Twinwright Studio 1.0.2

Twinwright Studio manual

2026-10-11

Contents

User manual

1. Getting started
2. The window and the workspaces
3. Machines and power
4. Setting up a job
5. Probing and finding the work zero
6. Making programs (CAM)
7. Programs and the editor
8. Running a program and resuming
9. Analysis and diagnostics
10. Running a program in the twin
11. Inspection, reports and setup sheets
12. Connecting a real machine
13. The AI assistant
14. Language and units
15. Keyboard and accessibility
16. Troubleshooting

Operations manual

1. System requirements
2. Installing and removing
3. Licences, seats and activation
4. Your data and backups
5. Updating Studio
6. Privacy and what Studio sends
7. Labs, classrooms and shared computers

Technical manual

1. The twin's models and their limits
2. The Twinwright controller
3. G-code and M-code
4. CAM strategies and parameters
5. Fits, runout and inspection methods
6. Machine profiles
7. MC-1 hardware

Reference

1. MCP tool reference
2. Settings reference
3. Alarm codes
4. Error codes
5. Command line
6. Machine profile schema
7. Protocol
8. File formats
9. Sender software with the Twinwright controller

User manual

Getting started

Twinwright Studio runs a digital twin of your CNC machine. You set the part up, make or open a program and run it in the twin before the real machine cuts. This chapter takes you from installing to a first part in about ten minutes.

Installing

Studio needs 64-bit Windows 10 or 11 (see System requirements).

1. Run `TwinwrightStudio-<version>-setup.exe`. You do not need administrator rights: Studio is installed for you.
2. Read the licence agreement and click **I Agree**.
3. Keep the folder, or choose another.
4. On the last page, keep **Run Twinwright Studio** ticked and click **Finish**.

Studio is in the Start menu as **Twinwright Studio**. Installing for many computers, and what goes where: Installing and removing.

The first start

Studio opens in the **Run** workspace with the machine **MC-1 · My workbench** switched on. Nothing is asked.

The machine bar shows **ALARM:11** and **Next step: Home**. This is normal: like a real machine, the twin does not know where its axes are until it has homed.

What the window shows

A new installation shows everything an engineer uses: all six workspaces, the curves and the colourings. If you only run programs, or teach, choose a level that shows less: **Tools > UI level** (see User levels).

Language and units

Studio starts in your system's language when it has it (English, Chinese, French, German, Japanese or Spanish), and otherwise in English. The units and the way numbers look follow your region.

To change them: **Tools > Language / 语言** and **Tools > Units**. See Language and units.

Editions and the trial

The edition you have shows in the top bar, for example **Learning**. Click it, or use **Help > Licence...**, to see it.

Edition	Adds
Learning	Pockets, contours, drilling, engraving and 3D parallel finishing; the whole twin, the editor and the AI
Maker	Adaptive clearing, every 3D finishing strategy, rest machining, two sides and several setups
Pro	Nesting, the program optimizer, reports (the inspection report, the setup sheet), inspection with the probe against the design

Each edition has everything of the editions above it in the table. A feature of a higher edition shows with its edition's name. Click it to read what it does, and to start a trial or open the licence.

When a trial or a licence ends, Studio goes back to the Learning edition. Your projects, programs and the G-code you made keep working.

Starting a trial

You can try the Maker and the Pro edition for 30 days each, once per computer.

1. Click a feature of that edition, for example **Optimize** in the **Program** workspace.
2. Click **Start a 30-day trial...**
3. Click **Start the trial**.

Studio asks Twinwright's licence server for the trial once, sending the edition, a hash of the computer's fingerprint and Studio's version. When the trial ends, a notice in the **Messages** tab says so.

Activating a licence

Open **Help > Licence...**.

To activate online:

1. Type the activation code (`TW-XXXXX-XXXXX-XXXXX-XXXXX`). Case, spaces and hyphens do not matter.
2. Click **Activate**. Result: **Activated.**

To activate a computer without the internet:

1. Under **Activate offline**, copy this computer's request code.
2. On a computer with the internet, open the licence page, paste the request code and download the licence file (`.twlic`).
3. Bring the file back and click **Import a licence file...**

Renew before it ends renews the licence online before it runs out. It sends only the licence's number, a hash of this computer's fingerprint and Studio's version.

Seats, moving to another computer and classrooms: Licensing.

A first part in ten minutes

The quickest first part is one of the example parts: Studio sets up the blank, the tools and the work zero for you.

Open the machine

The machine is already open: **MC-1 · My workbench**, in the **Run** workspace. Other machines are in **File > Machine library...**.

Power on and home

Press **Home** (F3), or the key the machine bar suggests.

Result: the three axes find their home one after the other, in about 20 s. The machine bar shows **No alarm**.

If the machine bar shows **E-stop**, release it first (F1), then press **Power on** (F2) and **Reset** (F12).

Open an example program

1. Choose **File > Example parts...**
2. Choose **NEMA 17 motor mount plate** (e01_motor_mount): a fitted bore, counterbores and slots, cut free on bridges. Result: Studio places the blank, loads the tools and the program, and sets the work zero on the blank. The console says when the example is ready.

Run it in the twin

1. Press **Cycle start** (F5).
2. To watch it faster, choose **Simulation > Speed > ×10 real time**.

The 3D view shows the tool cutting the blank. **Feed hold** (F6) pauses the program, **Cycle start** goes on.

Look at the result

- Drag in the 3D view to turn it; the wheel zooms.
- **View > Colouring > Workpiece: deviation from the part** colours what is left to cut and any gouge.
- **Analysis** (Alt+5) shows which lines took the time and the twin's estimates of force and chatter.

Set up the blank

For your own part, set the machine up in **Setup** (Alt+1) instead of loading an example:

1. Go to the step **Setup**.
2. Under **Workpiece and setup**, choose the **Shape**, the **Material** and the **Fixture**, and type the sizes.
3. Click **Place**.

See Setting up a job.

Measure the tool

1. Go to the step **Tool**, choose the tool and click **Change tool**.
2. Click **Measure reference tool**.

The machine must be homed and the spindle stopped.

Set the work zero

1. Go to the step **Zero** and jog the tool to the point you want as zero.
2. Click **Zero all**, or zero the axes one by one.

With a touch probe, the probe card finds the zero for you (see Probing).

Then open your program (**File > Open program...**) and run it as above.

Where to go next

- The window and the workspaces: what is where.
- Making programs: CAM in Studio.
- Running a program in the twin: what the twin checks and estimates.
- Running a program and resuming: runs, alarms and starting again.

A **?** on a panel opens this manual at that panel's chapter. **Help > Documentation** opens it at the start.

The window and the workspaces

Studio's window has the menus and a top bar at the top, six workspaces in the middle and the machine bar at the bottom. Each workspace does one main thing. The machine bar shows the machine's state and the next step wherever you are.

The parts of the window

From top to bottom:

- the **title row**: the menus on the left, the window's name in the middle, the window's keys on the right;
- the **top bar**: the machine, the workspace tabs, the edition, the AI's state, the user level and **?**;
- the **workspace**: the 3D view and the panels of the workspace you chose;
- the **dock** under the 3D view, in the workspaces that have one;
- the **machine bar**, the same in every workspace.

Studio has no status bar: the machine bar does that job.

The title row and the menus

The menus are **File**, **Edit**, **View**, **Machine**, **Program**, **Simulation**, **AI**, **Tools** and **Help**.

- Press Alt and a menu's letter to open it: Alt+F opens **File**.
- Press Alt alone to light the menu titles, then use the arrow keys, Enter and Esc.

While a menu is open, no key reaches the machine. A menu shows only the items of your user level (see User levels).

The top bar

From left to right:

- the brand mark: it opens **File > Machine library...**;
- the machine: its model, its name and its state. Click it, or press Ctrl+Shift+M, to switch machines: type to filter, choose with ↑ ↓ and Enter;
- the workspace tabs (see The six workspaces);
- the edition, for example **Learning**: click it to open **Help > Licence...**;
- the AI's state: **AI off**, **AI on**, **AI connected** or **AI awaits your OK**. Click it for the AI's settings (see The AI assistant);
- the user level: click it to choose another level;
- **?**: the list of keyboard shortcuts.

The machine bar

The machine bar runs along the bottom of the window. From left to right:

1. **The state**: **Idle**, **Run**, **Jog**, **Homing**, **Hold**, **Door**, **Tool change**, **Check mode**, **Sleep**, **Alarm**, **E-stop** or **Off**.
2. **The program**: its name, the line it is on, the section it is in, a progress bar, and the time run and the time left. With no program it shows **No program loaded**, whether the machine is homed, the tool and the work offset.
3. **The next step**: **No alarm** in green, or **Next step:** and the key that does it. In an alarm it shows the alarm, the next step and **All steps**, which lists the steps back to work (see Alarms and what to do).
4. **The clock**: the simulation's time, its speed and the real speed it reaches. The hourglass key pauses the simulation.
5. **The keys**: **Single block**, **Cycle start** (F5), **Feed hold** (F6) and **Stop** (F7). The key Studio suggests is filled in; a key you cannot use now is dimmed.
6. **E-stop** (F1), the round red key at the right end.

In a narrow window the bar leaves out the clock's time first, then the program's section and times (they move into the tip), then the clock's speeds, then the words of the next step.

The 3D view

The 3D view shows the machine, the workpiece and the tool as they are now.

- Drag with the left button to turn the view, with the right or middle button to move it. The wheel zooms towards the pointer.
- The keys at the top left choose a view: isometric, front, right or top. **Follow tool** keeps the tool in the middle. **Display** turns the tool-tip trail and the program path on and off.
- **Tool close-up** shows the tool at the cut in a small picture at the top right. It shows in the **Run** workspace while a program runs.
- The chips at the bottom name the workpiece and the tool.
- Click the workpiece or the trail: the bar under the view tells you which program line made that point, and the listing goes to that line.

The same choices are in **View > 3D view**. **View > Colouring** colours the trail and the workpiece by feed rate, cutting force, chip thickness, chatter margin or program line (Engineer level and up).

The dock

The dock sits under the 3D view. Its tabs depend on the workspace:

Workspace	Tabs
Run	**Program**, **Curves**, **Console**, **Messages**, **AI activity**
Setup	**Inspection results** (once a part has been measured), **Tool table**
CAM	**Check results**, **AI activity**, **Versions**, **Program preview**
Program	**Console**

Analysis, **Diagnostics** and shop mode have no dock.

- **Program**: the program's listing. **Follow the line** keeps the line being run in sight; the time bars show each line's time in this run or the last one.
- **Curves**: the run's readings over time.
- **Console**: what Studio and the controller said. Type a line of G-code or a `$` command in the line at the bottom and press Enter to send it; ↑ and ↓ bring back the lines before.
- **Messages**: warnings and findings, grouped by where they come from. A red number on the tab counts the warnings.
- **Tool table**: the machine's tools.
- **Inspection results**: the part last measured with the touch probe, check by check - its design, tolerance, measurement, deviation and the twin's estimate (see Inspection).
- **AI activity**: what AI clients did on this computer's machines.

Press Ctrl+J, or click the arrow at the right end of the tabs, to fold the dock to its tab row.

The six workspaces

Switch workspaces with the tabs in the top bar, with Alt+1 to Alt+6, with Ctrl+Tab and Ctrl+Shift+Tab, or from the **View** menu. Pressing **Cycle start** in **CAM** or **Program** takes you to **Run**. While a program runs, the **Run** tab has a green dot.

Setup

Setup (Alt+1) gets the machine and the part ready, one step at a time: **Power**, **Home**, **Setup**, **Tool**, **Zero**, **Program** and **Cut**. The panel on the right follows the next step until you pick one. See Setting up a job; the machines, power and homing: Machines and power.

CAM

CAM (Alt+2) makes programs: the job's setup, shapes and operations on the left, the chosen item's fields on the right, **Generate program** at the bottom. See Making programs. Engineer level and up.

Program

Program (Alt+3) holds the programs and the example parts, the editor and a preview with the program's checks. See Programs and the editor.

Run

Run (Alt+4) runs the loaded program. The right column shows the position, the spindle and feed, the overrides, the cut, the panel's keys, manual control, and the tools and zero. See Running a program and resuming.

Analysis

Analysis (Alt+5) looks back at runs: their curves, each line's time and forces, and the stability lobes. Engineer level and up. See Analysis and diagnostics.

Diagnostics

Diagnostics (Alt+6) shows the machine's power chain, the controller's inputs and the drives, and lets you inject faults into the twin. Technician level and up. See Analysis and diagnostics.

Messages and what to do next

Studio tells you what happened, and what to do, in these places:

- the machine bar: the next step, or the alarm and the steps back to work;
- the dock's **Messages** tab: warnings and findings;
- the **Console**: each line from the controller and from Studio;
- **Tools > Audit log...** (Ctrl+L): everything, with who did it and when.

Machine > Do the next step does the step the machine bar shows, and only that step. When an alarm stopped a program, the steps end with the line to start again from (see Starting again from a line).

User levels

A user level shows what a person needs at that stage and hides the rest. There are five:

Level	Adds
Demo	The **Run** workspace with the example parts; the **Program** and **Messages** tabs
Operator	**Setup** and **Program**; the overrides, the panel, manual control, the tools and zero; the **Machine** submenus; the console, the tool table, **AI activity**; settings, the audit log, the **AI** menu; switching machines
Technician	**Diagnostics**; arranging the workpiece; **Factory reset…**
Engineer	**CAM** and **Analysis**; the cut readings, the curves, **Colouring**, exporting curves, the part to compare with
Researcher	The **Model and data** card in **Diagnostics**

Each level has everything of the levels above it in the table. A new installation starts at **Engineer**.

To change the level, use **Tools > UI level**, the level in the top bar, or **Tools > Settings… > General**.

> **Note**
>
> A level arranges the window for teaching and learning. It does not lock anything: the machine's keys work at every level.

Light, dark and shop mode

View > Appearance chooses **As the system**, **Light** or **Dark**, and turns **Shop mode (large)** on or off. The two choices are separate: shop mode works in either look.

Studio's own panels change at once. Lists and text boxes that the system draws take the new look at the next start.

Shop mode is for working at the machine, at arm's length:

- the machine bar, its keys and the E-stop are larger, and so are the text and the listing;
- **Run** has its own layout: the position, the program and a large listing on the left, the gauges under the 3D view, the overrides, the panel's keys and the messages on the right, no dock;
- the 3D view has two view keys and no tool close-up.

Panels and their sizes

Drag the edge between two panels to change their sizes; double-click the edge to go back to the design size. The 3D view keeps at least 360 × 240 pixels at 100 % display scale. Panel edges move with the mouse only.

Studio keeps, for each machine, the panels' sizes, the dock's tab and whether it is folded, and the workspace you were in. It keeps the window's place and size for the next start.

Machines and power

The machines in Studio, how to switch between them, and how a machine is powered on and homed before it works.

Machines

Each machine keeps its own world: its position, its workpiece, its tools and offsets, its temperatures and its running hours. The machine in your window is the one that runs; the others wait as they were saved. In 1.0 one machine runs at a time.

Switching machines

1. Click the machine in the top bar, or press Ctrl+Shift+M (**File > Switch machine...**).
2. Type to find a machine, choose it with ↑ ↓, and press Enter. Result: Studio saves the machine you leave and starts again with the other one. That machine starts cold: power it on and home it.

You cannot switch while a program runs. The list also shows the **Practice machines** that AI clients use in the background; **View (read-only)** shows them in **AI activity**.

The machine library

File > Machine library..., or a click on the brand mark, shows your machines with their state, workpiece and tool, and the machine models. Click the brand mark again to go back.

For the selected machine:

- **Open**: switch to it.
- **Rename...**: give it a name of your own.
- **Copy as a new machine...**: a new machine with this one's world as it was last saved.
- **Show its folder**: its world on disk.
- **Factory reset... (two clicks within 5 s)**: the machine as it came (see Starting a machine afresh).

New machine... adds a machine of the model in its factory state. New machines, copies and factory resets need the Technician level. The **Real MC-1** row is a place held for later: in 1.0 a real machine is connected from the command line (see A real machine).

Power and homing

Powering on

The twin has the MC-1's power chain: the main switch QF1 gives the controller 24 V, and **Power on** closes the contactor KM1, which gives the drives 48 V and powers the spindle drive. The E-stop opens KM1.

1. Go to the step **Power**.
2. If the main switch is off, click **Main switch on**.
3. If the key reads **Pressed**, the E-stop is pressed: click it, or press F1, to release it.
4. Click **Power on** (F2). Result: the panel shows 48 V on and each drive **Ready**. The next step is **Home**.

Homing

Click **Home** (F3). Z goes up first, then X and Y together, each to the switch at its end of the travel; at real time this takes about 20 s.

Result: **Homed: X Y Z**, and the soft limits are on. Until the machine is homed after a power-on, the controller is in ALARM:11, and the tool setter, the probe and **Go to zero** wait.

Soft limits and the travel

The MC-1 travels 200 mm in X, 150 mm in Y and 80 mm in Z. After homing, each axis's machine coordinate is 0 at its home end: X runs from −200 to 0, Y from −150 to 0 and Z from −80 to 0, with 0 at the top of Z.

- A jog that would go past a limit is not done (error:15). A held jog stops just short of it.
- A program line that would go past a limit stops the program with ALARM:2. The position is kept: **Reset**, **Unlock**, then check the program and the work zero.
- An axis that hits a limit switch raises ALARM:1: **Reset**, **Unlock**, jog off the switch, then **Home**.

Setting up a job

The **Setup** workspace gets the machine and the part ready, one step at a time: power, homing, the blank and how it is held, the tools, and the work zero. Every step shows in the twin first, so you see the setup before anything moves. The machines, power and homing have a chapter of their own: Machines and power.

The Setup workspace

Setup (Alt+1) has the steps on the left, the 3D view in the middle and the chosen step's panel on the right. The dock under the view holds the **Tool table**.

The steps

Step	Its panel	Done when
Power	The main switch, **Power on**, **Reset**, **Unlock**	The controller runs and the drives have power
Home	**Home**, **Unlock**, **Reset**	All three axes are homed
Setup	The blank, how it is held and where	A blank is set up, or the table is left empty
Tool	The tool in the spindle, the tool setter	The reference tool is measured, a tool table is loaded or the work Z is set
Zero	Jogging, the work zero, the probe card	A work offset is set
Program	**Open program…**, **Example parts…**, **Check program**	A program is loaded
Cut	**Cycle start**, **Single block**	The program ran to its end

A step that is done has a green tick; the next step has a small arrow at its end. The panel shows the next step until you click another one; click the next step to have the panel follow again. The line under the steps says what to do now. The **?** on a panel opens its part of this manual.

The 3D view while you set up

While you change the blank's fields, the blank you are setting up is drawn see-through on the table: green when it can be set up like that, red when it cannot. **Place** puts it there; **Revert** or Esc drops the change.

The blank

Shape, material and sizes

In the step **Setup**, on the card **Workpiece and setup**:

1. Choose the **Shape**: **Block**, **Round bar (upright)**, **Tube (upright)**, **Hex bar (upright)**, **Round bar (lying)**, or **STL file (import the shape)…**.
2. Choose the **Material**, or type its name from another standard (C45, EN AW-6061, SUS304): the list shows the grade whose cutting data Studio takes, and the setup keeps the name you wrote.
3. Type the sizes the shape needs: **Length L**, **Width W**, **Height H**, **Diameter D** and so on.
4. Choose how it is held and where (below), then click **Place**. Result: the panel says what is on the table and its mass.

Sizes are in the display unit; in inches the fields also take fractions (see Inch fractions).

> Note
>
> A name from another standard is a near grade: the cutting data come from the nearest grade in the list. The material is what the supplier's certificate says.

A blank from an STL file

Choose **STL file (import the shape)…** in the shape list and pick the file (in mm). Studio sets the shape down on its lowest point and centres it; its bounding box gives the sizes. Material under an overhang counts as solid. If the file cannot be read, the console says why.

Holding the blank

Choose the **Fixture**: **Clamps · both ends**, **Clamps · both sides**, **Clamps · four sides**, **Vise**, **V-block (a bar lying down)**, **Double-sided tape**, **Not fixed (just placed)** or **Ideally fixed (for scripts)**.

The blank and its fixture must stay on the table (300 × 220 mm) and clear of the tool setter at the back left. While a program runs, the twin compares the cutting force with what the fixture holds: it warns when the margin is small, and a blank that slips moves in the twin, so what is cut after it is off.

Vise and parallels

Above the jaws is how far the blank's top stands above the jaws: 5 mm unless you change it. Studio chooses the parallels that give at least that. The vise opens from 3.25 to 77 mm, and the jaws must grip at least 2 mm of the blank.

Clamps

Strap clamps on studs in the table's tapped holes (M6, on a 25 mm grid): **both ends** along the blank's length, **both sides** across it, **four sides** both. Each clamp needs a hole 7–35 mm outside the blank's edge; when there is none, the panel names the clamp that finds no hole. Use the V-block or the vise for a bar lying down.

Double-sided tape

The tape holds the blank by its area on the table. Use it for light cuts and flat parts, not for a bar lying down. Once it lets go, only friction holds the blank. **Not fixed (just placed)** holds by weight and friction alone; **Ideally fixed (for scripts)** never slips, for scripts and tests.

Room above the blank

The programs that the CAM writes use the room above the blank like this:

- A program starts, ends and changes tools at the top of the Z travel.
- Between operations, the tool moves across at the safe height: 5 mm over the blank's top, or 2 mm over a clamp that stands higher than the blank.
- Where the set-up leaves less room than that (a long tool, a high vise, a tall blank), Studio lowers the safe height to what there is, at least 0.3 mm below the top of the travel, and says so with the program. When that would be less than 0.5 mm over the blank, it does not write the program and says why.

The setup sheet's **Z clearance** table checks the touch probe and every tool for each setup. See Z clearance.

Placing the blank

Position X and **Y** place the blank's middle in the twin's table coordinates (the table's centre is X 100, Y 75). **Angle °** turns it counter-clockwise, seen from above.

- **Place**: a new blank there. The blank before and what was cut in it are gone.
- **Move here**: the same blank moved, turned or held another way. What was cut stays.
- **Remove**: the blank off the table.
- **Revert**: the fields back to what is on the table.

Setting up waits while the spindle turns, the machine moves or a program runs.

Arranging in the 3D view

Click **Arrange workpiece** (Technician level and up), or use **View > 3D view > Arrange workpiece**. Then:

- drag the blank to move it, the orange handle to turn it, the blue handles to size it (with Shift in fine steps, with Ctrl in coarse ones);
- or use the keys: the arrow keys move it 1 mm (Shift 0.1 mm, Ctrl 10 mm), Q and E turn it 15° (Shift 1°), Enter sets it up, Esc puts it back.

Leaving the **Setup** workspace ends arranging.

Flipping the blank

Flip turns the blank over in its fixture, 180° about its length: top and bottom change places, and so do front and back. What was cut turns with it. The work offsets stay as they were, so set the work zero again (see Probing).

For a part machined from two sides, CAM plans the turning over between its setups (see Jobs with several setups).

Tools and the tool table

The dock's **Tool table** lists the machine's tools: the T number, the tool, and the specification the program reads. The tool in the spindle is marked **In the spindle**. The table comes with an example part, a CAM job or an AI client; you cannot change it in the window.

With a tool table, the program changes tools by itself (`M6`): the tool goes in, the machine measures it on the tool setter and carries on. Without one, an `M6` waits for you to put the tool in (**Change tool** in the step **Tool**), then **Cycle start**.

Changing the tool in the spindle

1. Go to the step **Tool**.
2. Choose the tool in the list **Tool**, for example **Flat end mill Ø6, 2 flutes (stick-out 15.5)**.
3. Click **Change tool**.

The spindle must stand and the machine be idle. If the new tool is longer or shorter, measure it (**Set tool offset**) or set the work Z zero again. The list has the standard tools, metric first (inch first when Studio shows inches). How a tool is described, for example `flat 6 n2 L15.5`: Tools and the tool table.

Measuring tools with the tool setter

The tool setter stands at the back left of the table. The machine must be homed and idle, and the spindle stopped.

1. With the tool you set the work Z zero with in the spindle, click **Measure reference tool**.
2. After each tool change, click **Set tool offset**. Result: Studio measures the new tool and sets its length offset (`G43.1`) from the difference.

The same keys are in **Machine > Tool setter and probe**. The reference is kept until Studio closes: measure it again after a restart.

The part to compare with

With the part's design, the twin shows where the cut part differs from it (Engineer level and up).

1. Set the work zero first: the part is placed at the program's zero when you load it.

2. In the step **Setup**, under **Part comparison**, click **Part STL (compare)…** and choose the part's STL file (in mm, its origin at the program's zero). Result: the workpiece is coloured by its deviation from the part (**View > Colouring > Workpiece: deviation from the part**): within ±20 µm counts as good, the other colours show stock left and gouges.

Stop comparing removes the part. The example parts load their own.

Saving and opening setups

- **Save setup…** writes the card's setup (the shape, sizes, material, fixture and place) to a JSON file.
- **Open setup…** reads one back into the card; click **Place** to set it up. It also opens an STL file as the blank's shape.
- **Export workpiece…** saves the workpiece as it is cut, as STL (in program, blank or machine coordinates) or as a 16-bit height map (PNG).

Each machine keeps its setup and its cut workpiece in its world when the window closes.

Jobs with several setups

A part that is machined from two sides, or held in two ways, is one job with several setups. In the **CAM** workspace, **Setup…** adds a setup to the job. The job tree then has a level for each setup, with its operations and its program under it, and a **Turning over** step between two setups. A job with one setup has no such level.

- Each setup has its own operations and its own program. The programs are named after the job and the setup's number, for example `plate-1.nc` and `plate-2.nc`.
- A setup starts from the part as the setups before it left it. In the twin, Studio runs setup 1, turns the part over as the **Turning over** step says, and writes the next setup's program for that part. In the 3D view, what was cut before and is now underneath is drawn dashed.
- The tree calls a setup front or back from how many times the part has been turned over.
- Under an operation's panel, **This setup** shows the setup's fixture, its zero, how the part was turned and the program.
- **Check results** has a row for each setup and one for the turning over. **Total** adds up the time at the machine and gives the worst chatter margin and the largest gouge. It also compares the finished part, after the last setup, with the design part.

A job has at most 8 setups. A nested job cannot have setups yet. Several setups are in the Maker edition and up (see Editions).

The setup sheet has a section for each setup, with what to do between them. See Setup sheets.

Turning the part over between setups

Select **Turning over** in the job tree. Its panel says how the part comes to the next setup. The settings belong to the setup after the step.

Setting	Choices
How it turns	**About U, 180°**: the left and right ends stay where they are; front and back change places, and so do top and bottom. **About V, 180°**: the front and back edges stay; left and right change places, and so do top and bottom. **Not turned (held another way)**: the part stays as it lies, and only the fixture changes.
Zero	**Front left top corner** or **Centre of the top**: where the work zero is found again (see Carrying the work zero over)
Fixture	How the part is held from this setup on: the same fixture, or another one

U runs along the machine's X axis and V along its Y axis, through the part as it lies in the setup before.

What the operator does lists the steps at the machine: let the program of the setup before finish (spindle stopped, coolant off), unclamp, turn the part over, clamp it, set the work zero again, and load the next program. **In the twin** says what the twin does with the part.

> **Caution**
>
> Turn the real part as the step says. The next program is checked in the twin for the part turned that way only. A part turned about the other axis, or not turned at all, is cut where nothing was checked.

Carrying the work zero over

When the part is turned over or held in another fixture, its corners are no longer where they were: the work zero (G54) has to be found again. The job tree says so next to the setup, and the setup's program starts from the zero you chose:

- **Front left top corner**: the front left corner of the part's top, as the part now lies. This is the zero after a turn or a change of fixture unless you choose another.
- **Centre of the top**: the middle of the part's top, for example for a round part.

Find the zero with the touch probe (see Probing). The setup sheet gives the probe steps for each setup.

When the part is neither turned nor held another way, the zero of the setup before is kept and not found again. A second setup that only uses other tools works like this; its section on the setup sheet says that the work zero is not touched again.

Probing and finding the work zero

With a touch probe in the spindle, Studio finds an edge, a corner, a bore's or a boss's centre or the top face, and sets the work zero from it. The probe card checks the cycle against the setup before it starts, and the twin runs it as the machine would.

The probe card is in **Setup**, step **Zero**, under the jog keys. In 1.0 the probe card works on the twin. On a real machine, probe with the machine's own control.

Before you start

You need:

- the blank set up on the table (step **Setup**);
- the touch probe in the spindle (see The probe in the tool table);
- the spindle stopped.

The machine does not have to be homed: a cycle that finds the machine in an alarm or not homed resets and homes it first, and the card says so.

> **Caution**
>
> On the MC-1, the vise with the 30 mm probe leaves no room above the blank (see Room above the blank). Hold the blank on double-sided tape or under clamps when you probe.

The probe in the tool table

1. In **Setup**, go to the step **Tool**.
2. In **Tool**, choose **3D probe Ø2 ball (stick-out 30, for edge finding)**.
3. Click **Change tool**.

Result: the probe card's first line, **In the spindle**, names the probe. When the spindle holds anything else, the card says to put the probe in first.

In a tool description the probe is written `probe 2 L30`: a 2 mm ball, 30 mm out of the holder.

Room above the blank: the Z budget

With Z at the top of its travel, the probe's tip has to clear the highest point of the setup: the blank's top, the vise's jaws or the clamps. Studio works this out for the tool in the spindle and for each tool in the tool table whenever the setup or the tools change. When a tool cannot get over the setup, the **Setup** step says how much it is short and suggests a shorter tool or a lower setup (clamps instead of the vise).

The probe card checks the same for the cycle: it moves the probe across at a safe height above the blank and the fixture, and shows that height in the checks.

The probe card

From the top:

- **In the spindle**: the probe, and whether it is calibrated (**Calibrated** with the time, or **Not calibrated**);
- **Touch-probe cycles**: the six cycles and the chosen one's fields;
- the checks;
- **Show the path in 3D** and **Start probing…**.

After a cycle, the card shows its result instead (see Setting the work zero from what was found). Under the probe card is the **Measure the part** card, and under that the **Reports** card (see Inspection).

Number fields take the display unit, and in inches also fractions. A field that does not read turns red and says why (see Typing numbers).

The checks

The card checks the cycle while you set it up, before anything moves. Each check is green (✓), amber (!) or red (✕). Any red check keeps **Start probing…** grey.

Check	Green	Amber or red
The probe	**A touch probe in the spindle, not triggered**	No probe in the spindle; the probe's input already triggered
The machine	**The spindle stands, the machine is homed and idle**	Amber: in an alarm or not homed, the cycle homes first. Red: the spindle turns, or the machine is busy
The soft limits	**Every move within the soft limits**	A move goes outside them
The safe height	**Safe height Z …**, and how far above the blank and the fixture	No room above the blank to move across
The depth	**Depth … below the top**, and for a blank in the vise how far it stands above the jaws	The ball would go below the jaws, or too close to the blank's bottom

When the twin follows a real machine, a red line says to probe there with the machine's own control.

Trying a cycle in the twin first

The checks come from a dry run: the twin works the cycle out without moving anything. **Show the path in 3D** draws its moves in the 3D view, in place of the program's path, until you leave the step.

Each touch goes the same way:

1. a protected move to the start (`G38.3`, F600);
2. a fast touch (`G38.2`, F300);
3. back off;
4. a slow touch (F20): the result is where this one touched.

Probing cycles

Choose a cycle, fill in its fields, and click **Start probing...**. Each cycle starts 3 mm from where it expects the face and searches 5 mm further. **Depth** is how far below the blank's top the ball's centre touches (3 mm unless you change it). **Start**: **From the blank** works from the blank in the setup; **Given X Y** starts from a point you give in work coordinates.

Edge

Finds one face of the blank. **Which face**: **Left (X−)**, **Right (X+)**, **Front (Y−)** or **Back (Y+)**. **Points**: **1 point**, or **2 points (the turn too)**, which also gives how far the blank is turned.

Result: **Face X** or **Face Y**, and with two points **Turn**.

Corner

Finds a corner from two faces. **Which corner**: **Front left**, **Front right**, **Back left** or **Back right**; **Outside** is the blank's corner, **Inside** a pocket's corner (give its place with **Given X Y**). **From the corner** is how far from the corner each face is touched (5 mm). **3 points (the turn too)** touches the X face twice and also gives the turn.

Result: **Corner X**, **Corner Y**, and with three points **Turn**.

Bore and boss

Bore centre touches a bore from its middle outwards; **Boss centre** touches a boss from outside, over its top. **Points**: **4 points** or **8 points**. **Diameter** is the nominal size; empty takes it from a round blank. A block blank needs the diameter and **Given X Y**.

Result: **Centre X**, **Centre Y** and **Diameter**.

Top face

Z surface touches the top face once, at the blank's middle or at **Given X Y**.

Result: **Surface Z**.

Setting the work zero from what was found

The result card shows each value in machine and work coordinates, each touch, and how the probe's calibration was applied. When the blank is turned, it says by how much; you can set the setup's angle to it in the **Setup** step.

To make the result the work zero:

1. Click **Set the zero...**
2. Tick the axes to set. Axes the cycle did not measure cannot be ticked: a corner gives no Z. Result: the dialog shows the line it will send, for example `G10 L2 P1 X-129.982 Y95.011`.
3. Click **Set as the G54 zero** (or the work offset that was in use).

The zero lands on what was found: the face, the corner, the centre or the surface. **Copy the program** copies the lines the cycle sent. **Measure again** goes back to the fields.

> Note
>
> The twin's numbers show the method and the flow. They are not this machine's measuring accuracy.

Calibrating the probe on a ring gauge

A probe triggers a little after it touches (pretravel), and not the same in every direction. Calibrating on a ring gauge measures this, and later cycles are corrected by it.

1. Set the ring gauge on the table and choose **Ring calibration**.
2. Type **Ring diameter** as the ring's certificate says.
3. **Position**: **From the blank (the tube's bore)** when the ring is set up as the blank, or **Given** its centre.

4. Click **Start calibrating…** Result: **Effective ball radius**, **XY pretravel**, **Three lobes** and **Ring centre**.
5. Click **Keep for this machine**, or **Don't keep**.

The calibration belongs to the machine and to that ball diameter, and goes with the machine's world archive. A factory reset forgets it. Each result says which calibration it used. Until you calibrate, the card shows **Not calibrated**: a bore then measures two pretravels too large, and a boss two too small.

When a cycle stops

When a cycle cannot finish, the result card says why in red, the console logs it, and **Measure again** takes you back to the fields. The usual reasons:

- a touch met nothing within its search: the face is not where the cycle expected it. The controller raises ALARM:5;
- the probe touched something on the way to a touch: something stands where the blank should not be;
- the touches do not make a circle (bore, boss, ring).

After an alarm, follow the steps on the machine bar: **Reset**, then **Unlock** (see Alarms and what to do). The probe may stand anywhere when the cycle stopped: look at the 3D view before you jog.

> **Caution**
>
> If the probe touches while the machine is in feed hold, press **Reset** and probe again. **Cycle start** would drive on past the contact.

Making programs (CAM)

Studio computes toolpaths itself. You describe the part as features: a pocket, a contour, holes, an engraving. Studio chooses the cutting data with the twin's physical models and writes the G-code. It then runs the program in the twin before you cut.

What the CAM does

- Shapes are data. A feature's shape is a rectangle, a circle, a polygon, a path from an SVG or DXF file, text, or a mesh (STL, 3MF, OBJ) for 3D work.
- Toolpaths are computed. Studio offsets the shape, clears the area, and links the moves.
- The twin chooses the cutting data. Speed, feed and depth come from the material, then from the twin's models of the spindle's power, the cutting forces, chatter and the tool's deflection. They are estimates.
- The twin checks the program. Studio can run the program in the twin before you cut, and change what the run shows to be too much.

Everything runs on your computer. Making a program needs no internet connection.

The quickest way is to describe the part to the AI assistant, for example:

```
Make a program for a pocket 40 × 30 mm, 3 mm deep, in 6061-T6, with a 6 mm flat end mill.
```

The assistant uses the `program_generate` tool. It says what it made, how long it takes, and what the twin found.

A job: features, tools and material

A job is what Studio keeps about the part:

- its name, which is also the program's name, for example `bracket.nc`;
- the material, for example 6061-T6 (other standards' names, such as EN AW-6061, are understood);
- the tool in the spindle and the tool table;
- the features, in the order they are cut.

Each program Studio writes is kept as a version, with the job that made it, in the programs folder. Nothing is lost when you make it again: the older versions stay.

Operations

Operation	What it cuts	Tool
Face	The top of the blank, flat	End mill
Pocket	An area to a depth: rectangle, circle, polygon or a shape	End mill
Contour	Along an outline: outside, inside or on it	End mill
Slot	A slot of a width along a line	End mill
Drill	Holes at points, pecked when deep	Drill
Bore	Holes larger than the tool, round a helix	End mill
Ream	Drilled holes to size	Reamer
Thread	Internal threads	Thread mill
Chamfer	An outline's edge	V-bit or chamfer mill
Engrave	Text or lines	V-bit or small end mill
V-carve	Text or shapes, the depth following the width	V-bit
3D roughing, 3D finishing	A mesh	End mill or ball end mill

Facing, pockets and contours

A pocket is cleared in layers. Its strategy sets the path in each layer:

- rings (the default): rings from the middle out to the wall;
- zigzag: passes to and fro;
- adaptive: a path that keeps the tool's engagement even, for deep pockets in harder materials.

Islands in a pocket are left standing. Rest machining clears only what a larger tool before it left.

A contour cuts along an outline, a layer at a time, climb milling. Tabs can hold a part that the contour cuts out.

Slots

A slot is cut full width in layers, then out to its width. A slot wider than the tool can be cut trochoidal: small loops along its line, for less load on the tool.

Holes: drilling, boring, reaming and threads

- Drilling uses the controller's drilling cycle. A hole deeper than three times the drill's diameter is pecked.

- Boring cuts a hole larger than the end mill round a helix. Up to twice the tool's diameter.
- Reaming brings a drilled hole to size with a reamer of the hole's fit. Drill the hole first.
- Thread milling cuts an internal thread with a thread mill, from the bottom up. Metric (M8), unified (1/4-20 UNC), pipe (G1/4) and tapered pipe threads (1/4 NPT, Rc1/4) are known by their names.

Chamfers, engraving and V-carving

A chamfer follows an outline's edge with a V-bit, to the width you give. Engraving follows lines or the strokes of text at one depth. V-carving cuts a shape with a V-bit: the depth follows the shape's width, so corners come out sharp.

3D roughing and finishing

A mesh (STL, 3MF or OBJ) is roughed in levels, then finished. The finishing strategies are parallel, waterline, steep and shallow, and pencil.

Tools and the tool table

A tool is written as its kind, its diameter and what else it needs, for example:

Spec	Tool
`flat 6 n3 L20`	Flat end mill, 6 mm, 3 flutes, 20 mm stickout
`ball 3 L12`	Ball end mill, 3 mm
`drill 3.4 118 L20`	Drill, 3.4 mm, 118° point
`v 6 90 L18`	V-bit, 6 mm, 90°
`reamer 6 H7 n6 L32`	Machine reamer, 6 mm, for H7 holes
`thread 5 60`	Thread mill, 5 mm, 60° profile
`flat 2 n2 s4 l6 L10`	Flat end mill, 2 mm, on a 4 mm shank that starts 6 mm above the tip
`thread 2 60 d1.4 n1 s4 l6 L9`	Thread mill, 2 mm head, 1.4 mm neck up to 6 mm, 4 mm shank above it

Give the stickout (`L`) for every tool. Studio then checks that the holder does not reach the blank, and the Z clearance on the setup sheet can judge the tool.

A small end mill on a stepped shank, and an undercut cutter (thread mill, T-slot, dovetail, lollipop), gives its shank's diameter (`s`) and how high above the tip the shank starts (`l`). Studio does not cut deeper than `l` where the shank would rub the walls of the cut. The twin checks the shank, and the touch probe's body, against the workpiece and the fixture.

The tool table lists the tools by their numbers. A feature names the tool it uses, and Studio adds the tool changes (`T2 M6`).

Cutting data from the twin

Studio starts from the material's cutting data for the tool. Then it uses the twin's models:

- the spindle's power: the cut uses at most about 80 % of it;
- chatter: the depth and speed stay where the twin's stability model expects a stable cut;
- the fixture: the cut's force stays well under what the fixture holds;
- the tool's deflection: finishing passes are kept light.

With verification on, Studio runs the program in the twin. If the run shows chatter or a load too high, Studio changes the program, for example with shallower layers, and tries again. Each version it tries is kept. The answer says what was changed and why.

These are estimates from physical models. Watch and listen to the first cut on the real machine.

Fits and tolerances

Write a size with its fit, for example `"d": "Ø12 H7"` for a hole or `"fit": "g6"` for a shaft. Studio cuts the size to the middle of the fit's zone, and finishes the wall in a pass of its own.

Common fits can be named: `sliding` (H7/g6), `locating` (H7/h6), `transition` (H7/k6), `press` (H7/p6).

A small, tight hole is better reamed than milled. Studio says so, and you can drill it, then ream it with a reamer of its class.

The twin estimates the cutter's runout, and Studio allows for it on the sizes that have a fit.

Several setups

A part machined from two sides needs two setups. Each setup has its own features, and says how the part comes to it: turned over about X or Y, held another way, its work zero found again. Studio writes one program for each setup, and checks each one in the twin on what the setups before it left. How to add setups and turn the part over: see Jobs with several setups.

Parts nested on a plate

Several parts can be cut from one plate. Studio places them close together, keeps a margin, and adds tabs so that the parts stay in the plate until the end.

Inch dimensions and fractions

With inch units, sizes can be written as fractions: `1/4`, `1 1/2`, `3/16"`. Inch tools are in the tool library, for example a 1/4" flat end mill. See Language and units.

Checking a program in the twin

Before you cut, run the program in the twin. The twin cuts its own workpiece and reports collisions, gouges, chatter, the spindle's load and the time. See Simulation.

> **Caution**
>
> The twin checks the program against the set-up you gave it. Set the twin up as the real machine is: the same blank, fixture, tools and work zero.

The setup sheet

A setup sheet tells the operator how to set the job up: the blank, the fixture, the tools, the work zero and the Z clearance. See Setup sheets.

Programs and the editor

The **Program** workspace (Alt+3) holds your programs and the example parts, an editor that checks a program as you type, and a preview of what the program does to the workpiece. Studio keeps a version of a program each time you save it or load it into the machine, so you can see what changed and go back.

Opening a program

The workspace has three parts:

- on the left, **Programs** (each program and its versions) and **Example parts**;
- in the middle, the editor;
- on the right, the 3D preview, **Load and check** and the program check's findings.

To open a program:

- **File > Open program...** (Ctrl+O) loads a G-code file into the machine, and into the editor if the editor has no unsaved changes. Studio reads your file and never writes it.
- Double-click a program or a version under **Programs**: its text goes into the editor. It does not go into the machine until you press **Load and check**.

The editor and the machine can hold different programs. When the machine has a program you have not opened in the editor, the status line says so: Ctrl+K loads the editor's program into the machine, Ctrl+Shift+R takes the machine's program into the editor (your edits are saved as a version first).

Example programs

Studio 1.0 comes with four example parts: a NEMA 17 motor mount plate, an instrument front panel, a microfluidic droplet chip and an isogrid lightweight panel. More come with updates. Double-click one under **Example parts**, or choose it from **File > Example parts...**.

Loading an example sets up the whole machine for it: the blank and the fixture, the tool table and the tool in the spindle, homing, the work zero, the program, and the part to compare with when the example has one.

> Note
>
> An example replaces the blank, the tool table and the G54 work zero you had. **Demo program** in the same list loads only the program and leaves the blank as it is.

The editor

The editor is a G-code editor:

- the words are coloured by kind: G and M codes, axes, arc words, feed, speed, tool, comments, expressions;
- the margin shows the line numbers and marks: the check's findings (red errors, amber warnings, blue notes), breakpoints, and the line the machine is running;
- after a run, a second margin shows each line's time in seconds: amber when the feed was not reached, red when something happened there (a rapid into material, a cut with the spindle stopped, a collision, an alarm). Click it for the line's figures from the last run.

The status lines under the editor show the line and column, whether the text is in the machine, the version, the check's counts, and the program's state after the line you are on: the modal codes, the feed, the speed, the tool and the position.

Expand gives the editor the whole workspace; **Restore** brings back the programs and the preview.

Colours and line numbers

The colours follow the window's look (light or dark). Lines that the controller has already taken ahead of the line being run are grey.

Finding and replacing

- Ctrl+F opens the find box. Enter finds the next match, Shift+Enter the one before. The search ignores case and goes round to the start.
- Ctrl+G goes to a line number.
- Ctrl+H opens the replace box under the find box. Enter replaces the match that is selected and goes to the next; Ctrl+Enter replaces all of them. Ctrl+Z undoes it.

Esc closes the boxes.

Breakpoints and pauses

A breakpoint stops the twin's time when the run gets to that line; the machine's motion does not change. F5 goes on.

- Ctrl+B, or a click in the marks margin, puts a breakpoint on the line.
- **Pause on** (Ctrl+Shift+B) stops the twin when something happens: the tool starts cutting, a rapid cuts material, a near miss or a collision, chatter, an alarm, a change of the controller's state, the cutting force, the spindle's load or the following error over a limit you choose, or a modal change. When the twin runs in real time, the chatter check is worked out beside the simulation, so a pause on chatter, and the chatter figures in the views, can come a little after the cut:

usually a few hundred milliseconds, and a second or two where many new cutting conditions come together (a pocket round islands). Until the check has answered for a tool, the margin reads **computing…**; the curves and the tool-tip trail wait for each cut's own answer, so they show that cut's margin. The run's report at its end is the same either way.
- The line's menu (right-click in the margin, or Shift+F10) also has **Run to line …** and **Clear all breakpoints and conditions**.

Breakpoints take effect once the program is in the machine.

The program check

The editor checks the program as you type (a moment after you stop), when it is loaded, and again after a run. There are two parts:

- the rules check every line against what the controller and the machine take. They need nothing else;
- the preview check runs the program against the workpiece as it is now. It needs the machine powered and homed and no program running. As you type it takes programs of at most 20 000 lines; a longer program is checked in full once it is loaded into the machine, and its problems show as red boxes in the 3D view and in the **Messages** tab.

Program > Check program is another check: it sends the loaded program through the controller's own check mode, line by line, while the machine does not move.

What the check finds

Each finding is an error, a warning or a note, shown under its line with its rule's number. Examples:

Rule	Kind	What it finds
S01	Warning	No start block before the first move: the program relies on the controller's state
S02	Warning	No `M2` or `M30` at the end
S04	Error	A cutting move before any `F`
S05	Warning	The spindle starts without `S`, or `S` outside the spindle's range
S07	Error	A code the controller does not take: expressions, `O` words, subprograms, `G41`/`G42`, `G68`
S10	Warning	A rapid below the work's top and deeper than the program has cut
S11	Error	An arc the controller refuses
S12	Warning	The water-cooled spindle cuts without its coolant (`M8`)
S14	Warning	A tool of another length than the one the work zero was set with, and nothing measures it

The preview check adds what the program would do to the workpiece: rapids into material, cutting with the spindle stopped, collisions with the fixture, moves beyond the travel, and a spindle running dry.

Fixing a finding

Some findings come with a suggestion, for example **Suggestion (insert before): M8**.

1. Put the cursor on the line, or press Alt+↓ and Alt+↑ to go from finding to finding.
2. Press Ctrl+. (or choose **Apply the suggestion** in the line's menu).

Result: the change is made as one step; Ctrl+Z undoes it. Read the suggestion before you apply it: it fixes the finding, not your intent.

Optimizing

Optimize (Ctrl+Shift+O) looks for a faster program that cuts the same part: it changes feeds and speeds, merges short lines and lowers retracts. It runs the original and the optimized program on two hidden copies of the machine, with the same setup. It needs a blank set up. **Optimize** is in the Pro edition and up.

If the result is better, it replaces the editor's text (Ctrl+Z gives the original back) and the comparison opens. The console gives the times, the highest cutting force and the chatter margin before and after, and what changed. Press **Stop optimizing** or Ctrl+Shift+O to stop it.

Versions

Studio saves a version of a program when you:

- press **Save version** (Ctrl+S);
- load it into the machine (**Load and check**, or **File > Open program…**);
- optimize it, or take a generated or the machine's program in place of your edits (your edits are saved first).

A version that is the same as the last one is not saved again. Versions are kept in `Documents\Twinwright Studio\programs\<program>\`, one file each, and are never pruned. **Programs** lists them under each program, with the time and why each was saved.

Comparing two versions

Compare (Ctrl+D) shows the newest saved version that differs from the editor's text, under the editor: changed lines amber, added green, removed red, and lines where only `F` or `S` changed blue. Press Ctrl+D again for an older version, Alt+↓ and Alt+↑ to go from difference to difference, Esc to close.

The heading counts the differences and, when both versions have run to the end in this session, compares their run times.

To compare two saved versions, open the newer one in the editor first, then press Ctrl+D.

Going back to a version

1. Under **Programs**, double-click the version. Result: its text goes into the editor. Your unsaved edits are saved as a version first; Ctrl+Z undoes the change.
2. Press **Load and check** (Ctrl+K) to load it into the machine.

Saving and exporting

- **Save version** (Ctrl+S) keeps a version in Studio's programs folder.
- Ctrl+Shift+S saves the editor's program as a file wherever you choose.
- A version is a plain `.nc` file: you can copy it from the programs folder.

File > Export curves (CSV)... saves the run's curves (see Analysis and diagnostics). The setup sheet and the inspection report are on the **Reports** card in **Setup** (see Inspection).

Running a program and resuming

The **Run** workspace runs the loaded program and shows what the machine does while it cuts. When a run stops, Studio tells you the next step, and can start the program again from a line.

Loading and starting

To load a program into the machine, do one of these:

- **File > Open program...** (Ctrl+O): choose a G-code file. Studio reads it; it never writes your file.
- **File > Example parts...**: choose an example. Studio also sets up the blank, the tools and the work zero for it.
- In the **Program** workspace, **Load and check** (Ctrl+K) loads the program in the editor.

The machine bar then shows the program's name, its number of lines and about how long it takes.

To start it, press **Cycle start** (F5) on the machine bar. **Cycle start** is filled in when the program is loaded, the machine is idle and homed. If the program check found problems, the console says so as the program starts.

Before you run a program on a real machine, check it: **Program > Check program** sends it through the controller's check mode. The controller reads every line and the machine does not move.

Watching a run

While a program runs:

- the 3D view shows the tool, the trail of its tip and the workpiece as it is cut. **Tool close-up** shows the tool at the cut;
- the dock's **Program** tab shows the listing: the line being run is marked, lines done are lighter, and a bar shows each line's time. **Follow the line** keeps the line in sight;
- **Program structure**, beside the listing, splits the program into its sections and layers (from the program's comments) with their times;
- the right column shows the readings.

