calibration

Calibration is how the cell learns exactly where a pallet station is. A cell whose stations have shifted will place boxes slightly off, and the error grows with every layer.

When to recalibrate

- After anything hits or moves a station.
- After the robot or its base has been moved.
- When the stack is visibly leaning, or boxes land a little further out on each layer.

If in doubt, calibrate. It costs a few minutes and it is not destructive.

Semi-automatic

You bring the robot to each measurement point yourself and confirm it. Slower, but it works when the station is far from where the cell expects it — which is exactly the case after something was knocked.

Automatic

The robot finds the points itself by touching the station. Faster, and the normal choice for a routine check. It needs the station to be roughly where it was, or the probe finds nothing.

Running a calibration program 1.2.1

Calibration programs are prepared by the integrator and exported to the robot with the project. On the live screen, an administrator finds them under **Calibration programs** and opens one.

The dialog shows the cell in 3D with the path the robot will follow drawn over it: each move, each probe with the surface it will touch, and the frame being measured. The list on the right names every step. Choose **Semi-auto** or **Full-auto**, then **Start**: the robot leaves HOME and the steps tick as it reports them.

In semi-auto, the robot stops before each probe and a translucent robot appears where the probe expects it to stand, with the arrow of the touch it is about to make. Use the jog to bring the real robot onto the ghost — the distance is shown — then press **Probe now**. In full-auto the robot chains every probe on its own.

Abort stops the robot at any time; the frame is never changed by an abort. Closing the dialog does not stop the run: it opens again on every screen while the run goes on.

Checking the result

The cell shows the correction it measured before applying it. A few millimetres is normal. A correction much larger than that means something really has moved — go and look before accepting it, because accepting a wrong measurement teaches the cell a wrong position.

If the probe finds nothing over its whole travel, or the correction is outside the accepted band, the cell refuses the result and says so. That is the system protecting you: fix the station, then calibrate again.

Speed and payload

Changing the speed

The speed control scales how fast the cell runs, from a crawl to full rate. It changes speed only: **the robot follows exactly the same path**, so slowing down is always safe and is the right thing to do while watching a new recipe for the first time.

Speed is not remembered between runs on purpose. A cell that was slowed for a good reason yesterday should not quietly still be slow today.

What payload means

The payload is what the tool is carrying. The robot uses it to decide how hard to accelerate and how to hold a position — a heavy box needs to be moved differently from an empty gripper.

The cell asks you to confirm the payload during [initialisation](#), because it has no sensor that can tell it.

Payload mismatch

Tool payload differs from the declared one means what the cell believes it is carrying does not match what it measures.

Stop and look at the tool. Either it is holding something it should not be — a box that was never released — or it is empty when the cell thinks it is loaded. Correct the payload at the next initialisation rather than continuing: a robot moving with the wrong payload is a robot moving with the wrong forces.

Operator card

Status bar

	State	What it means
	Offline	No link to the robot. Nothing will move.
	Initialising	Starting up. Wait.
	Ready	Armed and waiting for Start.
	Running	Producing.
	Paused	Stopped on purpose, resumable.
	Fault	Stopped by a problem. See below.
	Aborted	Cycle abandoned. Rearm to restart.

If it stops

Message on screen	What to do
Robot disconnected	Check the robot power and network; reconnection is automatic.
Emergency stop active	Release the E-stop, then rearm the cell.
Contact stop — the robot touched something	Free the contact (clear the robot), then press Reset .
Robot fault	Read the detail on screen, clear the cause, then press Reset .
Robot fault — cycle can resume	Clear the cause if needed, then Acquit & Resume to continue where it stopped.
Program stopped: ...	Clear the cause, then Reset . The code stays until the cell is rearmed.
Robot not in remote mode	Set the controller to AUTO with remote enabled, or press Reset .
Tool payload differs from the declared one	Check what the tool is holding, then correct the payload during init.

If it waits

Message on screen	What to do
Pallet ... waiting for validation	Check the pallet, then press its validation button.
Waiting for a pallet	Place an empty pallet on a free station.
Waiting for a part on the conveyor	Nothing. The cycle resumes when a part reaches the pick point.
Waiting for the next operation...	Nothing. Normal.
Stop requested — finishing the current step	Nothing. It stops at the end of the step.

Buttons

Button	Does
Start	Begins production with the selected recipe.
Pause / Resume	Stops at the end of the move / carries on.
Stop	Finishes the current step, then stops.
Abort	Abandons the cycle. The part in the tool stays in the tool.
Soft stop	Software emergency stop. Not a substitute for the red button.
Reset	Acknowledges a fault once its cause is gone.
Rearm	Puts an aborted or e-stopped cell back into service.
Home	Sends the robot to its HOME position.
Speed	Changes the running speed. Does not change the trajectory.

Pallet stations

No pallet — put one on · **To validate** — check it, then press validate · **Complete** — take it away · **Removed mid-cycle** — a pallet left during the cycle; press Reset after putting one back.

Do not open the fence. If the message is not in this list, or the same fault comes back: stop the cell and call

Name Phone

What you need

The computer

Stackmill runs on Windows and on Linux, on ordinary hardware. It draws the cell in 3D and plans trajectories, so it wants a real graphics card rather than an integrated one, and it wants memory more than it wants cores.

A server installation runs in Docker and needs nothing on the host beyond Docker itself — see [Add to an existing server](#).

The security module

Whether the machine has a **TPM 2.0** decides how its licence behaves, and it is worth establishing before you install rather than after:

- **With a TPM** — the licence binds to the machine and then works with no internet at all. This is the only configuration in which the software will drive a real robot.
- **Without one** — the licence stays online and the software checks in regularly. Such a machine runs simulations only.

Most industrial PCs and most business laptops have a TPM. [Licences](#) explains what follows from it.

The robot

Stackmill talks to **FANUC** controllers. The controller needs to be reachable over the network and set to accept remote commands; the options and the setup are in [FANUC controller setup](#).

Other brands are on the roadmap — the driver, the solver and the exporter are already written to be brand-agnostic inside.

The network

- Between Stackmill and the robot: a normal Ethernet link. The two must be able to reach each other; nothing is routed through us.
- Between Stackmill and the internet: needed **once**, to activate the licence. After that a machine with a TPM never needs it again, though without it the software cannot tell you a new version exists.

Install on a desktop

Download the installer from apemill.com. The same download is the demo: without a licence it edits, saves and simulates, and cannot export a program.

Windows **Linux**

Run the installer and follow it. It puts Stackmill in the usual place, adds a shortcut, and needs no further configuration.

Download the package for your distribution and install it the normal way. Stackmill needs a desktop environment — it draws a 3D view.

What gets installed

One application, and a folder for your work. **Projects, settings and the licence live outside the application**, which is why an [update](#) never touches them and why uninstalling does not take your programs with it.

