PathDxf2GCode

Utility for graphical definition of milling paths

H.M.Müller, Version 1.3, March 2025

Translated with the help of https://DeepL.com

Contents

1. Problem

A CAD program is typically used to design the results of a manufacturing process. In a subsequent (or parallel) step, one must then also describe that process itself, which might require additional designs—for example special tools, intermediate manufacturing results or, in the case of CNC manufacturing, „CNC programs“ that control the movement of such machines.

Nowadays, the construction typically results in a DXF file, which must then be converted to a G-code file. Of course, this conversion depends on the specific manufacturing process. Among other factors, one has to consider

- the type of CNC machine
- its properties
- the materials used
- the necessary manufacturing precision:

For the same final result, a 3D printer needs a different G-code file than a CNC milling machine or (if manufacturing is possible with it) a CNC lathe. But not only the machine is relevant: If a CNC milling machine is to mill a component out of block material, different G-code is required than for manufacturing from profiles. Finally, many specific parameters of a CNC machine determine the creation of correct and efficient G-code.

The basic question I had to answer for my mechanical designs was, then: Could I use a standard program for G-code creation, or would I need a homegrown solution?

2. Manufacturing process

To answer the question at the end of the previous section, it is necessary to talk about my manufacturing process, which is once again influenced by the final results I want to manufacture. Here are relevant properties of this process:

1. I want to build special mechanical *models* which are assembled from somewhat standardized *components*.
2. Due to the fact that the number of component designs is quite large (100s), I do not want to mass-produce them; rather, after designing a specific model from the components, they are specifically manufactured for that model.
3. The components are milled from a few types of aluminum profiles (angle profiles and flat sections). The resulting G-code must be based on milling these profiles.
4. Most components are quite small (about 100 mm in length, often below 50 mm). One milling run on the CNC router can therefore create a medium number (10...30) of components. Thus, the G-code program for a single run must be assembled from the designs of multiple components.
5. Some components need two milling runs; e.g., components milled from an angle profile require a separate run for each side of the profile.
6. In rare cases, it is necessary to change the milling bit between two runs on the same components (e.g. if both a straight bit and a T-slot bit are necessary).

3. Solution concept

Above all, the profiles that I want to use on my CNC milling machine are specific to my designs. There are certainly professional G-code generators that can be taught to use profiles; I'd rather not know the license costs on the one hand, and the configuration effort on the other ... (which may be wrong, because I'm overlooking an elegant solution; nevertheless, I'll take a chance; I could perhaps do some further research here: How to Convert DXF to G-code: 4 Easy Ways | All3DP).

Moreover, I only trust automatically generated G-code to a limited extent. Later, when I really understand the behavior of my machine, I will get involved—but for now I want to have every movement of the machine “specified by myself”.

That's why I've made my decision: I want to write the G-code generator myself.

However, I don't want to use more or less “intelligence” to back-calculate the milling paths from the final description of a component: I want to design these paths by hand too, using the same CAD tool that I use for component design:

- Primarily because it is easier;
- but also because of the aforementioned “full control” requirement.

The rough requirements for my manufacturing process and then the G-code generator look like this:

a) Representation of all path elements required for the milling paths in the CAD program; currently these are:

- straight segments,
- circular segments and
- drilled holes.

b) Representation of all values required for the milling paths (e.g. depth of a hole, height of a sweep) via parameters of graphic objects that

- can be easily specified via the CAD program,
- are visually distinguishable there; and
- can be read stably by the program from the generated DXF file;

c) for each path, the addition of start and end points and a readable designation that can be used to link paths in a compound drawing.

What I do *not* require is adaptability to different cutter diameters: the path must be redesigned for a different cutter.

The following two chapters describe

- the process to generate the G-code files from a user perspective (chapter 4);
- and then the internal structure of the G-code generator (Chapter 5).

4. User documentation

4.1. Main process

The design and manufacturing process with PathDxf2GCode is as follows:

4.1.1. Designing components

The components are constructed[1]. The layers[2] are designated with numbers without letters at the end, e.g. 2913 or 2913.4.

Separate path drawings are then created for each “path dimension”. As a rule, only one path drawing (one mounting) is required for components made of flat profiles, while two path drawings are required for components made of angle profiles (which I usually designate as L and R). Several path drawings are also necessary for components that require several different milling cutters (e.g. wheels with a groove created with a T-milling cutter).

4.1.2. Constructing the project

For my mechanical models, I will create a group of components together in one milling pass. Therefore, a *compound project design* is required that connects the milling paths for all these components.

For this purpose, project path drawings are created in a new CAD file for each path dimension, which arrange the respective paths of all components to be produced on the sacrificial plate (mounting plate) and link them one after the other. The project path drawings also show the dimensioned arrangement of the profiles on the mounting plate as well as the position of the milling coordinate check point and origin.

If only one component is created, the component path file is sufficient as input for the G-code generation.

4.1.3. Exporting path files

The path drawings are exported as DXF files. The directory structure can be arbitrary (see also /d parameters on p. 23), but two structures have proven themselves in practice:

- For projects without pre-constructed components, a new directory is created, usually with the date in the directory name. All necessary path files are stored there.
- For pre-constructed components and projects assembled from them, you create
  - the component paths in a component directory (or a few),
  - the project path drawings in a project directory as above.

For each DXF file, a corresponding PDF file is also created and stored. I need this for preparing and mounting the profiles, checking Z probe positions and overall visual control of the milling process.

1 I use the (old-fashioned) Becker-CAD 2D software, creating „classic“ two-dimensional projection drawings.
2 In Becker-CAD, layers are called „Folien“.

4.1.4. Generate G-code

PathDxf2GCode is called for each exported DXF file (see p. 23). The result is a G-code file with the same name but with file extension `.gcode`.

If errors occur during conversion, the path construction must be corrected in the CAD program (see p. 23) and exported again.

4.1.5. Milling

For each G-code file

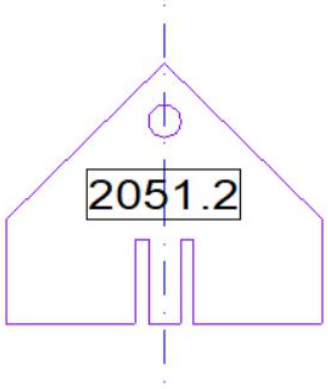
- the profiles are mounted or remounted accordingly (the PDF files with the material descriptions and dimensions are used for this manual work),
- the coordinate origin of the milling machine is set,
- and finally the milling process is started.

