

# **TSL-Help**

Version: hsbCAD2015  
Date: 23 January 2015  
Copyright: hsbCad bvba

# TSL

---

## The hsbCad customization scripting language

by [kris.riemslagh@hsbcad.com](mailto:kris.riemslagh@hsbcad.com)

*The "Tool Script Language" (TSL), sometimes called "tools and metal part description language" is an advanced feature of hsbCad, developed by the HSB team.*

*As you might know, the AutoCad drawing is an object oriented database that hosts entities. Each entity has a graphical representation, a set of properties, a set of grip and snap points,... hsbCAD provides a lot of such entities including an hsbBeam, an hsbPanel, but also all sorts of tooling's such as hsbTenon, hsbNailingLine,.... Tooling's have the ability to change/manipulate other entities, eg an hsbTenon will shape the solid representation and cnc-definition of an hsbBeam.*

*TSL brings the power to describe these tooling's to the customer of hsbCAD by means of writing a simple script.*

*Because a TSL entity can query the drawing database and the hsb building information model as well, it allows on top of that for the ultimate customization of automated wall elevation drawings.*

*This functionality is being developed to provide absolute freedom on the definition and application to timbers of customized toolings and metal parts.*



© hsbCAD 2015

**AUTODESK.**

CAD/CAM  
Preferred Industry Partner  
AutoCAD Architecture  
based on Autodesk Techno

# Table of Contents

Foreword	I
<b>Part I Release notes</b>	<b>1</b>
<b>Part II Introduction</b>	<b>56</b>
1 Introduction .....	57
2 Procedure .....	60
3 Auto insert with Beam, Element,...	61
4 Context menu .....	65
5 The old editor window .....	68
6 The new editor window .....	70
7 The newest editor window .....	71
8 Graphical metalpart definition .....	71
9 Double click .....	71
<b>Part III Syntax</b>	<b>73</b>
1 Introduction .....	74
2 Comments .....	74
3 Statements .....	75
4 Expression statements .....	75
5 Compound statements or blocks .....	75
6 Selection statements .....	76
7 Iteration statements .....	77
8 Declaration statements .....	79
9 Array declaration .....	80
10 Jump statements .....	82
11 Maths on double and int .....	83
12 Logical operations .....	84
13 Angle calculations .....	84
14 String .....	87
15 Format String .....	96
16 Using var declaration .....	100
17 Functions .....	101
<b>Part IV Geometric calculations</b>	<b>102</b>
1 Point and Vector manipulation .....	103
2 LineSeg .....	108
3 Line and Plane .....	110

4 PLine .....	114
5 PlaneProfile .....	124
6 Body .....	132
7 LineBeamIntersect .....	146
8 CoordSys, Matrix and transformation .....	148
9 Quader .....	151
10 CrossProduct revealed .....	154
11 DotProduct revealed .....	155
12 PropellerSurface .....	155
<b>Part V Basic operations</b>	<b>159</b>
1 Global functions .....	160
2 Metal parts declarations .....	165
3 Extra Grip/Insert points .....	169
4 Conversion to Hsb-units .....	175
5 Declaration of Shop drawing .....	177
6 Compare instances and posnum generation .....	177
7 DXA output .....	178
8 Changing variables from inside the script .....	180
9 OPM functions .....	181
10 Limited use and automatic tool insertion .....	187
11 Change instance group/layer .....	188
12 Thumbnails .....	190
13 Execution .....	193
14 Copy and erase with beams .....	195
15 Tips using elements .....	196
16 Map .....	199
17 File management .....	214
18 Spawn executable .....	217
19 Lisp ScriptInsert .....	218
20 S-type .....	221
21 Store state in dwg .....	224
22 ShedWizard .....	225
23 Btl generation .....	225
24 HundeggerPba generation .....	230
25 Option system .....	233
26 ModelX .....	236
ModelXComposeSettings .....	243
ModelXInterpretSettings .....	244
27 Performance profiling .....	245

28 Push command on stack .....	246
29 TestReport .....	247
30 ParentChildGrouping .....	248
31 OutlineEdgeInfo .....	252
32 XrefLocker .....	253
33 Grip .....	255
34 Tag .....	260
35 OnDbErase .....	262
36 Fastener graphics .....	263
<b>Part VI Objects</b>	<b>269</b>
1 AcadDatabase .....	270
2 Block .....	276
3 Group .....	284
4 Layout .....	291
5 DictObject .....	293
CncCurveStyle .....	295
CollectionDefinition .....	296
MetalPartCollectionDef.....	303
TrussDefinition.....	308
CurvedStyle .....	313
ExtrProfile .....	321
FastenerAssemblyDef .....	324
FastenerComponentData.....	329
FastenerArticleData.....	330
FastenerSimpleComponent.....	331
FastenerListComponent.....	331
GroupStates .....	332
MapObject .....	333
MasterPanelStyle .....	341
MultiPageStyle .....	343
MvBlockDef .....	345
PainterDefinition .....	352
RenderMaterial .....	354
SipStyle .....	357
SipComponent.....	360
SurfaceQualityStyle .....	361
TslScript .....	362
TruckDefinition .....	364
<b>Part VII Entities</b>	<b>370</b>
1 Entity .....	371
EntityCollection .....	393
2 ToolEnt .....	395
HardWrComp .....	398
FastenerGuideline .....	402
3 TslInst .....	404

4 GenBeam .....	423
5 Beam .....	443
6 Sheet .....	459
7 Sip .....	465
SipEdge .....	476
8 Element .....	478
ElemZone .....	496
ElemText .....	502
Construction directives .....	505
setBlockingRun.....	508
ElemNumber .....	510
9 ElementWall .....	512
10 ElementLog .....	518
LogCourse .....	522
11 ElementWallSF .....	524
12 ElementRoof .....	527
ElemRoofEdge .....	532
13 ElementMulti .....	533
SingleElementRef .....	538
14 BlockRef .....	538
15 ChildPanel .....	542
16 ClipVolume .....	545
17 CollectionEntity .....	550
18 CncCurveEnt .....	552
19 CurvedDescription .....	554
20 EcsMarker .....	557
21 EntPLine .....	558
22 EntCircle .....	560
23 ERoofPlane .....	561
EdgeTileData .....	566
24 ERoofPlaneOpening .....	569
25 FastenerAssemblyEnt .....	571
26 Grid .....	578
27 Opening .....	584
28 OpeningLog .....	592
29 OpeningSF .....	595
30 OpeningRoof .....	599
31 MassElement .....	604
32 MassGroup .....	607
33 MasterPanel .....	610
34 MetalPartCollectionEnt .....	617
35 MultiPage .....	619

MultiPageView .....	622
36 MvBlockRef .....	625
37 NailLine .....	630
38 PlotViewport .....	635
39 PressEnt .....	637
40 SawLine .....	640
41 ShopDrawView .....	644
42 Section2d .....	645
43 Slab .....	648
44 TrussEntity .....	650
TrussEnvelope .....	653
TrussConnection .....	655
TrussSupport .....	657
45 Wall .....	657

## Part VIII Tools on GenBeam 663

1 General Tool .....	664
2 Arc .....	666
3 Ari .....	668
4 BeamCut .....	672
5 Chamfer .....	674
6 ChamferedLap .....	675
7 CncExport .....	678
8 CncMessage .....	679
9 ComplexProfile .....	679
10 ConeDrill .....	681
11 ConvexConcaveProfile .....	682
12 Cut .....	685
13 Dado .....	687
14 DiagonalNotch .....	689
15 DimTool .....	693
16 Double cut .....	693
17 Dove tail connection .....	696
18 Drill .....	698
19 ExtrProfileCut .....	703
20 FreeProfile .....	705
21 FreeText .....	713
22 HalfCut .....	716
23 House .....	719
24 Housed dove tail connection .....	722
25 Kamatsugi .....	725

---

26 Klingschrot .....	727
27 LogNotch .....	730
28 Mark .....	735
29 MarkerLine .....	738
30 MetalTKey .....	740
31 Mortise .....	746
32 PanelStop .....	748
33 PanelWirechase .....	750
34 ParHouse .....	752
35 PropellerSurfaceTool .....	754
36 Rabbet .....	758
37 RevolutionMill .....	760
38 Round types .....	763
39 RoundWoodMill .....	765
40 ScarfJoint .....	766
41 SimpleScarf .....	768
42 Slot .....	769
43 SolidSubtract .....	769
44 SpecialMill .....	771
45 TirolerSchloss .....	773
<b>Part IX Analysed tools</b>	<b>776</b>
1 AnalysedTool .....	777
2 AnalysedArc .....	780
3 AnalysedAri .....	783
4 AnalysedBeamCut .....	786
5 AnalysedChamfer .....	793
6 AnalysedChamferedLap .....	796
7 AnalysedComplexProfile .....	799
8 AnalysedConvexConcaveProfile .....	803
9 AnalysedCut .....	806
10 AnalysedDiagonalNotch .....	809
11 AnalysedDoubleCut .....	811
12 AnalysedDove .....	814
13 AnalysedDovesugi .....	817
14 AnalysedDrill .....	820
15 AnalysedFreeProfile .....	823
16 AnalysedHalfCut .....	826
17 AnalysedHouse .....	829
18 AnalysedKamatsugi .....	833

---

19 AnalysedLogDove .....	836
20 AnalysedLogNotch .....	839
21 AnalysedMarker .....	842
22 AnalysedMarkerLine .....	845
23 AnalysedMortise .....	848
24 AnalysedParHouse .....	852
25 AnalysedPlaning .....	855
26 AnalysedPropellerSurface .....	857
27 AnalysedRabbet .....	860
28 AnalysedScarfJoint .....	863
29 AnalysedShoulderTenon .....	866
30 AnalysedSimpleScarf .....	869
31 AnalysedSlot .....	872
32 AnalysedSpecialMill .....	875
<b>Part X Shopdraw engine</b>	<b>878</b>
1 EntCollector .....	879
2 ActiveRule subMap .....	880
3 Selection filter .....	880
4 OnMapIO .....	883
5 Shopdraw categories .....	886
6 Shopdraw ViewData .....	888
7 Display for shopdraw multipage .....	891
8 Shopdraw recalc triggers .....	892
ShopDraw ChildPageArea .....	893
ShopDraw ChildPageCreate .....	895
ShopDraw ViewDataShow Set .....	897
ShopDraw ViewDataShow Set orientation .....	900
9 ShopDrawTemplateRule .....	908
<b>Part XI DimRequests</b>	<b>911</b>
1 DimRequest .....	912
2 DimRequestPoint .....	916
3 DimRequestLinear .....	917
4 DimRequestText .....	917
5 DimRequestAngular .....	918
6 DimRequestPitch .....	918
7 DimRequestObholz .....	919
8 DimRequestHeightLevel .....	919
9 DimRequestRadial .....	920
10 DimRequestMultiViewLine .....	920

11 DimRequestPLine .....	921
12 DimRequestChain .....	921
13 DimRequestHatch .....	922
<b>Part XII Tools on elements</b>	<b>925</b>
1 ElemNail .....	926
2 ElemNoNail .....	928
3 ElemSaw .....	930
4 ElemMill .....	933
5 ElemDrill .....	936
6 ElemMarker .....	939
7 ElemItem .....	940
8 ElemNailCluster .....	941
9 ElemConstructionBeam .....	943
10 PanelSplit .....	945
11 ElemConstructionMap .....	948
<b>Part XIII Tools on MasterPanels</b>	<b>950</b>
1 ElemSaw_MasterPanel .....	951
2 ElemMill_MasterPanel .....	954
3 ElemMarker_MasterPanel .....	956
4 ElemDrill_MasterPanel .....	958
<b>Part XIV Implementing insert in the script</b>	<b>960</b>
1 Different approaches possible .....	961
2 PrEntity .....	961
3 PrPoint .....	967
4 Global insert functions .....	972
5 Master slave insert .....	981
<b>Part XV Predefined Variables</b>	<b>984</b>
1 General predefined variables .....	985
2 Predefined status variables .....	988
3 Predefined variables T-connection .....	991
4 Predefined variables G-connection .....	995
5 Predefined variables X-connection .....	998
6 Predefined variables E-connection .....	1001
7 Predefined variables O-connection .....	1003
<b>Part XVI Drawing by TSL</b>	<b>1005</b>
1 Display .....	1006

2 Dim and DimLine .....	1021
3 DimAngular .....	1028
4 DimRadial .....	1030
5 DimDiametric .....	1033
6 Display text .....	1035
7 Display on Element .....	1038
8 Display in layout .....	1041
9 BOM in layout .....	1043
10 Viewport .....	1049
View Data .....	1057
11 Hatch .....	1061
<b>Part XVII Nester</b>	<b>1064</b>
1 NesterChild .....	1065
2 NesterMaster .....	1066
3 NesterData .....	1066
4 NesterCaller .....	1068
<b>Part XVIII dotNet</b>	<b>1072</b>
1 dotNet call .....	1073
2 Managed API .....	1076
ModelMapIO .....	1077
CallMapIO .....	1081
Modeless context .....	1085
Extrusion Profiles .....	1087
<b>Part XIX WebKit</b>	<b>1090</b>
1 WebKit .....	1091
<b>Part XX Samples</b>	<b>1093</b>
1 Example of Slot and Mortise .....	1094
2 Examples Cut, Beam cut and Drills .....	1095
3 Example Manipulation of Vectors .....	1102
4 TslArt1 .....	1106
5 TslArt2 .....	1109
6 TslArt3 .....	1113
<b>Part XXI Appendix</b>	<b>1115</b>
1 Formal language specifications .....	1116
2 Hungarian notation .....	1124
3 Special comments .....	1126
4 Tsl editor shortcuts .....	1127

5 Notepad++ .....	1128
6 VS Code .....	1137
<b>Index</b>	<b>1140</b>

I

**hsbCAD TSL**

---

---

# **Part**

I

## 1 Release notes

**17-June-2025** Added [global methods](#) `getVarInt`, `getVarDouble` and `getVarString`.  
added V27.5.1 and V28.2.4

**7-June-2025** Added static [Entity](#) method `filterEntitiesOfType`.  
added V27.5.1 and V28.2.4

**13-Apr-2025** Added optional argument to [PrPoint](#) `go` method to allow script to act on cancel.  
added V27.4.3 and V28.0.21

**20-Feb-2025** Added MapX methods to [HardWrComp](#).  
added V28.0.10

**30-Jan-2025** Added [Display](#) `showInDispComp` method which is important for [fastener graphics](#).  
added V28.0.9

**15-Jan-2025** [global method](#), and [TslInst](#) method `showDialog` now supports the controlling properties that change.  
added V27.3.6 and V28

**30-Dec-2024** Added `_kErase` and `_bOnDbErase`  
added V27.3.6 and V28 a [TslInstance](#) can react on being erased.

**19-Dec-2024** Added `setFreeDirectionsFromRealSolid` method to [BeamCut](#).  
added v27.3.2 and v26.9.22

**6-Dec-2024** Fixed error code for [XRefLocker](#) in case locking is not available.

**21-Oct-2024** Added `coordSysHsb` method on [ElementRoof](#).  
added since in 27

**28-March-2024** A word on [Tags](#)

**5-Feb-2024** Added `setControlsOtherProperties` on [OPM](#) classes [PropInt](#), [PropDouble](#) and [PropString](#).  
added since in 27

**2-Feb-2024** Added [PLine](#) `isCircle` method, and automatic circle streaming.  
added since in 26.5.5.

**1-Feb-2024** Added an optional argument to [TslInst](#).`dbCreate` to postpone initial execution to command ended.  
added since in 26.5.5.

**31-Jan-2024** Added an optional argument to store an Entity in a [Map](#).  
added since in 26.5.5.

**26-Dec-2023** Added `getDrawingPropertiesMap` and `setDrawingPropertiesFromMap` on [AcadDatabase](#).  
added since in 26.4.5

**22-Dec-2023** Added [Display](#) showDuringSnap.  
added since in 25.2.22 and 26.4.5

**30-Nov-2023** Added [GroupStates](#) DictObject.  
added since in 26.3.35

**7-Oct-2023** Fixed the wrong name to the correct [ModelX](#) name.  
fixed since in 25.1.120 and 26.2.10

**22-Aug-2023** Added method correctTextNormalForViewDirection and others to [Dim](#).  
added since in 25.1.114 and 26.1.19

**21-July-2023** Added some methods to [Layout](#).  
added since in 25.1.110

**18-July-2023** Added addLumberToInstallList and removeLumberFromInstallList method to [Sip](#).  
added since in 26.1.6 and 25.1.106

**18-July-2023** Added loadBearing method to [GenBeam](#).  
added since in 26.1.6

**23-May-2023** Added dbCreate method to [FastenerAssemblyDef](#).  
added since in 26

**17-May-2023** Added additional argument to [TslInst](#) and global method called mapWithPropValues reading bOnlyNamesAndValues.  
added since in 25.1.92

**16-May-2023** Added [TrussSupport](#).  
added since in 26

**21-Mar-2023** Added [Grip](#) and [TslInst](#) grips, setGrips methods, as well as \_Grip [predefined](#).  
added since in 25.1.83

**6-Mar-2023** Added coordSys, bodyExtents and extentsWcs method to [Entity](#).  
added since in 24.1.92 and 25.1.76

**6-Mar-2023** Added normalizeVectors method to [Quader](#).  
added since in 24.1.92 and 25.1.76

**4-Mar-2023** Added additional constructors to [DimRequestPitch](#).  
added since in 23.8.99, 24.1.92 and 25.1.76

**2-Mar-2023** Added [Tool](#) setFromMap method for [Drill](#), [Mortise](#), [House](#) and [BeamCut](#).  
added since in 25.1.76.

**2-Mar-2023** Added dCornerRadius method to [AnalysedHouse](#) and [AnalysedMortise](#).  
added since in 24.1.92 and 25.1.76

**6-Feb-2023** Added reconstructArcs for [PLine](#).  
added since in 25.1.72.

**1-Feb-2023 Added resolveSelfIntersect to [Body](#)**  
added since in 24.1.91, 25.1.65.

**30-Jan-2023 Added getLayout method to [Viewport](#)**  
added since in 24.1.91, 25.1.64.

**28-Jan-2023 Added some methods to [Layout](#).**  
added since in 24.1.91, 25.1.64.

**25-Jan-2023 Added simplify to [PLine](#) and [PlaneProfile](#).**  
added since in 24.1.91, 25.1.64.

**20-Jan-2023 Added splitPLine for [PlaneProfile](#).**  
added since in 24.1.90, 25.1.62.

**19-Jan-2023 Added intersectPLineAsDistances for [PLine](#).**  
added since in 24.1.90, 25.1.62.

**19-Jan-2023 Added sorted method for [array](#) of double and array of int.**  
added since in 24.1.90, 25.1.62.

**16-Jan-2023 Added shadowProfile for [CollectionEntity](#).**  
added since in 25.1.58.

**16-Jan-2023 Added setPosnumCompareVertices on [Tool](#).**  
added since in 25.1.58.

**13-Jan-2023 Added setEnumValues to [OPM](#) PropInt, PropDouble and PropString.**  
added since in 24.1.89 and 25.1.58.

**29-Dec-2022 From V26 on Tsl supports [functions](#).**  
added since in 26.0.8

**21-Dec-2022 Added isValid to [Body](#).**  
added since in 24.1.88 and 25.1.57.

**13-Dec-2022 Added HostId to [Entity](#) and [DictObject](#).**  
added since in 24.1.88 and 25.1.57.

**3-Dec-2022 Added [TruckDefinition](#).**  
added since in 26

**3-Dec-2022 Added [Quader](#) side predefines and dX, dY and dZ.**  
added since 24.1.87 and 25.1.55

**23-Nov-2022 Added blocks method to [MultiPage](#).**  
added since 24.1.87 and 25.1.53

**20-Nov-2022 Added [Kamatsugi](#) tool.**  
added since 25.1.52

**20-Nov-2022 Added additional methods to [AnalysedKamatsugi](#).**  
added since 24.1.87 and 25.1.52

---

**3-Oct-2022 Added replace and insert methods to [String](#) type.**  
added since 24.1.70 and 25.1.25

**17-Sep-2022 Added [DimRadial](#) and [DimDiametric](#) type.**  
added since 25.1.25

**15-Sep-2022 Added componentMaterials to [ExtrProfile](#).**  
added since 24.1.68 and 25.1.20

**15-Sep-2022 Added [PlaneProfile](#) bIsValid and self intersecting joinRing.**  
added since 24.1.68 and 25.1.20

**7-Sep-2022 Added [String](#) formatUnit additional rounding argument.**  
added since 23.8.61, 24.1.64 and 25.0.15

**6-Sep-2022 Added [TrussEnvelope](#) type.**  
added since 26.0.2

**16-Aug-2022 Added [DimAngular](#) type.**  
added since 25.1.1535

**10-June-2022 Added [EntCircle](#) type.**  
added since 24.1.49

**9-June-2022 Added [BlockRef](#) getResultBlock and allow [Block](#) to work with anonymous block names.**  
added since 24.1.49

**9-June-2022 Added [PLine](#) isClosed and isSelfIntersecting**  
added since 24.1.49

**25-April-2022 Added dbCreate on [Multipage](#)**  
added since 23.8.55 and 24.1.41

**13-April-2022 Added [global method](#) in3dGraphicsMode**  
added since 23.8.50 and 24.1.32

**13-April-2022 Added [PrPoint](#) jig support for highlight mode**  
added since 23.8.50 and 24.1.32

**11-April-2022 Added [global method](#) getUcs**  
added since 23.8.50 and 24.1.32

**1-April-2022 Added [Block](#) getExtents in specific direction**  
added since 23.8.49 and 24.1.30

**21-Mar-2022 Added [SipStyle](#) properties.**  
added since 23.8.40 and 24.1.29

**21-Mar-2022 Moved quader method to [GenBeam](#).**  
added since 23.8.40 and 24.1.29

**10-Jan-2022 Added `setDefinesFormatting` to `PropString` [OPM](#) method.**  
added since 24.1.11

**10-Jan-2022 Implemented `setOpeningOverwrite` to [ElementLog](#).**  
added since 23.8.19 and 24.1.11

**3-Dec-2021 Added a `showDialog` method to edit a list of [HardWrComp](#).**  
added since 23.8.13 and 24.0

**1-Dec-2021 Enabled editing of [Block](#) content and other Block methods.**  
added since 23.8.11 and 24.0

**23-Nov-2021 Added [TslInst](#) `showElementTools` and `setShowElementTools` method**  
added since 23.8.9 and 24.0

**23-Nov-2021 Added [Map](#) `copyMembersFrom` method**  
added since 23.8.9 and 24.0

**26-Oct-2021 Added [Element](#) `setFloorGroup` method**  
added since 23.8.1 and 24.0

**22-Oct-2021 Added [Body](#) `sliceBody` and `hasIntersection` with plane.**  
added since 23.8.1 and 24.0

**29-Sep-2021 Added [ToolEnt](#) `frozen` en `setFrozen` methods.**  
added since 23.7.15 and 24.0

**20-Sep-2021 The [PrPoint](#) class now has a method to `setSnapMode`.**  
added since 23.7.12 and 24.0

**18-Sep-2021 Added new type [SawLine](#) wrapping the entity which controls an [ElemSaw](#) tool**  
added since 23.7.12 and 24.0

**18-Sep-2021 The [NailLine](#) `dbCreate` default added to modelspace added.**  
added since 23.7.12 and 24.0

**11-Sep-2021 [DimLine](#) added `setUseDisplayTextHeights`**  
added since 23.7.9 and 24.0

**24-Aug-2021 Added additional [Entity](#) `subMapX` methods, to store `subMapX` with a given `CoordSys`.**  
added since 23.7.4 and 24.0

**23-Aug-2021 Added [GenBeam](#) method `copyToolsFrom`**  
added since 23.7.4 and 24.0

**29-June-2021 Added [ConeDrill](#) tool**  
added since 25.0

**24-June-2021 Added static method [Layout](#) `currentLayout` and automatic sorting of layouts according to tab index**  
added since 23.6.35 and 24.0

---

**22-June-2021 Added [global method](#) getViewCenter**  
added since 23.6.34 and 24.0

**1-June-2021 Added [Sheet](#) setXAxisDirectionInXYPlane**  
added since 22.1.137, 23.6.25 and 24.0

**10-May-2021 Exposed [MultiPageView](#)**  
added since 23.6.10 and 24.0

**28-Apr-2021 Exposed [MultiPage](#) entity**  
added since 23.6.3 and 24.0

**20-Apr-2021 Improved the [TestReport](#) methods to give immediate feedback.**  
added since 23.5.22 and 24.0

**16-Apr-2021 Added subMapX methods to all [tools](#)**  
added since 23.5.22 and 24.0

**24-Mar-2021 Added static method to [Block](#) called writeEntitySetIntoDwg**  
added since 23.5.9

**24-Mar-2021 Added [AcadDatabase](#) method called xrefDatabases**  
added since 23.5.9

**23-Mar-2021 [Element](#) added method getToolsOfTypeSaw, getToolsOfTypeNail,  
getToolsOfTypeDrill and getToolsOfTypeMill.**  
added since 23.5.9

**23-Mar-2021 Added member methods to [ElemSaw](#), [ElemNail](#), [ElemDrill](#) and [ElemMill](#).**  
added since 23.5.9

**19-Mar-2021 Added extra argument to writeToDwgFile method of [Display](#).**  
added since 22.1.132 and 23.5.9

**19-Mar-2021 Added [array](#) methods reverse and filterValid.**  
added since 22.1.132 and 23.5.9

**17-Mar-2021 Added extra argument to [DimLine](#)::collectDimPoints method.**  
added since 22.1.132 and 23.5.9

**11-Mar-2021 Added [FreeProfile](#) setMillSidePolyNormal method.**  
added since 22.1.132 and 23.5.5

**23-Feb-2021 Added [Group](#) turnGroupVisibilityOn and Off, groupVisibility, and ucs.**  
added since 23.4.23

**10-Feb-2021 Added [FreeProfile](#) setCutDefiningAsOne.**  
added since 23.4.18

**28-Jan-2021 Added database to return [AcadDatabase](#) method to [Entity](#) and [DictObject](#).**  
added since 22.1.130 and 23.4.12

**22-Jan-2021 Added [Viewport](#) turnGroupVisibilityOn and Off and groupVisibilityIsOn and Off.**

added since 23.4.4

**17-Jan-2021** Added `drawImage` method to [Display](#), and `encodeImageFromFile` and `encodedImageHeightOverWidth` to [String](#).

added since 23.4.4

**15-Jan-2021** Added `transparency` method to [Display](#).

added since 23.4.4

**15-Jan-2021** Added [global method](#) `getViewHeight`.

added since 23.4.4

**12-Jan-2021** Added additional arguments to [String](#) `qrEncode` method, also added `inQrEmbeddedImageSizeFactor`.

added since 23.4.1

**6-Jan-2021** Added `_kHidden` and `_kReadOnly` as possible arguments to [properties](#) `setReadOnly` method.

added since 23.3.10

**6-Jan-2021** Added [Display](#)::`drawHsbLogo` method.

added since 23.3.10

**12-Dec-2020** Added [ElementWall](#) methods `maximumHeight` and `roofCodeExpression`

added since 22.1.119 and 23.3.3

**30-Nov-2020** Extended [ModelXComposeSettings](#) with `addUniqueIdsAsHandle`, and [ModelX](#).`writeToDxxFile` with `bAddUniqueIdsAsHandle` argument.

added since 24.0.11

**30-Nov-2020** Declared 2 methods in [ModelXInterpretSettings](#) as obsolete.

added since 24.0.11

**27-Nov-2020** Added [global method](#) to get and set the `hsbShareProjectId`

added since 24.0.11

**23-Nov-2020** Added [Entity](#)::`uniqueId` method

added since 24.0.10

**6-Nov-2020** Added [PlotViewport](#) entity type

added since 23.1.3.

**5-Nov-2020** Added [Layout](#) type

added since 23.1.3.

**28-Oct-2020** Added additional `rgbR`, `rgbG` and `rgbB` [global method](#)

added since 23.1.3.

**24-Oct-2020** Added [global methods](#) `getBackgroundTrueColor` and `getViewDirection`.

added since 23.1.3.