Next

[Activate a licence](#), then [first launch](#).

Installing a Stackmill server

This turns a bare mini PC into a Stackmill server. It takes about twenty minutes, most of it waiting, and needs no Linux knowledge: the installation asks no questions.

The whole disk is erased. Use a machine with nothing on it.

What you need

- An x86-64 mini PC with at least 8 GB of memory and a 64 GB disk, with **TPM 2.0** and **Secure Boot** enabled in the BIOS. TPM 2.0 is often called Intel PTT or AMD fTPM.
- A USB stick of 8 GB or more.
- An Ethernet cable to a network that hands out addresses automatically, with internet access for the activation step. After that the machine can stay offline forever.
- A laptop on the same network.
- The server licence key for the customer, and access to the Apemill portal.

1. Download the image

In the portal, open **Server** and download the ISO. Next to it is a long checksum; keep the page open, you can compare it after writing the stick.

2. Write the USB stick

Use [balenaEtcher](#) (Windows, macOS, Linux) or Rufus on Windows. Choose the ISO, choose the stick, write. Nothing else to configure.

3. Install

Plug the stick and the Ethernet cable into the mini PC and switch it on. Open the boot menu — usually F12, F11, F8 or Esc depending on the make — and choose the USB stick.

One entry appears: **Install Stackmill server — ERASES THE WHOLE DISK**. It starts on its own after ten seconds.

The installation runs unattended for about ten minutes, then the machine reboots. Remove the stick when it does.

4. First start

The machine sets itself up: it encrypts the disk, ties the key to its own TPM chip, and starts Stackmill. Give it two or three minutes.

The screen then shows the version, the address to open, and not activated.

5. Activate

From the laptop, open **http://stackmill.local**. If that name does not resolve, use the address shown on the mini PC's screen.

Stackmill opens with a licence panel. Paste the server licence key and confirm. The machine binds itself to its TPM chip, and the screen on the mini PC now reads activated.

That is the whole installation. Internet is no longer needed: the licence works offline permanently. Only updates use the network again.

If something goes wrong

What you see	What it means
The licence is refused for a missing TPM	TPM 2.0 is off in the BIOS. Turn it on, restart, activate again. Nothing is lost.
No address on the screen	The network handed out none. Check the cable, then the network's address server.
<code>stackmill.local</code> does not open	Some Windows machines cannot resolve it. Use the address printed on the screen.
Two Stackmill servers on one network	The second answers to <code>stackmill-2.local</code> . Use the addresses instead.
The licence is refused for a virtual machine	Stackmill only runs on real hardware.

Notes for the site's IT

- The server uses port **80** for the interface and ports **22222**, **29750**, **33500**, **38890**, **41800**, **42400** for the connection between the interface and the robot cell. All of them on the local network only.

- Give the machine a fixed address by reserving one for it in the network's address server. The server does not need to be reconfigured.
- Nothing accepts a connection from outside the local network. Remote support opens on the customer's consent, from the machine outward, and the customer ends it whenever they want.
- Updates are never automatic. Someone decides, from the interface or by a command.

Adding Stackmill to a server you already run

For a Linux server the customer maintains themselves. If the machine is dedicated to Stackmill, the [server installation](#) is simpler and takes care of the operating system too.

What is needed

- Ubuntu 22.04, Ubuntu 24.04 or Debian 12 on x86-64, with root access.
- **TPM 2.0** (`/dev/tpmrm0`). Without it the licence needs permanent internet and the machine can only simulate, never drive a real robot.
- A physical machine. A virtual machine is refused: its security chip can be copied with it.
- Internet during the installation.
- The organisation id and registry token, on the portal's **Server** page.

Install

The portal shows the command, already filled in for the organisation:

```
curl -fsSL https://dist.apemill.com/install.sh | sudo bash -s -- --org <id> --token <token>
```

To read it first, download `install.sh` and its checksum from the same page, check it, then run it with the same options.

It installs Docker if the machine has none, downloads the Stackmill image, checks Apemill's signature on it, and starts the service. Running it twice changes nothing and never touches the data.

Useful options:

Option	For
<code>--http-port 8080</code>	when something already answers on port 80
<code>--data-dir /srv/stackmill</code>	to put the data on another disk
<code>--bundle stackmill-server-<version>.tar</code>	a machine with no internet: download the bundle from the portal and copy it over
<code>--version 1.4.0</code>	a specific version instead of the newest
<code>--uninstall</code>	remove Stackmill and keep the data

Activate

Open `http://<the server's address>`, paste the server licence key in the licence panel and confirm. Activation also stores the credential the machine needs for its updates, so nothing has to be typed again.

Operate

```
stackmillctl status      # version, licence, whether a cell is running
stackmillctl check      # is there a newer version
sudo stackmillctl update # install it: refuses while a cell runs, rolls back if it fails
sudo stackmillctl rollback # go back to the previous version
stackmillctl logs
sudo stackmillctl backup  # archive the data directory
```

The interface also has an **Update now** button, which runs exactly the same thing.

Without the installer

Stackmill runs from a plain `docker compose` on any Docker host: sign in to `docker.apemill.com` with the organisation id and token, pin a version in the compose file, and start it. There is then no `stackmillctl`, no **Update now** button and no remote support — the update dialog prints the two commands to run instead. Ask Apemill for the compose file.

What to back up

Only the data directory, `/opt/stackmill/data` by default. It holds the projects, the licence and the settings; everything else is reinstalled from the portal in minutes.

Restoring it onto a machine **without** a TPM needs the licence to be activated again: the licence server cannot tell a restored copy from a duplicated machine. With a TPM, restoring works as is.

Setting up the router 1.2.1

Every Stackmill machine has its own router. It links the robot, the server and the tablet, and gives the tablet its WiFi. Stackmill sets it up and changes its WiFi itself: nothing else is needed.

Supported routers

Stackmill sets up the **MikroTik hAP ax S**, the router delivered with the machine. It is the only model tested, and Stackmill refuses any other: a router that is not a hAP ax S is never changed.

For a machine on a network you manage yourself, or a virtual machine, there is nothing to set up. To use another router, contact Apemill: we add new equipment on request.

What is needed

- The router, new or already set up, plugged into the Stackmill server with a network cable, on one of its ports **2 to 5**.
- The password printed on the router's label.
- An administrator session in Stackmill.

A computer running Stackmill can set the router up too, plugged into the same ports. The dialog then shows the router's address: `192.168.88.1` for a new router.

Set it up

1. Open **File ► Router...**. Stackmill finds the router.
2. Enter the **admin password** printed on the router's label.
3. Check the **machine WiFi**: its name and password.
4. Check the **country**, then **Set up**.

The machine WiFi appears about a minute later. A router another machine of your organisation already set up needs nothing: Stackmill logs in on its own.