4.2. Path drawings

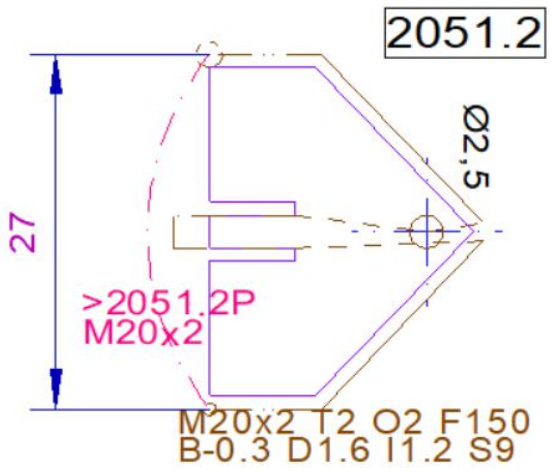
Path drawings are created for individual components. If—as described on p. 7—these are combined into projects, path drawings are also created for these (which then contain partial path embeddings for the components—see p. 16). This section describes the general elements of path drawings; their use is described in separate sections on components (p. 17) and projects (p. 19).

4.2.1. An example

Here is an example of a construction—in this case a component (drawing no. 2050+2051 K; K stands for “construction”):



The drawing for the associated milling path (here a component path) looks like this (drawing no. 2051 P1; P is the abbreviation of “path”):



Some important elements of each path are directly recognizable here:

a) The start of a path (under the text M20x2...) is marked by a small circle with a diameter of 1mm. A number of parameters must be specified there, including the material (flat stock with dimensions 20x2), the upper edge of the profile (here T2 = 2 mm above the zero plane), the cutter diameter (O2=2mm) and others.

The basic idea is that all essential parameters for milling are described at this one point.

- Die grundlegende Idee ist, dass alle wesentlichen Parameter für das Fräsen an dieser einen Stelle beschrieben werden.

b) For contour milling paths—especially on straight edges—the component path is drawn along the component at the radius distance of the milling cutter (here 1 mm, due to the 2 mm end mill).

c) Holes are drawn in the path with their actual diameter (the example shows a 2.5 mm hole at the tip of the component).

d) The end of the path is marked by a circle with a diameter of 2 mm.

e) The start and end markers are arranged so that the component path can be "stacked": A copy of the component path—or another path for the same profile—can be placed with its start point on the previous end point. This enables simple construction of the project path drawings (see p. 19).

f) A dash-dotted *proxy line* (_ . _) from the start to the end circle is intended to be copied into a project drawing using this component. The proxy line can be a straight line or an arc, whatever is more pleasing visually. The > sign in front of the path name is explained in the section on subpaths (see p. 16). Further path parameters are specified on the line, which are used to check the embedding in project path drawings (here, a material specification indicated by M).

g) Line types indicate the type of movement:

- ____ Continuous line = milling (milling movements)
- _ _ _ Dashed line = do not mill (sweep)
- _.._.._ Double dotted line = half milling; common uses are:
  - At the edges where the profile continues, this prevents it from being "milled off" and thus losing the mounting. In the example above, this can be seen at the top and bottom of the

non-beveled segments.

- In other places, it is used to mill “markings” into the material for subsequent processing—in the example for the narrow slots to be made with the band saw for the key tube to be inserted.

h) The path diagram contains dimensions of the external dimensions in order to estimate how much material is required and to ensure that the component can be placed within the working area of the milling machine.

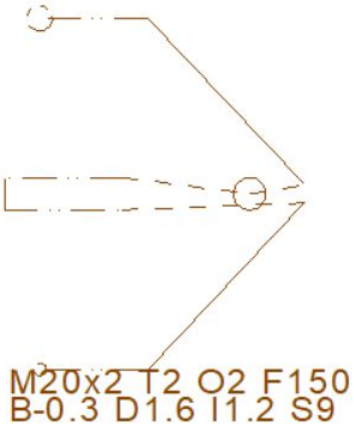
4.2.2. Path layer

What is not directly visible in the path drawing: The path lines and markings (but not the representative line for naming—see p. 16) must be on a layer[3] that is named with the path name, in this case 2051.2.

As quite a few layers are created in a CAD file in this way (one for each milling path, i.e. usually one or two for each component), these path layers must be organized sensibly. I follow the following structure:

- There is a main layer „Pfade“ („Paths“).
- A path drawing for the designs of drawings 1234 K receives a *parent path layer* 1234.** under “Pfade” (the two asterisks indicate path layers, in contrast to construction layers, which are grouped under 1234.*). If a drawing comprises several drawing numbers, e.g. 1234 K and 2345 K, separate construction path layers 1234.** and 2345.** are created for them.
- The *component path layers* for e.g. components 1234.1, 1234.2 etc. are located below the respective parent path layer. For components with only one path layer, the component path layer is named 1234.1P; for components made of angle profiles that require two layers, 1234.2L and 1234.2R are used. In other cases (e.g. several paths due to tool changes), other letters can also be appended. Layer names without letters at the end (e.g. 1234.1) are reserved for the construction layers.

The path can be checked visually by displaying only the component path layer. For the component above it looks like this:



3 In Becker-CAD: „Folie“.

I place the proxy path (the dash-dotted line) in the **-path layer of the component (one could place it in any layer; when copying to a project path drawing, you have to place it in the project path layer there anyway).

4.2.3. Structure of a path construction

Following the exemplary overview in the last sections, here is a complete description of how a path construction must be structured so that it can be processed by PathDxf2GCode.

Each path construction has the following structure:

Path with start and end markers and the following path elements:

1. Sweeps
2. Milling chains with
    - milling segments: Straight lines and arcs, either full-depth or half-depth
3. Drilled holes
4. Helical circles
5. Subpath embeddings
6. Z Probes

Paths and their elements have parameters that control milling. For example, milling segments have a milling depth, milling chains have an infeed, helical circles have a diameter. Some of the parameters are specified directly geometrically (such as the hole diameter), others are determined by texts that lie above the respective element (see p. 11).

Some parameters can be inherited via the path structure. For example, the milling depth for an entire component can be determined for the path; however, if individual holes are to be milled as blind holes, the milling depth can be changed locally.