**21-Oct-2020** [PropInt](#), [PropDouble](#) and [PropString](#) improved map access.

added since 23.0.104

---

**20-Oct-2020** Added [auto as keyword](#) to infer the type of a variable.  
added since 23.0.104, and 22.1.108

**19-Oct-2020** Added [String](#) mm2GermanDimensionFormat.  
added since 23.0.75

**18-Oct-2020** Added [OnGripPointDrag](#) event for the \_Pt0 and \_PtG points being modified to generate temporary graphics with \_bOnGripPointDrag TRUE.  
added since 23.0.103 and 22.1.108

**12-Oct-2020** Added [PrPoint](#) goJig method to support jiggling during select point.  
added since 23.0.75 and 22.1.108

**12-Oct-2020** Added [TsInst](#)::debug and setDebug.  
added since 23.0.75 and 22.1.108

**9-Oct-2020** Added [Entity](#) filterAcceptedEntities.  
added since 23.0.75

**9-Oct-2020** Added [PainterDefinition](#) filterAcceptedEntities.  
added since 23.0.75

**8-Oct-2020** Added [MultiPageStyle](#) methods like getListOfStereotypeOverrides.  
added since 23.0.75, 22.1.107

**6-Oct-2020** Added trueColor to [Display](#), added [global method](#) rgb, and added [Entity](#) methods trueColor, hasTrueColor and setTrueColor.  
added since 23.0.75

**16-Sept-2020** Simplified the addition of [context](#) root menu item.  
added since 23.0.75, 22.1.103, 21.4.98

**12-Sept-2020** Added [TsInst](#)::removeCatalogEntry.  
added since 23.0.74

**23-July-2020** Added [Entity](#)::formatObjectVariables for a given type.  
added since 22.1.87

**9-July-2020** [Dim](#) has now the possibility to get the text areas for a [Display](#).  
added since 22.1.84 and 23.0.46

**16-June-2020** [MetalPart](#) added body method.  
added since 21.4.95, 22.1.72 and 23.0.45

**16-June-2020** [ClipVolume](#) allows to explode Xref BlockReferences while collecting entitiesInClipVolume  
added since 22.1.72 and 23.0.45

**10-June-2020** Added opening methods to [MasterPanel](#)  
added since 22.1.72 and 23.0.45

**9-June-2020** Added MapX support to [SingleElementRef](#)

added since 22.1.71 and 23.0.45

**29-May-2020** Added [Beam](#)::dbCreate methods from a database resident acis solid.  
added since 22.1.69 and 23.0.43

**29-May-2020** Added [PainterDefintion](#).  
added since 23.0.43

**19-May-2020** Added [PlaneProfile](#) intersectPoints, ptMid, dX and dY methods  
added since 22.1.70 and 23.0.43

**10-May-2020** Added [Entity](#)::acceptObject and [Entity](#)::errorMessage.  
added since 23.0.43

**28-April-2020** Added [TslInst](#) mapWithPropValuesFromCatalog  
added since 22.1.63 and 23.0.43

**27-April-2020** Allow to clone with [beam split](#)  
added since 22.1.62 and 23.0.42

**23-April-2020** [Sip](#) added addRing method.  
added since 22.1.61 and 23.0.40

**15-April-2020** [PLine](#) added trim method.  
added since 22.1.54 and 23.0.36

**15-April-2020** [ChildPanel](#) added getMasterPanel method.  
added since 22.1.54 and 23.0.36

**27-March-2020** Improved [AnalysedBeamCut](#) quader method.  
added since 22.1.43 and 23.0.23

**25-March-2020** Added [CollectionDefinition](#) methods to support mirroring.  
added since 22.1.43 and 23.0.23

**16-March-2020** [TslInst](#) added flipAlignZ.  
added since hsbDesign22 22.1.39 and hsbDesign23 23.0.23.

**18-feb-2020** Posnum assignment criteria can be retrieved and set (see [global functions](#)).  
added since hsbDesign22 22.1.31.

**8-dec-2019** Added [FreeText](#) tool.  
added since hsbDesign22 22.1.11.

**19-oct-2019** Added [SipStyle](#).dbCreate.  
added since hsbDesign21 build 21.4.81 and hsbDesign22 22.1.6.

**19-oct-2019** Added [MasterPanelStyle](#).setThickness and dbCreate.  
added since hsbDesign21 build 21.4.81 and hsbDesign22 22.1.6.

**19-oct-2019** Added [Beam](#).stretchDynamicTo with a gap value.  
added since hsbDesign21 build 21.4.81 and hsbDesign22 22.1.6.

**17-oct-2019 Added [Entity.realBodyTry](#) and [Body.isNull](#) methods.**  
added since hsbDesign21 build 21.4.81 and hsbDesign22 22.1.6.

**5-oct-2019 Added [Map getBodyFaceLoops](#).**  
added since hsbDesign22 build 22.1.2

**9-Sep-2019 Added [ERoofPlane frontCutPerpToRoof](#) method.**  
added since hsbDesign21 build 21.4.75 and hsbDesign22 22.0.116.

**2-Aug-2019 Added [PLine offset](#) fillet type and [PLine append](#) method.**  
added since hsbDesign22 build 22.0.106 and build 21.4.66.

**29-July-2019 Added [XrefLocker](#) type.**  
added since hsbDesign22 build 22.0.106.

**29-July-2019 Added [allowXrefContentToBeModified](#) method to [ModelXInterpertSettings](#).**  
added since hsbDesign22 build 22.0.106.

**23-July-2019 Added [PLine offset](#).**  
added since hsbDesign22 build 22.0.105 and build 21.4.65.

**22-June-2019 Allowed [FreeProfile](#) to mill path only. Also exposed in [AnalysedFreeProfile](#).**  
added since hsbDesign22 build 22.0.100.

**13-June-2019 Added global [file management](#) method called [expandEnvironmentVariable](#).**  
added since hsbCAD2017 build 21.4.51 and build hsbDesign22 build 22.0.98.

**23-May-2019 Added on [array](#) of String [findNoCase](#) method.**  
added since hsbCAD2017 build 21.4.41 and build hsbDesign22 build 22.0.75.

**7-May-2019 Added [inPaperSpace](#) and [inLayoutTab](#) as well as [setCurrent](#) method to [Viewport](#).**  
added since hsbCAD2017 build 21.4.41 and build hsbDesign22 build 22.0.75.

**7-May-2019 Added [setMillAround](#) method to [House](#) and [Mortise](#).**  
added since hsbCAD2017 build 21.4.41 and build hsbDesign22 build 22.0.75.

**16-April-2019 Support for Tsl main [context menu](#) support.**  
added since hsbCAD2017 build 21.4.37 and build hsbDesign22 build 22.0.74.

**7-April-2019 Added [ClipVolume](#) entity.**  
added since hsbCAD2017 build 21.4.36 and build v22

**7-April-2019 Added [Section2d](#) entity.**  
added since hsbCAD2017 build 21.4.36 and build v22

**7-April-2019 Added [Body::rayIntersection](#).**  
added since hsbCAD2017 build 21.4.36 and build v22

**26-March-2019 Added multi-line support for [Display of text](#), also build in support in the [Display](#) methods [textLengthForStyle](#) and [textHeightForStyle](#).**  
added since hsbCAD2017 build 21.4.31 and build v22

**12-March-2019 Added [Entity](#)::transparency and setTransparency.**  
added since hsbCAD2017 build 21.4.15 and build v22

**26-Feb-2019 Added [PLine](#)::getTangentAtPoint.**  
added since hsbCAD2017 build 21.4.14 build v22

**26-Feb-2019 Added last and first methods to any [array](#).**  
added since hsbCAD2017 build 21.4.14 build v22

**26-Feb-2019 Added [Body](#) constructor for extrusion along path.**  
added since hsbCAD2017 build 21.4.14 build v22

**5-Feb-2019 Added [ERoofplane](#)::vertices and setVertices.**  
added since hsbCAD2017 build 21.4.12 build v22

**16-Oct-2018 Added [Map](#)::dbCreateBodyEntity, and added bAllowInvalid argument on the getBody method.**  
added since hsbCAD2017 build 21.3.117 build v22

**20-Sep-2018 Added addition argument to [Beam](#)::dbCreate from Body to convert into bounding body only taking into account cuts and double cuts.**  
added since hsbCAD2017 build 21.3.107 build v22

**18-Sep-2018 Added addition argument to [String](#)::tokenize**  
added since hsbCAD2017 build 21.3.106 build v22

**11-Sep-2018 Added [PropInt](#)::setMap, [PropString](#)::setMap, [PropDouble](#)::setMap, [PropInt](#)::getMap, [PropString](#)::getMap, [PropDouble](#)::getMap**  
added since hsbCAD2017 build 21.3.103 build v22

**4-Sep-2018 Added [Display](#)::drawQR method as well as [String](#)::qrEncode and [String](#)::qrDecode.**

These methods allow to generate QR codes and display them anywhere in the drawing.  
added since hsbCAD2017 build 21.3.101 build v22

**23-Aug-2018 Added some [PlaneProfile](#) methods: removeAllOpenings, allRings and createRectangle**  
added since hsbCAD2017 build 21.3.98 and v22.0.55

**16-Aug-2018 Added [String](#) find case sensitivity**  
added since hsbCAD2017 build 21.3.96 and v22.0.55

**29-May-2018 Added [ERoofPlane](#)::[EdgeTileData](#)**  
added 22.0.39

**27-May-2018 Added [ERoofPlane](#)::findParentRoofplane**  
added 22.0.39

**2-May-2018 Added additional argument to [Beam](#)::filterBeamsHalfLineIntersectSort**  
added since hsbCAD2017 build 21.3.52 and v22.0.39

**25-April-2018 Added [Viewport](#) switchToModelSpace and switchToPaperSpace and setCurrent**

added since hsbCAD2017 build 21.3.50 and v22.0.35

**8-Mar-2018 Added** [BlockRef](#)::dbCreate, coordSysScaled, getAttributeList, getAttributeMap, setAttributesFromMap, eraseAttributes, createMissingAttributes and setDefinition method  
added since hsbCAD2017 build 21.3.32 and v22.0.34

**23-Feb-2018 Added** [TslInst](#)::dbCreate method with catalog entry and event  
added since hsbCAD2017 build 21.3.23 and v22.0.32

**27-Jan-2018 Added** [ERoofPlane](#)::findContainingRoofplanes  
added since 22.0.32

**18-Jan-2018 Added** [Entity](#)::formatObject and [Entity](#)::formatObjectVariables  
added since hsbCAD2017 build 21.3.10 and 22.0.32

**18-Dec-2017 Added** [Mortise](#)::setExplicitRadius  
added since hsbCAD2017 build 21.3.1 and 22.0.32

**6-Dec-2017 Added** [Entity](#)::renamePropertiesInSet  
added since hsbCAD2017 build 21.2.10

**2-Dec-2017 Added** [TslScript](#) DictObject and [TslInst](#)::setScriptName  
added since hsbCAD2017 build 21.2.9

**29-Nov-2017 Added** [ViewData](#)::scaleGroup  
added since hsbCAD2017 build 21.2.8

**13-Oct-2017 Added** [PLine](#)::flipNormal.  
added since hsbCAD2017 build 21.1.46

**9-Oct-2017 Added support for** [ElemSaw](#), [ElemMill](#), [ElemMarker](#) and [ElemDrill](#) on [MasterPanels](#).  
added since hsbCAD2017 build 21.1.43 and 22.0.30.

**26-Sept-2017 Added** [Entity](#)::blockRef and [Entity](#)::blockRefs  
added since hsbCAD2017 build 21.1.39

**14-Sept-2017 Added** [GenBeam](#)::isotropic property.  
added since hsbCAD2017 build 21.1.34

**9-Sept-2017 Added** [GenBeam](#)::cuttingBodyOfToolEnt method.  
added since hsbCAD2017 build 21.1.33

**6-Sept-2017 Added** [MasterPanel](#)::addTool method.  
added since hsbCAD2017 build 21.1.30

**2-Sept-2017 Added** [Beam](#)::vecCenterOffset and [Beam](#)::vecCenterLogOffset, and corrected [Beam](#)::quader to respect extrusion profile center offsets.  
added since hsbCAD2017 build 21.1.29

**27-Aug-2017 Added** [CurvedStyle](#) setPtInsert.  
added since hsbCAD2017 build 21.1.29

**25-Aug-2017 Added language specific translation to [String](#).**  
added since hsbCAD2017 build 21.1.29

**23-Aug-2017 Added releasePosnum and assignPosnum to [TslInst](#).**  
added since hsbCAD2017 build 21.1.27

**10-Aug-2017 Added setFromHandle on [Group](#).**  
added since hsbCAD2017 build 21.1.25

**27-June-2017 GetPLine is now also applicable on Circles in the drawing.**  
added since hsbCAD2017 build 21.1.5

**15-June-2017 Added [string](#) tokenize method.**  
added since hsbCAD2017 build 21.1.5

**2-June-2017 Improved [ElemNoNail](#) tool and added [Element](#)::getToolsOfTypeNoNail**  
added since hsbCAD2017 build 21.1.1

**17-May-2017 Added [ImportModelX](#) support for ProjectInfo.**  
added since hsbCAD2017 build 21.0.125

**6-Apr-2017 Added sorted method on [array](#) of String.**  
added since hsbCAD2017 build 21.0.115

**6-Apr-2017 Changed the resolving of enum property values stored in a catalog.**  
First the property value is used. If not found, the enum index is used. If all that fails, then the value itself is used.  
added since hsbCAD2017 build 21.0.115

**5-Apr-2017 Added [AnalysedPropellerSurface](#).**  
added since hsbCAD2017 build 21.0.115

**17-Mar-2017 Added transparency to [Display](#) draw PlaneProfile filled.**  
added since hsbCAD2017 build 21.0.105

**16-Mar-2017 Added \_kShiftKeyPressed and \_kCtrlKeyPressed, see [context menu](#).**  
added since hsbCAD2017 build 21.0.103

**15-Mar-2017 Added [GenBeam](#) method getToolsStaticOfTypeBeamCut**  
added since hsbCAD2017 build 21.0.103

**6-Mar-2017 Added [Tsl editor shortcut](#) list.**

**4-Feb-2017 Added nameAt to [Map](#)**  
added since hsbCAD2017 build 21.0.85

**12-Jan-2017 Added toolIndex to [Daddo](#) tool**  
added since hsbCAD2017 build 21.0.71

**22-Dec-2016 Added exportAsMark to [MarkerLine](#) tool**  
added since hsbCAD2017 build 21.0.71

**19-Dec-2016 Added [MetalTKey](#) tool**

added since hsbCAD2017 build 21.0.68

**12-Dec-2016 Added [dbCopyEntitiesFrom](#) to [Element](#) class**

added in v21.0.67. The method supports cloning of multiple entities at once.

**29-Nov-2016 Added [AcadDatabase](#) class**

added v21.0.66

**29-Nov-2016 Added methods to [ElementMulti](#)**

added v21.0.66

**25-Nov-2016 Added [file and folder management](#) tools**

added hsbCAD v20.3.47 and v21.0.64

**24-Nov-2016 Added [global](#) method [create.NewGuid](#).**

added hsbCAD v20.3.47 and v21.0.64

**24-Nov-2016 Allowed [Group](#)::[collectEntities](#) to collect none Aec entities.**

added hsbCAD v20.3.47 and v21.0.64

**30-Sep-2016 Added [Sip](#)::[stretchEdgeTo](#) with [nOpeningIndex](#) and [nEdgeIndex](#).**

added v21.0.56 and v20.3.30

**31-Aug-2016 Added [Map](#) methods for storing and retrieving [PlaneProfile](#).**

added to v21.0.52

**10-Aug-2016 Added [Sheet](#) dbCreate method from Body.**

added to v21.0.51

**8-Aug-2016 Added [Point3d](#) projectPoint with direction.**

added to v21.0.51 and v20.3.19

**28-Jul-2016 Added [MasterPanel](#) ptRef and setPtRef**

added to v21.0.49 and v20.3.18

**27-Jul-2016 Added [Map](#) methods for storing and retrieving [Body](#).**

added to v21.0.49

**22-Jun-2016 Added RefDocs api to [Entity](#).**

added to v21.0.48

**20-Jun-2016 Find the [Tsl editor shortcuts](#) in the appendix.**

**25-Mar-2016 [SipStyle](#), [MapObject](#), [MasterPanelStyle](#), [MvBlockDef](#), [FastenerAssemblyDef](#), [CollectionDefinition](#), [MetaIPartCollectionDef](#), [TrussDefinition](#), [MultiPageStyle](#) and [SurfaceQualityStyle](#) now have importFromDwg and getAllEntriesFromDwg methods.**

added to v20.1.69 and v21.0.43

**15-Mar-2016 [Display](#) of zero length LineSeg as Point**

added to v20.1.65 and v21.0.39

**9-Mar-2016 Added methods [referenceFace](#) and [setReferenceFace](#) to [Beam](#)**

added to v20.1.61 and v21.0.39

**5-Mar-2016** Added catalog support for [HardWrComp](#).  
added to v20.1.59 and v21.0.39

**16-Feb-2016** Added addOldToolInfo to [ModelXComposeSettings](#).  
added to v20.1.44 and v21.0.39

**1-Nov-2015** [FastenerAssemblyEnt](#) added guidelineToolEnt.  
added to v20.1.14 and v21.0.21

**1-Nov-2015** [ToolEnt](#) and [GenBeam](#) added getAttachedFasteners.  
added to v20.1.14 and v21.0.21

**7-Oct-2015** [MasterPanel](#) getMainGrainDirectionFromChildPanels.  
added to v20.1.6 and v21.0.9

**7-Oct-2015** [ChildPanel](#) woodGrainDirection.  
added to v20.1.6 and v21.0.9

**30-Sept-2015** [NesterData](#) added RectangularNester.  
added to v21.0.6

**30-Sept-2015** [ModelXComposeSettings](#) added addAllGroups method.  
added to v20.1.3 and v21.0.6

**6-Sept-2015** Exposed members of [Cut](#).  
added to v20.0.114 and v21.0.5

**6-Sept-2015** Added methods to [GenBeam](#) to get static cut tools: getToolsStaticOfTypeCut.  
added to v20.0.114 and v21.0.5

**31-July-2015** Added to totalWidth and zFaceOffset [Wall](#).  
added to v19.1.114, v20.0.96 and v21.0.3

**25-June-2015** Added innerCylinderDiameter to [Drill](#) tool to support shear plate connector.  
added to hsbCAD20.0.88.

**22-May-2015** Added [OutlineEdgeInfo](#) example for Bvx export of sheeting.  
added to hsbCAD20.0.80.

**15-May-2015** Added [ParentChildGrouping](#) for nesting or stacking.  
added to hsbCAD20.0.80.

**15-May-2015** Added [Sheet](#)::setPIEnvelope method.  
added to hsbCAD20.0.80 and hsbCAD19.1.104

**15-May-2015** Added [GenBeam](#)::coordSys method.  
added to hsbCAD20.0.80 and hsbCAD19.1.104

**6-May-2015** Added member methods to [Drill](#) tool.  
added to hsbCAD20.0.76 and hsbCAD19.1.103

---

**6-May-2015** Added methods to [GenBeam](#) to get static drill tools: `getToolsStaticOfTypeDrill`, and to remove static Drill tools `removeToolStatic`.

See example in [Drill](#) tool. Added to hsbCAD20.0.76 and hsbCAD19.1.103

**9-April-2015** `dbCreate` method added to [OpeningSF](#) and [OpeningLog](#).

added to hsbCAD20.0.68 and hsbCAD19.1.101

**9-April-2015** `removeHsbData` method added to [Entity](#)

added to hsbCAD20.0.68 and hsbCAD19.1.101

**27-Mar-2015** `dbCreate` method added to [MetalPartCollectionEnt](#) and [CollectionEntity](#).  
added to hsbCAD20.0.68 and hsbCAD19.1.99

**27-Mar-2015** `dbCreate`, `setContent` and `clearContent` methods added to [CollectionDefinition](#), [TrussDefinition](#), [MetalPartCollectionDef](#).  
added to hsbCAD20.0.68 and hsbCAD19.1.99

**26-Mar-2015** Global method [setKeepReferenceToGenBeamDuringCopy](#) added.  
added to hsbCAD20.0.68 and hsbCAD19.1.99

**19-Mar-2015** [ToolEnt](#)::`cuttingBody` added and fixed [ToolEnt](#)::`removeGenBeamConnection`.  
added to hsbCAD20.0.66 and hsbCAD19.1.95

**10-Feb-2015** Moved methods `type`, `setType`, `openingDescr` and `setOpeningDescr` from [OpeningSF](#) and [OpeningLog](#) to [Opening](#).

**28-Jan-2015** [Map](#)::`toJsonContent` and `fromJsonContent` methods added.  
added to hsbCAD20.0.55

**9-Jan-2015** [TslInst](#)::`allowGripAtPt0` and `setAllowGripAtPt0` methods added.  
added to hsbCAD20.0.50 and hsbCAD19.1.81

**7-Jan-2015** [Entity](#)::`createPropSetDefinition` method added.  
added to hsbCAD20.0.50 and hsbCAD19.1.81

**22-Dec-2014** [ERoofPlane](#)::`runCorrector` method added.  
added to hsbCAD20.0.50 and hsbCAD19.1.80

**27-Oct-2014** [ModelX](#)::`CncExporter` added `exporterShortcuts`.  
added to hsbCAD20.0.41

**24-Oct-2014** [Sip](#)::`basicFoamBody` and [Sip](#)::`foamComponentIndex` added.  
added to hsbCAD20.0.40

**3-July-2014** [GenBeam](#)::`assignPosnum` method enhanced with second argument `bLookForEqual`.  
added to hsbCAD19.1.47 and hsbCAD20.0.22

**27-June-2014** Lisp command [hsb\\_recalcTslWithKey](#) extended with an optional argument for prompting the selection.  
added to hsbCAD19.1.47 and hsbCAD20.0.16

**15-May-2014** Added category to the [OPM](#) `PropString`, `PropInt` and `PropDouble`.

added to hsbCAD19.1.31 and hsbCAD20.0.16

**22-Mar-2014 Added [Entity::assignToGroup](#) method with second argument**

added to hsbCAD18.2.49, hsbCAD19.1.20 and hsbCAD20.0.8

**15-Dec-2013 Updated [PropellerSurfaceTool](#)**

added to hsbCAD2015

**14-Jan-2014 Added [PLine::createSmoothArcsApproximation](#)**

added to hsbCAD2015

**8-Jan-2014 Added [PropellerSurface](#)**

added to hsbCAD2015

**7-Jan-2014 Added [PropellerSurfaceTool::cuttingBody](#)**

added to hsbCAD2015 (20.0.5)

**7-Jan-2014 Added [Element::aca2HsbCatalogList](#) and [Element::aca2HsbRunRuleSet](#)**

added to hsbCAD2015 (20.0.5)

**30-Dec-2013 Added [ElementWall::cornerCleanup](#)**

added to hsbCAD2015 (20.0.5)

**24-Dec-2013 Added [TrussEntity::dbCreate](#)**

added to hsbCAD2014 (19.1.1) and hsbCAD2015

**20-Dec-2013 Added [PropellerSurfaceTool](#)**

added to hsbCAD2015

**1-Dec-2013 Added [Entity::createRealBodyStlFile](#)**

added to hsbCAD2014 (19.0.62) and hsbCAD2015

**17-Nov-2013 Added [Body::dbCreateAs3dSolid](#) and [Body::createStlFile](#)**

added to hsbCAD2014 (19.0.61) and hsbCAD2015

**14-Nov-2013 Added execute key argument to [TsInst::recalcNow](#)**

hsbCAD2013, hsbCAD2014 and hsbCAD2015

**14-Nov-2013 Added [ElementRoof::setVertexPoints](#)**

hsbCAD2014 and hsbCAD2015

**13-Nov-2013 Exposed [Element::triggerGenerateSheeting](#)**

hsbCAD2014 and hsbCAD2015

**27-Sep-2013 Adjusted [AnalysedRabbet](#) definition**

hsbCAD2013, hsbCAD2014 and hsbCAD2015

**20-Aug-2013 Added WebKit support.**

hsbCAD2015 build 20.0.1

**19-July-2013 Added optional argument to [Display::lineType](#) to set line type scale.**

hsbCAD2014 build 19.0.37

---

**3-July-2013** Added sample for export [ModelX](#) of entities of elements.

**21-May-2013** Added [CurvedStyle](#)::topCurve and added new CurvedStyle::dbCreate.  
hsbCAD2014 build 18.2.17

**17-April-2013** Added additional methods of [Element](#)::setProfNetto and profNetto.  
hsbCAD2014 build 19.0.24

**15-April-2013** Added [PlaneProfile](#)::splitSegments and [Element](#)::noNailProfile  
hsbCAD2013 build 18.2.15, hsbCAD2014 build 19.0.23

**14-March-2013** Added [AnalysedHalfCut](#).  
hsbCAD2013 build 18.2.13

**8-March-2013** Added recalcNow method to [TslInst](#).  
hsbCAD2013 build 18.2.11 and hsbCAD2014

**18-Jan-2013** Added [ConstructionDirective](#) setOpeningOverwrite with pline for sip generation.  
hsbCAD2013 build 18.2.7

**14-Jan-2013** Added StyleDescription to [DictObject](#).  
hsbCAD2013 build 18.2.6

**7-Dec-2012** Added missing member [Beam](#)::texture.  
hsbCAD2013 build 18.2.5

**9-Nov-2012** Fix [CurvedStyle](#) for ptRef.  
hsbCAD2013 build 18.2.1

**31-Oct-2012** Added [CurvedDescription](#) and [PressEnt](#) entities and enhanced the [CurvedStyle](#) exposure.  
hsbCAD2013 build 18.2.0

**18-Oct-2012** [Group](#)::collectEntities now accepts Entity as entity type to collect.  
hsbCAD2013 build 18.1.63

**6-Sept-2012** Allowed [Shopdraw](#) filter tsl to implement the entity selection phase.  
hsbCAD2013 build 18.1.51

**6-Sept-2012** Added [ExtrProfile](#) componentProfiles  
hsbCAD2013 build 18.1.51 and hsbCAD2012 build 17.2.38

**2-Aug-2012** Added [Sip](#) plShadow  
hsbCAD2013 build 18.1.45

**2-Aug-2012** [assignToGroups](#) now has an argument to set the zoneCharacter.  
hsbCAD2013 build 18.1.45

**27-June-2012** Added [GenBeam](#)::analysedToolsFromEnvelope  
hsbCAD2013 build 18.1.38

**12-June-2012** Added [Block](#)::dbCreateFromDxf  
hsbCAD2013 build 18.1.37 and hsbCAD2012 build 17.2.32

**1-June-2012** Added another `setOpeningOverwrite` [construction directive](#)  
hsbCAD2013 build 18.1.35

**25-May-2012** Added [Entity](#) `setRenderMaterialMapping`, see [RenderMaterial](#)  
hsbCAD2013 build 18.1.34

**25-May-2012** Added [Entity](#) `renderMaterial` and `setRenderMaterial`, as well as [RenderMaterial](#)  
hsbCAD2013 build 18.1.32

**9-May-2012** [TslInst](#)::`recalc` method added  
hsbCAD2013 build 18.1.29 and hsbCAD2012 build 17.2.25

**4-May-2012** [Sip](#) added method `realBodyOfComponentAt`  
hsbCAD2013 build 18.1.28 and hsbCAD2012 build 17.2.24

**4-May-2012** [Block](#) `getExtents` method added  
hsbCAD2013 build 18.1.28 and hsbCAD2012 build 17.2.24

**4-May-2012** Also resolving Tsl's found in <hsbInstall>\Custom and subfolders now.  
hsbCAD2013 build 18.1.28 and hsbCAD2012 build 17.2.24

**13-April-2012** [TrussEntity](#) can now get its [TrussDefinition](#) directly, which works for xrefs.  
hsbCAD2013 build 18.1.24

**5-April-2012** [ElementWallSF](#) `detCodeMap` added  
hsbCAD2013 build 18.1.22 and hsbCAD2012 build 17.2.18

**20-March-2012** [HundeggerPba](#) modifier added  
hsbCAD2013 build 18.1.21

**8-March-2012** Added [Entity](#) `xrefName` and `allBlockRefPaths`  
hsbCAD2013 build 18.1.15

**30-Jan-2012** [MassGroup](#) `assignPosnums` and `releasePosnums`  
hsbCAD2013 build 18.1.9 and hsbCAD2012 build 17.2.6

**25-Jan-2012** Added [Sip](#) `pIShadowCnc` added  
hsbCAD2013 build 18.1.9

**20-Jan-2012** Added `setParentEntity` and `releaseParentEntity` to [Opening](#)  
hsbCAD2013 build 18.1.8

**20-Jan-2012** Added `storingRoughDimensions` to [Opening](#)  
hsbCAD2013 build 18.1.8

**17-Jan-2012** Added [construction directive](#) for [blocking runs](#).  
hsbCAD2013 build 18.1.8

**12-Jan-2012** Added [AnalysedMarkerLine](#)  
hsbCAD2013 build 18.1.8 and hsbCAD2012 build 17.2.4.

**11-Nov-2011** Added `dbCreate` to [Opening](#)

---

hsbCAD2013 build 18.0.31

**12-Oct-2011 Added posnum and assignPosnums to [MassGroup](#)**  
hsbCAD2013 build 18.0.24 and hsbCAD2012 build 17.1.29.