 **ly the machine's own router**

Set up only the router delivered with the machine: Stackmill rewrites its WiFi and its logins.

Several machines in one room

Each machine has its own router and its own WiFi, so each tablet joins its own machine. Stackmill proposes a name no other machine of your organisation uses, such as `Stackmill-2`. Choose one that says which machine it is, for example `Stackmill-Line2`.

Change the machine WiFi

In **File ► Router...**, next to the machine WiFi, click **Change**, enter the new name and password, then **Change the WiFi**. No label password is needed.

Every tablet on that WiFi disconnects at once. Join the new WiFi from each tablet, then open Stackmill again.

The licence portal

Stackmill saves each router to the licence portal, on the **Server** page, **Router** card. Everyone in your organisation reads each router's machine WiFi there; your organisation's administrators also find its admin password. A change made from Stackmill appears there too. Without internet, the dialog shows **Waiting for the internet** and saves the router once the machine is online.

Connect the tablet

On the tablet, join the machine WiFi with its password, then open Stackmill. The machine WiFi is on 5 GHz: the tablet must support it.

Give the machine internet

Either way works:

- **A cable:** plug the site's network into the router's port **1**. Nothing else to do.
- **The site's WiFi:** in **File ► Router...**, **Scan**, pick the network or type its name, type its password and **Connect**.

The cable wins when both are there. A network that does not connect within 45 seconds is abandoned, and the machine goes back to the one it was on.

Activate a licence

A licence key looks like `APE-XXXX-XXXX-XXXX`. It is entered once, and then belongs to that machine.

Entering the key

Open the licence panel, type the key, confirm. **The machine needs internet at this moment** — this is the one point where it has to reach us.

What happens next depends on the machine, and [Licences](#) explains why: with a security module the licence is bound to the machine and works offline afterwards; without one it stays online and the software checks in regularly.

A machine with no internet

A cell on a factory floor often has no route out. The licence panel then produces a **request file**. Carry it to any machine that does have internet, upload it to the licence portal, and bring back the answer. Nothing else about the machine changes.

Checking the licence

The licence badge sits next to the top of the window. It gives the state, and opening it gives the detail — what the licence allows, and, for an online licence, when it last checked in.

Moving to another machine

A licence follows the machine it was bound to, so a new board, a cleared TPM or a new computer needs a new binding. Ask from the licence panel: it produces a request that we approve online, or a file to carry if the machine is offline.

Do this **before** decommissioning the old machine where you can. It is quicker to release a licence that is still running than to prove that one is gone.

A floating licence moves without asking: **Release the seat** on the old machine, then activate the key on the new one.

First launch

The start screen

Stackmill opens on a start screen rather than an empty 3D view: recent projects, the demo project, and a way to create a new one. Nothing is loaded until you choose.

Opening the demo project

Start here, even if you have a cell to build. The demo project is a complete, working palletising cell — robot, tool, pallet, recipe, program. Every guide in this documentation is followed inside it, and every screenshot is taken in it, so what you see on your screen is what you see on the page.

It is also safe: it is a simulation, and nothing in it can move a real robot.

Connecting to a robot

A project drives a robot only once you tell it which one and where. That is a [cell](#), and it is set up per project. Until then, everything you do is simulated — which is the right way to build a program.

Where projects are stored

In your own documents, not inside the application. A project is a single file: copy it, mail it, put it under version control. On a server installation, projects live in the server's data folder instead, so that everyone connecting to it sees the same list.

Updates

Stackmill checks for a new version a minute after it starts, then once a day. Nothing is ever installed on its own: an update is a decision, and on a production cell it is a decision with a date.

The check never blocks anything. A machine with no internet keeps working exactly as before; it simply never announces a new version.

Where it appears

Next to the licence badge, and in `Help ▶ Stackmill ©`. Both are visible in admin or developer sessions only — an operator running a cell is not shown an update. Opening the badge gives the version, its date and its release notes. `Help ▶ Update...` opens the same dialog at any time.

Installing on the desktop application

The dialog installs it: the download is verified against Apemill's signature, the application restarts on the new version. The button is disabled while a cell is running — stop the cell first.

Installing on a server

- **A server with the host service** (installed from the ISO or with `install.sh`) — click **Update now**. The server restarts and everyone connected is cut off for about a minute; the dialog shows the progress, the page reconnects, and the result appears as a notification. It refuses while a cell is running, verifies the image signature, and returns to the previous version by itself if the new one does not start.
- **A server without it** — the dialog shows the command to run on the machine, with the same guards:

```
sudo stackmillctl update
```

Without `stackmillctl`, set the new tag in `docker-compose.yml` and run `docker compose pull && docker compose up -d`.

- **In a browser, against a desktop machine** — the dialog gives the download link; install it on that machine.

Installing a given version

The dialog takes a version number, betas included. On a server, **Update to** installs exactly that one, with the same guards; without the host service: `sudo stackmillctl update --to <version>`.

On a desktop, it downloads that version's installer. An older version is flagged: a file saved by a newer version is not guaranteed to work with it.

Skipping several versions

Some versions cannot be reached in one jump, because the project file format changed on the way. Stackmill then names the version to install first. Install that one, then the newest.

Projects, licence and settings are never touched by an update. On a server they live in `/data`, which the update does not replace.

Units and frames

Units

Millimetres and degrees, everywhere, with no exceptions and no unit selector. Masses are kilograms. A field never means anything else than what this paragraph says.

Coordinate frames

A position is meaningless on its own; it means something in a frame.

- **World** — the fixed origin of the scene. Everything can ultimately be expressed in it.
- **User frames** — origins you create, usually one per pallet station or per machine. A [reference](#) object is how you make one.
- **Tool** — carried by the robot, at the [TCP](#). Moving in the tool frame means moving relative to the gripper, whichever way it is pointing.

Working in a user frame is what makes a cell maintainable: when the pallet station is moved by 30 mm, you correct the frame and every position on it follows.

Positions and orientation

A position is a point **and** an orientation — where the tool is and which way it faces. Two positions with the same point and different orientations are different positions, and the robot reaches them differently.

Robot configuration

An industrial arm can often reach the same position in more than one posture: shoulder front or rear, elbow up or down, wrist flipped. These are its **configurations**.

They matter because the robot cannot swap between them while moving. A trajectory that would need to change configuration mid-path is not a trajectory, and the software will tell you so rather than trying.

Robot

The arm. A project usually has one.

Adding a robot

Choose the model from the list. The model carries its geometry, its axis limits and its reach, so choosing correctly matters more than anything you will set afterwards.

Place it where it really is. Everything else in the project is measured from it.

Reach and configuration

A position is reachable if the arm can hold it in a valid [configuration](#) without exceeding an axis limit. The software can preview this: select a position and it will show the robot standing in it, or tell you it cannot.