The following sections describe the path elements and their parameters as well as the options for defining them in detail.

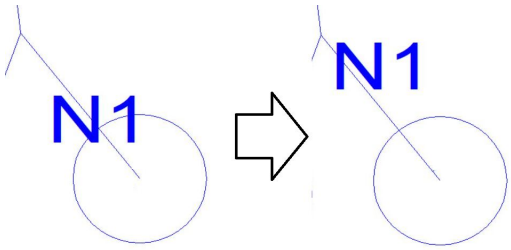
4.2.4. Parameter texts

Parameter texts must be placed in such a way that they overlap “their element”. Care must be taken to ensure that

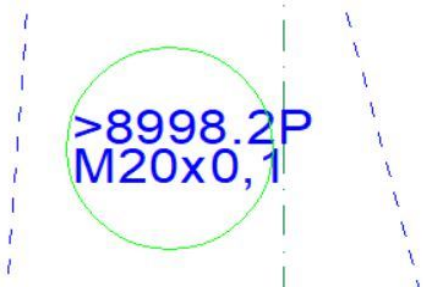
- the text is also in the path layer;
- if possible, only one text overlaps the element. However, PathDxf2GCode attempts to resolve multiple overlaps in this way: A text is assigned to the closest circle if possible; if there is no overlapping circle, to the closest arc; if none is found either, to the closest straight line.
- the overlap is wide enough: The overlap is checked by a circle around the center of the text, which, however, does not extend over the full width of the text, especially in the case of wide texts.

Here are some examples of problem cases and their solutions:

1. In the following example, the text N1 should refer to the line below; however, due to the preference for circles, this only works if the text is placed exclusively above the line, i.e., it does not intersect the circle:



2. The “search circle” (added here by hand) of the following wide text is smaller than the text: Although the P overlaps the line on the right, PathDxf2GCode does not find this assignment. This can be remedied by moving the text slightly to the right:

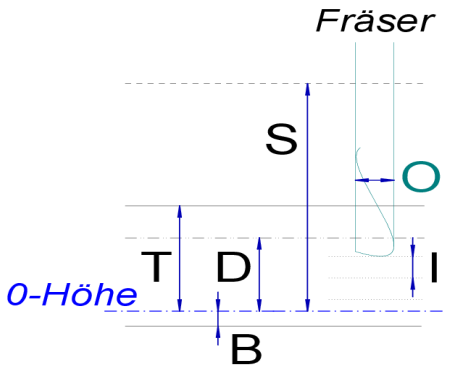


Formally, a parameter text consists of a sequence of parameter definitions, each of which consists of a parameter letter and a value. There must be no space between the letter and the value; the individual definitions are separated by line breaks or spaces.

The images above show definitions of one (N1), two (>8998.2P M20x0.1) and eight (see the image on p. 11) parameters. Most parameters are numerical values; they can be written with a point or a comma as a decimal separator. Material specifications can be any character string, path specifications after > must correspond to the regular expression for paths (see p. 19).

4.2.5. Geometry of the milling parameters

The following diagram shows the geometric milling parameters, which are discussed below. The B value, like all other values, is measured from the 0 height upwards; if “undercutting” is to be carried out as shown here, B must therefore have a negative value („Fräser“ here means „milling bit“, „0-Höhe“ is the zero level of the Z axis:



4.2.6. Paths

A path is the basic element for describing a milling process. PathDxf2GCode can only generate a G-code file for an entire path. A path consists of individual path elements that graphically describe the movement of a milling tool. These elements must therefore connect directly to each other, gaps must be bridged by sweeps. The following are permitted as graphical path elements:

- Straight lines (DXF: LINE)
- Arcs (DXF: ARC)
- Circles (DXF: CIRCLE)

Texts (DXF: TEXT and MTEXT) are also used to control the parameters of the elements. All other elements in a DXF file are ignored without an error message. If such other elements (e.g. splines) are parts of paths, PathDxf2GCode does not see a continuous path and the error message “No further segments found after point ...” is usually displayed (see also p. 24).

The start and end of a path as well as turn markers must be specially marked. Small circles with line type dash-double dash (DXF: PHANTOM) are used for this purpose:

- The start is marked by a circle with a diameter of 1 mm;
- the end is marked by a circle with a diameter of 2 mm;
- Turn marks are marked by circles with a diameter of 1.5 mm (for turn marks, see p. 17).

The following values can be defined at the start of the path (i.e. in a parameter text above the 1 mm start circle) (“TOMBIFDS”):

- T[4]: Material thickness in mm; mandatory if not specified individually for all segments.
- O: Cutter diameter in mm; mandatory.
- M: Material specification; mandatory.
- F: Milling speed in mm/min; must be specified either in the path or as command line parameter /f (see p. 23).
- I: Infeed in mm; must be specified either with the path or individually for milling chains and helical circles.
- B: Milling depth (above 0-level); milling segments, helical circles and drilled holes are milled to this depth if no other B value is specified.
- D: Marking depth; marking segments are milled to this depth unless a different D value is specified.

  If both B and D are specified, then D must be greater than B. This only serves to ensure that D and B cannot be “misused”: B should always be the “deep milling depth”, D always the “high marking depth”. However, this is not checked when specifying B or D on individual segments (see p. 14)—so you can “play tricks” with B and D there.
- S: Sweep height; sweeps are made at this height unless a different S value is specified for a

4 The letters are derived from the English words: T = Top; O is a diameter symbol; M = „Material“; F = „Feed“; I = „Infeed“; B = „Bottom“; D = „Depth“; S = „Sweep“; N = „Numbering“; K = „bacK“.

sweep or a milling segment.

4.2.7. Sweeps

Sweeps are represented by dashed (DXF: DASHED) or long dashed (DXF: HIDDEN) lines. The following parameters can be specified for dashed sweeps:

- S: Sweep height; if not specified, the value for the path is used.
- N: Sequence for branches; see p. 16.

Long-dashed sweeps do not accept parameter texts. They are helpful in situations where a path construction contains many texts, especially for project constructions, which usually only contain partial path embeddings (see p. 16 and 19).

Sweeps can be drawn as straight lines or as arcs. However, in both cases the milling bit will travel in a straight line from the start to the end point. For quicker operation, multiple subsequent sweeps that are at or above the S height of the complete path (which is assumed to be above all possible obstacles) are aggregated to a single sweep.

