PathDxf2GCode

Utility for graphical definition of milling paths

H.M.Müller, Version 0.6, March 2025

English version translated with DeepL.com

Table of contents

1. Problem 4
2. Manufacturing process 5
3. Solution concept 6
4. User documentation 7
   4.1. Work process 7
      4.1.1. Designing components 7
      4.1.2. Construct project 7
      4.1.3. Export path files 7
      4.1.4. Generate G-code 8
      4.1.5. Milling 8
   4.2. Path drawings 8
      4.2.1. An example 8
      4.2.2. Path layer 10
      4.2.3. Structure of a path construction 11
      4.2.4. Parameter texts 11
      4.2.5. Geometry of the milling parameters 12
      4.2.6. Paths 13
      4.2.7. Empty runs 14
      4.2.8. Milling chains and milling segments 14
      4.2.9. Drill holes 15
      4.2.10. Helix circles 15
      4.2.11. Partial path embeddings 16
      4.2.12. Star-shaped paths 16
      4.2.13. Dead-end paths - turnaround markers 17
   4.3. Components 17
      4.3.1. Component path DXF files 17
      4.3.2. Q-Project 19
   4.4. Projects 19
      4.4.1. Project numbers 19
      4.4.2. Project path drawings and sub-paths 19
   4.5. Z-Probes 20
      4.5.1. Basics 20
      4.5.2. Path drawing 21
      4.5.3. Measuring run 22
      4.5.4. Creating the Milling.gcode file 22
   4.6. Calling PathDxf2GCode 22
      4.6.1. Call parameters 22
      4.6.2. Problems 23
         4.6.2.1. "Path definition ... not found." 23
         4.6.2.2. "End marker missing." 23
         4.6.2.3. "S value missing.", "B value missing.", ... 23
         4.6.2.4. "No further segments found from point ..." 23
   4.7. Call of PathGCodeAdjustZ 24
      4.7.1. Call parameters 24
      4.7.2. Problems 24
   4.8. Milling 24
5. Program documentation 24
   5.1. Compiler phases 25

5.2. Class models 25
5.3. Special algorithms 25
5.3.1. Params.cs 25
5.3.2. PathModel.cs 26
5.3.2.1. CollectSegments 26
5.3.2.2. NearestOverlapping<T>, NearestOverlappingCircle, ...Line,.....Arc, CircleOverlapsLine, ...Arc, DistanceToArcCircle, GetOverlapSurrounding 26
5.3.2.3. CreatePathModel 26
5.3.2.4. WriteEmptyZ 26
5.3.3. PathModelCollection.cs 26
5.3.4. PathSegment.cs 26
5.3.4.1. MillChain.EmitGCode 26
5.3.4.2. HelixSegment.EmitGCode 26
5.3.4.3. SubPathSegment.EmitGCode 27
5.3.5. Transformation2.cs 27
5.3.6. Transformation3.cs 27
5.4. Currently known or suspected bugs 27
5.5. Missing features 27
5.5.1. <-embeddings 27

1. Problem

A CAD program is first used to design "components", i.e. the *end results of* a manufacturing process. The next step (or parallel to this) is to describe this manufacturing process itself and possibly provide further designs for it - special tools, intermediate steps of the components during production or, in the case of CNC production, "CNC programs" for controlling CNC machines.

For CNC production, a DXF file typically has to be converted into a G-code file nowadays. Of course, the conversion on the specific production, i.e. among other things,

- with which type of CNC machine
- with which properties
- from which material
- with what accuracy

is manufactured: For the same end result (in of shape), a 3D printer needs a different G-code file than a CNC milling machine or (if production 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 semi-finished products. In addition to these fundamental differences, a G-code generator also needs to know many specific parameters of the target CNC machine in order to generate correct and ideally efficient G-code.

The fundamental question that I have asked myself with my special designs is: Can I use a standard program or do I need a special solution?