Use it while designing. A pallet corner that is reachable only in a different configuration from the rest is a cell that will fail on the last box of a layer, not on the first.

The HOME position

HOME is where the robot waits and where recovery brings it back. Choose one that is clear of everything, reachable from anywhere the robot works, and comfortable to look at — an operator will see it more than any other.

Axes and limits

Each axis has a range, and the software will not plan through it. Where a cell is tight, that limit is what a trajectory fails on: the position is reachable, and the route to it is not.

Tool

What the robot grips with, mounted on its flange.

Jaws and opening

Describe the tool as it is: its shape, and how it grips. The opening is what the robot commands to take and release a part, and the software uses it to know when a part is held.

Today the shipped tools are suction; mechanical jaws and cylinder grip are on the [roadmap](#).

The TCP

The **Tool Centre Point** is the exact point the robot positions — usually the middle of the gripping face. It is the single most important measurement in the project: every position is relative to it, so a TCP that is 5 mm out puts every box 5 mm out.

Measure it, do not estimate it. If the stack is consistently offset in one direction, suspect the TCP before anything else.

Payload

The mass the tool carries, and where that mass sits. The robot uses it to decide how hard it can accelerate. Too low and the robot moves aggressively with a heavy box; too high and it crawls for nothing.

The operator confirms the payload at [start-up](#), because no sensor can tell the software whether the tool is currently holding something.

Gripping several parts

A tool can be described as taking more than one part at a time. The recipe then places them as a group, which is what makes a fast cell fast — but every part in the group must be reachable in one motion, so it constrains the pattern.

Position

A place. Positions are the vocabulary a program is written in.

Waypoints

A plain position the robot passes through or stops at: an approach, a clearance point, a place to wait. Waypoints exist to shape a movement — around an obstacle, or into a rack from the right direction.

Object positions

Where a part is picked from. The recipe needs to know exactly where a box arrives, in what orientation, and how the tool should approach it.

Container positions

Where a [container](#) sits — the pallet station. The recipe computes every drop from it, so this is the position that [calibration](#) measures and corrects.

Editing a position

A position can be typed in, or taught by moving the robot there and capturing it. Typed is exact and repeatable; taught is honest about where things really are. Most cells use both: typed for what was designed, taught for what was built.

Positions belong to a [frame](#). Put them in the frame of the thing they describe, and a station that moves takes its positions with it.

Reference

An origin. A reference carries a position and an orientation, and other objects hang off it.

What it is for

To give a group of things one place to be moved from. A pallet station is the usual example: the station's reference holds the container, its approach positions and its drop positions, so correcting the reference corrects the lot.

A cell built on references survives the real world. A cell built with everything in world coordinates has to be re-taught every time something is bumped.

Creating one

Add it, place it, then drag onto it whatever belongs to it. Its helper shows the three axes, which is worth leaving on while you build: an origin rotated 90° from what you assumed is very hard to see and very easy to check.

Moving a whole branch

Move the reference and everything under it moves. That is the whole point, and it is also the warning: check what is under a reference before you move it.

Volume

A box. The simplest way to tell the scene that something is there.

Size and placement

Three dimensions and a position. Use volumes for the things that occupy space without needing to be modelled in detail: a fence, a conveyor, a control cabinet, a machine the robot must reach around.

Collisions

A volume is most useful with collision on: it becomes an obstacle the planner must route around. That is what turns "the robot might hit the cabinet" into "the software will not plan through the cabinet".

Using it as an obstacle

Model obstacles slightly generously. A volume 20 mm larger than the real cabinet costs nothing and buys margin; a volume 20 mm smaller is a collision the software promised would not happen.

For anything shaped where a box is too crude, [import the geometry](#) instead.

Work zone

A volume with a rule attached: the robot must stay inside it, or must stay out of it.

What it protects

Work zones express the limits of a cell that are not physical objects — the reach you are willing to give the robot, the corridor a person walks through, the space above a conveyor.

It is not a safety function

A work zone is checked by the software when it plans. It is not a safety device and it does not replace the robot controller's own zone monitoring. See [Safety](#).

Defining one

Give it a size and a position, and say which side of it the robot belongs on. Like every other object it can hang off a [reference](#), which is how a zone follows the station it belongs to.

What happens at the border

A trajectory that would breach the zone is not produced: the software reports it instead of planning around the rule. That is deliberate — a zone quietly ignored is worse than no zone.

Container

What parts are stacked on, or into.

Pallets

The usual container: a footprint, a height, and a position in the cell. The [recipe](#) computes every drop position from the container, so its dimensions have to be the real ones.

A cell normally has one or two container positions — the [stations](#) an operator loads and unloads.

Boxes

Palletising into a box rather than onto a pallet is on the [roadmap](#).

Interlayers

A sheet placed between layers. Where the recipe uses them, it needs to know their thickness: an interlayer that is not declared makes every layer above it land slightly low, and the error accumulates up the stack.

Imported object

Geometry from somewhere else, when a [volume](#) is too crude to be useful.

Supported formats

Mesh geometry — the shapes a 3D file describes. STEP and other CAD assembly formats are on the [roadmap](#); today, convert to a mesh first.

Scale and origin

Two things go wrong with imported geometry, and both are worth checking immediately:

- **Scale** — a file in metres imported as millimetres is a thousand times too big, and it is obvious. A file in inches is 25.4 times out, and it is not.
- **Origin** — CAD files are often exported around an origin that meant something to the designer and nothing to you. Place the object by its origin, then check it against something you know.

Collision meshes

Detailed geometry is expensive to check for collisions. Where an imported object is only ever an obstacle, a simpler shape around it is faster and just as safe — usually a [volume](#).

Programs

The four kinds

Kind	What it is
Palletising	Generated from a recipe . This is what most cells run.
Standard	Written line by line, for logic a recipe does not cover.
Calibration	Measures where something really is. See Calibration .
External	A program that lives on the robot, called from Stackmill.

Programs live in folders and can call one another.

Palletising programs

You do not write these. You describe the product, the pallet and the pattern in a [recipe](#), and the program is generated — every pick, every drop, every approach, in the right order, with the trajectories planned.

That is the point of Stackmill: the part of the job that is arithmetic and repetition is done for you, and the part that is judgement is left to you.

Standard programs

For everything a recipe does not describe: waiting on a signal, driving a conveyor, choosing between two behaviours. Written as a list of instructions.

The instruction set

Instruction	Does
MOVE	Go to a position
SET	Give a variable a value
WAIT	Wait for a condition or a delay
IF	Do something only when a condition holds
FOR / WHILE	Repeat
CALL	Run another program
LBL	A named point in the program
PRINT / COM	A message, and a comment

Conditions are built from [variables](#). There is no text syntax to learn: an instruction is chosen and filled in.