The speed of a sweep cannot be defined in the G-code, it is a property of the milling machine. However, PathDxf2GCode needs this speed for the statistics calculation, so it must be specified as a /v command line option (see p. 23).

4.2.8. Milling chains and milling segments

Milling segments are the main “workhorses” of a G-code file. As a rule, several milling passes must be made for each milling segment, in which the milling head is advanced by a small infeed each time. PathDxf2GCode combines consecutive milling segments into milling chains and performs the milling passes for each entire milling chain. Milling segments are only straight sections and arcs; a milling chain ends at drilled holes, helical circles, sweeps, and subpaths. Milling chains can consist of several “branches” (“star chains”), see p. 16.

The optional parameters of a milling chain are specified at the first milling segment; they then apply to the entire chain:

- I: Infeed in mm; if this specification is missing, the I value of the path is used.
- K: Sweep height for the return travel between the milling passes. If this information is missing, the S value of the path is used instead.

Milling segments can be drawn with two line types:

- as continuous lines (DXF: CONTINUOUS); for these milling passes, the milling depth is specified with the B parameter;
- or as dash-double-dotted lines (DXF: DIVIDE) for “marking segments”; for these milling movements, the milling depth is defined with the D parameter.

These two segment types can be used to describe frequent cases of milling operations of different depths without too many individual details, e.g.

- milling of markings

- possibly also millings with retaining bars if no marking millings are required.

The optional parameters for individual milling segments are

- F: Milling speed in mm/min; if not specified, the specification for the path or, if this is also missing, the /f command line parameter is used.
- B: (for continuous line) or D (for marking line): Milling depth in mm; if not specified, the specification for the path is used.
- N: Sequence for stars (see p. 16).

4.2.9. Drilled holes

Drilled holes are holes that have exactly the diameter of the cutter (i.e. where the diameter drawn is equal to the O value of the path); they are rarely used. Like milling segments, drilled holes can be drawn either with a continuous line or with a double-dotted line (see p. 14). The following parameters can be specified for drilled holes:

- F: Milling speed in mm/min; if not specified, the specification for the path or, if this is also missing, the /f command line parameter is used.
- B or D (depending on the line type: milling depth in mm (bottom of the hole); if not specified, the specification for the path is used.
- (C—currently not yet supported: Specification of the depth from which G81 should be used instead of G01).

4.2.10. Helical circles

Helical circles are circular millings with an outer diameter larger than that of the milling cutter. The circle drawn indicates the outer diameter of the milling cut; this is helpful for the most common application, milling a hole.

> Note: All other milling paths—especially arcs (ARC)—describe the movement of the milling cutter center; only the helical paths draw the outer edge of the milling cut. This is confusing because it is difficult to visually distinguish a (long) arc from a circle. However, I leave it this way because of the frequent use of helical circles for hole milling.

Helical circles are milled using a spiral movement, which [currently] always takes place in a clockwise direction (G02). When milling out a hole using concentric helical circles from the inside to the outside, this results in up-cut milling.

> If a component is to be milled from the outside and climb milling is to be avoided, then the milling path must be individually composed of arcs that are traversed in the desired direction.

Helical circles can be drawn in the same way as milling segments, either with a continuous line or with a double-dotted line (see p. 14). The following parameters can be specified (“FBI/FDI”):

- F: Milling speed in mm/min; if not specified, the specification for the path or, if this is also missing, the /f command line parameter is used.
- B or D (depending on the line type): Milling depth in mm (bottom of the hole); if not specified,

the specification for the path is used.

- I: Height of a helix in mm; if this information is missing, the I value of the path is used.

For helical circles whose diameter is greater than twice the cutter diameter, PathDxf2GCode does *not* generate G-code for milling away the resulting core—therefore these are *circles,* and not *holes*. If it is necessary to mill away the core (for blind holes or to prevent a milled core from jamming or being thrown around), then several helical circles with increasing diameters must be constructed. PathDxf2GCode mills such concentric helical circles from the inside to the outside in any case.

4.2.11. Subpath embeddings

To mill several previously constructed components, their path drawings must be able to be referenced in project drawings. This is done with a dash-dotted line (DXF: DASHDOT)—both a straight line and an arc can be used.

The following options can be specified on the subpath element (“TOMK”):

Am Subpfad-Element können folgende Optionen angegeben werden („TOMK“):

- >: Name of the subpath to be embedded; mandatory. The subpath is searched for in all matching DXF files—see p. 19. The subpath is embedded from the start point to the end point.
- T: Material thickness in mm; mandatory.
- O: Cutter diameter in mm; mandatory..
- M: Material specification; mandatory.
- K: Sweep height for the return run between the milling passes. If this information is missing, the S value of the path is used instead.

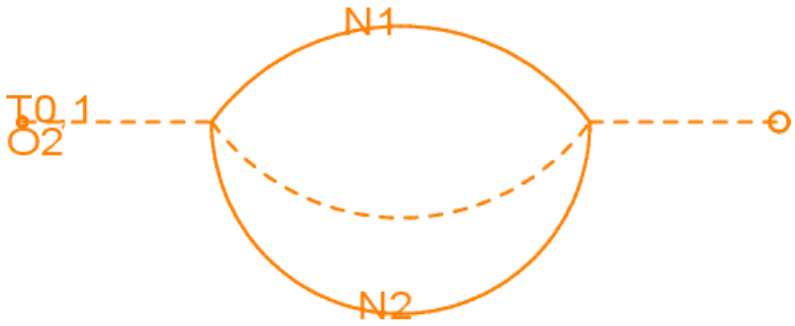
The following conditions apply to the subpath element:

- The specifications T, O and M at the start circle of the embedded subpath must match the corresponding values at the embedding point.
- The distance between the start and end points of the embedded subpath must match the distance between the start and end points of the embedding line (i.e., scaling of the embedded subpath is not possible).

4.2.12. Star-shaped paths

Paths can also be “star-shaped”, i.e. they can continue from one point in several directions (with millings or sweeps). So that a unique path can be calculated in such cases, the segments of the paths can be annotated with N texts, e.g. N1, N2 etc. Segments without N are always traveled after the segments marked with N, with shorter segments being placed at the front. Boreholes and helix circles are always milled before all outgoing segments.