2. Manufacturing process

To answer the question from the last section, I first need to describe the envisioned manufacturing process, which in turn driven by the end result. Here are relevant characteristics of this process:

1. What I want to build are individual, specially designed mechanical *models* that are assembled from pre-constructed *components*.
2. Due to the relatively large number of different components, it does not make sense to mass produce them "in stock" - instead, the necessary components are determined during project planning and then manufactured.
3. The components are manufactured from previously known, simple semi-finished products (currently these are aluminum flat and angle profiles). G-code generation can and should be adapted to these semi-finished products.
4. The components are generally small (max. 100 mm long, often less than 50 mm); the CNC milling machine can produce an average number (10...30) of components with one clamping of semi-finished products. It is therefore necessary that the *G-code file for a production run can be compiled* from the design descriptions of several components.
5. Some components must be re-clamped (usually all angle profiles in order to process the two sides of the leg separately).
6. In some cases, a tool change is necessary (from an end mill to a slot cutter).

3. Solution concept

Above all, the semi-finished products 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 semi-finished products; 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).

Furthermore, I only trust automatically generated G-code to a limited extent. Later, 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 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 also want to design these paths by hand, using the same CAD tool that I use for component design:

- Firstly, because it's easier;
- but also because of the aforementioned "default" 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
- Drill holes;

b) Display of all values required for the milling paths (e.g. depth of a hole, height of an empty run) via parameters of graphical objects that

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

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

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

The following two chapters describe

- the preprocess to generate the G-code files (chapter 4);
- and then the internal structure of the G-code generator (chapter 5).

4. User documentation

4.1. Work process

The design and production 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 clamping) is required for components made from flat semi-finished products, whereas two path drawings are required for components made from angle profiles (which I usually designate as L and R). Several path drawings are also required for components that require several different milling cutters (e.g. wheels with a groove created with a T-milling cutter).

4.1.2. Construct project

For the mechanical models, I will create a group of components together in one milling pass. Therefore, a comprehensive *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 manufactured on the sacrificial plate (clamping plate) and link them one after the other. The project path drawings also show the dimensioned arrangement of the semi-finished products on the clamping plate and 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 G-code generation.

4.1.3. Export path files

The path drawings are exported as DXF files. The directory structure can be arbitrary (see also /d parameters on p. 22), but two structures have proven themselves in practice:

- A new directory is created for projects without pre-constructed components, 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 into a project directory as above.

For each DXF file, a corresponding PDF file is also generated and stored.

1 I use the (old-fashioned?) Becker CAD 2D; and design "classically" using cracks, i.e. not with 3D objects. That's just the way it is.
2 In Becker-CAD the layers are called "slides".

4.1.4. Generate G-code

PathDxf2GCode is called for the exported DXF files (see p. 22). The result is a G-code file with the same name but with the extension .GCODE.

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

4.1.5. Milling

For each G-code file

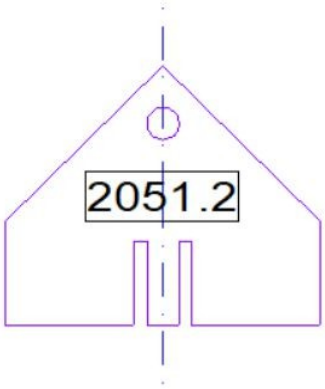
- the semi-finished products are clamped or reclamped accordingly (the PDF files with the material descriptions and dimensions are used for this manual work),
- the coordinate origin of the milling machine is set
- and then the milling pass is carried out.

4.2. Path drawings

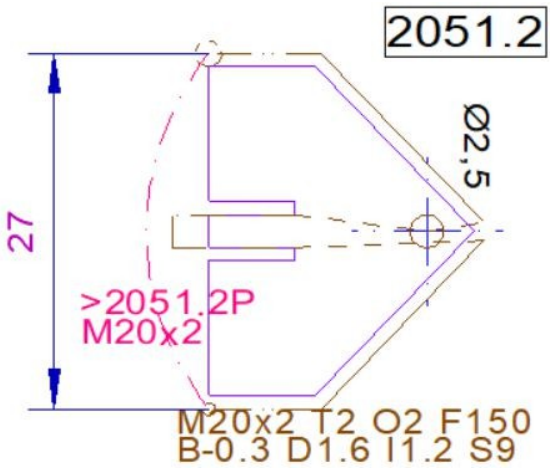
On the one hand, path drawings are created for individual components. If - as described on p. 7 - these are combined into projects, path drawings are also created for these (which then contain partial path embeddings for the components - see p. 16). This section describes the general elements of path 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 stands for "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 1 mm. A number of parameters must be specified at the start of the path, including the material (M20x2), the upper edge of the semi-finished product (here T2= 2 mm above the zero plane = 2mm), the cutter diameter (O2=2mm) and others.