External programs

Where a robot already has a program you want to keep, Stackmill can call it rather than replace it. Useful when commissioning a cell alongside something that already works.

Recipes

A recipe is one product on one kind of pallet. Changing product means changing recipe, and it is the only thing an operator picks before starting.

What a recipe holds

- The part: its dimensions and its mass.
- The container: which [pallet](#) it stacks on.
- The pattern: how parts sit in a layer, and how layers stack.
- The trajectories: how the robot gets from the pick to each drop.

Dispositions

The **disposition** is the arrangement of parts in a layer. Stackmill can generate it: give it the part and the pallet and it produces the arrangements that fit, ordered by how well they use the surface.

Generated dispositions are a starting point, not a verdict. The software optimises area; it does not know that your boxes have a printed face that must point outwards, or that the film wrapper needs a margin.

Different shapes are handled differently — rectangular parts, cylinders, bottles and bundles each have their own way of tiling a layer.

Layers and patterns

A stack is layers, and consecutive layers usually differ: rotating alternate layers is what keeps a pallet stable. The recipe holds the whole sequence, so a pallet is not simply the same layer repeated.

Trajectories

For every drop the robot needs a route: an approach, the placement, a retreat. These are generated with the disposition and checked against the scene, so a pattern that cannot be built without hitting the stack is reported rather than exported.

Variables

Values a program reads and writes, and the way Stackmill and the robot say things to each other.

Types

Booleans, integers, floating-point numbers, text, positions and joint positions — each also available as an array or as a fixed set of choices.

A variable can be marked read-only where it reports something rather than controls it.

User and generated variables

- **User variables** are yours: a counter, a mode, a threshold.
- **Generated variables** are produced by Stackmill from the project — the list of tools, the list of references, the recipes, the programs. They exist so a program can refer to "the current recipe" without hard-coding a number that changes the next time you edit the project.

Prefer a generated variable wherever one exists. A program written against them survives edits that a program written against numbers does not.

FANUC registers

On the robot, values live in registers: numeric registers, position registers, digital inputs and outputs. Stackmill maps what it needs onto them and writes the map when you [export](#).

You rarely need to care which register is which — but when reading a program on the pendant, or when a controller reports an error against a register number, this is the map that explains it.

Cells, devices and flows

A project describes an installation. A **cell** is that project connected to real hardware and running.

What a cell is

A cell names what is really there and how to reach it. Until a project has one, everything it does is simulated — which is the right way to build a program, and the reason the demo project is safe to open on any machine.

Devices

A device is one thing the cell talks to: a robot, or a person at a screen. A robot device carries its address and the programs it can run. Adding a device is how the cell learns that this project drives that robot.

Flows

A flow is logic that runs in the cell rather than on the robot: watch a value, react to it, tell a device to do something. Flows are built from nodes rather than written as code — log a message, set a value, wait for a value to reach a target, run a program on a device.

Use them for the coordination between things. The robot's own motion belongs in a [program](#).

The operator HMI

The screens an operator uses are part of the cell, not of the robot. They are built from a small set of components — a button, a label, a selector — each bound to a variable, so pressing a button on a tablet sets the value a flow or a program is watching.

That is why the same screens open on a laptop, a tablet or the FANUC pendant: they are clients of the cell. What an operator does with them is [Operating a cell](#).

Licences

A licence key (APE-XXXX-XXXX-XXXX) is entered once in the licence panel, while the machine has internet. The key is then bound to that machine, unless the licence is floating.

With a security module (TPM 2.0)

The licence is bound to the machine's TPM and works without internet afterwards. This is the normal case for a production cell: real robot motion is only possible on such a machine.

Without a security module

The licence stays online: the software checks in with the licence server every hour and cannot start without it. Such a machine runs simulations only.

Floating licence

A floating licence belongs to no machine: each machine that activates it takes a seat, up to the number bought. It stays online, and drives a real robot when the plan allows it. **Release the seat** or **Forget this licence** in the licence panel frees the seat at once; the portal frees one too, and a machine silent for 14 days loses its own.

Demo

Without a licence, Stackmill edits, saves and simulates. Export is disabled, and files saved in demo mode only open on the machine that saved them.

Education

An education licence is a normal licence whose programs stop after a fixed run time. The operator restarts them; the software is complete for teaching and unusable in production.

Hardware change

After a board swap or a cleared TPM, the licence panel issues a request that Apemill approves online, or a file to carry to the licence portal when the machine is offline.

The 3D view

The scene as it is. Everything in the project can be seen, selected and moved here.

Navigating

Orbit, pan and zoom with the mouse. The view is a camera, not a state: moving it changes nothing in the project, so it is always safe to look from somewhere else.

Selecting

Click an object to select it; its details appear in the object panel and its handles appear in the view. Selection is shared across panels — selecting in the tree selects in the view.

The gizmos

A selected object gets handles to move and rotate it. Drag them for a quick placement; type the numbers in the object panel when the value matters. Most cells are built by dragging roughly and then typing exactly.

Display options

Objects can show helpers — frames, jog directions, gripper opening — and collision shapes can be shown as well as solid geometry. All of it is display only: nothing here changes what the robot will do.

Objects panel

The tree

Every object in the scene, arranged as a tree. Children move with their parents — see [Objects](#) for why that matters.

Drag an object onto another to change its parent. Expand and collapse to keep a large cell readable.

Adding an object

The buttons at the top add each type: robot, tool, reference, volume, position, work zone, and import. A new object arrives at the tree root or under the current selection; put it where it belongs before you place it, and its placement will follow the parent.

Object info

The panel beside the tree shows the selected object: its name, type, position, size, and the settings specific to its type. It is also where the switches live — collision on or off, helper shown or hidden, locked to its parent or not.

Locking

Locking an object to its parent stops it being dragged out by accident. Once a cell is correct, lock the things that should never move again.

Programs panel

The list

Every program in the project, in folders. The panel has tabs for programs, variables and configuration.

Folders

Folders are for you, not for the robot. Group by station, by product, by whatever makes the list readable — the export does not care.

Creating a program

Choose the kind when you create it: palletising, standard, calibration or external. The kind cannot be changed afterwards, because each is a different thing — see [Programs](#).

Recipes

A palletising program holds its [recipes](#). Selecting one shows what it produces: the layer, the stack, the count, and the trajectories.

Variables panel

The list

Every variable, with its type, its default and its current value. Generated variables are shown alongside yours but cannot be edited — they come from the project.

Adding a variable

Give it a name and a type. Names are what programs read, so name for what the value means rather than where it is used.

Linking to an object

A variable can point at an object in the scene — a position, a tool, a reference. This is what lets a program say "go to the pick position of the current recipe" and stay correct after the scene is edited. See [Variables](#).