In some cases, the numbering with N is a little tricky because a segment connects to two star points. Here is such an example (constructed as a test case):



- After the sweep from the starting point to the left star point, the segment annotated with N1 is milled.
- When the right star point is reached, two sweeps without N could be taken (the straight one to the end point and the curved one to the left). However, the two milling movements annotated with N are tried first. Because the upper one has already been milled and is therefore ignored in the selection, the segment annotated with N2 is added to the path next.
- This milling run now reaches the star point on the left again. As both marked segments are now "used up" (because they have already been milled), the sweep in the middle is now selected.
- At the right star point, only the path to the end of the path remains.

4.2.13. Dead-end paths—turn markings

Paths may also contain "dead ends". At the end of such a dead end, a "turn marker" must be placed—a circle with a diameter of 1.5 mm and line type dashed-double dashed (DXF: PHANTOM)—which indicates that the further path has not simply been forgotten here. From a turn marker, PathDxf2GCode generates G-code that travels back to the last junction from which untraveled path segments start. The following "reverse trips" are currently generated:

- For sweeps and subpath embeddings: Sweep in the opposite direction
- For milling and marking segments: The same milling or marking segment in the opposite direction; the reason for this is to save the time for raising to and lowering from the sweep height for only short dead-end runs.

4.3. Components

4.3.1. Component path DXF files

The CAD drawing must be saved as a DXF file for reading by PathDxf2GCode. As PathDxf2GCode must later find the appropriate DXF file from the name of a component path in the project path drawing, each DXF file name must be able to provide information about the component paths it contains.

Before we look at the structure of a DXF file name, it is important to know that a path name consists of "path name subwords"; the exact decomposition into subwords is determined by a pattern (a "regular

expression") for path names; its default value is ([0-9]+)(?:[.]([0-9]+[A-Z]))?, which allows the following path name patterns:

- number
- number . numbers and capital letters

e.g. 1, 1234, 1234.1A, 1234.2R etc. (but not 1234.1, because the second group must end with a letter).

In detail, the pattern contains two "groups" separated by a dot; the second group including the dot is optional.

A DXF file name is now structured like this:

...path number range{,*path number range*...}....DXF

A path number range has one of the following two forms and meanings:

- *Path number*
  - A path number must match the pattern.
  - It indicates in the file name that the file contains paths that match the specified groups. A path number 1234 with only one group thus indicates that the file contains paths such as 1234, 1234.3A, 1234.12L etc., i.e. paths whose first group is the same as the specified group 1234. A path number 1234.5X in the file name, on the other hand, indicates that this file only contains the path 1234.5X: If a group is specified (such as .5X here), then it is expected that only paths matching exactly this are present in the file.
- *Path number–Path number*
  - Two path numbers according to the pattern, separated by a minus sign.
  - Such a range indicates that the DXF file contains paths in this range. Groups are always compared as text. For example, a file with the name 1234-1236.DXF can contain the path 1234.8A, but also the path 1235.99B or any paths beginning with 1236. A file with the name 1234.5-1234.99Z.DXF—where the trailing group is numeric—may contain the path 1234.8A, but also the path 1234.52B or 1234.99A, but not the path 1234.40A.

The areas of the paths of different files may overlap, e.g. the files 1234.DXF and 1234,1235.DXF and 1230-1239,1241.DXF may exist at the same time. If PathDxf2GCode searches for a component path 1234.2P in this case, for example, it will read and search through all these files. If the path is found more than once, PathDxf2GCode issues an error message.

If several DXF files do not differ in the first part (e.g. because they all only contain paths for components with the same first group 1234), then a sensible numbering is e.g. 1234,A.DXF, 1234,B.DXF etc.

To locate subpath files, the following directories are searched:

- First the directory where the currently processed DXF file is located;
- then all directories specified via /d parameters (see p. 19).

The file name should be entered in the path drawing so that it is selected consistently with the drawing numbers during DXF export without much thought.

4.3.2. Q project

For component paths that are to be used in projects, it makes sense to create a test project that uses all designed paths. The easiest way to do this is to create a project path drawing with export file Paths_Q.DXF for each component path drawing that is exported as DXF file Paths.DXF. In this path drawing

- copy exactly the substitute lines (including overlay texts) from the component path drawing;
- create suitable start and end circles with the necessary properties
- and connect these parts with empty runs.

The exported DXF file can now be checked for problems directly with PathDxf2GCode. In a further step, the components can also be milled in 6 mm plywood, for example, as a test.

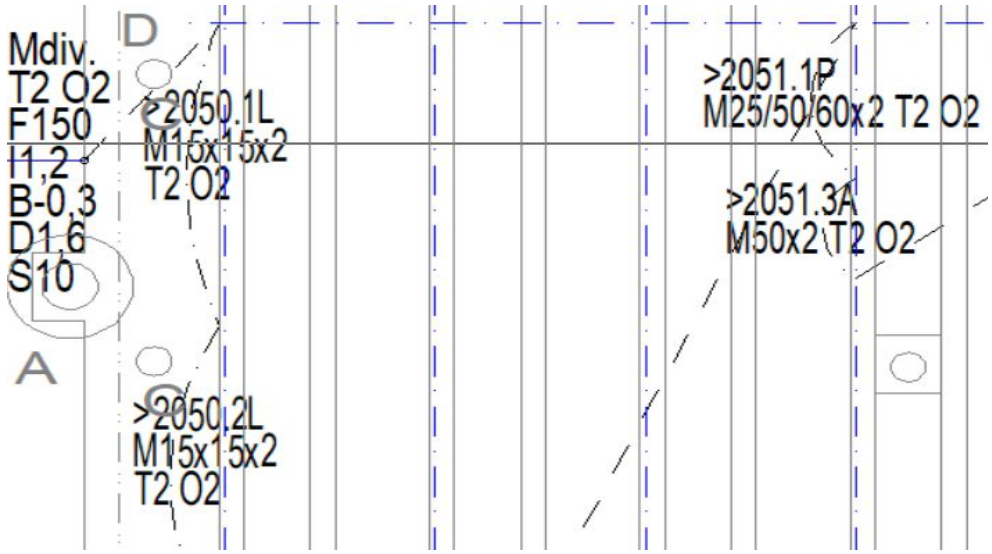
4.4. Projects

4.4.1. Project numbers

As mentioned, path drawings are created for projects. Projects are assigned drawing numbers from 8000 to 8999. Project numbers 8901...8999 are intended for test projects.