**27-Sept-2011 [Body](#):: addTool, addPart, subPart, copyPart and combine now return success of the operation instead of throwing an exception.**  
hsbCAD2013 build 18.0.17 and hsbCAD2012 build 17.1.26

**19-Sept-2011 Added MapX as [global method](#)**  
hsbCAD2013 build 18.0.13 and hsbCAD2012 build 17.1.23.

**13-Sept-2011 Added [EcsMarker](#) entity type method**  
hsbCAD2013 build 18.0.10 and hsbCAD2012 build 17.1.22.

**13-Sept-2011 Added setCoordSysOnly method to [MassGroup](#)**  
hsbCAD2013 build 18.0.10 and hsbCAD2012 build 17.1.22.

**25-Aug-2011 Added [ERoofplane](#) type method**  
hsbCAD2013 build 18.0.6.

**15-Aug-2011 Added [TestReport](#) class.**  
hsbCAD2013 build 18.0.5.

**1-Aug-2011 [ModelX](#)::callExporter added with entityset**  
hsbCAD2011 build 16.4.8 and hsbCAD2012 build 17.1.15.

**1-Aug-2011 Added [SurfaceQualityStyle](#)::quality**  
hsbCAD2012 build 17.1.14.

**29-July-2011 Added new [ChildPanel](#) dbCreate.**  
hsbCAD2012 build 17.1.14.

**28-July-2011 Added [Nester](#) silent flag**  
hsbCAD2012 build 17.1.13.

**1-July-2011 Added [ModelX](#)::callExporter**  
hsbCAD2011 build 16.4.3 and hsbCAD2012 build 17.1.9.

**28-June-2011 Added [PrEntity](#)::getPickFirstSS and setPickFirstSS**  
hsbCAD2012 build 17.1.8.

**28-June-2011 Added [Entity](#)::hyperlink and setHyperlink.**  
hsbCAD2012 build 17.1.8.

**28-June-2011 Allowed [Display](#)::dxfOut and dwgOut during \_bOnDbCreated.**  
hsbCAD2012 build 17.1.8.

**10-June-2011 Added timberName, timberGrade, timberMaterial and dTongueHeight to [ExtrProfile](#).**  
hsbCAD2012 build 17.1.4.

**10-June-2011 Added dimColor to [SipComponent](#).**

hsbCAD2012 build 17.1.4.

**6-June-2011 Added [MasterPanel](#)::surfaceQualityOverrideTop and Bottom as well as [MasterPanelStyle](#)::surfaceQualityTop and Bottom**

hsbCAD2012 build 17.1.2

**11-May-2011 Added SubMapX for [DictObject](#).**

hsbCAD2012 build 17.0.54.

**5-May-2011 Added new constructor for [Klingschrot](#) tool.**

hsbCAD2012 build 17.0.52.

**24-April-2011 Added [SurfaceQualityStyle](#)**

hsbCAD2012 build 17.0.48.

**24-April-2011 Added [Sip](#)::surfaceQualityOverrideTop and Bottom as well as [SipStyle](#)::surfaceQualityTop and Bottom**

hsbCAD2012 build 17.0.48.

**24-April-2011 Added [Sip](#)::profCncNesting**

hsbCAD2012 build 17.0.48.

**15-April-2011 Added [MultiPageStyle](#).**

hsbCAD2012 build 17.0.47.

**14-April-2011 Added [Element](#)::triggerGenerateConstruction method.**

hsbCAD2012 build 17.0.46.

**14-April-2011 Corrected the resolving of [Blocks](#) to not look into loaded xrefs.**

hsbCAD2012 build 17.0.46.

**11-April-2011 Added helper class [ElemNumber](#) and added [Element](#).setNumber.**

hsbCAD2012 build 17.0.45.

**10-April-2011 Added [BlockRef](#).**

hsbCAD2012 build 17.0.44.

**8-April-2011 Added [CncCurveStyle](#) and [CncCurveEnt](#).**

hsbCAD2012 build 17.0.44.

**25-Feb-2011 Added Sip tool [PanelWirechase](#).**

hsbCAD2012 build 17.0.41.

**25-Feb-2011 Added method setSideToCenter for [ElemMill](#) and [ElemSaw](#).**

hsbCAD2012 build 17.0.41.

**25-Feb-2011 Added method exposed to [ElementWall](#).**

hsbCAD2012 build 17.0.41 and hsbCAD2011 build 16.3.7.

**23-Feb-2011 Added method isVisible to [Entity](#)**

hsbCAD2012 build 17.0.40 and hsbCAD2011 build 16.3.5.

**21-Jan-2011 Added class [Klingschrot](#).**

hsbCAD2012 build 17.0.24

**10-Jan-2011 Added class [MassElement](#) and [MassGroup](#).**

hsbCAD2012 build 17.0.23

**29-Dec-2010 Added a [filterBeamsCapsuleIntersect](#) with LineSeg and radius as static method to [Beam](#).**

hsbCAD2012 build 17.0.21

**28-Dec-2010 Added class [FastenerAssemblyEnt](#) that refers to a [FastenerAssemblyDef](#).**

hsbCAD2012 build 17.0.21

**27-Dec-2010 Added class [FastenerGuideline](#) and methods to [ToolEnt](#).**

hsbCAD2012 build 17.0.21

**23-Dec-2010 Argument added to specify the version when firing [Display](#)::writeToDxfFile and [writeToDwgFile](#).**

hsbCAD2011 build 16.2.14

**5-Dec-2010 Method added to [Grid](#) to align a coordSys to the grid.**

hsbCAD2011 build 16.2.9.

**2-Dec-2010 The tool [ConvexConcaveProfile](#) can now act as a none end tool**

hsbCAD2011 build 16.2.8.

**26-Nov-2010 New tool [Ari](#) added**

hsbCAD2012 build 17.0.15

**25-Oct-2010 During the [ShopDrawViewDataShowSet](#) event, the [orientation](#) of the view can be modified.**

hsbCAD2011 build 16.1.27.

**22-Oct-2010 Multipage entity available in block resident ts'l's during shopdrawing as [entCollector](#).**

hsbCAD2011 build 16.1.26

**8-Oct-2010 [getEntitiesWithPosnum](#) methods added to [Entity](#)**

hsbCAD2011 build 16.1.18

**8-Oct-2010 [AssignPosnum](#) and [releasePosnum](#) methods added to [GenBeam](#)**

hsbCAD2011 build 16.1.18

**4-Oct-2010 Added new tool [SimpleScarf](#)**

hsbCAD2011 build 16.1.15

**13-Sept-2010 Added new shopdraw recalc event: [ShopDrawViewDataShowSet](#).**

Added hsbCAD2011 build 16.1.3

**7-Sept-2010 Corrected the value of [Opening](#)::headHeight.**

hsbCAD2011 build 16.0.105.

**31-August-2010 Added a .Net method to [assure an extrusion profile](#) is loaded.**

Added hsbCAD2011 build 16.0.102.

**1-July-2010 Added methods to read the [OpeningRoof](#) catalog.**  
Added hsbCAD2011 build 16.0.59 and hsbCAD2010 build 15.3.27

**1-July-2010 Methods added to construct a cone [Body](#)**  
Added in hsbCAD2011 build 16.0.59

**30-June-2010 Methods added to query the attached property sets of a [DictObject](#)**  
Added in hsbCAD2011 build 16.0.57

**25-June-2010 New types added to [AnalysedBeamCut](#).**  
`_kABC5AxisBirdsMouth` and `_kABC5AxisBlindBirdsMouth` are added in hsbCAD2011 build 16.0.52

**25-June-2010 Added methods to read the [ElementRoof](#) catalog.**  
Added hsbCAD2011 build 16.0.52 and hsbCAD2010 build 15.3.25

**25-June-2010 Added [ConvexConcaveProfile](#) tool.**  
Added hsbCAD2011 build 16.0.52.

**21-June-2010 Added `setLength` on [HalfCut](#) tool.**  
Added hsbCAD2011 build 16.0.47.

**12-June-2010 Added `dbCreate`, `setScale` and `setCoordSys` on [MvBlockRef](#)**  
Added hsbCAD2011 build 16.0.43.

**8-June-2010 Added method in [TslInst](#) to trigger recalculation on posnum being changed.**  
See also [compare](#). Added hsbCAD2011 (build 16.0.41) and hsbCAD2010 (build 15.3.22).

**17-May-2010 Added new [PLine](#)::`intersectPLine` and [PLine](#)::`extend`.**  
Added hsbCAD2010 build 15.3.15 and hsbCAD2011 build 16.0.31.

**14-May-2010 Added the originator, `modelDescription`, `materialDescription` and `additionalNotes` to [TslInst](#).**  
Added hsbCAD2011 build 16.0.31.

**14-May-2010 Extended command [Hsb\\_ListTslWords](#) for [Notepad++](#) support.**  
This command allows to generate a text file that can be used for syntax coloring in text editors (hsbCAD2011 build 16.0.31).

**4-May-2010 Added members to [Group](#): `nEquivalentStory`**  
Added hsbCAD2011 build 16.0.25.

**2-May-2010 Added members to [OpeningSF](#) for `styleNameSF` and `nFloorsToCarry`**  
Added hsbCAD2011 build 16.0.23.

**2-May-2010 Added new [Shopdraw template rule](#).**  
Added hsbCAD2011 build 16.0.23.

**2-May-2010 Added new [Shopdraw recalc triggers](#).**  
Added hsbCAD2011 build 16.0.23.

**30-Apr-2010 Added new methods to add and retrieve a list of entities to [Map](#). Also build in support for keys with a mapPath.**

A mapPath eg "aa\bb" for key "strKey" would result in a composite key: "aa\bb\strKey".  
Added hsbCAD2011 build 16.0.23.

**27-Apr-2010 Added new [Sip::getWirechaseSegs](#)**

Added hsbCAD2010 build 15.3.14 and hsbCAD2011 build 16.0.23.

**26-Apr-2010 Added new [Entity::createRealBodySatFile](#)**

Added hsbCAD2010 build 15.3.13 and hsbCAD2011 build 16.0.23.

**20-Apr-2010 Added new [Body::createSatFile](#)**

Added hsbCAD2010 build 15.3.12 and hsbCAD2011 build 16.0.23.

**8-Apr-2010 Added new [AnalysedPlaning](#).**

Added hsbCAD2011 build 16.0.21.

**7-Apr-2010 Added getClassificationMap method to [Entity](#) and [DictObject](#).**

Added hsbCAD2011 build 16.0.21.

**7-Apr-2010 Added roofNumber to [ERoofPlane](#)**

Added hsbCAD2010 build 15.3.8 and hsbCAD2011 build 16.0.21.

**6-Apr-2010 Added new [MvBlockDef](#) and [MvBlockRef](#).**

Added hsbCAD2011 build 16.0.21.

**30-Mar-2010 Added new [AnalysedLogDove](#) and [AnalysedChamfer](#).**

Added hsbCAD2011 build 16.0.19.

**30-Mar-2010 Added new [HardWrComp](#) type, and array of it to [ToolEnt](#).**

Added hsbCAD2011 build 16.0.18.

**24-Feb-2010 Added [global command](#) to push a command on the command stack of the current document.**

This command is called: "pushCommandOnCommandStack" (added hsbCAD2011 build 16.0.11)

**11-Feb-2010 Added new [LineSeg](#) length and [findCapsuleIntersections](#).**

Added hsbCAD2010 build 15.1.39, hsbCAD2009+ build 14.4.11.

**10-Feb-2010 Added new [LineSeg](#) distanceTo other LineSeg and Point3d.**

Added hsbCAD2010 build 15.1.38, hsbCAD2009+ build 14.4.11.

**1-Feb-2010 Added new [LineSeg](#) ClosestPointTo other LineSeg.**

Added hsbCAD2010 build 15.1.31, hsbCAD2009+ build 14.4.11.

**27-Jan-2010 Added new state variable [\\_bOnElementRead](#).**

This is set during HSB\_READCONSTRUCTION command. (added hsbCAD2011 build 16.0.11)

**24-Jan-2010 [Performance and profiling](#)**

(added hsbCAD2011 build 16.0.9)

**20-Jan-2010 Change the process of the tsl filter in the [shopdraw](#) multipage insertion**

Changes applicable from hsbCAD2009+ (build 14.4.6) and hsbCAD2010 (build 15.1.26)

**28-Dec-2009 Added hsbCadAPI in [managed API](#).**

(added hsbCAD2011 build 16.0.3)

**16-Dec-2009 Added other variant of [TslInst::callMapIO](#) with strExecuteKey**  
(added hsbCAD2010 build 15.1.19)

**11-Dec-2009 Added new makeFolder, see [spawn](#).**  
(added hsbCAD2010 build 15.1.16)

**6-Nov-2009 Added method openingDescr to [OpeningLog](#) and renamed for [OpeningSF](#)**  
(added hsbCAD2009+ build 14.2.10, hsbCAD2010 build 15.1.5)

**29-Oct-2009 Added nester API in Tsl.**  
The nester API consists of [NesterChild](#), [NesterMaster](#), [NesterData](#), [NesterCaller](#)  
(added hsbCAD2010 build 15.1.4)

**29-Oct-2009 Added method setFromHandle in [Entity](#).**  
(added hsbCAD2009+ build 14.2.8, hsbCAD2010 build 15.1.3)

**14-Oct-2009 The [findFile](#) and the [CallDotNet](#) are extended to auto search Company\Tsl and <hsbInstall>\Content subfolders.**  
(added hsbCAD2009+ build 14.1.43, hsbCAD2010 build 15.0.18)

**28-Sept-2009 Added new [AnalysedBeamCut](#) subtypes \_kABCRabbit and \_kABCDado.**  
(added hsbCAD2009+ build 14.1.39, hsbCAD2010 build 15.0.15)

**24-Sept-2009 Added new [AnalysedRabbit](#)**  
(added hsbCAD2010 build 15.0.15)

**23-Sept-2009 Added the subMapX to [Entity](#)**  
(added hsbCAD2010 build 15.0.13)

**21-Sept-2009 Added the sequenceNumber property to [TslInst](#).**  
(added hsbCAD2009+ build 14.1.35, hsbCAD2010 build 15.0.12)

**15-Sept-2009 Added [showdraw categories](#) types, and [ActiveRule](#) submap.**

**25-August-2009 Added cuttingType to [ElementWall](#).**  
(added hsbCAD2010 build 15.0.12 and hsbCAD2009+ build 14.1.26)

**25-August-2009 Added [findFile](#) method.**  
(added hsbCAD2010 build 15.0.12 and hsbCAD2009+ build 14.1.26)

**25-August-2009 Added [bits](#) method that returns either 32 or 64 depending on memory model of the current acad process.**  
(added hsbCAD2010 build 15.0.12 and hsbCAD2009+ build 14.1.26)

**2-June-2009 Added new [MetalPartCollectionEnt](#) and [MetalPartCollectionDef](#).**  
(added hsbCAD2010 build 15.0.3)

**26-May-2009 The [DimRequestObholz](#) has method setUseVerticalDimensions**  
(added hsbCAD2010 build 15.0.0)

**26-May-2009 The [DimRequestText](#) has method setShowLeaderLine.**

(added hsbCAD2009+ build 14.1.2)

**26-May-2009 The [Mark](#) tool has a method `setPosnumBeam`.**

(added hsbCAD2009+ build 14.1.1)

**30-April-2009 Tsl import in Tsl manager now supports multiple selection in the file dialog.**

(added hsbCAD2009+ build 14.0.82)

**29-April-2009 The Tsl editor help button now jumps to the index page in the help if something is selected when firing help.**

(added hsbCAD2009+ build 14.0.81)

**29-April-2009 Added `setXAxisDirectionInXYPlane` method to [Sip](#) type.**

(added hsbCAD2009+ build 14.0.81)

**28-April-2009 Added `callMapIO` method to [TslInst](#) type.**

(added hsbCAD2009+ build 14.0.79)

**27-April-2009 Added dxx file support for [Map](#) type.**

(added hsbCAD2009+ build 14.0.78)

**24-April-2009 Added to [TslInst](#) and as [global insert function](#) the set and get property values from a Map.**

(added hsbCAD2009+ build 14.0.78)

**23-April-2009 Added shopdraw tsl rule Map edit with [OnMapIO](#).**

(added hsbCAD2009+ build 14.0.78)

**17-April-2009 Added new [ElemConstructionMap](#)**

(added hsbCAD2009+ build 14.0.76)

**15-April-2009 Added new [ModelX](#) type.**

(added hsbCAD2009+ build 14.0.75)

**9-April-2009 Added new [AnalysedArc](#)**

(added hsbCAD2009+ build 14.0.73)

**8-April-2009 Added new [AnalysedScarfJoint](#)**

(added hsbCAD2009+ build 14.0.73)

**8-April-2009 Added new [TrussDefinition](#) and [TrussEntity](#)**

(added hsbCAD2009+ build 14.0.73)

**8-April-2009 Added new [CollectionDefinition](#) and [CollectionEntity](#)**

(added hsbCAD2009+ build 14.0.73)

**30-Mar-2009 Added new [AnalysedSimpleScarf](#)**

(added hsbCAD2009+ build 14.0.69)

**30-Mar-2009 Added methods `pEnvelope`, `pOpenings`, `addOpening` and `pEnvelopeCnc` to [Sip](#).**

(hsbCAD2009+ build 14.0.69)

**22-Mar-2009 Added new [MasterPanel](#) and [ChildPanel](#) and [MasterPanelStyle](#)**  
(added hsbCAD2009+ build 14.0.68)

**16-Mar-2009 Added methods to [OpeningSF](#)**  
(added hsbCAD2009+ build 14.0.63)

**9-Mar-2009 The stack of predefined variables has been optimized.**  
An optimization step during the compilation has been improved. A predefined variable will be removed during compilation if it is not used further on. Persistent variables like \_Map, \_Pt0,... are treated the same. This means they are not changed if they are not used. Previously this optimization was also present, but not for predefined arrays and not for a number of tagged predefines.  
(hsbCAD2009+ build 14.0.63)

**9-Mar-2009 Added new [BlockNames](#)**  
(added hsbCAD2009 build 13.6.10 and hsbCAD2009+ build 14.0.63)

**6-Mar-2009 Added new [AnalysedAri](#)**  
(added hsbCAD2009+ build 14.0.63)

**5-Mar-2009 Added new [AnalysedDovesugi](#)**  
(added hsbCAD2009+ build 14.0.62)

**4-Mar-2009 Added new [AnalysedKamatsugi](#)**  
(added hsbCAD2009+ build 14.0.61)

**7-Feb-2009 Added dbCreateAsMassElement to [Body](#).**  
Appends the body as entity. It is for debugging only.  
(added hsbCAD2009+ build 14.0.45)

**7-Feb-2009 Added hatchHidden property to [Beam](#).**  
(added hsbCAD2009+ build 14.0.41)

**28-Jan-2009 The tool [RevolutionMill](#) added.**  
(added hsbCAD2009+ build 14.0.36)

**22-Jan-2009 The [Entity](#)::optionAssignSameRuleSet method added.**  
(added hsbCAD2009+ build 14.0.34)

**7-Jan-2009 [Shopdraw](#) multipage insertion can be tuned by a special Tsl.**  
(added hsbCAD2009+ build 14.0.30)

**6-Jan-2009 Added method [PLine](#)::createConvexHull.**  
(added hsbCAD2009+ build 14.0.30)

**6-Jan-2009 Added curvedStyle to [Beam](#).**  
(added hsbCAD2009+ build 14.0.30)

**5-Jan-2009 Added new [AnalysedChamferedLap](#).**  
(added hsbCAD2009+ build 14.0.30)

**26-Dec-2008 Exposure of the [option system](#): the option event [\\_bOnOptionChanged](#), and an [Entity](#)::optionEvaluate method.**

- 25-Nov-2008 Added hatchSection to [Beam](#).
- 13-Nov-2008 Added new [AnalysedFreeProfile](#).
- 23-Oct-2008 Added new [AnalysedMarker](#).
- 9-Oct-2008 Added new [AnalysedParHouse](#).
- 9-Oct-2008 Added new [AnalysedShoulderTenon](#).
- 8-Oct-2008 Added new [AnalysedComplexProfile](#).
- 15-July-2008 Added the [DimRequest](#)::addInternalSet method.
- 22-July-2008 Extended the [DimRequestPLine](#) with setLineType.
- 15-July-2008 Added the [Sip](#)::lumberInstallList.
- 3-July-2008 Added the [Group](#)::dbErase.
- 1-July-2008 Added the [CurvedStyle](#) DictObject type.
- 1-July-2008 Some methods added to [TslInst](#): getListOfPropNames, hasPropInt,....
- 1-July-2008 Some methods added to [PLine](#): length, getDistAtPoint, getPointAtDist.
- 29-June-2008 New method stretchStaticTo for [Beam](#).
- 28-June-2008 Moved getPLine to [Entity](#) base class of [EntPLine](#).
- 25-June-2008 Added [DimRequestChain](#).
- 8-June-2008 Provided access to the Multipage entity during [shopdraw](#) generation.
- 5-June-2008 Moved the subMap and setSubMap methods from [GenBeam](#) down to [Entity](#).  
Added hasSubMapContainer, removeSubMap and subMapKeys methods as well.
- 3-June-2008 Added the [Group](#)::blsDeliverableContainer methods.
- 30-May-2008 Possibility to modify the list of Analysed tools passed on to the [Btl](#) generation.
- 18-May-2008 Added new [AnalysedLogNotch](#) and [AnalysedDiagonalNotch](#).
- 14-May-2008 Added findDescriptionForDetCode method to [ElementWallSF](#).
- 7-May-2008 Added new [AnalysedConvexConcaveProfile](#).
- 21-April-2008 Extended the methods of [OpeningSF](#).
- 21-April-2008 Extended the types of [TirolerSchloss](#) tool.
- 17-April-2008 Added dbRename to [Group](#).

17-April-2008 Added new [RoundWoodMill](#) tool.

10-April-2008 Added new [AnalysedSlot](#).

24-Mar-2008 New method [Mark](#)::setTextHeight added.

21-Mar-2008 New method [Opening](#)::parentEntity returns the Wall or the Opening assembly.

11-Mar-2008 New method [Beam](#)::composeBeamPacks returns an array of [EntityCollection](#).

3-Mar-2008 [ElementWall](#) ptStartOutline and ptEndOutline get and set added.

Since the start and end points of the [Wall](#) do no longer match with outline of the ElementWall, the get and set start and end points of the outline have been added.

29-Feb-2008 Added new [AnalysedSpecialMill](#).

28-Feb-2008 Added new [AnalysedDove](#).

25-Feb-2008 Added new [AnalysedMortise](#).

7-Feb-2008 Added the possibility to [Display](#) on a layer.

6-Feb-2008 Added version to [TslInst](#).

29-Jan-2008 Added a method to retrieve the most relevant [Grid](#) for an [Entity](#).

29-Jan-2008 Added some methods to [Grid](#).

28-Jan-2008 Added new [AnalysedHouse](#).

26-Jan-2008 Added the [ShopDrawView](#) type.

26-Jan-2008 Added setToolIndex to [SpecialMill](#) tool.

9-Jan-2008 Added the [TirolerSchloss](#) tool.

8-Jan-2008 Added the ability attach and remove a property set to [Entity](#).

20-Dec-2007 Added the ability to query the property sets attached to [Entity](#).

18-Dec-2007 Added the [Slab](#) type.

12-Nov-2007 [Display](#)::draw Entity added.

12-Nov-2007 Added setWoodGrainDirection on [Sip](#).

26-Oct-2007 Important bug fix when cancelling input during custom action.

When a custom action is fired (through the [context menu](#)), the prompt functions are allowed. If during such a prompt, the input is canceled, for instance by pressing escape, then the Tsl would be left in an undefined state. This bug is fixed now. If such an event occurs, the Tsl is now restored in its original state, the state before the custom action was fired.

25-Oct-2007 Added setDFloorHeight on [Group](#).

**10-Oct-2007** Added new [AnalysedDoubleCut](#) and [AnalysedDrill](#).

**8-Oct-2007** The [ERoofPlane](#) has a new member function `p1WallBoundary`.

**28-Sept-2007** The [Map](#) can now carry his own key and name

The key is accessed through `setMapKey` and `getMapKey`. The name is accessed through `getMapName` and `setMapName`.

**28-Sept-2007** The [AnalysedBeamCut](#) subtypes and [AnalysedCut](#) subtypes have been added as predefines.

**25-Sept-2007** Extended the [DimRequestPoint](#).

**25-Sept-2007** Added new [DimRequestPLine](#).

**24-Sept-2007** A method called `getBeamQuaderIntersectPoints` was added to [AnalysedBeamCut](#).

**24-Sept-2007** The method `hasIntersection` on [Line](#) has been extended.

The method now has an additional call by reference argument that allows to retrieve the intersection point directly.

**17-Sept-2007** A new mode is added for the [Dim\\_kDimClassic](#).

This mode will not do collision checking of the dimensioning text, but will show the dimension exactly as a classic autocad dimension.

**10-Sept-2007** [Global function](#) `getTickCount` added.

The function returns the number of milliseconds that have elapsed since the system was started, up to 49.7 days.

**7-Sept-2007** Added the `removeGenBeamConnection` method to [ToolEnt](#).

**16-August-2007** Added the `dispRepNames` method to [Entity](#).

**14-August-2007** Added some functions to modify the vertices and edges of a [PlaneProfile](#).

**25-June-2007** Added the `CutRib` values to [Beam](#).

**19-June-2007** The contents of a [Display](#) can be dumped into a DXF or DWG file.

This can be done by calling the method `writeToDxfFile` or `writeToDwgFile` on the Display.

**6-June-2007** A member function added to [Beam](#) to compare with other beams.

The function is called `isEqualComparingPosnumCriteria`.

**29-May-2007** The [DimRequest](#) type is introduced.

The DimRequest, and derived ones contribute to the new way of generating shop drawings.

**22-May-2007** Added the methods to get and set a [Map](#) from a string content.

**15-May-2007** The [Quader](#) type is introduced.

**10-May-2007** The method `filterClosePoints` added to [Line and Plane](#).

**10-May-2007 The [AnalysedBeamCut](#) type is introduced.**

The AnalysedBeamCut type represents an evaluated tool of type BeamCut. The type is derived from [AnalysedTool](#).

**10-May-2007 The [AnalysedCut](#) type is introduced.**

The AnalysedCut type represents an evaluated tool of type Cut. The type is derived from [AnalysedTool](#).

**5-May-2007 The [predefined status variable](#) \_bOnGenerateShopDrawing added.****25-April-2007 [Beam](#) ptCenSolid method added.****24-April-2007 The [String](#) methods are improved.**

String methods that did not return anything previously, now return their own string value. This allows to write statements similar to

```
String str = String("test").makeUpper();  
which was previously not possible.
```

**24-April-2007 Extended the set of [mathematical](#) functions.**

The following global methods are added: exp, log, log10, ceil, pow, sqr, sign, round.

**20-April-2007 The [AnalysedTool](#) type is introduced.****16-April-2007 Added the <MYPOS> substitution for the [Mark](#) tool.**

When the <MYPOS> string is part of the text placed by the Mark tool, that substring is replaced by the posnum number of the beam itself.

**29-March-2007 The [GenBeam](#) has a new member function called [getToolsOfTypeCncExport](#).****29-March-2007 The [CncExport](#) tool has been extended with 2 members.**

The map and the tool name which are used inside the constructor are now retrievable.

**29-March-2007 Improved the Tsl editor find/replace functionality a bit.**

Now the find dialog can be triggered by pressing Ctrl-F. The find will start from the cursor position. It is also possible to press F3 to in the editor field, to trigger the "Find Next" action.

**21-March-2007 Added the load bearing method to [ElementWallSF](#).****2-March-2007 The Tsl definition has a new toggle called: "Store state in dwg, do not recalc at dwgin".**

See the topic: [Store state in dwg](#).

**15-Febr-2007 Added [entitiesFromMultiElementBuild](#) function to [SingleElementRef](#).****14-Febr-2007 Enhanced the way the enumerated properties are stored in the drawing.**

Tsl enumerated properties will remember their value, even if the list of available values is changed. When a Tsl instance has a enumerated property, it used to remember the index of its value in the list. Now this is enhanced  
that the ts1 instance will remember its value. If the value is not found, the index is used.