Code view

Where a standard program is written and read.

Reading a line

Each line is one [instruction](#) with its arguments filled in. Lines are numbered, and blocks are indented, so the shape of the program is visible without reading it.

Adding an instruction

Insert a line and choose the instruction; the arguments it needs appear as fields. There is no syntax to get wrong — you cannot write a line that does not parse.

Blocks

`IF`, `FOR` and `WHILE` contain other lines. Lines inside them move with them, which is what makes restructuring a program safe.

Errors

A line that cannot work — a variable that no longer exists, a position that was deleted — is marked. A program with a marked line will not export.

Menus

Menu	Holds
File	New, open, save, and export to the robot
Edit	Undo, redo, and project-wide edits
View	What is shown in the 3D view, and the panel layout
Help	This documentation, the contact form, and Stackmill © with the version and the update check
Admin	Present only in an admin session: the cell's own settings and diagnostics

The licence badge and, in an admin or developer session, the update badge sit beside the menus rather than inside them — they report a state rather than perform an action.

Settings

Group	What it covers
Global	Language, units display, and the defaults a new project starts from
Startup	What happens when Stackmill opens, and which project it opens
Access	Who may do what: an operator session hides everything that edits or moves
Snapshots	How often the project is captured behind you, and how many are kept
Planning	How hard the trajectory planner works before giving up
Live	How the cell behaves when connected: what starts automatically, what needs confirming

Access is the one to set before a cell goes into production. An operator should be given a session that can start, stop, choose a recipe and recover — and nothing that can edit the project or move the robot by hand.

Live mode

The difference between a project you are editing and a cell that is running.

Entering live mode

A project with a [cell](#) can be taken live: Stackmill connects to the devices and the operator panels become real. Until then everything is simulated.

The live panels

Live mode replaces the editing panels with the operating ones — state, pallet view, controls, messages. They are documented from the operator's side in [Operating a cell](#), which is the right page to hand to whoever will use them.

Test mode

A cell can be exercised without the hardware: the panels behave as they would, driven by simulated state. Use it to check an operator screen before commissioning, and to train someone without occupying the cell.

Leaving live mode

Leaving live disconnects the devices and returns to editing. **It does not stop a running cell** — stop the cell first, and check that it has, before you go back to editing a project whose robot is still moving.

Building an operator HMI

The screens an operator sees are built in the project, not written in code.

The editor

A canvas holding the components of one screen. What you place is what an operator gets, on whatever device opens the cell.

The components

A small set on purpose: a **button**, a **label**, a **selector**, and a **separator**. A screen an operator can use under pressure is a screen with few things on it.

Binding to variables

Every component is bound to a [variable](#): a button sets one, a label shows one, a selector chooses among values. Nothing is wired directly to the robot — the variable is the contract, and a [flow](#) or a program does the rest.

Publishing to the panel

The screen is part of the project, so it reaches the panel when the cell does. Design it at the size of the real device — the FANUC tablet pendant, not your monitor — and test it in [test mode](#) before anyone depends on it.

A full palletisation

The whole job, once: from an empty project to a program a robot runs.

What we are building

One product, one pallet, one robot. A cell that picks boxes from a fixed position and stacks them until the pallet is full.

The scene

Add the [robot](#) and place it where it really is. Add a [reference](#) for the pallet station and place that. Everything else hangs off those two.

Model what the robot could hit — the fence, the conveyor, the cabinet — as [volumes](#) with collision on. It costs minutes now and saves a crash later.

The tool

Add the [tool](#) and mount it on the robot. Measure the [TCP](#) rather than estimating it, and enter the payload.

The container

Add the [container](#) under the station's reference, with the pallet's real dimensions. Add the pick position where boxes arrive.

The recipe

Create a palletising program and a [recipe](#) in it. Enter the part: its dimensions and its mass.

The disposition

Let Stackmill generate the layer arrangements, then choose. It orders them by how well they use the pallet; you choose on everything it cannot know — which face must point out, what the wrapper needs, how the boxes actually behave.

Set how layers alternate. Rotating alternate layers is what keeps a stack standing.

The trajectory

Generate the trajectories. Each drop gets an approach, a placement and a retreat, checked against the scene. Anything that cannot be reached, or cannot be reached without a collision, is reported here — before the robot ever moves.

Simulating

Run it. Watch the whole pallet build, and watch the last box of a layer as closely as the first: the corners are where reach and collision problems live.

Exporting

[Export to the robot](#). Stackmill writes the program and the register map.

Running it

Set up the [cell](#), take it live, and run the first pallet **slowly** — see [Speed](#). Then hand [Operating a cell](#) to whoever will run it.

Depalletising

What changes

Almost nothing. The same recipe describes the stack; running it in the depalletising direction takes it apart in reverse order instead of building it.

What does change is the world: a pallet arriving full is a pallet whose contents you did not place, so it may not be exactly where the recipe thinks.

Setting it up

Build the recipe as if you were palletising — see [A full palletisation](#) — then choose the direction. Check the first pick of the top layer and the last pick off the bottom: those are the two the geometry is most likely to get wrong.

Running it

The operator chooses the direction beside the recipe; see [Choosing a recipe and a target](#). A target by layer is more useful here than when building — it is how you take two layers off and stop.

Bundles and interlayers

Bundles

A bundle is several parts handled as one — a shrink-wrapped pack, or several boxes taken in a single grip. The recipe treats the bundle as the unit it places, so the pattern is computed from the bundle's dimensions, not the individual part's.

The tool has to be able to take the whole bundle in one motion. This is where a [multi-part grip](#) is described.

Interlayers

A sheet between layers. Declare its thickness in the recipe: an interlayer the recipe does not know about makes every layer above it land low, and the error adds up over the stack.

Stickers

Where a face must be labelled, the recipe needs to know which face and which way it points — which constrains the disposition. Decide this before choosing a layer pattern, not after: a pattern that puts half the boxes label-inwards is a pattern you cannot use.

Two stations

When two stations help

Two pallet positions let the cell fill one while an operator clears the other, so production does not stop for a pallet change. On a cell that fills a pallet in a few minutes, this is the difference between running and waiting.

Setting them up

Give each station its own [reference](#), and hang its container and positions off it. Two stations means two frames, and correcting one must not touch the other — which is exactly what references give you.

Each station is [calibrated](#) separately.

What the operator sees

Both stations appear in the pallet view, each with its own state and count. The cell chooses which to fill; the operator validates and clears. It is all in [Pallet stations](#), which is the page to hand over.

A calibration program

Calibration measures where something really is, and corrects the project to match.

What it measures

The position of a [container](#) — a pallet station — relative to the robot. That is the measurement everything else on the station depends on.

The probes

A probe is one touch: the robot approaches until it makes contact, and records where. Several probes on known faces are enough to place the station in space.