When the program ends, the **Messages** tab says how many lines ran and how long the run took in the simulation.

Progress and the time left

The machine bar shows the line being run out of all the lines, the share of lines done, and **run** and **left**:

- **run** is the machine time so far, in the simulation's time;
- **left** is the estimate for the rest, from the program's feeds and rapids.

After an alarm the bar shows the line the program stopped at.

The readings

The right column of **Run**, from the top:

- **Position**: each axis in work and machine coordinates, with a mark when the axis is homed. Under it, the modal line: the G-codes in force, the tool, the feed, the spindle speed and the M-codes. In an alarm, the spindle and the coolant show as the machine has them, and the program's values are in the tip;
- **Spindle and feed**: the spindle speed (the arc is the speed, the mark the speed asked for), the spindle's load (green up to 80 %, red over 100 %) and the feed;
- **Cut** (Engineer level and up): the twin's estimates of the cutting force, the power, the chatter margin and the feed per tooth;
- **Overrides**, **Panel**, **Manual** and **Tools and zero** (Operator level and up).

Click a section's title to fold it. In shop mode, **Position** also shows **To go**: the distance left in the current line, for each axis.

Feed hold, stop and overrides

- **Feed hold** (F6 or Space) slows the axes to a stop; the spindle keeps turning. **Cycle start** goes on.
- **Stop** (F7) holds first and resets the controller once the machine stands. Press it again to reset at once.
- **Reset** (F12) stops the program at once.

The overrides change the program's feed and spindle speed while it runs, from 10 % to 200 %:

Override	Keys	On screen
Feed	Ctrl+↑, Ctrl+↓: ±10 %	**Overrides**: drag the knob; Ctrl for 1 % steps
Spindle	Ctrl+→, Ctrl+←: ±10 %	**Overrides**: drag the knob
Rapid	**Machine > Overrides**: 25 %, 50 %, 100 %	**Overrides**: 100, 50, 25

Ctrl+0, or a double-click on a knob, puts the feed and the spindle back to 100 %.

Single block

Single block on the machine bar stops the program after each line. Press **Cycle start**, or **Program > Next block**, to run the next line. You can turn it on before you start or while the program runs; click it again to run on without stops.

Jogging and the handwheel

Jog in **Run** under **Manual** (it starts folded), or with the keys (see The keys for the machine).

- **Step**: 0.01, 0.1, 1, 10 or 50 mm a press, or **Hold to jog**: the axis moves while you hold the key.
- **Speed**: 100, 300, 1500 or 3000 mm/min.
- The pad: X and Y in a cross, Z beside them, **Stop jogging** (Esc) in the middle.
- **Handwheel**: choose **X**, **Y** or **Z** and the increment (**×1**, **×10**, **×100**: 0.001, 0.01 or 0.1 mm a detent), then drag the dial or turn the mouse wheel over it.

You can jog only when the machine is powered, idle and not in an alarm, and no program runs. A held jog stops when you let go of the key or when the window loses the focus.

Tools > Settings... > Panel and devices > Handwheel sets how the handwheel behaves: **Velocity** stops the axis when the wheel stops; **Path** moves every detent.

Alarms and what to do

In an alarm the state on the machine bar turns red, and the bar shows:

- the alarm and what it means, for example **ALARM:17 A motor drive reports a fault**;
- **Next step:** what to do now, with the key that does it;
- **All steps**, when there is more than one.

Click the key, or use **Machine > Do the next step**, to do the next step. After each step, Studio works the steps out again from the machine's state.

The other signs of an alarm: a line under **Position** when the position can no longer be trusted, the red line in the listing where the program stopped, and a label at the drive in the 3D view.

The alarms, their causes and what clears them: Alarms.

The steps back to work

All steps opens the steps back to work: the cause, then the steps in order, each with its key and why it is needed. The step to do now is marked. One click does only the next step.

For example, after a drive alarm (ALARM:17):

1. **Press the E-stop, cut the drives' power** (F1): a drive's alarm clears only when it loses power.
2. **Release the E-stop** (F1).
3. **Power on** (F2): the drives have power again.
4. **Reset** (F12): the controller keeps ALARM:17 after the drives recover.
5. **Home** (F3): the power was off, so the position is not to be trusted.

When the fault was injected in **Diagnostics**, the cause says so, with the time.

When an alarm stopped a program, the steps end with the line to start again from, for example the start of the layer it stopped in.

Starting again from a line

Studio can start a program from any line. It writes a start block that sets the machine up as the program would have it at that line, then runs the rest.

1. Stop the program. The machine must be powered and no program running.
2. In the **Program** tab of the dock, right-click the line and choose **Start from line ...**, then **OK**. In the editor, right-click the line's margin or press Shift+F10. Result: Studio loads "*name* (from line *n*)". It does not start it.
3. Press **Cycle start**.

The start block:

1. lifts Z to the top;
2. sets the units, the plane, the distance mode and the work offset the program had at that line;
3. changes the tool, if the program had changed it before that line;
4. turns on the coolant and the spindle at the program's speed, and waits for the spindle to come up to speed;
5. moves over the start point at a safe height, goes down to 2 mm above it, then feeds down;
6. goes on with the program from that line.

The preview check then checks the rest against the workpiece as it is.

> **Caution**
>
> Studio cannot follow some lines before the start: parameters, expressions, subprograms, `G10`, `G28`, `G30` and `G92`. The load's notes list them. Check the machine's state before you press **Cycle start**.

Trying the start on a copy first

After an alarm stopped a program, the steps end with **Preview the start…**. It opens the start block and its notes.

1. Click **Try on a copy**. Result: the start runs on a hidden copy of the machine, from the workpiece as it is, and Studio reports its time, any collision, the highest chatter margin and how much it removed.
2. If it looks right, follow the steps back to work, then start from the line.

Running with the simulation's speed

The twin runs in real time. **Simulation > Speed** runs it from ×0.5 to ×50 real time. The clock on the machine bar shows the simulation's time, the speed you chose and the speed the computer reaches.

Simulation > Pause the simulation, or the hourglass on the machine bar, stops the twin's time: nothing changes while it stands. **Cycle start** goes on. Breakpoints and pause conditions in the editor stop the twin's time the same way (see Programs and the editor).

Pausing the simulation is not a feed hold: the machine's state is frozen, not stopped.

Analysis and diagnostics

Analysis looks back at a run: where the time went, the twin's estimates of force and chatter, and how the cut part compares with the design. **Diagnostics** shows the machine's power chain and signals, and lets you inject faults into the twin to practise finding them.

Looking at a run

Analysis (Alt+5, Engineer level and up) fills in once a program has run.

- **Runs**, on the left, lists this session's runs: the program, how it ended, its machine time and when it started. Runs that ended in an alarm or an error are red.
- In the middle are the curves, and under them two tabs: **Line profile** and **Stability lobes**.
- **Last run**, on the right, sums up the last run: how far it got, the lines that took most time, the highest cutting force and the highest chatter margin with their lines, the material removed, the lines where the feed was not reached, and the events.

The curves, the profile and the summary are those of the last run.

Line profile lists the lines that took time, longest first: the time and its share, the feed asked for and the feed reached, why it was not reached, the cutting force, the chatter margin, the material removed and the events. Double-click a line, or press Enter on it, to go to it in the **Program** editor.

Why a line did not reach its feed:

- the axes' top speed;
- corners too close together to speed up between them;
- the arc's chord limit;
- the controller's planning, the sender or the controller's reading falling behind;
- a feed override under 100 %, or a feed hold.

Time: estimated and actual

Before a run, Studio estimates the machine time from the program's feeds and rapids: the machine bar shows **about** that time, and the **Program structure** in the dock shows it for each section. During and after a run, the times are the twin's machine time, with the acceleration and the controller's planning in them: the machine bar shows the time run and left, **Program structure** the time of the whole run, and **Line profile** each line's time.

All times are machine time in the simulation, not the time on your clock.

Force, chatter and chip thickness

The curves show the run's readings over the last 10, 30 or 120 s: the feed, the actual speed, the spindle's speed and load, the cutting force, torque and power, the chip thickness, the chatter margin and the following errors. Click a curve's name to show or hide it.

The chatter margin is the depth of cut divided by the depth that is stable at the spindle speed in use:

Margin	Means
Under 0.8	Stable
0.8–1	Near the limit
Over 1	Chatters

Stability lobes draws the stable depth of cut against the spindle speed, for the last cut (or, before any cut, for a full slot 1 mm deep along X). Click **Work out the stability lobes**. A red dot marks the cut as it was, a green dashed line the most stable speed.

View > Colouring colours the tool-tip trail by feed rate, cutting force, chip thickness, chatter margin or program line, and the workpiece by material and depth, deviation from the part or program line.

These are the twin's estimates from a physical model (see Twin models).

Comparing with the design part

To see where the cut part differs from the design:

1. In **Setup**, step **Setup**, click **Part STL (compare)…** and choose the part's STL file (in mm, its origin at the program's zero).
2. Run the program.
3. Choose **View > Colouring > Workpiece: deviation from the part**.

Result: the workpiece is coloured by how far each point is from the part: within ±20 µm counts as good, the other colours show stock left and gouges. **Stop comparing** removes the part. The example parts load their own part.

Comparing two runs

Analysis shows the last run. To compare two runs of a program:

- in the **Program** editor, **Compare** (Ctrl+D) gives both versions' run times and the time of the lines that changed, when both have run to the end in this session (see Comparing two versions);
- the AI assistant can draw two runs side by side (see The AI assistant).

Exporting

Export curves (CSV)… (in **Last run**, or **File > Export curves (CSV)…**) saves about the last 160 s of the curves as a CSV file: a time column in seconds, then a column for each curve, with its unit in the heading and the values in the display units.

Export workpiece… in **Setup** saves the cut workpiece as an STL file (in program, blank or machine coordinates) or as a height map.

The Diagnostics workspace

Diagnostics (Alt+6, Technician level and up) shows the machine's power chain as a diagram: power flows from left to right. Solid lines carry power, green when live; dashed lines are signals. Each part is green when it is good, grey when it is off, amber for a warning and red for an alarm or a fault.

Click a part to see its details in the column on the right. Under the diagram are the controller's inputs, fault injection and the drive alarms.

The power chain

The MC-1's chain, from the mains to the spindle:

- **Main switch QF1** and the **24 V supply** for the controller;
- the **Controller**: its state or its alarm;
- **E-stop SB0** and **KM1**, the contactor it opens;
- the **48 V supply** for the drives;
- the **X**, **Y** and **Z** drives and their axes;
- the **VFD**, the **Spindle** and the **Coolant pump**.

The controller's inputs

A lamp for each input: the limit switches (**X− limit**, **X+ limit**, **Y− limit**, **Y+ limit**, **Z limit**), the **Tool setter**, the **E-stop** and the **Drive alarm**. A lamp is grey when the input is off, and amber or red when it is on.

The **Tool setter** lamp is the controller's probe input: it also lights when the touch probe touches.

The drives

Each drive shows **Ready** and its following error, or **Disabled**, **Alarm**, **Starting** or **Off**. The details give the drives' power and the following error of each axis, with its highest value.

The spindle drive (VFD)

The **VFD** shows its output frequency and current, or its fault: **OC** (overcurrent), **OL** (overload) or **LU** (undervoltage). The details give the spindle speed and the speed asked for, the load and the pump. A spindle that ran without its coolant is listed under the warnings, with how long it ran dry.

Injecting faults into the twin

Fault injection lets you practise finding and clearing faults on the twin; the machine in your window is changed, no real one.

- **Wires**: set any of the machine's wires (the limit switches, the tool setter, the drives' alarm chain, the E-stop line, the KM1 circuit, the RS485 bus, the VFD's fault relay) to **Open**, **Short**, **Intermittent** or **Glitches** (a single 50 µs drop-out). **Normal** puts the wire right.
- **Drive alarm**: **X**, **Y** or **Z** makes that drive report a fault. A drive's alarm clears only when the drive loses power: follow the steps back to work on the machine bar.

Each injected fault is listed under the warnings, written in the console and recorded in the audit log. After a drive alarm, the machine bar's cause names the fault you injected and the time.

Alarms: each one and its recovery

Every alarm, what it means and its causes: Alarms. The machine bar gives the steps back to work for the alarm you have (see The steps back to work):

Alarm	Steps back to work
A drive's fault (ALARM:17 with a drive in alarm)	Press the E-stop, release it, **Power on**, **Reset**, **Home**
E-stop pressed (ALARM:10)	Release the E-stop, **Power on**, **Reset**, **Home**
ALARM:17 with the drives back	**Reset**, **Home**
Homing needed (ALARM:11)	**Home**
Hard limit (ALARM:1)	**Reset**, **Unlock**, jog off the limit switch, **Home**
ALARM:3, 8, 9	**Reset**, **Home**
Any other alarm	**Reset**, **Unlock**

Running a program in the twin

Draft (round 14): the text is being written; the sections may still grow.

What the twin is

The machine in the window is a digital twin of the MC-1: the Twinwright controller (kernel) runs in it, the same code as on the machine, and a physical model stands for the motors, the drives, the spindle, the switches and the cut. A program runs in the twin as it would on the machine, and the twin cuts its own workpiece. Every number it gives about the physical world is an estimate.

Before you run

The twin checks a program against the setup you give it, so set it up as the real machine is:

- the machine (**File > Switch machine...**);
- the blank and how it is held (the **Setup** workspace);
- the tools (the dock's **Tool table**);
- the work zero (**Machine > Work zero**).

File > Example parts... sets all of these and loads a program in one go.

> **Caution**
>
> A check is only as good as the setup: a different blank, fixture, tool or work zero on the real machine is not what the twin checked.

Run a program

1. Open the program: **File > Open program...**
2. Check it first: **Program > Check program** runs it through the controller's check mode, so the controller reads every line and the machine does not move.
3. Run it: **Cycle start** (F5). **Feed hold** (F6) pauses it, **Cycle start** goes on, **Stop** (F7) stops it. **Single block** runs it line by line, **Next block** runs the next line.

The twin runs in real time unless you choose otherwise: **Simulation > Speed** runs it from half to 50 times real time, and **Simulation > Pause the simulation** stops the twin's time (the machine's state does not change while it stands).

What the twin checks

What follows from the program and the setup's geometry:

- the controller's own alarms and errors, as the machine's controller gives them: soft limits, limit switches, a probe move that touches nothing;
- collisions of the tool holder, the collet nut and the spindle with the blank, the fixture and the table, and near misses;
- cutting in a rapid move (`G0`), and cutting while the spindle stands;
- cutting through the blank's bottom into the fixture or the table;
- a tool that would jam or break: a feed per tooth too high for the tool, a bending stress at the flutes' root over what the tool takes;
- the Z clearance: whether each tool, and the touch probe, clears the setup with Z at the top (see MC-1 hardware);
- with the part's design in the setup: material left on the part and cuts too deep (**View > Colouring > Workpiece: deviation from the part**).

The findings appear in the dock's **Messages** tab as they happen.

What the twin estimates

What the physical model works out:

- the cutting forces, torque and power, and the spindle's load: the spindle slows under load as a real one does, and too much load trips its drive;
- how the tool and the machine give way under the cut, in the shape cut;
- chatter: the margin along the path (**View > Colouring > Trail: chatter margin**), and the spindle speeds where the cut is stable (the **Analysis** workspace: **Work out the stability lobes**);
- the cutter's runout: slots and holes come out wider by it;
- the machine time, from the controller's own planning and the drives' motion;
- the cusps a ball-nose cutter leaves between its passes (their shape, not the surface's roughness).

The trail can be coloured by the feed rate, the cutting force or the chip thickness as well (**View > Colouring**), and the dock's **Curves** tab plots the run over time.

> **Note**
>
> The estimates use typical parameters - published cutting data, sharp carbide tools, the machine's design values - until your machine is measured. Use them to compare programs and to find problems before you cut, not as a measurement of the part.

What the twin does not model

- the surface's texture and roughness;
- built-up edge, chips and getting them out;
- thin walls bending under the cut;
- the part's own heat;
- tool wear: in 1.0 every tool is a new one (tools that cut aluminium are mostly changed first because aluminium sticks to the edge and the part shows burrs or a poorer finish);
- what happens after a tool breaks;
- details finer than the program check's grid: before anything runs, the program check looks at the workpiece in cells of about 0.1 mm (coarser on blanks over 100 × 100 mm: a million cells at most), so a rapid through a wall thinner than that, or a tool that grazes a wall by less, may pass the check unnoticed. Run the program in the twin to see it: the twin's own cut works in 0.02 mm cells.

Plastics and micro tools

Cutting PMMA (acrylic) with tools of 0.1–0.5 mm - microfluidic channels, for example - the twin also leaves out:

- the cut's temperature: PMMA softens towards its glass transition (about 105–115 °C), and from about 70 °C its chips weld and smear on the tool and in the channel. Keep the chip load and the speed in the range the tool's maker gives;
- the air jet or the coolant, and what it does to the heat and the chips;
- burrs along the edges of channels and pockets;
- thin walls: in PMMA, walls under about 0.15 mm thick bend or break under the cut; the twin cuts the shape the tool's path and its give leave, not the wall's own deflection;
- the damping of the cut itself: a tool this small bends at tens of kilohertz and cuts slowly, and the tool's flank rubbing the cut surface damps its vibration. The twin leaves this out, so its chatter margin for such tools is on the safe side.

The models and what they assume: Twin models.

The results and the report

- The workpiece in the 3D view is what the twin cut; its colouring shows the cut's depth or, with the part's design, the deviation from it.
- The dock's **Messages** tab lists the checks' findings; its **Curves** tab the forces, the load and the speeds over time. **File > Export curves (CSV)…** saves them.
- An AI client reads the same results through Studio's MCP tools: see the MCP tools reference.

Inspection, reports and setup sheets

After a part is cut, the touch probe measures it against the design. Studio writes what it measured, with the twin's estimate beside it, into an inspection report. Before a part is cut, a setup sheet tells the operator how to set the job up. Both are in the Pro edition.

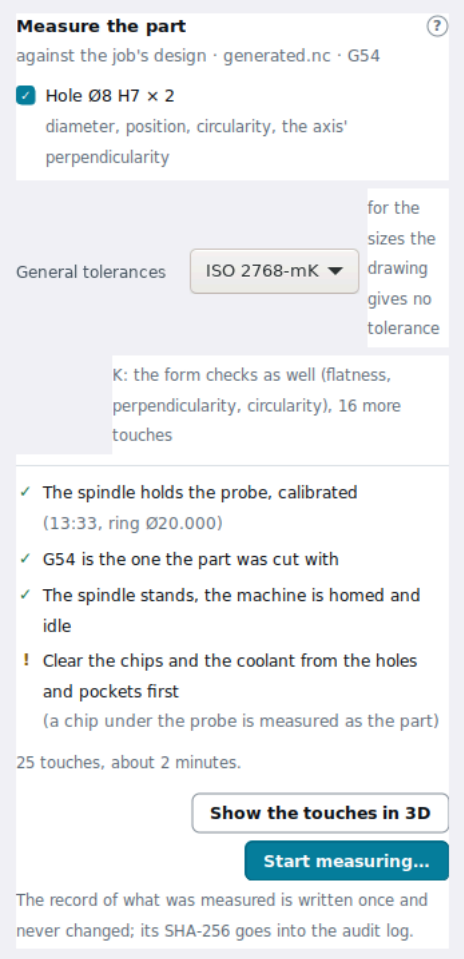
Measuring a part with the touch probe

Measure a part where it was cut, before you take it off the machine. The **Measure the part** card is in **Setup**, step **Zero**, under the probe card. It compares the part with the design of its CAM job: each feature's size, fit and tolerance. In 1.0 the card measures on the twin, as the probe card does. When the twin follows a real machine, measure there with the machine's own control.

You need:

- The part, cut and still held as it was cut. Do not move the work zero: the positions are measured from it.
- The touch probe in the spindle, calibrated on its ring gauge. See Probing.
- The program's CAM job: a program made in **CAM**, or by the AI assistant.

> **Note**
>
> Measuring is in the Pro edition. In the other editions the card shows with a **Pro** label, and **Start measuring…** says how to try it. Records and reports made before open in every edition.



To measure:

1. In **Setup**, click the step **Zero**. Result: the card says what it compares the part with: the program and the work offset it was cut with, for example G54.
2. Choose the features to measure. All are chosen at first. Result: each feature says what the probe measures on it. A feature the probe cannot measure is grey and says why.
3. Under **General tolerances**, choose the class for the sizes that have no tolerance of their own. Add H, K or L for the form checks (see Tolerances). Result: the card says how many touches it makes and about how long they take.
4. Read the checks: the probe in the spindle and calibrated, the work zero the one the part was cut with, the spindle standing, the machine homed and idle.
5. Clear the chips and the coolant from the holes and pockets. A chip under the probe is measured as the part.
6. To see where the probe will touch, click **Show the touches in 3D**. Result: the 3D view shows the touches from the side and the touches downward.
7. Click **Start measuring…**. Result: the probe touches each feature, and the card says what it measures now.

> ⚠ **Warning**
>
> The probe moves at feed rates near the part. Keep hands and tools away from the machine while it measures.

What is measured:

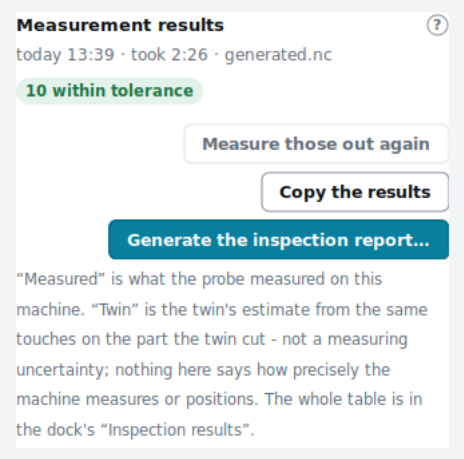
Feature	Checks
Pocket, contour	Its walls' size and position, its floor's depth
Hole, boss	Diameter and position
Slot	Width and depth
Face	Depth
Chamfer	Width, depth and angle
V groove (engraving or V-carving with a V-bit)	Width, depth, angle and offset from its line
Engraving with an end mill, thread, 3D surface	Not measured

The form checks add flatness, perpendicularity, circularity and straightness where they apply (see Form).

When the probe is done, the **Measurement results** card takes the measuring card's place. It shows:

- when the part was measured, how long it took, and the program;
- how many checks are within tolerance, how many out of tolerance, and how many are not judged: **not judged in the twin** (its cells there coarser than a quarter of the tolerance), **no tolerance** or **not measured**;
- each check out of tolerance, with its measured value, its limit and how far it is over or under;
- for the first feature out of tolerance, the most likely cause and what to do next.

In 1.0 the probe measures the part the twin cut, so the cause comes from the twin alone. If the twin's own part is out of tolerance at the same touches, the program cuts it so: change it in **CAM**. Otherwise the measured value and the twin's estimate differ only by what the twin models of the probe (the ball's effective radius and, for a typical probe, its pretravel, which a calibration on the ring gauge takes out) and by the size of the twin's cells where the probe touches. Calibrate the probe on the ring gauge and measure again. The causes are suggestions, not a diagnosis.



The card's keys:

- **Measure those out again**: measures the checks out of tolerance once more, for example after you calibrated the probe again.
- **Copy the results**: copies the table as text, one column per tab, for a spreadsheet.
- **Generate the inspection report...**: writes the report (see The inspection report).

The whole table is in the dock's **Inspection results** tab. Each row is one check, with its **Design** value, its **Tolerance**, what was **Measured**, the **Deviation**, the **Twin** estimate and the verdict: ✓, or ✗ with how far it is out. Rows out of tolerance are red, and the 3D view marks the results on the part.



Check	Desi...	Tolerance	Measu...	Deviation	Twin	
Hole 1 diameter	Ø8	H7 +0.015/0	8.001	+0.001	8.004	✓
Hole 1 position X	8.000	±0.2 (m)	8.000	+0.000	8.001	✓
Y	32.000	±0.3 (m)	32.000	+0.000	32.000	
Hole 1 circularity	○	0.015 (K)	0.002		0.003	✓
Hole 1 axis' perpendicularity	⊥	0.4 (K)	0.000		0.001	✓
Hole 2 diameter	Ø8	H7 +0.015/0	8.001	+0.001	8.004	✓
Hole 2 position X	52.000	±0.3 (m)	52.001	+0.001	52.001	✓
Y	32.000	±0.3 (m)	32.001	+0.001	32.000	
Hole 2 circularity	○	0.015 (K)	0.002		0.003	✓
Hole 2 axis' perpendicularity	⊥	0.4 (K)	0.000		0.001	✓

Each inspection is kept as a record. The record is written once and never changed, and its SHA-256 goes into the audit log (see Checking a copy against the audit log).

The AI assistant measures the same way, with the `inspect_part` tool. Ask, for example: "Inspect the part of bracket.nc with the probe".

1. The assistant says what it will measure, and how many touches and seconds the form checks add.
2. Confirm before the probe moves. Result: the answer lists each check with its measured value, the twin's estimate and the verdict.

Tolerances

Each check is judged against a tolerance. A size's own tolerance or fit comes from the CAM job, as it was typed in the size's field (for example 8 H7). The other sizes take the general tolerance.

General tolerances (ISO 2768-1)

ISO 2768-1 has four classes: f (fine), m (medium), c (coarse) and v (very coarse). A size without its own tolerance takes the class's tolerance for its size range. Choose the class on the card, under **General tolerances**. Or ask the assistant for it, for example: "Inspect it to ISO 2768-m".

Form: flatness, perpendicularity, circularity, straightness (ISO 2768-2)

Add H, K or L after the class, for example "mK". The probe then also checks the form:

Check	How it is measured
Flatness	A floor touched on a 3 × 3 grid
Perpendicularity	A wall, or a hole's axis, touched again at a second depth
Circularity	A hole or boss touched at 8 points
Straightness	A long wall touched at 3 to 8 points

The card, or the assistant, says how many more touches the form checks take before the probe moves.

> **Note**
>
> A touch probe that is not calibrated in each direction has its own lobing. The report says how large the twin's probe model takes it to be, next to each circularity.

Fits (ISO 286)

A size written with its fit, for example Ø8 H7, is judged against the fit's zone. See Fits and tolerances.

Chamfers and V grooves

Chamfer widths, depths and angles, and V-groove widths, depths and angles, take ISO 2768-1 class f unless you give their own tolerance.

What the twin estimates

The twin runs the same touches on its simulated workpiece. Its estimate is what the program cut in the twin. When the measured value and the estimate differ, look at the machine and the set-up: the tool's length, the work zero, the fixture, a worn tool.

The twin's estimate is a calculation from its physical models. It is not the machine's measuring uncertainty.

The inspection report

Every inspection is kept as a record. From a record, Studio writes the report:

- a CSV file (UTF-8), for a spreadsheet;
- a PDF file, for printing and filing.

The report lists the job, the setup, the tools, the probe and its calibration, and for each feature: the nominal size, the tolerance and its basis, the twin's estimate, the measured value, the deviation and the verdict. It says why a feature was not measured, and shows the twin's workpiece from above.

To write it, ask the assistant, for example: "Write the report of the last inspection". The same record gives the same files, byte for byte.

Where the reports are

Inspection reports and setup sheets are in the reports folder:

- Windows: Documents\Twinwright Studio\Reports
- Linux: ~/Documents/Twinwright Studio/Reports

If you moved the programs folder in the settings, the reports folder is inside it.

Checking a copy against the audit log

Studio records the SHA-256 of every record, report and setup sheet in its audit log. Each entry is chained to the one before it. You can check a copy that someone sends you:

- Ask the assistant: "Check this file against the audit log", with the file's path.
- Or run, on the computer where it was written:

```
mc1sim --inspect-verify <file>
```

The answer says whether the log holds the file's SHA-256, when it was written, and whether the log's chain is whole. A copy changed by a single bit is not found.

Setup sheets

A setup sheet is one page for the operator at the machine. It says what to set up and in which order. Studio writes one for each job, in the language you use. The paper is A4, or Letter in the United States and Canada.

To write one, ask the AI assistant, for example: "Write the setup sheet for bracket.nc". The assistant uses the `setup_sheet` tool.

> **Note**
>
> Setup sheets are in the Pro edition.

What a setup sheet shows

- Programs: Each program's name, version and number of lines, and the first 8 hexadecimal digits of its SHA-256. Compare them with the program loaded on the machine.
- Blank and fixture: The blank's size and material, and how the twin sets it up. For a vise: the parallels, the grip and how far the blank stands over the jaws.
- Tools: Each tool's number, kind, diameter, flutes, stickout and spec, and the operations that use it.
- Each setup: How the part gets there (turned over, held another way), where the work zero is, and the probe cycles that find it. The cycles have the names they have on the probe panel.
- Travel: The lowest and highest X, Y and Z the program moves to, in work coordinates.
- Z clearance: See Z clearance.
- Operations: Each operation's strategy and how it goes into the material, its tool, speed, feed and depth, and the time it takes.
- Top view: The blank, the vise jaws, the work zero and the outline of each feature.
- Notes: The CAM's notes, in English.

The times are estimated from the program: its feeds, and the machine's rapid rate and acceleration. Tool changes are not counted. A trial run in the twin gives times with acceleration and deceleration counted.

Z clearance

For each setup, the sheet checks that the touch probe and each tool get over the set-up. It also checks that each tool reaches the program's highest and lowest points. The heights are the twin's Z budget's, measured from the table top.

Item	Has to reach
Touch probe	1 mm over the set-up's highest point, with Z near the top of its travel
Each tool, up	0.5 mm over the set-up's highest point, and the program's highest point
Each tool, down	The lowest point of its own operations

Each row says **Enough**, or how many millimetres short it is, in red.

> **Caution**
>
> A row in red means that the program cannot run on this set-up. Use a tool with less stickout or a shorter probe, or hold the part lower, for example with clamps.

A clamp can stand higher than the program's highest point. The sheet then says so: the program must not cross the clamp. A trial run in the twin checks that.

Printing and signing

1. Open the setup sheet in a web browser.
2. Print it. Result: each setup after the first starts on a new page.
3. At the machine, fill in the box at the end: the operator, the date and the first-article inspection (pass or fail, and the number of its inspection report).

A new setup sheet of the same job replaces the file. Its SHA-256 is recorded in the audit log.

Connecting a real machine

Draft (round 14): the text is being written; the sections may still grow.

Two ways to work with a machine

- Drive the MC-1 from a sender of your own (bCNC, gSender or Universal G-code Sender) on its USB port. See Sender software.
- Put the twin beside it in shadow mode (preview): your sender connects to Studio instead of the machine, Studio passes every line on to the machine and runs the same lines in the twin, and compares the two.

Sending a program to a real machine from Studio itself is not in 1.0.

Shadow mode (preview)

> **Note**
>
> In 1.0 shadow mode is a preview that you start from the command line. It has been tested with a second twin standing in for the machine, not yet with a real one. The window does not yet have a place to choose the port, connect or see the comparison; that comes in a later version.

Start shadow mode

You need: the MC-1 switched on and connected to this computer by USB; your sender installed.

1. Find the machine's port. On Windows: **Device Manager > Ports (COM & LPT)**, for example COM5.
2. Start Studio with that port from a command prompt:

   ```
   TwinwrightStudio.exe --shadow COM5
   ```

   Result: Studio opens the port and the twin follows the machine.

3. In your sender, connect to Studio's sender port, `localhost:2323`, instead of the machine. Result: your sender's lines go to the machine through Studio, and the twin runs the same lines.

Studio's sender port listens on this computer only. To reach it from another computer, add `--bind any`.

What the twin compares

- The answers, line by line: each `ok` or error, and the alarms in their order.
- The path: the position the machine reports against the twin's path, and how far the twin is ahead or behind.
- The settings: once, when the machine is connected and before your sender connects. Connect your sender after Studio has linked to the machine.
- The controller: once, with the settings. Whether both run the same firmware and, with the Twinwright controller on both, its version and what the machine's board has: spindle, encoder, handwheel, axes, probe, connection. The twin does not model the board's auxiliary ports and SD card: the comparison lists them, and they do not count as a difference.
- The run times of the program's parts.

The guard

Start Studio with `--shadow-guard` as well, and the twin checks each line from your sender before it reaches the machine. When the twin predicts a crash, it holds the line back and puts the machine on feed hold. The window shows the guard's state under **Diagnostics**.

What cannot reach the machine

In shadow mode only your sender's lines reach the machine, with the twin's own questions that change nothing (the settings and the status).

- The window's controls act on the twin only, and the ones that would move or start the machine are refused while the twin follows a machine. The probe panel is locked.
- An AI client can only read: it sees the machine's state and changes nothing.
- An external handwheel connected to Studio acts on the twin only.

> **⚠ Warning**
>
> In shadow mode the window's stop and E-stop do not stop the machine. To stop the machine, use its own E-stop or your sender's stop.

Sender software

The senders tested with the Twinwright controller, and how to set them up: Sender software.

When something differs

A difference means that the twin's profile or settings are not this machine's, or that the machine does something the model does not. Check the settings first: the comparison lists the ones that differ.

The AI assistant

An AI can work on the twin with the same tools you have: it sets up, writes and checks programs and runs them in the simulation. You can talk to one in Studio's own assistant, or connect an AI client such as Claude Desktop. Anything that would change the window's machine waits for your yes. The AI is not part of any safety function.

Turning the AI on

The AI is off in a new installation. To turn it on, do one of these:

- click the AI's state in the top bar (**AI off**) and choose **The window's AI switch**;
- **Tools > Settings... > AI and MCP**: **AI switch**, then **OK**;
- send a message in the assistant: this turns the switch on.

Studio keeps the switch as you leave it for the next start. The **AI** menu (**AI assistant...**, **AI activity...**, **Connect AI clients...**) shows from the Operator level on.

With the switch off, AI clients can still use practice machines in the background, but not the machine in your window.

The AI service and its key

The assistant talks to an AI service you choose, with your own account:

- **Anthropic (Claude)**, **OpenAI**, **DeepSeek**, **Qwen (Tongyi)**, **Kimi (Moonshot AI)**, **Zhipu (GLM)**;
- **LM Studio (local)** or **Ollama (local)**: a model on your own computer, no key needed;
- **Custom (OpenAI-compatible)**: any service with an OpenAI-compatible interface, at the address you type.

To set it up:

1. Open **AI > AI assistant...**
2. Choose the **Provider**. Studio fills in its **Base URL**.
3. Paste your key in **API key** and click **Save key**. Result: the key is stored on this computer, encrypted. On Windows only your Windows account can read it. On Linux it goes into the desktop's keyring (Secret Service: GNOME Keyring, KWallet); without one it is encrypted with a key in a file only your user account can read, and the line under the key says so.
4. Click **List models** and choose a **Model**.

Studio does not write the key into its settings, the registry or the audit log. To delete a saved key, empty the field and click **Save key**.

> **Note**
>
> What you send the AI service goes to that service under your account and its terms: your messages, and with **Attach the window's program, setup and check results** on, the program, the setup and the check results, and the results of the tools the AI uses. To keep everything on your computer, use a local model.

Before your first message to a cloud service, the assistant says once what goes there. Click **Send** to go on, or **Use a local model...** to switch to LM Studio; your message stays in the box. **Tools > Settings... > AI and MCP** lists the services you agreed to; **Ask again** makes the assistant ask again.

The assistant panel

AI > AI assistant... opens the assistant in a window of its own.

- Type in the box at the bottom. Enter sends; Shift+Enter starts a new line. **Stop** stops the answer.
- **Attach the window's program, setup and check results (when they changed)** gives the AI what is in your window with each message. It is on by default.
- The conversation shows each tool the AI uses, with ✓ when it worked and ✗ when it did not.
- Pictures the AI makes (the workpiece, the toolpath, the deviation from the part, a comparison) appear in the conversation. Double-click one to enlarge it or save it.
- **New conversation** starts afresh. Conversations are not saved when you close Studio.

Closing the window only hides it; the conversation stays until you quit.

What the AI can and cannot do

With the switch on, the AI can:

- read the machines' state, and home, unlock, jog and send lines of G-code;
- set up the blank, the tools, the work zero and the part, probe and inspect;
- check, simulate, optimize, generate, edit and load programs, and start or restart a run;
- read runs and their results, compare them and make pictures of them;
- read the audit log and the licence's state.

Tools that belong to a paid edition (inspection, reports, nesting, the optimizer) work only in that edition.

The AI cannot change Studio's settings, the language, the units, the user level, the AI switch, how Studio asks you, the licence or the audit log.

Asking before it changes the machine

Before the AI changes the window's machine (in the AI's menu in the top bar and in **Tools > Settings... > AI and MCP**) has two choices:

- **Ask me every time** (the default);
- **Do not ask (the AI works the window directly)**.

When the AI wants to change the window's machine (move it, power it on, release the E-stop, load or run a program, change the setup or the tools), a bar opens over the machine bar. It says which AI asks and what it wants to do, and counts down.

1. Read the question.
2. Click **Allow** or **Refuse**. Enter does not answer: use the mouse, or Tab to a key and press Space.

No answer within 60 s counts as **Refuse**. A yes also covers that action's own follow-up requests for two minutes.

Studio never asks before the AI stops, holds or resets the machine, presses the E-stop or only reads. Practice machines and the hidden copies the AI simulates on are not asked about.

Read-only on a real machine

When the window's twin follows a real machine (shadow mode, a preview in 1.0; see A real machine), the AI only reads and analyses. It cannot move the machine, change its setup or work offsets, or run programs there, and it does not ask: Studio refuses those requests. Practice machines work as before.

AI activity

AI > AI activity... shows what AI clients do on this computer:

- **Background machines**: the practice machines the AI uses, which you do not see in the window ("window" is the machine in your window);
- **Runs**: the last 16 simulations and optimizations;
- **Connected AI clients**: who is connected, how, and their last call.

From here you can stop a run, look at a run's picture, close a background machine, and stop or start the twin host. The twin host is the background program that holds the practice machines. It starts when an AI client connects and quits after 30 minutes unused.

The dock's **AI activity** tab lists the AI's calls on the window's machine: time, call, machine, who and result.

Connecting other AI clients (MCP)

AI clients that speak MCP (Model Context Protocol) can use Studio's tools.

1. Open **AI > Connect AI clients...**
2. Click your client: **Claude Desktop**, **ChatGPT desktop (Codex and Work modes)**, **Cursor**, **VS Code (GitHub Copilot)** or **LM Studio (local open models)**. Result: for Claude Desktop and ChatGPT desktop, Studio shows the change it will make to the client's settings and makes it when you click **OK**; it keeps the client's old file next to it. The others open their own install page.
3. Quit the client completely and open it again.
4. Turn on **The window's AI switch** in Studio.

For any other client, **Other clients: copy the configuration** copies the settings to paste into it. The client starts `TwinwrightStudio.exe --mcp` in the background; **The command and configuration** shows the exact command.

Clients that take only an address

For clients that take only an address, turn on **Local URL (HTTP, for clients that only take a URL)** in the AI's menu and use **Copy the local URL (HTTP)**. The address works only on this computer and needs the token that comes with it. **New token** makes the old one stop working.

The tools, their arguments and their results: MCP tools.

What is recorded

Tools > Audit log... (Ctrl+L) records what the AI does, with the AI's name in the **Who** column. Filter **Who** by **AI** to see only the AI's records; double-click a call to see all its records together.

At the default **Standard** level, the log keeps:

- turning the AI switch on and off, and changing how Studio asks;
- each question, your answer and each AI action on the window's machine;
- each tool call, with a short form of its arguments;
- clients connecting and leaving;
- the assistant's messages: the first 180 bytes in the summary, up to 2000 bytes in the record.

Keys, tokens and passwords are written as `***`.

Language and units

Studio follows your system's language at the first start. You can change the language, the units and the way numbers, dates and money look at any time. Programs and files stay as they are.

Changing the language

Choose **Tools > Language / 语言** and a language: English, 简体中文 (Chinese, Simplified), Français, Deutsch, 日本語 or Español. The menu's name is the same in every language, so you can find it in a language you do not read.

The window changes at once; you do not restart Studio. Lines already in the console stay in the language they were written in.

Tools > Settings... > General > Language has one more choice, **As Windows**: Studio then follows the system's language at each start. A language Studio has no pack for falls back to English.

Units: millimetres and inches

Tools > Units chooses **As the region measures**, **Metric (mm)** or **Inch**. The change applies at once.

In inches, Studio shows:

Quantity	Metric	Inch
Length	mm	in
Feed	mm/min	in/min
Cutting speed	m/min	ft/min
Force	N	lbf
Torque	N·m	lbf·in
Power	W	hp
Temperature	°C	°F

The spindle speed stays in rpm. Jog steps become 0.0005, 0.005, 0.05, 0.5 and 2 in, and jog speeds 4, 12, 60 and 120 in/min.

The units change what Studio shows and reads, not your files. A program runs as it is written, in `G20` or `G21`. Programs Studio makes are written in millimetres (`G21`). The AI's tools and the logs work in millimetres.

Region: numbers, dates and money

Tools > Settings... > General > Region sets how numbers, money and dates look: **As Windows (Settings > Time & language)**, **As the language**, or a region from the list. On the same page:

- **Currency**: an ISO 4217 code such as USD, EUR or CNY; empty means the region's;
- **Currency symbol**: empty means the currency's own;
- **Machine-hour rate**: your cost of an hour of machine time, for the cost estimates; 0 shows none.

The region decides the decimal mark and the thousands separator in Studio's numbers, the dates in the runs, versions and reports, and the paper size of reports (Letter in the United States and Canada, A4 elsewhere). Numbers that belong to the machine, such as positions and the numbers in a program, always use a point and no thousands separator.

Typing numbers

Studio reads a typed number strictly in the fields where a mistake would cost a part: the CAM fields, the sizes and places in **Setup**, the probe's lengths and the spindle speed.

- Use one decimal mark: a point or a comma (`13.3` or `13,3`).
- Do not write a thousands separator.
- You may write the field's unit after the number: `13 mm`, `0.5 in`, `30°`.
- Digits typed with an input method (full-width digits) are read too.

`1,000` is a question when your region writes the decimal mark as a point: did you mean one or one thousand? Studio does not guess; it asks you to write the one you mean. Where the region writes a comma, `1,000` is one.

What "reads as" means

When what you typed is not written the way Studio writes numbers, the field shows what Studio read: type `13,3` and it shows **reads as 13.3**. In inches it shows four decimals: **reads as 0.3750**. Check it before you go on.

When Studio cannot read what you typed, the field turns red and a line under it says why, for example ""1,2,3" is not a length". Studio does not take the value: the field keeps the last one it took until you correct it.

Inch fractions

In inches, length fields also take fractions:

- `3/8`, `5/4`
- `1 1/2` or `1-1/2`
- `2 3/8"`, `1/2 in` (the inch mark may be `"`, `”` or `in`)

The field shows the decimal and the fraction it read: **reads as 0.3750 (3/8")**. Studio does not take a decimal point with a fraction (`1.5/2`), an improper fraction after a whole number (`1 5/4`) or spaces around the slash.

Studio shows inch sizes as fractions too, down to 1/64 in: tool diameters and lengths, the blank's sizes, hole diameters and depths read `1/4"` or `1 1/2"`. A size that is not a 1/64 shows four decimals, `0.2010 in`. Coordinates, feeds, speeds and measurements are always decimals.

In millimetres, a fraction is not read: "Millimetre fields take no fractions: write 4.7625 or switch to inch".

Tool descriptions read inch sizes in either system, for example `flat 1/4in n2`.

Spindle speed

You can type the spindle speed instead of stepping it by 1000 rpm:

1. In **Run**, open **Manual** in the right column.
2. Double-click the **Spindle speed** reading, or press Enter on it.
3. Type the speed and press Enter. Esc, or clicking elsewhere, keeps the old speed.

When the spindle is turning by hand (no program running), the new speed goes to the spindle at once. When it stands, Studio keeps the speed for the next **Spindle CW** or **Spindle CCW**. While a program runs, the program sets the speed and the field does not open.

A speed outside the spindle's range turns the field red and is not taken. The MC-1's range is 1000–24000 rpm.

What stays in English

- G-code, and the files Studio writes and reads (jobs, setups, recordings, licence files);
- the AI's tools and what goes to and from AI clients;
- the reference chapters made from the program itself: MCP tools, Command line and Machine profile schema;
- each audit record's message for programs (the log also keeps it in your language);
- in German, French, Japanese and Spanish in 1.0: the controller's error and alarm descriptions, the controller settings' help and some licence messages.

An inspection report's PDF is in your language when its fonts have your language's letters, and in English otherwise; its CSV file is always in your language.

Keyboard and accessibility

The machine, the menus, the panels' keys and the editor all work from the keyboard. Turning and zooming the 3D view, picking a point in it and resizing panels need a mouse. On Windows, a screen reader reads the panels' keys and readings.

Help > Keyboard shortcuts (or **?**) lists every key in Studio.

The keys for the machine

Key	Does
F1	**E-stop**: press; press again to release
F2	**Power on**
F3	**Home**
F4	**Unlock**
F5	**Cycle start**
F6 or Space	**Feed hold**
F7	**Stop**: feed hold, then a reset once the machine stands; press again to reset at once
F8	**Coolant** on or off
F9	**Spindle CW**
F10	**Spindle CCW**
F11	**Spindle stop**
F12 or Ctrl+X	**Reset**

Jogging and the handwheel:

Key	Does
← →	Jog X
↓ ↑	Jog Y
Page Down, Page Up	Jog Z
Shift with any of these	Jog while you hold the key
1–5	Step 0.01, 0.1, 1, 10 or 50 mm
6	**Hold to jog**: every jog key jogs while you hold it
Ctrl+1 to Ctrl+4	Jog speed 100, 300, 1500 or 3000 mm/min
Esc	**Stop jogging**
X, Y, Z	Choose the handwheel's axis
, and .	Turn the handwheel one detent down or up (with Shift: ten)

Overrides:

Key	Does
Ctrl+↑, Ctrl+↓	Feed +10 %, −10 %
Ctrl+→, Ctrl+←	Spindle +10 %, −10 %
Ctrl+0	Feed and spindle back to 100 %

Each press of a jog key moves one step. A held jog stops when you let go of the key or when the window loses the focus. In inches the steps are 0.0005–2 in and the speeds 4–120 in/min.

The F keys work wherever you are. The other machine keys do not work while you type in a text box, a list or the editor: click the 3D view to hand them back to the machine.

The digit keys and the , and . keys work by their place on the keyboard, so they work with any keyboard layout.

Emergency stop

Press F1, or click the round red key at the right end of the machine bar. In the twin this opens the E-stop circuit as the machine's button does: the drives and the spindle drive lose power, and the controller raises ALARM:10. Press F1 again to release it, then follow the steps back to work: **Power on**, **Reset**, **Home**.

> ⚠ **Warning**
>
> F1 and the red key stop the twin. They are not an emergency stop for a real machine. At a real machine, use the machine's own emergency stop button.

The keys for the window

Key	Does
Alt+1 to Alt+6	Go to **Setup**, **CAM**, **Program**, **Run**, **Analysis** or **Diagnostics**
Ctrl+Tab, Ctrl+Shift+Tab	The next or previous workspace
Ctrl+J	Fold or unfold the dock
Ctrl+Shift+M	Switch machines
Ctrl+O	**Open program…**
Ctrl+,	**Settings…**
Ctrl+L	**Audit log…**
Ctrl+Q	**Quit**
? or /	**Keyboard shortcuts**
Alt and a menu's letter	Open that menu; Alt alone lights the menu titles
Tab, Shift+Tab	Move between the keys of a panel; Space presses the key with the frame
Esc	Close a pop-up or a sub-window

> **Note**
>
> F1 is the E-stop, not help. To open the manual, use **Help > Documentation** or a panel's **?** key.

The editor in the **Program** workspace has its own keys:

Key	Does
Ctrl+E	Into the editor and back to the machine
Ctrl+K	**Load and check**
Ctrl+S	**Save version**
Ctrl+Shift+S	Save the program as a file
Ctrl+D	**Compare** with a saved version
Ctrl+F, Ctrl+G, Ctrl+H	Find, go to a line, replace
Ctrl+B	**Breakpoint** on the line
Ctrl+Shift+B	**Pause on** a condition
Ctrl+.	Apply the check's suggestion for the line
Alt+↓, Alt+↑	The next or previous finding or difference
Shift+F10 or the menu key	The line's menu
Ctrl+Shift+O	**Optimize**
Ctrl+Shift+G	**Generate…**
Ctrl+Shift+R	Take the program that is in the machine
Esc	Close the comparison

The manual's window: Ctrl+F searches, Alt+← and Alt+→ go back and forward, Esc closes it.

While you arrange a workpiece in **Setup**: the arrow keys move it (with Shift finer, with Ctrl coarser), Q and E turn it, Enter places it, Esc puts it back.

The full list

Help > Keyboard shortcuts shows every key above in one list. These commands have no key; use their menus:

- **Single block** and **Next block** (the **Program** menu and the machine bar);
- **Check program**;
- zeroing the axes and going to the zero (**Machine > Work zero**);
- the tool setter and the probe moves (**Machine > Tool setter and probe**);
- the spindle speed in steps of 1000 rpm, and the rapid override (**Machine**);
- pausing the simulation and its speed (**Simulation**).

Using a screen reader

On Windows, Studio works with screen readers through the system's accessibility interface. In each of Studio's panels the reader gets:

- each key's name, what kind of key it is, whether it is on, open or unavailable, its shortcut and its tip;

- the readings as text: the machine bar (state, program, progress, the next step, the clock), the top bar (machine, state, AI, level, edition) and the right column (position, spindle, gauges, the modal line);
- each move of the focus with Tab;
- the menus: each item's words, its shortcut, whether it is unavailable or ticked, and which one you are on.

Two things are told as they happen, wherever the focus is: an alarm (with the next step) and an AI's request to change the machine.

The manual's window gives the reader the page's text, with its headings and links.

In 1.0:

- other changes of the machine's state are not told: move the focus to the machine bar to hear them;
- the listing, the console, the messages, the editor and the tool table are read by the reader's own rules; the curves are not read;
- on Linux, screen readers are not supported.

What it reads in the 3D view

The 3D view itself gives the reader nothing. The numbers it shows are in the right column and the machine bar, which the reader reads. When you click a point of the workpiece, the sentence under the view (the program line and the position) is read.

Large text and shop mode

Studio follows the display scale you set in Windows, screen by screen. It has no text size setting of its own.

View > Appearance > Shop mode (large) makes the machine bar, its keys, the E-stop, the text, the listing and the position larger, and gives the **Run** workspace a layout for reading at arm's length (see Light, dark and shop mode).

When the system's animations are turned off, the 3D view jumps to a new view instead of gliding there.