- The basic idea is that all essential parameters for milling are described in this one place.

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).

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

e) 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 semi-finished product - 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 *representative line* between the start and end is used to name the path (this line can a straight line or an arc; this is only a visual question). The > sign in front of the path name is explained in the section on sub-paths (see p. 16). Further path parameters are specified on the line, which are used to check the embedding in project path drawings (here material specification with M).

g) Line types indicate the type of movement:

- _____Milling a continuous line= (milling movements)
- _ _ _ Dashed line = do not mill (empty run)
- _.._.._ Double-dotted line= Half-milling; common uses are:
  - At the edges where the semi-finished product continues, this prevents the semi-finished product from
    is "milled off" and thus loses the clamping. 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 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.

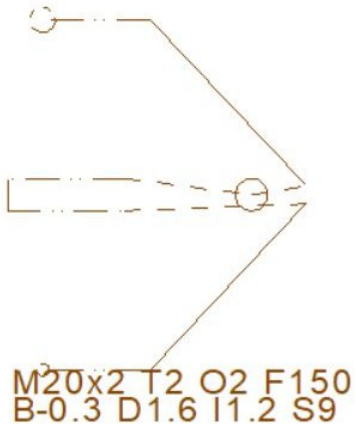
4.2.2. Path layer

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

As this creates quite a few layers in a CAD file (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 "Paths".
- A path drawing for the designs of drawings 1234 K receives a *design path layer* 1234.** under "Paths" (the two asterisks indicate path layers, as opposed to design 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 this.
- The *component path layers* for e.g. components 1234.1, 1234.2 etc. are located below the construction path layers. For components with only one path layer, this is 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 used. Layer names without letters at the end (e.g. 1234.1) are reserved for the design.

By displaying only the component path layer, you can check the path visually. It looks like this for the component above:



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

3 In Becker-CAD: Slide.

4.2.3. Creation of a path construction

Following the exemplary overview in the last sections, here is a detailed 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 path elements:

1. Empty runs
2. Milling chains
    - Milling segments: sections and curves; either full-depth or semi-deep
3. Drill holes
4. Helix circles
5. Sub-path embeddings
6. Z-Probes

Paths and their elements have parameters that control the milling process. 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 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.

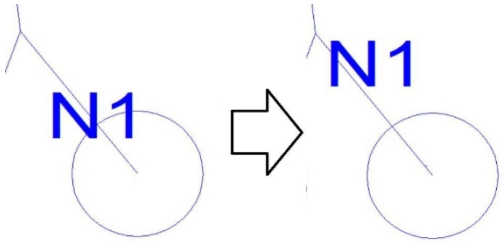
4.2.4. Parameter- Texts

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

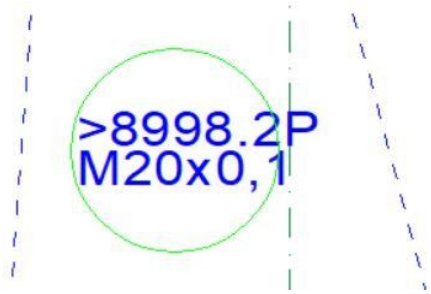
- the text is also in the path layer;
- if possible, only one text overlaps the element. However, PathDxf2GCode attempts to resolve multiple overlaps in this way: A text is to the closest circle if possible; if there no overlapping circle, to the closest arc; if none is found, to the closest distance.
- 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 with wide texts.