**7-Febr-2007 Added the [MapObject](#) type.**

**29-Jan-2007 Added the filterBeamsHalfLineIntersectSort method to [Beam](#).**

This method allows to easily integrate a new beam into an existing frame. It works well in conjunction with stretchDynamicTo.

**25-Jan-2007 Added the stretchDynamicToMultiple method to [Beam](#).**

This method allows to add a hexcontact tool to a beam.

**25-Jan-2007 Added the panhand and setPanhand methods to [GenBeam](#).**

**3-Jan-2007 Global function [callDotNetFunction2](#) allows to call a .Net function with the Map as argument and return value.**

----- 27-November-2006 Release of hsbCad 12.1 -----

**25-Nov-2006 On ElementWall split in length, the event \_bOnElementListModified is fired, see [predefined status variables](#).**

**9-Nov-2006 The property getAutoPropertyMap added to [Entity](#).**

**9-Nov-2006 The nTransType for Vector3d in the [Map](#) has been changed into nScaleType.**  
The modification has been done in a compatible way.

**9-Nov-2006 The realBody function of [Entity](#) now also has the possibility to specify the display set name.**

**9-Nov-2006 Added the function to set the dummy state of a [GenBeam](#) called setBIsDummy.**

**3-Nov-2006 Function setToAlignWorldToPlane and setToAlignPlaneToWorld added to [CoordSys](#).**

**15-Oct-2006 The [master slave insert](#) pattern explained.**

**13-Oct-2006 The [auto insert](#) of TSL's explained.**

**28-Sept-2006 Global function [callDotNetFunction1](#) allows to call a .Net function.**

**13-Sept-2006 Function [GenBeam](#)::removeToolsStatic added.**  
This function allows to remove all static tools of a certain type.

**8-Sept-2006 Function [Entity](#)::realBody extended with optional view direction.**

**8-Sept-2006 Added the [ElementMulti](#) and [SingleElementRef](#) types.**

**8-Sept-2006 Function createRectangle added to [PLine](#).**

**8-Sept-2006 Function segmentMinMax added to [Element](#).**

The function returns a segment that describes the diagonal of a box in the coordSys directions with the dimensions of the element.

**5-Sept-2006 Added predefined \_HatchPatterns for [Hatch](#).**

This array is an array of strings that contains all names of predefined hatch patterns available in the drawing.

4-Sept-2006 Added subMap get and set to [GenBeam](#).

3-Sept-2006 The [TslInst](#) can now read and write its properties to a catalog. Also the list of all catalog entries is available.

The member functions of TslInst called setPropValuesFromCatalog and setCatalogFromPropValues allow for advanced master-slave Tsl design.

28-August-2006 Possible to specify the angle of the [Hatch](#) pattern.

27-August-2006 Through a special [Beam](#)::dbCreate it is possible to convert a solid into a Beam.

26-August-2006 Function dbCreate added to [EntPLine](#).

26-August-2006 Added the [LineTypes](#) array.

25-August-2006 Added the [BeamTypes](#) array (see [GenBeam](#)::type function).

25-August-2006 Added the [ElemNailCluster](#) tool.

18-August-2006 Added the [PanelSplit](#) tool.

21-July-2006 Added [Hatch](#) type to use with draw of [PlaneProfile](#) in [Display](#).

20-July-2006 Function shadowProfile added to [Body](#).

10-July-2006 Added [ElementWallSF](#)::distributionZone and distributionType

8-July-2006 Added [ERoofPlane](#)::dbCreate and setCode routine.

8-July-2006 Added global function [setCatalogFromPropValues](#).

The setCatalogFromPropValues function allows to write the property values to the catalog.

6-July-2006 Added [Wall](#)::dbSplit routine.

28-June-2006 Added a [ElementWall](#)::ptArrow and [ElementWall](#)::setPtArrow function.

28-June-2006 Added a [LogCourse](#) type.

A LogCourse collects all beams or logs of one course. The list of log courses is available from [ElementLog](#)::logCourses.

20-June-2006 Added a new [Beam](#)::filterBeamCapsuleIntersect routine.

16-June-2006 Added the types [SipEdge](#), as well as the [Sip](#)::sipEdges function and [Sip](#)::stretchEdgeTo function.

14-June-2006 Added the types [SipStyle](#) and [SipComponent](#), as well as the [Sip](#)::style function.

26-May-2006 Added member function to [string](#) called formatTime to get date and time as string.

10-May-2006 Added member function to [Entity](#) called notes and setNotes.

These functions provide access to the property called Notes, accessible in the OPM, tab extended data.

**9-May-2006 Added member function to [Dove](#) tool called setAddKeyhole.**

**9-May-2006 The function TN added, as an extension to the T function.**

The T function translates the argument, the TN function also does that, but in addition, a newline character "\n" is placed in front of it.

**3-May-2006 Added the [ExtrProfileCut](#) tool.**

**30-Apr-2006 Added new constructor for tool [HalfCut](#).**

The new constructor contains an additional vector, identifying an accompanying cut. It allows for easy calculation of DoubleCut type of HalfCut.

**10-Apr-2006 Custom [context menu](#) actions.**

The tsl script can define its own list of custom context menu actions.

**8-Apr-2006 Added member function [Entity](#)::setDrawOrderToFront.**

**8-Apr-2006 Added global function [setPropValuesFromCatalog](#).**

The setPropValuesFromCatalog function allows to load the property values saved in a catalog. It works well together with the variable \_kExecuteKey.

**7-Apr-2006 Added argument to the [lisp Hsb\\_ScriptInsert](#).**

The scriptinsert function has been given an argument that can be passed to the script as variable \_kExecuteKey. Also the automatic loading of TSL's includes the searching of subfolders inside the Company/Tsl folder.

**1-Apr-2006 Member function [GenBeam](#)::envelopeBody revised.**

The envelopeBody has now possible 2 arguments: bUseExtrProfile (default FALSE), and bUseCuts (default FALSE).

**1-Apr-2006 Added member function [ToolEnt](#)::ptOrg.**

**8-Feb-2006 New type added [ERoofPlaneOpening](#)**

**8-Feb-2006 Added dFloorHeight to [Group](#)**

**8-Feb-2006 Added code to [ERoofplane](#)**

**27-Jan-2006 showDialog and showDialogOnce can now specify the catalog entry to use.**

This allows also the author to specify that the showDialog should start with the default values specified in the script.

**11-Jan-2006 Added the types [DictObject](#) and [ExtrProfile](#).**

**8-Jan-2006 Added the [ComplexProfile](#) tool.**

**22-Dec-2005 [Predefined\\_bOnWriteEnabled](#) added.**

During dwg file open, the TSL instances are executed to calculate their graphical representation. But during this execution, the TSL is not allowed to change anything in the Autocad database: it is not allowed to do dbCreate, dbErase... actions.

The predefined variable `_bOnWriteEnabled` allows to detect if the script is running in the normal mode (`_bOnWriteEnabled == TRUE`), or in this dwg file open mode (`_bOnWriteEnabled == FALSE`).

**13-Dec-2005 Element [construction directives](#) and [ElemConstructionBeam](#) only shown in debug.**

The construction directives, and the tool `ElemConstructionBeam` will not be shown anymore in normal display mode. If debugging is turned on, or in stepping through mode, the construction directives and the `ElemConstructionBeam` tool are shown.

**19-Nov-2005 [setCncSplinterFree](#) is been added as memberfunction to [Beam](#).**

**7-Nov-2005 Lock and setLock functions added to [Element](#).**

**28-Oct-2005 New type [Grid](#) added.**

The `Grid` exposes the `Hsb_Grid` entity. Also the `Group::grid` member function is added to find the Grid that belongs to a group.

**26-Oct-2005 The member functions beam(), sheet(), genBeam(), nailline(),... from [Element](#) have been changed.**

These routines used to return all the entities from a certain type that belong to the element. Now an additional filter is added that returns only those entities that belong to model space. This way, entities that belong to a block will not appear in the returned arrays.

**16-Oct-2005 Add a tool as static tool to a [GenBeam](#) with addToolStatic.**

**12-Oct-2005 The member function of [setFlatWidth](#) of [HousedDove](#) has been added.**

**27-Sep-2005 A [predefined status](#) variable was added `_bOnDbCreated`.**

This integer variable is set to TRUE during the first execution of the `TslInst`, after it has been appended to the Autocad database.

**27-Sep-2005 The member function of [NailLine](#) calculateAllowedNailLineSegments has been extended.**

**12-Sep-2005 The [TslInst](#) type has been extended.**

The member functions `setPropInt`, `setPropDouble` and `setPropString` have been added. Also the `dbCreate` has been extended with an optional argument `Map`.

**9-Sep-2005 The [ElementRoof](#) type has been extended.**

The member functions to get and set values for `KneeWall2`, `WallPlate2` and `Strut2` have been added.

**2-Sep-2005 The default behaviour of how the `tsl` instance copies and is erased with the beams can be overwritten.**

The behaviour can be overwritten with the function [setEraseAndCopyWithBeams](#).

**30-Aug-2005 The [Entity](#) has a member `dbCreate` from a `Map`.**

This `dbCreate` allows to create an entity from its `Map` description. Currently only Beam entities are supported, but others will follow.

**30-Aug-2005 [Map](#) class has been extended.**

The `Map` has been made tolerant to duplicate keys. It is now possible to append items to the map.

Also submaps can be added with type `_kObject`.

**17-Aug-2005** With the dimstyle of [Display](#) the linear scaling can be overwritten.

**17-Aug-2005** [Coordsys](#) has a scale method.

The scale method returns the determinant to the power 1/3, which corresponds to the scaling if it is a orthogonal transformation.

**13-July-2005** Now possible to change the thickness of [Sheet](#).

The thickness can be set through `setDH`.

**12-July-2005** Added `setDPosZOutlineFront` and `Back` to [Element](#).

**12-July-2005** Added `findFoamRemovalPanelEdge` to [Sip](#).

Also added normal as member function to [Plane](#) and `blsZeroLength` as function to [Vector3d](#).

**8-July-2005** Added `pt1` and `pt2` member to [ElemText](#).

**8-July-2005** Extended the [PanelStop](#) tool.

**8-July-2005** Added depth parameter to [FreeProfile](#) tool.

**5-July-2005** Added additional [round](#) type for housing and mortise.

**5-July-2005** Added `removeAt` member function of [Map](#).

**1-July-2005** Changed the return type of [Sheet](#)::`joinRing` into an array of Sheets.

**30-June-2005** `DeltaDistribution` added to [ElementWallSF](#).

**27-June-2005** Items can be inserted and removed from TSL [arrays](#).

**27-June-2005** New beam tools [MarkerLine](#) and [LogNotch](#) added.

**26-June-2005** Functions `getSlice`, `combine`, `decomposeIntoLumps`, `hideDisplay` and `intersectPoints` implemented on [Body](#).

**24-June-2005** The [Opening](#) type in TSL refers to ADT opening, but also to ADT opening assembly.

**24-June-2005** Project data can be set through some [global](#) functions.

**23-June-2005** Function for array of LineSeg's added.

`ProjectLineSegs` added to [Plane](#), and `transformLineSegs` added to [CoordSys](#), and draw `lineSegments` added to [Display](#).

**22-June-2005** New base type introduced [ToolEnt](#).

The [TslInst](#), [NailLine](#) and [ERoofPlane](#) entities are derived from this. In fact all hsbCad tool entities are derived from this.

**21-June-2005** New type [OpeningLog](#) added.

**21-June-2005** Function `realBody` implemented on [Entity](#).

21-June-2005 New type [OpeningRoof](#) added.

20-June-2005 [Element](#) member function elementGroup added.

20-June-2005 During insert, the current [group](#) is available as \_kCurrentGroup.  
Also it is possible to create a group, without adding an entity to it.

20-June-2005 Added the dbCreate to the [ElementRoof](#) type.

19-June-2005 The types [ElementRoof](#) and [ElemRoofEdge](#) are added.  
The class ElementRoof represents an hsbCad roof/floor element.

14-June-2005 The lisp insert command [Hsb\\_ScriptInsertNoPrompt](#) can now also accept a Map.

9-June-2005 Routine setCoordSys added to [Beam](#).

2-May-2005 The tsl instance now uses a display representation, called Model.  
By using an adt-display-representation, the tsl will show in the Adt object viewer, as well as in the Adt sections.

Also the Display can be set to show only in a particular adt-display-representation, by specifying the name of the adt-display-representation in the [Display](#)::showInDispRep.

24-April-2005 The dxaExportObject was added as advanced feature for [dxa output](#).

19-April-2005 Added the company location and the company detail locations (see [spawn](#)).  
These path locations are available as predefines: \_kPathHsbCompany, \_kPathHsbWallDetail,...

11-April-2005 Added the setEdgeRecessType to [PanelStop](#).

11-March-2005 Added the dbSplit and dbJoin to [Sip](#).

11-March-2005 For a roof and floor [Element](#), the plEnvelope is implemented.

11-March-2005 Possible to query the list of groups that an [entity](#) belongs to.  
This can be done through the function groups.

11-March-2005 The type [Group](#) was added.  
This class represents an hsb group from the console.

3-March-2005 The type [EntPLine](#) was added.  
This Entitiy derived class represents an Autocad database reference to a polyline.

2-March-2005 Added missing [OpeningSF](#) functions.

21-Feb-2005 Added [Element](#)::setZone function, to alter the zone data of an existing element.  
The [ElemZone](#) data members can also be changed through new functions: setDH, setColor, setMaterial, setCode, setDVar, setStrVar.

20-Feb-2005 Added [Mark](#)::suppressLine function.

17-Feb-2005 To get the layer from an [Entity](#), one can use [layerName\(\)](#).

From the layername one can also find the color index with the global function [colorFromLayer](#).

**15-Feb-2005** Added [Line](#)::orderPoints second argument, tolerance.

**21-Jan-2005** SetUseDirection added to [Drill](#) tool.

**12-Jan-2005** SetFeed added to [Chamfer](#) tool.

**11-Jan-2005** New [insertCycleCount](#) global function.

This function allows the author to keep track of the number of instances that are inserted for one user command.

**20-Dec-2004** New [FreeProfile](#) constructor with a list of points.

**14-Dec-2004** [PLine](#) addVertex added that takes a second point.

This addVertex allows to define a circle segment through 3 points.

**29-Nov-2004** Large (file) sized thumbnail bitmaps are now allowed.

**23-Nov-2004** Move/rotate/copy/mirror of E-type TSL revised (see comments in [Predefined variables E-connection](#)).

**23-Nov-2004** Added a new OPM property of the E-type called "Flip Y-axis".

This property allows to influence the \_Y0 value.

**18-Nov-2004** The setWallRoofLine added to [ElementWall](#).

**18-Nov-2004** The dFloorGroupHeight added to [Element](#).

**13-Nov-2004** The quantity and setQuantity functions added to [Element](#).

----- 28-October Release of version 2004.11 of hsbCad -----

**27-Oct-2004** [ElemMill](#) and [ElemSaw](#) has 2 new contructors, while 2 other contructors are deprecitated.

**24-Sep-2004** [Sheet](#) entities can now be joined and splitted, but also manipulated by subtracting and adding PLines.

**20-Sep-2004** The global routine [exportToDxi](#) allows to export the EMetal information to the dxi file.

**15-Sep-2004** The [ElementWallSF](#) has member functions to set and get the construction detail overwrites.

**1-Sep-2004** The [PlaneProfile](#) has been made more robust. Also the new member routine ringIsOpening was added.

**31-Aug-2004** Added the member functions hsId and setHsId to [GenBeam](#).

**28-Aug-2004** The showDialog added to [TslInst](#).

When one TSL calls a dbCreate of another one, it can also call the showDialog of this other tsl, to prompt the user to set the other tsl its values.

**27-Aug-2004** The [setTextPosition](#) added to [Mark](#).

**23-Aug-2004** New [Beam](#)::filterBeams and [GenBeam](#)::filterGenBeams routines added.

**20-Aug-2004** The [setExtrProfile](#) added to [ElemConstructionBeam](#).

**17-Aug-2004** A new TSL [S-type](#) is introduced.

This type is used to describe the hsbCad subassembly.

**16-Aug-2004** The export to an [element-dxa](#) can be specified.

This can be done through the call of exportWithElementDxa.

**11-Aug-2004** The [TsInst](#)::dbCreate can accept an array of entities.

The dbCreate has also the option to force the insertion into model space.  
Also a new member function was added to TsInst TsInst::entity().

**21-July-2004** The type [Wall](#) was added.

**20-July-2004** The type [ElementWallSF](#) was added.

**20-July-2004** For each beam tool the automatic dimensioning info can be enabled.

To do this, the following routines can be used: [setAutoDimInfo](#).

**20-July-2004** Enumerated properties important change

Enumerated properties, eg PropString initialized with an array of values, are now stored differently inside the drawing. Enumerated properties are stored by their index, and not by their value. This means that when drawings are read in an hsbCad version with another language, translated enum values will be translated differently, resulting in the correct current value of the property.

**16-July-2004** The [setProfNetto](#) added to [Element](#).

**16-July-2004** The [ElemText](#) holds the automatic generated texts around the element.

**15-July-2004** It is also possible to assign an entity to an elements floor group.

The routines to be used are the [global assignToElementFloorGroup](#) and  
[Entity](#)::[assignToElementFloorGroup](#).

**14-July-2004** The [Map](#) type can be written to or read from an Xml file.

This is done with the writeToFile and readFromFile routines.

**12-July-2004** From TSL it is now possible to fire the execution of another program.

This is done with the [spawn](#) or [spawn\\_detach](#) routine.

**12-July-2004** TSL properties dialog during insert remembers values

The TSL properties dialog when used during insert, will remember its last used values. When inserting a TSL, two sets of initial values can be used, the default values specified inside the script, or the values previously used during insert.

The TSL dialog will now remember its values previously used during insert. These values are also stored automatically in the catalog with name "\_LastInserted". The default values specified in the script are available in the catalog called "\_Default".

**11-July-2004 TSL properties dialog has catalog**

The TSL properties dialog now has the possibility to store a set of property values inside a catalog, much the same way as the other tool catalogs.

**8-July-2004 Added a new tool [PanelStop](#).**

**8-July-2004 The script itself, but also any entity can be assigned to a particular layer.**  
The routines to be used are the [global assignToLayer](#) and [Entity::assignToLayer](#).

**7-July-2004 Added dedicated [PrEntity](#) routines beamSet and elementSet.****7-July-2004 PropInt, PropDouble and PropString enhancements, (see [OPM functions](#)).**

Properties will work properly in Adt4 and Adt5, and associated autocad series.  
It is also possible for these versions to set a little help text, that will appear at the bottom of the OPM.

Also properties can be made read only.

For PropDouble it is now possible to set the format: length, angle, nounit,....

**5-July-2004 Added global project data, as [global](#) functions.**

ProjectName, projectNumber, projectStreet,... are now available.

**2-July-2004 Added a new tool [HalfCut](#).****30-June-2004 [Viewport](#) type has been extended**

It is now possible to change the viewports location in paperspace, as well as the transformation from model to paper space.

**23-June-2004 A Display can now be set to draw for shopDrawing of an entity.**

To do this the routine [Display::showInShopDraw](#) should be used.

**1-June-2004 Added the member function of [Element](#)::definition.**

This routine returns the definition description of an element.

**1-June-2004 Added a new tool [ScarfJoint](#).****17-May-2004 Graphical metalpart definition enhancement**

It is now possible to select a polyline in the drawing, and translate it into TSL code automatically by use of the menu command "Graphical metalpart definition", "define-Pline".

**13-May-2004 Added a predefined array [DimStyles](#).**

This array is an array of strings that contains all names of dimstyles in the drawing.

---

----- 28-April Release of version 2004.5 of hsbCad  
-----

**17-April-2004 Added [PlaneProfile](#)::area and [Element](#)::profNetto and [Element](#)::profBrutto.****16-April-2004 Added a new tool for beams called [Chamfer](#).****6-April-2004 The [GenBeam](#) has a new member function eToolsConnected.**

This routine allows to return a list of all the etools that operate on a GenBeam, Beam, Sheet, Sip.

**6-April-2004 The [ERoofPlane](#) type is added.**

This type is derived from Entity, and references the roof plane entity of Hsb.

**6-April-2004 The [NailLine](#) has two new member functions [filterGenBeamsBeingNailed](#) and [removeGenBeamsWithNoNailingBeamCode](#).****6-April-2004 The [Sheet](#) has a new member function [profShape](#).****4-April-2004 Added the [Display](#)::[draw](#) of a [PlaneProfile](#).****3-April-2004 Added the [Opening](#)::[openingType](#)**

The openingType can be door, window, opening.

**2-April-2004 The routine [extractContactFaceInPlane](#) was added to [Body](#).**

This routine allows to extract a profile of a body.

**30-March-2004 The [LineSeg](#) type is added.**

This type describes a 3d line segment.

**30-March-2004 The [PlaneProfile](#) type is added.**

This type describes a list of closed 3D planar polylines.

**25-March-2004 The [Sheet](#) has new member functions [plEnvelope](#) and [plOpenings](#).****25-March-2004 The [PLine](#) has new member functions [ptMid](#), [ptStart](#) and [ptEnd](#).**

Also the coordSys and the createCircle were added to PLine.

**24-March-2004 The new type [NailLine](#) is added.**

The NailLine is derived from Entity, so it is an entity in the autocad database/drawing. This is in contrast with the

ELEMNAIL, which is a tool that can be applied on an element. All nailLines of an element can be queried from the [element](#).

Also special routines are added to easily calculate new nailing lines.

**23-March-2004 A panel generation construction directive routine has been added to Element called [setCutLocation](#).**

This routine allows to generate a cut during the generation process of the panels.

**21-March-2004 Added the [addMeToGenBeams\(\)](#) and [addMeToGenBeamsIntersect\(\)](#) functions to [all tools](#).**

These routines allow to add the tool in question to each of the elements of the array of beams, sheets, sips or genbeams.

These routines are especially handy to add a tool to e.g. all beams of a wall element.

**21-March-2004 Added the [cuttingBody\(\)](#) function to [Drill](#) and [BeamCut](#), [Slot](#), [House](#),...**

This routine allows to get the Body will be used in performing the operation.

**21-March-2004 Added the [filterGenBeamsIntersect](#) for [Body](#)**

This routine allows to filter an array of Beam, Sheet, Sip or just GenBeam of which the body has intersection with a given Body.

**11-March-2004 Added the [strCutN](#) and [strCutP](#) for [Beam](#)**

The strCutN, strCutP, strCutNC and strCutNP are added.

**8-March-2004 Added the [Opening](#)::description and [OpeningSF](#)::descrSF**

The description ends up in the property of the Adt opening while the descrSF is part of the HSB data.

**7-March-2004 A new Element tool added [ElemConstructionBeam](#).**

This tool gives the directive to create a beam at time of element/panel construction generation.

**3-March-2004 The dbJoin of [Beam](#) is added**

By using it, two beams can be joined if their axis is parallel.

**3-March-2004 The [ElemSaw](#) tool has a setAngle property**

This property allows to specigy the angle to cut bevelled edges. The angle is defined by the right hand rule going along the curve in the sawing direction.

**28-Jan-2004 Extended the [FreeProfile](#) tool so it can be used on Sheet and Sip as well.**

The FreeProfile tool will now work on all GenBeam derived entities. Also the PLine with which it is constructued can now be closed.

**28-Jan-2004 Added [Body](#)::allVertices routine which collects all vertices of a body.**

**27-Jan-2004 The maximum length a Tsl script can have is extended from 32K characters to 512x32K characters.**

**23-Jan-2004 The [last property changed](#) is remembered, but also the [last grippoint moved](#).**

The predefined \_kNameLastChangedProp keeps the name of the last property changed, but also the last grippoint moved. The name of the grippoint is \_PtG0,\_PtG1,... or \_Pt0.

**20-Jan-2004 [Opening](#) setWidth has option with fixed point.**

When changing the width of an opening, one can specify which point to keep as fixed point.

**20-Jan-2004 Enhanced exploding an existing TSL with element tools.**

When the TSL instance has element tools, eg. an ElemNail tool, this tool will be exploded as an ElementNail entity, with its proper OPM properties.

**10-Jan-2004 It is possible to draw text always parallel to the screen.**

This can be done by specifying the nTextOrientation in the [Display draw of a text](#).

**28-Dec-2003 During insert the [showDialog](#) and the [showDialogOnce](#) global function are available.**

Calling it during insert, will pop up a dialog and query the user to specify the input variables of the properties specified before that call.

The property values are initialized with the default values in the script. The second time that the dialog is shown during that insert action, the values are however remembered.

**18-Dec-2003 For the [Dim](#) type [setReadDirection](#) and [setDeltaOnTop](#) are added**

These routines allow to manipulate the graphical appearance of the Dimension.

**4-Dec-2003 A new Element tool added [ElemItem](#).**

This tool allows to attach Item information to an Element.

**2-Dec-2003 A new type of reactor mechanism is implemented.**

To fire the execution of the Tsl instance whenever a specific entity in the drawing changes, one can use the routine [setDependencyOnEntity](#).

**1-Dec-2003 The entity array [Entity](#) is added that can store any entity reference.**

Also the routine [getEntity](#) allows for any entity in the drawing to be selected.

For the Entity the routines [gripPoints](#), [typeName](#) and [typeDxfName](#) are added.

**17-Nov-2003 A new beam tool is added, called [DiagonalNotch](#).**

The DiagonalNotch tool is meant for log connections.

**14-Nov-2003 The routine [setJapaneseMarking](#) is added as [general](#) member function of all beam tools.**

This routine allows to set the japanese marking flag for a particular tool to TRUE or FALSE.

**13-Nov-2003 For each beam tool the CNC machine can be specified.**

To do this, the following routines can be used: [allowMachineForCNC](#) and [excludeMachineForCNC](#).

**13-Nov-2003 A new special beam tool is added, called [CncExport](#).**

The CncExport tool enables to define arbitrary machine tools for the Hsb machine generation process. The tool is meant for advanced use only.

**12-Nov-2003 New [Map](#) type.**

The map type is a key-value constainer. The key is always of type String, but the value can be of almost any type.

**5-Nov-2003 Intersections of 2 solids can be calculated.**

The routine are member functions of Body, and are called [intersectWith](#), and [hasIntersection](#).

**5-Nov-2003 [GenBeam](#) has a member functions to query its solid.**

The routines are called [realBody](#) and [envelopeBody](#).

**5-Nov-2003 [ElemZone](#) routines added.**

A number of routines are added to query the information of an ElemZone of a particular Element.

**4-Nov-2003 dbCreate of [Sheet](#) added.**

This routine allows to create a sheet, much the same way as to create a beam.

**4-Nov-2003 A new type [ElementWall](#) exists.**

This type is derived from Element, and ElementLog is derived from it. The function [getConnectedElements](#) returns

a list of other wall elements that are connected with it. Two elements are connected if their floorplan outline touches.

**10-Oct-2003 Calculate the length and height of a string that will be displayed with a certain dimension style.**

The functions that calculate this are member functions of [Display](#), and are called [textLengthForStyle](#) and [textHeightForStyle](#).

**10-Oct-2003 To format a distance into the proper string representation, architectural, inches,... one can use the [formatUnit](#) routine.**

The formatUnit routine is a member function of [String](#), and uses a dimstyle with a lunit and precision setting.

**10-Oct-2003 During dimensioning the text can also be placed perpendicular to the [dim](#) line.**  
 To do this a different Dim constructor needs to be used, one that specifies for the middle and end dimension what the direction is.

**8-Oct-2003 Any entity can be assigned to a group, or element group.**  
 The routines to be used are [Entity::assignToGroup](#) and [Entity::assignToElementGroup](#).

**7-Oct-2003 During insert the [showDialog](#) global function is available.**  
 Calling it during insert, will pop up a dialog and query the user to specify the input variables of the properties specified before that call.

**3-Oct-2003 New command [Hsb\\_ListTslWords](#).**  
 This command allows to generate a text file that can be used for syntax coloring in text editors.  
 Currently only WinEdit is supported.

**28-Sept-2003 The [FreeProfile](#) type has a new property [setCncMode](#).**

**25-Sept-2003 Order operators are added for the [String](#) type.**  
 It is now possible to order 2 strings e.g.: if (str1<str2)...