Troubleshooting

What to do when something does not work as you expect. Each section says what you see, what it usually means and where the full answer is. If none of them fits, collect the diagnostics and write to us (the last section).

Studio does not start

- Studio needs 64-bit Windows 10 or 11; the installer stops on other versions. See System requirements.
- If the window opens but the 3D view stays empty, or says that it is not available: **Tools > Settings... > Files**, tick **Software rendering (when the graphics card has trouble; at the next start)**, and start Studio again. Updating the graphics card's driver often fixes the cause. See When the 3D view cannot start.
- If Studio closed while it started, the next start offers the report (see Studio closed unexpectedly).

The machine is in alarm

In an alarm the state on the machine bar turns red. The bar names the alarm and what it means, and gives the **Next step** with the key that does it; **All steps** lists the way back to work when there is more than one. Click the key, or use **Machine > Do the next step**: Studio works the steps out again after each one.

- A probe move that touched nothing stops with ALARM:5: do the steps (**Reset**, **Unlock**), then check where the cycle expected the face.
- A move past the machine's travel stops at the soft limit: look at the line the listing marks red and at the work zero it ran with.
- After an alarm the position may no longer be trusted (a line under **Position** says so): home again before you cut.

The steps for each case: Alarms and what to do. Every alarm, its causes and what clears it: Alarms.

A line of the program answers an error

`error:` and a number means the controller refused that line: it did not run it, and the machine did not move for it. The message gives the number and what it means.

- **Program > Check program** sends the whole program through the controller's check mode while the machine does not move, and lists the first lines it refused: "line 5: error:22 (...)". Fix those lines and check again.
- The commonest causes are a code the controller does not take, a word missing (an arc without its centre or radius, a cutting move before any `F`) or two codes in one line that cannot go together.

Every error and its causes: Errors.

The program check finds a problem

The editor checks the program as you type, when it is loaded and after a run. Each finding stands under its line with its rule's number and its kind: an error, a warning or a note.

- An **error** is a line the controller would refuse or a move the setup cannot take; fix it before you run.
- A **warning** is a line that runs but may not do what you mean: a rapid below the work's top, a spindle started without `S`, a tool change after which the program moves down before it is over its start.
- The preview part of the check needs the machine powered and homed and no program running; until then it waits.

The rules and how to fix a finding: The program check. What a run in the twin checks beyond that: What the twin checks.

The probe cycle will not start

Start probing... stays grey while a check on the probe card is red, or while something is missing; the line under the keys then says what (give X and Y, give the ring's diameter, wait for a call that is going on).

- **The spindle holds no touch probe**: in **Setup**, step **Tool**, choose the touch probe and click **Change tool**.
- **The spindle turns**: stop it (**Spindle stop**, F11).
- **No room above the blank to move across**: the line gives the probe tip's height with Z at the top, the setup's top and how much is missing. Use a shorter probe, or hold the blank lower: clamps or double-sided tape instead of the vise.
- **The twin follows a real machine**: probe there with the machine's own control.
- A cycle that touched nothing stops with ALARM:5 (see The machine is in alarm).

See When a cycle stops.

Licence and activation

Help > Licence shows the edition in force, the licence's state and when it ends.

- **Activating online**: type the code as you received it; **Activate** is available once the code reads as one. The server's answer appears under it. When the answer is that the licence is on too many computers, has ended, is paused, unpaid or refunded, **Open the licence page** opens it on our website: there you free a computer, renew or manage the subscription.
- **Without internet**: the same dialog has the offline steps. Studio gives a request code; on a computer with internet you get the licence file for it on the licence page, then import that file here.

- A subscription that is not renewed keeps working for 14 more days (the licence shows **In grace**). After that, or when a licence ends, Studio is the Learning edition again; your projects and programs are untouched.

See Licensing, and When a licence ends.

The AI assistant does not answer

- The line under the input says what the assistant is doing (waiting for the service, the twin working on a tool).
- Check the **Provider**, the **Base URL** and the **Model**; for a cloud service, that a key is saved (**Save key**).
- For a local model, its server must run (LM Studio's server, Ollama) and a model be chosen (**List models**).
- An answer can take a while; **Stop** ends it.
- The first message to a cloud service waits for you: Studio shows once what goes to it, and sends only when you agree (asked once for each service). **Use a local model…** keeps everything on this computer. See Privacy.
- When the AI wants to change the window's machine, a bar at the bottom of the window asks you. With no answer in 60 s, the request counts as refused.

See The AI assistant.

Studio closed unexpectedly

At the next start, Studio asks what to do with the report of what happened: open its folder, or save it as a file. Nothing is sent. The report leaves out programs, file and folder names, keys, codes and people's names; reports older than 30 days are deleted at the start. See Privacy.

Getting help

Help > Collect diagnostics information… puts what support needs into one ZIP file in `Documents\Twinwright Studio\Reports`: the version, the settings (keys written as `***`), the machine archive, the controller's settings, the last 7 days of the audit log, the crash reports and the licence's state. **The program and the setup now** is left out unless you tick it. Paths, file names and the computer's user name are removed.

Open the file and look before you send it; Studio sends nothing itself. The address to write to is in **Help > About**.

Operations manual

System requirements

What a computer needs to run Twinwright Studio, and what the 3D view needs from its graphics.

Windows

- 64-bit Windows 10 or Windows 11 (x64). The installer stops on other versions.
- Nothing else to install: Studio is one program file and uses only the libraries that come with Windows.

Linux

- Linux on x86-64 with glibc 2.38 or later: Ubuntu 24.04, Debian 13, Fedora 39 and newer, and the distributions built on them.
- A desktop with X11 or Wayland.
- For the AppImage: FUSE 2 (on Ubuntu, the package `libfuse2t64`). Without it, use the `.tar.gz`.
- For online activation and trials: libcurl (on Ubuntu, the package `libcurl4t64`; desktops usually have it). Without it, Studio activates offline.
- For the AI services' keys: the desktop's keyring (Secret Service: GNOME Keyring or KWallet). Without one, Studio keeps the keys in a file only you can read.

Graphics

The 3D view uses Direct3D 11, on any graphics card or built-in graphics that has it (feature level 10.0 or higher). Without one, Studio draws with Windows' software renderer (WARP): it works, but the view is slower (at most 30 frames a second) and there is no tool close-up.

Help > About shows which one Studio uses under **Display**.

When the 3D view cannot start

The rest of the window works without the 3D view. If the view stays empty or shows that it is not available:

1. Open **Tools > Settings... > Files**.
2. Tick **Software rendering (when the graphics card has trouble; at the next start)**.
3. Close Studio and start it again.

Updating the graphics card's driver often fixes the cause.

Linux

On Linux, the 3D view uses OpenGL 4.1 or later: the Mesa drivers have it for the graphics of the last ten years. Without a graphics driver, Mesa's software renderer draws the view, more slowly; to try it, start Studio from a terminal with `LIBGL_ALWAYS_SOFTWARE=1` before the program's name.

Memory and disk

- Memory: 8 GB or more.
- Disk: about 62 MB for the program and its manual, and room for your data. Your data grows with use: the audit log is kept for 180 days and up to 2 GB by default (**Tools > Settings... > Log**), and every program version and report is kept (see Your data and backups).

Screens and scaling

- At least 1280 × 720; Studio is laid out for 1600 × 1000 and more.
- Any display scale. Studio follows each screen's scale, and moving the window to another screen rescales it.

A real machine

In 1.0, connecting to a real machine is a preview, from the command line (see A real machine): a serial port (USB) at 115 200 baud, or a controller on the network.

Studio uses the network as follows:

- G-code senders connect to the twin on this computer, port 2323;
- the browser view uses port 8765, on this computer;
- AI clients that take an address use port 8766, on this computer, when you turn it on;
- online activation and renewal go to the licence server.

Studio's ports listen on this computer only: other computers cannot reach them.

On Linux, the serial port belongs to the `dialout` group: add your account to it once (`sudo usermod -aG dialout $USER`) and log in again.

Installing and removing

How to install Twinwright Studio, without questions on many computers at once, what goes where, and how to remove it.

For Linux, see On Linux.

Installing

1. Run `TwinwrightStudio-<version>-setup.exe`.
2. Read the licence agreement and click **I Agree**.
3. Keep the folder (`%LOCALAPPDATA%\Programs\Twinwright Studio`), or choose another.
4. On the last page, choose whether to start Studio and whether to put a shortcut on the desktop.

The installer speaks English or Chinese, following the system's language.

For you or for everyone on the computer

The installer installs Studio for the person who runs it. It needs no administrator rights, and the Start menu entry and the desktop shortcut are that person's. Each person who uses the computer installs Studio for themselves.

The licence belongs to the computer, not to the person: every user of the computer shares it (see The licence folder).

Studio changes no file associations, no `PATH` and no firewall settings, and installs no service.

Installing without questions

```
TwinwrightStudio-<version>-setup.exe /S [/DESKTOP] [/D=<folder>]
```

- `/S` installs without pages and does not start Studio.
- `/DESKTOP` adds the desktop shortcut.
- `/D=` sets the folder. It must be the last argument, without quotes, even when the folder has spaces.

To remove Studio without questions: `uninstall.exe /S` in the program's folder.

What goes where

What	Where
The program	`%LOCALAPPDATA%\Programs\Twinwright Studio`
Your work: programs, versions, reports, the audit log	`Documents\Twinwright Studio`
The program's own state: settings, machines, AI keys	`%LOCALAPPDATA%\Twinwright\Studio`
The licence	`%ProgramData%\Twinwright\licence`

The program's folder

`TwinwrightStudio.exe`, the manual (`manual\`), `README.txt`, `THIRD-PARTY-NOTICES.txt` and `uninstall.exe`. The folder stays the same over updates, so AI clients you connected keep working.

Your documents

`Documents\Twinwright Studio` (the Documents folder Windows gives you, also when it is redirected):

- `programs\`: every program's versions, and the CAM jobs' versions;
- `Reports\`: inspection reports, setup sheets and diagnostics packages;
- `logs\`: the audit log.

Tools > **Settings...** > **Files** and **Log** can move the programs and the log elsewhere (at the next start).

The program's own state

`%LOCALAPPDATA%\Twinwright\Studio`: the settings, the window's place, each machine's world (its workpiece, tools, offsets and controller settings), the AI services' keys (encrypted for your Windows account) and crash reports (kept 30 days).

The licence folder

`%ProgramData%\Twinwright\licence` holds the licence files, trials included, for every user of the computer. Studio makes it at the first activation or trial. If Studio cannot write there, an administrator can make the folder and let users write in it.

For special setups, these environment variables move the folders:

Variable	Moves
`TWINWRIGHT_STATE_DIR`	The program's own state

Variable	Moves
`TWINWRIGHT_WORKDIR`	The programs folder
`TWINWRIGHT_LOGDIR`	The audit log
`TWINWRIGHT_LICENCE_DIR`	The licence folder

Updating to a new version

Close Studio, then run the new version's installer. It installs into the same folder; your work, settings, machines and licence stay as they are. Studio does not check for updates itself (see Updates).

If a copy of Studio is still running in the background (an AI client may have started one), the installer moves the old program file aside and removes it at the next update or removal.

Removing

Use **Settings > Apps > Twinwright Studio > Uninstall** in Windows, or run `uninstall.exe` in the program's folder.

What removing leaves behind

Removing takes away the program, its shortcuts and its entry in **Apps**. It leaves:

- your work in `Documents\Twinwright Studio`;
- the program's own state in `%LOCALAPPDATA%\Twinwright\Studio`;
- the licence folder;
- the `twinwright` entry in the settings of AI clients you connected.

Delete these by hand to remove everything. Deleting the licence folder does not give a trial back: the licence server gives each computer one trial of each edition.

On Linux

Installing

Studio for Linux is one file, an AppImage. Nothing is installed in the system.

1. Check the file: `sha256sum TwinwrightStudio-<version>-x86_64.AppImage` must give the SHA-256 on the download page.
2. Put it where it can stay, under a name without the version, so that AI clients you connect keep working after an update, and make it executable:

```
mkdir -p ~/Applications
mv TwinwrightStudio-<version>-x86_64.AppImage ~/Applications/TwinwrightStudio.AppImage
chmod +x ~/Applications/TwinwrightStudio.AppImage
```

1. Start it: `~/Applications/TwinwrightStudio.AppImage`, or double-click it in the file manager.

The AppImage needs FUSE 2. If it does not start for want of it, install it (`sudo apt install libfuse2t64` on Ubuntu 24.04 and later), or take the `.tar.gz` instead: unpack it anywhere and start `TwinwrightStudio` in its folder.

What goes where on Linux

What	Where
The program, with its manual	the AppImage, where you put it
Your work: programs, versions, reports, the audit log	`~/Documents/Twinwright Studio`
The program's own state: settings, machines	`~/.local/state/twinwright/studio` (under `$XDG_STATE_HOME` when it is set)
The AI services' keys	the desktop's keyring; without one, a file only you can read in the state folder
The licence	`/etc/twinwright/licence`, or `~/.local/share/Twinwright/licence`

The licence folder is the computer's, for every user, when your account can make it or write in it. Otherwise Studio keeps the licence in `~/.local/share/Twinwright/licence` (under `$XDG_DATA_HOME` when it is set): for you alone, and still for this computer only. To share one licence among the users of a computer, make the folder once with `sudo install -d -m 1777 /etc/twinwright/licence`.

Trial and activation on Linux

Trials, activation and renewal go online as on Windows, through the system's libcurl. Without libcurl or without internet, activate offline (see Licences).

Updating and removing on Linux

To update, quit Studio and put the new AppImage in place of the old one, under the same name (step 2 above). To remove Studio, delete the AppImage. Both leave your documents, the state folder, the licence folder, the keys in your keyring and the `twinwright` entry in AI clients you connected; delete these by hand to remove everything.

Licences, seats and activation

Without a licence, Studio runs as the Learning edition. A paid licence adds the features of its edition. This chapter is for the person who buys, activates and looks after licences. Prices are on the website.

If paid licences are not on sale yet when you read this, the pricing page on the website says so. Until then Studio runs as the Learning edition, and you can try Maker or Pro free for 30 days (see Try a paid edition); the sections on activation apply once licences are sold.

Editions

Edition	Adds	Who may use it
Learning	The whole twin, the editor, the real machine, the examples, the AI connection, and CAM for pockets, contours, drilling, engraving and 3D parallel finishing	Anyone, free, for learning, teaching and personal projects that earn no money
Maker	Adaptive clearing, every 3D finishing strategy, rest machining, two sides and several setups	One person, including selling what that person makes
Pro	Everything in Maker, plus nesting, the program optimizer, and inspection with the probe (against the design) with its reports	Your organisation, for any commercial use, per seat

An **MC-1 owner** licence is Pro for one seat, for use with the MC-1 whose serial number it names. It passes to the next owner with the machine.

A paid licence comes as one of these kinds:

- **Subscription**: by the month or by the year, until you cancel it.
- **Bought outright**: yours for good. It covers every version released within 12 months of the purchase, or of a paid renewal of updates.
- **MC-1 owner**: like a licence bought outright, tied to the machine.

Seats and computers

A paid seat is used by one person at a time, on up to three computers. A computer that several people share counts once.

Each activation binds the licence to one computer by a hash of the computer's fingerprint. The fingerprint itself never leaves the computer. To see which computers a licence is on, open the licence page, twinwright.com/licence, and enter the activation code.

Try a paid edition

You can try each paid edition free for 30 days, once on each computer.

1. Use a paid feature, for example adaptive clearing in the CAM panel. Result: Studio says which edition the feature needs.
2. Choose **Start a 30-day trial...**.
3. Read what the trial adds and until when, then choose **Start the trial**. Result: Studio asks Twinwright's licence server for the trial, and the edition's features work until the date shown.

For a trial Studio sends the edition, a hash of the computer's fingerprint and Studio's version, nothing else. The server gives each computer one trial of each edition: if you reinstall Studio during the trial, starting it again brings back the same end date. When it ends, Studio is the Learning edition again, and your projects and programs are untouched.

On a computer without internet, Studio shows a request code instead. On a computer with internet, open the licence page, twinwright.com/licence, choose the edition under "Try on a computer without the internet", paste the request code and download the trial file. Then import the file on the first computer, as in Activate a computer without internet.

Activate online

You need the activation code from the e-mail you received with your licence, in the form `TW-XXXXX-XXXXX-XXXXX-XXXXX`, and an internet connection.

1. Choose **Help > Licence...**.
2. Choose **Activate online...**.
3. Type the activation code. Upper or lower case, spaces and hyphens do not matter.
4. Choose **Activate**. Result: the licence file is in the licence folder, and the dialog shows the edition.

Activating sends only the activation code, a hash of this computer's fingerprint and Studio's version.

If the licence is already on as many computers as it allows, free one on the licence page first (see Move a licence to another computer).

Activate a computer without internet

Use this for a lab or workshop computer that is never online. You need the activation code, a second computer with internet, and a way to carry a small file, such as a USB stick.

1. On the computer without internet, choose **Help > Licence...**.
2. Choose **Activate offline...**.
3. Choose **Copy**. Result: the request code is on the clipboard. It holds a hash of this computer's fingerprint and Studio's version, no personal information.
4. Paste the request code into a text file, and carry it to the computer with internet.

5. There, open twinwright.com/licence. Under "A computer without internet", paste the activation code and the request code.
6. Choose "Download the licence file". Result: a file named like `twinwright-TWL-….twlic` downloads.
7. Carry the file back. In Studio, choose **Help > Licence...** and **Import a licence file...**, then open the file. Result: the licence file is in the licence folder.

A computer without internet does not renew by itself. Repeat these steps:

- before a subscription's period ends;
- for a licence bought outright, once more after its first 30 days (the refund period), to get the permanent file;
- after you buy more years of updates.

The licence folder

The licence folder belongs to the computer, not to one Windows account. A licence in it works for everyone who uses the computer.

System	Folder
Windows	`%ProgramData%\Twinwright\licence`
Linux	`/etc/twinwright/licence`; where it cannot be written: `~/.local/share/Twinwright/licence`

The environment variable `TWINWRIGHT_LICENCE_DIR` sets another folder. To open the folder, choose **Help > Licence...** and **Open the licence folder**.

The folder holds the licence files (`*.twlic`, trials included) and a file with the last time Studio looked. It can hold several licence files, for example an MC-1 owner licence and a subscription. The edition in force is the highest of all valid licence files.

> **Note**
>
> Do not edit the files. A licence file that has been changed no longer counts: download it again from the licence page and import it.

If Studio cannot create or write the folder, an administrator creates it and lets the users write in it. On Linux, run the import as root (`sudo`), or set `TWINWRIGHT_LICENCE_DIR`.

Renewal and the 14 days' grace

With **Renew before it ends (it sends only the licence's number, a hash of this computer's fingerprint and Studio's version)** on, Studio renews by itself when it is online:

- a subscription, from five days before its period ends;
- a licence bought outright, at the end of the refund period, and from 14 days before its updates end when you have bought more years.

Studio tries once when it starts, at most once a day. A failed try does not interrupt you. It sends only what the option names.

If a subscription is not renewed, everything keeps working for 14 more days. The licence shows **In grace** and the days left. After that, Studio is the Learning edition.

When a licence ends

Ending, cancelling or a failed check only ever return Studio to the Learning edition:

- your projects open as before;
- the G-code you made can still be exported and sent to the machine;
- the paid operations wait until you renew.

Studio never deletes your files or locks your data.

Updates for a licence bought outright

A licence bought outright covers the versions released within 12 months of its purchase, or of a paid renewal of updates. A later version runs as the Learning edition, and the licence shows **Updates over**. Keep using the last version it covers, or renew a year of updates on the licence page. See Updating Studio.

Move a licence to another computer

1. On the licence page, enter the activation code and choose "Show my computers".
2. Choose "Free this computer" beside the computer you no longer use.
3. Activate the licence on the new computer.

You can free up to three computers in 30 days. For more, write to hello@twinwright.com.

New hardware or a reinstalled system changes the fingerprint. Studio then says **This licence is bound to another computer.** Free the old entry, then activate again. Until then, Studio runs as the Learning edition.

Lost activation code

Use **Lost your activation code?** on the licence page. Where it asks for an e-mail address, enter the one you bought with: each licence bought with it gets a new code by e-mail. Where it asks you to write to us, write to hello@twinwright.com from the address your activation code was sent to; we check it and send you a new code. Either way the old code stops activating, and computers already activated keep working.

The computer's clock

If the clock is set back by more than two days, Studio says **This computer's clock is earlier than the last time Studio looked.** The later time counts, so no licence or trial gets days back. Set the date and time right.

Studio keeps the last time it saw in two places. The later of the two counts, so deleting one of them brings no days back.

If a licence or a trial shows as ended too early after the clock was once set to a date in the future, set the date and time right and write to hello@twinwright.com.

For administrators: the command line

The program takes these options without opening its window. A folder is optional; without one, the computer's licence folder is used.

```
TwinwrightStudio.exe --licence-status [FOLDER] [--json]
TwinwrightStudio.exe --licence-request [FOLDER]
TwinwrightStudio.exe --licence-import FILE [FOLDER]
```

- `--licence-status` lists each licence file and trial, whether it holds and why; `--json` gives every field.
- `--licence-request` prints this computer's request code.
- `--licence-import` checks a licence file and copies it into the licence folder.

On Linux the program is `twinwright-studio`. AI clients can read the same status with the read-only MCP tool `licence_status`.

What Studio sends, and what our server keeps: Privacy.

Your data and backups

Studio keeps your work in your documents and the machines' state in a folder of its own. This chapter lists what is where, what to back up and how to move to another computer.

What is where

Two folders hold everything that is yours:

- `Documents\Twinwright Studio`: your work;
- `%LOCALAPPDATA%\Twinwright\Studio`: the program's own state.

Jobs, programs and their versions

`Documents\Twinwright Studio\programs\<program>\` holds one file for each version of a program (`v0001.nc`) and a note of when, by whom and why it was saved (`v0001.json`). The CAM job that made a program is versioned next to it (`job-v0001.json`). Studio never prunes them.

Tools > Settings... > Files can put the programs folder elsewhere (at the next start).

Reports

`Documents\Twinwright Studio\Reports\`: inspection reports (PDF and CSV), setup sheets (HTML) and diagnostics packages (ZIP).

The machines and their worlds

Each machine has a world in the state folder: the table, the workpiece with its cuts, the tools and the tool table, the work offsets, the controller's settings, the temperatures, the running hours and the probe's calibration. Studio writes it when you close the window. The first machine's world is in `world\`, the others' in `machines\<id>\world\`.

Settings

`settings.json` in the state folder holds all of **Tools > Settings...**; `window.json` the window's place and size.

The audit log

`Documents\Twinwright Studio\logs\` holds a file a day (`audit-YYYY-MM-DD.jsonl`). The records are chained, so a change or a gap shows when you verify the log (**Tools > Settings... > Log > Verify integrity**). By default the log keeps 180 days and at most 2 GB; **Tools > Settings... > Log** changes that. Copy the whole folder to keep the chain whole.

What to back up

Close Studio, then copy both folders: `Documents\Twinwright Studio` and `%LOCALAPPDATA%\Twinwright\Studio`.

Two things do not move with a copy:

- the AI services' keys: they are encrypted for your Windows account on this computer. Type them again on the new computer;
- the licence: it belongs to the computer. Activate it again on the new one (see Licensing).

Moving to another computer

1. On the old computer, close Studio and copy both folders.
2. Install Studio on the new computer and start it once, then close it.
3. Copy both folders into the same places on the new computer.
4. If you moved the programs or the log folder in the settings, check that those folders exist.
5. Start Studio, activate the licence and type the AI keys again.
6. Connect your AI clients again (**AI > Connect AI clients...**).

Starting a machine afresh

A factory reset puts the machine in your window back as it came: its world goes (the workpiece, the tools, the offsets, the controller's settings, the probe's calibration) and Studio starts again.

1. Choose **File > Factory reset...** (Technician level and up).
2. Within 5 seconds, choose it again to confirm.

Only that machine is reset. Your programs, reports, the audit log, the settings and the other machines stay. The reset is recorded in the audit log.

Updating Studio

Studio does not check for new versions by itself, and it never updates itself. You decide when to update.

Find out whether there is a new version

1. In Studio, choose **Help > About**. Result: the line under the program's name shows your version, for example "Version 1.0.0 · built 2026-10-10 · commit 4f1c2ab".
2. Open the download page, twinwright.com/download. Result: the page names the current version and the installer's SHA-256 checksum.

If the versions differ, the download page has the newer one.

Install an update

You need the new installer. You do not need administrator rights: Studio installs for your Windows account.

1. Download the installer from the download page.
2. Optional: compare its SHA-256 checksum with the one on the page. In PowerShell, run `Get-FileHash` with the file's name.
3. Start the installer. If Windows says the publisher is unknown, do not run it: write to hello@twinwright.com.
4. If Studio is open, close it when the installer asks.
5. Follow the installer. Result: the new version replaces the program in the same folder.

Your settings, AI keys, program versions, audit log and licence stay as they are: they are not in the program's folder. AI clients you connected keep working, because the program's path does not change.

For many computers, the installer runs silently: `setup.exe /S`. See Installing Studio.

Which versions your licence covers

Licence	Versions with its paid features
Learning edition	Every version
Subscription	Every version, while the subscription runs
Bought outright, MC-1 owner	Versions released within 12 months of the purchase, or of a paid renewal of updates

A version released after a licence's updates end runs as the Learning edition. The licence then shows **Updates over**. Before you update, check the date in **Help > Licence...**. Keep the installer of the last version your licence covers.

To keep updating, renew a year of updates on the licence page. See Licences, seats and activation.

Computers without internet

1. On a computer with internet, download the installer.
2. Carry it to the computer without internet, for example on a USB stick.
3. Install it there as above.

A newer version does not change the licence file. Renew it as before: Activate a computer without internet.

Go back to an earlier version

1. Keep a backup of your files before you update.
2. Uninstall Studio: in Windows Settings, under Apps.
3. Install the earlier version's installer.

> **Note**
>
> An earlier version may not open projects or settings saved by a later one. Keep the backup until you are sure.

Privacy and what Studio sends

Studio works on your computer. Studio itself sends only what starting a trial and activating and renewing a paid licence need. There is no telemetry, no update check and no automatic crash upload. If you connect the AI assistant to a cloud service, your messages and what you attach go to that service: see The AI assistant and cloud services. This chapter lists what goes out, what stays on the computer, and how to remove it. The website's privacy policy, twinwright.com/privacy, is the binding text.

What Studio sends

When	To	What
You start a trial of a paid edition	Twinwright's licence server, over HTTPS	The edition, a hash of the computer's fingerprint, Studio's version
You activate a paid licence	Twinwright's licence server, over HTTPS	The activation code, a hash of the computer's fingerprint, Studio's version
Studio renews a paid licence (see Renewal)	Twinwright's licence server, over HTTPS	The licence's number, the same hash, Studio's version

That is all Studio itself sends to Twinwright. The fingerprint never leaves the computer as it is; only a hash of it does. A computer activated offline sends nothing at all.

Three things go out only when you set them up:

- **The AI assistant, connected to a cloud service.** It sends your messages and what you attach to that service; see below.
- **An AI client you connect over MCP**, such as Claude or ChatGPT. The client sends what it reads from the twin to its own service, under its terms.
- **A web page Studio opens**, such as the licence page. Your browser opens it; Studio adds nothing to it.

The AI assistant and cloud services

The AI assistant works with the provider you choose in it: a cloud service such as Anthropic, OpenAI or DeepSeek, with your own account and API key, or a local model.

When you send a message to a cloud service, Studio sends to that service, from your computer and with your key:

- your message, with the conversation so far;
- the window's machine, setup, program (with line numbers) and the problems the checks found, while **Attach the window's program, setup and check results (when they changed)** is ticked. The box is ticked at first. Studio attaches them to the first message, and again only when they have changed;
- the results of the tools the assistant uses in Studio, such as the check or simulation of a program, the machine's status or entries of the log, and pictures of results for services that accept pictures.

Before the first message goes to a cloud service, Studio shows once what it will send.

The service handles all of this under its own terms; Twinwright receives none of it. To send less, untick the box before you send, or start again with **New conversation**.

With a local model (LM Studio, Ollama), all of this stays on the computer.

On this computer, the audit log keeps the first 180 bytes of each message (see The audit log), and the API keys are stored encrypted (see What Studio keeps on the computer).

What Studio keeps on the computer

What	Where (Windows)	Who sees it
Settings, the state of the twin, AI keys	`%LOCALAPPDATA%\Twinwright\Studio`	Your Windows account
Program versions	`Documents\Twinwright Studio`	Your Windows account
The audit log	`Documents\Twinwright Studio\logs`, unless Settings name another folder	Your Windows account
Inspection reports, setup sheets, diagnostic files	`Documents\Twinwright Studio\Reports`	Your Windows account
Reports of unexpected exits	`%LOCALAPPDATA%\Twinwright\Studio\crash`, deleted after 30 days	Your Windows account
Licence files, trials included	`%ProgramData%\Twinwright\licence`	Everyone on the computer

AI keys are stored encrypted for your Windows account (Windows DPAPI). Other accounts on the same computer cannot read them.

The audit log

The audit log is a file a day, `audit-YYYY-MM-DD.jsonl`, in the log folder. It records:

- what the AI and you had the machine do;
- the first 180 bytes of each message in the AI panel;
- licence checks: the licence's number, the edition and the result, without the holder's name.

Keys, tokens and passwords are written as `***`. Settings set how much the log records and how long it keeps it. The log stays on the computer; Studio never uploads it.

If Studio closes unexpectedly

At the moment of the failure, Studio writes a small report: where it failed, the order of calls that led there, and the list of program modules. It writes no memory contents.

At the next start, Studio adds the version information and the last 200 lines of its log, cleaned:

- paths and file names are replaced by placeholders;
- names and e-mail addresses are removed;
- there is no G-code, no AI conversation, no licence code and no fingerprint.

Studio then asks once. You can open the report's folder, save the report as a file, or close the message. Nothing is sent. To help us find the cause, send the file to hello@twinwright.com yourself.

Diagnostic information

When you ask us for help, we may ask for diagnostic information. In the **Help** menu, choose Collect diagnostic information. Studio saves a ZIP file in the reports folder with the items you tick:

Item	Ticked at first
Version, system, graphics, language and units	Yes
Settings, without the AI keys	Yes
The machine profile and the controller's `$` settings	Yes
The last 7 days of the log and the reports of unexpected exits, cleaned as above	Yes
The licence status: edition and end date, without codes or the fingerprint	Yes
The current program and setup: your part's data	No

Open the file and read it before you send it. Nothing is sent automatically.

What listens on the network

Studio's servers for AI clients (MCP) and for sender software listen only on this computer (`127.0.0.1`). Other computers cannot reach them.

The command-line option `--bind any` opens the twin's network ports, such as the one for sender software, to your network. The MCP server stays local.

> ⚠ **Warning**
>
> In shadow mode, the sender port passes commands to the real machine. With `--bind any`, anyone on your network who reaches that port could move the machine. Use `--bind any` only for the twin, on a network you control.

What our server keeps

For each paid licence, the licence server keeps:

- the hashes of up to three computers a seat, with the dates each was first and last seen, so that you can free one on the licence page;
- the activation code only as a hash: not even we can read it back;
- for 90 days, a log of trials, activations, renewals and changes, with the computer's hash, Studio's version and your IP address shortened to its network part.

For each trial, it keeps the computer's hash with the edition and the trial's dates, so that each computer has each trial once.

Nothing about your work reaches the server.

Removing what Studio keeps

1. Free the computer on the licence page, so that it no longer counts (see Move a licence to another computer).
2. Uninstall Studio.
3. Delete the folders in the table above that you no longer need. Result: nothing of Studio remains on the computer.

Uninstalling removes the program, not your files: your projects, program versions, logs and reports stay until you delete them.

Labs, classrooms and shared computers

This chapter is for the person who sets up Studio on computers that several people use: in a lab, a workshop or a classroom.

One licence for the whole computer

The licence folder belongs to the computer, not to one Windows account (see The licence folder). A licence activated on a computer works for everyone who signs in to it.

A paid seat is used by one person at a time, on up to three computers, and a computer that several people share counts once. So:

- give a shared computer, such as the one beside the machine, a seat of its own;
- people who also use the paid features on their own computers need seats for those.

Activate the shared computer once, with the account of the person who looks after it.

What each person keeps

Studio installs for each Windows account. Each person has their own:

- settings and AI keys: the keys are encrypted for that account, and other accounts cannot read them;
- program versions, reports and audit log, in their own `Documents\Twinwright Studio`.

A trial of an edition counts once for the computer, for everyone on it: the licence server gives each computer one trial of each edition.

For installing on many accounts and computers, see Installing Studio.

Classrooms

The Learning edition is free on every computer, for learning and teaching. Nothing needs to be activated: install Studio, and the whole twin, the editor and the CAM for everyday parts work.

Give each student their own Windows account if you can. Their settings, programs and logs then stay apart.

To teach with the paid features, schools and training centres can get Pro seats. Write to hello@twinwright.com.

Computers that are restored at each start

Some lab computers return to a saved image at each start, which also empties the licence folder. On such a computer:

1. Activate the computer once, offline (see Activate a computer without internet), and keep its licence file.
2. Put the file into the image's licence folder, or import it at each start:

```
TwinwrightStudio.exe --licence-import D:\licences\lab-pc-3.twlic
```

A licence file is made for one computer. Each computer of the room needs its own request code and its own file.

Computers without internet

Activate them offline, and renew them by hand: before a subscription's period ends, and once after the first 30 days of a licence bought outright. See Activate a computer without internet.

When the licence folder cannot be written

On Windows, any account can create `%ProgramData%\Twinwright\licence`. The account that creates it can change its files; the other accounts can read them. That is enough: one person imports the licence, and everyone uses it.

If your organisation's policies do not allow this:

- an administrator creates the folder and lets the users write in it; or
- the system environment variable `TWINWRIGHT_LICENCE_DIR` names a folder everyone can read.

On Linux, the folder is `/etc/twinwright/licence` when your account can make it or write in it; otherwise Studio uses your own `~/.local/share/Twinwright/licence` (under `$XDG_DATA_HOME` when it is set), for you alone. To share one licence among the computer's users, make the folder once with `sudo install -d -m 1777 /etc/twinwright/licence`.

AI on a shared computer

Each person connects their own AI provider or AI client, with their own key. The audit log of each account records what that person and the AI had the machine do. See Privacy and what Studio sends.

Technical manual

The twin's models and their limits

The twin is the MC-1 as a program: the Twinwright controller (kernel) runs in it, and physical models stand for the parts of the machine around the controller. Every number the twin gives about the physical world is an estimate from a physical model. Its parameters come from the machine's design, the parts' data sheets and published values; each MC-1 is to be measured at commissioning and its twin set to the measurements (planned). Each section below ends with what its parameters rest on.

Draft (round 14): the text is being written; the sections may still grow.

The controller

The controller in the twin is the Twinwright controller (kernel), built from the same source as the machine's firmware. What is modelled is the board around it: its timers and step pulses, its inputs with their debounce, its link to the computer (the USB serial port and its handshake), the time the controller's foreground takes.

Based on: the kernel's own code; the foreground's time is an estimate until it is measured on the board.

Motion and drives

Each axis is driven by a closed-loop stepper drive and a ball screw. The twin models them at four levels of detail; the default is a three-mass model:

- the drive's position loop: the motor's magnetic spring around the commanded position, its torque falling with speed, pull-out when the load is too high, the drive's alarm when the following error grows too large;
- friction in the guides, the nut and the bearings, and gravity on Z;
- the transmission's compliance between the motor, the screw and the table, the nut's backlash, the frame under the X axis;
- the drive's power: the E-stop chain, the contactor, the time the drives take to come up.

Based on: the design (masses, inertias, stiffness by hand calculation and finite-element analysis), the drives' and motors' data sheets; friction and the drive's internal parameters are assumptions until measured.

Geometry and accuracy model

When it is on, the twin moves the tool tip by the machine's geometric errors: each axis's lead error and its periodic part, two straightness errors and three angular errors acting through their Abbe arms, the squareness between the axes, the nut's backlash, and the screws' and the spindle's growth with heat. The limit switches see the slides, the tool setter and the probe see the tool tip, as on the machine. The model is off by default: the nominal twin is ideal.

Based on: tables of a typical assembled machine drawn from the parts' grades. They are not the machine's specification; they are replaced by the machine's ISO 230-2 measurement at commissioning (planned).

Heat

The spindle and the ball screws warm with use and cool at rest; the twin keeps their temperatures between sessions and moves the tool tip and the screws' lead by them.

Based on: first-order thermal models with assumed parameters until the machine's warm-up is measured.

Spindle and drive (VFD)

The spindle's drive is modelled as the machine's: the controller talks to it over RS-485 in the drive's own protocol, byte by byte at the configured speed, and the drive answers as its manual describes, including its parameters that decide whether it listens. Behind it: the motor's torque from its slip, its current and the drive's overcurrent, overload and undervoltage trips, the DC bus that keeps the drive alive through a short power dip, the water pump.

Based on: the drive's and the motor's data; the bus capacitance, the current's magnetising part and the friction are assumptions until measured.

The cut

The blank is a height field with finer regions where you ask for them. Every move of the tool removes what the tool's shape sweeps, so the part's shape comes from the program itself.

The cutting force comes from the material's cutting coefficients, which change with the chip's thickness and the cutting speed, worked out tooth by tooth on the tool's real shape (flat, ball, bull nose, V). From the force the twin estimates the spindle's load and power, the tool's deflection and the risk of breaking the tool: the bending stress at the flute root against the carbide's strength.

Based on: published cutting data for the materials (each material's source is in the machine profile), a sharp tool, typical tool geometry.

Chatter

For each cut the twin computes the depth at which it would start to chatter at the spindle speed in use, from the tool's and the machine's vibration modes and the cut's geometry, and picks a spindle speed where the cut is stable. It also estimates where the walls end up from the cut's forced vibration.

The tool point has three modes in each direction: the machine's structure, the tool holder in the collet and spindle, and the tool's own bending. The bending mode's frequency comes from the tool's shape - its flutes, neck and shank and how far it sticks out: about 14 kHz for a 6 mm end mill 20 mm out, over 100 kHz for a 0.2 mm one. Tools under about 1 mm chatter, if at all, at their own bending, at a depth that hardly depends on the spindle speed; the twin works that depth out in time for every cut.

Based on: the frame's modes from finite-element analysis, an assumed holder mode, and the tool's bending estimated from its geometry with beam theory; the damping of all three is assumed (3 %). None of them is measured yet: they are to be measured by a tap test on the built machine, and until then the stable depths are estimates. The damping of the cut itself is not modelled: for small tools the margin is on the safe side.

Runout

A cutter whose axis is off the spindle's cuts with its outermost tooth: bores come out larger and bosses smaller (finding 187). The twin cuts with that envelope; CAM allows for it in fits. A tool can carry its own runout - a precision collet's class, or as measured - in its spec (`tir3`: its TIR in micrometres); otherwise the machine's is used.

Based on: an assumed runout until it is measured on the machine.

Tool wear

In 1.0 every tool is a new one: the shape it cuts and its forces are those of a sharp tool, however long it has cut. Counting the life a program uses for each tool is planned for a later version. Tools that cut aluminium are mostly changed first because aluminium sticks to the edge and the part shows burrs or a poorer finish, not because the flank has worn.

The touch probe

The probe is a tool whose ball switches the probe input when it touches the part. The twin works out the touch exactly from the blank's shape and from the cut surfaces, slopes included. Two models: an ideal probe that switches as the ball touches, and a typical probe with pretravel that depends on the direction, three lobes, scatter and the stylus breaking when it is pushed too far.

Based on: typical values for a touch-trigger probe; calibrate the probe on a ring gauge before you measure with it.

The Z clearance

For the setup on the table the twin works out the highest point to clear and, for the tool in the spindle and each tool in the tool table, the tool tip's height at the top of Z. A tool that cannot clear the setup is reported when the setup or the tool changes (finding 188).

Based on: the machine's design and the tools' lengths as given.

Collisions and workholding

The twin checks the tool, its holder and the spindle against the part, the fixture and the machine, and warns before a near collision. The workholding (clamps, vise, V-block, tape, a fixed part) holds the part with a force; a cut that pushes harder can make it slip.

Based on: the machine's and the fixtures' geometry from the design; the holding forces are estimates.

What is not modelled

- the surface's texture and roughness beyond the walls' position;
- built-up edge, chips and their evacuation;
- thin walls bending under the cut;
- the part's own heat;
- what happens after a tool breaks.

The acceptance tests that need a built machine (the drives' and the sound's measurements, the cutting forces, the surface, the heat, the frame's modes, shadow mode with a real machine) are in the twin's design notes (docs/06, section 7).

The Twinwright controller

The Twinwright controller (kernel) is Twinwright's own CNC controller firmware. In Studio it runs the twin's machine. The same code is built for the MC-1's controller board.

It speaks the Grbl 1.1 serial protocol, so G-code senders work with it. Settings, alarms and errors keep the numbers senders know.

What it does

- **Motion:** look-ahead planning over 100 blocks, corner speeds from `$11`, and arcs on the true circle. Motion runs in 1 ms slices. Each axis keeps to its maximum rate (`$110`–`$112`) and acceleration (`$120`–`$122`).
- **Machine functions:**
  - homing, hard and soft limits, probing and the tool setter;
  - the spindle drive (VFD) over Modbus, and coolant;
  - tool changes as `$341` sets them;
  - the safety door with parking;
  - overrides, jogging and the handwheel.
- **Programs on the controller:** programs on its file system run with O-word subroutines and macros (see O-words).
- **G-code:** RS274/NGC and LinuxCNC. This includes a tool table, cutter radius compensation, path blending, expressions and parameters (see G-code and M-code).

The serial protocol, the real-time commands and the `$` commands are in Protocol. Every setting is in Settings, every alarm in Alarm codes, and every error in Error codes.

Compared with other Grbl controllers

Senders written for Grbl work with the controller. This section lists what you notice when you are used to another Grbl controller.

Where the controller stops

In these cases some Grbl controllers carry on; the Twinwright controller stops or raises an alarm.

- **The spindle drive stops by itself** (a trip, an overload, or its own stop key): the feed stops within 0.2 s, and the controller raises `ALARM:14`.
- **The spindle drive does not answer:** the axes slow down to a stop first, so the position is kept. Then the controller raises `ALARM:19`.
- **Tool change:**
  - The spindle always stops for a tool change.
  - If a tool change goes on without a confirmation, the spindle and coolant come back as they were.
- **Resume after a feed hold or the door:** the spindle reaches its speed before the tool moves.
- **The door opens again while the machine returns from parking:** the machine stops and rises to the parking height again. It waits for the next cycle start.
- **The sender is lost** (USB unplugged, the port closed): the job stops. `$1030` controls this; it is on by default.
- **Probe and homing switches:** glitches on the wires do not count as a touch.
- **UTF-8 text in a program** (Chinese, "Ø", "–") is read as text, never as a real-time command.
- **An R-form arc with no chord in its plane**, for example `G91 G2 X0 Z1 R5`, answers `error:33`.

More G-code

Some Grbl controllers refuse or ignore these; the Twinwright controller runs them as the standards define them:

- the tool table (`G43 Hn`, `G43.2`, `G10 L1/L10/L11`);
- cutter radius compensation (`G41`, `G42`);
- path blending (`G64 P`) and exact stop (`G61.1`);
- O-word subroutines and control flow, `#` parameters and expressions;
- `G52`, polar coordinates (`@`, `^`) and `M99`;
- `G85`, `G86` and `G89` drill and bore, where some Grbl controllers accept them and do not move;
- what CAM post-processors often write: a `%` line with a name, `G49 H0`, Fanuc program number lines (`O1000`) and Centroid's `M25`.

Closer to the program

- **Arcs** follow the true circle. Some Grbl controllers cut them into chords up to 4 × `$12` off the circle.
- **Arcs in inverse time** (`G93`) take the programmed time. Some Grbl controllers run them several times faster.
- **Dwells** (`G4`) last as long as programmed. Some Grbl controllers round them to 50 ms steps.
- **Peck drilling:** each hole gets the pecks it needs. When the depth is a whole number of pecks, some Grbl controllers drill one more full peck from the retract height.
- **Large arcs near the soft limits** do not raise false alarms.
- `M72` restores the tool length offset that `M70` or `M73` saved. Some Grbl controllers keep a later offset when the saved one was 0 in the same offset mode (`G43.1`), so after `M72` the same program can run there with another tool length offset.
- **Acceleration:** every axis keeps to `$120`–`$122`, also on small arcs. A program with very many small arcs can take up to about 2 % longer than on some Grbl controllers. An ordinary program takes about 0.1 % less.

Protocol details

- **Status reports:**

    - In a feed hold or with the door open, the feed in `FS:` is 0: the axes stand.
    - The distance to go is reported until the axes stop.
    - During a critical alarm, `Bf:` shows the planner empty.
    - With the parser state on in `$10`, `[GC:]` comes once per change.
- **Input checks:**
    - A bad setting number such as `$45x` answers `error:3`.
    - An integer setting written as `$1=25.0` stores 25; some Grbl controllers store 250.
    - `M6` and a motion in one line answer `error:40`.
    - `G65` or `G66` with other commands in one line answer `error:21`.
- **Alarms and start-up:**
    - There is no 100 ms pause after an alarm.
    - The first start with empty settings storage does not answer `error:7`.
- **Texts and settings:**
    - The descriptions of settings, alarms and errors are Twinwright's own.
    - The controller has settings of its own: `$1030` (stop when the sender is lost), and `$1000` and up for input shaping (off by default).
- `$39=0` (legacy real-time characters off): `?`, `!` and `~` are text inside `$` lines and comments, and commands everywhere else. The controller decides from the bytes as they arrive. So it answers `?` also with the door open or in a feed hold, where some Grbl controllers sometimes do not.

Not supported

- `G65 P7` (Modbus messages).
- Fanuc system variables. `#4120` and the like are ordinary parameters and read 0.
- Threading (`G33`, `G76`, `G33.1`) without a spindle encoder. The MC-1 has none, so these answer `error:20`.
- Lathe roughing cycles (other Grbl controllers have none either).
- Networking, Trinamic drivers and other makes of spindle drive, which some Grbl controllers add as plug-ins.

Senders

The senders that were checked, and their settings, are in G-code senders. To compare a real machine with its twin, see Real machine and shadow mode.

G-code and M-code

The Twinwright controller (kernel) runs the G-code below. The codes follow RS274/NGC and LinuxCNC. A code the controller does not support answers `error:20`, and the line does nothing.

For the real-time commands (single bytes such as `?`, `!` and `~`), see Protocol. For how the controller differs from other Grbl controllers, see The Twinwright controller.

G-code

Code	Meaning	Notes
`G0`, `G1`	Rapid move, linear feed	
`G2`, `G3`	Arc clockwise, counter-clockwise (I J K or R), helix	The tool follows the true circle. An R-form arc with no chord in its plane answers `error:33`.
`G4 P`	Dwell, P seconds	The dwell lasts as long as programmed.
`G5`, `G5.1`	Cubic spline, quadratic spline	A `G5` without I and J continues in the direction the previous move ended.
`G7`, `G8`	Lathe diameter mode, radius mode	Lathe mode only (`$32=2`).
`G10 L2`, `G10 L20`	Set a work offset	
`G10 L1`, `G10 L10`, `G10 L11`	Set the tool table	32 tools. The table is kept when the controller is off.
`G17`, `G18`, `G19`	Select the plane	
`G20`, `G21`	Inch units, millimetre units	
`G28`, `G30`	Go to a stored position through an intermediate point	
`G28.1`, `G30.1`	Store the current position	
`G33`, `G33.1`, `G76`	Spindle-synchronized motion, rigid tapping, threading cycle	These need a spindle encoder. The MC-1 has none, so they answer `error:20`.
`G38.2`–`G38.5`	Probe	The result is the point where the probe triggered, reported as `[PRB:]`. Set zero from that point, not from where the axes stopped.
`G40`	Cutter compensation off	
`G41`, `G42`, `G41.1`, `G42.1`	Cutter radius compensation: D from the tool table, or `.1` with a diameter	G17 plane only. Entry and exit moves as in LinuxCNC. A move that would gouge is refused.
`G43 Hn`, `G43.2 Hn`	Tool length offset from the tool table, added offset	
`G43.1`	Dynamic tool length offset	
`G49`	Tool length offset off	
`G50`, `G51`	Scaling off, on	
`G52`	Local coordinate system	Shares its offset with `G92`, as in LinuxCNC.
`G53`	Machine coordinates for this line	
`G54`–`G59`, `G59.1`–`G59.3`	Work coordinate systems	`$486` can lock `G59.1`–`G59.3`.
`G61`	Exact path	
`G61.1`	Exact stop	The axes stop at the end of every programmed move.
`G64 P Q`	Path blending	Where it saves time, the corner between two straight moves becomes an arc within P of the corner. Q is accepted and not used.
`G65 P1`–`P6`	Built-in macros	Settings, tool offsets, parameters, machine state, probe choice, spindle wait. `P7` (Modbus messages) is not supported.
`G65 P100` and up, `G66`, `G66.1`, `G67`	File macros, modal macro calls	P must be an integer. No other command may share the line (`error:21`).
`G73`, `G81`, `G82`, `G83`	Drilling cycles, peck drilling	Each hole gets the pecks it needs.
`G84`, `G85`, `G86`, `G89`	Tapping and boring cycles	
`G80`	Cycle off	
`G90`, `G91`	Absolute distances, incremental distances	
`G91.1`	Incremental arc centres	
`G92`, `G92.1`, `G92.2`, `G92.3`	Coordinate offset: set, clear, suspend, restore	
`G93`, `G94`, `G95`	Inverse time, units per minute, units per revolution	Arcs in inverse time take the programmed time.
`G96`, `G97`	Constant surface speed, constant spindle speed	Lathe mode only.
`G98`, `G99`	Cycle retract to the start height, to the R plane	

These G-codes are not supported (`error:20`):

- `G5.2`, `G5.3` (NURBS);
- `G17.1`–`G19.1` (UVW planes);
- `G70`–`G72` (lathe cycles);
- `G90.1`.

M-code

Code	Meaning	Notes
`M0`, `M1`	Program pause, optional pause	The `0x88` real-time command switches `M1` pauses on or off.
`M2`, `M30`	Program end	
`M3`, `M4`, `M5`	Spindle clockwise, counter-clockwise, off	
`M6`	Tool change	`$341` sets what M6 does. M6 and a motion in one line answer `error:40`.
`M7`, `M8`, `M9`	Coolant: mist, flood, off	On the MC-1, M8 runs the spindle's water cooling.
`M25`	Z up to the homed height	Centroid's code, written by some CAM post-processors.
`M48`, `M49`	Overrides on, off	
`M50`–`M53`	Feed override, spindle override, adaptive feed, feed hold: on or off	
`M56`	Parking on or off	
`M60`	Pallet change pause	
`M61`	Set the current tool without a change	
`M62`–`M65`	Digital output on or off, with the next move or at once	On boards with auxiliary outputs.
`M66`	Wait for an input	The wait starts when the moves before it have ended.
`M70`–`M73`	Save and restore the modal state	
`M98 P L`, `M99`	Call a subroutine, return	In programs run from the controller's file system. An `M98` sent line by line answers `error:39`. An `M99` in the main program ends it.