Here are some examples of problems and their solutions:

1. In the following example, the text N1 should to the line below; however, due to the preference for circles, this only works if the text is only arranged above the line:



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

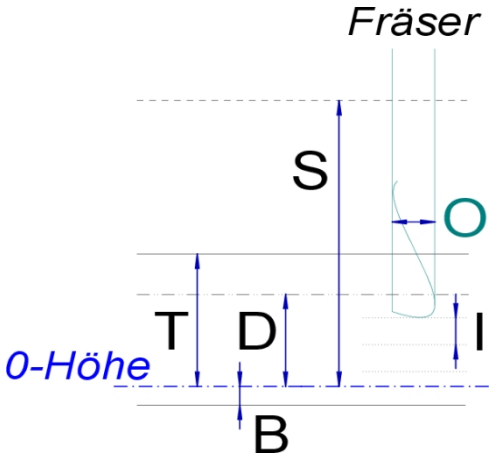


A parameter text consists of a sequence of parameter definitions, each of which consists of a parameter letter and a value. There must no spaces between the letter and the value; the individual definitions are separated by line breaks or spaces.

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 any character string, path specifications according to> and< 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:



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 the movement of a milling tool. These elements must therefore connect directly to each other, gaps must be bridged by empty runs. Graphical path elements are permitted:

- 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 from point ... found*" is usually displayed (see also p. 23).

The start and end of a path as well as turn markers must be specially . 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 through a circle with a diameter of 2 mm;
- Reversible markings using circles with a diameter of 1.5 mm (for reversible markings, see p. 17).

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

- T[4]: Material thickness in mm; mandatory if not specified individually for all segments.
- O: Cutter diameter in mm; mandatory specification.
- M: Material specification; mandatory specification.
- F: Milling speed in mm/min; must be specified either in the path or as call parameter /f (see page 22).
- I: Infeed in mm; must be specified either for the path or individually for milling chains and helix circles.
- B: Milling depth (above 0-level); milling segments, helix circles and drill holes are milled to this depth unless a different B value is specified.
- D: Marking depth; marking segments are milled to this depth if no other D value is specified.

  If both B and D are specified, then D must greater than B. This only serves to ensure that D and B cannot be "misused": B should always be the "deep milling depth", D is always the "high marking depth". However, this is not checked when specifying B or D on individual segments (see p. 14) - so you can "trick" with B and D there.

4 The abbreviations are mainly from English designations: T= Top; O is a diameter symbol; = "Material"; F= "Feed"; I = "Infeed"; B= "Bottom"; D= "Depth"; S= "Sweep"; N= "Numbering"; K= "bacK".

- S: Empty run height; empty runs take place at this height unless a different S value is specified for an empty run or a milling segment.

4.2.7. Empty runs

Empty runs are represented by dashed (DXF: DASHED)[5] or long dashed (DXF: HIDDEN) lines. The following parameters can be specified for dashed empty runs:

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

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

The speed of an empty run cannot be specified 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 call parameter (see p. 22).

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 distance in each case. PathDxf2GCode combines *consecutive milling segments* into *milling chains* and performs the milling passes for the entire milling chain.
Milling segments are only straight sections and curves; a milling chain ends at drilled holes, helix circles and empty runs. Milling chains can consist of several "branches" ("star chains"), see p. 16.

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

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

The milling segments can be drawn with two line types:

- as continuous lines (DXF: CONTINUOUS); the milling depth for these milling movements is defined 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 types of segments can be used to describe frequent cases of different depths of milling without too many individual details, e.g.

- Milling of markings
- possibly also milling with retaining bars if no marking milling required. The

optional parameters for individual milling segments are:

- F: Milling speed in mm/min; if not specified, the specification is used for the path

5 Curved empty runs are not supported because it is assumed that the empty run height is high enough to drive over all obstacles.

or, if this is also missing, the /f call parameter.

- 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. Drill holes

Drill 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, drill holes can be drawn either with a continuous line or with a double-dotted line (see p. 14). The following parameters can specified for drill holes:

- F: Milling speed in mm/min; if not specified, the specification for the path or, if this is also missing, the /f call 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. Helix circles

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

> Attention: 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 motion, 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 synchronized milling is to be avoided, the milling path must be composed of individual arcs that are traversed in the desired direction.

