Asymptote package CAD.asy*

Mark Henning, Germany†

29 September 2006

Contents

1 Introduction **1**

2 Important rules for using this package **1**

3 Usage **2**

4 Example **5**

1 Introduction

The package `CAD.asy` provides basic pen definitions and measurement functions for simple 2D CAD drawings according to DIN 15.

2 Important rules for using this package

If a function is declared like this:

```
void foo(int nParam1, string strParam2)
```

You may call it `foo(0, 'abc');` or `foo(nParam1=0, strParam2='abc');`. The definitions of the functions in this package may change in future version: the order of the parameters may be changed, new parameters may be inserted. Therefore it is *strongly recommended* always calling the functions `foo(nParam1=0, strParam2='abc');`.

*This document describes version 1.0.

†URL: http://www.markhenning.de

3 Usage

To use the capabilities of the package, import it:

```
import CAD;
```

All functions are encapsulated in the structure `sCAD`. As the pen definitions may differ depending on the size of your drawing, you have to initialize the structure before drawing:

```
static sCAD Create(int nLineGroup = 1)
```

The parameter `nLineGroup` depends on the size of your drawing. It specifies the line group to define the pens and thus the width used for the lines. The parameter has to be within the range $[0; 3]$. The larger the value, the thicker the lines. Code example:

```
sCAD cad = sCAD.Create();
```

The created variable `cad` then provides the most important pens. Available pens are:

- The pens of group A:
  - `pA`
  - `pVisibleEdge`
  - `pVisibleContour`
  - `pUsableWindingLength`
  - `pSystemLine`
  - `pDiagramCurve`
  - `pSurfaceStructure`
- The pens of group B:
  - `pB`
  - `pLightEdge`
  - `pMeasureLine`
  - `pMeasureHelpLine`
  - `pMeasureLineBound`
  - `pReferenceLine`
  - `pHatch`

  - `pWindingGround`
  - `pDiagonalCross`
  - `pBendLine`
  - `pProjectionLine`
  - `pGrid`
- The pens of group C:
  - `pC`
  - `pFreehand`
- The pens of group E:
  - `pE`
  - `pSurfaceTreatmentAllowed`
- The pens of group F:
  - `pF`
  - `pInvisibleEdge`
  - `pInvisibleContour`
- The pens of group G:
  - `pG`
  - `pMiddleLine`
  - `pSymmetryLine`
  - `pPartialCircle`
  - `pCircularHole`
  - `pDivisionPlane`
  - `pTransferLine`
- The pens of group J:
  - `pJ`
  - `pCuttingPlane`
  - `pSurfaceTreatmentRequested`
- The pens of group K:
  - `pK`
  - `pContourBeforeDeformation`
  - `pAdjacentPartContour`
  - `pEndShapeRawMaterial`

- pContourEligibleType
- pPartInFrontOfCuttingPlane

All pens of one group are the same. So there is no difference between the pen `pA` and the pen `pVisibleEdge`. You may use the short names or the descriptive ones. The list of groups is not complete. I had no idea how to implement the pens of group D. For me, group H seems to be obsolete, and there is no group I contained in DIN 15. In the case I did something wrong translating the German names into English, the source file also contains the original German names of each pen. Whenever a drawing function does not allow you to select the pen, the right pen is selected automatically.

```
void MeasureLine(picture pic = currentpicture,
                 Label L,
                 pair pFrom,
                 pair pTo,
                 real dblLeft = 0,
                 real dblRight = 0,
                 real dblRelPosition = 0.5,
                 bool bSmallBound = false)
```

This is the basic function to draw labeled straight measurement lines. `pic` denotes the picture the line has to be drawn into, `L` is the label of the line. The pairs `pFrom` and `pTo` are the start and the end point of the distance to measure, respectively. Note, that these two points do *not* refer to the start and end point of the line, as it may be longer than the measured distance.

The line is extended on its left side (= the part of the line 'before' the start point) by the distance `dblLeft`. On its right side (= the part of the line 'behind' the end point) it is extended by the distance `dblRight`. `dblLeft` and `dblRight` must be ≤ 0. If only one of both values is zero, it is set to the value of the other one, because according to DIN 15 it is not allowed to draw a measurement line asymmetrically. The position of the arrows indicating begin and end of the distance depends on `dblLeft` and `dblRight`. If both values are 0, the measurement arrows are drawn within the measurement distance. Otherwise they are drawn outside.

The parameter `dblRelPosition` refers to the relative position of the label between the start and end point. This means: The value 0 positions the label at the start point, 0.5 refers to the center between the start and the end point. Finally, the value 1 will position the label at the end point. If `dblLeft` or `dblRight` are nonzero, you may position the label at the left side of the start point (< 0) or at the right side of the start point (> 1).