These M-codes are not supported (`error:20`):

- `M19` (spindle orientation);
- `M67`, `M68` (analog outputs; the MC-1 has none);
- `M100`–`M199` (user M-codes).

Words, syntax and parameters

Line structure

- Comments `( … )` and `;` are allowed.
- `(MSG,…)` sends the text to the sender as a message.
- `(PRINT,…)` sends a message with the values of the parameters it names.
- `(DEBUG,…)` does the same while `#5599` is on.
- Block delete `/`, line numbers `N` and `%` lines (with or without a name) are allowed.
- Case does not matter, and spaces inside words are ignored.

Parameters and expressions

- Numbered parameters `#1`–`#5000` belong to the program. They read 0 until the program sets them.
- `#5001` and up, and the named system parameters, can only be read.
- Named parameters `#<name>`: reading a name that was never set answers `error:39`. `EXISTS[#<name>]` tells whether it was.
- Expressions in `[ ]` use the operators and functions of LinuxCNC. Angles are in degrees.
- A value that a line sets cannot be read in the same line.

O-words

- Programs run from the controller's file system can use subroutines, `if`, `elseif`, `else`, `while`, `do` and `repeat`.
- Subroutines and loops sent line by line are collected, then run when they are complete.
- A subroutine can also be a file of its own: `<name>.ngc`, or `<name>.macro`.

Polar coordinates

`@` is the radius and `^` the angle, in the XY plane around the work origin.

Errors in a line

- Two codes of the same modal group in one line answer `error:21`.
- Two commands that both use the axis words in one line answer `error:24`.
- The full list of error codes is in Error codes.

CAM strategies and parameters

What each strategy does, what its parameters mean, and how the twin sets the cutting data. The numbers the twin gives are estimates from its physical models.

The parameters below are the ones a feature takes in a job (the `program_generate` tool's features). Lengths are in millimetres, angles in degrees.

Clearing a pocket: rings, zigzag and adaptive

A pocket's area is its shape less its islands, offset inwards by the tool's radius. It is cleared layer by layer, each layer no deeper than the depth of cut (`ap`). The strategy (`strategy`) sets the path in a layer:

Strategy	Path	Use it for
`rings` (default)	Offsets of the area from its deepest spot out to the wall, each `ae` apart	Most pockets
`zigzag`	Straight passes to and fro at `angle`, `ae` apart	Open, shallow areas
`adaptive`	A path that keeps the tool's engagement near `ae` everywhere	Deep pockets, harder materials

A pocket with a fit finishes its walls at size in a pass of its own. See Fits.

Adaptive clearing cuts with a small radial engagement at a deeper `ap`. Its links move clear of the material, and it falls back to rings where the area is too narrow for it.

Rest machining

With `rest: true`, a pocket clears only what the tools before it left standing: the corners and narrow parts a larger tool could not reach. Studio works out what is left from the earlier pockets in the same program, or from the twin's workpiece when it has one.

Contours and tabs

A contour (`side`: `outside`, `inside` or `on`) follows an outline a layer at a time, climb milling. `outside` and `inside` offset the outline by the tool's radius; `on` follows it as it is.

Tabs (`tabs`: count, width and height) leave short bridges at the bottom, so that a part cut out stays in place. A tab is 1 mm high unless you give its height; where the cut is 1 mm deep or less, it is half the depth high.

Slots: layers and trochoidal

A slot (`x0`, `y0`, `x1`, `y1`, `width`) is cut with the strategy:

- `layers` (default): the slot's middle at the full tool width in layers, then out to its width on each side;
- `trochoidal`: small loops along the slot, each `step` further on. The tool is engaged over a short arc only, so it can cut deeper in one pass. The slot must be wider than the tool.

How a tool goes into the material: helix, ramp, plunge

An end mill does not plunge into solid material. Each layer starts with:

- a helix, where the area has room for one (its diameter and angle are given in the answer's `entry`);
- a ramp along the path, where it does not;
- a straight plunge only into a hole or outside the blank.

The answer says, for each operation, how it went in first.

Holes

Drilling cycles

Holes at `points`, `depth` deep, are drilled with the controller's cycles. A hole deeper than three times the drill's diameter is pecked, 1.5 times the diameter a peck, unless `peck` gives another. The cycle's retract plane is 1 mm over the top.

Boring by helical interpolation

A hole of diameter `d` larger than the end mill is cut round a helix, then finished at size at the bottom. The hole can be at most twice the tool's diameter; larger holes are pocketed (`shape: circle`).

Reaming

A reamer (`reamer 6 H7`) of the hole's fit enters at its feed and leaves at the same feed, with no rapid out of the hole. The hole must be drilled before it in the same job, 0.05–0.6 mm smaller, or marked `predrilled`. The reamer's speed is a quarter of the end mill's cutting speed for the material, its feed per revolution 2.5 times the drill's.

Thread milling, tapered pipe threads too

A thread mill goes down the bore, then cuts the thread from the bottom up round a helix of one pitch a turn: right hand counter-clockwise (climb milling), left hand clockwise. The thread is named (`thread`: `M8`, `M8x1`, `1/4-20 UNC`, `G1/4`) or given by `d` and `pitch`. Tapered pipe threads (`1/4 NPT`, `Rc1/4`) are cut round their cone, each turn in 48 arcs.

Chamfers, engraving and V-carving

- Chamfer: a V-bit follows the outline's edge at the depth that gives the `width`. The width is limited by the cone.
- Engrave: the tool follows lines, or text in a single-stroke or an outline font, at one `depth`.
- V-carve: the V-bit follows the shape's medial axis. Its depth at each point gives the width there, so the shape's corners come out sharp. `max_depth` flattens wide areas at that depth.

3D roughing and finishing

A mesh is roughed in levels (`rough3d`), each level cleared adaptively. It is then finished (`finish3d`) with:

Strategy	Path
`parallel`	Passes in one direction across the part
`waterline`	Level contours, for steep walls
`steep_shallow` (default)	Waterline on the steep areas, parallel on the shallow ones
`pencil`	Along the inner corners, after the others

`rest_from` finishes only what an earlier tool left.

Cutting data from the twin

Studio starts from the material's data for the tool's kind: the cutting speed range and the feed per tooth. Values you give in `cutting` (`rpm`, `feed`, `ap`, `ae`, `finish_ap`) are kept as you give them. The other values are set with the twin's models.

Stability and chatter

The twin's chatter model computes the stability lobes for the tool in its holder: the depth of cut that stays stable at each spindle speed. The lobes lie only where the model of the tool and the machine puts them, so Studio does not aim at a lobe's peak: for each speed it takes the least stable depth within 10 % of it, chooses a speed within 0.7 to 1.8 times the one the material gives, and cuts at most 80 % of that depth.

Power, force and the tool's deflection

- Power: the cut takes at most about 80 % of the spindle's power at that speed.
- Force: the cutting force stays under half of what the fixture holds sideways.
- Deflection: a finishing pass is kept light enough that the twin's estimate of the tool's deflection stays small.
- Corners and arcs: the feed on an arc is limited by the acceleration the machine has, so that the controller does not slow down below the feed the cut was planned for. Finishing passes slow down in a pocket's inner corners, where the tool's engagement grows.

The closed loop: trying the program in the twin

With verification on, Studio runs the program on a fresh twin. When the run shows chatter, a load too high or a tool deflecting too much, Studio changes the program and tries again:

- shallower layers,
- another spindle speed, within 0.7 to 1.8 times the one set, where the cut stays stable even if the lobes lie a little off,
- spring passes on a fit's walls.

Each version tried is saved. A long program is tried behind the call, and its report comes later (`run_status`). The answer says what changed and why.

Several setups and nesting

- Setups: Studio writes each setup's program, then checks the setups in order in the twin, each on the workpiece the setups before it left. Where that workpiece differs from the blank, the setup's program is written again for it.
- Nesting: parts are placed on the plate by their outlines, a margin apart. Each part keeps its features, turned and moved with it.

Limits

- Three axes. No fourth axis, no tool tilt.
- STEP files are not read; export STL or 3MF from the CAD program.
- The cutting data and the twin's checks are estimates from physical models of the machine, the tool and the material. A tool or material that differs from its model cuts differently.
- Editions: adaptive clearing, the 3D finishing strategies other than `parallel`, rest machining and several setups are in the Maker edition and up; nesting and the program optimizer are in the Pro edition.

How large a job can be

Past these limits Studio does not write the program. It says why, and what to change.

What	Limit
Setups in a job	8
Nesting	16 kinds of part, 200 copies of each
A mesh (STL, 3MF, OBJ, glTF)	5 million triangles; the file up to 1 GB
An SVG, DXF or text file	64 MB; a DXF drawing up to 2 million segments
Adaptive clearing	The walk's grid holds about 1.5 million cells. A pocket the size of the MC-1's travel clears with a 3 mm end mill; with a 1 mm one, use rings or zigzag, or a larger stepover.

A job can have up to 32000 features; holes alike go as one feature with their points. The operation list, the reasons and the closed loop's repairs cover the first 64 features; a later feature that chatters is reported, not repaired. The setup sheet lists the first 64 operations and says so.

The program check and the setup sheet read up to 2 million moves. An arc counts as its short chords: about 50 for a full circle of 3 mm radius, about 220 for one of 50 mm. Past that, they say where they stopped. The setup sheet still gives the whole program's time, but not the operations' times or the tools' heights after that line. The trial run in the twin checks the whole program.

How long Studio takes to write a program, measured on two cores of a desktop PC (the trial run in the twin comes after):

Job	Time	Memory
20 pockets with islands on a 180 x 130 mm plate	under 1 s	18 MB
A pocket the size of the travel, adaptive, 6 mm end mill	12 s	47 MB
The same with a 3 mm end mill	37 s	70 MB
1000 characters engraved	12 s	25 MB
200 washers nested on a plate	5 s	30 MB
3D roughing of a relief the size of the travel (half a million triangles), 6 mm end mill	9 min	240 MB
The same relief from 3 million triangles	20 min	1 GB
3D parallel finishing of it, 2 mm ball end mill, 0.01 mm scallop (half a million triangles)	6 min	270 MB
The same from 3 million triangles	18 min	1 GB

For 3D work:

- Make the mesh only as fine as the part needs. Finishing takes about as much longer as the mesh has more triangles.
- On a gently undulating surface, finish with `parallel` or `steep_shallow`, not `waterline`. Waterline makes many small loops there: on the relief above it moved more in the air than in the material, and its program had 2.8 million lines.

Fits, runout and inspection methods

How Studio cuts a size to a fit, how it allows for the cutter's runout, and how it measures a part with the touch probe. The twin's numbers here are estimates from its physical models; Studio claims no positioning accuracy for the machine.

Fits (ISO 286)

A size can carry a fit: a hole's diameter (`"d": "Ø12 H7"`), a shaft's (`"fit": "g6"` on an outside contour), a slot's width, or a rectangle's X and Y sizes. Studio takes the fit's limit deviations from the ISO 286-2 tables for the size, or the limit deviations as you write them (`"tol": "+0.015/0"`, in millimetres).

Cutting to the middle of the zone

The wall is roughed with an allowance, then finished in a pass of its own to the middle of the fit's zone. Fits of grade IT6 and finer get a spring pass: the finishing lap again, at size, to take off what the tool's deflection left. The closed loop adds spring passes where the twin's estimate of the deflection asks for them.

The answer says, for each fit, the zone, the diameter it was cut to, and why.

Common fits

Name	Pair	Use
`sliding`	H7/g6	Turns or slides, small clearance
`locating`	H7/h6	The closest clearance fit: put together by hand
`transition`	H7/k6	A little clearance or a little interference: locating, light press
`press`	H7/p6	Interference: pressed in

A pair names both parts: the hole takes the first, the shaft the second.

The cutter's runout

A cutter that runs out cuts a hole larger than itself. The twin models the runout of the tool in its holder as an envelope. For a size with a fit, Studio moves the finishing path inwards for a hole, outwards for a shaft, by that envelope, so that the size the twin estimates is still the middle of the zone. A reamed hole is not changed: it is the reamer's size.

The envelope is the twin's estimate for its model of the tool and holder. Measure the first part.

Measuring with the touch probe

A touch and its compensation

Each touch is a sequence of probing moves (G38):

1. a protected move to near the surface, which stops if the probe triggers on the way;
2. a fast probe to the surface;
3. a move back until the probe releases, then a little further;
4. a slow probe to the surface: only this one's trigger point is used;
5. a move back to where it started.

The trigger point is the ball's centre. Studio adds the ball's effective radius in the touch's direction. The effective radius and the probe's lobing come from its calibration on a ring gauge of a known diameter.

Holes, bosses, walls and floors

- Holes and bosses: four or eight touches round the wall at one depth. Studio fits a circle to the ball's centres by least squares, then adds the effective radius.
- Walls: touches square to the wall, at points along it.
- Floors and tops: touches straight down.
- Positions are given in the program's coordinates, from the work zero.

General tolerances (ISO 2768-1)

A size without its own tolerance takes the general tolerance of the class asked for (f, m, c, v), from ISO 2768-1's table for its size range: linear sizes, and the heights and angles of chamfers.

When the twin's grid is coarse for a tolerance, the twin's estimate for that check is said to be too coarse to judge; the measured value is judged all the same.

Form checks (ISO 2768-2, first step)

The class H, K or L after the ISO 2768-1 class (for example "mK") adds form checks, judged by ISO 2768-2's tables:

Check	Touches	Table
Flatness	A floor on a 3 × 3 grid; the spread of the points from their plane	1
Straightness	A long wall, 3 to 8 points along it; the spread from their line	1
Perpendicularity	A wall at a second depth; its lean over the wall's height, against the machine's Z	2
Circularity	A hole or boss at 8 points; the spread of the radii	4 (the diameter's tolerance)

Two touches on a wall can only be told apart when they are far enough from each other. A wall too short for that, for example under a chamfer of its edge, is listed with the reason instead of a value.

A probe calibrated only on its effective radius has lobing of its own. It shows in the circularity: the report says how large the twin's probe model takes it to be.

Chamfers and V grooves

- A chamfer is touched twice down its slope, at 0.3 and 0.7 of its width from the outline. The line through the two ball centres runs parallel to the slope: it gives the chamfer's width, depth and angle.
- A V groove (an engraving or a V-carving) is touched twice on each flank. The two lines give the groove's depth at its apex, its width at the top, its included angle and its offset from the line it was cut along.
- A thread is not measured. Its crests are the straight bore, and the twin's probe meets only the cells' tops; the answer says so.

What the twin's estimate means

The twin cuts its simulated workpiece with the same program, then runs the same touches on it with the same probe model. Its estimate is what that calculation gives. It shows the program's own effect on each size, as far as the twin models it: the tool's deflection, its runout, the cutting forces.

The estimate is not the machine's measuring uncertainty. The report says nothing about how precisely the machine measures or positions.

Limits

- The probe measures what it touches. A feature narrower than the ball, or a surface the ball cannot reach square, is not measured; the answer says why.
- The twin's workpiece is a grid. Its estimates are only as fine as that grid; where it is too coarse for a tolerance, the twin's estimate is not judged.
- Form checks are a first step of ISO 2768-2: flatness, straightness, perpendicularity and circularity. Symmetry and parallelism are not checked yet.

Machine profiles

Draft (round 14): the text is being written; the sections may still grow.

What a machine profile is

A machine profile is the file the twin builds its machine from: one file per machine model, in JSON. It holds the machine's geometry and its axes, the drives and the ball screws, the spindle and its drive, the sensors and their wiring, the workholding, the cutting data for each material, the tool setter and the touch probe, the sounds, and what the controller is built with. The controller's settings come with it as a settings file.

The profile records where its values come from (its provenance): the design, a data sheet, a published source, an assumption, or a measurement.

Where it comes from

The MC-1's profile is generated from the machine's parametric design: the same parameters that make its drawings and its bill of materials make its profile, so the twin and the machine's design cannot drift apart. The profile keeps checksums of the design files it was made from. A schema checks its form; the reference chapter Machine profile schema lists every key.

Your machine's values

Each MC-1 is to be measured at commissioning and its twin set to the measurements (planned; the procedure and its tools are written and tested on the twin):

- the limit switches: where they switch, their hysteresis and repeatability;
- each axis to ISO 230-2: backlash, the lead error table, the steps per millimetre;
- the cutting coefficients from test slots;
- the spindle's inertia and friction from a coast-down;
- the touch probe on a ring gauge.

The measured values are recorded with the date, the tool that measured them and how; the profile is rebuilt with them, each marked as measured with the design's value kept in its history, and the controller settings they change are sent to the machine.

Several machines

Studio can keep several machines of a model. Each keeps its own world between sessions (where it stands, its workpiece, its tools, its controller's memory) and its own window layout; the machine model's profile is the same for all of them until a machine is measured.

The reference

Every key of the profile, with its type, unit and the MC-1's value: Machine profile schema.

MC-1 hardware

Draft (round 14): the text is being written; the sections may still grow.

The MC-1 as it is designed: the figures in this chapter are design values. Each built machine is to be measured at commissioning, and its twin set to what was measured (see Machine profiles).

The machine

The MC-1 is a desktop three-axis mill for aluminium, brass and engineering plastics. Its gantry is fixed; the table moves in Y, the gantry's carriage in X, the spindle in Z.

Travels X / Y / Z	200 / 150 / 80 mm
Table	300 × 220 × 35 mm, M6 tapped holes on a 25 mm grid
Size, width × depth × height	about 508 × 490 × 564 mm, with motors, cable chains and levelling feet, without the control box
Mass	about 57 kg, without the control box
Frame	6061-T6 aluminium plates, bolted
Linear guides	HIWIN HGR15 rails (6), HGW15CC flange blocks (12)
Ball screws	SFU1204 (12 mm, 4 mm lead), C7, on BK10 / BF10 supports
Motors	NEMA 23 closed-loop steppers, 1.2–1.3 N·m (3), with jaw couplings
Spindle	800 W, water-cooled, Ø65 × 195 mm, ER11 collet
Control box	a separate box beside the machine, on its right

Axes and drives

Each axis is a closed-loop stepper driving its ball screw directly through a jaw coupling. The drives report a fault on one line: the three drives' alarm outputs are wired in series to the controller's motor-fault input, so a drive that faults, or loses its power, stops the machine with an alarm.

The design's fastest moves are 3000 mm/min in X and Y and 1500 mm/min in Z.

Limit switches and homing

Five M8 inductive switches mark the ends of travel: one at each end of X and of Y, each on its own input, and one on the Z slide for both ends of Z. The switches are normally closed: a broken wire or a loose plug reads as a triggered switch, so the controller raises an alarm and refuses to home instead of driving into the end stop.

Homing runs Z first, then X and Y, each towards its positive end (X+, Y+, Z+). The machine must be homed after power-on.

Z and gravity

The Z slide carries the spindle, about 6.35 kg. While the drives are powered they hold Z. When they are not powered (after an E-stop, a drive fault or a power cut), Z with the spindle can creep down.

> ⚠ **Warning**
>
> After an E-stop or a drive fault, home the machine again before you continue. Do not reach under the spindle while the drives are off.

Commissioning includes a test with the drives switched off: if Z creeps, the machine gets a Z motor with a brake, or a counterweight.

Spindle

The spindle is a water-cooled 800 W motor spindle, Ø65 × 195 mm with an ER11 collet, two poles, up to 24 000 rpm. A Huanyang HY01D523B drive (VFD, single-phase 220 V in) runs it; the controller sets its speed and reads its state over RS-485, through an isolating module. A water pump cools the spindle.

- The drive's fault relay is wired into the motor-fault line too: when the drive trips, the controller stops the program with an alarm instead of feeding a stopped tool into the work.
- On an E-stop the drive loses its power with the drives, and the spindle runs down by itself (it is not braked).

Controller and wiring

The controller is a Teensy 4.1 on a T41U5XBB carrier board, with opto-isolated inputs. It runs the Twinwright controller (kernel), compatible with the Grbl 1.1 serial protocol, and it connects to the computer over USB.

Input	What is on it
Limits	X−, X+, Y−, Y+ and Z, five inputs

Input	What is on it
Probe	the tool setter and the touch probe
E-stop	the E-stop button's second contact
Motor fault	the three drives' alarm outputs and the spindle drive's fault relay, in series
Cycle start, Feed hold	the buttons SB2 and SB3 on the control box

Power

Mains goes through a circuit breaker. A 24 V supply is always on: it feeds the controller, the sensors and the control circuit, so the controller stays connected to the computer and reports the E-stop. A contactor (KM1) switches the 48 V supply of the drives, the spindle drive and the pump.

The control box's buttons

Four Ø22 buttons on the control box's door, seen from the front:

	left	right
top row	SB1 power on (green)	SB0 E-stop (red)
bottom row	SB2 cycle start (green)	SB3 feed hold (yellow)

- **SB1 power on** closes KM1, which then holds itself. After an E-stop or a power cut the machine does not power up by itself: release the E-stop and press SB1 again.
- **SB3 feed hold** does what your sender's feed hold (`!`) does: the axes stop under control, the spindle keeps running. It sits under the E-stop, so a hand that misses the E-stop hits feed hold, not cycle start.
- **SB2 cycle start** continues, as your sender's cycle start (`~`) does.

> **Caution**
>
> Feed hold is not a safety function. To stop the machine for safety, use the E-stop.

Tool setter and touch probe

The tool setter sits at the table's left rear, held by two M6 screws in the grid; it takes the corner of the work area at about X 0–57, Y 130–150 mm. The tool setter and a touch probe share the controller's probe input.

With Z at the top, the tip of a tool that sticks out 15.5 mm from the collet is 80 mm above the table, and the spindle's nose 130 mm. A longer tool's tip is lower by its extra length. A touch probe held in an ER collet is usually 60–90 mm long from the nut to the stylus tip: with Z at the top, such a probe clears only low setups, about 20 mm above the table for a 70 mm probe.

Studio checks this for you: when you set up the work, change a tool or edit the tool table, it compares the setup's highest point with each tool's tip at the top of Z and says which tool does not clear and by how much.

Workholding

The table's M6 holes on a 25 mm grid take clamps, a subplate or a vise. The twin knows a 75 mm self-centring vise on a 10 mm aluminium subplate, parallels, strap clamps, double-sided tape and a V-block.

- With the vise on its subplate, the jaws' tops are about 66 mm above the table, so a part held in the vise stands about 71 mm high or more.
- For parts you want to probe, use a low setup: strap clamps on the subplate (the stock's top 15–30 mm above the table), or a low vise. A touch probe does not fit above a part held in the 75 mm vise.
- Taking the 10 mm subplate away gives tools 10 mm more room; for a touch probe it is not enough.

Safety

- **E-stop (SB0)**: it opens the contactor's coil circuit, which switches off the drives, the spindle drive and the pump, and tells the controller through its second contact. The axes and the spindle lose power; the spindle runs down by itself; Z may creep down (see Z and gravity). Releasing the E-stop does not restart anything: press SB1, wait for the drives to start, reset the controller and home.
- **Faults**: a drive fault, a spindle drive trip, or a broken limit switch wire stops the machine with an alarm.
- **Feed hold** is a controlled stop for the work, not for safety.
- The MC-1 has no enclosure. Keep away from the spindle while it turns and while the axes can move, wear eye protection, and fit bellows or brush covers over the X and Z screws against chips.
- None of the machine's safety functions uses AI, and the AI assistant in Studio is not a safety function.

What is measured at commissioning

The design values above are to be checked on each built machine, and its twin set to what was measured:

- each limit switch's trigger point (inductive switches swept from the side can trigger about 2 mm later than drawn), then the homing pull-off and the soft limits;
- the drives' alarm outputs: unplugging one, or switching off the 48 V, must give an alarm;
- the Z test with the drives switched off;
- the spindle drive's parameters, a run-up on the drive's panel without load, then control over RS-485;
- the control box: the E-stop and SB1 as above, every limit switch and the tool setter unplugged one by one (each must give an alarm), and the panel buttons.

How the twin takes the measured values: Machine profiles.

Reference

MCP tool reference

Twinwright Studio offers its twin to AI clients (Claude, ChatGPT, Cursor and others) over the Model Context Protocol. A client starts the program in the background as `TwinwrightStudio.exe --mcp` (the command is shown under AI > Connect AI clients..., The command and configuration) and calls these 31 tools. Each acts on one machine: the one the user sees in the window (`"machine": "window"`, while the window's AI switch is on) or an invisible practice machine (`"sandbox"`). Parameters marked * are required.

Contents

- Machines: `machine_status`, `machine_control`, `machine_mdi`
- Set-ups: `setup_workpiece`, `setup_tool`, `setup_work_zero`, `inspect_part`, `inspect_report`, `setup_sheet`, `probe_cycle`, `setup_part`, `setup_copy`, `setup_example`
- Programs: `program_check`, `program_simulate`, `program_run`, `program_load`, `program_debug`, `program_restart`, `program_optimize`, `geometry_import`, `geometry_nest`, `program_generate`, `program_edit`, `program_versions`
- Runs and results: `run_status`, `run_profile`, `result_compare`, `result_image`, `log_query`
- The licence: `licence_status`

Machines

`machine_status` - Machine status

read-only · the same call twice does no more than once

The state of a machine: controller state and alarm (with its meaning), homed or not, machine and work positions, G54, the tool in the spindle, the spindle, the workpiece on the table and the program's progress - and the other machines, a line each. detail "full" adds the controller's modes, offsets and planner queue; detail "program" gives the loaded program's text in pages. Changes nothing. Tell the user which machine you looked at: only the window's is the one they see.

Parameters

Name	Type	Default	Description
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"
`detail`	string	`"brief"`	(one of `brief`, `full`, `program`)
`offset`	integer	`0`	detail program: the first line (0-based)
`limit`	integer	`400`	

Returns (structured content): `machine`, `notice`, `status`, `machines`, `program`.

`machine_control` - Operate a machine

changes the machine (destructive)

An operator's action on a machine, then its state. home: homes (resets and unlocks first if it is in an alarm; does nothing when already homed - force: true homes again). unlock: $X, when the position is still known. reset: Ctrl-X. hold / resume: feed hold / cycle start. stop: stops the running program. move: to x, y and/or z (mm) in the work (G54) or machine frame, rapid or at feed; moving X/Y it first goes up to the top of Z, where homing left it (safe_z). power_on / power_off: the main switch (and START). estop / estop_release. restart: a practice machine starts afresh (never the window's). On the user's machine the user allows each change in the window; when the window's twin follows a real machine (shadow mode) it takes none of these, stops included - the real machine is stopped at its own controls or in its sender.

Parameters

Name	Type	Default	Description
`action` *	string		(one of `home`, `unlock`, `reset`, `hold`, `resume`, `stop`, `move`, `power_on`, `power_off`, `estop`, `estop_release`, `restart`)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"
`x`	number		
`y`	number		
`z`	number		
`frame`	string	`"work"`	(one of `work`, `machine`)
`feed`	number		mm/min (G1); omitted: rapid (G0)

Name	Type	Default	Description
`safe_z`	boolean	`true`	
`force`	boolean	`false`	
`drive`	string		restart: the drive model (L0 ideal motion, fastest; L1 closed-loop drives, the default; L2 flexible transmission with backlash; L3 electrical) (one of `L0`, `L1`, `L2`, `L3`)

Returns (structured content): `machine`, `notice`, `status`, `action`, `did`, `next`.

`machine_mdi` - Send MDI lines

changes the machine (destructive)

Send G-code or $ lines to a machine as an operator types them: one at a time, each with the controller's answer - ok, or the error and what it means. After a refused line the rest are not sent (the controller would refuse them too) and the result says so. Waits until the machine is idle again and gives its state. For homing, unlocking, stopping or moving, machine_control is simpler.

Parameters

Name	Type	Default	Description
`lines` *	array of string		
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"
`timeout_s`	number	`120`	simulated seconds a line may take

Returns (structured content): `machine`, `notice`, `status`, `lines`.

Set-ups

`setup_workpiece` - Place a blank

changes the machine (destructive)

Put a blank on a machine's table (replacing the one there), take it off (op "remove") or turn it over where it stands (op "flip": the other side up, what was machined stays - for a part's second side; refused while a program runs). Table coordinates in mm: X 0..200 left to right, Y 0..150 front to back, centre [100, 75]. Homes first if needed. The twin refuses setups that do not fit and says why. A thin sheet cut through: fixture tape on a sacrificial plate (sacrificial) - the blank that much higher, the cuts through it go into the plate, not the table.

Parameters

Name	Type	Default	Description
`op`	string	`"place"`	(one of `place`, `remove`, `flip`)
`turn`	string	`"u"`	op flip: u about the blank's u axis (its length as placed, X at angle 0): front and back change places; v about its v axis: left and right change places (the blank also turns 180 degrees about Z) (one of `u`, `v`)
`shape`	string	`"block"`	(one of `block`, `cylinder`, `tube`, `hex`, `rod`, `stl`)
`size`	array of number		block: [x, y, z] mm; cylinder/rod: [diameter, length]; tube: [outer, inner, length]; hex: [across flats, length]
`material`	string	`"6061-T6"`	(one of `6061-T6`, `7075-T6`, `ADC12`, `HPb59-1`, `H59`, `T2`, `Q235`, `45`, `304`, `HT200`, `POM`, `PMMA`, `PA6`, `HDPE`, `MDF`, `hardwood`, `wax`)
`fixture`	string	`"vise"`	(one of `vise`, `clamps`, `tape`, `fixed`, `vblock`, `none`)
`at`	array of number	`[100, 75]`	blank centre on the table [x, y] mm
`angle`	number	`0`	rotation about Z, degrees
`res`	number		simulation cell size in mm (0.01..0.2; smaller is finer and slower)
`file`	string		shape stl: the blank's STL on this computer
`keep_zero`	string		set G54 on the new blank at once, at this corner (as setup_work_zero does); left out, G54 stays where it was (one of `front-left-top`, `center-top`)
`sacrificial`	object		fixture tape only: the blank taped on a sacrificial plate, so that what is cut through goes into the plate, not the table. The blank sits higher by the plate and the tape; the tool may cut into the plate (the summary's setup.sacrificial_cut_mm says how deep), the collet and the spindle may not; the tape holds as on the table
`sacrificial.material`	string	`"MDF"`	a material of the list above (for the drawing and the set-up sheet)
`sacrificial.thickness`	number	`10`	mm, 3 to 25

Name	Type	Default	Description
`sacrificial.size`	array of number		[length, width] mm along the blank's own axes; left out: the blank's footprint and 10 mm each side
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `placed`, `flipped`, `refused`, `reason`, `setup`.

`setup_tool` - Tool and tool table

changes the machine · the same call twice does no more than once

The tool in a machine's spindle (spec) and/or its tool table for programs that change tools with M6 (tools: {"1": spec, ...}; the change and the length measurement then happen by themselves). Tool spec: kind and diameter in mm, then options. Kinds: 'flat D', 'ball D', 'bull D RC' (corner radius), 'v D ANGLE' (V-bit), 'drill D ANGLE', 'tslot D THICKNESS dNECK', 'dovetail D ANGLE dNECK', 'thread D PITCH', 'probe D' (a touch probe with a ball of D; ' typical' a kinematic one: it triggers its pretravel past the touch, scattered with a seed; ' ideal' at the touch, the default). Options: 'nN' flutes (default 2), 'LMM' stick-out from the collet (the length reference tool sticks out 15.5 mm), ' sD lR' a shank of diameter D and the reach R below it, ' tirU' the cutter's runout in the collet, TIR in um (a precision collet's class, or as measured; default: the machine's). Examples: 'flat 6 n3 L20', 'ball 3 n2 L16', 'v 6 60 L18', 'flat 1.5 n2 s4 l5 L14', 'flat 0.2 n2 s4 l1 L15 tir3', 'probe 2 L30 typical'.

Parameters

Name	Type	Default	Description
`spec`	string		e.g. 'flat 6 n3 L20'
`tools`	object of string		the tool table for M6: {"1": "flat 6 n3 L20", ...}
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `tool`, `tool_table`.

`setup_work_zero` - Set G54 on the blank

changes the machine · the same call twice does no more than once

Set a machine's G54 work zero on its blank - the front-left top corner (default: the corner at the machine's front left, X and Y least, when the blank lies square - turned over "v" too) or the top centre, with the tool now in the spindle as the length reference; an optional offset [x, y, z] mm. Homes first if needed.

Parameters

Name	Type	Default	Description
`corner`	string	`"front-left-top"`	(one of `front-left-top`, `center-top`)
`offset`	array of number	`[0, 0, 0]`	
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `g54_mpos`.

`inspect_part` - Inspect the part with the probe

changes the machine

Check the finished part on the machine with the touch probe in the spindle against its design: from the features of a program program_generate made (program; or features as it takes them) the checks - a pocket's sizes, centre and depth, a hole's diameter and position, a slot's width and depth, a faced step, a rectangle's or a circle's contour - each measured by a probe cycle (as probe_cycle runs them: results from the trigger points) and judged against the design (the tolerance given, else ISO 2768-1 - class f unless tol.class says) and against the twin's prediction (the same touches on the workpiece the twin cut: the twin's estimate, not a measuring uncertainty). Depths from a touch on the top where no feature reaches. A chamfer: two touches down onto its face - its angle, its width from the outline and its depth there (its V-bit: the feature's tool, else the job's; the material's side from the job's pocket or contour with that outline). V-carving, engraving, threads and 3D surfaces are said not measured. ISO 2768-2's class after part 1's as a drawing writes it (tol.class "mK") adds form checks and touches: a floor's flatness (a grid of 9 instead of 1), the walls' perpendicularity to Z (4 more at a second depth), a long wall's straightness (5 along it, 20 mm and longer), a circle's circularity (8 round instead of 4) and its axis' perpendicularity (4 more) - the answer's form_cost and the window's question say how many and how long before; only: [..] does part of

them; one letter adds none. Least-squares estimates from a few touches, not ISO 1101's minimum zone nor a CMM's form measurement (summary.form_method). The work offset must be the one the part was cut with. Each inspection is kept as a record (written once, its SHA-256 in the audit log): inspect_report makes its CSV and PDF. plan: true only says what it would do. Professional edition.

Parameters

Name	Type	Default	Description
`program`	string		a program program_generate made: the features of the job kept with it
`version`	integer		that job's version (0: the last)
`features`	array of object		instead of program: the features as program_generate takes them
`tool`	string		with features: the tool they were cut with (a drilling's holes are its size)
`only`	array of integer		only these features (1, 2 ... in the job)
`tol`	object		+/- mm (an angle: degrees) instead of ISO 2768-1
`tol.size`	number		
`tol.depth`	number		
`tol.position`	number		
`tol.angle`	number		a chamfer's, degrees
`tol.class`	string	`"f"`	ISO 2768-1's class where no tolerance is given (a drawing's ISO 2768-mK: m) - with ISO 2768-2's after it (mK: H, K or L) its form checks too, more touches (one of `f`, `m`, `c`, `v`, `fH`, `fK`, `fL`, `mH`, `mK`, `mL`, `cH`, `cK`, `cL`, `vH`, `vK`, `vL`)
`probe`	object		the probe's compensation instead of this machine's calibration: r_eff, lobing, phase_deg, pretravel_z
`plan`	boolean	`false`	only plan it, nothing moves and nothing is recorded: each feature, what it measures there or why not, the touches and about how long, the form checks' share, the classes, the program zero it would measure in (zero_mpos), each touch's point and way (points) - with the spindle's probe, else the tool table's first, else probe 2 L30 (planned_with)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `plan`, `features`, `summary`, `reference`, `cycles`, `twin_cell_mm`, `inspection`, `record`, `sha256`, `record_error`, `error`, `form_cost`.

`inspect_report` - Inspection report

changes the machine · the same call twice does no more than once

The report of an inspection inspect_part recorded, written by the program itself: a CSV (UTF-8, for a spreadsheet) and a PDF - the job, setup, tools, the probe and its ring-gauge calibration, each feature's nominal, tolerance and its basis (ISO 2768-1's class and table, or given), the twin's estimate, the measured value, the deviation and the verdict, why a feature was not measured, and the twin's workpiece from above. Into the reports folder (Documents/Twinwright Studio/Reports); their SHA-256 into the audit log. The same record gives the same bytes again. The twin's estimate is the twin's own calculation for the same touches on its simulated workpiece, not the machine's measuring uncertainty; the report says nothing about how precisely the machine measures or positions. The PDF is in English where the language in use needs letters its fonts lack (the CSV is in it). verify: a file checked instead - its SHA-256 searched for in the audit log and the log's chain checked. It moves nothing. Professional edition.

Parameters

Name	Type	Default	Description
`inspection`	string	`"last"`	the record's name as inspect_part gave it (last: the one recorded last)
`format`	string	`"both"`	(one of `csv`, `pdf`, `both`)
`unit`	string		lengths (default: the units in use) (one of `mm`, `in`)
`paper`	string		the PDF's (default: the region's) (one of `a4`, `letter`)
`verify`	string		a file's path: check it against the audit log instead of writing a report

Returns (structured content): `machine`, `notice`, `inspection`, `record`, `record_sha256`, `csv`, `pdf`, `found`, `chain_whole`, `sha256`.

`setup_sheet` - Setup sheet

changes the machine · the same call twice does no more than once

The setup sheet of a job program_generate made, for the operator at the machine: one HTML page to print (A4 or Letter: the region's) in the language in use - each program's name, lines and SHA-256 (to match it with the one loaded), the blank and its fixture as the twin sets it up (the vise's parallels, the grip), the tools, each setup's way there (turned over, held another way), its work zero and the probe panel's cycles that find it, the program's travel, its Z clearance (the probe and each tool over the set-up and to the program's highest and lowest points - the twin's Z budget: enough, or how much short), the operations (strategy, tool, speed, feed,

depth, the twin's estimate of each one's time), a top view, the notes, a box to sign. The blank as the machine has it set up (the call's machine), else the job's. Into the reports folder (Documents/Twinwright Studio/Reports) as <job>-setup-sheet.html; its SHA-256 into the audit log. The times are the twin's estimates from the program, not measured. It moves nothing. Professional edition.

Parameters

Name	Type	Default	Description
`program` *	string		the name program_generate was given (its job is kept with it; a job in setups: each setup's program)
`version`	integer		the job's version (left out: the last)
`paper`	string		left out: the region's (one of `a4`, `letter`)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `file`, `sha256`, `bytes`, `paper`, `programs`, `z_clearance`.

`probe_cycle` - Touch-probe cycle

changes the machine

Measure with the touch probe in the spindle (setup_tool "probe 2 L30") and, if asked, set the work zero from what it finds: cycle edge (a face of the blank: side x-, x+, y-, y+), corner (front-left, front-right, back-left, back-right; inside: a pocket's corner), bore (its centre and diameter, 4 or 8 points), boss (the same from outside), surface (its Z) or calibrate (a ring gauge of known d: the probe's effective ball radius, its lobes and pretravel - this machine's later cycles here use them). Each touch: a protected move to its start (G38.3), a fast touch, the release (G38.4), a slow touch (F25 at most); the results come from the slow touches' trigger points ([PRB:]), not from where the axes stopped. Left out, at comes from the blank the twin knows. Refused: no probe in the spindle, its input already triggered, the spindle turning, a move outside the soft limits. set {x, y, z}: the active work offset moved so the feature gets these work coordinates (G10 L2); the result's set_parts give each axis found as its G10 L2 word (the feature at work 0, or at set's value). Uncalibrated, the results are off by the probe's pretravel. dry: the plan only - checks (probe, machine, soft limits, safe height, depth: ok, warn or bad), each touch's path (machine coordinates), the program and its time; nothing moves. calibrate keep false: its numbers and a cal_id, kept later with save (this session's measurement only).

Parameters

Name	Type	Default	Description
`cycle` *	string		(one of `edge`, `corner`, `bore`, `boss`, `surface`, `calibrate`)
`side`	string		edge: the face (x- the blank's least X, touched from outside moving +X; y- its front) (one of `x-`, `x+`, `y-`, `y+`)
`corner`	string	`"front-left"`	(one of `front-left`, `front-right`, `back-left`, `back-right`)
`inside`	boolean	`false`	corner: a pocket's corner, its walls touched from inside (at needed)
`at`	array of number		[x, y] in the active work coordinates: a point on the face (edge), the corner, the centre (bore, boss, calibrate), where to touch (surface) - left out: from the blank
`d`	number		bore, boss: the nominal diameter; calibrate: the ring gauge's known diameter, mm (left out: a tube's or cylinder blank's)
`depth`	number	`3`	how far below the top the ball's centre touches, mm
`top`	number		the work Z the depth is from (left out: the blank's top)
`points`	integer		edge 1, or 2 (span apart: the face's turn too), 3, 4 or 8 over span (its straightness too: about their least-squares line); corner 2, or 3 (two on its X face: the blank's turn and where its faces meet); bore, boss 4 (default) or 8 (its roundness too) (one of `1`, `2`, `3`, `4`, `8`)
`span`	number		mm between the first and the last touch on one face (left out: 0.6 of the face)
`grid`	array of integer		surface: [nx, ny] touches over size (25 at most): the plane through them - its tilt and flatness
`size`	array of number		surface grid: [w, h] mm about at (left out: 0.6 of the blank)
`inset`	number	`5`	corner: mm from the corner along each face
`clear`	number	`3`	mm between the probe and where the face should be when a touch starts
`search`	number	`5`	mm past where it should be a touch looks
`fast`	number	`300`	
`slow`	number	`20`	(at most 25)
`set`	object		move the active work offset so the feature found gets these work coordinates (the axes the cycle finds)
`set.x`	number		
`set.y`	number		
`set.z`	number		

Name	Type	Default	Description
`probe`	object		the probe's compensation instead of this machine's last calibration here: r_eff, lobing, phase_deg, pretravel_z (mm, degrees)
`dry`	boolean	`false`	the plan only: the checks, the path, the program and its time - nothing moves, nobody is asked
`keep`	boolean	`true`	calibrate: keep the calibration on the machine (false: its numbers and a cal_id only)
`save`	string		calibrate: keep the calibration this session measured with keep false (its cal_id) - nothing measured
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `cycle`, `touches`, `result`, `work`, `set`, `comp`, `archive`, `notes`, `program`, `sent`, `error`, `dry`, `checks`, `plan`, `seconds`, `set_parts`, `cal_id`.

`setup_part` - Set the design part

changes the machine · the same call twice does no more than once

The design part (an STL of the finished part, mm) to compare a machine's workpiece with. origin: machine mm of the part's origin; omitted: the program zero (as CAM writes it). An empty stl takes it away.

Parameters

Name	Type	Default	Description
`stl` *	string		
`origin`	array of number		
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `part`.

`setup_copy` - Copy a setup

changes the machine (destructive) · the same call twice does no more than once

Copy one machine's setup - blank and fixture, the tool in the spindle, the tool table, G54, the design part - onto an invisible practice machine, to try programs there at full speed (default: from the user's window to "sandbox"). The workpiece comes as it is there, cuts and all (stock: "fresh" for a new blank).

Parameters

Name	Type	Default	Description
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`from`	string		the machine to copy (default: the user's window)
`to`	string	`"sandbox"`	a practice machine: sandbox or sandbox-<name>

Returns (structured content): `machine`, `notice`, `copied_from`, `workpiece`, `tool`.

`setup_example` - Example parts

changes the machine (destructive)

Without id: the example parts that come with the twin (id, title, material, blank, summary). With id: a machine set up as the example's operator would - blank, tool table, homing, G54 on the blank, the design part - with the example's program loaded, not started. To verify an example without setting anything up: program_simulate {example: id}.

Parameters

Name	Type	Default	Description
`id`	string		
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `examples`, `example`, `ready`.

Programs

`program_check` - Check a program

read-only · the same call twice does no more than once

Check a program without running it: the controller's own check mode ($C) - every line it refuses, with the error explained - the preview of its path against a machine's setup (estimated time, collisions, rapids into the stock, travel) and the static rules S01-S20 (start block, end, F and S missing or out of range, dense short lines, plunges, rapids below Z0, the water-cooled spindle without M8, arcs in inverse time, G33 without a spindle encoder, characters the controller takes for real-time commands, drilling cycles' holes a single-precision controller plunges once more, Z down after a tool change before X and Y, ...) with a suggested line for each. Runs on a hidden copy of the machine's setup, its workpiece as it is (stock): the machine does not change.

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text - or leave it out and give name (and version) of a program in the working folder (program_versions)
`name`	string	`"program.nc"`	
`version`	integer		with name, no gcode: that saved version's text (0: the last)
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`example`	string		an example's setup (and program, when gcode is omitted) instead of the machine's: see setup_example
`setup`	object		an explicit setup instead of the machine's: {spec: the workpiece's setup as machine_status shows it (status.workpiece), tool, tools: {T: spec}, g54: [x,y,z] MPos, part: STL path}
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `ok`, `refused`, `preview`, `static_rules`.

`program_simulate` - Simulate a program

read-only · the same call twice does no more than once

Run a program to its end on a hidden copy of a machine's setup - the machine itself does not change; the copy starts from the machine's workpiece as it is, cuts and all (stock: "fresh" for a new blank) - as fast as the computer goes: moves hundreds of times faster than real time, cutting a few times faster (a heavy cut can be slower than real time). The report: completed or not, machine time, alarms, collisions, near misses, rapids into material, cuts with the spindle stopped, workpiece slip, chatter margin, force and power peaks, removed volume, deviation from the design part, the slowest lines - and a run id for run_profile, result_compare and result_image. Programs with M6 need tools. Longer than budget_s: the answer is the run id (run_status waits).

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text - or leave it out and give name (and version) of a program in the working folder (program_versions)
`name`	string	`"program.nc"`	
`version`	integer		with name, no gcode: that saved version's text (0: the last)
`tools`	object of string		the tool table for M6: {"1": "flat 6 n3 L20", ...}
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`example`	string		an example's setup (and program, when gcode is omitted) instead of the machine's: see setup_example
`setup`	object		an explicit setup instead of the machine's: {spec: the workpiece's setup as machine_status shows it (status.workpiece), tool, tools: {T: spec}, g54: [x,y,z] MPos, part: STL path}
`check_first`	boolean	`true`	
`timeout_s`	number	`3600`	simulated seconds allowed
`budget_s`	number	`40`	wall-clock seconds this call waits before it answers with a run id (at most 45: clients give up after 60 s; run_status waits for the rest)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `run`, `completed`, `simulated_time_s`, `issues`, `status`.

`program_run` - Run a program on the machine

changes the machine (destructive)

Run a program on the machine itself - the one loaded, or gcode (it replaces the loaded one): on the user's machine in real time, after the user allows it in the window; on a practice machine (sandbox) at full speed, cutting its workpiece. Checked first, the loaded program too (check_first): not run when the controller refuses a line or the check ends in an alarm - and on the user's machine not when the check's preview finds a collision, a rapid move into material, a cut with the spindle stopped or a move out of travel. Same report as program_simulate. To find out first what a program would do, use program_simulate.

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text - or leave it out and give name (and version) of a program in the working folder (program_versions)
`name`	string	`"program.nc"`	
`version`	integer		with name, no gcode: that saved version's text (0: the last)
`tools`	object of string		the tool table for M6: {"1": "flat 6 n3 L20", ...}
`check_first`	boolean	`true`	
`fits_only`	boolean	`false`	nothing run, nothing moved: only the twin's grid made finer where the touch probe will measure the fits of the saved program name (its CAM job), as a run does before it starts - for a program started some other way (the window's own start); the answer's fit_regions says how many regions, where (program mm) and their cells, or why none (no job, no fits, several setups). Before the cut, never after: what is cut already stays as coarse as it was
`timeout_s`	number	`3600`	simulated seconds allowed
`budget_s`	number	`40`	wall-clock seconds this call waits before it answers with a run id (at most 45: clients give up after 60 s; run_status waits for the rest)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `run`, `completed`, `simulated_time_s`, `issues`, `status`.

`program_load` - Load a program

changes the machine · the same call twice does no more than once

Load a program into a machine without starting it. On the user's machine the window shows how it differs from the program loaded there and the user allows it; they may then start it themselves.

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text - or leave it out and give name (and version) of a program in the working folder (program_versions)
`name`	string	`"program.nc"`	
`version`	integer		with name, no gcode: that saved version's text (0: the last)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `lines`.

`program_debug` - Debug a program

read-only

Debug a program on a hidden copy of a machine's setup. run_to: runs gcode from its start until a file line (line) or a condition (when: force_n, load_pct, ferr_mm, margin, near_mm as limits; cut, rapid_cut, collision, alarm, state true; modal ["units","wcs","tool","spindle",...]) and stops virtual time there - mid-motion, exactly as the program has it, at the same instant every time. Then inspect (the line, modes, positions, speed, the cut, the planner's queue), step (until: time with ms, block, or pause), continue (to the next pause or the end), end.

Parameters

Name	Type	Default	Description
`action` *	string		(one of `run_to`, `inspect`, `step`, `continue`, `end`)
`gcode`	string		the program's text
`name`	string	`"program.nc"`	
`line`	integer		file line (1-based)
`when`	object		conditions, e.g. {"force_n": 60, "cut": true}

Name	Type	Default	Description
`tools`	object of string		the tool table for M6: {"1": "flat 6 n3 L20", ...}
`until`	string	`"time"`	(one of `time`, `block`, `pause`)
`ms`	number	`10`	
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`example`	string		an example's setup (and program, when gcode is omitted) instead of the machine's: see setup_example
`setup`	object		an explicit setup instead of the machine's: {spec: the workpiece's setup as machine_status shows it (status.workpiece), tool, tools: {T: spec}, g54: [x,y,z] MPos, part: STL path}
`timeout_s`	number	`3600`	simulated seconds allowed
`budget_s`	number	`40`	wall-clock seconds this call waits before it answers with a run id (at most 45: clients give up after 60 s; run_status waits for the rest)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `reached`, `pause`, `where`.

`program_restart` - Restart from a line

changes the machine (destructive)

The program loaded on a machine from a file line on, as an operator restarts after a broken tool: a start block from the program's own state before that line (units, plane, work offset, tool, spindle speed and direction with its spin-up, coolant, over the start point at the safe height, down to it, feed, distance mode), then the rest. start: true loads and runs it (on the user's machine after the user allows it).

Parameters

Name	Type	Default	Description
`line` *	integer		
`start`	boolean	`false`	
`safe_z`	number		machine Z to go up to first (default 2 mm below home)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `line`, `started`, `program`.