Probes need somewhere solid and repeatable to touch. Choose faces that are part of the station, not part of what happens to be on it.

Semi-automatic and automatic

- **Semi-automatic** — you bring the robot to each point. Works when the station has moved further than the automatic search will look.
- **Automatic** — the robot finds the points itself. Faster, and the routine choice.

Both are run from the operator screen; see [Calibration](#).

Applying the result

The measured correction is shown before it is applied, and refused outright if the probe found nothing or the correction is outside the accepted band. Look at a large correction before accepting it — accepting a bad measurement teaches the cell a wrong position, and every drop after it inherits the error.

A standard program

For the logic a recipe does not describe.

Creating it

Create a program of kind **standard**. It starts empty, as a list of lines.

The instructions

Add lines and fill in their arguments — the set is in [Programs](#). There is no syntax to learn and no way to write a line that does not parse.

Variables and conditions

Conditions are built from [variables](#). Where you need to refer to something in the project — a tool, a recipe, a position — use the generated variable rather than a number: it survives edits that a number does not.

Calling it from a recipe

A palletising program can call a standard program at defined moments — before a pick, after a drop, at the end of a layer. That is how a cell does something extra without you writing the palletising itself.

FANUC controller setup

What the robot needs before Stackmill can drive it. Do this once, when commissioning.

Controller options

Some of what follows depends on options being present on the controller. Check them before quoting a cell — an option that is missing is a purchase, not a setting. Ask us if you are not sure which your installation needs.

Controller options

Stackmill needs the controller to accept commands over the network and to run the programs it generates. Which option codes that requires depends on the controller generation.

Network

Give the controller a fixed address on the same network as the machine running Stackmill, and check that the two can reach each other before anything else. Most "the robot does not respond" problems are this.

REMOTE mode

The controller must be in **AUTO** with remote enabled, or it will accept nothing. Stackmill reports Robot not in remote mode and refuses to start rather than pretending.

Registers and I/O

Stackmill uses numeric registers, position registers and digital I/O to exchange state with the robot. The map is written when you [export](#). Avoid using the same registers for anything else — a register written by both sides is a fault that will look like something else.

Payload compensation 1.2.1

On a collaborative robot (CRX), Stackmill weighs the part the robot lifts (tare). It needs the **Dynamic payload** setting, first in the robot's **Configuration** tab (on by default), and this controller setup:

1. Set **both payload schedules 1 and 2 to the empty tool**: mass and centre of gravity of the tool alone. Schedule 1 is the empty tool, schedule 2 the loaded one; the robot adds the part to schedule 2 when it weighs it.

2. In the DCS collaborative robot setup, enable **Payload setting compensation** and set **Payload number to compensate** to 2 (see the FANUC collaborative robot manual, How to Estimate Payload Setting Compensation).
3. Apply the DCS changes and cold-start the controller.

Without it, turn **Dynamic payload** off: the payload schedules of the tools and the recipes are used as before, and there is no tare.

Program memory 1.2.1

A cell with many recipes can fill the controller's program memory: a program then fails to load. With FANUC's **TP DRAM/FILE Storage** option installed, tick **Shadow memory** in the robot's **Configuration** tab and export again. The generated programs are then stored outside the program memory and loaded when first run; the programs you edit stay where they are.

Without the option, leave it off: the controller ignores or refuses the setting.

Loading the Karel programs

Some functions run on the controller itself and are loaded once. After loading, cold-start the controller so it picks them up.

Exporting to the robot

What is exported

The robot's own program — the motions, the logic and the positions — plus the register map Stackmill uses to talk to it. Everything the robot needs to run the cell, and nothing that only matters inside Stackmill.

A program with an error will not export. That is deliberate: the last place to discover a broken reference is on the pendant.

Transferring

Send the export to the controller over the network, or carry it. Either way the controller ends up holding the program Stackmill generated.

Checking on the pendant

Look at the program on the pendant before running it — not to edit it, but to confirm that what arrived is what you exported and that the register numbers are the ones you expect.

Do not edit it there. The next export overwrites it, and a change that exists only on the pendant is a change nobody will find. Change it in Stackmill.

Re-exporting

Any change to the scene, the recipe or the program means exporting again. Names matter here: a renamed tool or program can change what the export is called, so check what you are overwriting on the controller.

Backup and restore

What to back up

- **The project file.** It holds the scene, the programs and the recipes — everything you built.
- **The robot's own program**, if it was edited outside Stackmill.
- On a server, the **data folder**, which holds the projects, the settings and the licence.

A project

One file. Copy it, or keep it under version control — a project is worth a history, and "yesterday's version worked" is a question you will eventually ask.

A server

Everything the server keeps lives in its data folder, which an [update](#) never touches. Back that folder up and you can rebuild the machine from nothing.

Restoring

A project opens on any machine with a licence that allows it — with one exception: a file saved in **demo mode** only opens on the machine that saved it.

Restoring a server means installing it again and putting the data folder back. Test that before you need it, not after.

FAQ

Licences

Does Stackmill need the internet?

Once, to activate the licence. After that a machine with a security module never needs it again. A machine without one checks in regularly and will not start if it cannot.

What happens if your licence server is down?

A machine with a security module is unaffected. A machine without one cannot start — which is one reason a production cell should have a TPM.

Can I move my licence to another machine?

Yes, by asking for a rebinding; a floating licence only needs its seat released on the old machine. See [Activate a licence](#).

What can the demo do?

Everything except export. It edits, saves and simulates, and files it saves open only on the machine that saved them.

Installation

Windows or Linux?

Both, for the desktop application. A server installation runs in Docker.

Do I need a server?

Only if you want operator screens on tablets or several people in the same cell. One person programming at a desk does not.

Programs

Do I have to write robot code?

No. Stackmill generates the robot's own program and you export it. What the exported program contains is in [Exporting to the robot](#).

Can I edit the generated program on the pendant?

You can, and it will be overwritten the next time you export. Change it in Stackmill.

The robot

Which robots are supported?

FANUC today. Other brands are on the [roadmap](#).

Which controller options do I need?

See [FANUC controller setup](#).

The cell

Can two robots share a pallet?

Not today. It is on the [roadmap](#).

How many pallets can a cell have?

One or two stations is the normal case, and what the operator screens are designed around.

Glossary

The words this documentation and the software use, in the sense they use them.

Cell

One robot, its tool, the pallets around it and the programs that drive it — everything that makes up one working installation. Also the thing you start and stop.

Container

What parts are stacked on or into: a pallet, a box. See [Container](#).

DI / DO

Digital input and digital output — the on/off signals the robot controller exchanges with the rest of the machine.

Frame

A reference for positions. A position means nothing on its own; it means something in a frame. See [Units and frames](#).