**25-Sept-2003 The [swap](#) function is implemented on all array types.**

**25-Sept-2003 The module parameter of a beam, and [genbeam](#) can be queried and set.**

**25-Sept-2003 [Construction directives](#) can be added to the element.**

**23-Sept-2003 A contact entity can be added by the use of [Beam::stretchDynamicTo](#) function.**

**21-Sept-2003 [TslInst](#) properties and grippoints added.**

**21-Sept-2003 New stretch value possible during [addTool](#) of an endtool.**  
 The nStretch argument of addTool can have 3 different values: [\\_kStretchNot](#), [\\_kStretchOnInsert](#) or [\\_kStretchOnToolChange](#).

**18-Sept-2003 New beam tool added, called [DimTool](#).**

The DimTool allows to specify information that will appear in the shopdrawing of the beam.

**17-Sept-2003 New PLine routine added [area](#).**

**17-Sept-2003 New Entity routine added [handle](#).**

**17-Sept-2003 New ElementLog routines added [setContourPtsLeft and Right](#).**  
 Also a routine to calculate the height of a certain log course was added: dHeightFromCourseNr.

**15-Sept-2003 New input routines [getString](#), [getInt](#) and [getDouble](#) added.**

**15-Sept-2003 PLine close and [setNormal](#) functions added (see [PLine](#)).**

**12-Sept-2003 Changing debug text height will not change the current dimstyle.**  
 Before, the points and vectors that were displayed in debug mode, used the current dimstyle. Also, changing the text height through the "Change text height" in the TSL Editor window, would cause a style overwrite. This behaviour has been changed now.

Point and Vectors can be visualized in debug, using the textstyle from the dimstyle that is specified in the Hsb\_settings->dimension->TSL. The height of the text, that also determines the scaling of the vectors, can be adjusted by the "Change text height" in the TSL Editor window. Changing the text height will not result in a style overwrite.

If the value is set to 0 of the "Change text height" in the TSL Editor window, the textheight from the dimstyle is used.

**20-Aug-2003 TSL editor: drop down list for selecting viewing direction has been added.**

**7-Aug-2003** The new type [Opening](#) and [OpeningSF](#), which represents an opening in a wall, eg a window, a door,...

**5-Aug-2003 A new report method is added:** [reportNotice](#).

The reportNotice will print a message in a modeless dialog, in contrast with reportMessage that will print in the autocad console. Also compiler messages are diverted to this communication channel.

**3-Aug-2003 A number of member functions are added to set material, grade, label,... of beam, panel, sheet types, see [GenBeam](#).**

**31-July-2003** The number of [execution loops](#) can be set explicitly.

**30-July-2003 A new type added, called [Viewport](#), which represents a viewport in paper space.**

In the "[Display in layout](#)" section, practice is discussed to draw in a layout, using model space viewport information. Also a nice example is given there.

**28-July-2003** New beam tool added: [CncMessage](#).

**13-July-2003** New beam tool added: [SpecialMill](#).

**7-July-2003** New beam tool added: [ChamferedLap](#).

**2-July-2003** The width and height of a beam can be changed by calling `setD` on the beam (see [Beam](#)).

**30-June-2003** Prop values can also be set from within the script (see [OPM functions](#)). To change one property based on the value of another, can be done by using the predefined [kNameLastChangedProp](#).

**24-June-2003** The new type [Body](#), which represents a 3d solid is added.

**17-June-2003** [CrossProduct revealed](#) and [dotProduct revealed](#).

**14-June-2003** To select an element in the drawing, one can use the [getElement](#) or the [getElementFromEntity](#).

**6-June-2003** Allow to addVertex to a polyline with a radius, see [PLine](#) and example. Also some member functions are added to facilitate polyline manipulation: [reverse](#), [convertToLineApprox](#), [projectPointsToPlane](#).

**5-June-2003** New beam tool added: [SolidSubtract](#).

**3-June-2003** With the member function [dbCreate](#) a new beam can be created by the script.

**31-May-2003** One more function to filter a set of beams [filterBeamsTConnection](#) has been added.

**28-May-2003** Added the collectDimPoints for beams, sheets, as function on a [DimLine](#). Also added the [dwgName](#) and [dwgFullName](#) functions.

**22-May-2003** Display can also exclude a view direction with [addHideDirection](#).

**20-May-2003** Special [element](#) construction and deletion triggers are added.

**12-May-2003** The two functions [filterBeamsContactCut](#) and [filterBeamsContactCutHead](#) allow easy filtering of beams.

Also two other related routines are added to the beam: [findPlaneContactCut](#) and [findPlaneContactCutHead](#).

**12-May-2003** [TslInst](#) is the type that represents an Tsl instance in the drawing.

**8-May-2003** Query the element from an entity.

The element that a beam or sheeting belongs to, can be found executing the element() function (see [Entity](#)).

**28-Apr-2003** String manipulation routines are added.

See the [string class](#), and the [format specification](#) for a description.

**28-Apr-2003** Element code and number accessible.

For an [Element](#), the code(), also called type and number() are now available.

**7-Apr-2003** [DoubleCut](#) extended for cuts parallel to the beam axis.

**7-Apr-2003** Dimensioning with [Dim](#) can also generate chain dimensions.

**27-Mar-2003** Values can be specified in "mm" and "inch" at the same time, using the [Unit](#).

**26-Mar-2003** [MetalParts](#) can be copied, added and subtracted, as well as transformed.

**26-Mar-2003** The list of TSL's will be merged when inserting a block in the drawing.

On block insert, the list of TSL's will be merged. Different possibilities can exist.

- \* The TSL appears only in the current DB (drawing) -> nothing done.
- \* The TSL appears only in the block DB (drawing) -> TSL just inserted.
- \* The TSL appears both in current and block DB and description, script, preview,... are all the same  
-> TSL is not inserted, all links to the TSL in the block DB are changed to links in the current DB.
- \* The TSL appears both in current and block DB and description or script or preview or... is different -  
> TSL in block DB is renamed (eg from aa to aa\_0) and inserted.

**19-Mar-2003** All entities can be [copied, erased and transformed](#). Also beams can be [spit](#), and [copied](#) without tools.

On Beam, Sheet, Sip, but also other entities can be copied by the function call dbCopy.

Entities can be erased from the drawing database using the dbErase.

Besides that, entities can be moved, rotated,... by using the transformBy.

**14-Mar-2003** The [CoordSys](#) type has been extended to represent a transformation matrix.

The transformation matrix can be set by setToRotation, setToTranslation, setToAlignCoordSys,...

Basic transformation functions have been added to the [Point3d](#), [Vector3d](#), [Line](#), [Plane](#), [PLine](#), [CoordSys](#), [DimLine](#), [Dim](#), [MetalPart](#)... but also to all tools.

**14-Mar-2003** The radius of a [Drill](#) can be changed by [setRadius](#).

**14-Mar-2003** The different components of a [vector or a point](#) can be accessed.

To use the x component of a point the X() member function is available: pt.X()

To change a particular component, one can use the setX(): pt.setX(10);

**14-Mar-2003** Dimensioning is possible through the use of [DimLine and Dim](#) types.

**12-Mar-2003** The group that the TSL instance belongs to can be assigned by using [assingToGroups](#) and [assignToElementGroup](#).

**28-Feb-2003** The [editor window](#) explained.

**28-Feb-2003** The [display](#) can be done by the TSL.

**20-Feb-2003** Some [PLine](#) member functions are added.

From a PLine the list of vertices can be queried through the call vertexPoints.

Also a check can be done to see if a point lies on a PLine: isOn().

**14-Feb-2003** Element type refers to wall element, with a number of zones.

A new type: [Element](#) is added. It refers to the collection of beams and sheetings that construct the wall element.

Each beam or sheeting lies in one of the element zones. The type ElemZone describes the zone properties.

**11-Feb-2003** New tool added: [FreeProfile](#)

A new tool is added. It is called [FreeProfile](#).

**23-Jan-2003** The definition of [\\_PtG0](#), [\\_PtG1](#)... has changed into [\\_PtG\[0\]](#), [\\_PtG\[1\]](#)...

To allow easy appending, looping, counting over the grippoints stored in an instance, the list of grippoints has been redefined as an array.

Therefor the previous definition of the grippoints: [\\_PtG0](#),... is no longer valid. From now on, the grippoints are accessed through the array [\\_PtG\[\]](#).

Older scripts are searched for the appearance of [\\_PtGxx](#), and where it is found, it is replaced by [\\_PtG\[xx\]](#).

This automatic correction is done at import (importing mcr files), as well as at dwgin (loading existing scripts inside a drawing).

No user actions are required for this conversion.

**23-Jan-2003** New tools: [Rabbet](#) and [Daddo](#)

Some new tools are added. They are called [Rabbet](#) and [Daddo](#).

**20-Dec-2002** Autmatic [block](#) insert in the drawing

When a Block-drawing is used by the script, the script will first look in the table of loaded blocks to find a matching block name.

If no such block was found, the file system is checked if there exists a block with the specified name. The file system is checked at the last used block load-path. If the script execution finds that block drawing, it is inserted into the drawing automatically.

It is also possible to specify a specific block file location path in the name of the block.

#### **15-Dec-2002 Initial value of OPM Enum property can be set**

A new constructor of [OPM property](#) values was added.

#### **15-Dec-2002 New Arc tool**

A new tool is added. It is called [Arc](#).

#### **15-Dec-2002 [closestPointTo](#) member function for Line added**

The Line type has a new member function to calculate the closest point to another Line.

#### **13-Dec-2002 New ParHouse tool**

A new tool is added. It is called [ParHouse](#).

#### **12-Dec-2002 Beam position control**

Through the use of the [setPtCtrl](#) function, points can be set through which the axis of the beam will go.

#### **10-Dec-2002 Infinite loop prevention**

During script execution a counter is incremented each time a loop is visited. When the loop counter reaches 1000000, the program stops. This means that an infinite loop, eg.: while(1); which would otherwise run forever, is now broken after a little time.

#### **10-Dec-2002 Line-Beam intersection**

A new type is added: [LineBeamIntersect](#). This type can be constructed with a line, or a polyline, and a beam. The type LineBeamIntersect has member functions: pt1, pt2, vecNrm1, vecNrm2, bHasContact, nNumPoints.

With these functions, intersection dependent calculations can be done.

#### **1-Dec-2002 Select preview entity**

In the script edit window, a new button is added called "Select preview entity". Selecting a preview entity from the drawing

will result in a copy of the selected entity, together with its beams it is putting tools on. Also the properties of that particular instance will be copied. This results in an exact preview situation as the one in the drawing. This preview entity can then be used in stepping through mode.

When the script custom menu command "Edit script" is used, the selected entity is used as preview entity.

#### **3-Nov-2002 TSL editor enhancements**

New auto-complete feature while typing, and search and replace function added to the TSL editor.

#### **7-Oct-2002 New beam member functions**

Two new member functions are added to the beam: [blsMyEnd](#) and [blsStraightCut](#)

#### **7-Oct-2002 DoubleCut tool added**

A new tool type is defined, called [DoubleCut](#)

#### **3-Oct-2002 angleTo, rotateBy and acos, atan functions added**

There are 3 new global functions added [acos](#), [asin](#) and [atan](#). Also the two member functions

of Vector3d are added: [angleTo](#) and [rotateBy](#)

#### **2-Oct-2002 Direction test function**

A number of routines are added to facilitate the [parallel](#), [and perpendicular](#) testing of vectors.

#### **2-Oct-2002 Housed dove tail tool added**

A new tool called [HousedDove](#) is added. Also the manual has been updated for the [House](#) tool.

#### **2-Oct-2002 Round types explained in this manual**

See [round types](#) topic.

#### **30-Sep-2002 All new release notes that are TSL specific, will only appear here.**

#### **30-Sep-2002 [closestPointTo](#) member function for Line added**

The Line type has a new member function to calculate the closest point to a given point.

#### **30-Sep-2002 [EraseInstance](#)**

A new global function is added that erases the tool script entity from the Autocad drawing database.

#### **30-Sep-2002 Script entity context menu: added Edit script**

See [context menu](#) topic.

#### **30-Sep-2002 The beam information / member functions have been extended**

See [beam](#) topic. Information: grade, volume, solidlength, label, sublabel,... is now available.

#### **26-Sep-2002 Help button in editor window**

When the help button in the TSL editor window is pressed, the tool script help file is opened, and the topic is selected in the find function that corresponds with the selected text in the editor.

#### **25-Sep-2002 [hsb\\_scriptinsert](#) lisp command**

A lisp command has been added to insert an instance of a TSL (metalpart and or tool). This lisp command can be used to customize a toolbar. When executing the autocad command (hsb\_scriptinsert "aa" 0)

an instance of the TSL named "aa" is added with the option 0. The option 0 means multiple instances can be inserted. If the option is 1, only one instance will be inserted at a time. If the script named "aa" does not exist at that time, nothing is done.

#### **23-Sep-2002 [break, continue and return](#)**

The TSL (Tool Script Language) has been extended with three instructions: "break", "continue" and "return".

The "break" and "continue" are valid inside a loop instruction, while the "return", can be used anywhere.

Please see the TSL help file for details.

#### **23-Sep-2002 Debugging improved**

The stepping through the TSL-code in the TSL editor window in debug mode has been improved.

#### **17-June-2002 TSL: MetalPart polyline extrusion**

The TSL language has been extended with the possibility to have a polyline extrusion where the polyline is located in the middle of the extrusion body. This is usefull for generating metalparts with a variable thickness, and a centered location, eg. centered on the beam axis.

```
> PLine plPoly;
> Vector3d vecExtrusion;
> double dFlag;
> MetalPart mp( plPoly, vecExtrusion , dFlag );
> MetalPart mp2( plPoly, vecExtrusion); // default value of dFlag is 1
```

If the (dFlag==1) then the polyline is at one side of the extrusion body.  
 If the (dFlag==0) then the polyline is in the midplane of the extrusion body.  
 Other values of dFlag mean a linear combination of the above offset.

When adding MetalPart polyline extrusion by the "Graphical metalpart definition" there is a toggle box provided for the above option.

#### **27-May-2002 Name of TSL**

The TSL scripts are saved in the Autocad database in an "Autocad named dictionary" with key name equal to the script name.

Therefore the scriptnames cannot have special characters: \ | " ; = \* ` < > :

The following characters are allowed: <space> \_ [ ] ' + - { } % \$ # ( ) ! @ ~ ^

When attempting the use of an invalid name, the user will be requested to insert a valid name.

#### **14-May-2002 Automatic block insert by TSL**

When a TSL scripts needs a block drawing of a block that is not yet loaded in the drawing, a file dialog is automatically shown to load the block drawing inside the current one.

#### **29-April-2002 Reset OPM for TSL**

When different TSL scripts are used in the drawing, the OPM might display the wrong number, or the wrong names of some properties. To assure the OPM displays the correct information, a new command was added: HSB\_RESETOPM. The command can also be fired from the context menu of

TSL entities. The correct properties will be shown of the first TSL in the selection set.

The OPM (Object property manager) caches the number and names of the properties in the drawing. They are grouped on the basis of the entity type. This sometimes leads to wrong number, or the wrong names of some properties.

#### **6-1-2002 Metalpart: OPM property notes added.**

As hardware information, the metalpart itself contains properties as

- notes (instance specific, set in OPM)
- material (set by script)
- model (set by script)
- description (property of the script holder)

#### **6-1-2002 Metalpart: visualization of toolings in timber.**

While writing/editing a metalpart script, one can test the contents of the script by either pressing preview, or pressing init/compile + step through buttons. During this stepping through phase, the last complex tool added to a beam is visualized in the preview window. Complex tools are Mortise, Slot,...

#### **1-1-2002 Metalpart: Bitmap of preview window.**

The bitmap of the preview window can be saved with the metalpart file. For that, the bitmap is converted to an ASCII representation.

#### **1-1-2002 Tool script language (TSL) extension: Enumerated properties.**

All property parameters can be enumerated: have a value out of a set of predefined values. For making a property an enumerated one, the initial value of the PropXXX needs to be an array.

Syntax:

```
PropInt plnt(<nIndex>, <arInt> [, <strNameOPM>] );
PropDouble pDouble(<nIndex>, <arDouble> [, <strNameOPM>] );
PropString pString(<nIndex>, <arString> [, <strNameOPM>] );
```

### **30-12-2001 Tool script language (TSL) extension: reportError, reportWarning, reportMessage**

Three new global commands are added:

Syntax:

```
reportMessage(<strMessage>);
reportWarning(<strMessage>);
reportError(<strMessage>);
```

The reportMessage will write to standard output of autocad (the F2 window).

The reportWarning will pop up a comment box.

The reportError will also pop up a comment box, but will also stop the execution of the script. By this you can prevent the further execution of the script.

Tip1: You can concatenate strings by + operator.

Tip2: You can convert int's and double's to strings automatically.

eg. int n = 4; String str = String("Text no.:" ) + n;

### **23-12-2001 Tool script language (TSL) extension: material, model and [dxaout](#).**

The tool script language (TSL) is extended with the following

3 global functions: "material", "model" and "dxaout".

Both MATERIAL and MODEL keys will appear only once in the dxa-structure.

The last strValue set will be used. There is no uniqueness condition on the user keys U\_STRKEY.

### **13-12-2001 Tool script language (TSL) extension: tool Mark**

A new tool is added to the language: Mark.

There exist a number of possible constructors.

A typical constructor is

```
Mark mrk( pntLocation1, pntLocation2, vecNormalFace, strText);
```

The Mark-tool can only be added to a beam. When added,

it will mark the beam with 1 or 2 lines, and with some  
text on the face of the beam with outer normal given.

The mark will be placed over the complete height/width  
of the beam.

### **11-Dec-2001 Tool script language (TSL) extension: Mortise as tenon**

There is a "male"-Mortise = Tenon tool type available.

It is defined by

```
Mortise( ptOrg, vecX, vecY, vecZ, [dLenX, dLenY, dLenZ,
[nFlagX, nFlagY, nFlagZ]], nEndSide, [nRounded]);
with nEndSide == 2.
```

### **11-Dec-2001 Tool script language (TSL) extension: unlimited properties**

The number of PropDouble, PropInt and PropString values that  
can be added by the script is now unlimited. The amount that  
will be displayed in the OPM is however limited to: 8 PropInt,  
16 PropDouble, 8 PropString.

Remember that the first available index is 0.

eg.

```
PropDouble dDiam( 0, U(30,"mm"), "Diameter"); // index 0
PropDouble dDiam2( 15, U(30,"mm"), "Diameter"); // largest index that is displayed
PropInt nNum( 8, "Number of ..."); // index 8, allowed but not displayed in OPM
```

### **10-Dec-2001 Metalpart insertion**

The same code/sequence is now used when inserting a metalpart:

- through the menu command;
- making a new one in the graphical metalpart definition;
- selecting preview beams in the the metalpart definiton dialog.

First the primary beam(s) need to be selected, then the other beam(s) need to be selected, then the required insert points need to be selected.

### **6-Dec-2001 Tool script language (TSL) extensions**

A number of things have been added to the tool script language:

- The following types are available:
  - \* T: a T connection;
  - \* E: location on the axis of primary \_Beam0;
  - \* G: Mitre (gehrung) type of connection;
  - \* O: arbitrary placement in space, no beams required (new);
  - \* X: cross connection (new);
- The L type does not exist anymore. It is just an E-type with an extra required beam.
- All types can have a number of required beams. These are selected at insertion time of a new instance.
- A script can now have a number of insert/grip points. They are given at insertion time of a new instance.
- The internally stored UCS coordinate system in the metalpart is transformed during rotation. This means if the metalpart is defined relative to the UCS (\_PtU, \_XU, \_YU, \_ZU), they can be rotated, mirrored and moved.

### **9-10-2001 Tool script language (TSL) extension: Hardware**

The toolscript is extended with a type "Hardware". The "Hardware" type is an extention compared with the "Bolt" type. It has an extra parameter: type. The general contructor is the following:

```
Hardware name(<string type>, <string description>,<string model>,
             <double length>,<double diam>,<int number>,<string material>);
```

compared to

```
Bolt name(<string description >,<string model>,
           <double length>,<double diam>,<int number>,<string material>);
```

The following two lines are completely equivalent:

```
Bolt b1("Art324","M20",40,20,4,"steel");
Hardware hw("Bolt", "Art324", "M20", 40, 20, 4, "steel");
```

The same types can be used, including the user defined type, as in the "Hardware information dialog". The dialog can be used to insert, at the cursor location, a new "Hardware" instruction.

### **2-10-2001 New command 'Graphical Metal Part Definition' in Menu Tooling**

Metal parts can now be recorded graphically. Typically polylines and circles that represent the outline of the MetalPart will be drawn. (The metal part will be extruded along the polyline normal). The geometrical data

can be recorded relative to the beam dimensions and to a user defined points. The following tools can also be recorded graphically: Slot, BeamCut, Drill, EndMortise, Mortise. When using a polyline to record a tool, the polyline normal defines the tool path.

#### **25-09-2001 Metalpart/tool description insertion**

While inserting a new metalpart description into the drawing, a number of beams can be selected. These beams can be interpreted to make multiple metalparts, or to be added to the same metalpart. In order to choose between these two options, the selection dialog now contains an extra set of radio buttons.

#### **15-09-2001 Tool script language (TSL) extension: new tool "House"**

The tool description language (Macro language) contains a new tool: "House".

The tool has a similar set of parameters as the Mortise tool, except that it has not a parameter endside. (The mortise tool can contain an sawcut.)

The declaration syntax is:

```
House name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>,<X-length>,<Y-width>,<Z-width>);  
House name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>,<X-length>,<Y-width>,<Z-width>,  
<X-flag>,<Y-flag>,<Z-flag> );  
// Default rounded == 1  
House name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>,<X-length>,<Y-width>,<Z-width>,  
<X-flag>,<Y-flag>,<Z-flag>,<rounded> );
```

#### **15-09-2001 Tool script language (TSL) extension: mortise tool has rounded flag**

The tool description language (Macro language) syntax for the Mortise tool is extended with one parameter: rounded. The default value (that is also compatible with the previous version) is rounded=1. This means that the mortise will be manufactured rounded.

#### **23-08-2001 Shopdrawings of metalparts.**

Currently the view-instruction in the metalpart description has a parameter for automatic dimensioning the metalpart. For the shopdrawing the metalpiece is projected onto a plane, resulting in a collection of lines.

This collection of lines is now cleaned, in the sence that colinear edges are joined into one edge whenever possible. This reduces the amount of points, and therefore rationalizes the diminsioning.

#### **24-07-2001 Tool script language (TSL) extension: Bolt command**

In the tool macro language there is a new instruction added: Bolt.

To declare a bolt, the following instruction is used:

```
Bolt name(<String name>,<String type>,<double length>,<double diam>,  
<int number>,<String material>);  
// eg. Bolt b1("Art324","M20",40,20,4,"steel");
```

All properties of the bolt will appear in the DXA file, when using Access / Excel commands.

A bolt that is declared can be visualized (currently only very simple) in the metal part drawing with the following command:

```
<Bolt>.visualize(<Point3d location>,<Vector3d direction>,color);
```

### 24-07-2001 Metalpart numbering

Metalparts now can be numbered together with the beams. They are also written in the dxa-output for MS-Access and MS-Excel. An ECS, entity coordinate system was added for comparing different metalpieces.

Currently, metalpieces and beams will be given separate numbers.

In order to have separate sets of numbers of beams and metalpieces one should execute the |Numbering| command more than once.

Each time, the appropriate entities are toggled for selection, and a different starting number is given. e.g. Number beams from 1, and metalpieces from 1000.

Every metalpart has an ECS (entity coordinate system). The UCS can be set to an object's ECS by using the command UCS->New->OBject.

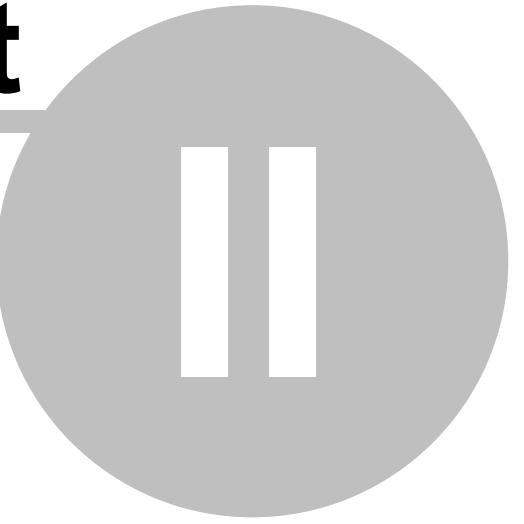
The ECS of the metal part is used to perform the compare action. Entities are compared in order to give them an equal number. Every type of metalpiece has its own ECS predefined: \_PtE, \_XE, \_YE, \_ZE.

The macro-writer can alter the ECS of the metal part when it is required:

e.g. \_XE = \_XW; \_YE = \_YW; \_ZE = \_ZW;

if the metalpiece would be aligned with the world coordinate system.

# **Part**



III

## 2 Introduction

### 2.1 Introduction

'Where shall I begin, please your Majesty?' he asked.  
 'Begin at the beginning,' the King said gravely, 'and go on till you come to the end: then stop.'  
 From Alice in wonderland.

The "Tool Script Language" (TSL), sometimes called "tools and metal part description language" is an advanced feature of hsbCad. It allows the creation of a customized drawing entity through the writing and execution of a simple script.

As you might know, the AutoCad drawing is an object oriented database that hosts entities. Each entity has a graphical representation, a set of properties, a set of grip and snap points,... hsbCAD provides a lot of such entities including an hsbBeam, an hsbPanel, but also all sorts of tooling's such as hsbTenon, hsbNailingLine,... Tooling's have the ability to change/manipulate other entities, eg an hsbTenon will shape the solid representation and cnc-definition of an hsbBeam.

TSL brings the power to describe these tooling's to the customer of hsbCad by means of writing a simple script.

Because a TSL entity can query the drawing database and the hsb building information model as well, it allows on top of that for the ultimate customization of automated wall elevation drawings.

The following is an non-exclusive list of functionality that can be expressed by the TSL script:

- Define an 3d solid shape that represents a metal part. The shape can be of arbitrary complexity, composed through logical solid operations on basic shapes (unite, intersect, subtract), and additional solid manipulation operations (drill, cut,...).
- Define how metal parts are compared for numbering, and define the contents of its representation in the bill of material (BOM).
- Define the set of tools (drill, cut, marking, mill, slot,...) that are done on each of the timbers that the TSL is involved with.
- Automatically update the shape, tools,... when timber properties are changed.
- Define the parameters by which the TSL entity is to be influenced.
- Define the set of tools that are grouped logically together and act on a wall/floor/roof element. E.g. the millings, nailings, solid representation, BOM,... of an electrical outlet in a stick frame wall.
- Give directives to the hsbWall generation module on how to define the construction around an opening.
- Define the BOM as graphical entity in the wall/floor/roof element elevation drawing.
- Define the way the wall/floor/roof element elevation drawing is dimensioned.
- ...

This TSL functionality is being developed to provide absolute freedom on the definition and application to timbers of customized toolings and metal parts.

The language aims to maintain and expand the principle of dynamic timber-tool relations used on all HSB tools and joint types. The dynamic property of tools and metal parts is based on the

predefinition of relations between entities. It allows automatic reaction of the tools or metal parts to the most common routines applied to the timbers that contain them and vice versa.

Tools and Metal parts are also created as autonomic drawing entities that can be independently selected, modified, erased or applied. Such an entity in the Autocad drawing is called a TSL instance, or TSL entity. The TSL entity has OPM available properties, which are defined inside the script. Also the TSL entity can have references to any number of beams, and can have any number of grip points. How the TSL entity looks like, or what actual tooling the TSL entity does on the beams is defined in a script. The result of the execution of the script, is the TSL instance.

The feature that makes the Tsl concept unique to its kind is that it allows the definition of custom entities. This power is brought to the Tsl programmer. Except for the C++ API of Autodesk, there is no other API that allows to do this. The level of customization that can be implemented using custom entities is not even comparable with the customization possibilities exposed through API's like VBA, or even .NET. The main power in this type of customization lies in the fact that the entities that live in the Autocad database are tailored to the needs of the customer. We are talking about

- Complete freedom in parameterized graphical representation including solid shape, grips and snap points,...
- Complete freedom in defining properties appearing in the Autocad property manager
- Custom reaction mechanism, reacting on other entities inside the Autocad database
- Manipulate the hsbCad entities directly (timbers, wall elements,...)
- Custom output of these entities in the BOM,...

Example:

i. When a customer specific fixture for eg a sink needs to be defined, a procedural API would allow to insert some existing lines or predefined entities or even blocks, but the Tsl API allows to define the custom entity ..... together with all the tooling on the timbers, cutting on the wall element, appearance in BOM, correct custom properties. Again, this level of customization is not possible with other API's.