`program_optimize` - Optimize a program

changes the machine · the same call twice does no more than once

Make a program faster without asking more of any tool than the program already does: it runs on a hidden copy of the setup; lighter cuts get feeds up to the heaviest cut the same tool already makes (or the limits in goals), moves in air go at the axes' top speed, finishing passes and plunges keep theirs, a chattering tool gets the steadiest speed near its own (0.7 to 1.8 times it, the depth within 10 % of it the deepest) where that takes the cut half as deep again, collinear lines are merged, rapid retracts come down to just over the blank and the fixture, arcs are fitted where the sender could not keep up. The result runs again on a second copy (lines over a limit are lowered and it runs once more). Answers with the program, its changes and the comparison; saved as a version (author optimizer) when it is better in every respect. Takes two or three runs of the program: a run id comes back after budget_s.

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text
`name`	string	`"program.nc"`	
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`example`	string		an example's setup (and program, when gcode is omitted) instead of the machine's: see setup_example
`setup`	object		an explicit setup instead of the machine's: {spec: the workpiece's setup as machine_status shows it (status.workpiece), tool, tools: {T: spec}, g54: [x,y,z] MPos, part: STL path}
`goals`	object		
`goals.feeds`	boolean	`true`	
`goals.speeds`	boolean	`true`	

Name	Type	Default	Description
`goals.merge`	boolean	`true`	
`goals.retracts`	boolean	`true`	
`goals.arcs`	boolean		default: when the sender could not keep up
`goals.chip_mm`	number		
`goals.force_n`	number		
`goals.load_pct`	number		
`goals.load_cap`	number	`80`	
`goals.feed_max`	number		
`goals.arc_acc`	number		mm/s2 towards an arc's centre: its feed no more than 60 sqrt(arc_acc r) - the controller's (the MC-1's kernel: sqrt(3)/2 of the axes' 300 = 260); default 0, no bound
`goals.air_feed`	number		
`goals.finish_ae`	number	`0.15`	
`timeout_s`	number	`3600`	simulated seconds allowed
`budget_s`	number	`40`	wall-clock seconds this call waits before it answers with a run id (at most 45: clients give up after 60 s; run_status waits for the rest)
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `optimized`, `verdict`, `gcode`, `status`, `run`.

`geometry_import` - Import a shape

changes the machine · the same call twice does no more than once

Make a shape the programs can cut, from what you write - SVG path data (svg_path, its numbers in mm: e.g. a heart "M 10,30 A 20,20 0,0,1 50,30 A 20,20 0,0,1 90,30 Q 90,60 50,90 Q 10,60 10,30 z"), an SVG or DXF drawing (svg, dxf: the text) or file (an .svg or .dxf path; a DXF's layers: [names] reads only those), formulas in t (curve: {x: "16*sin(t)^3", y: "13*cos(t)-5*cos(2*t)-2*cos(3*t)-cos(4*t)", t: [0, "2*pi"]}), basic shapes added or cut in order (shapes: rect, circle, ellipse, slot, polygon, star, path), single-stroke text (text, font, text_height, align) or points - instead of working out its points yourself. Then where it goes: width_mm or height_mm (scaled to fit), scale, rotate, mirror, and center [x, y] (its box's middle) or at [x, y] (its lower left corner) in the program's mm (Z0 on the blank's top). SVG's y points down: it is turned the right way up. Saved under name as a version (the same shape again: not saved again); the answer says its size and box, outlines and islands, area, the sharp inner corners a pocket's tool cannot get into, the largest tool that fits and the narrowest place, its symmetry - with a picture: look at it before cutting. Then program_generate {features: [{op: "pocket", geometry: name, depth}]} (or contour, engrave). A mesh (file: .stl, .obj, .3mf, .glb, .gltf; units, up, width_mm or scale, depth_mm, rotate, mirror, center or at, top) is saved the same way for 3D: rough3d and finish3d take it. With only name: a saved shape again (version: an older one); with nothing: the shapes there are. Units are mm; a shape is data - it does not touch a machine.

Parameters

Name	Type	Default	Description
`name`	string		the shape's name: saved under it, used by it (geometry: name)
`version`	integer		with only name: that saved version (default the last)
`svg_path`	string		SVG path data: M L H V C S Q T A Z, lower case relative
`svg`	string		an SVG document's text
`dxf`	string		a DXF drawing's text (ASCII): lines, arcs, circles, polylines, splines, ellipses, blocks
`file`	string		an .svg or .dxf file's path - or a mesh for 3D: .stl, .obj, .3mf, .glb, .gltf
`layers`	array or string		DXF: only these layers (default: all that are on)
`join`	number	`0.01`	DXF: mm within which ends are joined into closed outlines
`curve`	object		formulas in t (radians; sin cos tan sqrt abs exp ln pow pi ...)
`curve.x` *	string		
`curve.y` *	string		
`curve.t`	array of ['number', 'string']		
`curve.closed`	boolean	`true`	
`shapes`	array of object		rect {x,y,w,h[,r]}, circle {cx,cy,d}, ellipse {cx,cy,rx,ry}, slot {x0,y0,x1,y1,w}, polygon {points} or {cx,cy,n,d}, star {cx,cy,n,d,d_inner}, path {svg}; op: add, cut, intersect

Name	Type	Default	Description
`text`	string		text: single strokes to engrave (font simplex, duplex, script: Latin letters, digits, signs) or the letters' outlines to pocket or V-carve (an outline font)
`font`	string		simplex (default), duplex, script: single strokes; sans (Noto Sans, comes with Studio) or a font of the computer's by its name - "微软雅黑", "宋体", "Arial" ... (Chinese: a computer's font): outlines
`text_height`	number		the capitals' height, mm (10)
`align`	string		(one of `left`, `center`, `right`)
`points`	array of array		a polygon: [[x, y], ...]
`units`	string	`"mm"`	svg_path's numbers; a DXF's that does not say its own; an STL's or OBJ's (mm, cm, m, in) (one of `mm`, `cm`, `m`, `in`, `pt`, `px`)
`up`	string	`"z"`	a mesh: which of its file's axes points up (most OBJ: y) (one of `z`, `-z`, `y`, `-y`, `x`, `-x`)
`depth_mm`	number		a mesh: its Z size (Z alone: a relief made shallower)
`top`	number	`0`	a mesh: the Z its highest point goes to
`bottom`	number		a mesh: the Z its lowest point goes to (instead of top)
`px_per_in`	number	`96`	
`flip_y`	boolean		
`scale`	number		
`width_mm`	number		
`height_mm`	number		
`mirror`	string		(one of `x`, `y`)
`rotate`	number		degrees, counter-clockwise
`center`	array of number		
`at`	array of number		
`tolerance`	number	`0.005`	mm: curves that are not circles become lines this close
`fill_rule`	string	`"evenodd"`	(one of `evenodd`, `nonzero`)

Returns (structured content): `machine`, `notice`, `name`, `version`, `size_mm`, `bbox`, `shapes`.

`geometry_nest` - Lay parts out on the blank

read-only · the same call twice does no more than once

Lay several parts out on the machine's blank (or a plate given) and cut them out - nesting. Each part a shape: geometry (a name from geometry_import) or one given as geometry_import takes it (svg_path, dxf with layers, shapes ...); count copies of it, each turned by quarter turns where that fits them better (turns: 0, 90, 180, 270; default all), its own features (holes, pockets ...) written in its shape's coordinates. They go in the largest first, each where it fits lowest, then leftmost - the shapes, not their boxes: a small part in a larger one's notch or its hole (in_holes) - gap apart (default the tool's diameter + 1 mm) and margin (1) inside the plate (default the blank's top). Nothing is machined: the answer says where each copy went, how much of the plate the parts use, the copies that did not fit (placed less than asked), with a picture; and features - each copy's own where it went, then every copy cut out, those in holes first (a contour outside its shape, depth deep - default through the blank and 0.2 - with tabs, 3 unless given) - for program_generate as they are, with nest (kept with the job).

Parameters

Name	Type	Default	Description
`parts` *	array of object		the parts, 1 to 16, e.g. [{geometry: "bracket", count: 4}, {shapes: [{type: "circle", cx: 0, cy: 0, d: 20}], count: 3}]
`plate`	array of number		[x0, y0, x1, y1] where the parts go, the program's mm (default the blank's top)
`depth`	number		how deep each part is cut out (mm; default the blank's thickness + 0.2)
`gap`	number		mm between two parts (default the tool's diameter + 1)
`margin`	number	`1`	mm from the plate's edge to the parts' outlines
`tabs`	integer or object		the cut-outs' tabs: a count or {count, width, height, holes: false - none round the parts' holes} (default 3)
`tool`	string		the tool the gap is for (default the one in the spindle)
`setup`	object		the setup to measure the blank and the tool on instead of the machine's, as program_generate takes it (spec, tool, tools, g54): a hidden copy set up so
`in_holes`	boolean	`true`	a small part may go into a larger one's hole (cut out before it); false: the holes taken as filled
`outlines`	boolean	`false`	each placed copy's outline where it went and each part's own: {outer: [[x, y], ...], holes: [...]}; each copy's tabs where the program will leave them: [[x, y, nx, ny], ...] on its outline, the normal away from its material - for drawing the layout

Name	Type	Default	Description
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `placed`, `parts`, `used_pct`, `features`, `nest`.

`program_generate` - Generate a program from features

changes the machine · the same call twice does no more than once

Write a program from machining features instead of line by line: face, pocket, slot, contour (outside, inside or on an outline; tabs hold a part cut through), drill, bore (helix), ream (a reamer widening holes drilled before to its size and class: drill, then ream), engrave (single-stroke text, or a shape's lines), chamfer (a V-bit along edges), thread (a thread mill in a bored hole), vcarve (a V-bit's tip along an outline's middle lines, each point as deep as the V reaches: lettering in an outline font, any closed shape). A pocket or a contour takes any outline, islands and all: a shape made with geometry_import (geometry: its name), SVG path data right in the feature (svg_path with geometry_import's width_mm, center ...), a rectangle with corner radius, a circle, polygon points, or the closed outline another program cuts (from_program). E.g. {op: "pocket", geometry: "heart", depth: 4}; {op: "pocket", svg_path: "M 10,30 A 20,20 0,0,1 50,30 A 20,20 0,0,1 90,30 Q 90,60 50,90 Q 10,60 10,30 z", width_mm: 40, center: [30, 30], depth: 4}; {op: "pocket", from_program: "heart.nc", offset: 3, depth: 13} (a hollow where heart.nc cut the heart's outline; offset: grown by the tool's radius). A pocket is cleared in rings from its deepest spot out - or, strategy "adaptive", by a walk that cuts a set width (ae, 10 % of the tool) all the way: deeper layers, a faster feed for the thinner chips, no slotting but the entry; or "zigzag", straight passes to and fro - its walls, round the outside and every island, finished to size (a slot: strategy "trochoidal", loops along it); the answer says where the tool cannot reach (corners narrower than its radius stay round). Speeds, feeds and depths come from the blank's material and the tool, within the spindle's range: with verify, the twin's stability model picks the speed and keeps each depth (roughing steps, the walls' finishing passes) to 80 % of the stable one before the program is written. Ramps and helices into material, climb milling. The clearance height stays within what the setup leaves above the blank. With verify (default) it runs on a hidden copy of the setup, its workpiece as it is (stock), and where the tool would still chatter it is written again shallower there - a program too long to try within the call likewise behind it (verification.check_run: run_status gives the last version's report, closed_loop what it changed). Saved as a version (author generator; each one the loop tried too). operations says each feature's lines, layers and values - the first 64; past them (a nest's) feature_spans gives every feature's [first line, last line, layers, tool]. A hole's or a shaft's diameter (a circle's pocket or contour, a bore, a reaming) may carry its fit (ISO 286): d "Ø8 H7", or d 8 with fit "H7", a pair "H7/g6" (the hole takes H7, the shaft g6) or a common fit (sliding, locating, transition, press), or tol "+0.015/0" - cut to the middle of its zone, a spring pass where the twin's estimate of the tool's give asks for one (operations[].fit says what and why: the twin's estimate). So may a slot's width (width "6 N9", or width 6 with fit "N9": a keyway's N9, P9, JS9 - cut with a cutter narrower than it, its walls to size) and a rectangle's pocket or contour, both its sizes (fit "H7"; outside: a shaft's class, a key's h9). The program's zero: the machine's G54 (setup_work_zero). A part machined from two sides or more: setups instead of features - a program each, setup 1 on the machine as it is, each later one turned over (turn) or held another way (fixture) from the one before, G54 set again on the blank (zero); the job and every program saved together.

Parameters

Name	Type	Default	Description
`stock`	string	`"current"`	current: the machine's workpiece as it is, cuts and all (carry on with a part already cut); fresh: a new blank (a whole program from the start) (one of `current`, `fresh`)
`features`	array of object		each {op, ...}: face {x0,y0,x1,y1,depth}; pocket {geometry name | svg_path (+ width_mm, center) | shape rect x0,y0,x1,y1[,r] | shape circle cx,cy,d | shape polygon points [[x,y],..] | from_program name[,version]; [offset]; [strategy adaptive[, ae] | zigzag[, angle]]; [rest true]; depth}; slot {x0,y0,x1,y1[,width],depth[,strategy trochoidal[,step]]}; contour {the same outlines, side outside|inside|on, depth}; drill {points [[x,y]..],depth[,peck]}; bore {points,d,depth}; engrave {text,x,y,height,depth[,font] | geometry name, depth}; chamfer {an outline, width} (a V-bit); thread {points,d,pitch,depth[,hand]} (a thread mill); vcarve {an outline, or text with font sans or a computer's font, text_height, center[, max_depth]} (a V-bit); rough3d {geometry: a mesh's name[, allowance, stepdown, ae, area]} (a flat end mill); finish3d {geometry[, strategy, scallop or stepover, angle, steep_angle, area, rest_from]} (a ball end mill; strategy pencil: along the creases); contour tabs: n or {count,width,height,holes}
`setups`	array of object		instead of features: a part machined from two sides or more, a program each, in order. Setup 1 is the machine's setup as it is; each later one turned over (turn) or held another way (fixture) from the one before, G54 set again on the blank (zero). Within the call each later program is written for the blank as it was (what the setups before cut is not known yet). name, tool, material, cutting and safe_z here are every setup's defaults. E.g. setups: [{features: [{op: "pocket", shape: "rect", x0: 8, y0: 8, x1: 24, y1: 22, depth: 3}]}, {turn: "u", features: [{op: "face", x0: 0, y0: 0, x1: 40, y1: 30, depth: 1}]}]
`nest`	object		geometry_nest's nest (its parts laid out on a plate): kept with the job beside the features it made; given without features, the parts are laid out here (plate, depth and the parts needed) - not with setups
`tool`	string		default: the machine's tool in the spindle
`material`	string		default: the machine's blank's; a code of the machine's table (6061-T6, 45, 304 ...) or another standard's name for one (EN AW-6061, C45, SUS304 ...: read as that code, the nearest grade - for its cutting data)
`cutting`	object		rpm, feed, ap (depth of cut), ae (stepover), finish_ap to use instead
`setup`	object		the setup to generate for and verify on instead of the machine's (as program_optimize takes it)
`safe_z`	number	`5`	the clearance height over Z0; kept within what the setup leaves above the blank
`name`	string	`"generated.nc"`	
`title`	string		the job's name to show (one line, 80 characters at most; kept with the job, not in the program)
`description`	string		a line about the job (200 characters at most; kept with the job)
`verify`	boolean	`true`	

Name	Type	Default	Description
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `gcode`, `verification`, `operations`, `setups`, `total`.

`program_edit` - Change a program by intent

changes the machine · the same call twice does no more than once

Change a program without sending it again line by line: edits applied in order - set_s value (every S), set_f value or scale (every F), shift_z by and scale_z factor (the cuts: Z at or below the program's zero; all: every Z), split into (from..to: one pass cut in that many layers, a ramp kept a ramp - back up and over to its start in between), safe_z z (the clearance height: every Z above zero); from/to: file lines (default all). G53, G28, G30, G10 and G92 lines are left alone; programs in absolute mm (G90 G21). The program: gcode, or name (and version: default the last) from the working folder. The result is saved as a new version (save false: not) with the difference; check it with program_check {name, version}.

Parameters

Name	Type	Default	Description
`gcode`	string		the program's text (or name and version)
`name`	string		
`version`	integer		with name, no gcode: that saved version (0: the last)
`edits` *	array of object		each {op, ...}: set_s {value}; set_f {value | scale}; shift_z {by[, all]}; scale_z {factor[, all]}; split {from, to, into}; safe_z {z}; any of them with from, to (file lines)
`save`	boolean	`true`	

Returns (structured content): `machine`, `notice`, `gcode`, `version`, `changed_lines`, `diff`.

`program_versions` - Program versions

changes the machine · the same call twice does no more than once

Program versions in the working folder (Documents/Twinwright Studio/programs/<name>/vNNNN.nc with who, when and why). action list: the versions (without name: every program, the last changed first); get: a version's text (default the last); save: gcode as a new version (reason: what changed and why; the same text is not saved twice); diff: how b differs from a - texts, or name with a_version and b_version - with their simulation results side by side when both ran on this host. The user's own files are never written.

Parameters

Name	Type	Default	Description
`action`	string	`"list"`	(one of `list`, `get`, `save`, `diff`)
`name`	string		
`version`	integer		
`gcode`	string		
`reason`	string		
`a`	string		
`b`	string		
`a_version`	integer		
`b_version`	integer		

Returns (structured content): `machine`, `notice`, `name`, `version`, `versions`, `gcode`, `programs`.

Runs and results

`run_status` - Runs

changes the machine

Without run: the runs kept on this host (the last 16) - program, machine, who started it, running or done, a summary. With run: waits up to wait_s for it (at most 45 s: clients give up after 60 s) and gives its full report - how it ended first (done, stopped by whom, an alarm, out of time) and at which line; stop: true ends it early. A call that is cancelled or times out never stops a run.

Parameters

Name	Type	Default	Description
`run`	string		a run id (program_simulate's, program_run's), or "latest": your newest run
`wait_s`	number	`40`	at most 45
`stop`	boolean	`false`	

Returns (structured content): `machine`, `notice`, `run`, `completed`, `simulated_time_s`, `issues`, `status`, `runs`.

`run_profile` - Where a run's time went

read-only · the same call twice does no more than once

A run line by line (a run id from program_simulate or program_run; default the latest): the time each line took (they add up to the run), the feed asked for and reached and why it fell short (the axes' top speed, corners, arcs, the planner's look-ahead, a sender that could not keep up, the override, a hold), the cutting force, chip, chatter margin, spindle load, material removed, cutting air, events. sort: time (slowest first), risk (events first) or line; from/to: a range of file lines; paged (offset, limit).

Parameters

Name	Type	Default	Description
`run`	string		a run id (program_simulate's, program_run's), or "latest": your newest run
`sort`	string	`"time"`	(one of `time`, `risk`, `line`)
`limit`	integer	`20`	
`offset`	integer	`0`	
`from`	integer		
`to`	integer		
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `run`, `total_s`, `lines`.

`result_compare` - Compare with the design part

read-only · the same call twice does no more than once

How a machined workpiece deviates from its design part: cells within tolerance, material left above the part, cut into it. run: the workpiece a run left; else the machine's workpiece now.

Parameters

Name	Type	Default	Description
`run`	string		a run id (program_simulate's, program_run's), or "latest": your newest run
`tol`	number	`0.02`	
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `comparison`, `in_tolerance_pct`.

`result_image` - Picture of a result

read-only · the same call twice does no more than once

A picture (PNG) and its key numbers in words - kind: workpiece = the workpiece in 3D (view iso, top, front, right; color material, height, line = the program line that cut there last, deviation; path: a run's toolpath over it); toolpath = the path coloured by feed, force, chip, margin, line, z or limit (why the programmed feed was not reached), rapids dashed, the highest value marked with its line (view top or iso); deviation = the design part's deviation map from above, the worst places marked with their program lines (needs a part); profile = the slowest lines, coloured by what limited them; timeline = cutting force, spindle load, feed, chatter margin and the program line over time; comparison = two runs side by side (runs). run: a run of program_simulate - its end state; without it, workpiece and deviation show the machine now and the others its latest simulation (a simulation runs on a copy: to see its result, name its run). A picture larger than about 100 KB is drawn smaller (clients limit a result's size). The key numbers come in the text too.

Parameters

Name	Type	Default	Description
`kind`	string	`"workpiece"`	(one of `workpiece`, `toolpath`, `deviation`, `profile`, `timeline`, `comparison`)

Name	Type	Default	Description
`run`	string		a run id (program_simulate's, program_run's), or "latest": your newest run
`runs`	array of string		comparison: two run ids, A then B
`view`	string		(one of `iso`, `top`, `front`, `right`)
`color`	string		workpiece: material, height, line, deviation; toolpath: feed, force, chip, margin, line, z, limit
`path`	boolean	`false`	
`top`	integer	`15`	profile: lines shown, at most 40
`tol`	number	`0.02`	deviation: the tolerance band, mm
`width`	integer	`960`	
`height`	integer	`640`	
`machine`	string		which machine. Leave it out for the machine the user sees and means - the one in their Twinwright Studio window ("window"; needs the window open with its AI switch on). "sandbox" (or "sandbox-<name>") is an invisible practice machine of yours, faster than real time (far faster moving, a few times while cutting), shared by the user's AI clients: the user does not see it - use it only to try something out, and say so when you report. No other names. Left out while the window's AI switch is off: "sandbox"

Returns (structured content): `machine`, `notice`, `image`, `kind`, `of`.

`log_query` - Audit log

read-only · the same call twice does no more than once

The audit log: who did what, when, to which machine - the user in the window, the AI clients, the machines themselves (alarms, homing, programs started and ended, collisions), the twin host. Newest first; filters: since / until (a time's start, e.g. "2026-09-29T17:5"), category (machine, operator, program, setup, ai, system), action (a prefix: "machine.alarm", "ai.tool"), who (user, ai, system, or a name), machine, text, severity (warn, error); paged. Useful to see what the user did in the window, or what became of a machine.

Parameters

Name	Type	Default	Description
`since`	string		
`until`	string		
`category`	string		(one of `machine`, `operator`, `program`, `setup`, `ai`, `system`)
`action`	string		
`who`	string		
`machine`	string		
`text`	string		
`severity`	string		(one of `warn`, `error`)
`limit`	integer	`30`	
`offset`	integer	`0`	

Returns (structured content): `machine`, `notice`, `records`, `matching`.

The licence

`licence_status` - Licence and edition

read-only · the same call twice does no more than once

This copy's edition and licence: the edition in force (in_force: learning, maker, pro, shop; classroom, trial), the licence files in the computer's licence folder - each one's kind, holder, seats, dates (valid_until, renew_after, updates_until), days left (a subscription past its period: its days of grace) and whether it holds, and why not - the local trials used, whether a bought licence's updates cover this release, every paid feature with the edition it needs, why it is closed and whether this release has it at all (shipped false: not built yet - never offer it), and the site's pages for the next step (pricing: buy or upgrade; licence: a year more of updates, managing a subscription, offline activation). Changes nothing; gives no activation code and no computer fingerprint. Use it to explain a refusal ("edition: ... comes with the pro edition") and what the user can do.

No parameters.

Returns (structured content): `edition`, `in_force`, `files`, `trials_used`, `features`, `links`.

Settings reference

Every setting of the controller. `$<n>` shows one, `$<n>=<value>` changes it, and `$$` lists them all. The values given are the MC-1's. A setting marked *restart* takes effect after the controller restarts.

Contents

- General
- Control signals
- Limits
- Coolant
- Spindle
- Tool change
- Homing
- Probing
- Parking/Safety door
- Jogging
- ModBus
- Stepper
- VFD
- X-axis
- Y-axis
- Z-axis

General

Setting	Name	Unit	Type	Range	MC-1	Description
`$10`	Status report options		bit field		15871	What the real-time status report contains. The parser state is sent as its own line, only when it changed. Bits: 0 Position in machine coordinate, 1 Buffer state, 2 Line numbers, 3 Feed & speed, 4 Pin state, 5 Work coordinate offset, 6 Overrides, 7 Probe coordinates, 8 Buffer sync on WCO change, 9 Parser state, 10 Alarm substatus, 11 Run substatus, 12 Enable when homing, 13 Distance-to-go
`$11`	Junction deviation	mm	decimal		0.002	How far the path may deviate at a corner between two moves; it sets the corner speed. Smaller is slower and more exact.
`$12`	Arc tolerance	mm	decimal		0.002	How exactly arcs are followed.
`$13`	Report in inches		boolean		0	Positions and rates in reports are in inches.
`$28`	G73 Retract distance	mm	decimal		0.100	The short retract between the pecks of G73.
`$32`	Mode of operation		choice		0	Mill, laser or lathe. Choices: 0 Normal, 1 Laser mode, 2 Lathe mode
`$39`	Enable legacy RT commands		boolean		1	'?', '!' and '~' act as real-time commands everywhere, also inside $ commands and comments.
`$60`	Restore overrides		boolean		0	Overrides go back to 100 % at the end of a program.
`$62`	Sleep enable		boolean		0	Go to sleep after a long idle time.
`$63`	Feed hold actions		bit field		3	What a feed hold does besides stopping the motion. Bits: 0 Disable laser during hold, 1 Restore spindle and coolant state on resume, 2 Set RPM to minimum (except for laser M4)
`$64`	Force init alarm		boolean		0	Start in the alarm state after power-on.
`$384`	Disable G92 persistence		boolean		0	G92 offsets are not kept over a power cycle.
`$398`	Planner buffer blocks *(restart)*		integer	30–1000	100	Moves the planner looks ahead over.
`$481`	Autoreport interval *(restart)*	ms	integer	100–1000	0	Send status reports at this interval without being asked (0: off).
`$484`	Unlock required after E-Stop		boolean		1	After an emergency stop the machine stays locked until unlocked ($X) or homed.
`$486`	Lock coordinate systems		bit field		0	Coordinate systems that G10 may not change. Bits: 0 G59.1, 1 G59.2, 2 G59.3
`$676`	Reset actions		bit field		15	What a reset clears. Bits: 0 Clear homed status if position was lost, 1 Clear offsets (except G92), 2 Clear rapids override, 3 Clear feed override
`$1030`	Stop the job when the host is lost		boolean		1	When the link of the sender that runs a job goes down for 0.1 s (USB unplugged, the port closed), the job stops as at a stop (0x19): the motion slows down to a stop, spindle and coolant go off, the lines it sent that have not run are dropped, the position is kept. The sender is told once it is back. A program from the file system runs on without its sender.

Control signals

Setting	Name	Unit	Type	Range	MC-1	Description
$14	Invert control inputs		bit field		0	Control inputs that are inverted. Bits: 1 Feed hold, 2 Cycle start, 6 EStop, 8 Motor fault
$17	Pullup disable control inputs		bit field		0	Control inputs without the internal pull-up resistor. Bits: 1 Feed hold, 2 Cycle start, 6 EStop, 8 Motor fault

Limits

Setting	Name	Unit	Type	Range	MC-1	Description
$5	Invert limit inputs		axis mask		7	Limit inputs that are inverted, e.g. for normally-closed switches.
$18	Pullup disable limit inputs		axis mask		0	Limit inputs without the internal pull-up resistor.
$20	Soft limits enable		boolean		1	Refuse motion beyond the travel ($130 ...). Needs homing, so that the position is known.
$21	Hard limits enable		bit field		1	Stop at once when a limit switch triggers. Strict mode: no motion at all while a switch is active. Bits: 0 Enable, 1 Strict mode (bit 0 enables the rest)

Coolant

Setting	Name	Unit	Type	Range	MC-1	Description
$15	Invert coolant outputs		bit field		0	Coolant outputs that are active low. Bits: 0 Flood, 1 Mist
$673	Coolant on delay	s	decimal	0.5–20	0.0	Time the coolant gets to start after M7/M8.

Spindle

Setting	Name	Unit	Type	Range	MC-1	Description
$30	Maximum spindle speed	RPM	decimal		24000.000	The highest spindle speed.
$31	Minimum spindle speed	RPM	decimal		6000.000	The lowest spindle speed.
$340	Spindle at speed tolerance	percent	decimal		5.0	Motion after M3/M4 waits until the spindle speed is within this many percent of the programmed one (0: no wait).
$394	Spindle on delay	s	decimal	0.5–60	15.0	Time the spindle gets to come up to speed after M3/M4 (with $340: the longest wait before an alarm).
$539	Spindle off delay	s	decimal	0.5–60	0.0	Time the spindle gets to stop after M5.

Tool change

Setting	Name	Unit	Type	Range	MC-1	Description
$341	Tool change mode		choice		3	What M6 does. Choices: 0 Normal, 1 Manual touch off, 2 Manual touch off @ G59.3, 3 Automatic touch off @ G59.3, 4 Ignore M6
$342	Tool change probing distance	mm	decimal		50.0	How far the tool setter is searched for.
$343	Tool change locate feed rate	mm/min	decimal		25.0	The slow rate that locates the tool setter exactly.
$344	Tool change search seek rate	mm/min	decimal		200.0	The fast rate that searches for the tool setter.
$345	Tool change probe pull-off rate	mm/min	decimal		200.0	The rate that moves off the tool setter.
$346	Tool change options		bit field		0	Options of the tool change. Bits: 0 Restore position after M6, 1 Change tool at G30, 2 Fast probe pull off
$485	Keep tool number over reboot		boolean		0	The number of the tool in the spindle is kept over a power cycle.

Homing

Setting	Name	Unit	Type	Range	MC-1	Description
$22	Homing cycle		bit field		541	The homing cycle and its options. Bits: 0 Enable, 1 Enable single axis commands, 2 Homing on startup required, 3 Set machine origin to 0, 4 Two switches shares one input, 5 Allow manual, 6 Override locks, 9 Per axis feedrates, 10 Run startup scripts only on homing completed (bit 0 enables the rest)
$23	Homing direction invert		axis mask		0	Axes that home towards their negative end.
$26	Homing switch debounce delay	milliseconds	integer		250	Time to wait after a switch triggered before the next homing move.
$27	Homing switch pull-off distance	mm	decimal		2.000	How far homing moves back off the switch.
$43	Homing passes		integer	1–128	1	How many times the switch point is located.
$44	Axes homing, first phase		axis mask		4	Axes homed first.
$45	Axes homing, second phase		axis mask		3	Axes homed second.
$46	Axes homing, third phase		axis mask		0	Axes homed third.

Probing

Setting	Name	Unit	Type	Range	MC-1	Description
$6	Invert probe inputs		bit field		1	Inverts the probe input, e.g. for a normally-closed tool setter. Bits: 0 Probe
$19	Pullup disable probe inputs		bit field		0	The probe input without the internal pull-up resistor. Bits: 0 Probe
$65	Probing options		bit field		0	How probing motion is handled. Bits: 0 Allow feed override, 1 Apply soft limits

Parking/Safety door

Setting	Name	Unit	Type	Range	MC-1	Description
$41	Parking cycle		bit field		0	Retract the spindle when the safety door opens. Bits: 0 Enable, 1 Deactivate upon init, 2 Enable parking override control (bit 0 enables the rest)
$42	Parking axis		choice		2	The axis that parks. Choices: 0 X, 1 Y, 2 Z
$56	Parking pull-out distance	mm	decimal		5.0	How far the tool is first pulled out slowly.
$57	Parking pull-out rate	mm/min	decimal		100.0	The rate of that pull-out.
$58	Parking target	mm	decimal	≥ -100000	-5.0	Where the parking axis goes, in machine coordinates.
$59	Parking fast rate	mm/min	decimal		500.0	The rate of the rest of the parking move.
$61	Safety door options		bit field		0	What stays on while the safety door is open. Bits: 1 Keep coolant state on door open
$392	Spindle on delay	s	decimal	0.5–60	4.0	Time the spindle gets to come up to speed when the door closes again.
$393	Coolant on delay	s	decimal	0.5–20	1.0	Time the coolant gets to start when the door closes again.

Jogging

Setting	Name	Unit	Type	Range	MC-1	Description
$40	Limit jog commands		boolean		0	Jogs stop at the soft limits instead of being refused.

ModBus

Setting	Name	Unit	Type	Range	MC-1	Description
$374	ModBus baud rate		choice		3	Baud rate of the Modbus line (the VFD's must match). Choices: 0 2400, 1 4800, 2 9600, 3 19200, 4 38400, 5 115200
$375	ModBus RX timeout	milliseconds	integer	50–250	50	How long to wait for a Modbus answer.

Setting	Name	Unit	Type	Range	MC-1	Description
$681	ModBus serial format		choice		0	Parity of the Modbus line (the VFD's must match). Choices: 0 8-bit no parity, 1 8-bit even parity, 2 8-bit odd parity

Stepper

Setting	Name	Unit	Type	Range	MC-1	Description
$0	Step pulse time	microseconds	decimal	≥ 2.0	5.0	Length of the step pulses, in microseconds. The drives need a minimum (see their manual; MC-1's closed-loop drives need 2.5).
$1	Step idle delay	milliseconds	integer	≤ 65535	255	How long the drives stay enabled after motion stops, so the axes settle before the current is switched off. 255 keeps them enabled.
$2	Step pulse invert		axis mask		0	Step outputs that are active low.
$3	Step direction invert		axis mask		5	Axes whose direction output is inverted (the axis runs the wrong way otherwise).
$4	Invert stepper enable output(s)		axis mask		7	Drive enable outputs that are active low (most drives). With one shared enable signal only X counts.
$29	Pulse delay	microseconds	decimal	≤ 20	5.0	Time between setting the direction outputs and the step pulse.
$37	Steppers to keep enabled		axis mask		7	Drives that stay enabled when the machine is idle or in an alarm.
$680	Stepper enable delay	ms	integer	≤ 500	0	Time between enabling the drives and the first step.

VFD

Setting	Name	Unit	Type	Range	MC-1	Description
$460	VFD spindle ModBus address		integer	≤ 255	1	The Modbus address of the VFD.

X-axis

Setting	Name	Unit	Type	Range	MC-1	Description
$100	X-axis travel resolution	step/mm	decimal		800.00000	Steps per millimetre of the axis: steps per motor turn times microsteps, divided by the screw lead.
$110	X-axis maximum rate	mm/min	decimal		3000.000	The highest speed of the axis, also for rapids.
$120	X-axis acceleration	mm/sec^2	decimal		300.000	The acceleration of the axis.
$130	X-axis maximum travel	mm	decimal		202.000	The travel of the axis from its home position, for the soft limits.
$180	X-axis homing locate feed rate	mm/min	decimal		50.0	The slow rate that locates the switch point of the axis exactly.
$190	X-axis homing search seek rate	mm/min	decimal		1200.0	The fast rate that searches for the switch of the axis.
$1000	X-axis input shaper		choice		0	Shapes the axis' motion so that it does not excite a resonance of the machine at the frequency below: ZV is exact at it, the others take out a wider band less exactly. The motion comes to rest up to 1.5 periods later. Choices: 0 None, 1 ZV, 2 ZVD, 3 MZV, 4 EI, 5 2HUMP_EI
$1010	X-axis input shaper frequency	Hz	decimal	15–500	100.0	The resonance the axis' input shaper takes out.
$1020	X-axis input shaper damping ratio		decimal	≤ 0.5	0.100	The damping ratio of that resonance.

Y-axis

Setting	Name	Unit	Type	Range	MC-1	Description
$101	Y-axis travel resolution	step/mm	decimal		800.00000	Steps per millimetre of the axis: steps per motor turn times microsteps, divided by the screw lead.
$111	Y-axis maximum rate	mm/min	decimal		3000.000	The highest speed of the axis, also for rapids.
$121	Y-axis acceleration	mm/sec^2	decimal		300.000	The acceleration of the axis.
$131	Y-axis maximum travel	mm	decimal		152.000	The travel of the axis from its home position, for the soft limits.

Setting	Name	Unit	Type	Range	MC-1	Description
$181	Y-axis homing locate feed rate	mm/min	decimal		50.0	The slow rate that locates the switch point of the axis exactly.
$191	Y-axis homing search seek rate	mm/min	decimal		1200.0	The fast rate that searches for the switch of the axis.
$1001	Y-axis input shaper		choice		0	Shapes the axis' motion so that it does not excite a resonance of the machine at the frequency below: ZV is exact at it, the others take out a wider band less exactly. The motion comes to rest up to 1.5 periods later. Choices: 0 None, 1 ZV, 2 ZVD, 3 MZV, 4 EI, 5 2HUMP_EI
$1011	Y-axis input shaper frequency	Hz	decimal	15–500	100.0	The resonance the axis' input shaper takes out.
$1021	Y-axis input shaper damping ratio		decimal	≤ 0.5	0.100	The damping ratio of that resonance.

Z-axis

Setting	Name	Unit	Type	Range	MC-1	Description
$102	Z-axis travel resolution	step/mm	decimal		800.00000	Steps per millimetre of the axis: steps per motor turn times microsteps, divided by the screw lead.
$112	Z-axis maximum rate	mm/min	decimal		1500.000	The highest speed of the axis, also for rapids.
$122	Z-axis acceleration	mm/sec^2	decimal		200.000	The acceleration of the axis.
$132	Z-axis maximum travel	mm	decimal		82.000	The travel of the axis from its home position, for the soft limits.
$182	Z-axis homing locate feed rate	mm/min	decimal		50.0	The slow rate that locates the switch point of the axis exactly.
$192	Z-axis homing search seek rate	mm/min	decimal		600.0	The fast rate that searches for the switch of the axis.
$1002	Z-axis input shaper		choice		0	Shapes the axis' motion so that it does not excite a resonance of the machine at the frequency below: ZV is exact at it, the others take out a wider band less exactly. The motion comes to rest up to 1.5 periods later. Choices: 0 None, 1 ZV, 2 ZVD, 3 MZV, 4 EI, 5 2HUMP_EI
$1012	Z-axis input shaper frequency	Hz	decimal	15–500	100.0	The resonance the axis' input shaper takes out.
$1022	Z-axis input shaper damping ratio		decimal	≤ 0.5	0.100	The damping ratio of that resonance.

Alarm codes

The controller's alarms. An alarm stops the machine and locks it until you clear it: `$X` or homing, as the alarm says. `$EA` lists them on the controller itself.

Code	Meaning
`ALARM:1`	Hard limit: a limit switch was hit during motion. The position is probably lost; home again
`ALARM:2`	Soft limit: the motion target is outside the machine's travel. The position is kept; you may unlock
`ALARM:3`	Reset or emergency stop during motion. The position is probably lost; home again
`ALARM:4`	Probe: the probe was not in the expected state when the probing cycle started
`ALARM:5`	Probe: no contact within the programmed travel
`ALARM:6`	Homing: the homing cycle was reset
`ALARM:7`	Homing: the safety door was opened during homing
`ALARM:8`	Homing: the pull-off did not clear the limit switch. Check the pull-off distance ($27) and the wiring
`ALARM:9`	Homing: no limit switch found within the search distance. Check the travel settings and the wiring
`ALARM:10`	Emergency stop. Release it, then reset
`ALARM:11`	Homing required: run the homing cycle ($H)
`ALARM:12`	A limit switch is active. Clear it before going on
`ALARM:13`	Probe protection: the probe was triggered outside a probing cycle
`ALARM:14`	The spindle did not reach its speed in time
`ALARM:15`	Homing: the second switch of an auto-squared axis was not found
`ALARM:16`	The power-on self test failed
`ALARM:17`	A motor drive reports a fault
`ALARM:18`	Homing: the homing settings are not valid
`ALARM:19`	Modbus: no answer or a bad answer from a device (e.g. the VFD)
`ALARM:20`	The I/O expander does not answer
`ALARM:21`	The non-volatile storage (settings memory) failed
`ALARM:22`	A buffer overflowed

Error codes

The answer `error:N` to a line the controller refused. The line did nothing. `$EE` lists them on the controller itself.

Code	Meaning
`error:1`	A G-code word has no letter: every word is a letter followed by a number
`error:2`	A number is missing or is not a valid number
`error:3`	Unknown or unsupported '$' command
`error:4`	A negative value was given where only positive values are allowed
`error:5`	Homing is not enabled in the settings ($22)
`error:6`	The step pulse must be at least 2 microseconds long
`error:7`	Stored settings could not be read; the affected settings were set to their defaults
`error:8`	This '$' command is only accepted while the machine is idle
`error:9`	G-code is not accepted during an alarm or while jogging
`error:10`	Soft limits need homing: enable homing ($22) first
`error:11`	The line was longer than the controller accepts and was not executed
`error:12`	The value would make the step rate higher than the controller can produce
`error:13`	The safety door is open
`error:14`	The text is too long to be stored (startup line or build info)
`error:15`	The jog target is outside the machine's travel; the jog was not done
`error:16`	A jog command needs '=' and may only contain jog words
`error:17`	Laser mode needs a PWM spindle output
`error:18`	The controller was reset
`error:19`	The value must be greater than zero
`error:20`	The block holds an unsupported or invalid G-code command
`error:21`	Two commands of the same modal group in one block
`error:22`	No feed rate has been set
`error:23`	This command needs a whole number
`error:24`	Two commands in the block both want the axis words
`error:25`	A word appears twice in the block
`error:26`	The command needs axis words and the block has none
`error:27`	Invalid line number
`error:28`	A value word the command needs is missing
`error:29`	G59.1, G59.2 and G59.3 are not supported
`error:30`	G53 only works with G0 and G1
`error:31`	The block has axis words that no command uses
`error:32`	An arc (G2/G3) needs at least one axis word in its plane
`error:33`	The motion target is invalid
`error:34`	The arc radius is invalid
`error:35`	An arc (G2/G3) needs at least one offset word (I, J or K) in its plane
`error:36`	The block has value words that no command uses
`error:37`	G43.1 needs the configured tool length axis
`error:38`	The tool number is higher than supported, or no such tool
`error:39`	A value is out of range
`error:40`	Not allowed while a tool change is waiting
`error:41`	The spindle must run for this motion (CSS or spindle-synchronized)
`error:42`	Threading needs the ZX plane (G18)
`error:43`	The feed rate is higher than the maximum
`error:44`	The spindle speed is out of range
`error:45`	A limit switch is active: only homing is allowed

Code	Meaning
`error:46`	Home the machine first ($H)
`error:47`	The current tool is not known: set it with M61
`error:48`	Conflicting value words
`error:49`	The power-on self test failed: switch the controller off and on
`error:50`	The emergency stop is active
`error:51`	A motor drive reports a fault
`error:52`	The setting value is out of range
`error:53`	No such setting on this controller
`error:54`	The retract position is below the drilling depth
`error:55`	Two auto-squared axes cannot be homed together
`error:56`	This coordinate system is locked ($486)
`error:57`	Unexpected program start/end mark (%)
`error:58`	The port is not available
`error:60`	The SD card could not be mounted
`error:62`	The directory could not be listed
`error:63`	No such directory
`error:64`	The SD card is not mounted
`error:65`	No file system is mounted (no card?)
`error:66`	The file system is read only
`error:71`	Expression: unknown operator or function
`error:72`	Expression: division by zero
`error:73`	Expression: argument out of range
`error:74`	Expression: invalid argument
`error:75`	Expression: syntax error
`error:76`	Expression: the result is no number
`error:77`	Authentication required
`error:78`	Access denied
`error:79`	Not allowed while a critical event is active
`error:80`	This O-word statement needs a program the controller can read again (from its file system)
`error:81`	Unknown or misplaced O-word statement
`error:82`	Too many O-word blocks or calls open at once
`error:83`	No room left for the O-word statement
`error:84`	The file could not be opened
`error:85`	The file system could not be formatted
`error:86`	The port cannot be used
`error:87`	There is a tool in the spindle
`error:88`	There is no tool in the spindle
`error:89`	The file could not be deleted
`error:253`	User-defined error

Command line: mc1sim

The options of the twin's program `mc1sim`, as its usage text gives them (85 options).

```
usage: mc1sim [options] scenario.nc ...     (batch, deterministic)
       mc1sim [--tcp PORT] [--ws PORT] [options]  (real time: TCP sender, browser UI)
       mc1sim --replay FILE [options]         (replay a recorded real-time session)
       mc1sim --play FILE.mcap [--ws PORT | --gui] (play a --mcap recording in the UI, with a timeline)
       mc1sim                                 (single-file build: the machine in its own window,
                                               a sender port on 2323 and the web view on 8765)
```

option	what it does
`--machine FILE`	machine archive (default: built in, or twin/machine/mc1.machine.json)
`--max-time S`	stop after S seconds of virtual time (default 3600 in batch, none in real time)
`--fg-quantum US`	longest foreground wait between resumes (default 50)
`--link KIND`	the sender's line: usb_cdc (the Teensy's native USB), uart (through a USB-serial adapter) or telnet (Ethernet); default the board's (machine archive controller.board)
`--baud N`	a UART at N baud (implies --link uart)
`--link-param K=V`	change a number of the link, e.g. adapter_timer_ms=1 or host_in_latency_us=0; host_dtr=0: the twin itself does not hold DTR (the native USB answers only once a host raises it), connect_dtr=0: a TCP sender does not raise it either
`--fg-cost off|L,P,S`	the foreground's CPU time in us: a line parsed, a block planned, a step segment prepared (default the board's; off: none)
`--pos X,Y,Z`	power-on position, machine mm (default from the archive)
`--sw-delay US`	limit sensors' response time (default from the archive)
`--tool-offset MM`	tool tip below the reference tool (default 0)
`--drive L0|L1|L2|L3 feed drives: L0 ideal steps, L1 closed-loop steppers moving rigid axes,`	L2 rotor, screw and table apart (coupling, torsion, Hertzian nut, play, ringing), L3 the motors' two phases and the drives' current loops on rigid axes
`--l3-dyno [V]`	the L3 motor on a dynamometer: torque against speed, as JSON (V: another supply)
`--lobes SPEC`	a side-milling cut's stability lobes by three methods (zero-order, linear time domain, semi-discretisation), JSON: "tool=flat 6 n3 L20;mat=6061-T6;ae=1;ap=1;dir=x; rpm=6000:24000:500;m=100;methods=zoa,td,sdm"
`--chatter-sim SPEC`	the cut in time, nonlinear (each tooth out of the cut where its chip is gone): the vibration and whether it chatters, JSON: "...;ap=2;rpm=15000;fz=0.02;t=0.5" (default: the machine archive's drives.model)
`--geom on|off`	the G- and T-layer: geometric errors, lead error, backlash, thermal offsets (machine archive "accuracy"; default off)
`--fidelity batch|live`	what feeds back into the simulation (docs/06 section 8.3): batch - the full resolution, as tests and the AI's practice machines have it; live - what a virtual machine at 1x keeps up with (the cut's path merged to an eighth of a cell). Default live with the window, --ws or --tcp and no --lockstep, else batch; a replay or a branch takes its recording's
`--threads N`	threads for the cut's sweep (the results are the same with any number): 1 this one alone; default the processors less one, at most 8 (or TWINWRIGHT_THREADS)
`--chatter-async on|off`	the chatter check on a thread of its own, below the simulation's: its answers taken in the order the cuts came - the same as at once, later (the summary waits for them; the views show the tool's last one meanwhile). Default on where the machine keeps the wall clock's pace (the window, --ws, --tcp; not --lockstep)
`--set PATH=VALUE`	change a machine archive entry as it is read, e.g. drives.motor.detent_nm=0.01 or drives.alm.wired=false (repeatable)
`--mpg N`	the handwheel UART as if the firmware were built with MPG_ENABLE N (2: the handwheel's own 0x8B takes control; 0: none) - default as the archive's firmware options say (a replay: as its log says)
`--mpg-port PORT`	an external handwheel on the handwheel UART's cable, over TCP (with --mpg 2): its bytes reach the controller at the handwheel's baud rate (logged), the controller's come back
`--panel SPEC`	how the operator's buttons are wired: cycle_start=pin|sender|mpg, feed_hold=... (pin: the board's input, Pn: and its debounce; sender: '~' '!' from the sender; mpg: on the handwheel's stream) - default as the archive's controller.panel, else the sender's (a replay: as its log says)
`--fault WIRE=MODE`	wiring fault from power-on: x_min x_max ... z_switch setter alm estop estop_km1 rs485 (the VFD's bus) vfd_relay (its fault relay in the ALM chain) = open|short|intermittent (wire names follow the machine archive)
`--seed N`	sensors with their scatter (repeatability, response jitter) drawn from seed N; without: nominal sensors. Intermittent wires use it too
`--nvs FILE`	load EEPROM image; --save-nvs FILE writes it at exit. Batch runs start blank; real-time and replay runs from the MC-1 settings image (built in, or build/mc1.nvs made by make) unless --nvs FILE or --nvs blank

option	what it does
`--world DIR`	the persistent world (docs/06 8.5): the machine as the last session left it - where the axes stood, the workpiece clamped and cut, the tool, the EEPROM (written through to DIR/eeprom.bin), the temperatures and the board's clock moved on by the time it was off, the odometers; saved there at the end. The window keeps one in its state folder; batch runs and replays only with this
`--world-reset`	start the world from the factory state (its files removed first)
`--world-off S`	as if the machine had been off S seconds (else the wall clock's time since)
`--predict FILE`	with --shadow: the program the operator streams; each run of it the real machine starts is predicted by a background twin set up like the machine (homed, the workpiece, tool and offsets), flat out - its own run time against the machine's
`--nvs-through FILE`	the controller's EEPROM kept in FILE, as the board's flash keeps it: loaded from FILE if it exists (else from the image above, and FILE made), every write landing in FILE at once - settings, offsets and tools carry over to the next session. Not in replays (a recording has its start image beside it: LOG.nvs)
`--params-trace FILE the machine archive's values the run used (JSON): where each sits, its`	provenance (source, uncertainty), what it feeds, whether --set changed it
`--steps FILE`	write every step pulse (tick step_bits dir_pins)
`--spindle-trace FILE`	each millisecond the spindle turns: seconds, its angle (revolutions) and where the steps put the axes (machine mm) - a thread's pitch and phase (src/spenc.h)
`--hil-log FILE`	write what crosses between the controller and the machine as the HIL emulator's frames (src/hil.h): step groups, enables, coolant, RS-485, the machine's answers
`--hil-replay FILE`	the machine driven by such a log, the controller off (HIL's follower)
`--wav FILE`	the run's sound as the machine makes it (src/sound.h): 48 kHz stereo, 32-bit float; the same run gives the same file
`--sound-player FILE,FACTOR`	the window's player of the sound at speed FACTOR, without a device: the live frames played in this thread into a WAV (tests: the pitch at any factor)
`--sound-analyse FILE[,DBFS_AT_94[,T0,T1]]`	a WAV measured as a sound level meter reads it: LZeq, LAeq, 1/3-octave bands, the strongest lines (JSON; default -20: 94 dB SPL is -20 dBFS rms)
`--sound-listener X,Y,Z`	where --wav and --sound-player listen: model mm at the GLB pose (the machine archive's "sound"), facing the tool; default the archive's operator in front
`--sound-room off|small|workshop`	the room --wav and --sound-player hear it in (RT60 0.35 s, 0.7 s; default off: the dry sound)
`--events FILE`	write inputs, output lines and markers with ticks
`--echo`	print controller output with virtual time
`--factor F`	real-time factor (default 1)
`--www DIR`	serve the UI from DIR (default: built in, or micro_cnc/)
`--bind local|any`	listen on localhost only (the default) or on every interface (in shadow mode the sender port passes to the real machine: any lets the network move it; a twin in WSL reached from Windows needs any)
`--open`	open the web UI in the default browser
`--gui`	the native window (Windows; the default when started without arguments)
`--lang TAG`	the language (en, zh-CN, a pack's tag; default: the settings, else the system's; qps-ploc: the pseudo-language that shows untranslated texts and tight layouts)
`--lang-dir DIR`	one more folder with language packs (.po); see twin/lang/README.md
`--units U`	metric | imperial | a units.json system; --region auto|lang|de-DE; --currency EUR
`--lang-list`	the languages found; --lang-check FILE.po checks a pack; --lang-template FILE.pot writes the texts to translate
`--warp`	draw the 3D view in software (WARP) even with a graphics card
`--selftest DIR`	window self-test: off-screen, homes and runs the demo program at --factor 5, pictures and selftest.json into DIR, exit status 1 on failure
`--selftest-setup DIR`	the same for setting up workpieces (panel, dragging, keys, clamps, vise)
`--record FILE`	log real-time input for --replay
`--mcap FILE`	record signals, events, serial lines and UI frames (PlotJuggler, Foxglove)
`--vcd FILE`	record the controller's pins and the machine's lines (GTKWave, PulseView)
`--rec-period MS`	sampling period of the recordings (default 1)
`--stock x0,x1,y0,y1,ztop,zbot[,res]`	a block held as if bolted down (machine mm; cells default 0.02 mm)
`--setup JSON|@FILE`	the workpiece set up as on the machine: shape, size, material, fixture (clamps, vise, vblock, tape, none, fixed), position; e.g. '{"shape":"block","size":[60,40,20], "material":"6061-T6","fixture":"vise","at":[100,75]}' (see twin/core/README)

option	what it does
`--tool 'flat 6 n2'`	the cutter: flat|ball|bull|v|drill D [Rc|angle] [nN] (default from the archive)
`--part FILE.stl[@X,Y,Z]`	the part the program should make, its origin at X,Y,Z machine mm (default 0,0,0)
`--stock-stl FILE[,RES[,FRAME]]`	write the workpiece as a binary STL at the end (cells RES, default 0.1; FRAME machine (default), program: the program's zero, blank: its own corner)
`--stock-png FILE[,RES]`	write its height map, or the deviation from --part, as a PNG
`--branch S`	with --replay and a real-time option: replay the log to S seconds, then carry on live
`--mtconnect PORT`	MTConnect adapter (SHDR) for an agent on PORT (7878 is usual)
`--lockstep PORT`	a script drives the twin over localhost PORT (twin/py/mc1twin): virtual time moves only when it says, as fast as the host allows (api.h has the requests); 0: any free port, printed on standard output as lockstep-port N
`--api-token T`	the lockstep client's first request must be hello with token T
`--exit-with-client`	the session ends when its lockstep client goes (or none came within 30 s)
`--control PORT`	the control port (AI access, as the window's AI switch opens it): the lockstep API's requests from several local clients while time runs as usual; 0 any port
`--control-file F`	write its port and token to F as JSON (the MCP server reads it)
`--workdir DIR`	programs' versions here (default Documents/Twinwright Studio/programs; TWINWRIGHT_WORKDIR)
`--control-token T`	its token (default: a random one)
`--control-confirm yes|no|ask`	how its confirmations are answered (default yes: no window to ask)
`--own-call JSON`	{"tool":..., "args":{...}}: a tool called from the window's own interface (its own channel, the AI switch off or on), printed as "own-call N {...}" (tests; up to 12)
`--mcp`	the MCP server for AI clients on standard input and output (they start it; see twin/docs/05 section 9): background twins, and the window when its AI switch is on
`--mcp-http PORT`	the same over Streamable HTTP at http://127.0.0.1:PORT/mcp
`--ai`	(Windows) the window with its AI switch on (it remembers the switch anyway)
`--mtconnect-devices FILE`	write the matching MTConnect device model (Devices.xml; - for stdout)
`--fmi-model-description FILE`	write the FMI 2.0 modelDescription.xml of the lockstep API's FMI-shaped interface (fmi_* requests; twin/py/mc1twin/fmi.py; - for stdout)
`--shadow HOST[:PORT]|COMn|/dev/ttyACM0`	shadow mode: the sender port relays to the real machine's controller (Telnet, port 23, or its USB serial port); the twin runs the same lines alongside and compares answers, path, settings and run times (use --factor 1, the default)
`--shadow-guard`	with --shadow: each line of the sender's is looked at before it reaches the machine; a predicted crash holds it back and feed-holds the machine (the views release it)
`--version`	the program's version and the kind of build (public, internal; a public build with the test key says so)
`--licence-status [DIR] [--json] | --licence-request [DIR] | --licence-import FILE [DIR] | --licence-canon FILE | --licence-keys`	the licence files (docs/14): their state; this computer's request code for offline activation; a licence file into the folder (DIR: the machine's licence folder unless given); a file's signed bytes; the keys this build takes
`--licence-activate CODE [DIR] [--json] | --licence-trial maker|pro [DIR] [--json] | --licence-renew [DIR] [--json] | --licence-renew-due [DIR] | --licence-code CODE | --licence-server`	online (product/23, product/30): activate this computer with an activation code; ask the licence server for this computer's 30-day trial of an edition; renew the licence that is due, now; whether the window's start would renew (a day since the last try); a typed code in its canonical form; the server this build talks to (the last two send nothing)

Machine profile schema: MC-1

Every key of the machine archive the twin is built from (24 sections), with the value the MC-1's archive gives - for the twin's internal error model (`accuracy`, the coupling's play, and where their values come from) the keys only. The values are the model's parameters - typical values until a machine is measured - not specifications of the machine and not its accuracy; every number the twin gives about the physical world is an estimate.