Helix circles can be drawn in the same way as milling segments, either with a continuous or 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 call 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 spiral duct in mm; if this information is missing, the I value of the path is used.

For helical circles with a diameter greater than twice the cutter diameter, PathDxf2GCode *does not* generate G-code to mill away the resulting core - therefore these are also

Circles and no *holes*. If it is necessary to mill away the core (for blind holes or to prevent a milled core from jamming or being thrown away), then several helical circles with increasing diameters must be constructed. PathDxf2GCode mills such concentric helix circles from the inside to the outside in any case.

4.2.11. Partial path embeddings

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

The following options can be specified on the sub-path element ("TOMK"):

- > or <: Name of the partial path to be embedded; mandatory specification. The partial path is searched for in all matching DXF files - see p. 19.> means that the partial path embedded from the start point to the end point;< means that the partial path is embedded from the end point to the start point (i.e. in the opposite direction)[6].
- T: Material thickness in mm; mandatory specification.
- O: Cutter diameter in mm; mandatory specification.
- M: Material specification; mandatory specification.
- K: Empty travel height for the return travel 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:

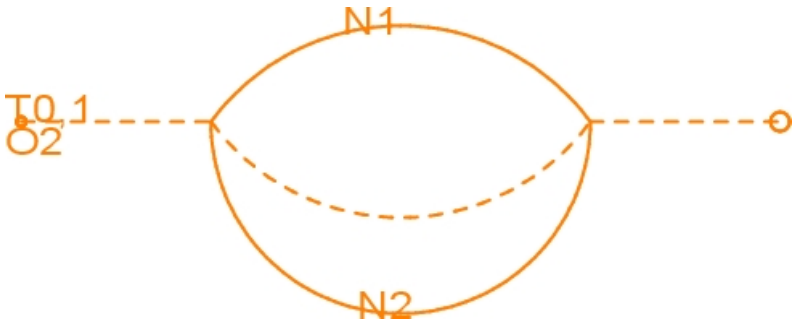
- The T, O and M specifications for the starting circle of the embedded partial path must match the corresponding values at the embedding point.
- The distance between the start and end points of the embedded sub-path must match the distance between the start and end points of the embedding line (i.e.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

4.2.12. Star-shaped paths

Paths can also be "star-shaped", i.e. they continue from one point in several directions (with milling or empty travel). So that a clear path can be calculated in such cases, the segments of the paths can be provided 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, 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):

6 < is not yet supported. The problem with this is, among other things, that milling is then carried out with climb milling instead of up-cut milling.



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

4.2.13. Dead-end paths - turning 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 a double line type (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 empty runs and sub-path embedding: Empty run 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 raising to and lowering from the empty driving 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 filename, it is important to know that a pathname of "pathname subwords"; the exact decomposition into subwords is determined by a pattern (a "regular expression") for pathnames; its default value is `([0-9]+)(o:[.]([0-9]+[A-Z]))o?`, which allows the following pathname patterns:

- Number
- Number . NumbersCapital letter

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 a dotthe 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 correspond to 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 exactly matching it are present in the file.
- *Path number-Path number*
  - Two path numbers according to the pattern, separated by a minus.
  - Such an area indicates that the DXF file contains paths in this *area*. 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 in each case - can 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 meaningful numbering is
e.g. 1234,A.DXF, 1234,B.DXF etc.

The following directories are for sub-paths:

- First the directory where the currently processed DXF file is located;
- then all directories that are 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