The syntax of the TSL language is similar to that of the C language. An extensive help document containing tons of examples is provided to assist in the writing of TSL scripts.

As any other modern computer language, the script uses variables. Each variable has a name, and a specific type. The variable can hold any value of that type. During the execution of the script, expressions are evaluated. Each time the variable is found inside an expression, the variable is substituted by its value. Values can also be assigned to a variable through the use of an assignment. When a variable is defined, it can also be given a value. This is called initialization or construction.

*[Example of variable definition :*

```

int j = -10;           // define variable with name j, type integer and initial value -10
int i = j;             // define variable with name i, type integer and initial value the
value of variable j
j = i - 10;            // assign to the variable with name j, the value of i, with 10
subtracted from it
Point3d pt1( 10, 2, 5); // define variable with name pt1, of type Point3d, and initial
coordinates (10,2,5)
PropDouble dGap(0, 10, "Gap size"); // define a variable with name dGap, type
PropDouble. A variable of this type can be initialized by the use of 3 parameters: an
index, a value, and a description string. In this case, the index is 0, the value is 10, and
the string is "Gap size"
—end example]
```

The script itself has also a type. The different types are T,G,C,O and E. The type will determine the behaviour, the location, and the set of predefined variables. For instance a T type is typically used for a connection of 2 beams, where one beam, called the male beam, touches the female beam at the side. Very much like the 2 lines of the letter T itself. The vertical line corresponds with the male beam, while the horizontal line corresponds with the female beam. Ofcourse the beams do not have to be perpendicular. The behaviour of this type is the same as an Hsb tenon connection:

- the tool will be erased automatically if one of the two defining beams is erased;
- the tool will be copied automatically if both beams are in the selection set;
- the tool will be triggered for recalculation if the location or properties of one of the beams changes;
- the location of the T-type is determined by the location where the male beam axis hits the surface of the female beam.

For a T-type a large set of predefined variables exist. For instance the location where the male beam hits the surface of the female beam is a variable of type Point3d and name \_Pt0. The value is set to the correct location before the script is executed. Therefor it is called predefined. The axis of the male beam is a Vector3d with name \_X0. The normal to the female face is \_Z1.

Below you find an example of a T connection between 2 beams. The TSL, called simpleCut cuts the male beam at a certain distance from the surface of the female beam. In the image some OPM properties of the script are shown, including the one that is defined inside the script.

*[Example of a T-type, with script name "simpleCut":*

```

Unit(1,"mm"); // whenever U() is used in this script, "mm" values are meant.

// define an OPM property, index0, initial value 10mm, called "Gap size"
PropDouble dGap(0,U(10),"Gap size"); // dGap can be used as floating point variable

// display the location and direction of points and vectors for debugging purposes only
_Pt0.vis();
_X0.vis(_Pt0); // visualize the vector, starting in point _Pt0
_Z1.vis(_Pt0);

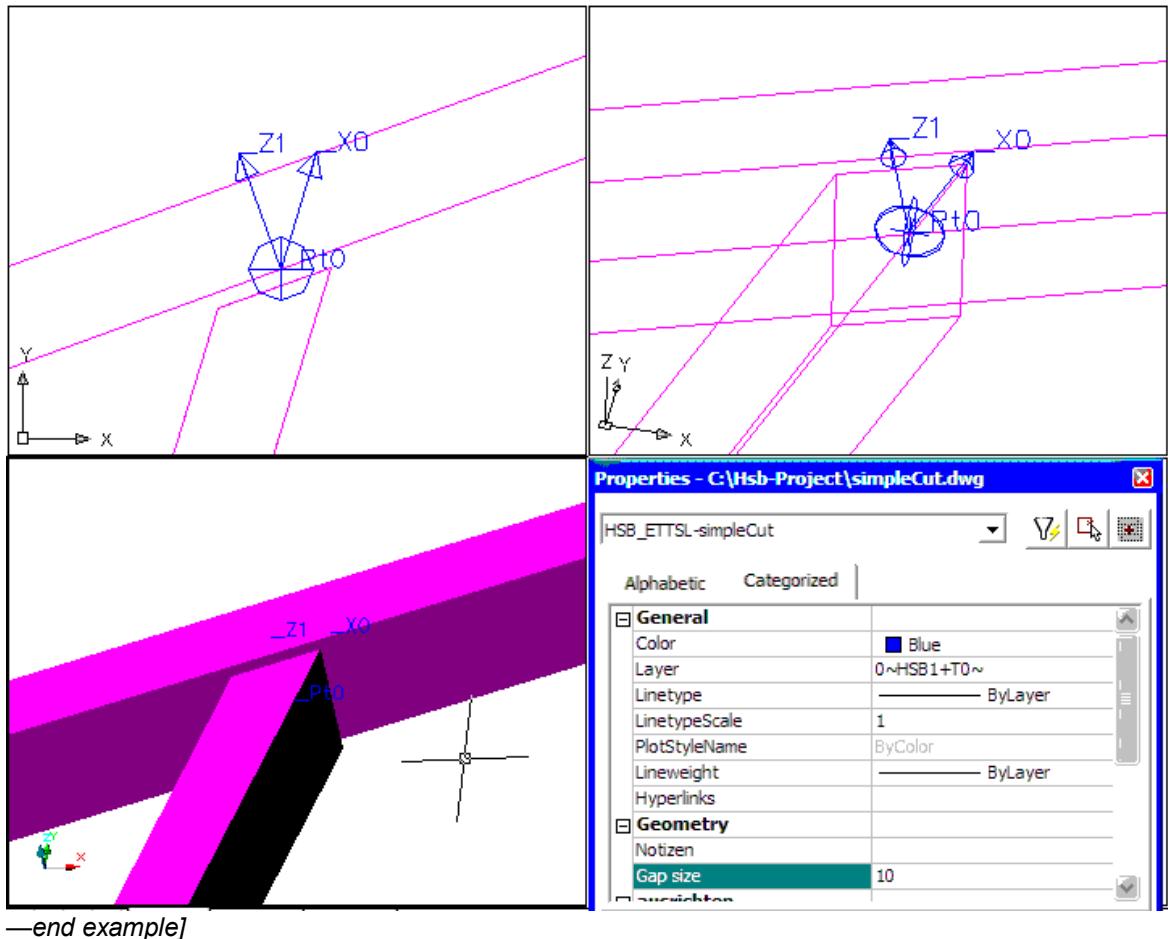
Line ln(_Pt0, _X0); // define a line with name ln, through point _Pt0, and along vector
_X0
Plane pl(_Pt0, _Z1); // define a plane with name pl through _Pt0, with normal _Z1

// find the intersection point of the line and the plane, but at a distance of dGap from the
plane
// in the oposite direction of the plane normal
Point3d ptCut = ln.intersect(pl, -dGap);

// define a Cut tool with a point and a normal, the variable name is ct
Cut ct(ptCut, _Z1);

// Add the tool to the beam. When added for the first time, the tool will remove all other
// endtools in that direction on the beam. 1 means that the endtool is active during
insert.
_Beam0.addTool(ct,1);

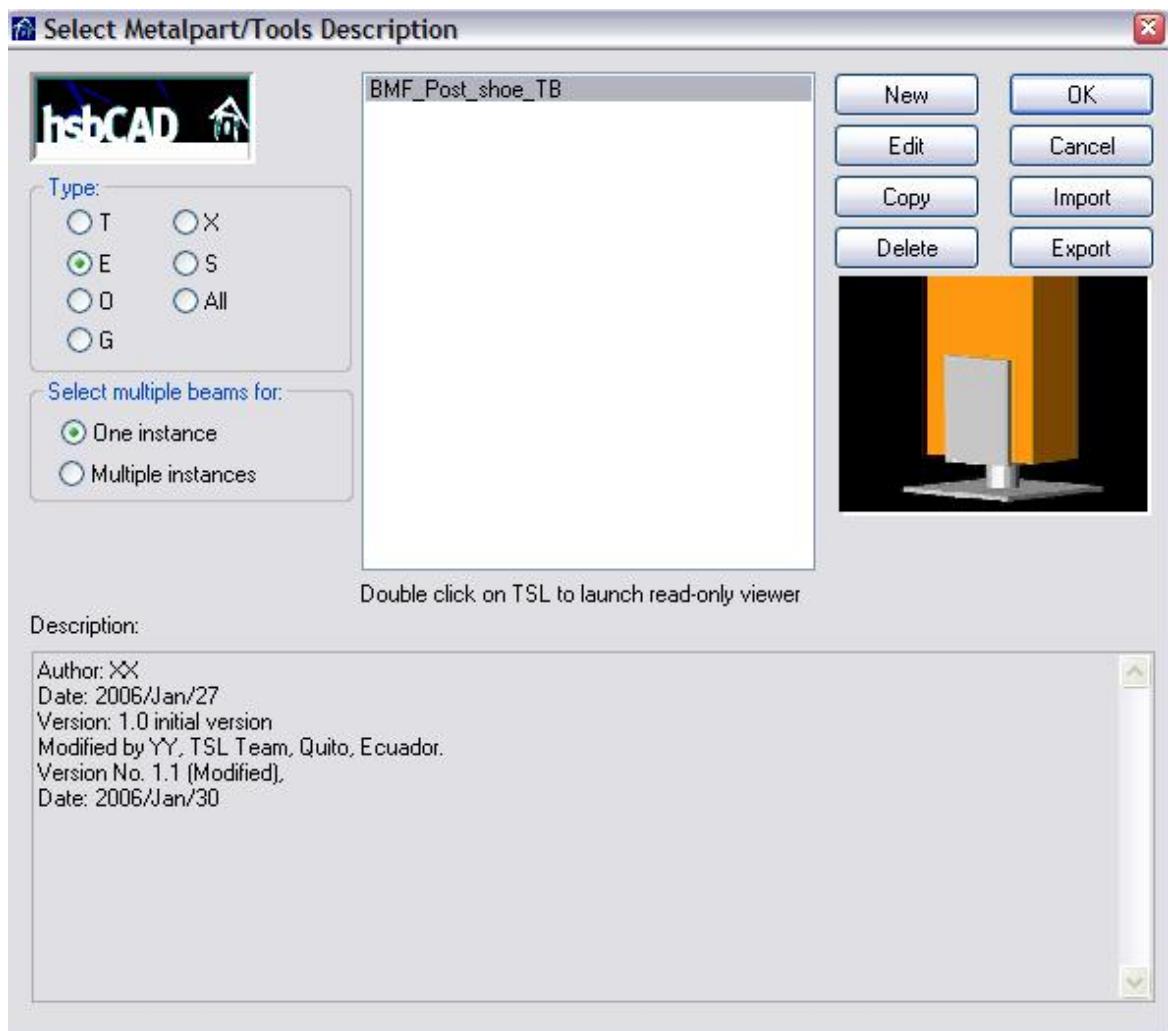
```



## 2.2 Procedure

When the menu command is fired to insert a TSL instance, the select dialog is shown. The dialog is called "Select Metalpart/Tools description".

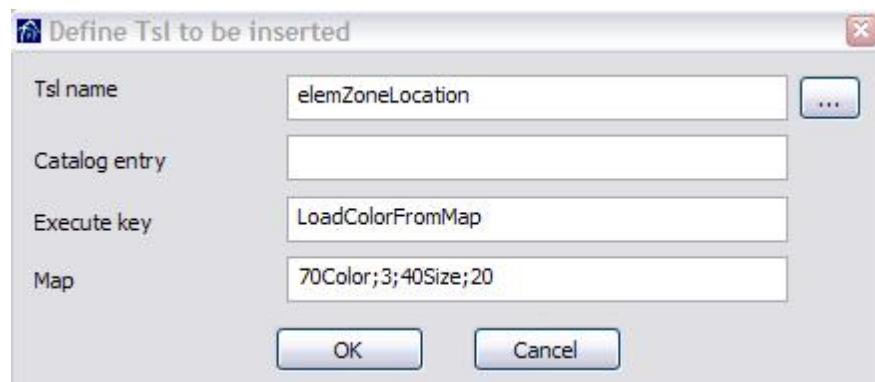
When hitting the New, Edit or Copy button, the [TSL editor window](#) will appear.



## 2.3 Auto insert with Beam, Element,...

When a TSL is selected to be added automatically with a tool, beam or an element, there are a number of parameters involved:

- Tsl name: the script name itself
- Catalog entry: Each Tsl script has a catalog of property values. The catalog can be maintained through the property dialog of the Tsl.
- Execute key: The execute key is a user definable running state variable. It is a parameter that is available from within the script by the \_kExecuteKey variable. It can be used to specify a special action.
- Map: The contents of the persistent map, \_Map can be specified as a dotcomma separated key value list.

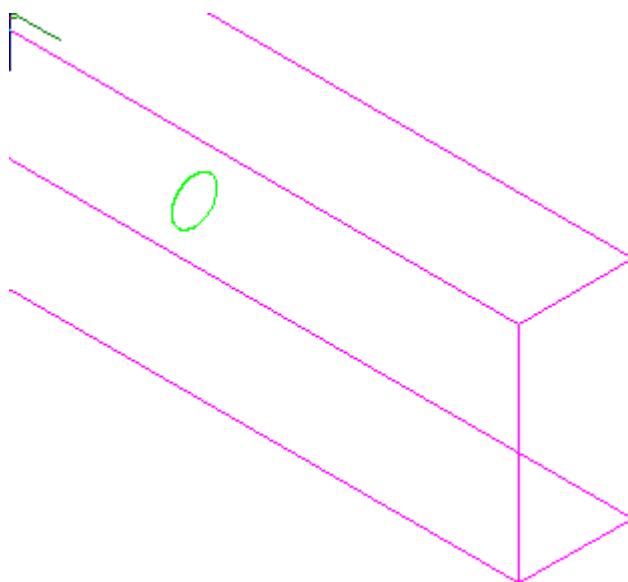


Below you find examples of Tsl's that use the Execute key and Map contents, as shown in the image above.

---

[Example E-type to be attached to a beam:

Purpose is to draw a circle at the center of a Beam. The color is set through a property of the TSL. The radius of the circle has a default value of 100 mm, but can be overwritten by the contents of the Map.



```
Unit(1, "mm");

PropInt pColor(0,-1,"Color index"); // -1 is color by entity

if (_Beam.length()==0) { eraseInstance(); return; }
Beam bm0 = _Beam[0];
```

---

```

// calculate _Pt0 from arrow location
_pt0 = bm0.ptCen();

// initialize a property from the map, on the event called
"LoadColorFromMap"
reportMessage("\nProcess beam:" +bm0.posnum());
if (_kExecuteKey=="LoadColorFromMap") {
    reportMessage("\n\t_kExecuteKey==" + _kExecuteKey);
    if (_Map.hasInt("Color")) {
        int nColor = _Map.getInt("Color");
        pColor.set(nColor);
        reportMessage("\n\tColor set from map to "+nColor);
    }
}

// use the size from the map if present
double dSize = U(100);
if (_Map.getDouble("Size")) {
    dSize = _Map.getDouble("Size");
}

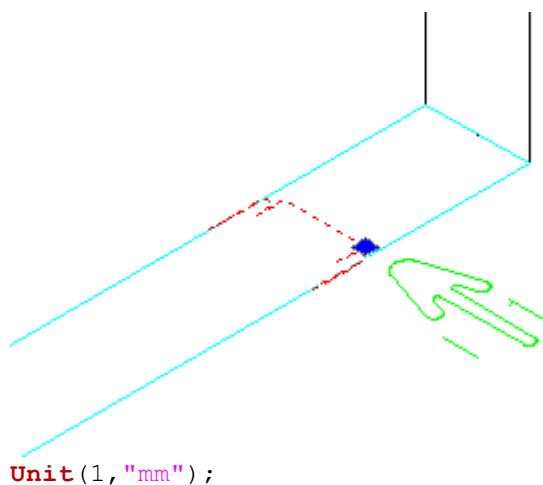
Display dp(pColor);

PLine plCircle;
plCircle.createCircle(_Pt0,bm0.vecX(),dSize);
dp.draw(plCircle);

—end example]

```

[Example O-type, insert implemented in script, to be appended to an element:  
The purpose of the Tsl is to give a graphical representation of the thickness of the different zones of the element.



```

PropInt pColor(0,-1,"Color index"); // -1 is color by entity

if (_bOnInsert) {

    _Element.append(getElement());
    return;
}

if (_Element.length()==0) { eraseInstance(); return; }
Element el = _Element[0];

// calculate _Pt0 from arrow location
_Pt0 = el.ptOrg();
ElementWall elWall = (ElementWall) el;
if (elWall.bIsValid()) {
    _Pt0 = elWall.ptArrow();
}

// move point to XY plane of element
CoordSys cs = el.coordSys(_Pt0);
_Pt0 = cs.ptOrg();

//cs.vis(1);

// initialize a property from the map, on the event called
"LoadColorFromMap"
reportMessage("\nProcess element:" + el.number());
if (_kExecuteKey=="LoadColorFromMap") {
    reportMessage("\n\t_kExecuteKey==" + _kExecuteKey);
    if (_Map.hasInt("Color")) {
        int nColor = _Map.getInt("Color");
        pColor.set(nColor);
        reportMessage("\n\tColor set from map to " + nColor);
    }
}

// use the size from the map
double dSize = U(100);
if (_Map.hasDouble("Size")) {
    dSize = _Map.getDouble("Size");
}

setDependencyOnEntity(el);

Display dp(pColor);

for (int z=-5; z<6; z++) {

    ElemZone zn = el.zone(z);
    double dPosZ = zn.dPosZ();
    Vector3d vecZ = zn.vecZ();
    Point3d pt0 = _Pt0 - 0.5*abs(z)*dSize*cs.vecX() + dPosZ*cs.vecZ();
}

```

---

```

double dH = zn.dH();

Point3d pt1 = pt0-dSize*cs.vecX();
Point3d pt2 = pt0;
Point3d pt3 = pt0+dH*vecZ;
Point3d pt4 = pt3-dSize*cs.vecX();

PLine pl(pt1,pt2,pt3,pt4);
dp.draw(pl);

PlaneProfile pr = el.profBrutto(0);
pr.vis(1);
}

```

*—end example]*

## 2.4 Context menu

When a script entity is selected in the drawing, and the context menu is opened (often right click on mouse), some TSL specific menu commands are found.

### Properties dialog

When multiple tsl entities are selected, it is possible to modify only some parameters of certain tsl. Also the property values can be stored in a catalog. The command is called "Hsb\_PropertiesDialog".

### Reset OPM

The OPM (Object property manager) of autocad, caches the set of properties of an entity. The amount, and type of properties of an entity is not supposed to change. But using the TSL script, this is possible.

- One could have definitions of properties depending on the value of some other properties or conditions;
- Also after an entity has been created, and appended to the drawing, one can change the script, adding some more properties.

If the amount or type of some properties is changed, the OPM gets confused. To reset the OPM for a particular entity, on can use the command "Hsb\_ResetOpm", which can also be fired from the context menu.

### Edit script

To go directly to the editor window of the script definition, the command "Hsb\_ScriptDef" can be used, which is also available from the context menu.

### TSL Viewer

Besides the script editor, there is also a TSL script viewer available. The command is called "Hsb\_TslViewer".

### Custom

The TSL script can define its own list of custom context menu actions. They appear under the sub menu called "Custom". For each of these custom actions, the command "Hsb\_RecalcTslWithKey"

is fired. In fact this command can also be used in eg the toolpallete to activate special executions of tsl instances.

The custom actions are defined inside the script using the global function:

```
addRecalcTrigger(int nTriggerType, String strTriggerKey);
```

When the ts is executed, the predefined variable `_kExecuteKey` is set to the `strTriggerKey`. The `nTriggerType` must be equal to `_kContext`. As you can see in the example below, the variable `_bOnRecalc` is also TRUE during such an execution.

All input routines are made available during this special execution mode. This means `getEntity`, `getPoint`,... but also the `setPropValuesFromCatalog`,...

If multiple tsl entities are selected, only the common context actions are made available in the context menu. The recalculation of all of these tsl's will be triggered. Please keep this in mind when you add user input calls to the custom action.

The maximum number of context triggers that will be made visible is 100.

The following is obsolete since 21.4.98 and 22.1.103 and 23.0.75

If the trigger string is **prefixed with ../()**, then the entry will appear in the main context menu, as opposed to the normal "Custom" flyout (available from hsbCAD2017 build 21.4.37 and build hsbDesign22 build 22.0.74). Eg:

```
String strChangeEntity = T("../|Change entity|");
```

It is good practice to keep the ../ outside the translation part.

Added since 21.4.98 and 22.1.103 and 23.0.75:

To put a command in the main context menu use `_kContextRoot` instead of `_kContext`. No need to use the ../ in front of the trigger key.

Since hsbCAD v19.1.47, the lisp command `Hsb_RecalcTslWithKey` also accepts a second argument, which is the overwrite of the prompt.

Fire

**(hsb\_recalcTslWithKey "myrecalc" "Select the ones to recalculate")**

to have "myrecalc" be set as `_kExecuteKey` and "Select the ones to recalculate" to prompt the user to select the tsl instances.

Since hsbCAD v19.1.103, and v20.0.75, the key and prompt can contain a | character, in which case they string will presented to the translation engine.

When the left or right shift key is pressed or ctrl key is pressed during launching of the command, the following variables are assigned a value (present from hsbcad v21.0.103);

`_kShiftKeyPressed`: 1 if left shift key is pressed, 2 if right shift key is pressed, 0 otherwise

`_kCtrlKeyPressed`: 1 if left ctrl key is pressed, 2 if right ctrl key is pressed, 0 otherwise

---

*[Lisp code illustrating the custom context menu. Also the use of `_kExecuteKey`, and the `setPropValuesFromCatalog` are illustrated.*



```

int arColorInd[]={1,2,3};
String arColorName[] = {T("|red|"),T("|yellow|"),T("|green|")};
PropString pColor(0,arColorName,"Color");

if (_bOnInsert) {

    // the _kExecuteKey is set by the (Hsb_ScriptInsert "tslname"
    "ExecuteKey")
    reportMessage("\n_kExecuteKey is " + _kExecuteKey);

    // During insert, an execute key or any other string can be used
    to load the properties saved in the catalog.
    setPropValuesFromCatalog(_kExecuteKey); // might change the value
    of pColor

    _Entity.append(getEntity());
    _Pt0 = getPoint();

    // do not return here, but continue the script performing the
    action later on.
}

if (_Entity.length()==0) {
    eraseInstance();
    return;
}

reportMessage(scriptName() + ", execute key: " + _kExecuteKey
+ ", shift pressed: " + _kShiftKeyPressed
+ ", ctrl pressed: " + _kCtrlKeyPressed );

String strChangeEntity = T("|Change entity|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    _Entity[0] = getEntity();
}

String strLoadCatalogGreen = T("|Load catalog entry called
\"green\"|");
addRecalcTrigger(_kContext, strLoadCatalogGreen );
if (_bOnRecalc && _kExecuteKey==strLoadCatalogGreen) {
    reportMessage("\ncolor was "+pColor);
}

```

```

        setPropValuesFromCatalog("green"); // might change the value of
        pColor
        reportMessage(" and is changed into "+pColor);
    }

    // Do the actual action
    Entity ent = _Entity[0];

    int nEntColor = ent.color();
    int nPropColor = arColorInd[arColorName.find(pColor,0)];

    if (nEntColor!=nPropColor) {
        ent.setColor(nPropColor);
    }

—end sample]

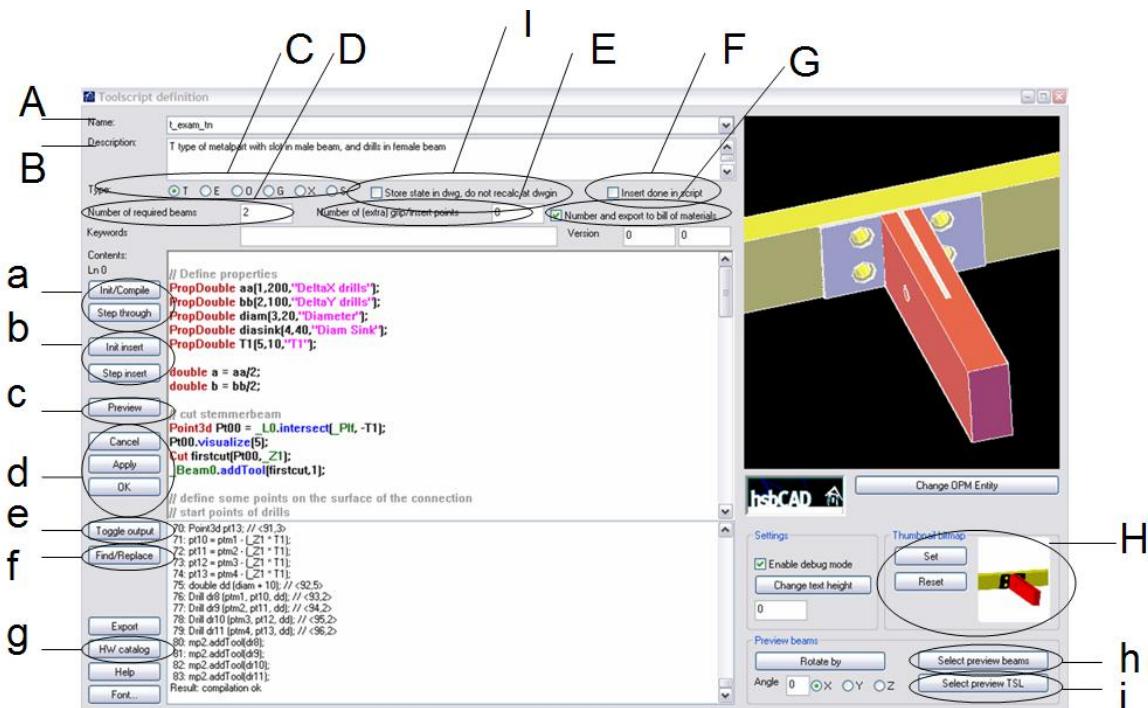
```

## 2.5 The old editor window

The editor window appears when the edit, copy or new button is pressed in the "[Select Metalpart/Tools description](#)" dialog.

The editor window will also appear when the "[Edit script](#)" is selected in the context menu of a TSL instance.

The TSL editor window is shown below. It allows to define the contents and all the properties of a TSL script.



### TSL script properties

- A. Each script has a name. The script name has to be unique inside the drawing.

- B. The description is a chunk of information that is shown inside the "[Select Metalpart/Tools description](#)" dialog.
- C. The type of the TSL, determines how the instance will behave during beam copy and erase. For a T-type, at least 2 beams need to exist. For an E type one is sufficient. An O type does not require the existence of any beams. The type also determines the set of [predefined variables](#): \_H1 has no meaning for a E-type with only one beam, \_X is only defined for a G-type, the location of \_Pt0 is calculated differently for each type... The type also determines the default insert procedure.
- D,E. During insert of a TSL instance into the drawing, a number of extra beams, and grippoints might be required to be selected. These can be specified in these fields. The numbers that are indicated here only influence the number of beams and points that the user will be queried for during selection. The numbers do not determine the maximum number of beams or points that the script can use or work with, or can actually have. It is very well possible that some beams are added to the TSL instance after the TSL has been inserted into the drawing, by applying the "Integrate tooling" command.
- F. When the flag "Insert done in script" is off, then a default procedure is followed during the insertion of a new TSL instance into the drawing. If the flag is on however, the script has to implement the actions that need to be taken, and the queries that need asked while inserting a new instance into the drawing. Examples can be found at the "[Implementing insert in the script](#)" topic, and following.
- G. If the TSL instance describes a metalpart, the metalpart needs to be represented by a solid. If solids of different TSL instances are the same (and if they have the same key), they can be given the same number. A numbered TSL will appear in the DXA file format, which is used to export data to bill of material. The numbering and the export to bill of materials will only be done if the flag is on.
- H. A thumbnail bitmap can be assigned to each TSL. This thumbnail will be used during file browsing, and during selection of the TSL.
- I. The toggle called: "Store state in dwg, do not recalc at dwg" is a newer toggle to improve opening speed of drawings. See the topic on [store state in dwg](#).