4.4.2. Project path drawings and subpaths

Project path drawings are drawn on a copy of the clamping and sacrificial plate. Such a drawing might look like this:



At least the following parameters must be specified for a project path:

- O, because the milling diameter for a project milling pass is fixed; and must be checked against the O specification in embedded paths;
- T, because for safety reasons it must always be clear from which depth drilling (G01) instead of

an sweep (G00) is necessary for vertical runs.

- usually S, because project paths practically always contain sweeps whose height must be defined.

A project path drawing is constructed as follows:

a) Create a copy of the 1084 Ph drawing[5] and rename it. The names of the project path drawings have the following form:

project-number.milling-pass

The milling passes are numbered and letters are added after the milling pass to identify the clamping and/or the milling cutter, e.g.

8001.1P and 8001.2P	for two milling passes of different components
8002.1L and 8001.2R	for two milling passes of the same components with clamping L and R
8003.1LS, 8003.2LT, 8003.3RT	for three milling passes of the same components with end mill (S) and then with T-slot milling cutter (T) for left-hand clamping (L) and then with T-slot milling cutter for right-hand clamping (R).

b) The subpath proxy lines (see p. 16) including the texts assigned to them (in particular the path names) of the components to be milled are placed. The start of a further component path can be placed at the end of the previous one if this is possible in terms of space and the profile.

- The proxy lines including text are copied from the component path drawings[6], but the start and end markings (1 and 2 mm circles) are *not* copied (reason: there may only be one start and one end marking in a path drawing).

c) Unconnected groups of paths are connected by sweeps (usually long-dashed lines).

d) The start point and end point at the far left are connected to the subpaths with sweeps.

e) The drawing is exported as a DXF file; the file name should be the same as the drawing name (e.g. 8001.1P.DXF). In addition, PDFs of the project path drawings should also be saved on the control computer of the milling machine, as they form the working document for clamping and unclamping the profiles and components and, if necessary, for replacing the milling cutter.

4.5. Z-Probes

4.5.1. Basics

I have a problem with the clamping on my milling machine: Even on the dressed clamping plate, the profiles do not lie in one plane, but are up to about half a millimeter lower or higher. This doesn't matter with fully milled profiles, but not with 3D milling, and especially not when a profile is

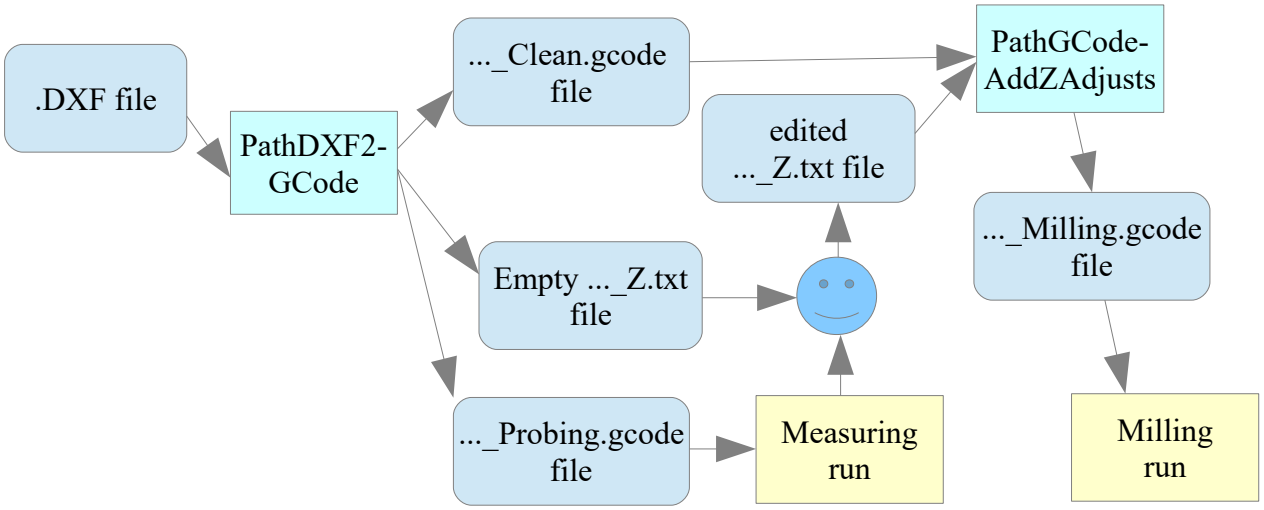
5 In BeckerCAD, a drawing is copied by re-opening the same MOD file and then selecting the drawing with „Hinzufügen“ („Add“). Afterwards, one must manually rename the drawing.

6 In BeckerCAD, in the component path drawing, create a copy of the proxy lines *outside the drawing*, select these copies and transfer them to the project path drawing with „Selektieren → Selektierte Elemente einfügen“; finally, move these copies to the correct position in the project path drawing.

reclamped and finish-milled from the other side. I have come up with the following aid for this:

- A path drawing can contain Z probe points at which the actual Z value is measured by a measuring run before the actual milling process (the milling cutter moves to the points with a G38.2; I have to record the displayed value manually by UGS in a text file).
- In the G-code file that PathDxf2GCode generates, a formula expression is stored for each Z-coordinate that describes the correction of the Z-value as a function of the measured values at the Z-probe points.
- After the measurement run, another program called PathGCodeAdjustZ is used to “recalculate” the G-code file with the values from the text file and the formulas.
- I then carry out the milling process with this improved G-code file.

The entire workflow looks like this:



4.5.2. Path drawing with Z probes

The Z-probe points are marked in the drawing by circles with line type dash-double dash (“PHANTOM”) and diameter 6 mm. They can be located anywhere (i.e. not necessarily on the milling path). PathDxf2GCode calculates the path for the measurement run in the _Probing.gcode file itself by always moving from the starting point to the next Z-probe point that has not yet been visited; at the end, the program returns to the starting point.