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

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

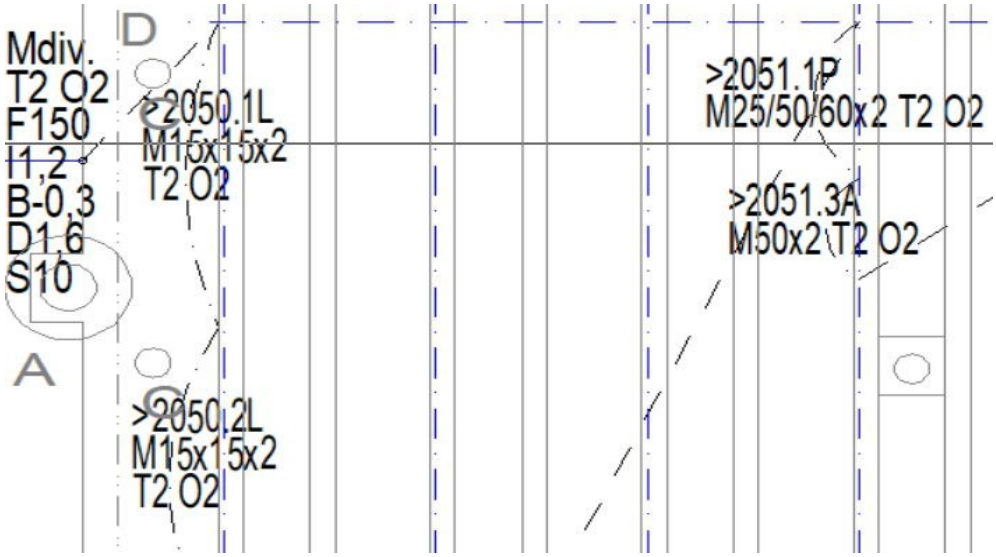
4.4. Projects

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 sub-paths

Project path drawings are drawn on a copy of the clamping and sacrificial plate. This then looks like this, for example:



At least the following parameters must 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 at what depth drilling (G01) is required for vertical travel instead of empty travel (G00).

- usually S, because project paths practically always contain empty runs, the height of which must be defined.

A project path drawing is constructed as follows:

a) Create a copy of the drawing 1084 Ph[7] and rename it. The names of the project path drawings have the following form:

Project number.milling pass

The milling passes are totaled 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 partial path lines (see p. 16) and 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 space and the semi-finished product .

- The substitute lines and text are copied from the component path drawings[8] , but the start and end markings (1 and 2 mm circles) *are not* copied (reason: there can only be one start and one end marking in a path drawing).

c) Unconnected groups of paths are linked by empty runs (usually dashed lines).

d) The start point and end point (outside left) are also connected with empty runs.

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 milling machine's control computer, as they form the working document for clamping and unclamping the semi-finished products and components and, if necessary, for replacing the milling cutter.

4.5. Z-Probes

4.5.1. Basics

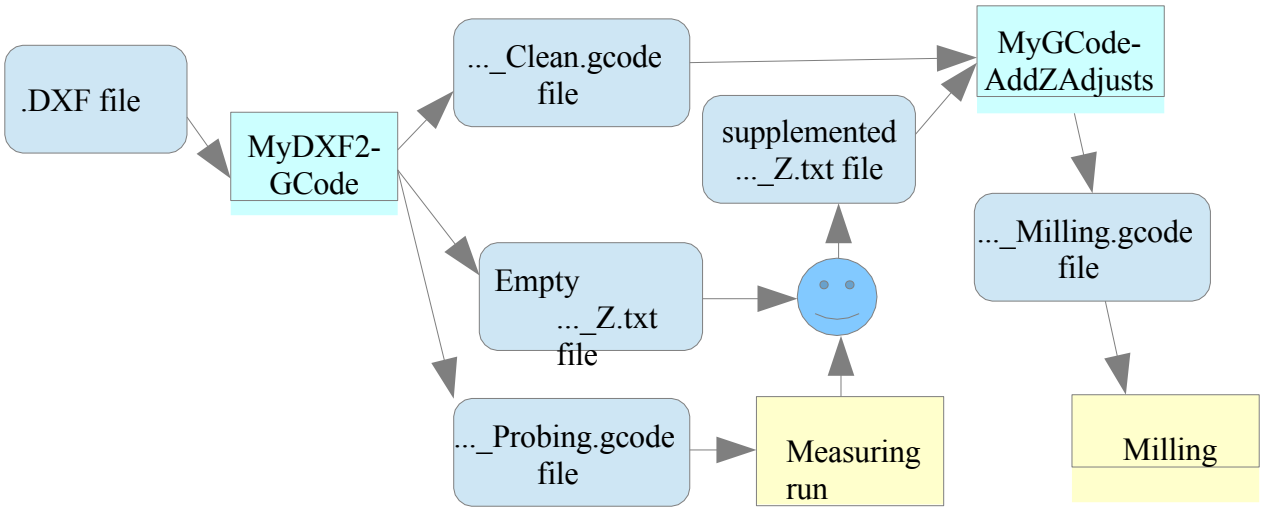
I have a problem with the clamping on my milling machine: Even on the dressed clamping plate, the semi-finished products do not lie in one plane, but are up to about half a millimeter lower or higher. This doesn't matter with fully milled profiles, but not with 3D milling, and especially not when a profile is reclamped and finish-milled from the other side. I have come up with the following aid for this:

7 In BeckerCAD, a sheet is copied by opening the MOD file a second time and copying the sheet to be copied with "Add". It must then be renamed.

8 The easiest way to do this in BeckerCAD is as follows: Copy paths and texts in the component path drawing next to the drawing; then transfer them to the project path drawing using "Select → Insert selected elements"; and either move or copy them there (if necessary several times).

- A path drawing can contain Z test 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 enter the displayed value manually in a text file from UGS).
- 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-sample 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

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 file ..._Probing.gcode itself by always moving from the starting point to the next Z-probe point that has not yet been visited; at the end, it returns to the starting point.

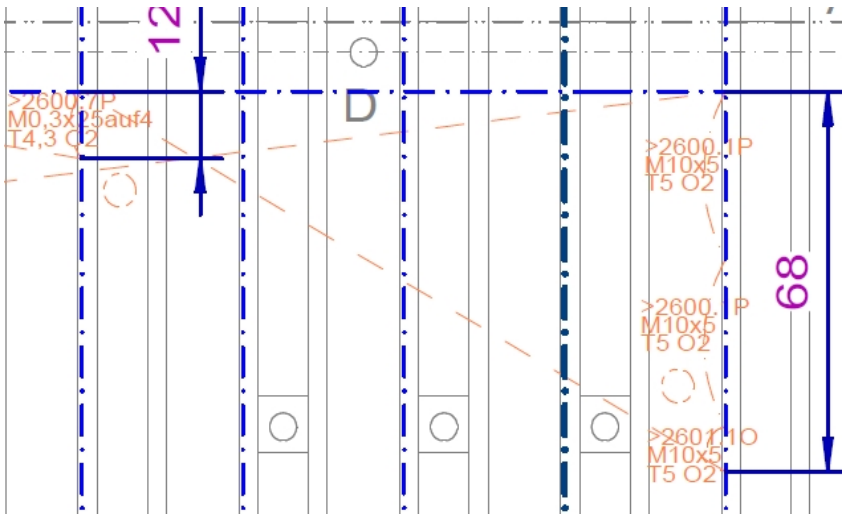
The Z-probes can arranged in two ways:

- outside the semi-finished products; the sensor must then be placed at the respective point during the measuring run.
- on a semi-finished product; then a conductive contact between the semi-finished product and the sensor must be established during the measuring run.

A T parameter can be specified there to inform PathDxf2GCode of the height of the sensor or the semi-finished product:

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

Here is a path drawing that contains two such Z-probes, each near subpaths:



The _Z.txt file 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 semi-finished product 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 semi-finished product, 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 permitted as decimal points; spaces or tabs can be placed before and after the numbers.

[IDEA: Give the points names that then appear in the _Z file?]

4.5.4. Creating the Milling.gcode- file

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

[PITFALLS???? TRY IT OUT!]

4.6. Calling up PathDxf2GCode

4.6.1. Call parameters

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

```
flatkKxf?oCoav %001.?7.1fl.KXF %001.?7.?fl.KXF
```

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

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

```
/k      Hilfv-Atzvigv
/f 000 Fuäsgvsckwitaigkvit it uu/uit; flflicktwvut
/v 000 Maxiualgvsckwitaigkvit füu Lvvufakutvt it uu/uit; flflicktwvut
/c      Übvupuüfvt allvu flfaav it avu KXF-Katvi oktv o-Coav-outputv; wvtt /c tickt
        atgvgvbvt wiua, aatt aatt aiv KXF-Katvi tuu vitvt flfaa vttkaltvt
/x zzz oibt füu allv auf aivsv Rvgvx passvvt Hvxtv aus, wvlckvu KXF-Objvkt siv
        zugvouatvt sita
/a zzz Suckpfaa for uvfvuvtzivutv KXF-Katvivt
/p zzz Rvg.Ausauuck füu flfaabvzvicktutgvt
/l zzz Mvlautgsspuackv
```