HOME

The position the robot returns to between jobs and after a recovery. Chosen so the robot is clear of everything.

Jog

Moving the robot a little at a time, by hand, while holding a control. See [Moving the robot by hand](#).

Payload

What the tool is carrying. The robot needs it to know how hard to accelerate. See [Speed and payload](#).

Pendant

The handheld or tablet screen attached to the robot controller. Stackmill's operator screens run on it.

Recipe

One product on one kind of pallet: the box, the pattern, the stack. See [Recipes](#).

Register

A numbered slot in the robot controller holding a value or a position. Stackmill writes the ones its programs need.

REMOTE

The controller mode in which the robot accepts commands from outside. Without it, Stackmill can talk to the robot but cannot make it move.

Station

One pallet position in a cell. See [Pallet stations](#).

TCP

Tool Centre Point — the exact point of the tool that the robot positions. Get it wrong and every position in the project is wrong by the same amount.

TPM

A security module in the machine. With one, a licence works offline; without one, it does not. See [Licences](#).

Work zone

A volume the robot must stay inside, or out of. See [Work zone](#).

Troubleshooting

Problems an operator meets are in [When it faults](#). This page is for the ones that stop you before the cell even runs.

The application

It will not start, or closes at once.

Start it again once. If it still fails, send us what the screen said and the version.

The 3D view is black, or very slow.

The machine is drawing with the wrong graphics device. Check that Stackmill is using the real graphics card rather than the integrated one.

The licence

"The licence server is unreachable."

The machine cannot reach us. That matters only at activation and, on a machine with no security module, at every check-in. See [Activate a licence](#).

The licence stopped working after a hardware change.

A new board or a cleared security module breaks the binding. Ask for a rebinding from the licence panel.

Export is disabled.

The software is running as a demo. A file saved in demo mode also opens only on the machine that saved it.

The robot connection

Stackmill cannot see the robot.

Check power and network first — the state bar says Offline and reconnects on its own once the robot answers.

The robot is reachable but nothing moves.

Almost always the controller is not in remote mode. The screen says Robot not in remote mode. See [FANUC controller setup](#).

The cell

The stack leans, or boxes land further out on each layer.

The cell's idea of where the pallet is has drifted. [Calibrate](#).

The same fault returns after every reset.

Stop and call us rather than resetting again. A fault that repeats is telling you something the reset is not fixing.

Safety

Stackmill moves an industrial robot

Everything on this site describes software that commands a machine capable of causing serious injury. Nothing here replaces the safety equipment of your cell, the training of the people using it, or your own risk assessment.

Before anything else

- Nobody inside the fence while the cell can move.
- The physical emergency stop is the emergency stop. The [soft stop](#) in the software is a convenience and depends on software and a network being alive.
- Anyone who may move the robot by hand must be trained for it. That page is written for trained staff and says so.

What Stackmill does not do

It is not a safety system, and it is not certified as one. It does not:

- detect people, or know that anyone is in the cell;
- replace the guarding, the interlocks, the light curtains or the safety controller;
- override the robot controller's own safety functions, which remain in charge.

Collision checking, work zones and trajectory validation exist to protect your parts and your cell from programming mistakes. They are a design aid, not a protective device.

A simulation is not a proof

A simulation is exactly as right as what it was told. A robot mounted a few centimetres from where the project says, a tool whose measurements were typed in wrong, a pallet not where the cell believes — each of these produces a simulation that runs perfectly and a cell that does not.

Run a new program slowly the first time, and watch it. [Speed](#) changes only the speed, never the path, so a slow first run tells you the truth about the path.

Your responsibilities

Commissioning an industrial cell, assessing its risks, guarding it and declaring its conformity are the responsibility of whoever puts it into service. If you are not sure what that means for your installation, ask us before you commission it — that conversation is part of what we sell.

Remote support

Sometimes the fastest way to solve something is for us to look at the machine while you are in front of it.

How to ask for a session

Write to contact@apemill.com with the version and what you are seeing. If a session is the right answer, we will agree a time — remote support happens **with someone present at the machine**, never on a cell nobody is watching.

What we can see

A session gives us access to that machine only, for as long as it lasts. We use it to read logs, look at the project and reproduce what you are seeing.

We do not open a session on our own initiative, and a machine with no route to the internet cannot have one at all.

Ending a session

The session ends when you end it, or when we do. If you want it to stop, stop it — the cell does not depend on it, and nothing we were doing is worth more than your certainty about who is connected to your machine.

What's new

Every feature added since Stackmill 1.0, newest first. If you are running an older version, this is what upgrading gives you — [Updates](#) explains how to take one.

Version	What it adds	Where
1.2.1	Setting up the router	read
1.2.1	Part tare	read
1.2.1	Running a calibration program	read
1.2.1	Program memory	read

Roadmap

We no longer publish dates. A date on a roadmap is a promise about a quarter, and it was never the thing anyone actually needed to know. What follows is what we are building and what we intend to build, in order of how close it is.

What has already shipped is in the [release notes](#); what changed since the version you are running is in [what's new](#).

Next

Work that is underway.

- **Production supervision** — parts counted, cycle time, downtime and alarms, kept over time and exported as CSV or JSON.
- **A production dashboard** — the same figures live, on the operator panel and in the browser.

Planned

Committed to, not started.

- Support for more robot brands alongside FANUC.
- German, in the software and in this documentation.
- Floating licences, for a team that shares machines.
- A trajectory timeline, with an estimated cycle time before the cell runs.
- Cells with several robots working together.
- Stability checking for a finished pallet.
- Mechanical grippers — jaws and cylinder grip. Today the tool is suction only.
- A linear axis (7th axis) driven by the robot.
- Importing CAD assemblies (STEP).
- Importing part of another project into the one you are working on.
- Several recipes running against different zones of the same cell.
- Changing a part's dimensions without stopping production.
- Predictive maintenance, from the robot's own diagnostics.

On request

Real requests from customers that we build when one of them needs it. If something here is what you need, [tell us](#) — that is what moves it up.

- Palletising inside boxes.
- Integration with a warehouse management system (WMS).
- Conveyors and automated guided vehicles.
- Importing and exporting pallet plans through an API.
- Loading pallets into a truck.
- Storage layout optimisation.
- 2D and 3D vision — measuring parts as they arrive, correcting the grip.
- Trajectories optimised for energy rather than speed.
- Several robots palletising on the same pallet.

Release notes

What changed, version by version, newest first. What is coming is on the [roadmap](#).

1.2.0

The current release.

- Live control: cells, devices and flows, with an operator HMI that runs on a tablet or the FANUC pendant.
- Server installation: an ISO that installs a ready machine, and an installer for a server you already run.
- In-application updates, with a check that never blocks and an install that never happens on its own.
- This documentation.