`schema`

the archive format and its version

key	type	unit	value	what it is
`schema`	string		"mc1-machine/0.1"	

`generated`

what made this archive, from which inputs (hashes of params.py and wiring.py)

key	type	unit	value	what it is
`generated.by`	string		"twin/gen/gen_machine.py"	
`generated.params_sha256`	string		"6528e49911ce4825f7555ab9e42991051ee85046b9ed9511f960cfbb…	
`generated.wiring_sha256`	string		"4f378185683bcbd33fb0a42e8fa2b79e93474c861a9a2f76da1e7ee1…	
`generated.report_json`	object			the design checks' report the archive was made from: its path and when it was written
`generated.report_json.path`	string		"out/report.json"	
`generated.report_json.modified`	string		"2026-10-07T18:16:05"	

`units`

the base units

key	type	unit	value	what it is
`units.length`	string		"mm"	
`units.mass`	string		"kg"	
`units.time`	string		"s"	

`name`

key	type	unit	value	what it is
`name`	string		"MC-1"	

`coordinates`

the machine coordinate frame in words (MPos = machine - home offset)

key	type	unit	value	what it is
`coordinates`	string		"machine X 0..200 left to right, Y 0..150 tool relative t…	

`power_on_pos`

where the axes stand at power-on, machine mm (X, Y, Z)

`axes`

the three linear axes, in the order X, Y, Z

key	type	unit	value	what it is
`axes[].id`	string		"X"	
`axes[].moves`	string		"X carriage with the Z axis and spindle"	

key	type	unit	value	what it is
axes[].travel	array (2)		[0.0, 200.0]	the axis' travel [min, max], machine mm
axes[].home_dir	string		"+"	the direction homing seeks its switch
axes[].steps_per_mm	number	per mm	800.0	steps per mm (the controller's $100-$102)
axes[].dir_sign	number		-1	+1: the motor's positive direction is the axis' positive direction
axes[].direction	object			which way the motor turns the screw and the axis moves: for the drive model and the wiring's polarity
axes[].direction.motor_end	string		"+"	
axes[].direction.screw_hand	string		"right"	
axes[].direction.dir_low_turns	string		"cw (seen from the shaft end; assumed)"	
axes[].direction.machine_per_model	number		1	
axes[].direction.dir_low_moves	number		-1	
axes[].switch	object			the axis' limit / home switches: position, contact, sensor physics
axes[].switch.min	number		-2.0	
axes[].switch.max	number		202.0	
axes[].switch.contact	string		"NC"	
axes[].switch.inputs	string		"separate"	
axes[].switch.type	string		"M8 inductive NPN NC, Sn 2.0 mm"	
axes[].switch.sn_mm	number	mm	2.0	
axes[].switch.gap_design_mm	number	mm	1.0	
axes[].switch.gap_mm	array (2)	mm	[1.0, 1.0]	
axes[].switch.spot_r_cad_mm	number	mm	3.0	
axes[].switch.anchor	string		"cad"	
axes[].switch.target	array (2)		["steel", "steel"]	
axes[].switch.unit_sr	array (2)		[1.0, 1.0]	
axes[].switch.lambda_mm	number	mm	0.7	---------- limit sensors and wiring The inductive sensors' switching model follows research note F (1.3, 1.4, 7): the flag sweeps across the sensing face at gap g; the damping of the oscillator is D(g, e) = exp(-g / lambda) * F(e / w) with a logistic coverage F, and the sensor switches when it reaches the damping of a target at distance Sr straight in front: F(e_on / w) = exp(-(Sr - g) / lambda). e_on is the flag edge's position relative to the sensor axis. Sr = Sn * material factor * unit spread * (1 + tc * (T - 20 C)). The CAD places the sensors for an idealised sensing spot of radius SENSE_R (e_on = -SENSE_R): "anchor": "cad" keeps the designed switch points for the nominal sensor and moves them by what the model predicts for other gaps, materials, units and temperatures; "anchor": "model" uses the model's switch point outright (about 1.9 mm later: the design review's sensitivity case). All values unverified until measured (F 7).
axes[].switch.w_mm	number	mm	1.0	
axes[].switch.hyst_mm	number	mm	0.15	
axes[].switch.response_ms	number	ms	1.0	
axes[].switch.jitter_ms	number	ms	0.25	
axes[].switch.repeat_sigma_mm	number	mm	0.003	
axes[].switch.temp_c	number	°C	20.0	
axes[].switch.temp_coeff_per_k	number	1/K	0.0	
axes[].switch.ready_ms	number	ms	30.0	IEC 60947-5-2 "time delay before availability": after its supply comes up the output stays off (F 5: 30 ms); the controller boots later (controller.boot_ms), so it never sees it (unverified)
axes[].hard_stop	array (2)		[-8.0, 208.0]	where the moving part hits the frame [min, max], machine mm

key	type	unit	value	what it is
`axes[].moving_mass_kg`	number	kg	13.83	

`sensor_materials`

inductive sensors: switching distance reduction factor by target material

key	type	unit	value	what it is
`sensor_materials.steel`	number		1.0	
`sensor_materials.stainless`	number		0.7	
`sensor_materials.brass`	number		0.45	
`sensor_materials.aluminium`	number		0.4	
`sensor_materials.copper`	number		0.35	

`wires`

the field wiring by signal: the controller input, cable, route and length

key	type	unit	value	what it is
`wires.x_min`	object			one signal's wire: the controller input, the cable, its route and length, and the cable chains it bends in (flex)
`wires.x_min.input`	string		"X_LIMIT"	
`wires.x_min.cable`	string		"W09"	
`wires.x_min.route`	string		"固定，横梁后侧"	
`wires.x_min.length_mm`	number	mm	728.0	
`wires.x_min.flex`	string		""	
`wires.x_max`	object			
`wires.x_max.input`	string		"X_LIMIT_MAX"	
`wires.x_max.cable`	string		"W10"	
`wires.x_max.route`	string		"固定，横梁后侧"	
`wires.x_max.length_mm`	number	mm	488.0	
`wires.x_max.flex`	string		""	
`wires.y_min`	object			
`wires.y_min.input`	string		"Y_LIMIT"	
`wires.y_min.cable`	string		"W12"	
`wires.y_min.route`	string		"固定，底板下方"	
`wires.y_min.length_mm`	number	mm	167.0	
`wires.y_min.flex`	string		""	
`wires.y_max`	object			
`wires.y_max.input`	string		"Y_LIMIT_MAX"	
`wires.y_max.cable`	string		"W11"	
`wires.y_max.route`	string		"固定，底板下方"	
`wires.y_max.length_mm`	number	mm	212.0	
`wires.y_max.flex`	string		""	
`wires.z_switch`	object			
`wires.z_switch.input`	string		"Z_LIMIT"	
`wires.z_switch.cable`	string		"W08"	
`wires.z_switch.route`	string		"Z 随动线束 + X 拖链，与主轴线分开绑扎"	
`wires.z_switch.length_mm`	number	mm	1493.0	
`wires.z_switch.flex`	string		"XZ"	
`wires.setter`	object			
`wires.setter.input`	string		"PROBE"	

key	type	unit	value	what it is
wires.setter.cable	string		"W13"	
wires.setter.route	string		"Y 拖链"	
wires.setter.length_mm	number	mm	1274.0	
wires.setter.flex	string		"Y"	
wires.alm	object			
wires.alm.input	string		"MOTOR_FAULT"	
wires.alm.cable	string		""	
wires.alm.route	string		"电控箱内：三台驱动器 ALM 与变频器故障继电器串联"	
wires.alm.length_mm	number	mm	0.0	
wires.alm.flex	string		""	
wires.estop	object			
wires.estop.input	string		"E-STOP"	
wires.estop.cable	string		""	
wires.estop.route	string		"电控箱：急停 SB0 第二组常闭（SB0-2）"	
wires.estop.length_mm	number	mm	0.0	
wires.estop.flex	string		""	
wires.estop_km1	object			
wires.estop_km1.input	string		""	
wires.estop_km1.cable	string		""	
wires.estop_km1.route	string		"电控箱：急停 SB0 第一组常闭（SB0-1），在 KM1 线圈的自锁回路里"	
wires.estop_km1.length_mm	number	mm	0.0	
wires.estop_km1.flex	string		""	
wires.rs485	object			
wires.rs485.input	string		"RS485"	
wires.rs485.cable	string		""	
wires.rs485.route	string		"电控箱内：控制板 RS485 A/B → 变频器 RS+/RS-，屏蔽双绞线"	
wires.rs485.length_mm	number	mm	0.0	
wires.rs485.flex	string		""	
wires.vfd_relay	object			
wires.vfd_relay.input	string		"MOTOR_FAULT"	
wires.vfd_relay.cable	string		""	
wires.vfd_relay.route	string		"电控箱内：变频器故障继电器的常闭触点 TB–TC，串在 ALM 链里（发现 107）"	
wires.vfd_relay.length_mm	number	mm	0.0	
wires.vfd_relay.flex	string		""	

faults

wiring fault models (glitches, intermittent contacts)

key	type	unit	value	what it is
faults.glitch_us	number	µs	50.0	injected faults: a glitch opens the contact this long; an intermittent wire drops out while its cable flexes (per metre of travel of the axes it follows), for a random time
faults.intermittent	object			an intermittent wire: it drops out while its cable flexes, so often per metre of travel of the axes it follows, each time for a random time in the range (ms)
faults.intermittent.per_m	number	per m	5.0	
faults.intermittent.ms	array (2)		[0.2, 3.0]	

`drives`

the feed drives: ball screws, closed-loop steppers, their power (KM1), the L1/L2 models

key	type	unit	value	what it is
`drives.model`	string		"L1"	
`drives.note`	string		"rigid axis driven by a closed-loop stepper: magnetic spr…	
`drives.lead_mm`	number	mm	4.0	
`drives.motor`	object			the stepper motor: holding torque, rotor inertia, steps, detent torque, phase resistance and inductance, torque constant, rated current, iron loss
`drives.motor.type`	string		"NEMA23 closed-loop hybrid stepper, 23HS22-2804D-E1000 cl…	
`drives.motor.holding_nm`	number	N·m	1.26	
`drives.motor.rotor_inertia_kgm2`	number	kg·m²	3e-05	
`drives.motor.full_steps_per_rev`	number	per revolution	200	
`drives.motor.detent_nm`	number	N·m	0.05	
`drives.motor.power_up_jump`	boolean		true	the drive's first current after a power-up (its phase reset): the rotor snaps to the nearest stable position of that state, up to two full steps either way (architecture 4.4; unverified on the drive)
`drives.motor.phase_resistance_ohm`	number	Ω	0.9	L3 (research note D, L3): the phases - R and L of the 2.8 A class (typical catalogue values, unverified), the torque constant fitted to the 48 V pull-out plateau (0.8 N m at 2.8 A; the 1.26 N m holding torque is both phases at rated current), the detent's amplitude as above
`drives.motor.phase_inductance_mh`	number	mH	2.9	
`drives.motor.torque_constant_nm_a`	number	N·m/A	0.286	
`drives.motor.rated_current_a`	number	A	2.8	
`drives.motor.iron_loss_nm_per_krpm`	number	N·m per 1000 rpm	0.0377	the laminations' hysteresis and eddy losses as a drag growing with speed: fitted so that L3 on its dynamometer gives the 48 V pull-out curve above (9 W at 1500 rpm; to be measured, architecture 7.3 step 8)
`drives.coupling_inertia_kgm2`	number	kg·m²	2.6e-06	
`drives.pullout_48v_rpm_nm`	array (9)	rpm/(N·m)		
`drives.drive`	object			the closed-loop drive: encoder counts, holding current, damping, the following error it alarms at and how that alarm is reset, current loop, DC bus, the L3 sub-steps
`drives.drive.type`	string		"CL57T class closed-loop drive, 48 V"	
`drives.drive.loop`	string		"closed"	
`drives.drive.encoder_counts_per_rev`	number	per revolution	4000	
`drives.drive.holding_current`	number		0.5	
`drives.drive.damping_ratio`	number		0.5	
`drives.drive.smoothing_ms`	number	ms	0.0	
`drives.drive.following_error_limit_counts`	number		400	
`drives.drive.alarm_reset`	string		"power"	
`drives.drive.open_loop_below_rpm`	number	rpm	0	
`drives.drive.current_loop_hz`	number	Hz	2000.0	then only the holding current at standstill (0: always closed) L3: its current loop's bandwidth, its bus capacitance and overvoltage alarm (assumed), 5 us steps
`drives.drive.bus_capacitance_uf`	number	µF	470.0	
`drives.drive.overvoltage_v`	number	V	60.0	
`drives.drive.l3_sub_steps`	number		20	
`drives.drive.max_phase_advance_deg`	number	°	0.0	

key	type	unit	value	what it is
drives.alm	object			the alarm chain: the drives' ALM outputs and the spindle drive's fault relay in series to the controller's motor-fault input
drives.alm.wired	boolean		true	
drives.alm.vfd_relay	boolean		true	
drives.alm.chain	string		"the three ALM outputs and the VFD's fault relay (its mul…	
drives.power	object			the drives' supply: the 48 V supply behind the contactor KM1, its timing, the supply's hold-up and rise, the drives' boot
drives.power.source	string		"PSU1 48 V behind KM1 (drives, VFD); the 24 V supply befo…	
drives.power.km1	string		"24 V DC coil, latched by the START button SB1 through it…	
drives.power.km1_latched	boolean		true	
drives.power.km1_drop_ms	number	ms	15	
drives.power.km1_pickup_ms	number	ms	25	
drives.power.psu_holdup_ms	number	ms	40	
drives.power.psu_rise_ms	number	ms	150	
drives.power.drive_boot_ms	number	ms	800	
drives.step_s	number	s	0.0001	
drives.l2	object			L2 (--drive L2): the axial chain between the motor side and the table - stiffness by the nut's distance from its BK bearing (the design's hand calculation at 100 N), the nut's play (the accuracy section's backlash), modal damping assumed; integrated in sub-steps of step_s. The drive's current loop (lag) and velocity loop (damping of the motor side on its magnetic spring) are assumed: a closed-loop drive is tuned not to overshoot - fit both to encoder logs (architecture 7.3 step 1) The nut's play closes with an impact: the grease film makes it strongly damped at a few mm/s (Hunt-Crossley damping, growing with the penetration; restitution assumed)
drives.l2.zeta_axial	number		0.03	
drives.l2.sub_steps	number		5	
drives.l2.current_lag_ms	number	ms	0.2	
drives.l2.servo_damping_ratio	number		0.7	
drives.l2.smoothing_ms	number	ms	1.0	
drives.l2.contact_restitution	number		0.0	
drives.l2.impact_speed_mm_s	number	mm/s	5.0	
drives.l2.masses	number		3	three masses (note D, L2): the coupling's mode sits near 1.75 kHz - 20 sub-steps of 5 us; its damping assumed; the friction model of guides and nut: "karnopp" (Stribeck + viscous with a stick band) or "lugre" (pre-sliding, friction lag at reversals)
drives.l2.sub_steps_3	number		20	
drives.l2.zeta_coupling	number		0.05	
drives.l2.friction	string		"karnopp"	
drives.l2.stribeck_ratio	number		1.25	
drives.l2.stribeck_v_mm_s	number	mm/s	2.0	
drives.l2.viscous_n_s_mm	number	N·s/mm	0.02	
drives.l2.lugre_sigma0_n_um	number	N/µm	10.0	
drives.axes	array (3)			per axis (X, Y, Z): screw, inertias, masses, friction, the L2 elastic model
drives.axes[].id	string		"X"	
drives.axes[].screw_length_mm	number	mm	327.0	
drives.axes[].screw_inertia_kgm2	number	kg·m²	3.347015168018846e-06	

key	type	unit	value	what it is
drives.axes[].rotating_inertia_kgm2	number	kg·m²	3.594701516801885e-05	
drives.axes[].moving_mass_kg	number	kg	13.83	
drives.axes[].equivalent_mass_kg	number	kg	102.52570477707619	
drives.axes[].gravity_n	number	N	0.0	
drives.axes[].friction_n	number	N	57.06657900384689	
drives.axes[].l2	object			the axis in the L2 drive model (--drive L2): the masses, the axial and torsional stiffness by the nut's distance from the fixed bearing, friction, the coupling, the frame under the bearing
drives.axes[].l2.rotating_mass_kg	number	kg	88.69570477707619	
drives.axes[].l2.table_mass_kg	number	kg	13.83	
drives.axes[].l2.k_axial_n_um	array (2)	N/µm		
drives.axes[].l2.friction_motor_n	number	N	47.12388980384689	
drives.axes[].l2.friction_table_n	number	N	9.9426892	
drives.axes[].l2.motor_mass_kg	number	kg	77.22965443852422	three masses: the rotor with half the coupling, the screw with the other half, the table
drives.axes[].l2.screw_mass_kg	number	kg	11.46605033855195	
drives.axes[].l2.coupling_n_um	number	N/µm	1189.2873303312676	
drives.axes[].l2.torsion_gj_nmm2	number	N·mm²	64770810.92360646	the screw's torsion from the coupling to the nut: its root section's G J over the BK journal, the drive journal (as lengths of root section) and the distance BK -> nut
drives.axes[].l2.torsion_extra_mm	number	mm	61.593853491210936	
drives.axes[].l2.k_rest_n_um	array (2)	N/µm		the axial chain without the nut (shaft, BK, mount), by the nut's distance from the BK
drives.axes[].l2.nut_hertz_n_um15	number	N/µm^1.5	31.87940238298543	
drives.axes[].l2.frame	object			the frame under the axis' fixed bearing as one mode along the axis: stiffness, mass, damping
drives.axes[].l2.frame.k_n_um	number	N/µm	33.15	
drives.axes[].l2.frame.m_kg	number	kg	17.64	
drives.axes[].l2.frame.zeta	number		0.02	

drives.axes[].l2.coupling_play_um

The coupling's play, part of the twin's internal error model like the backlash: its value is not published.

key	type	unit	what it is
drives.axes[].l2.coupling_play_um	number	µm	

accuracy

geometric and thermal errors of the tool relative to the work (Phase 4)

The twin's internal error-model parameters, off by default; their values are not published.

key	type	unit	what it is
accuracy.enabled	boolean		
accuracy.note	string		
accuracy.units	string		
accuracy.sign	string		
accuracy.axes	object		
accuracy.axes.X	object		
accuracy.axes.X.lead_um	array (45)	µm	
accuracy.axes.X.periodic_um	number	µm	
accuracy.axes.X.periodic_mm	number	mm	

key	type	unit	what it is
accuracy.axes.X.periodic_phase	number		
accuracy.axes.X.straight_um	object	µm	
accuracy.axes.X.straight_um.Y	array (45)		
accuracy.axes.X.straight_um.Z	array (45)		
accuracy.axes.X.angle_urad	object	µrad	
accuracy.axes.X.angle_urad.A	array (45)		
accuracy.axes.X.angle_urad.B	array (45)		
accuracy.axes.X.angle_urad.C	array (45)		
accuracy.axes.X.backlash_um	number	µm	
accuracy.axes.X.gravity_loaded	boolean		
accuracy.axes.Y	object		
accuracy.axes.Y.lead_um	array (35)	µm	
accuracy.axes.Y.periodic_um	number	µm	
accuracy.axes.Y.periodic_mm	number	mm	
accuracy.axes.Y.periodic_phase	number		
accuracy.axes.Y.straight_um	object	µm	
accuracy.axes.Y.straight_um.X	array (35)		
accuracy.axes.Y.straight_um.Z	array (35)		
accuracy.axes.Y.angle_urad	object	µrad	
accuracy.axes.Y.angle_urad.A	array (35)		
accuracy.axes.Y.angle_urad.B	array (35)		
accuracy.axes.Y.angle_urad.C	array (35)		
accuracy.axes.Y.backlash_um	number	µm	
accuracy.axes.Y.gravity_loaded	boolean		
accuracy.axes.Z	object		
accuracy.axes.Z.lead_um	array (21)	µm	
accuracy.axes.Z.periodic_um	number	µm	
accuracy.axes.Z.periodic_mm	number	mm	
accuracy.axes.Z.periodic_phase	number		
accuracy.axes.Z.straight_um	object	µm	
accuracy.axes.Z.straight_um.X	array (21)		
accuracy.axes.Z.straight_um.Y	array (21)		
accuracy.axes.Z.angle_urad	object	µrad	
accuracy.axes.Z.angle_urad.A	array (21)		
accuracy.axes.Z.angle_urad.B	array (21)		
accuracy.axes.Z.angle_urad.C	array (21)		
accuracy.axes.Z.backlash_um	number	µm	

key	type	unit	what it is
accuracy.axes.Z.gravity_loaded	boolean		
accuracy.square_urad	object	µrad	
accuracy.square_urad.xy	number		
accuracy.square_urad.xz	number		
accuracy.square_urad.yz	number		
accuracy.abbe	object		Abbe arms: where each axis' angular errors are referred to (params.py geometry, model frame: y towards the beam, z up from the base top; the reference tool tip is at z = tip_z0 + Z)
accuracy.abbe.tip_z0	number		
accuracy.abbe.X	object		
accuracy.abbe.X.y	number		
accuracy.abbe.X.z	number		
accuracy.abbe.Z	object		
accuracy.abbe.Z.y	number		
accuracy.abbe.Z.dz	number		
accuracy.abbe.Y	object		
accuracy.abbe.Y.x	number		
accuracy.abbe.Y.y	number		
accuracy.abbe.Y.z	number		
accuracy.thermal	object		
accuracy.thermal.alpha_um_m_k	number	µm/(m·K)	
accuracy.thermal.screw_dt_k	array (3)	K	
accuracy.thermal.bk_at	array (3)		
accuracy.thermal.spindle_growth_um	number	µm	
accuracy.thermal.dynamic	object		first-order dynamics (research note D 8 T-layer, "second version"): each screw warmed by its nut and BK drag times its speed through a thermal resistance, the spindle grown by its bearings (by speed) and its motor's copper (by current) - time constants and gains assumed, to be fitted to a 2 h warm-up log (architecture 7.3 step 14); off unless asked for (the static offsets above, and %thermal, stay as set)
accuracy.thermal.dynamic.enabled	boolean		
accuracy.thermal.dynamic.screw_k_per_w	number	K/W	
accuracy.thermal.dynamic.screw_tau_s	number	s	
accuracy.thermal.dynamic.spindle_um_per_w	number	µm/W	
accuracy.thermal.dynamic.spindle_tau_s	number	s	
accuracy.thermal.dynamic.spindle_bearing_w_at_24000	number	W	
accuracy.thermal.dynamic.spindle_copper_w_rated	number	W	
accuracy.thermal.frame	object		
accuracy.thermal.frame.alpha_um_m_k	number	µm/(m·K)	
accuracy.thermal.frame.dt_k	number	K	
accuracy.thermal.frame.tool_to_table_um_k	number	µm/K	
accuracy.thermal.frame.capacity_j_k	number	J/K	
accuracy.thermal.frame.g_w_k	number	W/K	
accuracy.thermal.frame.motors_w	number	W	
accuracy.thermal.frame.spindle_share	number		
accuracy.thermal.ambient_dt_k	number	K	the room above the reference (20 C): the frame follows it (at once without the dynamics)
accuracy.thermal.part	object		the workpiece (research note L 8): 10-25 % of the cutting power goes into it (0.15 taken), out through the fixture - 0.2 K/W with good contact (vise, clamps), 1 K/W on tape or lying loose
accuracy.thermal.part.heat_share	number		
accuracy.thermal.part.g_good_w_k	number	W/K	

key	type	unit	what it is
accuracy.thermal.part.g_poor_w_k	number	W/K	
accuracy.source	string		

spindle

the spindle drive: the Huanyang VFD (HY protocol, PD parameters), the motor, the pump

key	type	unit	value	what it is
spindle.vfd	object			the spindle drive (Huanyang HY series): rating, supply, the RS-485 port and the HY protocol, timing, the DC bus and its protection levels, its parameters
spindle.vfd.model	string		"Huanyang HY01D523B (HY series)"	
spindle.vfd.rated_power_w	number	W	1500	
spindle.vfd.rated_current_a	number	A	7.0	
spindle.vfd.supply	string		"1 x 220 V through KM1 (build.py ELECTRICAL)"	
spindle.vfd.port	string		"RS-485 to RS+/RS-/DCM (wiring.py PIN_MAP 主轴)"	
spindle.vfd.protocol	string		"HY: address, function, length, data, CRC-16 (Huanyang v1)"	
spindle.vfd.reply_latency_ms	number	ms	5.0	
spindle.vfd.min_gap_ms	number	ms	5.0	
spindle.vfd.boot_s	number	s	1.5	
spindle.vfd.bus_capacitance_uf	number	µF	680.0	
spindle.vfd.lu_v	number	V	200.0	DC bus: without mains it feeds the control and the motor until undervoltage (LU: output off), then the control's own supply stops below control_off_v
spindle.vfd.control_off_v	number	V	100.0	
spindle.vfd.control_w	number	W	15.0	
spindle.vfd.oc_factor	number		2.0	
spindle.vfd.overload_factor	number		1.5	
spindle.vfd.overload_s	number	s	60.0	
spindle.vfd.dc_bus_v	number	V	311.0	
spindle.vfd.factory	object			the drive's PD parameters as they leave the factory (its manual)
spindle.vfd.factory.PD000	number		0	
spindle.vfd.factory.PD001	number		0	
spindle.vfd.factory.PD002	number		0	
spindle.vfd.factory.PD004	number		50.0	
spindle.vfd.factory.PD005	number		50.0	
spindle.vfd.factory.PD006	number		2.5	
spindle.vfd.factory.PD007	number		0.5	
spindle.vfd.factory.PD008	number		220.0	
spindle.vfd.factory.PD009	number		15.0	
spindle.vfd.factory.PD010	number		8.0	
spindle.vfd.factory.PD011	number		0.0	
spindle.vfd.factory.PD014	number		10.0	
spindle.vfd.factory.PD015	number		10.0	
spindle.vfd.factory.PD026	number		0	
spindle.vfd.factory.PD119	number		150	

key	type	unit	value	what it is
spindle.vfd.factory.PD141	number		220.0	
spindle.vfd.factory.PD142	number		7.0	
spindle.vfd.factory.PD143	number		4	
spindle.vfd.factory.PD144	number		1440	
spindle.vfd.factory.PD163	number		0	
spindle.vfd.factory.PD164	number		1	
spindle.vfd.factory.PD165	number		0	
spindle.vfd.factory.PD118	number		1	the protection and compensation group as the twin reads it (numbers and meanings from the manual's list, unverified on the unit): PD118 overvoltage stall prevention (1 on: the decel ramp waits while the DC bus is high), PD120 stall prevention level at constant speed (% of PD142), PD121 its deceleration time (s), PD123 over-torque detection (0 off, 1 warn at constant speed, 2 trip), PD124 its level (% of PD142), PD125 its time (s), PD147 slip compensation (% of the rated slip made up at rated current; 0 off)
spindle.vfd.factory.PD120	number		150	
spindle.vfd.factory.PD121	number		5.0	
spindle.vfd.factory.PD123	number		0	
spindle.vfd.factory.PD124	number		150	
spindle.vfd.factory.PD125	number		0.1	
spindle.vfd.factory.PD147	number		0	
spindle.vfd.factory.PD041	number		8	PD041 the PWM carrier: 0-15 for 0.7, 1, 1.5, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 15, 17, 20 kHz; the manual's factory value for the A01D523B (the HY01D523B) is 8 = 8 kHz (research note N 2.3). Only the sound uses it
spindle.vfd.params	object			the PD parameters commissioning sets (the design's table wiring.VFD_PARAMS)
spindle.vfd.params.PD001	number		2	
spindle.vfd.params.PD002	number		2	
spindle.vfd.params.PD004	number		400.0	
spindle.vfd.params.PD005	number		400.0	
spindle.vfd.params.PD006	number		20.0	
spindle.vfd.params.PD007	number		0.5	
spindle.vfd.params.PD008	number		220.0	
spindle.vfd.params.PD009	number		16.0	
spindle.vfd.params.PD010	number		4.0	
spindle.vfd.params.PD011	number		100.0	
spindle.vfd.params.PD014	number		6.0	
spindle.vfd.params.PD015	number		6.0	
spindle.vfd.params.PD141	number		220.0	
spindle.vfd.params.PD142	number		5.0	
spindle.vfd.params.PD143	number		2	
spindle.vfd.params.PD144	number		3000	
spindle.vfd.params.PD163	number		1	
spindle.vfd.params.PD164	number		2	
spindle.vfd.params.PD165	number		3	
spindle.motor	object			the spindle motor, an induction motor: rating, poles, slip, breakdown torque, no-load current, inertia, friction
spindle.motor.model	string		"GDZ65-800"	
spindle.motor.rated_power_w	number	W	800.0	
spindle.motor.rated_voltage_v	number	V	220.0	
spindle.motor.rated_current_a	number	A	5.0	
spindle.motor.rated_freq_hz	number	Hz	400.0	
spindle.motor.poles	number		2	

key	type	unit	value	what it is
spindle.motor.rated_rpm	number	rpm	24000.0	
spindle.motor.rated_slip	number		0.04	
spindle.motor.breakdown_torque_ratio	number		2.0	
spindle.motor.no_load_current_a	number	A	2.0	
spindle.motor.inertia_kgm2	number	kg·m²	8e-05	
spindle.motor.friction_nm	number	N·m	0.005	
spindle.motor.viscous_nms	number	N·m·s	1e-05	
spindle.coolant_pump	string		"on relay K2 (COOLANT flood), supplied through KM1 (wirin…	
spindle.encoder	object			an incremental encoder on the spindle (G33, G76): lines a revolution and an index pulse once a revolution
spindle.encoder.ppr	number		0	lines a revolution (0: none); the controller's $38 is this (the other firmware: one channel's pulses) or 4 times it (the kernel: quadrature counts)

supply

the 24 V supply of controller, sensors and relays

key	type	unit	value	what it is
supply.psu24	string		"PSU2 24 V 60 W, before KM1: controller, sensors, the coi…	QF1 (the utility) feeds PSU2 directly and, through KM1, PSU1 and the VFD (wiring.py)
supply.psu24_holdup_ms	number	ms	80.0	
supply.psu24_rise_ms	number	ms	100.0	

tool_setter

the tool length sensor: where it stands (machine X, Y), its top (machine Z)

key	type	unit	value	what it is
tool_setter.x	number		25.0	
tool_setter.y	number		150.0	
tool_setter.top_z	number		-42.0	the contact (research note F 7, all unverified): it opens after the plunger has gone down pretravel_mm, the point scatters by repeat_sigma_mm (with --seed), it closes again reset_hyst_mm higher, and it bounces (closes for a moment) bounces times within bounce_ms as it opens
tool_setter.overtravel	number		3.0	
tool_setter.contact	string		"NC"	
tool_setter.diameter	number		30.0	
tool_setter.pretravel_mm	number	mm	0.2	
tool_setter.repeat_sigma_mm	number	mm	0.001	
tool_setter.reset_hyst_mm	number	mm	0.01	
tool_setter.bounces	number		2	
tool_setter.bounce_ms	number	ms	0.5	

touch_probe

a touch probe tool (ball stylus on the PROBE input): ideal - it switches as the ball touches - or typical: a kinematic seat that opens once the stylus has given way by its pretravel (by the approach, scattered with a seed), closes again less deflected; its stylus breaks past its travel (docs/06 12.31)

key	type	unit	value	what it is
touch_probe.model	string		"ideal"	

key	type	unit	value	what it is
touch_probe.stylus_travel_mm	number	mm	2.0	A touch probe is a tool ("probe 2 L30": a ball stylus on the PROBE input with the setter). "ideal": it switches as the ball touches; "typical": a kinematic touch-trigger probe's seat opens once the stylus has given way by its pretravel - along Z, in XY with three lobes round the stylus (three seats), each touch's scattered by repeatability_sigma_mm with --seed - and closes again release_hysteresis_mm less deflected (docs/06 12.31). The MC-1's probe is an option not chosen yet: the typical values are orders of magnitude for a hobby-class probe, an assumption (research O: such probes claim 1-5 um repeatability, unverified), no specification of any probe.
touch_probe.geometry	object			the probe's shape in the collision zone (the jaws, the clamps, the workpiece): its stylus over the ball, its body from body_from_mm above the ball's lowest point up to the collet; 0: not given
touch_probe.geometry.stylus_d_mm	number	mm	1.5	the stylus' diameter over the ball (thinner than the ball: the ball stands for it)
touch_probe.geometry.body_d_mm	number	mm	20.0	the body's diameter (0: no body)
touch_probe.geometry.body_from_mm	number	mm	15.0	where the body starts, above the ball's lowest point (the stylus' length below it)
touch_probe.typical	object			the typical probe's model: pretravel along Z and sideways, the three-lobed pattern, repeatability, release hysteresis - typical of touch-trigger probes, until a probe is calibrated
touch_probe.typical.pretravel_z_mm	number	mm	0.005	
touch_probe.typical.pretravel_xy_mm	number	mm	0.015	
touch_probe.typical.lobing_xy_mm	number	mm	0.005	
touch_probe.typical.lobing_count	number		3	
touch_probe.typical.lobing_phase_deg	number	°	0.0	
touch_probe.typical.repeatability_sigma_mm	number	mm	0.001	
touch_probe.typical.release_hysteresis_mm	number	mm	0.003	

sound

where the sounds come from (model mm at the GLB pose, the motion group each rides on) and the default listener, for the sound engine's panning

key	type	unit	value	what it is
sound.frame	string		"model mm, Z up, at the GLB pose (viewer.glb_pose); moves...	
sound.sources	object			the machine's sources of sound, by name
sound.sources.x_motor	object			one source of sound: where it is (model mm at the GLB pose) and the motion group it rides on
sound.sources.x_motor.at	array (3)		[254.0, 160.0, 250.0]	
sound.sources.x_motor.moves	string		"frame"	
sound.sources.y_motor	object			
sound.sources.y_motor.at	array (3)		[0.0, 256.0, 22.0]	
sound.sources.y_motor.moves	string		"frame"	
sound.sources.z_motor	object			
sound.sources.z_motor.at	array (3)		[0.0, 91.0, 465.0]	
sound.sources.z_motor.moves	string		"X"	
sound.sources.spindle	object			
sound.sources.spindle.at	array (3)		[0.0, 0.0, 206.5]	
sound.sources.spindle.moves	string		"Z"	
sound.sources.tool_tip	object			
sound.sources.tool_tip.at	array (3)		[0.0, 0.0, 59.0]	
sound.sources.tool_tip.moves	string		"Z"	
sound.sources.box	object			
sound.sources.box.at	array (3)		[430.0, 50.0, 140.0]	
sound.sources.box.moves	string		"frame"	

key	type	unit	value	what it is
`sound.sources.setter`	object			
`sound.sources.setter.at`	array (3)		[-75.0, 75.0, 97.0]	
`sound.sources.setter.moves`	string		"Y"	
`sound.sources.pump`	object			
`sound.sources.pump.at`	array (3)		[-335.0, 0.0, 40.0]	
`sound.sources.pump.moves`	string		"frame"	
`sound.listener`	object			where the listener stands and what it faces (model mm)
`sound.listener.at`	array (3)		[0.0, -695.0, 559.0]	
`sound.listener.looks_at`	array (3)		[0.0, 0.0, 99.0]	
`sound.radiation`	object			the cut's sound through the machine's modes (cutting.dynamics): Pa/N well above them, each mode's share of it (the rest radiates flat)
`sound.radiation.pa_per_n`	number	Pa/N	0.005	the cut's sound: the force's harmonics through the machine's modes (cutting.dynamics: the structure's X and Y modes, the tool's) as their accelerations radiate (research note N 2.4) - pa_per_n well above the modes, each mode's share of it (a peak of share / (2 zeta) at its frequency), the shares' rest flat
`sound.radiation.structure_x`	number		0.35	
`sound.radiation.structure_y`	number		0.35	
`sound.radiation.tool`	number		0.3	

`collision`

collision proxies: the table, the spindle stack above the tool tip (coaxial cylinders), what stands on the table (table-local mm, origin at its centre)

key	type	unit	value	what it is
`collision.table`	object			the table's proxy: its size, its origin in machine XY, the reference tool tip's machine Z on it
`collision.table.size`	array (2)		[300.0, 220.0]	
`collision.table.origin_machine_xy`	array (2)		[100.0, 75.0]	
`collision.table.ref_tip_on_table_z`	number		-80.0	
`collision.spindle`	array (6)			
`collision.spindle[].id`	string		"tool"	
`collision.spindle[].r`	number		3.0	
`collision.spindle[].h`	array (2)		[0.0, 15.5]	
`collision.table_items`	array (7)			
`collision.table_items[].id`	string		"setter_flange"	
`collision.table_items[].shape`	string		"box"	
`collision.table_items[].min`	array (3)		[-107.0, 66.0, 0.0]	
`collision.table_items[].max`	array (3)		[-43.0, 84.0, 5.0]	
`collision.groups`	array (3)			
`collision.groups[].id`	string		"z_slide"	
`collision.groups[].moves`	string		"XZ"	
`collision.groups[].min`	array (3)		[-73.5, 45.0, 95.0]	
`collision.groups[].max`	array (3)		[73.5, 65.0, 255.0]	
`collision.near_miss_mm`	number	mm	1.0	a gap this small without contact is a near miss (0: none)

`cutting`

cutting-force coefficients (Altintas' linear edge-force model), materials, tool compliance and structural dynamics for the stock model

key	type	unit	value	what it is
`cutting.material`	string		"6061-T6 (assumed, research note C 1.5)"	
`cutting.ktc_n_mm2`	number	N/mm²	750.0	
`cutting.krc_n_mm2`	number	N/mm²	250.0	
`cutting.kac_n_mm2`	number	N/mm²	150.0	
`cutting.kte_n_mm`	number	N/mm	5.0	
`cutting.kre_n_mm`	number	N/mm	5.0	
`cutting.kae_n_mm`	number	N/mm	2.0	
`cutting.kc_plunge_j_mm3`	number	J/mm³	2.4	
`cutting.tool_default`	string		"flat 6 n2"	
`cutting.stock_res_mm`	number	mm	0.02	
`cutting.chunk_mm`	number	mm	0.5	the cut's time resolution (A10): chunks of 0.5 mm / 20 ms, or "tooth": a tooth period (at most 0.05 mm) at a time; polyline points within merge_tol_um of a segment's chord merged into it (the drives' tiny deviations at L1/L2 stay in the swept surface); the height pyramid skips the blocks the tool passes over; the sweep looks only at the cells each segment can reach below the blank's top (its profile's inverse: a V-bit engraving shallow reaches a tenth of a millimetre - round 16; false: as before, the same cut)
`cutting.chunk_ms`	number	ms	20.0	
`cutting.force_step`	string		"chunk"	
`cutting.merge_tol_um`	number	µm	0.1	
`cutting.pyramid`	boolean		true	
`cutting.sweep_reach`	boolean		true	
`cutting.variation`	boolean		true	the shear coefficients follow chip thickness and cutting speed where a material has orthogonal cutting data ("oblique"); false: constant coefficients everywhere (--set cutting.variation=0)
`cutting.tool_geometry`	object			the end mills' geometry for that transformation: a two-flute aluminium cutter (the values Ozsoy used); the helix also spreads the force's harmonics (stock.c force_harmonics, docs/06 M3)
`cutting.tool_geometry.helix_deg`	number	°	30.0	
`cutting.tool_geometry.rake_deg`	number	°	10.5	
`cutting.tool_strength`	object			the end mill's strength (research note L 4.4; the twin's M4): carbide rod's transverse rupture strength, the share of it a tool takes at once, the flute core's diameter for strength (x D)
`cutting.tool_strength.trs_mpa`	number	MPa	4000.0	
`cutting.tool_strength.allowable`	number		0.35	
`cutting.tool_strength.core_d`	number		0.7	
`cutting.force_model`	string		"tooth"	the cutting forces (V0 P1, stock.c and tooth.c): "tooth" - the tooth-level kernel, disc by disc and tooth by tooth (engagement from the cells cut, the tool's profile, the helix, runout, the minimum chip, Kienzle's law where a material has it); "chunk" - one engagement arc per chunk and the flat end mill's closed forms, as before. The runout at the tool's tip (TIR; research note L 3.3: ER11 collet and spindle 5-15 um), the offset's direction drawn per cutter from the seed unless phase_deg gives it; envelope: the outermost tooth cuts the part, the cutter reaching R + rho max cos(lambda - j 2 pi / N) - holes larger, bosses smaller (finding 187; false: the nominal radius, as before); the edge of a new carbide tool (radius; the minimum chip as a share of it, aluminium 0.2-0.4: note L 3.2)

key	type	unit	value	what it is
cutting.runout	object			the cutter's runout in the collet (TIR), its direction (null: drawn for each tool from the session's seed), and whether the cut takes its envelope (finding 187)
cutting.runout.tir_um	number	µm	10.0	
cutting.runout.phase_deg	null	°	null	
cutting.runout.envelope	boolean		true	
cutting.edge	object			the cutting edge: its radius, and the minimum chip as a share of it - a thinner chip only ploughs
cutting.edge.radius_um	number	µm	2.0	
cutting.edge.h_min_ratio	number		0.3	
cutting.materials	array (17)			the workpiece materials the setup offers, with their cutting data
cutting.materials[].key	string		"6061-T6"	
cutting.materials[].name	string		"Aluminum alloy 6061-T6"	
cutting.materials[].group	string		"VDI 22 / ISO N"	
cutting.materials[].density_g_cm3	number	g/cm³	2.7	
cutting.materials[].ktc_n_mm2	number	N/mm²	750.0	
cutting.materials[].krc_n_mm2	number	N/mm²	250.0	
cutting.materials[].kac_n_mm2	number	N/mm²	150.0	
cutting.materials[].kte_n_mm	number	N/mm	5.0	
cutting.materials[].kre_n_mm	number	N/mm	5.0	
cutting.materials[].kae_n_mm	number	N/mm	2.0	
cutting.materials[].kc_plunge_j_mm3	number	J/mm³	2.4	
cutting.materials[].vc_m_min	array (2)	m/min	[152, 335]	
cutting.materials[].alpha_um_m_k	number	µm/(m·K)	23.6	
cutting.materials[].c_j_kg_k	number	J/(kg·K)	896	
cutting.materials[].color	string		"#b9bdc3"	
cutting.materials[].source	string		"aluminium sharp-tool set (note C 1.5)"	
cutting.materials[].note	string		"measured sharp-carbide set (note C 1.5); Kienzle 800/0.2…	
cutting.materials[].oblique	object			orthogonal cutting data: shear stress, shear and friction angles as functions of the chip, the speed and the rake; the reference chip and speed; the source
cutting.materials[].oblique.tau_s_mpa	array (4)	MPa	[205.928, 204.038, 0.016, -0.056]	
cutting.materials[].oblique.phi_n_deg	array (4)	°	[13.866, 62.929, 0.005, 0.259]	
cutting.materials[].oblique.beta_a_deg	array (4)	°	[20.835, -4.901, -0.007, 0.291]	
cutting.materials[].oblique.h_ref_mm	number	mm	0.02	
cutting.materials[].oblique.vc_ref_m_min	number	m/min	452.4	
cutting.materials[].oblique.source	string		"M. Ozsoy, MATECA 6(3) 263-271 (2025), Table 1 (orthogona…	
cutting.compliance	object			the static give at the tool tip: the tool (modulus, core, flute length), the holder's joint, the frame, the drives by the nut's position
cutting.compliance.note	string		"static compliance at the tool tip, force on the tool: to…	
cutting.compliance.tool_e_n_mm2	number	N/mm²	600000.0	
cutting.compliance.flute_core	number		0.8	

key	type	unit	value	what it is
cutting.compliance.flute_length_d	number		2.5	
cutting.compliance.tool_density_kg_m3	number		14500.0	the tool's mass for its own bending mode (cutting.dynamics): cemented carbide, the fluted section a share of the full circle's area (two to four flutes: about 0.6-0.75; assumed)
cutting.compliance.flute_area	number		0.65	
cutting.compliance.joint_um_per_100n	number	µm per 100 N	10.0	
cutting.compliance.frame_um_per_100n	array (3)	µm per 100 N		
cutting.compliance.frame_portal_um_per_100n	array (3)	µm per 100 N	[2.3492, 0.0, 0.0]	
cutting.compliance.drive_um_per_100n	object	µm per 100 N		each axis' drive, axially, by the nut's distance from the fixed bearing
cutting.compliance.drive_um_per_100n.X	array (2)			
cutting.compliance.drive_um_per_100n.Y	array (2)			
cutting.compliance.drive_um_per_100n.Z	array (2)			
cutting.compliance.source	string		"research note C 3.1-3.3; frame: out/fea/fea_result.json ...	
cutting.dynamics	object			the tool point's frequency response for chatter: the structure's and the tool's natural frequencies and damping, the spindle's speed range, input shaping
cutting.dynamics.note	string		"tool-point FRF, per direction x (feed axes X, Y): struct...	
cutting.dynamics.structure_fn_hz	array (2)	Hz	[161.0, 196.0]	
cutting.dynamics.tool_fn_hz	number	Hz	1500.0	
cutting.dynamics.zeta	array (3)		[0.03, 0.03, 0.03]	
cutting.dynamics.rpm_range	array (2)	rpm	[6000.0, 24000.0]	
cutting.dynamics.source	string		"research note C 4.1, 4.4 (ZOA; the holder mode's frequen...	
cutting.dynamics.input_shaping	object			the kernel's input shaping (K1.10, kernel/docs/08-input-shaping.md): what the structure modes ask of it
cutting.dynamics.input_shaping.note	string		"not needed at the commissioned accelerations ($120-$122 ...	
cutting.dynamics.input_shaping.commissioned	object			the input shaping settings as commissioned
cutting.dynamics.input_shaping.commissioned.$1000	number		0	
cutting.dynamics.input_shaping.commissioned.$1001	number		0	
cutting.dynamics.input_shaping.commissioned.$1002	number		0	
cutting.dynamics.input_shaping.raised_acceleration	object			the input shaping settings for raised accelerations: the modes they suppress
cutting.dynamics.input_shaping.raised_acceleration.$1000	number		2	
cutting.dynamics.input_shaping.raised_acceleration.$1010	number		161.0	
cutting.dynamics.input_shaping.raised_acceleration.$1020	number		0.05	
cutting.dynamics.input_shaping.raised_acceleration.$1001	number		2	
cutting.dynamics.input_shaping.raised_acceleration.$1011	number		196.0	
cutting.dynamics.input_shaping.raised_acceleration.$1021	number		0.05	
cutting.dynamics.input_shaping.source	string		"kernel line K1.10 (0bd7ffe): tests/shaper_test.py residu...	