Example calls:

```
flatkKxf?oCoav /k

flatkKxf?oCoav /f150 /v1000 /a..\Bautvilv "?913 flk.KXF" flatkKxf?oCoav

/f150 /v1000 /c /a..\Bautvilv "?050-?051 fl.1v.KXF"
```

4.6.2. Problems

4.6.2.1. "Path definition ... not found." Reason: A

path with this name was not created. 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 from point ... found."

Reason: At the specified point "it does not go any further".

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 the point in the drawing[9] and connect the lines and circle centers exactly.
- Duplicate elements that lie exactly on top of each other; after moving 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 up PathGCodeAdjustZ

4.7.1. Call parameters

The _Clean.gcode files for which the Z-correction is to be performed 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:

```
flatkoCoavAajustZ %001.?7.1fl_Clvat.gcoav %001.?7.?fl.KXF
```

The following options can also specified:

```
/k      Hilfv-Atzvigv
/l zzz Spuackv
```

4.7.2. Problems

—

4.8. Milling

For the milling process

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

the control computer of the milling machine.

The following milling process is not described here.

5. Program documentation

PathDxf2GCode is essentially a 4-pass compiler written in C#, which transforms the DXF input into the G-code output in several phases. All 4 phases are executed for each input

9 To find the problem location in BeckerCAD: select "Point" (small circle), then the coordinates of the current position are shown in the status window when the mouse is moved.

file is run through completely; other files read in for embedded paths are only read once and then stored in a cache.

PathDxf2GCode's own code comprises approx. 900 NLOC; in addition, there is the netDxf library (much larger with 19000 NLOC) (see haplokuon/netDxf: .net dxf Reader-Writer (github.com)).

5.1. Compiler phases

The 4 phases are:

1. DXF file→ DxfDocument; the netDxf library is used for this. The central method of this phase is KxfFilv.LoaaKxfKocuuvtt.
2. DxfDocument→ RawPathModel; the 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;
   - Markings (start and end point as well as turnaround markings)
   - and path parameters.

   The central method of this phase is flatkMoavl.HuatsfouuKxf?flatkMoavl.
3. Combining milling segments into milling chains; the central method for this is flatkMoavl.CuvatvflatkMoavl.
4. PathModel→ G-code file; the central method of this text output is flatkMoavl.EuitoCoav.

5.2. Class models

___STILL MISSING__

5.3. Special algorithms

5.3.1. Params.cs

Params objects have a parent pointer:

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

5.3.2. PathModel.cs

5.3.2.1. 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

This results in a RawPathModel.

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

Find objects for the assignment of parameter texts.

5.3.2.3. CreatePathModel

Here, the actual PathModel is generated from the RawPathModel from the results of the following steps:

A. Linking the segments to form a path, including reversing at turning points

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

C. Creating the parameter objects

D. Sorting the Zprobes

5.3.2.4. WriteEmptyZ

Output of the empty Z-file if ZProbes are available.

5.3.3. PathModelCollection.cs

Management of all paths read from DXF files.

5.3.4. PathSegment.cs

5.3.4.1. MillChain.EmitGCode

Edges are created at an I distance 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: As a rule, edges are therefore run on one level.

5.3.4.2. HelixSegment.EmitGCode

A helix is of semicircles, each of which descends by I/2.

5.3.4.3. SubPathSegment.EmitGCode

A transformation3 is created between the SubPathSegment and the referenced path, which converts the coordinates into the calling model.

5.3.5. 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, 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.

5.3.6. Transformation3.cs

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

5.4. Currently known or suspected bugs

___STILL MISSING__

5.5. Missing features

5.5.1. <- Embeddings

___STILL MISSING__