The Z-probes can be arranged in two ways:

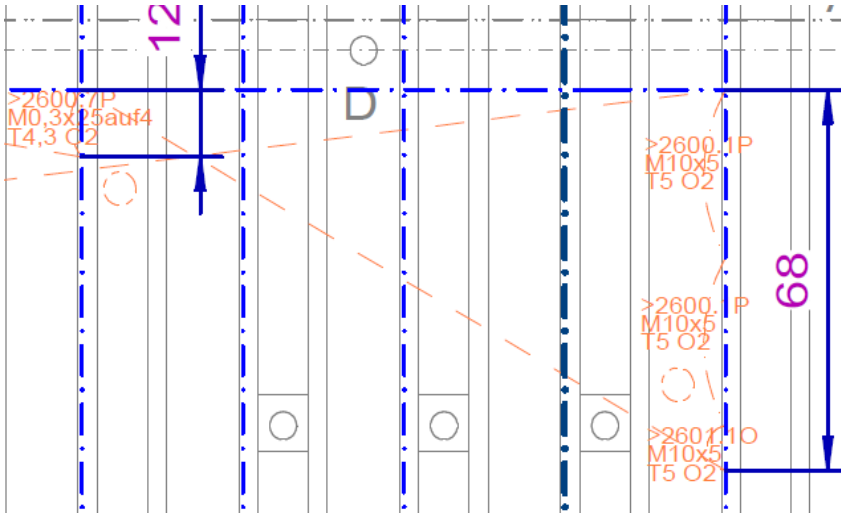
- outside the profiles; in this case, the probe must be placed at the respective point during the measuring run.
- on a profile; in this case, a conductive contact between the profile and the probe must be established during the measurement run.

Two parameters can be specified at a Z probe:

- T: Sensor or material thickness in mm; if the specification is missing, the corresponding path specification is adopted at the starting point.

- L: name of Z probe; this is shown in the _Z.txt file so that one can identify the Z probe locations in addition to their coordinate position.

Here is a path drawing that contains two such Z-probes, each in the vicinity of subpaths:



The resulting _Z.txt file looks like this:

```
@[138.000|299.000] #2001=
@[242.000|264.000] #2002=
```

4.5.3. Measuring run

For the measuring run, the milling cutter must be connected to one of the measuring connections. At the Z-probe points, either the probe must be placed as described above or a conductive contact must be established between the profile and the probe.

After starting the _Probing.gcode file, the milling cutter moves to all Z-probe points in the order in which they are noted in the _Z.txt file. If the milling cutter touches the probe or the profile, it stops for 4 seconds in each case so that the Z value can be transferred to the _Z.txt file. To do this, the Z value read in UGS is entered there manually, e.g. like this:

```
@[138.000|299.000] #2001=5,1
@[242.000|264.000] #2002= 5.25
```

Both commas and periods are allowed as decimal points; spaces or tabs can be placed before and after the numbers.

4.5.4. Creating the _Milling.gcode file

A PathGCodeAdjustZ call (see p. 24) finally generates the final _Milling.gcode file from the _Clean.gcode and _Z.txt files.

4.6. Calling PathDxf2GCode

4.6.1. Command line options

All DXF files for which G-code files are to be generated are transferred when the program is called:

```
PathDxf2GCode 8001.27.1P.DXF 8001.27.2P.DXF
```

PathDxf2GCode stores the generated G-code files in the same directory as the transferred DXF files.

The following options can also be specified (see also p. 13):

```
/h     Help text
/f 000 Milling speed in mm/min; required
/v 000 Maximum speed for sweeps in mm/min; required
/c     Check all paths in DXF file without writing GCode; if /c is not
       provided the DXF file must contain only one layer path
/x zzz For all texts matching this regular expression, write assigned
       DXF objects; this is helpful for debugging parameter texts
/d zzz Search path for references DXF files
/p zzz Regular expression for path names
/l zzz Language
```

Example calls:

```
PathDxf2GCode /h

PathDxf2GCode /f150 /v1000 /d..\Components “2913 Ph.DXF“

PathDxf2GCode /f150 /v1000 /c /d..\ Components “2050-2051 P.1v.DXF“
```

4.6.2. Problems and their solutions

4.6.2.1. „Pfaddefinition … nicht gefunden.“

Path definition ... not found.

Possible triggers:

- Starting point of a path not provided with line type “dash-double dash” (DXF: PHANTOM) → Solution: Provide starting point with line type “dash-double dash”.

4.6.2.2. „End marker missing.“

Reason: A path does not contain a valid end marker.

Possible triggers:

- End point of a path not provided with line type “dash-double dash” (DXF: PHANTOM) → Solution: Provide end point with line type “dash-double dash”.

4.6.2.3. “S value missing.”, “B value missing.”, ...

Reason: A segment cannot determine the required value.

Possible triggers:

- The value is neither defined at the start of the path nor at the segment → Solution: Define value.

4.6.2.4. „No further segments" at the specified point."

Reason: No further segments connected at the specified point.

Possible triggers:

- End without continuation (e.g. two lines do not meet at a point; or a line does not end exactly at the center of a hole circle) → Solution: Find point in the drawing[7] and connect lines and circle centers exactly to each other.
- Duplicate elements that lie exactly on top of each other; after traveling over one element and returning over the other "it doesn't go any further" → Solution: Remove duplicate elements.
- A line may not be in the path layer → Solution: Find the point in the drawing and move the line to the correct layer.
- Elements are used in the path that PathDxf2GCode does not support (e.g. splines) → Solution: Find the point in the drawing and replace elements from there with supported elements (see p. 13).

4.7. Calling PathGCodeAdjustZ

4.7.1. Command line options

The _Clean.gcode files for which the Z-correction is to be carried out are usually specified when the program is called. The names of the associated _T.txt files and the result file (_Milling.gcode file) are derived from this. Instead of the _Clean.gcode files, the _Z.txt files and also the DXF files from which the _Clean.gcode files were generated can also be specified:

```
PathGCodeAdjustZ 8001.27.1P_Clean.gcode 8001.27.2P.DXF
```

The following options can also be specified:

```
/h     Help text
/l zzz Language
```

4.7.2. Problems and their solutions

4.7.2.1. "Line does not have the format '(comment) #...=value'"

Usually the values are missing after the equal signs.

If you want to mill without Z-adjustments, you can also send the respective _Clean.gcode file directly to the milling machine.

7 In BeckerCAD, one can find the problem location as follows: Select „Point" drawing, then move the mouse or enter the coordinates in the entry dialog.

4.8. Milling run

For the milling process

- the generated G-code files
- and the associated PDFs of the path constructions (both project path constructions and component path constructions)

are transferred to the control computer of the milling machine.

The subsequent milling process is not described here.

5. Program documentation

5.1. Overview

PathDxf2GCode is essentially a 6-pass compiler written in C#, which transforms the DXF input into the G-code output in several phases. All 6 phases are run through completely for each input file; other files read in for embedded paths are only read once and then stored in a cache.