`workholding`

the fixtures: the tapped grid, vise, parallels, strap clamps, tape, V-blocks, friction

key	type	unit	value	what it is
`workholding.grid`	object			the table's tapped grid (model.py: M6 x 12 in columns GRID_XS either side, 9 rows from y -100)
`workholding.grid.thread`	string		"M6"	
`workholding.grid.x`	array (8)		[-125.0, -100.0, -75.0, -50.0, 50.0, 75.0, 100.0, 125.0]	
`workholding.grid.y`	array (9)		[-100.0, -75.0, -50.0, -25.0, 0.0, 25.0, 50.0, 75.0, 100.0]	
`workholding.vise`	object			a 75 mm self-centring vise (5th Axis V75100X class, note H 2.2 [R30]) on a 10 mm aluminium subplate bolted into the grid: the only low vise that leaves room under MC-1's 80 mm of Z (jaws 66 mm up)
`workholding.vise.name`	string		"75 mm self-centering precision vise (V75100X class) + 10…	
`workholding.vise.self_centring`	boolean		true	
`workholding.vise.jaw_width`	number		75.0	
`workholding.vise.jaw_height`	number		21.15	
`workholding.vise.jaw_thickness`	number		12.0	
`workholding.vise.step_height`	number		2.5	
`workholding.vise.min_opening`	number		3.25	
`workholding.vise.max_opening`	number		77.0	
`workholding.vise.base_height`	number		35.0	
`workholding.vise.body_width`	number		75.0	
`workholding.vise.body_length`	number		100.0	
`workholding.vise.plate_thickness`	number		10.0	
`workholding.vise.plate_length`	number		130.0	
`workholding.vise.plate_width`	number		110.0	
`workholding.vise.clamp_force_n`	number	N	7400.0	
`workholding.parallels`	object			a standard parallel set (note H 2.6 [R45]): 1/8" thick, 1/2"..1-5/8" high in 1/8" steps, 6" long
`workholding.parallels.thickness`	number		3.18	
`workholding.parallels.min_height`	number		12.7	
`workholding.parallels.step`	number		3.175	
`workholding.parallels.max_height`	number		41.3	
`workholding.parallels.length`	number		152.4	
`workholding.clamp`	object			M6 serrated step clamp (Te-Co 30501M, 64 x 25 x 13) with a step block behind the stud; the stud screws into the grid; 5.4 N m gives 4.45 kN in the stud (Carr Lane, note H 2.3 [R37][R39])
`workholding.clamp.strap_length`	number		64.0	
`workholding.clamp.strap_width`	number		25.0	
`workholding.clamp.strap_thickness`	number		13.0	
`workholding.clamp.nose_on_part`	number		8.0	
`workholding.clamp.heel_length`	number		14.0	
`workholding.clamp.reach_min`	number		7.0	
`workholding.clamp.reach_max`	number		35.0	
`workholding.clamp.stud_diameter`	number		6.0	
`workholding.clamp.washer_diameter`	number		12.0	
`workholding.clamp.nut_height`	number		7.0	
`workholding.clamp.bolt_preload_n`	number	N	4450.0	

key	type	unit	value	what it is
workholding.tape	object			3M VHB 4910, 1.0 mm acrylic foam (note H 2.4 [R41]): 480 kPa overlap shear after 72 h; half of it counted (peel from the moment of a lateral force above the bond line, imperfect contact) [assumed]
workholding.tape.thickness	number		1.0	
workholding.tape.shear_mpa	number	MPa	0.48	
workholding.tape.effective_share	number		0.5	
workholding.vblock	object			a large V-block pair (Fowler X-Blox, 50.8 x 38.1 x 38.1, 38 mm capacity, 90 deg [R43]) with its strap clamp (force assumed)
workholding.vblock.length	number		38.1	
workholding.vblock.width	number		50.8	
workholding.vblock.height	number		38.1	
workholding.vblock.v_width	number		28.0	
workholding.vblock.clamp_force_n	number	N	2000.0	
workholding.friction	object			friction coefficients - the part on the table, in the jaws, under a strap - and the safety factor of the slip check
workholding.friction.part_on_table	number		0.3	physics, not design values (note H 2.7): the part slips when the cut beats these
workholding.friction.jaw_on_part	number		0.35	
workholding.friction.strap_on_part	number		0.35	
workholding.friction.safety_factor	number		3.0	

total_mass_kg

key	type	unit	value	what it is
total_mass_kg	number	kg	65.75	

controller

the controller: firmware, input filtering, boot time, the $-settings, the pin map

key	type	unit	value	what it is
controller.firmware	string		"Twinwright controller firmware (the kernel, 1.0) with it…	
controller.board_profile	string		"Teensy 4.1 + T41U5XBB: 24 MHz step timer, USB-CDC"	
controller.board	object			the controller board as the MCU side is modelled: step timer, step pulses and the $29 delay, input debounce, the sender's link (architecture 4.2)
controller.board.id	string		"teensy41_t41u5xbb"	
controller.board.text	string		"Teensy 4.1 (i.MX RT1062, Cortex-M7 600 MHz) on a Brookwo…	
controller.board.my_machine_board	string		"BOARD_T41U5XBB"	
controller.board.f_step_timer_hz	number	Hz	24000000	the step timer's clock (hal.f_step_timer); the twin's virtual clock runs at it
controller.board.timer_period_extra	number		1	
controller.board.timer_max_cycles	number		1048575	
controller.board.wake_up_ms	number	ms	2.0	
controller.board.step_pulse_latency_us	number	µs	0.2	
controller.board.step_toff_min_us	number	µs	2.0	
controller.board.pulse_timer_hz	number	Hz	150000000	
controller.board.signals	array (4)		["estop", "feed_hold", "cycle_start", "motor_fault"]	
controller.board.debounce_ms	number	ms	40	control inputs are acted on this long after their edge, if still on (0: at once)

key	type	unit	value	what it is
controller.board.debounced	array (4)		["reset", "feed_hold", "cycle_start", "safety_door"]	
controller.board.link_default	string		"usb_cdc"	the sender's line as the firmware is configured (my_machine.h USB_SERIAL_CDC): native USB, a UART, or Telnet
controller.board.links	object			the lines the board offers the sender: usb_cdc and telnet are packets the foreground polls, uart a byte per slot with a receive interrupt and the host's USB-serial adapter
controller.board.links.usb_cdc	object			the USB CDC link: frames, packets, buffers, polling, the host's latencies and jitter
controller.board.links.usb_cdc.frame_us	number	µs	125.0	
controller.board.links.usb_cdc.packet	number		512	
controller.board.links.usb_cdc.rx_capacity	number		4096	
controller.board.links.usb_cdc.poll_us	number	µs	50.0	
controller.board.links.usb_cdc.poll_bytes	number		20	
controller.board.links.usb_cdc.tx_buffer	number		6124	
controller.board.links.usb_cdc.tx_flush_us	number	µs	75.0	
controller.board.links.usb_cdc.host_out_latency_us	number	µs	250.0	
controller.board.links.usb_cdc.host_in_latency_us	number	µs	500.0	
controller.board.links.usb_cdc.jitter_us	number	µs	250.0	
controller.board.links.uart	object			the UART link through a USB-serial adapter: baud rate, the adapter's packets and timer, the host's latencies and jitter
controller.board.links.uart.baud	number		115200.0	
controller.board.links.uart.adapter	string		"FT232R-class USB-serial adapter"	
controller.board.links.uart.adapter_packet	number		62	
controller.board.links.uart.adapter_timer_ms	number	ms	16.0	the adapter's latency timer: a part-filled packet waits this long
controller.board.links.uart.host_out_latency_us	number	µs	1000.0	
controller.board.links.uart.host_in_latency_us	number	µs	1000.0	
controller.board.links.uart.jitter_us	number	µs	500.0	
controller.board.links.telnet	object			the network (Telnet) link: packets, buffers, polling, latencies and jitter
controller.board.links.telnet.frame_us	number	µs	0.0	
controller.board.links.telnet.packet	number		1460	
controller.board.links.telnet.rx_capacity	number		2920	
controller.board.links.telnet.poll_us	number	µs	1000.0	
controller.board.links.telnet.poll_bytes	number		1460	
controller.board.links.telnet.tx_buffer	number		1024	
controller.board.links.telnet.tx_flush_us	number	µs	1000.0	
controller.board.links.telnet.host_out_latency_us	number	µs	200.0	
controller.board.links.telnet.host_in_latency_us	number	µs	200.0	
controller.board.links.telnet.jitter_us	number	µs	300.0	
controller.board.fg_cost_us	object	µs		the foreground's CPU time per unit of work (a line parsed, a block planned, a step segment prepared), us

key	type	unit	value	what it is
`controller.board.fg_cost_us.line`	number		15.0	The foreground's CPU time (architecture 4.2 "前台耗时模型"): charged per unit of work - a line parsed and executed, a block planned (with the planner's recalculation), a step segment prepared - so a slow foreground starves the step segment buffer as on a real MCU. Estimates for the 600 MHz Cortex-M7 with its FPU (assumed): measure on the board with a GPIO toggled around each (03-commissioning).
`controller.board.fg_cost_us.plan_block`	number		5.0	
`controller.board.fg_cost_us.segment`	number		3.0	
`controller.board.identity`	object			What the controller says about itself in $I (research note I 1.2, docs/06 G3): the driver's own lines as the iMXRT1062 driver sets them (driver.c: hal.info, f_mcu, driver_version; USB_SERIAL_CDC 2: the options) and the board map's name - read the board's $I at commissioning and put its lines here (FREE MEMORY then too). twin_marker: one more line, [PLUGIN:<marker>], tells the twin apart; "" gives the board's $I exactly (a sender cannot tell them apart - docs/06 8.6: by choice only)
`controller.board.identity.driver`	string		"iMXRT1062"	
`controller.board.identity.mcu_mhz`	number	MHz	600	
`controller.board.identity.driver_version`	string		"260616"	
`controller.board.identity.driver_options`	string		"USB.2"	
`controller.board.identity.board`	string		"T41U5XBB"	
`controller.board.identity.twin_marker`	string		"MC-1 twin"	
`controller.firmware_options`	object			what the controller's firmware is built with: its build options, signals, probe, plugins, storage, axes, the driver's options
`controller.firmware_options.my_machine_h`	object			the build options as the firmware's my_machine.h names them
`controller.firmware_options.my_machine_h.BOARD_T41U5XBB`	number		1	
`controller.firmware_options.my_machine_h.USB_SERIAL_CDC`	number		2	
`controller.firmware_options.my_machine_h.SPINDLE0_ENABLE`	string		"SPINDLE_HUANYANG1"	
`controller.firmware_options.my_machine_h.MODBUS_ENABLE`	number		1	
`controller.firmware_options.my_machine_h.ESTOP_ENABLE`	number		1	
`controller.firmware_options.my_machine_h.MOTOR_FAULT_ENABLE`	number		1	
`controller.firmware_options.my_machine_h.MPG_ENABLE`	number		0	
`controller.firmware_options.signals`	array (4)		["estop", "feed_hold", "cycle_start", "motor_fault"]	
`controller.firmware_options.probe`	boolean		true	
`controller.firmware_options.toolsetter`	boolean		false	
`controller.firmware_options.plugins`	array (2)		["MODBUS", "HUANYANG VFD"]	
`controller.firmware_options.nvs`	string		"flash"	
`controller.firmware_options.axes`	number		3	
`controller.firmware_options.driver`	object			the board driver's options: debounce, reboot, clock, planner blocks, receive buffer
`controller.firmware_options.driver.software_debounce`	boolean		true	the iMXRT1062 driver's own (assumed from its source; the twin's driver the same): the control inputs are debounced in software (OPT S), $REBOOT restarts the MCU (NEWOPT REBOOT), the RT1062's RTC (NEWOPT RTC - though that driver never sets hal.driver_cap.rtc, so a real board answers $RTC with error:3: finding 94)
`controller.firmware_options.driver.reboot`	boolean		true	
`controller.firmware_options.driver.rtc`	boolean		true	

key	type	unit	value	what it is
controller.firmware_options.driver.planner_blocks	number		100	
controller.firmware_options.driver.rx_buffer	number		1024	
controller.panel	object			the control box's buttons and the controller inputs they are on
controller.panel.cycle_start	object			the cycle start button (SB2): how it reaches the controller, its input and pin, its contact
controller.panel.cycle_start.via	string		"pin"	
controller.panel.cycle_start.input	string		"CYCLE_START"	
controller.panel.cycle_start.teensy_pin	number		17	
controller.panel.cycle_start.button	string		"循环启动按钮 SB2（绿色，1 常开），电控箱门"	
controller.panel.cycle_start.contact	string		"NO"	
controller.panel.feed_hold	object			the feed hold button (SB3): how it reaches the controller, its input and pin, its contact
controller.panel.feed_hold.via	string		"pin"	
controller.panel.feed_hold.input	string		"FEED_HOLD"	
controller.panel.feed_hold.teensy_pin	number		16	
controller.panel.feed_hold.button	string		"进给保持按钮 SB3（黄色，1 常开），电控箱门"	
controller.panel.feed_hold.contact	string		"NO"	
controller.input_filter_us	number	µs	20.0	the board's isolated inputs: shorter pulses do not reach the pin (optocoupler; an RC filter would add to it). Limit inputs are not debounced in software (research F 2, 6)
controller.boot_ms	number	ms	50.0	
controller.settings	object			the $-settings (setting number -> value), also in mc1_settings.nc
controller.settings.$3	number		5	
controller.settings.$100	number		800	
controller.settings.$101	number		800	
controller.settings.$102	number		800	
controller.settings.$110	number		3000.0	
controller.settings.$111	number		3000.0	
controller.settings.$112	number		1500.0	
controller.settings.$120	number		300.0	
controller.settings.$121	number		300.0	
controller.settings.$122	number		200.0	
controller.settings.$130	number		202.0	
controller.settings.$131	number		152.0	
controller.settings.$132	number		82.0	
controller.settings.$10	number		15871	
controller.settings.$11	number		0.002	
controller.settings.$12	number		0.002	
controller.settings.$0	number		5.0	
controller.settings.$29	number		5.0	
controller.settings.$1	number		255	
controller.settings.$37	number		7	
controller.settings.$5	number		7	
controller.settings.$6	number		1	
controller.settings.$22	number		541	
controller.settings.$20	number		1	

key	type	unit	value	what it is
controller.settings.$21	number		1	
controller.settings.$23	number		0	
controller.settings.$44	number		4	
controller.settings.$45	number		3	
controller.settings.$46	number		0	
controller.settings.$180	number		50	
controller.settings.$181	number		50	
controller.settings.$182	number		50	
controller.settings.$190	number		1200	
controller.settings.$191	number		1200	
controller.settings.$192	number		600	
controller.settings.$26	number		250	
controller.settings.$27	number		2.0	
controller.settings.$43	number		1	
controller.settings.$65	number		32	
controller.settings.$341	number		3	
controller.settings.$342	number		50	
controller.settings.$343	number		25	
controller.settings.$344	number		200	
controller.settings.$346	number		0	
controller.settings.$30	number		24000	
controller.settings.$31	number		6000	
controller.settings.$340	number		5	
controller.settings.$374	number		3	
controller.settings.$375	number		50	
controller.settings.$394	number		15	
controller.settings.$460	number		1	
controller.settings_file	string		"twin/machine/mc1_settings.nc"	
controller.pin_map	array (16)			
controller.design_settings_table	array (28)			

`viewer`

the 3D views: the GLB model and its node groups, pose conventions, cable chains

key	type	unit	value	what it is
viewer.glb	string		"out/web/mc1.glb"	Everything twin/web needs to draw the machine: the GLB (moving groups) plus the drag chains, cable routes and Z service loop, which the viewer rebuilds per pose.
viewer.groups	object			the 3D model's motion groups: which part moves with which axis
viewer.groups.frame	string		"fixed_frame"	
viewer.groups.Y	string		"Y_table"	
viewer.groups.X	string		"X_carriage"	
viewer.groups.Z	string		"Z_slide"	
viewer.groups.dyn	string		"cable_chains"	
viewer.glb_pose	string		"model mm, Z up; exported at x_c = y_t = z_up = 0 where x…	
viewer.table_top_z	number		59.0	

key	type	unit	value	what it is
viewer.tool_tip	string		"spindle axis at model y = 0; tool tip = (x_c, 0, table_t…	
viewer.chains	object			the cable chains' geometry for the 3D view
viewer.chains.x	object			the X cable chain: link size, pitch, bend radius, length, where it is fixed, how it moves
viewer.chains.x.h	number		22.0	
viewer.chains.x.w	number		42.0	
viewer.chains.x.pitch	number		20.0	
viewer.chains.x.R	number		38.0	
viewer.chains.x.L	number		400.0	
viewer.chains.x.fixed	number		150.0	
viewer.chains.x.y	number		204.5	
viewer.chains.x.z	number		340.0	
viewer.chains.x.move_dx	number		40.0	
viewer.chains.x.lift	number		1.0	
viewer.chains.x.fill	number		0.85	
viewer.chains.x.loop_sign	number		-1	
viewer.chains.y	object			the Y cable chain: link size, pitch, bend radius, length, where it is fixed, how it moves
viewer.chains.y.h	number		15.0	
viewer.chains.y.w	number		22.0	
viewer.chains.y.pitch	number		15.0	
viewer.chains.y.R	number		20.0	
viewer.chains.y.L	number		360.0	
viewer.chains.y.fixed	number		150.0	
viewer.chains.y.x	number		-169.0	
viewer.chains.y.z	number		0.0	
viewer.chains.y.move_dy	number		0.0	
viewer.chains.y.lift	number		1.0	
viewer.chains.y.fill	number		0.85	
viewer.chains.y.loop_sign	number		-1	
viewer.routes	object			the cables' routes for the 3D view, as points (model mm)
viewer.routes.x_motor	array (5)			
viewer.routes.y_motor	array (6)			
viewer.routes.y_sensor_1	array (4)			
viewer.routes.y_sensor_2	array (4)			
viewer.routes.x_sensor_L	array (7)			
viewer.routes.x_sensor_R	array (7)			
viewer.routes.x_chain_in_zmotor	array (4)			
viewer.routes.x_chain_out	array (7)			
viewer.routes.y_chain_in_setter	array (5)			
viewer.routes.y_chain_out	array (6)			
viewer.route_groups	object			the routes that ride on a cable chain, and which

key	type	unit	value	what it is
viewer.route_groups.x_chain_in_zmotor	string		"x"	
viewer.route_groups.y_chain_in_setter	string		"y"	
viewer.cable_dia	object			each route's cable diameter (mm)
viewer.cable_dia.x_motor	number		9.0	
viewer.cable_dia.y_motor	number		9.0	
viewer.cable_dia.y_sensor_1	number		4.0	
viewer.cable_dia.y_sensor_2	number		4.0	
viewer.cable_dia.x_sensor_L	number		4.0	
viewer.cable_dia.x_sensor_R	number		4.0	
viewer.cable_dia.x_chain_in_zmotor	number		9.0	
viewer.cable_dia.x_chain_out	number		18.0	
viewer.cable_dia.y_chain_in_setter	number		4.0	
viewer.cable_dia.y_chain_out	number		4.0	
viewer.service_loop	object			the Z cables' service loop at three heights of Z, as points (model mm)
viewer.service_loop.0.0	array (7)			
viewer.service_loop.40.0	array (7)			
viewer.service_loop.80.0	array (7)			

provenance

where the values come from: a path pattern ("a[]" every item of a, "x/y_*" these keys, * a prefix; it covers what is under it; the most specific pattern wins) -> its record (B1)

key	t
provenance.axes[].travel	
provenance.axes[].travel.source	s
provenance.axes[].travel.note	s
provenance.axes[].travel.uncertainty	s

key	t
provenance.axes[].travel.refs	a (
provenance.axes[].travel.values	s
provenance.axes[].travel.history	a (
provenance.axes[].travel.history[].date	s
provenance.axes[].travel.history[].change	s
provenance.axes[].steps_per_mm	c
provenance.axes[].steps_per_mm.source	s
provenance.axes[].steps_per_mm.note	s
provenance.axes[].steps_per_mm.uncertainty	s
provenance.axes[].steps_per_mm.refs	a (
provenance.axes[].steps_per_mm.values	s
provenance.axes[].steps_per_mm.history	a (
provenance.axes[].steps_per_mm.history[].date	s
provenance.axes[].steps_per_mm.history[].change	s
provenance.axes[].switch.min/max	c
provenance.axes[].switch.min/max.source	s
provenance.axes[].switch.min/max.note	s
provenance.axes[].switch.min/max.uncertainty	s
provenance.axes[].switch.min/max.refs	a (
provenance.axes[].switch.min/max.values	s
provenance.axes[].switch.min/max.history	a (
provenance.axes[].switch.min/max.history[].date	s
provenance.axes[].switch.min/max.history[].change	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm	c
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.source	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.note	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.uncertainty	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.refs	a (
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.values	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.history	a (
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.history[].date	s
provenance.axes[].switch.sn_mm/gap_mm/gap_design_mm/spot_r_cad_mm.history[].change	s

key	t
provenance.axes[].switch.lambda_mm/w_mm	c
provenance.axes[].switch.lambda_mm/w_mm.source	s
provenance.axes[].switch.lambda_mm/w_mm.note	s
provenance.axes[].switch.lambda_mm/w_mm.uncertainty	s
provenance.axes[].switch.lambda_mm/w_mm.refs	a (
provenance.axes[].switch.lambda_mm/w_mm.values	s
provenance.axes[].switch.lambda_mm/w_mm.history	a (
provenance.axes[].switch.lambda_mm/w_mm.history[].date	s
provenance.axes[].switch.lambda_mm/w_mm.history[].change	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm	c
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.source	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.note	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.uncertainty	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.refs	a (
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.values	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.history	a (
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.history[].date	s
provenance.axes[].switch.hyst_mm/response_ms/jitter_ms/repeat_sigma_mm.history[].change	s
provenance.axes[].switch.temp_c/temp_coeff_per_k	c
provenance.axes[].switch.temp_c/temp_coeff_per_k.source	s
provenance.axes[].switch.temp_c/temp_coeff_per_k.note	s
provenance.axes[].switch.temp_c/temp_coeff_per_k.uncertainty	s
provenance.axes[].switch.temp_c/temp_coeff_per_k.refs	a (
provenance.axes[].switch.temp_c/temp_coeff_per_k.values	s
provenance.axes[].switch.temp_c/temp_coeff_per_k.history	a (
provenance.axes[].switch.temp_c/temp_coeff_per_k.history[].date	s
provenance.axes[].switch.temp_c/temp_coeff_per_k.history[].change	s
provenance.axes[].switch.ready_ms	c
provenance.axes[].switch.ready_ms.source	s
provenance.axes[].switch.ready_ms.note	s
provenance.axes[].switch.ready_ms.uncertainty	s
provenance.axes[].switch.ready_ms.refs	a (
provenance.axes[].switch.ready_ms.values	s
provenance.axes[].switch.ready_ms.history	a (
provenance.axes[].switch.ready_ms.history[].date	s
provenance.axes[].switch.ready_ms.history[].change	s
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr	c
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.source	s
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.note	s
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.uncertainty	s

key	t
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.refs	a (
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.values	s
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.history	a (
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.history[].date	s
provenance.axes[].switch.contact/inputs/type/anchor/target/unit_sr.history[].change	s
provenance.axes[].hard_stop	o
provenance.axes[].hard_stop.source	s
provenance.axes[].hard_stop.note	s
provenance.axes[].hard_stop.uncertainty	s
provenance.axes[].hard_stop.refs	a (
provenance.axes[].hard_stop.values	s
provenance.axes[].hard_stop.history	a (
provenance.axes[].hard_stop.history[].date	s
provenance.axes[].hard_stop.history[].change	s
provenance.axes[].moving_mass_kg	o
provenance.axes[].moving_mass_kg.source	s
provenance.axes[].moving_mass_kg.note	s
provenance.axes[].moving_mass_kg.uncertainty	s
provenance.axes[].moving_mass_kg.refs	a (
provenance.axes[].moving_mass_kg.values	s
provenance.axes[].moving_mass_kg.history	a (
provenance.axes[].moving_mass_kg.history[].date	s
provenance.axes[].moving_mass_kg.history[].change	s
provenance.axes[].id/moves/home_dir	o
provenance.axes[].id/moves/home_dir.source	s
provenance.axes[].id/moves/home_dir.note	s
provenance.axes[].id/moves/home_dir.uncertainty	s
provenance.axes[].id/moves/home_dir.refs	a (
provenance.axes[].id/moves/home_dir.values	s
provenance.axes[].id/moves/home_dir.history	a (
provenance.axes[].id/moves/home_dir.history[].date	s
provenance.axes[].id/moves/home_dir.history[].change	s
provenance.axes[].dir_sign/direction	o
provenance.axes[].dir_sign/direction.source	s
provenance.axes[].dir_sign/direction.note	s
provenance.axes[].dir_sign/direction.uncertainty	s
provenance.axes[].dir_sign/direction.refs	a (
provenance.axes[].dir_sign/direction.values	s
provenance.axes[].dir_sign/direction.history	a (
provenance.axes[].dir_sign/direction.history[].date	s

key	t
provenance.axes[].dir_sign/direction.history[].change	s
provenance.sensor_materials	o
provenance.sensor_materials.source	s
provenance.sensor_materials.note	s
provenance.sensor_materials.uncertainty	s
provenance.sensor_materials.refs	a (
provenance.sensor_materials.values	s
provenance.sensor_materials.history	a (
provenance.sensor_materials.history[].date	s
provenance.sensor_materials.history[].change	s
provenance.wires	o
provenance.wires.source	s
provenance.wires.note	s
provenance.wires.uncertainty	s
provenance.wires.refs	a (
provenance.wires.values	s
provenance.wires.history	a (
provenance.wires.history[].date	s
provenance.wires.history[].change	s
provenance.faults	o
provenance.faults.source	s
provenance.faults.note	s
provenance.faults.uncertainty	s
provenance.faults.refs	a (
provenance.faults.values	s
provenance.faults.history	a (
provenance.faults.history[].date	s
provenance.faults.history[].change	s
provenance.controller.input_filter_us	o
provenance.controller.input_filter_us.source	s
provenance.controller.input_filter_us.note	s

key	t
provenance.controller.input_filter_us.uncertainty	s
provenance.controller.input_filter_us.refs	a (
provenance.controller.input_filter_us.values	s
provenance.controller.input_filter_us.history	a (
provenance.controller.input_filter_us.history[].date	s
provenance.controller.input_filter_us.history[].change	s
provenance.controller.boot_ms	c
provenance.controller.boot_ms.source	s
provenance.controller.boot_ms.note	s
provenance.controller.boot_ms.uncertainty	s
provenance.controller.boot_ms.refs	a (
provenance.controller.boot_ms.values	s
provenance.controller.boot_ms.history	a (
provenance.controller.boot_ms.history[].date	s
provenance.controller.boot_ms.history[].change	s
provenance.controller.board	c
provenance.controller.board.source	s
provenance.controller.board.note	s
provenance.controller.board.uncertainty	s
provenance.controller.board.refs	a (
provenance.controller.board.values	s
provenance.controller.board.history	a (
provenance.controller.board.history[].date	s
provenance.controller.board.history[].change	s
provenance.controller.board.links	c
provenance.controller.board.links.source	s
provenance.controller.board.links.note	s
provenance.controller.board.links.uncertainty	s
provenance.controller.board.links.refs	a (
provenance.controller.board.links.values	s
provenance.controller.board.links.history	a (
provenance.controller.board.links.history[].date	s
provenance.controller.board.links.history[].change	s
provenance.controller.board.identity	c
provenance.controller.board.identity.source	s
provenance.controller.board.identity.note	s
provenance.controller.board.identity.uncertainty	s
provenance.controller.board.identity.refs	a (
provenance.controller.board.identity.values	s
provenance.controller.board.identity.history	a (

key	t
provenance.controller.board.identity.history[].date	s
provenance.controller.board.identity.history[].change	s
provenance.controller.board.fg_cost_us	o
provenance.controller.board.fg_cost_us.source	s
provenance.controller.board.fg_cost_us.note	s
provenance.controller.board.fg_cost_us.uncertainty	s
provenance.controller.board.fg_cost_us.refs	a (
provenance.controller.board.fg_cost_us.values	s
provenance.controller.board.fg_cost_us.history	a (
provenance.controller.board.fg_cost_us.history[].date	s
provenance.controller.board.fg_cost_us.history[].change	s
provenance.controller.settings	o
provenance.controller.settings.source	s
provenance.controller.settings.note	s
provenance.controller.settings.uncertainty	s
provenance.controller.settings.refs	a (
provenance.controller.settings.values	s
provenance.controller.settings.history	a (
provenance.controller.settings.history[].date	s
provenance.controller.settings.history[].change	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file	o
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.source	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.note	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.uncertainty	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.refs	a (
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.values	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.history	a (
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.history[].date	s
provenance.controller.firmware/board_profile/pin_map/design_settings_table/settings_file.history[].change	s
provenance.controller.firmware_options	o
provenance.controller.firmware_options.source	s
provenance.controller.firmware_options.note	s
provenance.controller.firmware_options.uncertainty	s
provenance.controller.firmware_options.refs	a (
provenance.controller.firmware_options.values	s
provenance.controller.firmware_options.history	a (
provenance.controller.firmware_options.history[].date	s
provenance.controller.firmware_options.history[].change	s
provenance.controller.firmware_options.driver	o
provenance.controller.firmware_options.driver.source	s
provenance.controller.firmware_options.driver.note	s

key	t
provenance.controller.firmware_options.driver.uncertainty	s
provenance.controller.firmware_options.driver.refs	a (
provenance.controller.firmware_options.driver.values	s
provenance.controller.firmware_options.driver.history	a (
provenance.controller.firmware_options.driver.history[].date	s
provenance.controller.firmware_options.driver.history[].change	s
provenance.controller.board.signals	o
provenance.controller.board.signals.source	s
provenance.controller.board.signals.note	s
provenance.controller.board.signals.uncertainty	s
provenance.controller.board.signals.refs	a (
provenance.controller.board.signals.values	s
provenance.controller.board.signals.history	a (
provenance.controller.board.signals.history[].date	s
provenance.controller.board.signals.history[].change	s
provenance.controller.panel	o
provenance.controller.panel.source	s
provenance.controller.panel.note	s
provenance.controller.panel.uncertainty	s
provenance.controller.panel.refs	a (
provenance.controller.panel.values	s
provenance.controller.panel.history	a (
provenance.controller.panel.history[].date	s
provenance.controller.panel.history[].change	s
provenance.supply	o
provenance.supply.source	s
provenance.supply.note	s
provenance.supply.uncertainty	s
provenance.supply.refs	a (
provenance.supply.values	s
provenance.supply.history	a (
provenance.supply.history[].date	s
provenance.supply.history[].change	s
provenance.power_on_pos	o
provenance.power_on_pos.source	s
provenance.power_on_pos.note	s
provenance.power_on_pos.uncertainty	s
provenance.power_on_pos.refs	a (
provenance.power_on_pos.values	s

key	t
provenance.power_on_pos.history	a (
provenance.power_on_pos.history[].date	s
provenance.power_on_pos.history[].change	s
provenance.drives.motor	o
provenance.drives.motor.source	s
provenance.drives.motor.note	s
provenance.drives.motor.uncertainty	s
provenance.drives.motor.refs	a (
provenance.drives.motor.values	s
provenance.drives.motor.history	a (
provenance.drives.motor.history[].date	s
provenance.drives.motor.history[].change	s
provenance.drives.motor.detent_nm	o
provenance.drives.motor.detent_nm.source	s
provenance.drives.motor.detent_nm.note	s
provenance.drives.motor.detent_nm.uncertainty	s
provenance.drives.motor.detent_nm.refs	a (
provenance.drives.motor.detent_nm.values	s
provenance.drives.motor.detent_nm.history	a (
provenance.drives.motor.detent_nm.history[].date	s
provenance.drives.motor.detent_nm.history[].change	s
provenance.drives.motor.torque_constant_nm_a	o
provenance.drives.motor.torque_constant_nm_a.source	s
provenance.drives.motor.torque_constant_nm_a.note	s
provenance.drives.motor.torque_constant_nm_a.uncertainty	s
provenance.drives.motor.torque_constant_nm_a.refs	a (
provenance.drives.motor.torque_constant_nm_a.values	s
provenance.drives.motor.torque_constant_nm_a.history	a (
provenance.drives.motor.torque_constant_nm_a.history[].date	s
provenance.drives.motor.torque_constant_nm_a.history[].change	s
provenance.drives.motor.iron_loss_nm_per_krpm	o
provenance.drives.motor.iron_loss_nm_per_krpm.source	s
provenance.drives.motor.iron_loss_nm_per_krpm.note	s
provenance.drives.motor.iron_loss_nm_per_krpm.uncertainty	s
provenance.drives.motor.iron_loss_nm_per_krpm.refs	a (
provenance.drives.motor.iron_loss_nm_per_krpm.values	s
provenance.drives.motor.iron_loss_nm_per_krpm.history	a (
provenance.drives.motor.iron_loss_nm_per_krpm.history[].date	s
provenance.drives.motor.iron_loss_nm_per_krpm.history[].change	s
provenance.drives.motor.power_up_jump	o

key	t
provenance.drives.motor.power_up_jump.source	s
provenance.drives.motor.power_up_jump.note	s
provenance.drives.motor.power_up_jump.uncertainty	s
provenance.drives.motor.power_up_jump.refs	a (
provenance.drives.motor.power_up_jump.values	s
provenance.drives.motor.power_up_jump.history	a (
provenance.drives.motor.power_up_jump.history[].date	s
provenance.drives.motor.power_up_jump.history[].change	s
provenance.drives.pullout_48v_rpm_nm	c
provenance.drives.pullout_48v_rpm_nm.source	s
provenance.drives.pullout_48v_rpm_nm.note	s
provenance.drives.pullout_48v_rpm_nm.uncertainty	s
provenance.drives.pullout_48v_rpm_nm.refs	a (
provenance.drives.pullout_48v_rpm_nm.values	s
provenance.drives.pullout_48v_rpm_nm.history	a (
provenance.drives.pullout_48v_rpm_nm.history[].date	s
provenance.drives.pullout_48v_rpm_nm.history[].change	s
provenance.drives.lead_mm/coupling_inertia_kgm2	c
provenance.drives.lead_mm/coupling_inertia_kgm2.source	s
provenance.drives.lead_mm/coupling_inertia_kgm2.note	s
provenance.drives.lead_mm/coupling_inertia_kgm2.uncertainty	s
provenance.drives.lead_mm/coupling_inertia_kgm2.refs	a (
provenance.drives.lead_mm/coupling_inertia_kgm2.values	s
provenance.drives.lead_mm/coupling_inertia_kgm2.history	a (
provenance.drives.lead_mm/coupling_inertia_kgm2.history[].date	s
provenance.drives.lead_mm/coupling_inertia_kgm2.history[].change	s
provenance.drives.model/note/step_s	c
provenance.drives.model/note/step_s.source	s
provenance.drives.model/note/step_s.note	s
provenance.drives.model/note/step_s.uncertainty	s
provenance.drives.model/note/step_s.refs	a (
provenance.drives.model/note/step_s.values	s
provenance.drives.model/note/step_s.history	a (
provenance.drives.model/note/step_s.history[].date	s
provenance.drives.model/note/step_s.history[].change	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n	c
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.source	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.note	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.uncertainty	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.refs	a (

key	t
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.values	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.history	a (
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.history[].date	s
provenance.drives.axes[].id/screw_length_mm/screw_inertia_kgm2/rotating_inertia_kgm2/moving_mass_kg/gravity_n.history[].change	s
provenance.drives.axes[].equivalent_mass_kg	c
provenance.drives.axes[].equivalent_mass_kg.source	s
provenance.drives.axes[].equivalent_mass_kg.note	s
provenance.drives.axes[].equivalent_mass_kg.uncertainty	s
provenance.drives.axes[].equivalent_mass_kg.refs	a (
provenance.drives.axes[].equivalent_mass_kg.values	s
provenance.drives.axes[].equivalent_mass_kg.history	a (
provenance.drives.axes[].equivalent_mass_kg.history[].date	s
provenance.drives.axes[].equivalent_mass_kg.history[].change	s
provenance.drives.axes[].friction_n	c
provenance.drives.axes[].friction_n.source	s
provenance.drives.axes[].friction_n.note	s
provenance.drives.axes[].friction_n.uncertainty	s
provenance.drives.axes[].friction_n.refs	a (
provenance.drives.axes[].friction_n.values	s
provenance.drives.axes[].friction_n.history	a (
provenance.drives.axes[].friction_n.history[].date	s
provenance.drives.axes[].friction_n.history[].change	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg	c
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.source	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.note	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.uncertainty	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.refs	a (
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.values	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.history	a (
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.history[].date	s
provenance.drives.axes[].l2.rotating_mass_kg/table_mass_kg/motor_mass_kg/screw_mass_kg.history[].change	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um	c
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.source	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.note	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.uncertainty	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.refs	a (
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.values	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.history	a (
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.history[].date	s
provenance.drives.axes[].l2.k_axial_n_um/torsion_gj_nmm2/torsion_extra_mm/k_rest_n_um.history[].change	s

key	t
provenance.drives.axes[].l2.coupling_n_um	o
provenance.drives.axes[].l2.coupling_n_um.source	s
provenance.drives.axes[].l2.coupling_n_um.note	s
provenance.drives.axes[].l2.coupling_n_um.uncertainty	s
provenance.drives.axes[].l2.coupling_n_um.refs	a (
provenance.drives.axes[].l2.coupling_n_um.values	s
provenance.drives.axes[].l2.coupling_n_um.history	a (
provenance.drives.axes[].l2.coupling_n_um.history[].date	s
provenance.drives.axes[].l2.coupling_n_um.history[].change	s
provenance.drives.axes[].l2.nut_hertz_n_um15	o
provenance.drives.axes[].l2.nut_hertz_n_um15.source	s
provenance.drives.axes[].l2.nut_hertz_n_um15.note	s
provenance.drives.axes[].l2.nut_hertz_n_um15.uncertainty	s
provenance.drives.axes[].l2.nut_hertz_n_um15.refs	a (
provenance.drives.axes[].l2.nut_hertz_n_um15.values	s
provenance.drives.axes[].l2.nut_hertz_n_um15.history	a (
provenance.drives.axes[].l2.nut_hertz_n_um15.history[].date	s
provenance.drives.axes[].l2.nut_hertz_n_um15.history[].change	s
provenance.drives.axes[].l2.frame	o
provenance.drives.axes[].l2.frame.source	s
provenance.drives.axes[].l2.frame.note	s
provenance.drives.axes[].l2.frame.uncertainty	s
provenance.drives.axes[].l2.frame.refs	a (
provenance.drives.axes[].l2.frame.values	s
provenance.drives.axes[].l2.frame.history	a (
provenance.drives.axes[].l2.frame.history[].date	s
provenance.drives.axes[].l2.frame.history[].change	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n	o
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.source	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.note	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.uncertainty	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.refs	a (
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.values	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.history	a (
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.history[].date	s
provenance.drives.axes[].l2.friction_motor_n/friction_table_n.history[].change	s
provenance.drives.drive	o
provenance.drives.drive.source	s
provenance.drives.drive.note	s
provenance.drives.drive.uncertainty	s

key	t
provenance.drives.drive.refs	a (
provenance.drives.drive.values	s
provenance.drives.drive.history	a (
provenance.drives.drive.history[].date	s
provenance.drives.drive.history[].change	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg	c
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.source	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.note	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.uncertainty	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.refs	a (
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.values	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.history	a (
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.history[].date	s
provenance.drives.drive.open_loop_below_rpm/current_loop_hz/bus_capacitance_uf/overvoltage_v/max_phase_advance_deg.history[].change	s
provenance.drives.drive.l3_sub_steps	c
provenance.drives.drive.l3_sub_steps.source	s
provenance.drives.drive.l3_sub_steps.note	s
provenance.drives.drive.l3_sub_steps.uncertainty	s
provenance.drives.drive.l3_sub_steps.refs	a (
provenance.drives.drive.l3_sub_steps.values	s
provenance.drives.drive.l3_sub_steps.history	a (
provenance.drives.drive.l3_sub_steps.history[].date	s
provenance.drives.drive.l3_sub_steps.history[].change	s
provenance.drives.alm	c
provenance.drives.alm.source	s
provenance.drives.alm.note	s
provenance.drives.alm.uncertainty	s
provenance.drives.alm.refs	a (
provenance.drives.alm.values	s
provenance.drives.alm.history	a (
provenance.drives.alm.history[].date	s
provenance.drives.alm.history[].change	s
provenance.drives.power	c
provenance.drives.power.source	s
provenance.drives.power.note	s
provenance.drives.power.uncertainty	s
provenance.drives.power.refs	a (
provenance.drives.power.values	s
provenance.drives.power.history	a (
provenance.drives.power.history[].date	s

key	t
provenance.drives.power.history[].change	s
provenance.drives.l2	o
provenance.drives.l2.source	s
provenance.drives.l2.note	s
provenance.drives.l2.uncertainty	s
provenance.drives.l2.refs	a (
provenance.drives.l2.values	s
provenance.drives.l2.history	a (
provenance.drives.l2.history[].date	s
provenance.drives.l2.history[].change	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s	o
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.source	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.note	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.uncertainty	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.refs	a (
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.values	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.history	a (
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.history[].date	s
provenance.drives.l2.zeta_axial/zeta_coupling/servo_damping_ratio/current_lag_ms/smoothing_ms/contact_restitution/impact_speed_mm_s.history[].change	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um	o
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.source	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.note	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.uncertainty	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.refs	a (
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.values	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.history	a (
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.history[].date	s
provenance.drives.l2.stribeck_ratio/stribeck_v_mm_s/viscous_n_s_mm/lugre_sigma0_n_um.history[].change	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k	o
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.source	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.note	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.uncertainty	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.refs	a (
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.values	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.history	a (
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.history[].date	s
provenance.cutting.materials.alpha_um_m_k/c_j_kg_k.history[].change	s
provenance.spindle.vfd.params	o
provenance.spindle.vfd.params.source	s
provenance.spindle.vfd.params.note	s

key	t
provenance.spindle.vfd.params.uncertainty	s
provenance.spindle.vfd.params.refs	a (
provenance.spindle.vfd.params.values	s
provenance.spindle.vfd.params.history	a (
provenance.spindle.vfd.params.history[].date	s
provenance.spindle.vfd.params.history[].change	s
provenance.spindle.vfd.factory	o
provenance.spindle.vfd.factory.source	s
provenance.spindle.vfd.factory.note	s
provenance.spindle.vfd.factory.uncertainty	s
provenance.spindle.vfd.factory.refs	a (
provenance.spindle.vfd.factory.values	s
provenance.spindle.vfd.factory.history	a (
provenance.spindle.vfd.factory.history[].date	s
provenance.spindle.vfd.factory.history[].change	s
provenance.spindle.vfd.protocol	o
provenance.spindle.vfd.protocol.source	s
provenance.spindle.vfd.protocol.note	s
provenance.spindle.vfd.protocol.uncertainty	s
provenance.spindle.vfd.protocol.refs	a (
provenance.spindle.vfd.protocol.values	s
provenance.spindle.vfd.protocol.history	a (
provenance.spindle.vfd.protocol.history[].date	s
provenance.spindle.vfd.protocol.history[].change	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms	o
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.source	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.note	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.uncertainty	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.refs	a (
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.values	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.history	a (
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.history[].date	s
provenance.spindle.vfd.reply_latency_ms/min_gap_ms.history[].change	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*	o
provenance.spindle.vfd.boot_s/oc_factor/overload_*.source	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*.note	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*.uncertainty	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*.refs	a (
provenance.spindle.vfd.boot_s/oc_factor/overload_*.values	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*.history	a (

key	t
provenance.spindle.vfd.boot_s/oc_factor/overload_*.history[].date	s
provenance.spindle.vfd.boot_s/oc_factor/overload_*.history[].change	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v	c
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.source	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.note	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.uncertainty	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.refs	a (
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.values	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.history	a (
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.history[].date	s
provenance.spindle.vfd.bus_capacitance_uf/lu_v/control_off_v/control_w/dc_bus_v.history[].change	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port	c
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.source	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.note	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.uncertainty	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.refs	a (
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.values	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.history	a (
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.history[].date	s
provenance.spindle.vfd.model/rated_power_w/rated_current_a/supply/port.history[].change	s
provenance.spindle.motor.model/rated_*/poles	c
provenance.spindle.motor.model/rated_*/poles.source	s
provenance.spindle.motor.model/rated_*/poles.note	s
provenance.spindle.motor.model/rated_*/poles.uncertainty	s
provenance.spindle.motor.model/rated_*/poles.refs	a (
provenance.spindle.motor.model/rated_*/poles.values	s
provenance.spindle.motor.model/rated_*/poles.history	a (
provenance.spindle.motor.model/rated_*/poles.history[].date	s
provenance.spindle.motor.model/rated_*/poles.history[].change	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a	c
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.source	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.note	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.uncertainty	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.refs	a (
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.values	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.history	a (
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.history[].date	s
provenance.spindle.motor.rated_slip/breakdown_torque_ratio/no_load_current_a.history[].change	s
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms	c
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.source	s

key	t
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.note	s
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.uncertainty	s
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.refs	a (
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.values	s
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.history	a (
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.history[].date	s
provenance.spindle.motor.inertia_kgm2/friction_nm/viscous_nms.history[].change	s
provenance.spindle.coolant_pump	o
provenance.spindle.coolant_pump.source	s
provenance.spindle.coolant_pump.note	s
provenance.spindle.coolant_pump.uncertainty	s
provenance.spindle.coolant_pump.refs	a (
provenance.spindle.coolant_pump.values	s
provenance.spindle.coolant_pump.history	a (
provenance.spindle.coolant_pump.history[].date	s
provenance.spindle.coolant_pump.history[].change	s
provenance.spindle.encoder	o
provenance.spindle.encoder.source	s
provenance.spindle.encoder.note	s
provenance.spindle.encoder.uncertainty	s
provenance.spindle.encoder.refs	a (
provenance.spindle.encoder.values	s
provenance.spindle.encoder.history	a (
provenance.spindle.encoder.history[].date	s
provenance.spindle.encoder.history[].change	s
provenance.tool_setter	o
provenance.tool_setter.source	s
provenance.tool_setter.note	s
provenance.tool_setter.uncertainty	s
provenance.tool_setter.refs	a (
provenance.tool_setter.values	s
provenance.tool_setter.history	a (
provenance.tool_setter.history[].date	s
provenance.tool_setter.history[].change	s
provenance.tool_setter.overtravel/contact	o
provenance.tool_setter.overtravel/contact.source	s
provenance.tool_setter.overtravel/contact.note	s
provenance.tool_setter.overtravel/contact.uncertainty	s
provenance.tool_setter.overtravel/contact.refs	a (
provenance.tool_setter.overtravel/contact.values	s

key	t
provenance.tool_setter.overtravel/contact.history	a (
provenance.tool_setter.overtravel/contact.history[].date	s
provenance.tool_setter.overtravel/contact.history[].change	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms	c
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.source	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.note	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.uncertainty	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.refs	a (
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.values	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.history	a (
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.history[].date	s
provenance.tool_setter.pretravel_mm/repeat_sigma_mm/reset_hyst_mm/bounces/bounce_ms.history[].change	s
provenance.touch_probe.model	c
provenance.touch_probe.model.source	s
provenance.touch_probe.model.note	s
provenance.touch_probe.model.uncertainty	s
provenance.touch_probe.model.refs	a (
provenance.touch_probe.model.values	s
provenance.touch_probe.model.history	a (
provenance.touch_probe.model.history[].date	s
provenance.touch_probe.model.history[].change	s
provenance.touch_probe.stylus_travel_mm	c
provenance.touch_probe.stylus_travel_mm.source	s
provenance.touch_probe.stylus_travel_mm.note	s
provenance.touch_probe.stylus_travel_mm.uncertainty	s
provenance.touch_probe.stylus_travel_mm.refs	a (
provenance.touch_probe.stylus_travel_mm.values	s
provenance.touch_probe.stylus_travel_mm.history	a (
provenance.touch_probe.stylus_travel_mm.history[].date	s
provenance.touch_probe.stylus_travel_mm.history[].change	s
provenance.touch_probe.geometry	c
provenance.touch_probe.geometry.source	s
provenance.touch_probe.geometry.note	s
provenance.touch_probe.geometry.uncertainty	s
provenance.touch_probe.geometry.refs	a (
provenance.touch_probe.geometry.values	s
provenance.touch_probe.geometry.history	a (
provenance.touch_probe.geometry.history[].date	s
provenance.touch_probe.geometry.history[].change	s
provenance.touch_probe.typical	c