Usually, there is enough space to draw the measurement arrows. If you wish to use small circles instead of the arrows, set `bSmallBound` to `true`.

```
real GetMeasurementBoundSize(bool bSmallBound = false)
```

Returns the size of the arrow or the small bound circle used for measurement lines.

```
guide GetMeasurementBound(bool bSmallBound = false)
```

Returns the correctly scaled guide of the arrow or the small bound circle used for measurement lines.

```
void MeasureParallel(picture pic = currentpicture,
                     Label L,
                     pair pFrom,
                     pair pTo,
                     real dblDistance,
                     // Variables from MeasureLine
                     real dblLeft = 0,
                     real dblRight = 0,
                     real dblRelPosition = 0.5,
                     bool bSmallBound = false)
```

Usually, measurement lines are placed outside the drawing with reference lines to the measured distance. `pFrom` and `pTo` denote the points of the drawing marking the begin and the end of the distance to measure, and not the begin and the end of the measurement line as in the case of the function `MeasureLine`. The measurement line is placed `dblDistance` away from the distance to measure. If you would be at `pFrom` and look into the direction `pTo`, it is placed on the left for positive values of `dblDistance`. For negative values, it is positioned on the right. For the meaning of the other parameters see the function `MeasureLine`.

```
guide MakeFreehand(pair pFrom, pair pTo,
                   real dblRelDivisionLength = 12.5,
                   real dblRelDistortion = 2.5,
                   bool bIncludeTo = true)
```

To draw a line between two points, which is not straight, but more like a freehand line, use this function. The pairs `pFrom` and `pTo` denote start and end point, respectively. `dblRelDivisionLength` is the length of the parts, the line is subdivided into. `dblRelDistortion` is the amount of distortion. Both sizes are given relative to the width of a freehand line. If `bIncludeTo` is `true`, the point `pTo` is added to the path. Otherwise it is not. This may be useful for converting a guide containing more than two points to a freehand line.

4 Example

To produce figure 1, use this code:

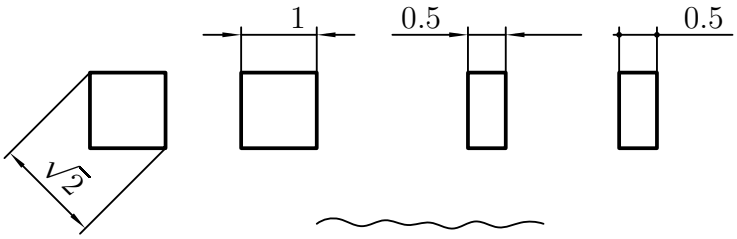


Figure 1: Example showing the measurement capabilities and a small freehand line.

```
import CAD;

sCAD    cad = sCAD.Create();

// Freehand line
draw(g = cad.MakeFreehand(pFrom = (3, -1)*cm, (6, -1)*cm),
     p = cad.pFreehand);

// Standard measurement lines
draw(g = box((0, 0)*cm, (1, 1)*cm), p = cad.pVisibleEdge);
cad.MeasureParallel(L = "$\sqrt{2}$",
                    pFrom = (0, 1)*cm,
                    pTo = (1, 0)*cm,
                    dblDistance = -15mm);

// Label inside, shifted to the right; arrows outside
draw(g = box((2, 0)*cm, (3, 1)*cm), p = cad.pVisibleEdge);
cad.MeasureParallel(L = "1",
                    pFrom = (2, 1)*cm,
                    pTo = (3, 1)*cm,
                    dblDistance = 5mm,
                    dblLeft = 5mm,
                    dblRelPosition = 0.75);

// Label and arrows outside
draw(g = box((5, 0)*cm, (5.5, 1)*cm), p = cad.pVisibleEdge);
cad.MeasureParallel(L = "0.5",
                    pFrom = (5, 1)*cm,
                    pTo = (5.5, 1)*cm,
                    dblDistance = 5mm,
                    dblLeft = 10mm,
                    dblRelPosition = -1);

// Small bounds, asymmetric measurement line
draw(g = box((7, 0)*cm, (7.5, 1)*cm), p = cad.pVisibleEdge);
cad.MeasureParallel(L = "0.5",
                    pFrom = (7, 1)*cm,
```

```
pTo = (7.5, 1)*cm,
dblDistance = 5mm,
dblLeft = 2*cad.GetMeasurementBoundSize(
                bSmallBound = true),
dblRight = 10mm,
dblRelPosition = 2,
bSmallBound = true);
```