PathDxf2GCode's own code comprises approx. 1250 NLOC, that of PathGCodeAdjustZ approx. 150 NLOC. In addition, there is the (with 19000 NLOC much larger) netDxf library (see haplokuon/netDxf: .net dxf Reader-Writer (github.com)).

5.2. Github project

PathDxf2GCode is hosted on at https://github.com/hmmueller/PathDxf2GCode .

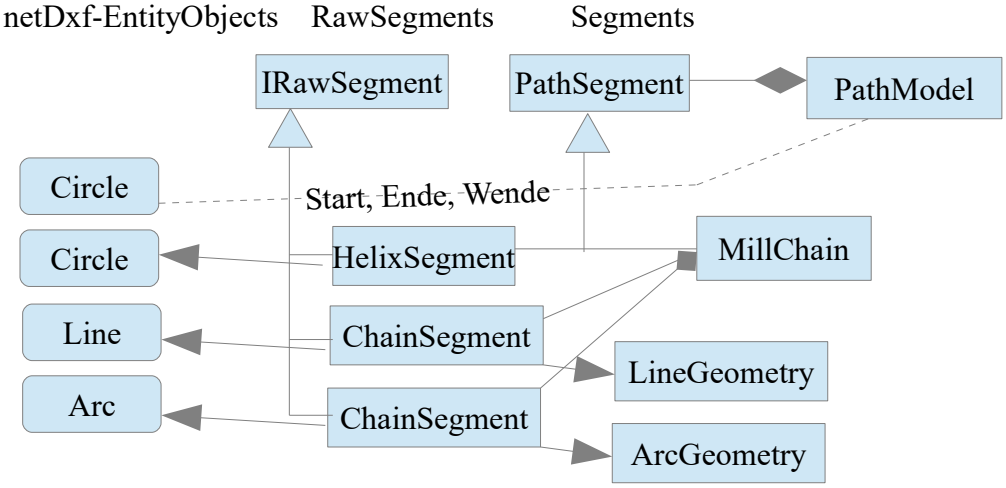
5.3. Compiler passes

The 6 passes are:

1. DXF-Datei → `DxfDocument`; the netDxf library is used for this. The central method of this pass is `DxfFile.LoadDxfDocument`.
2. `DxfDocument` → `RawPathModel`; `RawPathModel` is a temporary representation of the paths from a path drawing. Each path consists of
   - path segments (types: `MillSegment`, `SweepSegment`, `HelixSegment`, `DrillSegment` and `SubpathSegment`),
     - where MillSegments have a `MillingGeometry` of type `LineGeometry` or `ArcGeometry`;
   - Markers (start and end point as well as turn markers)
   - and path parameters.

   The central method of this pass is `PathModel.TransformDxf2PathModel`.
3. combining milling segments into milling chains; the central method for this is `PathModel.CreatePathModel`.
4. `PathModel` → `GCode`-Liste; the central method of this text output is `PathModel.EmitGCode` together with the `EmitGCode`-methods in the `GCode`-hierarchy.
5. Peephole optimizer of the GCode list (currently for summarizing consecutive sweeps); the central method for this is `GCodeHelpers.Optimize`, together with the `Gcode`-class hierarchy.
6. output of the G-code file; the central methods are `WriteGCodes` and `WriteZProbingGCode` in the `Program` class, the `AsString`-methods of the `Gcode`-hierarchy and the `WriteEmptyZ`-methods.

5.4. Basic data structures



5.5. Special data structures and algorithms in PathDxf2GCode

5.5.1. GCodeHelpers.cs

5.5.1.1. Optimize

This method is a peephole optimizer for the generated GCode list. Patterns to be optimized are detected using regular expressions, for which each GCode object is represented by a character (propery `Letter`)
.

- Currently, a peephole optimizer is implemented for the aggregation of subsequent sweeps (with comments in between).

5.5.2. Params.cs

Params-objects have a parent pointer:

- `ChainParams, SweepParams, HelixParams, DrillParams, SubpathParams, ZProbeParams` → `PathParams`
- `MillParams` → `ChainParams`

5.5.3. PathModel.cs

5.5.3.1. CollectSegments

This is where

- the collection of all EntityObjects on paths
- the assignment of parameter texts to objects

- the evaluation of special markers (start, end, turning points, ZProbes)
- the loading of sub-paths

is done to create a `RawPathModel`.7

5.5.3.2. NearestOverlapping<T>, NearestOverlappingCircle, ….Line, ...Arc, CircleOverlapsLine, ...Arc, DistanceToArcCircle, GetOverlapSurrounding

Find objects for the assignment of parameter texts.

5.5.3.3. CreatePathModel

Here, the proper `PathModel` is created from the `RawPathModel`, with the following steps:

A. Concatenation of the segments into a path, including backtracking at turning points

B. Construction of MillChains from successive ChainSegments (Mark and MillSegments)

C. Creating the parameter objects

D. Sorting of the ZProbes

5.5.4. PathModelCollection.cs

Management of all paths read from DXF files.

5.5.5. PathSegment.cs

5.5.5.1. MillChain.EmitGCode

Edges are created in I-spacing for each segment of a MillChain. These edges are then run as close together as possible. If an edge connects at the same height, it is run next: Edges of one level are therefore usually milled before deeper layers are milled.

5.5.5.2. HelixSegment.EmitGCode

A helix is composed of semicircles, each of which moves down by I/2.

5.5.5.5. SubPathSegment.EmitGCode

A Transformation3 is created between the SubPathSegment and the referenced path, which converts the coordinates into the calling model.

5.5.6. Transformation2.cs

The angle between two vectors is only calculated in netDxf using the main branch of arccos. This always results in angle values between 0 and p, which is not correct. The only way I could think of to solve this was to perform a “rotation test”: rotate the first vector by arccos and -arccos and check which rotation actually results in the second vector.

5.5.7. Transformation3.cs

The Z-coordinate is not subjected to the usual transformation, but is adjusted using ZProbes.

5.6. Special data structures and algorithms in PathGCodeAdjustZ

5.6.1. ExprEval.cs

Simple LL(1) parser for a small subset of the G-code expressions used for the Z-adjustments. Both [...] and (...) are permitted as bracket pairs.

5.. Currently known or suspected problems, missing features

None that would particularly bother me. See Github project for details.