key	t
provenance.touch_probe.typical.source	s
provenance.touch_probe.typical.note	s
provenance.touch_probe.typical.uncertainty	s
provenance.touch_probe.typical.refs	a (
provenance.touch_probe.typical.values	s
provenance.touch_probe.typical.history	a (
provenance.touch_probe.typical.history[].date	s
provenance.touch_probe.typical.history[].change	s
provenance.sound.sources	o
provenance.sound.sources.source	s
provenance.sound.sources.note	s
provenance.sound.sources.uncertainty	s
provenance.sound.sources.refs	a (
provenance.sound.sources.values	s
provenance.sound.sources.history	a (
provenance.sound.sources.history[].date	s
provenance.sound.sources.history[].change	s
provenance.sound.sources.pump	o
provenance.sound.sources.pump.source	s
provenance.sound.sources.pump.note	s
provenance.sound.sources.pump.uncertainty	s
provenance.sound.sources.pump.refs	a (
provenance.sound.sources.pump.values	s
provenance.sound.sources.pump.history	a (
provenance.sound.sources.pump.history[].date	s
provenance.sound.sources.pump.history[].change	s
provenance.sound.listener	o
provenance.sound.listener.source	s
provenance.sound.listener.note	s
provenance.sound.listener.uncertainty	s
provenance.sound.listener.refs	a (
provenance.sound.listener.values	s
provenance.sound.listener.history	a (
provenance.sound.listener.history[].date	s
provenance.sound.listener.history[].change	s
provenance.sound.radiation	o
provenance.sound.radiation.source	s
provenance.sound.radiation.note	s
provenance.sound.radiation.uncertainty	s

key	t
provenance.sound.radiation.refs	a (
provenance.sound.radiation.values	s
provenance.sound.radiation.history	a (
provenance.sound.radiation.history[].date	s
provenance.sound.radiation.history[].change	s
provenance.collision	o
provenance.collision.source	s
provenance.collision.note	s
provenance.collision.uncertainty	s
provenance.collision.refs	a (
provenance.collision.values	s
provenance.collision.history	a (
provenance.collision.history[].date	s
provenance.collision.history[].change	s
provenance.collision.groups	o
provenance.collision.groups.source	s
provenance.collision.groups.note	s
provenance.collision.groups.uncertainty	s
provenance.collision.groups.refs	a (
provenance.collision.groups.values	s
provenance.collision.groups.history	a (
provenance.collision.groups.history[].date	s
provenance.collision.groups.history[].change	s
provenance.collision.near_miss_mm	o
provenance.collision.near_miss_mm.source	s
provenance.collision.near_miss_mm.note	s
provenance.collision.near_miss_mm.uncertainty	s
provenance.collision.near_miss_mm.refs	a (
provenance.collision.near_miss_mm.values	s
provenance.collision.near_miss_mm.history	a (
provenance.collision.near_miss_mm.history[].date	s
provenance.collision.near_miss_mm.history[].change	s
provenance.cutting.materials	o
provenance.cutting.materials.source	s
provenance.cutting.materials.note	s
provenance.cutting.materials.uncertainty	s
provenance.cutting.materials.refs	a (
provenance.cutting.materials.values	s
provenance.cutting.materials.history	a (
provenance.cutting.materials.history[].date	s

key	t
provenance.cutting.materials.history[].change	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3	c
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.source	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.note	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.uncertainty	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.refs	a (
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.values	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.history	a (
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.history[].date	s
provenance.cutting.material/ktc_n_mm2/krc_n_mm2/kac_n_mm2/kte_n_mm/kre_n_mm/kae_n_mm/kc_plunge_j_mm3.history[].change	s
provenance.cutting.tool_default/tool_geometry	c
provenance.cutting.tool_default/tool_geometry.source	s
provenance.cutting.tool_default/tool_geometry.note	s
provenance.cutting.tool_default/tool_geometry.uncertainty	s
provenance.cutting.tool_default/tool_geometry.refs	a (
provenance.cutting.tool_default/tool_geometry.values	s
provenance.cutting.tool_default/tool_geometry.history	a (
provenance.cutting.tool_default/tool_geometry.history[].date	s
provenance.cutting.tool_default/tool_geometry.history[].change	s
provenance.cutting.tool_strength	c
provenance.cutting.tool_strength.source	s
provenance.cutting.tool_strength.note	s
provenance.cutting.tool_strength.uncertainty	s
provenance.cutting.tool_strength.refs	a (
provenance.cutting.tool_strength.values	s
provenance.cutting.tool_strength.history	a (
provenance.cutting.tool_strength.history[].date	s
provenance.cutting.tool_strength.history[].change	s
provenance.cutting.force_model	c
provenance.cutting.force_model.source	s
provenance.cutting.force_model.note	s
provenance.cutting.force_model.uncertainty	s
provenance.cutting.force_model.refs	a (
provenance.cutting.force_model.values	s
provenance.cutting.force_model.history	a (
provenance.cutting.force_model.history[].date	s
provenance.cutting.force_model.history[].change	s
provenance.cutting.runout	c
provenance.cutting.runout.source	s
provenance.cutting.runout.note	s

key	t
provenance.cutting.runout.uncertainty	s
provenance.cutting.runout.refs	a (
provenance.cutting.runout.values	s
provenance.cutting.runout.history	a (
provenance.cutting.runout.history[].date	s
provenance.cutting.runout.history[].change	s
provenance.cutting.edge	o
provenance.cutting.edge.source	s
provenance.cutting.edge.note	s
provenance.cutting.edge.uncertainty	s
provenance.cutting.edge.refs	a (
provenance.cutting.edge.values	s
provenance.cutting.edge.history	a (
provenance.cutting.edge.history[].date	s
provenance.cutting.edge.history[].change	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach	o
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.source	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.note	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.uncertainty	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.refs	a (
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.values	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.history	a (
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.history[].date	s
provenance.cutting.stock_res_mm/chunk_mm/chunk_ms/force_step/merge_tol_um/pyramid/sweep_reach.history[].change	s
provenance.cutting.variation	o
provenance.cutting.variation.source	s
provenance.cutting.variation.note	s
provenance.cutting.variation.uncertainty	s
provenance.cutting.variation.refs	a (
provenance.cutting.variation.values	s
provenance.cutting.variation.history	a (
provenance.cutting.variation.history[].date	s
provenance.cutting.variation.history[].change	s
provenance.cutting.compliance	o
provenance.cutting.compliance.source	s
provenance.cutting.compliance.note	s
provenance.cutting.compliance.uncertainty	s
provenance.cutting.compliance.refs	a (
provenance.cutting.compliance.values	s

key	t
provenance.cutting.compliance.history	a (
provenance.cutting.compliance.history[].date	s
provenance.cutting.compliance.history[].change	s
provenance.cutting.dynamics	o
provenance.cutting.dynamics.source	s
provenance.cutting.dynamics.note	s
provenance.cutting.dynamics.uncertainty	s
provenance.cutting.dynamics.refs	a (
provenance.cutting.dynamics.values	s
provenance.cutting.dynamics.history	a (
provenance.cutting.dynamics.history[].date	s
provenance.cutting.dynamics.history[].change	s
provenance.workholding	o
provenance.workholding.source	s
provenance.workholding.note	s
provenance.workholding.uncertainty	s
provenance.workholding.refs	a (
provenance.workholding.values	s
provenance.workholding.history	a (
provenance.workholding.history[].date	s
provenance.workholding.history[].change	s
provenance.total_mass_kg	o
provenance.total_mass_kg.source	s
provenance.total_mass_kg.note	s
provenance.total_mass_kg.uncertainty	s
provenance.total_mass_kg.refs	a (
provenance.total_mass_kg.values	s
provenance.total_mass_kg.history	a (
provenance.total_mass_kg.history[].date	s
provenance.total_mass_kg.history[].change	s
provenance.viewer	o
provenance.viewer.source	s

key	t
provenance.viewer.note	s
provenance.viewer.uncertainty	s
provenance.viewer.refs	a (
provenance.viewer.values	s
provenance.viewer.history	a (
provenance.viewer.history[].date	s
provenance.viewer.history[].change	s
provenance.schema/generated/units/name/coordinates	c
provenance.schema/generated/units/name/coordinates.source	s
provenance.schema/generated/units/name/coordinates.note	s
provenance.schema/generated/units/name/coordinates.uncertainty	s
provenance.schema/generated/units/name/coordinates.refs	a (
provenance.schema/generated/units/name/coordinates.values	s
provenance.schema/generated/units/name/coordinates.history	a (
provenance.schema/generated/units/name/coordinates.history[].date	s
provenance.schema/generated/units/name/coordinates.history[].change	s

provenance.accuracy

Where the twin's internal error-model parameters come from (the error model is off by default); the records' values are not published.

key	type	unit	what it is
provenance.accuracy	object		
provenance.accuracy.source	string		the category: design, cad (off the CAD model), datasheet, literature (research notes, catalogues), calc (from other values), assumption, measured (on this machine), model (a simulation setting)
provenance.accuracy.note	string		where exactly, and what to measure
provenance.accuracy.uncertainty	string		how far off the values may be, in words ("+-20 %", "exact")
provenance.accuracy.refs	array (3)		references: research notes, datasheets, source files
provenance.accuracy.values	string		a digest of the values the pattern covers: when they change, the history gets an item
provenance.accuracy.history	array (1)		the entry's changes, dated (source, note, uncertainty, values)
provenance.accuracy.history[].date	string		
provenance.accuracy.history[].change	string		
provenance.accuracy.thermal.frame	object		
provenance.accuracy.thermal.frame.source	string		
provenance.accuracy.thermal.frame.note	string		
provenance.accuracy.thermal.frame.uncertainty	string		
provenance.accuracy.thermal.frame.refs	array (2)		
provenance.accuracy.thermal.frame.values	string		
provenance.accuracy.thermal.frame.history	array (1)		
provenance.accuracy.thermal.frame.history[].date	string		
provenance.accuracy.thermal.frame.history[].change	string		
provenance.accuracy.thermal.ambient_dt_k	object		
provenance.accuracy.thermal.ambient_dt_k.source	string		
provenance.accuracy.thermal.ambient_dt_k.note	string		

key	type	unit	what it is
`provenance.accuracy.thermal.ambient_dt_k.uncertainty`	string		
`provenance.accuracy.thermal.ambient_dt_k.refs`	array (0)		
`provenance.accuracy.thermal.ambient_dt_k.values`	string		
`provenance.accuracy.thermal.ambient_dt_k.history`	array (1)		
`provenance.accuracy.thermal.ambient_dt_k.history[].date`	string		
`provenance.accuracy.thermal.ambient_dt_k.history[].change`	string		
`provenance.accuracy.thermal.part`	object		
`provenance.accuracy.thermal.part.source`	string		
`provenance.accuracy.thermal.part.note`	string		
`provenance.accuracy.thermal.part.uncertainty`	string		
`provenance.accuracy.thermal.part.refs`	array (1)		
`provenance.accuracy.thermal.part.values`	string		
`provenance.accuracy.thermal.part.history`	array (1)		
`provenance.accuracy.thermal.part.history[].date`	string		
`provenance.accuracy.thermal.part.history[].change`	string		
`provenance.accuracy.thermal.dynamic`	object		
`provenance.accuracy.thermal.dynamic.source`	string		
`provenance.accuracy.thermal.dynamic.note`	string		
`provenance.accuracy.thermal.dynamic.uncertainty`	string		
`provenance.accuracy.thermal.dynamic.refs`	array (1)		
`provenance.accuracy.thermal.dynamic.values`	string		
`provenance.accuracy.thermal.dynamic.history`	array (1)		
`provenance.accuracy.thermal.dynamic.history[].date`	string		
`provenance.accuracy.thermal.dynamic.history[].change`	string		

`provenance.drives.axes[].l2.coupling_play_um`

Where the coupling's play comes from; the record's values are not published.

key	type	unit	what it is
`provenance.drives.axes[].l2.coupling_play_um`	object		
`provenance.drives.axes[].l2.coupling_play_um.source`	string		the category: design, cad (off the CAD model), datasheet, literature (research notes, catalogues), calc (from other values), assumption, measured (on this machine), model (a simulation setting)
`provenance.drives.axes[].l2.coupling_play_um.note`	string		where exactly, and what to measure
`provenance.drives.axes[].l2.coupling_play_um.uncertainty`	string		how far off the values may be, in words ("+-20 %", "exact")
`provenance.drives.axes[].l2.coupling_play_um.refs`	array (0)		references: research notes, datasheets, source files
`provenance.drives.axes[].l2.coupling_play_um.values`	string		a digest of the values the pattern covers: when they change, the history gets an item
`provenance.drives.axes[].l2.coupling_play_um.history`	array (1)		the entry's changes, dated (source, note, uncertainty, values)
`provenance.drives.axes[].l2.coupling_play_um.history[].date`	string		
`provenance.drives.axes[].l2.coupling_play_um.history[].change`	string		

1759 keys.

Protocol

The Twinwright controller (kernel) speaks the Grbl 1.1 serial protocol, with the extensions that current Grbl senders use. A sender written for Grbl works with it. The senders that were checked, and their settings, are in G-code senders.

Connect

Where	How
The twin in Studio	TCP port 2323 on this computer, one sender at a time.
The MC-1	Its native USB port, a serial port (COM*n* on Windows). The baud rate does not matter; use 115200 where a sender asks for one. The controller answers once the sender raises DTR.

Lines and answers

- **Line ends:** a line ends with LF or CR. A CR LF or LF CR pair counts once; two LFs are two lines.
- **Answers:** the controller answers every line with `ok` or `error:N`, in the order the lines came. Error codes lists every N.
- **Line length:** up to 256 characters. A longer line answers `error:11`.
- **Character counting:** the receive buffer holds 1024 bytes. A sender may keep that many unanswered bytes in flight. `Bf:` in the status report gives the free space.
- **Planner:** 100 blocks. `Bf:` gives the free blocks too.
- **Lines that wait:** in a feed hold or with the door open, lines wait in the buffer. They are answered once the cycle starts again. A reset, `0x19` (stop), `0x03` or `0x85` (jog cancel) drops the lines still waiting.
- **After an error:** the next G-code line does not run and answers the same error. This goes on until a `$` command, an empty line or a reset.

Real-time commands

A real-time command is a single byte. The controller acts on it as it arrives, even in the middle of a line. It is not part of any line and gets no `ok`.

Byte	Command	What it does
`0x18` (Ctrl-X)	Reset	Stops everything at once and clears what is queued. A reset while the axes move raises `ALARM:3`: home again. Not taken while the E-stop is on.
`0x03` (Ctrl-C)	Reset, E-stop on	As Ctrl-X, also taken while the E-stop is on.
`0x19` (Ctrl-Y)	Stop	The axes slow down to a stop, and the lines still queued are dropped. The position is kept, and no alarm is raised.
`ESC`, then `0x14`	Reboot	Restarts the controller, on boards that can restart themselves.
`?`	Status report	One status report.
`0x80` (or `0x05`)	Status report	One status report, except while automatic reports are on.
`0x87`	Full status report	One status report with every field.
`~`, `0x81`	Cycle start	Resumes after a feed hold, the door, `M0`/`M1` or a tool change.
`!`, `0x82`	Feed hold	The axes slow down to a stop and hold. During a jog, it cancels the jog.
`0x83`	Parser state	Reports the modal state, as `$G` does.
`0x84`	Safety door	Acts as the door opening: hold, and park if parking is set up.
`0x85`	Jog cancel	Ends a jog and drops the jog lines still queued.
`0x88`	Optional stop	Switches `M1` pauses on or off.
`0x89`	Single block	Switches single block on or off.
`0x8B`	Handwheel mode	With a handwheel connected: switches its mode. Otherwise ignored.
`0x8C`	Automatic reports	Switches automatic status reports on or off, when `$481` sets their interval. Otherwise ignored.
`0x90`	Feed override 100 %	
`0x91`, `0x92`	Feed override +10 %, −10 %	Range 10–200 %.
`0x93`, `0x94`	Feed override +1 %, −1 %	Range 10–200 %.
`0x95`, `0x96`, `0x97`, `0x98`	Rapid override	100 %, 50 %, 25 %, 5 %.
`0x99`	Spindle override 100 %	
`0x9A`, `0x9B`	Spindle override +10 %, −10 %	Range 10–200 %.
`0x9C`, `0x9D`	Spindle override +1 %, −1 %	Range 10–200 %.

Byte	Command	What it does
`0x9E`	Spindle stop	In a feed hold only: stops the spindle, or starts it again.
`0x9F`	Soft E-stop	Stops at once and raises the E-stop alarm.
`0xA0`	Coolant, flood	Switches flood coolant on or off.
`0xA1`	Coolant, mist	Switches mist coolant on or off.
`0xA3`	Tool change done	Confirms a manual tool change.

The controller drops other control bytes, and the bytes `0x80`–`0xBF` that have no command.

- **UTF-8 text** in comments and messages is read as text. A byte in `0x80`–`0xBF` that belongs to such a character is never taken as a real-time command.
- `$39=0` (legacy real-time characters off): `?`, `!` and `~` are text inside `$` lines and comments, and commands everywhere else. With `$39=1` (the default) they are always commands.

Welcome and identity

After power-on and after every reset, the controller says:

```
Grbl 1.1f ['$' or '$HELP' for help]
```

Senders take a line that starts with "Grbl" as a restart.

`$I` answers with these lines, then `ok`:

Line	What it says
`[VER:Twinwright 1.1f.<date>:<text>]`	The protocol version and its build date. `<text>` is what `$I=<text>` stored.
`[OPT:…]`	Option letters, the planner blocks (100) and the receive buffer (1024).
`[NEWOPT:…]`	Option names: `ENUMS`, `RT+` or `RT-` (`$39`), `HOME`, `ES`, `MPG`, `LATHE`, `REBOOT`, `TC`, `SS` (spindle sync), `SED`, `RTC`, `SD`, `FS`, `YM`.
`[FIRMWARE:Twinwright]`	The firmware's name.
`[SIGNALS:…]`	The control inputs the board has, as letters.
`[NVS STORAGE:…]`	Where the settings are kept.
`[DRIVER:…]`, `[DRIVER VERSION:…]`, `[DRIVER OPTIONS:…]`, `[BOARD:…]`	The board.
`[KERNEL:<version>+<build>]`	The controller's version and build: the commit of a release, or `dev`.
`[KNCAP:…]`	What the board has (below).
`[PLUGIN:…]`	Functions named as senders know them: `MODBUS` and `HUANYANG VFD` for the spindle drive.

What the board has: KNCAP

`[KNCAP:]` is one line of `key=value` pairs, separated by commas. Keys are only ever added at the end and never change their meaning. Read the line by key, not by position. A key that is missing means "not known".

Key	Values	Meaning
`port`	`teensy41`, `stm32h7`, `stm32f4`, `twin`, `host`, `sim`, `unknown`	Which build of the controller this is.
`pv`	a number	The version of the controller's port interface.
`view`	a number	The version of the controller's state view.
`spindle`	`vfd`, `pwm`, `none`	The spindle drive: Modbus, a PWM output, or none.
`encoder`	`0`, `1`	A spindle encoder is fitted.
`sd`	`0`, `1`	The board has an SD card slot.
`mpg`	`0`, `2`	A handwheel port, by its mode number.
`aux`	`di/do/ai/ao`	Auxiliary inputs and outputs: digital in, digital out, analog in, analog out.
`axes`	letters	The axes, for example `XYZ`.
`probe`	`0`, `1`	A probe input. On the MC-1 the tool setter uses it too.
`link`	`usb`, `uart`, `tcp`, `none`	The sender's link. `usb` is a USB serial port that answers once DTR is up.

Shadow mode compares `[KERNEL:]` and `[KNCAP:]` between a real machine and its twin (see Real machine and shadow mode).

Status reports

A status report comes after `?`, `0x80` or `0x87`. With automatic reports on (`$481`, `0x8C`), it also comes by itself. It is one line:

```
<Idle|MPos:0.000,0.000,0.000|Bf:100,1023|FS:0,0|Pn:XYZ>
```

The state comes first:

- `Idle`, `Run`, `Jog`, `Home`, `Check` (check mode), `Sleep`, `Tool` (a tool change);
- `Hold:0` (held) and `Hold:1` (slowing down);
- `Door:0` (closed, ready to resume), `Door:1` (stopped, still open), `Door:2` (opened: holding or parking) and `Door:3` (closed, restoring);
- `Alarm`.

The fields after it depend on `$10` and on the situation:

Field	Meaning
`MPos:` or `WPos:`	Machine or work position (`$10` bit 0).
`Bf:`	Free planner blocks, free receive buffer bytes.
`Ln:`	The line number of the block that runs.
`FS:`	Feed rate, the spindle speed asked for, and the speed the drive reports. A spindle that only goes on and off reports `F:`. In a feed hold the feed is 0.
`Pn:`	Active inputs: limits by axis letter, `P` probe, `D` door, `H` feed hold, `S` cycle start, `R` reset, `E` E-stop.
`WCO:`	The work coordinate offset: every few reports, and after a change.
`Ov:`	Feed, rapid and spindle overrides in %: every few reports, and after a change.
`A:`	Accessories: `S` or `C` spindle clockwise or counter-clockwise, `F` flood, `M` mist.
`Sl:`	The spindle drive's load.
`DTG:`	The distance to go, until the axes stop (`$10` bit 13).
`H:`	Homed or not.
`T:`	The tool, during a tool change.
`TLR:`	A reference tool length is set (`$TLR`).
`MPG:`	The handwheel has control (1) or not (0).
`In:`	The result of the last `M66`.
`WCS:`, `Sc:`, `D:`	The work coordinate system, the scaling, lathe diameter mode (full reports).
`AR`, `AR:`	Automatic reports off, or on with their interval (full reports).
`FW:`	The firmware's name (full reports).
`SD:`	How far a program run from the controller's file system has got.

Other lines the controller sends

Line	When
`ALARM:N`	An alarm. Alarm codes lists every N.
`[MSG:…]`	A message: the controller's own, or a program's `(MSG,…)`, `(PRINT,…)` or `(DEBUG,…)`.
`[GC:…]`	The modal state, after `$G` or `0x83`. With `$10` bit 9, once after each change.
`[PRB:x,y,z:1]`	The point where the probe triggered, and whether it touched (1) or not (0).
`[TLO:…]`	The tool length offset.
`[PARAM:n=value]`	After `$#=n`.
`[G54:…]` … `[G59.3:…]`, `[G28:…]`, `[G30:…]`, `[G92:…]`, `[T:…]`	After `$#`: offsets, stored positions, the tool table.
`>line:ok`, `>line:error:N`	A startup line (`$N0`, `$N1`) ran at start.

`$` commands

`$HELP Commands` lists them on the controller itself.

Command	What it does
`$`, `$HELP`, `$HELP <group>`	Help; the commands; a settings group with details.
`$$`, `$<n>`, `$<n>=<value>`, `$$=<n>`, `$+`	All settings; one setting; change it; one setting's details; all settings again.
`$SED=<n>`	Describe a setting.

Command	What it does
`$#`, `$#=<n>`, `$#=<name>`	Offsets, the probe result, stored positions; one parameter.
`$G`	The modal state.
`$I`, `$I+`, `$I=<text>`	What the controller is (above); everything; store a text with it.
`$N`, `$N0=<line>`, `$N1=<line>`	Show or store the startup lines.
`$C`	Check mode: G-code is checked and nothing moves. A reset ends it.
`$X`	Unlock after an alarm.
`$H`, `$HX`, `$HY`, `$HZ`	Home all axes, or one axis.
`$J=<G-code>`	Jog.
`$B`, `$S`, `$O`	Block delete, single block, optional stop: on or off.
`$SLP`	Sleep.
`$TLR`, `$TPW`	Take the last probing as the reference tool length; measure the new tool on the touch plate.
`$RST=$`, `$RST=#`, `$RST=*`	The MC-1's factory settings (homing and the limits on, its travel); clear the offsets; both, and the startup lines.
`$EA`, `$EAG`, `$EE`, `$EEG`, `$EG`, `$ES`, `$ESG`, `$ESH`, `$E*`	List the alarms, error codes, setting groups and settings, as lines or as CSV.
`$SPINDLES`, `$SPINDLESH`	The spindles.
`$PINS`, `$PINSTATE`, `$PORTS`	The board's pins; the auxiliary inputs and outputs; the serial ports.
`$LIM`, `$HSS`, `$LEV`	The limit inputs, the home switches, the last input events.
`$RTC`, `$RTC=<date and time>`	The controller's clock (ISO 8601).
`$REBOOT`	Restart the controller.
`$F`, `$F+`, `$F=<file>`, `$F<=<file>`, `$FR`	List the programs on the controller's file system, run one, send one back, start over.
`$FD=`, `$FMD=`, `$FRD=`, `$FF=yes`, `$FI`, `$FM`, `$FU`, `$CWD=`, `$PWD`	Delete, make or remove a folder, format, information, mount, unmount, working folder.

Every setting is in Settings.

File formats

Draft (round 14): the text is being written; the sections may still grow.

Programs

A program is a G-code text file in the dialect of Grbl 1.1, which the Twinwright controller speaks. Studio reads any such file. The programs its CAM writes look like this:

```
(Sensor bracket)
(tool T1 flat 6 n3 L20, 6061-T6: S13263, F836, plunge F279, depth of cut 1.20 mm, stepover 2.40 mm)
(program zero: on the blank's top; the machine's top first; T1 is the length reference)
G21 G90 G17 G94 G54
G53 G0 Z0
T1 M6
G53 G0 Z0
M3 S13263
G4 P3 (up to speed)
(face X-2..62 Y-2..42, 0.3 mm deep: 1 layer of 12 passes)
...
M5
M9
G53 G0 Z0
M30
```

Part	What it is
Header	Comments: the job's title; the first tool, the material and the main cutting data; where the program's zero is
Modes	`G21 G90 G17 G94 G54`: millimetres, absolute, the XY plane, feed per minute, work offset G54
`G53 G0 Z…`	A move to the top of Z, in machine coordinates: at the start, at each tool change and at the end. The height is where homing leaves Z.
`T… M6`	A tool change. The controller measures the new tool on the tool setter.
`M3 S…`, `G4 P…`	The spindle on, then a dwell while it comes up to speed. A water-cooled spindle's pump (`M8`) comes first.
Feature comments	Each feature starts with a comment that says what it cuts
Moves	`G0`, `G1`, and arcs `G2`/`G3` with `I` and `J`. Drilling uses `G81`, or `G83` with pecks, its `R` plane 1 mm over the top.
End	`M5`, `M9`, the move to the top, `M30`

Comments are in parentheses. Lines have no numbers.

Studio keeps each version of a program in the programs folder (Documents/Twinwright Studio/programs), in a folder of the program's name:

File	What it holds
`v0001.nc`	The program's text, as it was saved
`v0001.json`	Its version, the time (UTC), who made it, why, the version it came from, its hash and its lines
`v0001.ops.json`	What each feature became (operation, strategy, tool, lines, speed, feed): a setup sheet reads it

Jobs and setups

A job is the description of a part that the CAM makes programs from. It is a JSON object:

Key	Meaning
`name`	The program's name, for example `bracket.nc`
`title`, `description`	Words for the program's header and the setup sheet
`material`	The material, for example `6061-T6`; other standards' names are understood
`tool`	The tool in the spindle, as a spec (`flat 6 n3 L20`)
`tools`	The tool table: `{"1": "flat 6 n3 L20", "2": "drill 3.4 118 L20"}`. Left out: the machine's.
`cutting`	Values that override the chosen ones: `rpm`, `feed`, `ap`, `ae`, `finish_ap`
`safe_z`	The height the moves across go at, in millimetres over the work zero
`zero`	Where the work zero is: `front-left-top` (the default) or `center-top`
`features`	The features, in the order they are cut. See Making programs.
`setups`	Instead of `features`: one object for each setup
`nest`	Instead of `features`: parts nested on a plate

A setup has its own `features`, and says how the part comes to it from the setup before:

Key	Meaning
`turn`	`u`: turned over about X (front and back change places); `v`: about Y (left and right)
`fixture`	Held another way from this setup on (`vise`, `clamps`, `tape` ...)
`zero`	The work zero found again: `front-left-top` or `center-top`. Left out after a turn or a new fixture: `front-left-top`; otherwise the zero of the setup before.
`name`, `tool`, `tools`, `cutting`, `safe_z`	The setup's own, over the job's

```
{"name": "bracket.nc", "title": "Sensor bracket", "material": "6061-T6",
 "tools": {"1": "flat 6 n3 L20", "2": "drill 3.4 118 n2 L20"},
 "features": [
  {"op": "face", "x0": -2, "y0": -2, "x1": 62, "y1": 42, "depth": 0.3, "tool": 1},
  {"op": "pocket", "shape": "rect", "x0": 6, "y0": 6, "x1": 28, "y1": 34, "r": 3, "depth": 4, "tool": 1},
  {"op": "pocket", "shape": "circle", "cx": 44, "cy": 20, "d": "Ø12 H7", "depth": 6, "tool": 1},
  {"op": "drill", "points": [[44, 6], [44, 34]], "depth": 8, "tool": 2}]}
```

Each version of a job is kept beside the program it made, as `job-v0001.json`: its version, the time (UTC), who made it (`operator` or `ai:<client>`), a note, the program version it made, and the job itself. A job in setups also lists each setup's program, the version made and its hash.

Recordings (MCAP)

`--mcap FILE` records a run into an MCAP file (mcap.dev, format version 0): a channel for each kind of data, JSON messages, each channel with its JSON schema, not chunked and not compressed, and a summary at the end with the statistics and a CRC. Studio's tests read the files back against the MCAP specification; tools that read MCAP, such as PlotJuggler and Foxglove, are meant to open them, but that has not been tried yet.

- `--rec-period MS` sets how often the sampled channels are written (1 ms unless given).
- A recording changes nothing in the run: a run recorded and the same run not recorded are the same bit for bit.
- `--play FILE.mcap` plays a recording back in Studio, with a timeline: play, pause, seek, 0.25–10 times the speed. Nothing is simulated while it plays.

Time

Each message's log time is the twin's virtual time, in nanoseconds from power-on. Most channels also carry `t`, the same time in seconds. A sampled channel writes a message only when something in it changed.

The header

A metadata record named `mc1sim` says what the run was:

Key	Meaning
`machine`	The machine's name
`drive_model`	The drive model of the run: `L0`, `L1`, `L2`, `L2/3` (three masses) or `L3`
`f_cpu`	The controller's clock, Hz
`nvs`	A hash of the controller's stored settings when the run began, 16 hexadecimal digits
`sample_period_ms`	The sampling period
`time`	`virtual, ns from power-on`

Channels

Channel	Written	Field	Type	Unit	Meaning
`/machine`	sampled	`pos`	number[3]	mm	Where each axis is: the mechanics, not the controller's idea of it
		`mpos`	number[3] or null	mm	The controller's machine position (MPos); null while the controller is off
		`lost`	number[3]		The steps each axis has lost
		`feed`	number	mm/min	The controller's feed rate now
		`tip`	number[3]	mm	The tool tip against the work: with the twin's error model on, moved by its errors; off, the same as `pos`
`/drives`	sampled, with a drive model from L1	`ferr_um`	number[3]	µm	Each drive's following error
		`state`	string[3]		Each drive's state
		`km1`	boolean		The contactor KM1 closed
		`bus48`	boolean		The drives' 48 V supply on

Channel	Written	Field	Type	Unit	Meaning
`/spindle`	sampled, with a spindle drive	`f_out`	number	Hz	The drive's output frequency
		`rpm`	number	rpm	The spindle's speed
		`amps`	number	A	The drive's output current
		`load_pct`	number	%	The load as the drive reports it
		`torque_nm`	number	N·m	The motor's torque
		`bus_v`	number	V	The drive's DC bus voltage
		`cmd_rpm`	number	rpm	The speed the controller asks for
		`run`, `pump`	boolean		The drive running; the pump on
		`power`	string		The drive's own supply: `ready`, `boot` or `off`
		`fault`	string		The drive's fault, as its display shows it
`/controller`	on change	`state`	string		The controller's state (`Off` while it is off)
		`alarm`	integer		The alarm it is in (0: none)
		`homed`	integer		The axes homed, a bit each: X 1, Y 2, Z 4
		`limits`	integer		The limit inputs that are active, a bit each
		`probe`, `estop`, `motor_fault`, `door`	boolean		The probe, E-stop, motor-fault and door inputs (the MC-1 has no door)
		`on`	boolean		The controller powered
`/power`	on change	`mains`, `v24`, `mcu`, `km1`, `bus48`	boolean		Mains; the 24 V supply; the controller running; KM1; the 48 V supply
		`vfd_mains`, `vfd`, `pump`	boolean, string, boolean		The spindle drive's mains; its state (`none`, `ready`, `boot`, `off`); the pump
`/serial/in`	as it comes	`text`	string		The bytes the controller received; control characters as `\u` escapes
`/serial/out`	as it comes	`line`	string		Each line the controller wrote
`/events`	as they happen	`text`	string		Each entry of the event log: marks, the supply, the drives, the spindle drive, collisions, broken wires
`/cut`	sampled, with a blank	`cutting`	boolean		Material coming off; while not, the figures below are 0 but `rpm` and `removed_mm3`
		`mrr_mm3_s`	number	mm^3/s	The material removal rate
		`ap_mm`, `ae_mm`	number	mm	The axial depth and the radial width of the cut
		`fz_mm`	number	mm	The feed per tooth
		`rpm`	number	rpm	The spindle's speed
		`fx_n`, `fy_n`, `fz_n`	number	N	The cutting force along the machine's axes
		`torque_nm`, `power_w`	number	N·m, W	The cutting torque and power
		`kc_j_mm3`	number	J/mm^3	The specific cutting energy
		`removed_mm3`	number	mm^3	The material removed so far
`/input`	as it comes	`line`	string		The session's inputs, line by line as `--record` writes them (see Session logs)
`/shadow`	as it comes, in shadow mode	`host_s`	number	s	The computer's time
		`dir`	string		`t` to the real machine, `r` from it
		`data`	string		The bytes, each as one character U+0000–U+00FF, up to 1 KB a message
`/ui/state`, `/sound`	for the player				The views' state frame (30 a second while it changes) and the sound's frame (every 10 ms of virtual time when it changes). Studio's own: their fields may change from version to version.

Pin traces (VCD)

`--vcd FILE` writes the controller's pins as a VCD file (IEEE 1364), which GTKWave and PulseView read: the scope `mcu` has the controller's pins, each change at the clock tick it happens (the step pins as pulses of `$0` microseconds); the scope `machine` has the lines outside the controller: the supplies, KM1, the spindle drive, the pump, the E-stop button, the tool setter, the five switches. The timescale is 1 ns.

Session logs

`--record FILE` writes a session's inputs, each with its virtual time: what the senders sent, the real-time commands, the views' commands. `--replay FILE` runs the session again, bit for bit, with the same version of Studio and the same machine files. `--replay FILE --branch S` replays it to S seconds and goes on live from there; with `--record` the branch has a log of its own, which starts from its parent's (`# branch <parent> <ticks>`). A log is text; its lines are Studio's own and may change from version to version.

The world archive

What the twin keeps between sessions, as a real machine does when it is switched off and on again: where the axes stand, the blank with its cuts, the tool in the spindle, the controller's memory and the board's clock, and the heat, which cools down while the machine is off. The controller starts cold: the machine is homed again.

The window keeps each machine's world in the state folder (on Windows `%LOCALAPPDATA%\Twinwright\Studio`): the first machine's in `world`, the others' in `machines\<id>\world`. `--world DIR` uses another folder; `--world-reset` starts from the factory state. Batch runs, tests, replays, recordings played back and the AI's background machines keep no world unless given `--world`. One program at a time opens a world; a second one starts from the factory state and saves nothing.

File	What it holds
`world.json`	The session's end (below), written to `world.json.tmp` first, then renamed
`stock.snap`	The blank's cells, cut or not (binary, `TWSTOCK1`)
`eeprom.bin`	The controller's stored memory: its settings, offsets and tool data, written through as it changes

`world.json`:

Key	Meaning
`world`	The format's version: 1
`saved_at`, `saved_epoch`	When it was saved: UTC, and seconds since 1970
`version`	The version of Studio that saved it
`pos`	Where the axes stood, mm (X, Y, Z)
`tool`	The tool in the spindle, as a spec (`flat 6 n3 L20`)
`tool_table`	The job's tool table: `{"1": "flat 6 n3 L20", ...}`
`tool_auto`	The twin plays the operator at a tool change: it puts in the tool from the table and presses cycle start
`setup`	The workholding and the blank, as the window's Save setup writes them (null: none)
`cells`	`stock.snap` holds the blank's cells
`thermal`	The heat: `screws` (each screw over the frame), `frame` (over 20 °C), `ambient` (the room over 20 °C), `part_heat` and `part_offset` (the workpiece over the frame: what the cuts put in, and an offset given), all in K; `spindle_um`, the spindle's growth in µm
`rtc`	The board clock's reading, s
`probe_cal`	The touch probe as last calibrated: the ball's diameter, its effective radius, the lobing and its phase, the ring gauge's diameter, the largest deviation, when and by whom (only once it has been calibrated)
`odometer`	`travel_mm` and `reversals` of each axis, `spindle_h` (the spindle's hours) and `on_h` (the program's hours), added up over the sessions

At the next start the axes are where they stood; the same blank comes back with its cells, and the tool and the table; the heat has cooled for the time the machine was off, each part towards its rest by its own time constant; the board clock has run on. `--world-off S` says how long the machine was off (for tests; otherwise the time since it was saved). `--pos`, `--tool`, `--stock`, `--setup` and `--part` given at the start win over the archive.

Licence files

A licence file (`.twlic`) is a JSON object in UTF-8, signed with Ed25519. Studio checks it on the computer, without the network. Where the files live and how they are used: Licences, seats and activation.

This example is from the shared test vectors. It is signed with the test key (`kid test`), which release builds do not accept:

```
{"v": 1, "kid": "test", "id": "TWL-TEST000001", "edition": "pro", "kind": "subscription", "seats": 1,
 "holder": "Weber Lab, TU Example", "issued": "2026-11-02T10:00:00Z", "valid_until": "2027-11-02T10:00:00Z",
 "renew_after": "2027-10-28T10:00:00Z", "updates_until": null, "fp": ["0123456789abcdef0123456789abcdef"],
 "mc1_serial": null,
 "sig": "h3FTqZnDYtq8Q7_0ZuELLeDrTfyZnkTPgvDWSywTyl7BCJJxGIyhrBQUjYHJiE1_WOWaFarqbveFiMLrEfUyCA"}
```

Fields

Field	Type	Meaning
`v`	integer	The format's version: 1. Studio treats a file of another version as not valid.

Field	Type	Meaning
`kid`	string	Which of Twinwright's keys signed the file
`id`	string	The licence's number: `TWL-` and 10 letters and digits. Renewal sends it.
`edition`	string	`maker` or `pro`
`kind`	string	`subscription`, `perpetual` (bought outright) or `owner` (MC-1 owner)
`seats`	integer	The number of seats
`holder`	string	Who the licence is for: a person or an organisation. Never an e-mail address.
`issued`	date and time	When the file was signed, UTC, `YYYY-MM-DDTHH:MM:SSZ`
`valid_until`	date and time, or null	A subscription: the end of the paid period. Bought outright: the end of the refund period in the first file, null in the permanent one.
`renew_after`	date and time, or null	From when Studio tries to renew the file
`updates_until`	date, or null	Bought outright, MC-1 owner: the versions released up to this day have the paid features
`fp`	array of strings	The computers the licence is bound to: the hashes of their fingerprints, up to three a seat
`mc1_serial`	string, or null	MC-1 owner: the machine's serial number. Studio does not check the machine when it runs.
`sig`	string	The Ed25519 signature of the other fields, base64url without padding

A field Studio does not know is part of the signed bytes, and Studio ignores what it means.

The signed bytes

The signature covers the file without `sig`, written in a canonical form:

- keys in Unicode code point order;
- no whitespace;
- strings with JSON's shortest escapes;
- integers without a decimal point; no other numbers, and no key twice.

Studio writes the file in this form before it checks the signature. Spaces, line breaks and the order of keys in the file therefore do not matter. A changed value makes the check fail.

The fingerprint's hash

The hash is the first 32 hexadecimal digits of SHA-256 over the bytes `twinwright-fp-v1`, a zero byte, and the computer's ID:

System	The computer's ID
Windows	The value `MachineGuid` under `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography`
Linux	`/etc/machine-id`

Without either, Studio makes a random ID and keeps it in the licence folder (`machine.id`); deleting it makes the computer a different one. The ID itself never leaves the computer.

For example, the ID `8b1c3a0e-5f2d-4c7a-9e61-2d0f4b7a1c55` gives the hash `0a2d31bb82176516415efa7b3a340562`.

The request code

The request code for offline activation is the canonical UTF-8 JSON `{"fp": <the hash>, "version": <Studio's version>}`, in base64url without `=` padding. For example, the hash `0123456789abcdef0123456789abcdef` and version 1.0.0 give:

```
eyJmcCI6IjAxMjM0NTY3ODlhYmNkZWYwMTIzNDU2Nzg5YWJjZGVmIiwidmVyc2lvbiI6IjEuMC4wIn0
```

`TwinwrightStudio.exe --licence-request` prints this computer's request code.

Trial files

A trial is a licence file like the others, signed by Twinwright's licence server: `"kind": "trial"`, this computer's hash in `fp`, and `valid_until` 30 days after the trial began. It is never renewed. A trial file changed by hand no longer holds.

The machine archive

Every key of a machine's archive, with the MC-1's values: Machine profile schema.

Sender software with the Twinwright controller

This page is for people who want to drive the MC-1 from a G-code sender of their own (bCNC, gSender or Universal G-code Sender) instead of from Studio. The MC-1's controller is the Twinwright controller (kernel), compatible with the Grbl 1.1 serial protocol and the extensions current senders use, so these senders work with it, with the few settings and caveats below.

Everything here was tested on the MC-1 digital twin, which runs the same controller code as the machine. Checking it on a real machine is still to come. Tested versions: bCNC 0.9.16, gSender 1.6.4, UGS Platform 2.1.27.

Connect

Where	How
The twin in Studio's window	Studio takes one sender at a time on `localhost:2323` (this computer only; start Studio with `--bind any` to reach it from the network). bCNC: port `socket://localhost:2323`. gSender: Ethernet, address `127.0.0.1`, port `2323`. UGS: `tcp://localhost:2323`.
The MC-1 itself	Native USB: the controller shows up as a serial port (COM*n* on Windows). Pick that port. The baud rate does not matter on native USB, but some senders want one: 115200 works.

What the controller says about itself:

- Its welcome line is `Grbl 1.1f ['$' or '$HELP' for help]`, like a classic Grbl controller. Senders look for a welcome line that starts with "Grbl" to notice that the controller restarted (see section 3).
- Its name is in the answer to `$I`: `[VER:Twinwright 1.1f.…]` and `[FIRMWARE:Twinwright]`. Senders that show the firmware show "Twinwright".

Pick the firmware in the sender

Sender	Setting	Why
bCNC 0.9.16	Controller: **GRBL1**	bCNC does not read the firmware's name. Tested with GRBL1.
gSender 1.6.4	Default firmware: the choice other than **Grbl**	gSender recognises a firmware by its name in `$I`. Our controller gives its own name, so gSender uses the firmware you set as the default. Tested with that choice.
UGS 2.1.27	Firmware: **GRBL**	Tested: settings, programs, alarms and homing after a reset, with no extra Soft Reset (section 3). UGS's other firmware choice works as well, but needs one more Soft Reset after a reset; jogging was tested with that choice only.

After an alarm or a reset

- **E-stop.** The MC-1's E-stop opens the power contactor. Release it, press the machine's Start button, then reset in the sender and home. Until Start is pressed the alarm (ALARM:10) does not clear.
- **ALARM:11 (homing required).** Home (`$H`). Unlocking with `$X` is refused (error:46).
- **ALARM:2 (soft limit).** The program asked for a move outside the travel. Homing leaves Z at the top of its travel (machine Z 0), so any move above it, such as `G53 G0 Z5`, raises ALARM:2. Reset, then home or unlock.
- **bCNC** recovers after any reset: it sees the "Grbl" welcome line and forgets the lines that were still on their way, so `$H` goes out.
- **bCNC after Stop.** bCNC's Stop is a feed hold, then a reset. Afterwards bCNC sends the modes it had before (coordinate system, units, tool length offset) back to the controller in a line or two, and in a few cases the controller refuses them, as other Grbl controllers do: with a tool length offset (`G43.1`) in effect the line gets error:24 (bCNC sends a `G43.1Z...` of its own after its first Stop, so from the second Stop on it always happens); under `G80`, `G43.1Z...` gets error:31; with scaling (`G51`) in effect, `G51:X` gets error:1. After an error every G-code line that follows gets the same error until an empty line, a `$` command or a reset. So after a Stop, check the coordinate system, the units and the tool length offset with `$G` and `$#` before you jog or carry on. If they are wrong, set the zero again and measure the tool length again (tested on the twin, 2026-10-06).
- **UGS with its other firmware choice.** After a reset that UGS did not send itself, for example one from Studio's window or the controller restarting by itself, press **Soft Reset once more** in UGS before Home. Until then Home does nothing. With that choice UGS treats only another welcome line as a restart. A reset from UGS's own Soft Reset button is fine. On the MC-1 itself such resets are rare: it has no reset input and one USB host. With the firmware set to "GRBL" this does not happen: after such a reset UGS shows CONNECTING, and Home works at once (tested on the twin, 2026-10-06).

Tool changes (M6)

None of the three senders, with their default settings, lets the MC-1 change the tool and measure the new one:

Sender	Default	What to do
bCNC	Skips every M6 line: the next tool's cuts are made with the tool still in the spindle.	Set its tool change to **Manual tool change (TLO)** and measure on the MC-1's tool setter (section 5).
gSender	Tool change "Ignore": M6 is sent as a comment and the program does not stop.	Set its tool change to **Fixed Tool Sensor** at the MC-1's tool setter (section 5).
UGS	Handles M6 itself: it retracts, stops the spindle and asks you to change the tool, but it never measures it. A longer tool then cuts too deep, and on a real machine it can crash into the part.	Run one tool per program and set the tool's length or work zero before each, or use Studio for programs with tool changes.

More about UGS and M6:

- It also stops at an "M6" inside a comment, such as FreeCAD's `( M6 T1 )`.

- A second M6 that comes right after a tool change can slip past it to the controller. The controller then waits in the Tool state for a confirmation that UGS never sends, and no button in UGS's window works any more. On the MC-1 only the E-stop ends it: press it, release it, press Start, then Soft Reset and Home in UGS.
- If the controller is reset during a UGS job before UGS has reached an M6, UGS may report the job as complete, but its tool-change dialog still comes. Dismissing it can start the whole program again from the beginning. After a reset in the middle of a job, close the file in UGS before you go on.

Measuring tool length on the MC-1's tool setter

The tool setter is at the back left of the table; `G59.3` is stored right above it.

- **bCNC, Manual tool change (TLO).** Tool change position above the tool setter. Probe position `G59.3`. Probing distance 50 mm, seek 200 mm/min, slow 25 mm/min (the MC-1's `$342`–`$344`), command G38.2. Measure the first tool with "Calibrate" before running the program. bCNC then sets `G43.1` for each new tool.
- **gSender, Fixed Tool Sensor.** Sensor location `G59.3`. Measure the initial tool in the wizard. gSender then shifts the work Z for each new tool (`G10 L20`) instead of using a tool length offset. It takes the length from where the axis stopped after the touch, not from the trigger point the controller reports, so each change can be a little off. In the twin, run tool changes at 1× speed: gSender reads that position after a fixed short pause, and a twin running faster can give it one from before the stop.
- **Start at least 0.1 mm below the top of Z** (machine Z −0.1 or lower) when the tool change begins. Both senders return to that height after measuring. Starting right at the top, the small scatter of a measurement can put that return above the soft limit, and the job stops with ALARM:2 before it cuts.

Setting work zero with a touch probe

- **UGS** (its probe module) works out the zero from the trigger points the controller reports (`[PRB:]`). This is the right way.
- **gSender** (3D probe) and **bCNC** (its probe page, then its Z=0 / X / Y buttons) set the zero where the axes stopped after the touch. The machine always runs on a little past the trigger before it stops, and the faster the final touch, the further. Use a slow final touch, or set the zero with Studio's probe cycles, which use the trigger points.
- **UGS's hole centre** (its probe module's Hole Center): from where you leave the probe it probes -X, +X, -Y and +Y, each by half the hole diameter you type in, and sets the centre midway between the slow touches' trigger points. How far it reaches depends on that diameter: the start may be at most (typed diameter - actual bore) / 2 + the ball's radius from the centre. With the bore's own diameter typed in, that is only the ball's radius (1 mm for a 2 mm ball); type it a few mm larger (24 to 26 for a 20 mm bore) to allow for placing the probe by eye.
- **When UGS's hole centre does not reach the wall** (finding 192, UGS's own behaviour): that probe ends in ALARM:5 and UGS stops. But it set G54's X0 Y0 at the start point before probing, and as the cycle did not finish, G54 stays there. After unlocking, set the zero again before cutting. Its probe cycle also stays half-way: disconnect and connect again before probing again. It also takes a point the controller reports as not touched (`[PRB:...:0]`) for a touch.
- Calibrate the probe first (Studio's probe cycles: a ring gauge). An uncalibrated probe reads every bore larger and every boss smaller by twice its pretravel.

Other things to know

- **bCNC over the network.** On Windows, bCNC reads a TCP connection only about every 2 seconds, so its display can be 2 seconds old. "Idle" in bCNC does not mean the machine has stopped. A USB connection does not have this delay.
- **gSender, M0.** An M0 within the first 10 lines of a program does not pause (gSender turns it into a comment). M1 always pauses.
- **gSender, continuous jog.** Holding a jog button is refused on the MC-1 (error:15): gSender asks for a long jog beyond the travel. Jog in steps. On some Grbl controllers, `$40=1` (limit jog commands) ends long jogs at the soft limits instead of refusing them; the MC-1's controller does not act on that setting yet and still refuses them (tried with gSender on the twin, 2026-10-06).
- **UGS, long lines.** UGS never sends a line longer than 127 characters: the job stops without a message. Keep lines shorter (UGS removes comments first).
- **UGS, M30.** UGS removes M30. A program that relies on M30 to stop the spindle and coolant leaves them running: end programs with M5 and M9 before M30.
- **UGS, comments.** If a line has two comments, UGS also removes the code between them.
- **Disconnecting during a job.** The controller keeps running the lines it already has. gSender and UGS send no stop when you disconnect, so the machine runs on, with the spindle turning. Stop the job before you disconnect. bCNC holds and resets before it closes the connection.
- **Comments with non-ASCII characters** (such as Chinese, "Ø", "–"): all three senders remove comments before they send, so these characters never reach the controller. The controller itself reads UTF-8 text in comments and in `(MSG,...)` as text, never as a real-time command.