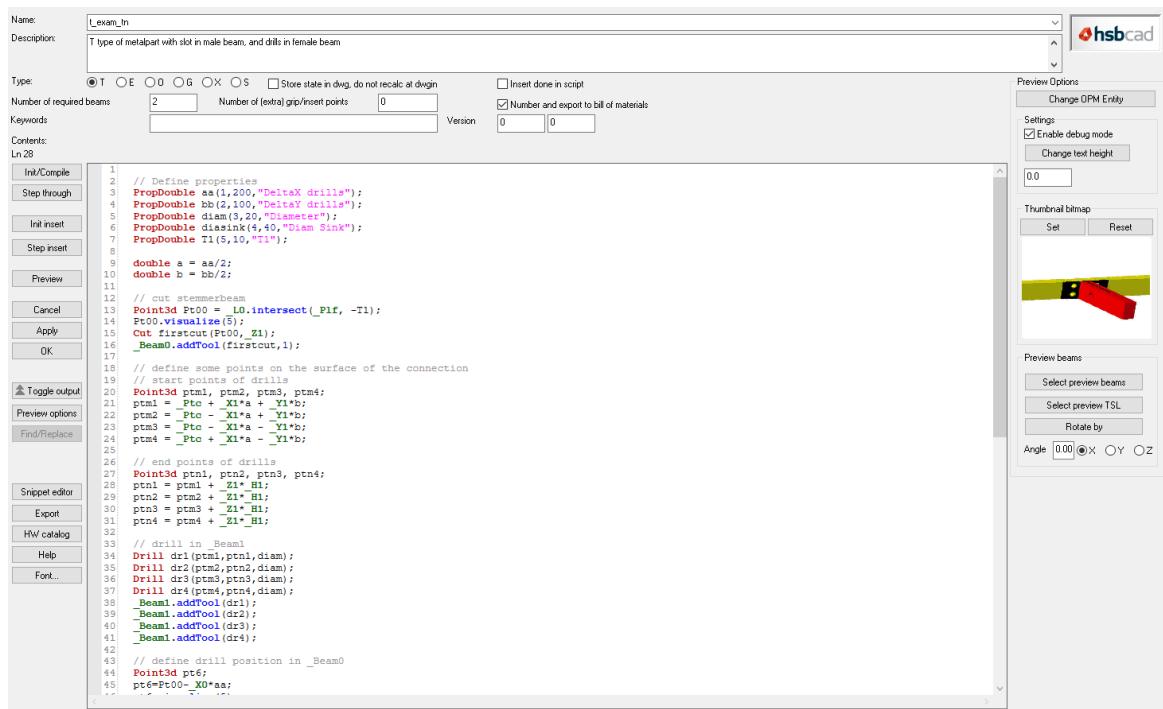
#### Editor functionality

- a. Before the script can be executed, the script is compiled. Compilation means that the text representation is interpreted, and translated into an internal binary representation. Compilation is done automatically when the OK or APPLY button is pressed, but to check the syntax of the script, the INIT/COMPILE button can be pressed too. In order to debug the script, one can step through the script, line by line, and investigate the values of the variables. When the STEP THROUGH button is pressed, the script will be executed to the line with the cursor location. To restart the execution, the INIT/COMPILE button must be pressed again. The status of the compilation, but also the values of the variables can be seen after pressing the TOGGLE/OUTPUT button.
- b. If the script is implementing the insert procedure, the "Implementing insert in script"-flag is on, the special execution of the script can be debugged, and stepped through. During insertion, the predefined variable \_bOnInsert will have a TRUE value. Also the prompt classes in the TSL will be able to pose a question, and accept point and entity selections, during insert. From this it is obvious that the execution of the script will be different during insert, compared with the normal execution. The INIT INSERT and STEP INSERT buttons allow to step through, and debug the execution while inserting the script.
- c. Hitting the PREVIEW button, will execute the script, and display the result in the graphical preview window.
- d. When the OK or APPLY button is pressed, the script is compiled. If the compilation is ok, the TSL instances inside the drawing that refer to this script are updated. The difference between APPLY and OK, is that on OK the editor window will be closed, and on APPLY, the editor window will stay open. If CANCEL is pressed, the changes are discarded.

- e. The TOGGLE OUTPUT, reduces the size of the source text window, and shows an output window, with the status of compilation, or the values of the variables during step through execution.
- f. Find and replace assists in editing the text source of the TSL script.
- g. To write a hardware instruction line in the script, with values from the hardware catalog the HARDWARE CATALOG button can be used. First put the cursor at the position where the new hardware instruction need to be inserted, then hit the button.
- h. Whenever the preview button is pressed, the script is executed with a standard configuration of some beams. Eg. a T connection will have 2 perpendicular beams, an E type will only have one. If the script needs to be shown or tested for a particular configuration of beams, the beams can be selected from the drawing on SELECT PREVIEW BEAMS button pressed. In the case the insert is implemented inside the script, the script will be executed in insert mode. If insert is not implemented inside the script, the default insert procedure is followed for selecting beams and points for that particular type.
- i. As an alternative to selecting a set of beams and points, an existing TSL instance can be selected to be used as source for the preview entity.

## 2.6 The new editor window

In hsbcad v21, a new Tsl editor was introduced.



The new tsl editor also has an extended [list of shortcuts](#).

## 2.7 The newest editor window

Since some versions we have a 3rd generation of editor. This one supports break points, state of variables at that break point, intellisense while typing, and so much more:

The screenshot shows the TSL Grip Point Drag editor window. The title bar says "TslGripPointDrag: 0.10". The menu bar includes Home, Settings, Find, Replace, Comment, Maps, Watch, Output, Preview, Windows, Editor, and Help. The Editor X pane contains C++ code for drawing circles and lines. The Watch pane shows variable states:

Name	Value	Type
_Pt0	(1234,162,0)	Point3d
_ZU	(0,0,1)	Vector3d
_kGripPointDrag	6	int
_kExecuteKey	String	
_bOnInsert	0	int
_bOnGripPointDra	0	int
_ThisInst	TslInst(3A21)	TslInst
_PtG	> Count: 1	Point3d[]
dp	Display(0)	Display
i	0	int

At the bottom, there's a message "Result: compilation ok" and tabs for Maps, Watch, and Output.

## 2.8 Graphical metalpart definition

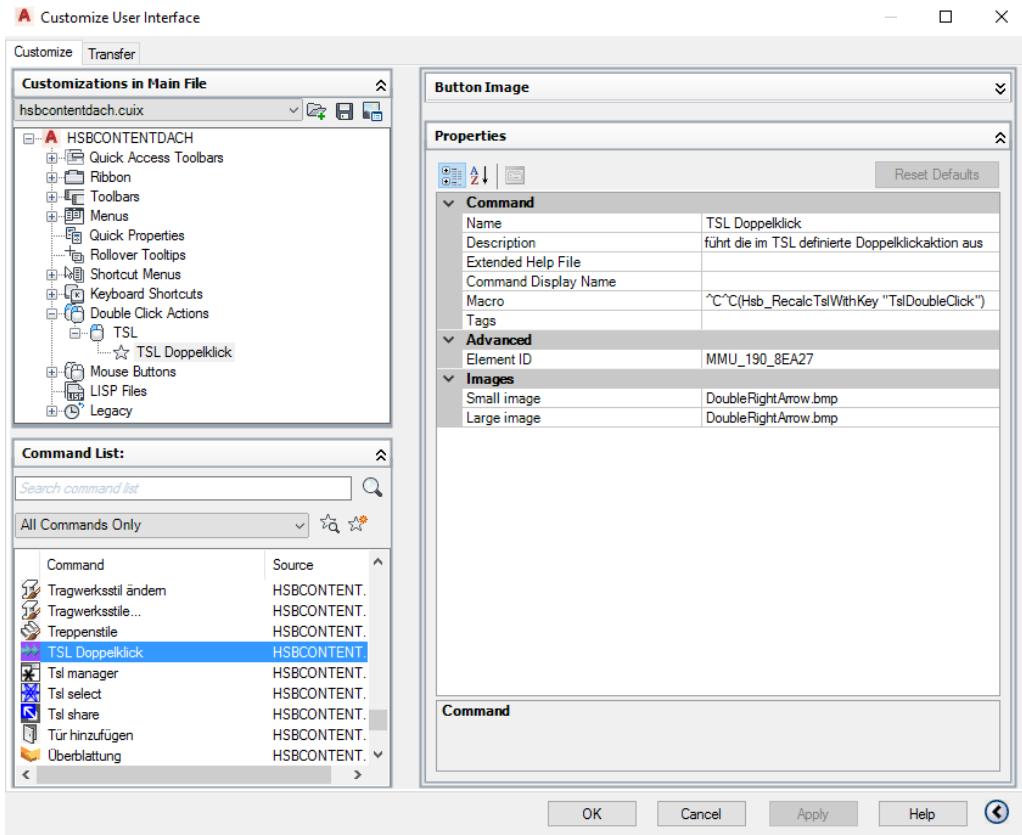
There are two basic methods to create tools or metal pieces:

1. From polylines drawn on the faces of the timbers describing the shape of the tools or metal parts (the program will automatically develop the script code and generate the solids described by the polylines), or
2. By developing and entering the script code for the generation of the tool or metal piece.

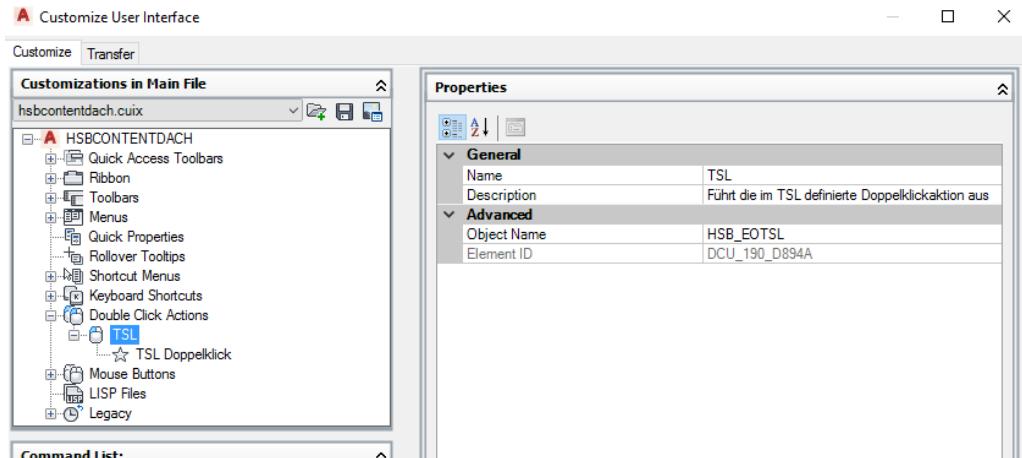
## 2.9 Double click

To add double click actions for tsl you must add a command and the corresponding action:

Add a new command `^C^C(Hsb_RecalcTslWithKey "TslDoubleClick")` in one of the cuix files.



- 1.
2. Add new action

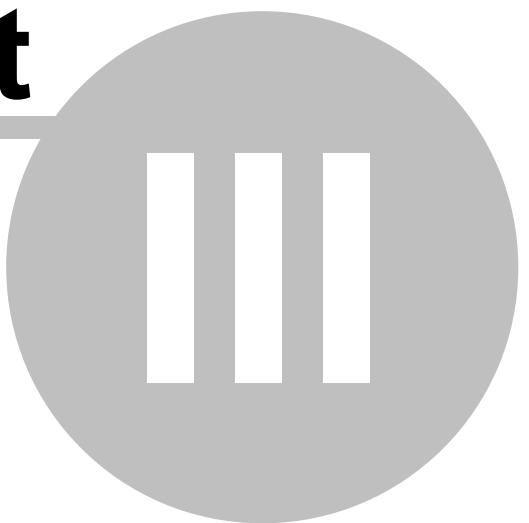


Then you can do the following in Tsl similar to [context menu](#) actions:

```
String sDoubleClick= "TslDoubleClick";
if (_bOnRecalc && _kExecuteKey==sDoubleClick)
{
    [...]
```

# **Part**

---



## 3 Syntax

### 3.1 Introduction

The toolscript uses a syntax similar to that of the C language.

The script or language consists of a number of statements and comments.

Some other conventions adopted for the toolscripts are:

- the ; symbol acts as terminator of the statements;
- Spaces and new lines are not relevant, they are simply considered as token separators,
- Dots are function operators, they call a function to be applied on the variable entered before the dot.

### 3.2 Comments

Comments can be notes about the development, tool or metal piece but they can also be used to explain the meaning of variables for later maintenance.

There are two types of comments: comments to the end of the line, starting with //, or comments enclosed by /\* and \*/

[Example:

```
// this is a comment to end of line

/* this is a comment enclosed, it can appear inside an instruction */

/* another comment spanning over
   multiple lines */

/*
  This is the
  powerfull C-type of comment
  // still ok
  The drawback however, is that it is not nestable !!
*/
```

// This // comment is however nestable, as  
// // you see ?

—end example]

### 3.3 Statements

Except as indicated, statements are executed in sequence. The following types of statements exist:

*statement:*

- expression-statement*
- compound-statement*
- selection-statement*
- iteration-statement*
- declaration-statement*
- jump-statement*

Each of the above types are explained in more detail in the following sections: [expression](#), [compound](#), [selection](#), [iteration](#), [declaration](#) and [jump statements](#).

### 3.4 Expression statements

Expression statements have the form

*expression-statement:*  
    *expression opt ;*

The expression is evaluated and its value is discarded. All side effects from an expression statement are completed before the next statement is executed. An expression statement with the expression missing is called a null statement. [Note: A lot of statements are expression statements—usually assignments. A null statement is useful to supply a null body to an iteration statement such as a while statement]

[Example:

```
int i, j;
i = 3;    // assignment is expression
i+3;     // is a valid expression statement, but i is not changed
i++;     // expression statement where i is incremented
—end example]
```

### 3.5 Compound statements or blocks

So that several statements can be used where one is expected, the compound statement (also, and equivalently, called "block") is provided.

*compound-statement:*  
    { *statement-seq opt* }  
*statement-seq:*  
    *statement*  
    *statement-seq statement*

A compound statement defines a local scope. [Note: a declaration is a *statement*. ]

[Example:

```
int i = 2;
int j = 2;
{
```

```

int i = 4;      // ok, i will have value 4 inside {}
j = 4;          // no redeclaration
}
int k = i + j; // k = 2 + 4;
—end example]

```

## 3.6 Selection statements

Selection statements choose one of several flows of control.

*selection-statement:*

```

if ( condition ) statement
if ( condition ) statement else statement

```

*condition:*

```

expression
type-specifier-seq declarator = assignment-expression

```

If the condition yields true the first statement is executed. If the else part of the selection statement is present and the condition yields false, the second statement is executed.

The statement in a *selection-statement* (both statements, in the else form of the if statement) implicitly defines a local scope. If the statement in a selection-statement is a single statement and not a *compound-statement*, it is as if it was rewritten to be a compound-statement containing the original statement.

[Example:

```

if (x)
    int i;

```

can be equivalently rewritten as

```

if (x) {
    int i;
}

```

Thus after the if statement, i is no longer in scope.]

The rules for *conditions* apply both to *selection-statements* and to the for and while statements.

A name introduced by a declaration in a *condition* (either introduced by the *type-specifier-seq* or the *declarator* of the condition) is in scope from its point of declaration until the end of the statements controlled by the condition. If the name is re-declared in the outermost block of a statement controlled by

the condition, the declaration that re-declares the name is ill-formed.

[Example:

```

if (int x = 3) {
    int x; // ill-formed, redeclaration of 'x'
}
else {
    int x; // ill-formed, redeclaration of 'x'
}
—end example]

```

The value of a *condition* that is an initialized declaration is the value of the declared variable implicitly converted to type int. The value of a *condition* that is an expression is the value of the expression,

implicitly converted to int; if that conversion is ill-formed, the program is ill-formed. The value of the condition will be referred to as simply "the condition" where the usage is unambiguous.

[Example:

```

int j = 10, i = 0;
Proplnt nSelectionFlag(0,1);

if ( (nSelectionFlag>=0) && (nSelectionFlag<4) ) { // more complex

    if (nSelectionFlag==1) {
        j = 20;
        i++;           // equivalent with i = i+1;
    }
    else if (nSelectionFlag!=2) {
        j = 20;
        i--;           // equivalent with i = i-1;
    }
    else j=30;          // not a compound statement
}
else if (nSelectionFlag<0) {
    j += 3;           // equivalent with j = j+3;
}
else {
    j = j%4;          // j is set to the j modulus 4;
}

—end example]

```

## 3.7 Iteration statements

Iteration statements specify looping.

*iteration-statement:*

```

while ( condition ) body-statement
do body-statement while ( expression );
for ( for-init-statement condition opt ; post-expression opt ) body-statement

```

*for-init-statement:*

```

expression-statement
simple-declaration

```

*post-expression:*

```

expression

```

*body-statement:*

```

statement

```

[Note: a *for-init-statement* ends with a semicolon. ]

The statement in an *iteration-statement* implicitly defines a local scope which is entered and exited each time through the loop. If the statement in an iteration-statement is a single statement and not a *compound-statement*, it is as if it was rewritten to be a compound-statement containing the original statement.

[Example:

```

while ( --x >= 0)
    int i;

```

can be equivalently rewritten as

```
while (--x >= 0) {
    int i;
}
```

Thus after the while statement, *i* is no longer in scope. ]

### 1 The while statement

In the while statement the statement is executed repeatedly until the value of the condition becomes false. The test takes place before each execution of the statement.

When the condition of a while statement is a declaration, the scope of the variable that is declared extends from its point of declaration to the end of the while *statement*.

### 2 The do statement

The expression is implicitly converted to bool; if that is not possible, the program is ill-formed.

In the do statement the substatement is executed repeatedly until the value of the expression becomes false. The test takes place after each execution of the statement.

[Example:

```
int j = -10;
do {
    j--;
    // will always be executed once
}
while (j>0);
int mj = j; // mj will be -11
—end example]
```

### 3 The for statement

The for statement

for (*for-init-statement* *condition opt* ; *post-expression opt*) *body-statement*  
is equivalent to

```
{
    for-init-statement
    while ( condition ) {
        body-statement
        post-expression ;
    }
}
```

except that names declared in the *for-init-statement* are in the same declarative-region as those declared in the *condition*. [Note: Thus the first statement specifies initialization for the loop; the condition (6.4) specifies a test, made before each iteration, such that the loop is exited when the condition becomes false; the expression often specifies incrementing that is done after each iteration.]

Either or both of the condition and the expression can be omitted. A missing *condition* makes the implied while clause equivalent to while(true).

If the *for-init-statement* is a declaration, the scope of the name(s) declared extends to the end of the *for-statement*. [Example:

```
int i = 42;
int a[10];
for (int i = 0; i < 10; i++) {
    a[i] = i;
}
int j = i; // j = 42
—end example]
```

## 3.8 Declaration statements

A declaration statement introduces one or more new identifiers into a block; it has the form  
*declaration-statement*:

*block-declaration*

If an identifier introduced by a declaration was previously declared in an outer block, the outer declaration

is hidden for the remainder of the block, after which it resumes its force.

Variables with automatic storage duration are initialized each time their *declaration-statement* is executed. Variables with automatic storage duration declared in the block are destroyed on exit from the block.

Every type has a default constructor.

[Example:

```
int j = -10;
int i, j=4, k(4), l[3]; // block declaration
int &ri = i;           // ri is declared as a reference to i

int ll;               // default constructor for int
MetalPart mp1; // default constructor for MetalPart
                   // => no storage assigned
—end example]
```

In order to use a variable, the variable must be declared. The type of the variable must be set. To do this, the type name is written before the variable name. Examples are:

[Example:

```
int name1;
double name2;
Point3d name3;
Vector3d name4;
String name5;
```

—end example]

In this manual, you will find a lot of type definitions see ([Point3d](#), [PLine](#), [Beam](#),...). Each type definition contains the declaration of the list of constructors, and the list of member functions that can be called on a variable of that type. Here we see part of the definition of the Point3d type.

```
class Point3d {
    Point3d(double x, double y, double z);
    Point3d(Point3d pt);
    ...
    // individual component access
    double X() const;
    double setX(double dComponentX);
    ...
};
```

The const at the end of a function declaration informs the user that by calling the function, the class instance will not change. In case the variable is a reference, the const expresses that the referenced value will not change.

For the Point3d type, this means the following. If we have a point pt1 defined: "Point pt1(10,20,30);", and we call the function X() upon this pt1, the point pt1 itself will not change. The function is declared const. But if we call setX(...), the point will indeed change.

```
pt1.setX(11); // change the x-coordinate of the point.
double xx = pt1.X(); // just use the x component of the point. The point pt1 does not change.
```

Since a beam refers to (or references) a beam-entity in the autocad drawing, const functions will not change the beam-entity in the drawing, while non-const functions do change the entity. So with a beam, bm defined eg.:

```
Beam bm = _Beam0;
bm.filterBeamsContactCut(...); // calling this one on bm does not change the beam
bm.dbSplit(...); // but calling this one, does change the beam in the drawing.
```

## 3.9 Array declaration

An array is a 0-based indexed one dimensional list of items of the same type. Each item can be addressed with its index. The first valid index is 0, the last index is (length()-1). In TSL, arrays are not limited in size, only by the available memory. Also the size of arrays can change at any time. When using an index to access an array the index is always checked against the allowed index range of that array instance at run time. An error will be reported whenever an attempt is made with an invalid index.

Arrays can be declared of all types. This is done by providing an array size in the declaration, or an initialization list. The array size, if specified, can be an expression. The syntax is:

```
<type> name[int nSize];
eg.: int nBB[4];
```

An array can be initialized at time of declaration. In this case, the size is determined by the initialization list. No size must be specified. The values in the initialization list can be expressions:

```
<type> name[] = { <value1>, <value2>, ... , <valueN> };
eg.: int nCC[] = { 10, 20, 22, 30 };
```

The size of an array is not fixed. Arrays can change length dynamically. For that the following functions are available.

To append one new item to the array:

```
<array>.append(<item>);
```

To append an existing array of items to the array:

```
<array>.append(<array of items>);
```

To set the length of the array to a specific value, either bigger, equal or smaller than the existing length:

```
<array>.setLength(int nNewLength);
```

The length can be queried by the function:

```
int <array>.length() const;
```

To remove or insert an item in the array one can use the following functions:

```
<array>.insertAt(int nIndex, <item>);
```

```
<array>.removeAt(int nIndex);
```

To access the values of the items of an array the [] operator is used. The index of an element is 0-based: the first element has index 0, and the last element has index (length()-1). But there are 2 special methods

```
<array>.last(); // added v21.4.14
```

```
<array>.first(); // added v21.4.14
```

The following function returns the index of the argument value:

```
int nIndex = <array>.find(<value>[,int nDefaultIndex]);
```

If the value is not found the default index is returned. If the default index is not specified, the value -1 is returned.

The following function inter changes the contents of the items with the indices nIndex1 and nIndex2:

```
<array>.swap(int nIndex1, int nIndex2);
```

If one of the indices is out of the index range, nothing is done.

The method reverse flips the sequence in the array itself. It is a void method, meaning it changes the array itself, and does not return the result.

```
void <array>.reverse(); // added since 22.1.132 and 23.5.9
```

The method filterValid returns a new array with only entities which pass the test blsValid(). The method only works on arrays of any type of entity. It returns the same type. The method does not change the array itself.

```
Entity[] <entity array>.filterValid() const; // added since 22.1.132 and 23.5.9
```

There is a special function on an array of String, double and int: sorted

```
String[] arSorted = arNotSorted.sorted(); // added hsbcad v21.0.115
```

```
double[] arSorted = arNotSorted.sorted(); // added 24.1.90, 25.1.62
```

```
int[] arSorted = arNotSorted.sorted(); // added 24.1.90, 25.1.62
```

For an array of strings, there is a special find as well:

```
int nIndex = String[].findNoCase(<value>[,int nDefaultIndex]);
```

The use of arrays is illustrated in the following example.

[Example:

```

int aiList[] = { 10, 20, 30 };
aiList.append(40);
int nSum = 0;
for (int i=0; i<aiList.length(); i++) { // loop over all elements: 4
    nSum += aiList[i];           // sum every element
}
double adList[5];
// initialization in for loop
for (int i=0; i<adList.length(); i++) adList[i] = 1;

String strList[] = { "message", "warning" };
if (nSum>20) strList.append("error");

// use array as default value, will turn property into enumerated one.
PropString errType(2,strList,"Type of error");

// find the index position of the chosen property
// If it is not in the list, return index 0.
int nIndex = strList.find(errType,0);

```

—end example]

## 3.10 Jump statements

There are tree jump statements available

*jump-statement:*

```

break ;
continue ;
return ;

```

The break and continue statement can only be used inside an iteration statement. The return statement can be used anywhere in the code.

When inside an iteration statement the break is called, the iteration statement is stopped imediately, and control is passed to the next statement after the iteration statement. The *condition* of the iteration statement is not executed. For a for loop, the *post-expression* is not executed anymore.

When inside an iteration statement the continue is called, then the control is passed to the end of the iteration *body-statement*. The *condition* and *post-expression* are executed, and depending on that result, the *body-statement* is re-executed.

The return statement immediately ends the execution of the script.

[Example:

```

int s=0;
for (int i=0; i<4; i++) {
    s++;

```

```

        if (i==1) continue; // jump to the end of the for-loop body
        s+=10;
        if (i==2) break; // end the for-loop execution
        s+=100;
    }
    reportMessage("Sum=" +s); // report the value of s

```

—end example]

In the above example, the loop is initially set to be executed 4 times, for values of i: 0,1,2,3. After the first execution, the value of s is 111. The second execution is terminated by the continue statement, which results in an s value of 112. The third execution is terminated by the break, with an s value of 123. Because the break statement was executed, the for-loop is terminated, and the fourth sweep is not done. The value of s is 123 at the end.

## 3.11 Maths on double and int

A complete set of mathematical functions and operators is defined:

sqrt, +, -, \*, /, --, ++, % (mod), +=, -=, \*=, /=, abs, exp, log, log10, ceil, pow, sqr, sign, round

Besides these, [logical and bitwise operators](#) exist. The classic [goniometrical functions](#) are also available.

---

[Example code:]

```

double dd ;
dd = sqrt(4); // dd = 2
dd = exp(1); // dd = 2.71828
dd = pow(9,0.5); // dd = 3
dd = ceil(4.345); // dd = 5
dd = round(4.345); // dd = 4
dd = round(4.845); // dd = 5
dd = ceil(-4.345); // dd = -4
dd = round(-4.845); // dd = -5
dd = log(exp(3)); // dd = 3
dd = exp(log(4.56)); // dd = 4.56
dd = log10(1); // dd = 0
dd = log10(10); // dd = 1
dd = sign(-2.3); // dd = -1
dd = sign(0); // dd = 0
dd = sign(23); // dd = 1
dd = sqr(10); // dd = 100

```

—end example]

## 3.12 Logical operations

Equal: ==, less than: <, greater than: >, less or equal: <=, greater or equal: >=, not equal: !=, and: &&, or: ||, not: !, also the ( and ) can be used.

The result of a logical expression is an integer e.g.: int a = 10>0;

A <false> is represented by 0, a <true> is any other integer value.

There are some predefined values: true, false, TRUE, FALSE: e.g.: int a = true;

These logical operations can be used to compose a condition in the following expression:

<condition> ? <true expression> : <false expression >;

E.g.: double a = (3<2) ? 10 : 5; // a will be given the value 5

Logical expressions are typically used in [jump statements](#) (if else), and in [iteration statements](#) (for, while).

---

## 3.13 Angle calculations

A number of functions are available to calculate and use angles. Remember, angles are always expressed in degrees (0,360).

For rotateBy and angleTo function see [Point and Vector manipulation](#).

---

```
double sin(double dAngle); // returns a value in the range [-1.1]
double cos(double dAngle); // returns a value in the range [-1.1]
double tan(double dAngle);
```

The sin, and cos functions return a value in the range [-1,1]. The tan is an unlimited function. The acos and asin functions must have an argument which is in the range [-1,1]. If not, an error will be thrown.

The sin of an angle between two unit vectors is also given by the value of the crossProduct of the two vectors. The dotProduct of two unit vector is giving the cos value.

---

```
double asin(double x); // inverse of sin, returns an angle [-90,90]
double acos(double x); // inverse of cos, returns an angle [0,180]
double atan(double x); // arctangent, returns an angle [-90,90]
double atan(double x, double y); // returns the atan of x/y in the range [-180,180[
```

The atan returns the arctangent of x. or of y/x. If x is 0, atan returns 0. If both parameters of atan are 0, the function returns 0. atan returns a value in the range -90 to 90 degrees; atan with two arguments returns a value in the range -180 to 180 radians, using the signs of both parameters to determine the quadrant of the return value.

---

Remember:

The following functions can be used as long as we have a right-angled-triangle (has one angle of 90°) see figure below.

$$c = \frac{a}{\cos \beta} \quad b = \frac{a}{\tan \alpha} \quad c = \frac{a}{\sin \alpha}$$

$$a = c \times \sin \alpha$$

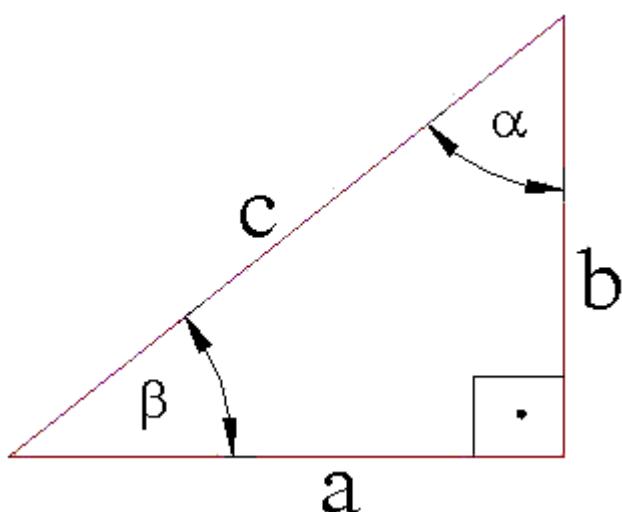
$$a = b \times \tan \alpha$$

$$b = c \times \cos \alpha$$

$$\tan \alpha = \frac{a}{b}$$

$$\beta = 90 - \alpha$$

$$c = \sqrt{a^2 + b^2}$$



The following rules lets you find sides and angles in non right-angled triangles (see figure below).

Sine rule:

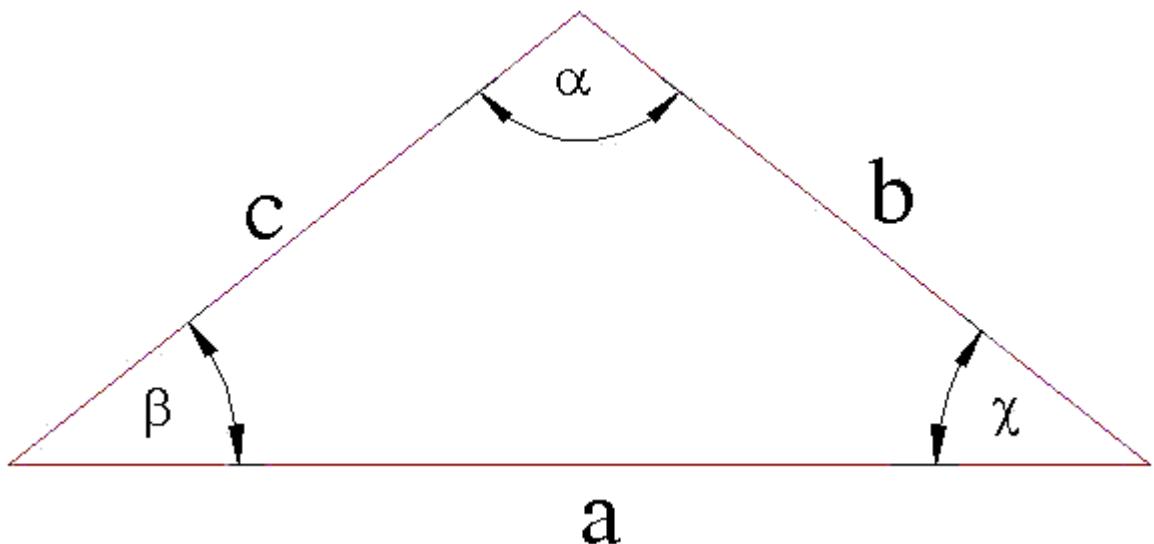
$$\frac{a}{\sin \alpha} = \frac{b}{\sin \beta} = \frac{c}{\sin \gamma}$$

$$a = \frac{b \times \sin \alpha}{\sin \beta} \quad b = \frac{a \times \sin \beta}{\sin \alpha} \quad c = \frac{a \times \sin \gamma}{\sin \alpha}$$

Cosine rule:

$$b^2 = c^2 + a^2 - 2ca \cdot \cos \beta$$

$$\cos \beta = (c^2 + a^2 - b^2) / (2ca)$$



[Example X-type:

```

String strReport;

strReport += "\n Angle between the beams is: " + _X0.angleTo(_X1);

Vector3d vecR = _X0.rotateBy(-120,_X1); // rotate _X0 around _X1
_X0.vis(_Pt0,1);
_X1.vis(_Pt0,1);
vecR.vis(_Pt0,1);
double dAng = _X0.angleTo(vecR); // value [0,180[
double dAng2 = _X0.angleTo(vecR,_X1); // value [0,360[

strReport += "\n dAng is: " + dAng ; // results in 30
strReport += "\n dAng2 is: " + dAng2 ; // results in 330

double dCos = cos(dAng2);
strReport += "\n cos(dAng2): " + dCos ;
strReport += "\n acos(cos(dAng2)): " + acos(dCos) ;

double dSin = sin(dAng2);
strReport += "\n sin(dAng2): " + dSin ;
strReport += "\n asin(sin(dAng2)): " + asin(dSin) ;

double dTan = tan(dAng2);
strReport += "\n tan(dAng2): " + dTan ;
strReport += "\n atan(tan(dAng2)): " + atan(dTan) ;

reportMessage(strReport);

```

—end example]

## 3.14 String

Inside a string the '\n' and '\t' characters can be used for newline and tab. Also the '\' character must be escaped, and substituted by '\\'. And of course the double quote character " must also be escaped, and entered as \".

Since hsbCAD 17.0.42, the atof method and the format method allow for a unit type argument. This controls the format in which the value is displayed. The format can be `_kLength` (default), `_kNoUnit`, `_kArea`, `_kVolume`, `_kAngle`.

---

```

class String {

    String token(int nIndex) const;
    String token(int nIndex, String strListOfSeparators) const; // default
        bCountSequencialSeparatorsAsOne is FALSE
    String token(int nIndex, String strListOfSeparators, int bCountSequencialSeparatorsAsOne)
        const;

    String[] tokenize(String strListOfSeparators) const; // (added v21.1.5 and v22.0.24) default
        bCountSequencialSeparatorsAsOne is TRUE
    String[] tokenize(String strListOfSeparators, int bCountSequencialSeparatorsAsOne) const; //
        (added v21.3.106 and v22.0.24)

    int length() const;

    char getAt(int nIndex) const;

    String spanIncluding(String strOther) const;
    String spanExcluding(String strOther) const;

    String mid(int nFirst, int nCount) const;
    String left(int nCount) const;
    String right(int nCount) const;

    int find(String strToFind, int nFirst) const;
    int find(String strToFind, int nFirst, int bCaseSensitive) const; // default bCaseSensitive is
        TRUE (added v21.3.96 and v22.0.55)

    int atoi() const;
    double atof() const; // standard C style conversion from string to double.
    double atof(int nFormat) const; // (added 17.0.42) Conversion from string to double with support
        for length and other formats: _kLength, _kNoUnit, _kArea, _kVolume, _kAngle

    int replace(String strOld, String strNew); // added v24.1.70 and v25.1.25) returns number of
        replacements done
    int insert(int indexAt, String strToInsert); // added v24.1.70 and v25.1.25) returns new length of
        string

-----

```

---

```

String delete(int nFirst, int nCount);
String trimLeft();
String trimLeft(String strCharsToTrim);
String trimRight();
String trimRight(String strCharsToTrim);
String makeUpper();
String makeLower();

String format(String strFormat, int nV); // see format specification.
String format(String strFormat, double dV);
String format(String strFormat, char cV);
String format(String strFormat, String strV);

String formatUnit(double dV, int nUnits, int nPrec);
String formatUnit(double dV, int nUnits, int nPrec, int nCuttOff); // added V23.8.61, V24.1.64,
V25.0.15
String formatUnit(double dV, String strDimStyle);
String formatUnit(double dV);
String formatUnit(double dV, int nFormat); // (added 17.0.42) The format can be _kLength
(default), _kNoUnit, _kArea, _kVolume, _kAngle

String formatTime(String strFormat); // see format specification.

String mm2GermanDimensionFormat(double dV); // (added 23.0.75)

// QR code
String qrEncode(String strToEncode); // (added 21.3.101) default nMargin is 4
String qrEncode(String strToEncode, int nMargin); // (added 21.3.101) given a normal string,
return the QR:01010 representation of it. See also Display::drawQR.
String qrEncode(String strToEncode, int nMargin, int nErrorCorrection); // (added 23.4.1)
default nErrorCorrection 0
String qrEncode(String strToEncode, int nMargin, int nErrorCorrection, int
nAreaPercentSpaceForImage); // (added 23.4.1) default
nAreaPercentSpaceForImage 0

String qrDecode(String strToDecode); // (added 21.3.101) decode means convert the QR:01010
representation into a readable string
double inQrEmbeddedImageSizeFactor() const; // (added in build v23.4.4)

// images
String encodeImageFromFile(); // (added 23.4.4), assuming the current string is a file name,
read image file, and return the encoded image into a string, which can be drawn by
the Display::drawImage.
double encodedImageHeightOverWidth() const; // (added in build v23.4.4), if the current
string is an encoded image, return the ratio of height over width.
};

String T(String strToTranslate);
String TN(String strToTranslate); // The resulting string is prefixed with "\n"
String T(String strToTranslate, String str2LetterLangCode); // added hsbcad21.1.29 translate string
in any specific language

```

---

```
String TN(String strToTranslate, String str2LetterLangCode); // added hsbcad21.1.29 translate
string in any specific language
```

```
String operator+(String strPart1, String strPart2);
```

```
int operator<(String str1, String str2);
int operator>(String str1, String str2);
int operator==(String str1, String str2);
int operator!=(String str1, String str2);
int operator<=(String str1, String str2);
int operator>=(String str1, String str2);
```

---

```
String operator+(String strPart1, String strPart2);
```

The "+"operator is the concatenation operator for strings.

```
String T(String strToTranslate);
String T(String strToTranslate, String str2LetterLangCode); // added hsbcad21.1.29 translate string
in any specific language
```

Strings can be translated by using the T() global function:

```
T("string")
```

When a string needs to be translated, the string is looked up in the translation table. The translation table is read from hsb map files. Multiple translation files are read:

HSB\_LANGUAGE\_ARX.MAP and HSB\_LANGUAGE\_MCR.MAP are found in the installation folder of hsbcad, in language specific subfolder in Lang.

Also loaded into the translation table are an additional number of translation map files, always called 'Hsb\_Language.map'. These are found in the following folders:

```
<hsbInstall>\Bonus, <hsbInstall>\Custom, <hsbCompany>\Lang,
<hsbCompany>\ElementStickFrame>\Lang
```

Each time the 'Hsb\_Language.map' is read in the root folder, and on top of that the language specific one is loaded.

The reloading of the translation files into memory can be triggered by the command:  
HSB\_REREADTABLE.

[Example of a translation file

```
# Comments start with '#'.
# After the '#' character anything can be written.
# Every line contains a combination "key ; value" with ';' the separator.
```

```
# The key and the value can contain the '|' character. This character is automatically
removed when the value is returned.
# White spaces (tab, space,...) before and after each key and value are removed while
loading the translation table.
# Lines without a ';' character are not interpreted.
```

# Every key must be unique. A second appearance of the same key will overwrite the first appearance.

# The key and value must not contain the ';' character.

# When a key is not found during translation, the key is returned as default value.

```
|zapf| ; tenon
—end example]
```

[Example any type:

```
PropString pStr1(0, "|Beam|", T("|String to translate|"));
PropString pLanguage1(1, "de", T("|Language|"));
pLanguage1.setDescription(T("|Language is indicated by 2 letter
code: eg 'de', 'ja', 'it', 'en'|"));

Display dp(-1);
dp.draw(T(pStr1), _Pt0, _XU, _YU, 0, 0);
dp.draw(T(pStr1, pLanguage1), _Pt0, _XU, _YU, 0, -2); // translate
in another language
dp.draw(T(pStr1, "abc"), _Pt0, _XU, _YU, 0, -4); // if language is
unknown, the string is not translated, but | are removed
—end example]
```

**String token(int nIndex) const;**

**String token(int nIndex, String strListOfSeparators) const;**

**String token(int nIndex, String strListOfSeparators, int bCountSequentialSeparatorsAsOne) const;**

If the string represents a list of elements, the token function can be used to find a particular element. The element is specified by its 0-based index. To find the nth element in a e.g. a comma separated list of elements, one has to call token(nth,""). The dot comma is the default separator.

**int length() const;**

Return the number of characters in the string.

**char getAt(int nIndex) const;**

Return the character at the 0-based index in the string.

**String spanIncluding(String strOther) const;**

Returns a substring from this string that contains characters in the string that are in *strOther*, beginning with the first character in this string and ending when a character is found in this string that is not in *strOther*. **SpanIncluding** returns an empty substring if the first character in this string is **not** in the specified set of *strOther*.

**String spanExcluding(String strOther) const;**

Returns a substring that contains characters in the string that are not in *strOther*, beginning with the first character in the string and ending with the first character found in the string that is also in *strOther* (that is, starting with the first character in the string and up to but **excluding** the first character in the string that is found *strOther*). It returns the entire string if no character in *strOther* is found in the string.

**String mid(int nFirst, int nCount) const;**

Extracts a substring of length *nCount* characters from this **String** object, starting at position *nFirst* (zero-based). The function returns a copy of the extracted substring. **Mid** is similar to the Basic MID\$ function (except that indexes are zero-based).

For multibyte character sets (MBCS), *nCount* refers to each 8-bit character; that is, a lead and trail byte in one multibyte character are counted as two characters.

**String left(int nCount) const;**

Extracts the first (that is, leftmost) *nCount* characters from this **String** object and returns a copy of the extracted substring. If *nCount* exceeds the string length, then the entire string is extracted. **Left** is similar to the Basic LEFT\$ function (except that indexes are zero-based).

**String right(int nCount) const;**

Extracts the last (that is, rightmost) *nCount* characters from this **String** object and returns a copy of the extracted substring. If *nCount* exceeds the string length, then the entire string is extracted. **Right** is similar to the Basic RIGHT\$ function (except that indexes are zero-based).

```
int find(String strToFind, int nFirst) const;
int find(String strToFind, int nFirst, int bCaseSensitive) const; // default bCaseSensitive is TRUE
(added v21.3.96 and v22.0.55)
```

Returns the zero-based index of the first character in this **String** object that matches the requested substring *strToFind*. Returns -1 if the substring or character is not found. *nFirst* is the index of the character in the string to begin the search with, or 0 to start from the beginning. If *bCaseSensitive* is FALSE, the lookup will not be case sensitive. By default, the lookup is case sensitive.

```
int atoi() const;
double atof() const;
double atof(int nFormat) const; // (added 17.0.42) The format can be _kLength (default), _kNoUnit,
                               _kArea, _kVolume, _kAngle
```

Returns the **double**, **int** value produced by interpreting the input characters as a number. The return value is 0, or 0.0 (for **atof**) if the input cannot be converted to a value of that type. The version with the *nFormat* uses the type to interpret the value. Architectural notation of imperial units is supported.

**String delete(int nFirst, int nCount);**

Call this member function to delete a character or characters from a string starting with the character at *nIndex*. If *nCount* is longer than the string, the remainder of the string will be removed.

```
String trimLeft();
String trimLeft(String strCharsToTrim);
```

Call the version of this member function with no parameters to trim leading whitespace characters from the string. When used with no parameters, **trimLeft** removes newline, space, and tab characters. Use the versions of this function that accept parameters to remove a particular character or a particular group of characters from the beginning of a string.

```
String trimRight();
String trimRight(String strCharsToTrim);
```

Call the version of this member function with no parameters to trim ending whitespace characters from the string. When used with no parameters, **trimRight** removes newline, space, and tab characters. Use the versions of this function that accept parameters to remove a particular character or a particular group of characters from the end of a string.

```
String makeUpper();
String makeLower();
```

Convert all characters in this string to uppercase or lowercase characters.

```
String format(String strFormat, int nV);
String format(String strFormat, double dV);
String format(String strFormat, char cV);
String format(String strFormat, String strV);
```

Call this member function to write formatted data to a **String** in the same way that **sprintf** formats data into a C-style character array. The *strFormat* is a string that contains some format specifications. A [format specification](#) always begin with a percent sign (%) and is read left to right. When a format specification (if any) is found, it converts the value of the argument after *strFormat* and outputs it accordingly.

```
String formatUnit(double dV, int nUnits, int nPrec);
String formatUnit(double dV, int nUnits, int nPrec, int nCuttOff); // added V23.8.61, V24.1.64,
V25.0.15
String formatUnit(double dV, String strDimStyle);
String formatUnit(double dV);
String formatUnit(double dV, int nFormat); // (added 17.0.42) The format can be _kLength (default),
_kNoUnit, _kArea, _kVolume, _kAngle
```

To format a distance into the proper string representation, architectural, inches,... one can use the **formatUnit** routine. This routine will use the scaling, the lunits, and precision of the dimension style to format the value, and turn it into a string. If the *strDimStyle* is not specified, the current dimstyle is used (*hsb\_settings*, *dimension*, *dimstyle* of *TSL*). Also the *LUNITS* and *PREC* value

can be specified directly. For the meaning of LUNITS, see autocad variable LUNITS (1 Scientific; 2 Decimal; 3 Engineering; 4 Architectural; 5 Fractional).

The version with the nFormat, does not use a dimstyle, but uses the settings specified in the AecDwgSetup command.

Regarding iCuttOff:

If iCuttOff is 0, the rounding will be done at 0.5 of the given precision.

If cutoff is +1, the rounding is done at 0.499123, and if the value is -1, the rounding is done at 0.500877. This supports consistent rounding of 0.5 value.

So icutoff == 1 promotes rounding up at the cutoff value, and -1 promotes rounding down.

For icutoff == 1, negative numbers round to bigger values, less negative.

```
String qrEncode(String strToEncode); // (added 21.3.101) default nMargin is 4
String qrEncode(String strToEncode, int nMargin); // (added 21.3.101) given a normal string, return
                                                 the QR:01010 representation of it. See also Display::drawQR.
String qrEncode(String strToEncode, int nMargin, int nErrorCorrection); // (added 23.4.1) default
                                                 nErrorCorrection 0
String qrEncode(String strToEncode, int nMargin, int nErrorCorrection, int
                                                 nAreaPercentSpaceForImage); // (added 23.4.1) default
                                                 nAreaPercentSpaceForImage 0
```

This method is used to convert a string into its QR:0010010100101101001 notation. This notation contains a 1 for each pixel in the qr code that is on.

Often a margin is required around the qr code, so this can be added as 0's to the string as well. (default Margin is 4 meaning 4 pixels around the qr code top, bottom, left and right).

The nErrorCorrection is a value that controls the redundancy of pixels in the qr code.

- 0 (default) means low redundancy: 7%

- 1 medium, 15%

- 2 quality: 25%

If nAreaPercentSpaceForImage is bigger than 0, and below or equal to 5, part of the pixels of the qr code will be turned to 0.

This area provides a space in the middle of the qr code, where another image can be put.

[Example any type:

```
reportMessage("\nThis message starts on a new line \nand continues on the next.");
String strBlockPath("c:\\Hsb-Company\\");
String strBlockFile = strBlockPath + "block" + ".dwg";
String strTranslated = T("Insert block name |"); // is included in the translation file of HSB
String strTranslated2 = T("zap!"); // might be in the HSB_LANGUAGE_MCR.MAP file
reportMessage(strTranslated + " " + strTranslated2);
```

—end example]

[Example O-type:

```
if (_bOnInsert){
    _Element.append(getElement("select wall with code A and number 003"));
```

```

        return;
    }

reportMessage("\nElement code " + _Element0.code() ); // output "A"
reportMessage("\nElement number " + _Element0.number()); // output "003"

char cZ = 'z';
String str = cZ + "aab" + _Element0.code() + _Element0.number() ; // becomes "zaabA003"
reportMessage("\n" + str); // output "zaabA003"
reportMessage("\n" + str.spanIncluding("az")); // output "zaa"
reportMessage("\n" + str.spanExcluding("0")); // output "zaabA"
reportMessage("\n" + str.mid(3,2)); // output "bA"
reportMessage("\n" + str.left(4)); // output "zaab"
reportMessage("\n" + str.right(4)); // output "A003"
reportMessage("\n" + str.length()); // output 8
reportMessage("\n" + str.charAt(3)); // output "b"

str.delete(2,1);
reportMessage("\n" + str); // output "zabA003"
str.makeLower();
reportMessage("\n" + str); // output "zaba003"
str.makeUpper();
reportMessage("\n" + str); // output "ZABA003"
str = str + " ";
reportMessage("\n" + str); // output " ZABA003 "
str.trimRight();
reportMessage("\n" + str); // output "ZABA003"
str.trimLeft("ABZ");
reportMessage("\n" + str); // output "003"

int ii = str.Atoi();
String str1; str1.format("This is the atoi result: %i.",ii);
reportMessage("\n" + str1); // output "This is the atoi result: 3."

double dd = str.Atof();
String str2; str2.format("This is the atof result: %.3f !",dd);
reportMessage("\n" + str2); // output "This is the atof result: 3.000 !."

```

—end example]

[Example O-type: *example that illustrates enhanced line input following the formatting specified in AecDwgSetup command.*

```

Unit(1,"inch"); // in this Tsl U(1) means 1 inch

// define a set of properties that can be used during insert, and
// that support storing the values into a catalog
PropDouble pLen(0,U(1),T("Length"));
PropDouble pAng(1,30,T("Angle"));
pAng.setFormat(_kAngle);
PropDouble pArea(2,U(1)*U(1),T("Area"));
pArea.setFormat(_kArea);

```

```

if (_bOnInsert)
{
    // set the property values from the catalog
    setPropValuesFromCatalog(T("_LastInserted"));

    {
        String strName = T("Length");
        String strVal; strVal.formatUnit(pLen, _kLength); // method
only available from 17.0.42 on
        String strPrompt; strPrompt.format(T("|set length value <%
s>|"),strVal);
        String strValNew=getString(strPrompt);
        strValNew.trimLeft();
        strValNew.trimRight();

        // keep existing value if empty or white string
        if (strValNew.length()!=0) {
            double dd;
            dd = strValNew.atof(_kLength); // method only
available from 17.0.42 on
            pLen.set(dd);
            reportMessage("\nNew value: "+pLen+ " for "+strName );
        }
        else {
            reportMessage("\nKept value: "+pLen+ " for "+strName );
        }
    }

    {
        String strName = T("Angle");
        String strVal; strVal.formatUnit(pAng, _kAngle); // method
only available from 17.0.42 on
        String strPrompt; strPrompt.format(T("|set angle value <%
s>|"),strVal);
        String strValNew=getString(strPrompt);
        strValNew.trimLeft();
        strValNew.trimRight();

        // keep existing value if empty or white string
        if (strValNew.length()!=0) {
            double dd;
            dd = strValNew.atof(_kAngle); // method only available
from 17.0.42 on
            pAng.set(dd);
            reportMessage("\nNew value: "+pAng+ " for "+strName );
        }
        else {
            reportMessage("\nKept value: "+pAng+ " for "+strName );
        }
    }
}

```

```

{
    String strName = T("Area");
    String strVal; strVal.formatUnit(pArea, _kArea); // method
only available from 17.0.42 on
    String strPrompt; strPrompt.format(T("|set area value <%
s>|"),strVal);
    String strValNew=getString(strPrompt);
    strValNew.trimLeft();
    strValNew.trimRight();

    // keep existing value if empty or white string
    if (strValNew.length()!=0) {
        double dd;
        dd = strValNew.atof(_kArea); // method only available
from 17.0.42 on
        pArea.set(dd);
        reportMessage("\nNew value: "+pArea+" for "+strName );
    }
    else {
        reportMessage("\nKept value: "+pArea+" for
"+strName );
    }
}

// save the property values to the catalog for next use
setCatalogFromPropValues(T("_LastInserted"));
}

```

—end example]

## 3.15 Format String

There are 2 types of format. The formatting of strings and numbers, which can be used in the `String::format` function, and the formatting of date and time which is used in the `String::formatTime`.

---

### Format of strings and numbers

A format specification, which consists of optional and required fields, has the following form:

`%[flags] [width] [.precision] type`

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a `type` character (for example, `%s`). If a percent sign is followed by a character that has no meaning as a format field, the character is outputted. For example, to print a percent-sign character, use `%%`.

The optional fields, which appear before the *type* character, control other aspects of the formatting, as follows:

#### *type*

Required character that determines whether the associated *argument* is interpreted as a character, a string, or a number.

Character	Type	Output Format
c	char	Specifies a single character.
i	int	Signed decimal integer.
e	double	Signed value having the form [ – ] <i>d</i> . <i>dddd</i> e [sign] <i>ddd</i> where <i>d</i> is a single decimal digit, <i>dddd</i> is one or more decimal digits, <i>ddd</i> is exactly three decimal digits, and sign is + or –.
E	double	Identical to the e format except that E rather than e introduces the exponent.
f	double	Signed value having the form [ – ] <i>ddd</i> . <i>ddd</i> , where <i>ddd</i> is one or more decimal digits. The number of digits before the decimal point depends on the magnitude of the number, and the number of digits after the decimal point depends on the requested precision.
g	double	Signed value printed in f or e format, whichever is more compact for the given value and precision. The e format is used only when the exponent of the value is less than –4 or greater than or equal to the precision argument. Trailing zeros are truncated, and the decimal point appears only if one or more digits follow it.
G	double	Identical to the g format, except that E, rather than e, introduces the exponent (where appropriate).
s	String	Specifies a character string.

#### *flags*

Optional character or characters that control justification of output and printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.

Flag	Meaning	Default
–	Left align the result within the given field width. The default is right align.	
+	Prefix the output value with a sign (+ or –) if the output value is of a signed type. The default is the sign appears only for negative signed values (–).	
0	If width is prefixed with 0, zeros are added until the minimum width is reached. If 0 and – appear, the 0 is ignored. If 0 is specified with an integer format (i) the 0 is ignored. No padding.	
blank (' ')	Prefix the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear. No blank appears.	
#	When used with the e, E, or f format, the # flag forces the output value to contain a decimal point in all cases. Decimal point appears only if digits follow it. When used with the g or G format, the # flag forces the output value to contain a decimal point in all cases and prevents the truncation of trailing zeros. Ignored when used with c, i or s. Decimal point appears only if digits follow it. Trailing zeros are truncated.	

#### *width*

Optional number that specifies the minimum number of characters output.

The second optional field of the format specification is the width specification. The *width* argument is a nonnegative decimal integer controlling the minimum number of characters printed. If the number of characters in the output value is less than the specified width, blanks are added to the left or the

right of the values — depending on whether the – flag (for left alignment) is specified — until the minimum width is reached. If *width* is prefixed with 0, zeros are added until the minimum width is reached (not useful for left-aligned numbers).

The width specification never causes a value to be truncated. If the number of characters in the output value is greater than the specified width, or if *width* is not given, all characters of the value are printed (subject to the [precision](#) specification).

If the width specification is an asterisk (\*), an [int](#) argument from the argument list supplies the value. The *width* argument must precede the value being formatted in the argument list. A nonexistent or small field width does not cause the truncation of a field; if the result of a conversion is wider than the field width, the field expands to contain the conversion result.

#### *precision*

Optional number that specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

Type	Meaning	Default
c	The precision has no effect. Character is printed.	
i	The precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than <i>precision</i> , the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds <i>precision</i> . Default precision is 1.	
e, E	The precision specifies the number of digits to be printed after the decimal point. The last printed digit is rounded. Default precision is 6; if <i>precision</i> is 0 or the period (.) appears without a number following it, no decimal point is printed.	
f	The precision value specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. Default precision is 6; if <i>precision</i> is 0, or if the period (.) appears without a number following it, no decimal point is printed.	
g, G	The precision specifies the maximum number of significant digits printed. Six significant digits are printed, with any trailing zeros truncated.	
s	The precision specifies the maximum number of characters to be printed. Characters in excess of <i>precision</i> are not printed. Characters are printed until a null character is encountered.	

---

#### Format of date and time

Formatting codes, preceded by a percent (%) sign, are replaced by the corresponding [time](#) component. Other characters in the formatting string are copied unchanged to the returned string.

E.g.:

"%B %d, %Y" will give "Januari 1, 2006"

"%x" will give the same, if my local date settings are set like that

"%H:%M" will give me 24:11 as time format

"%H:%M, %B %d, %Y" will give "24:11, Januari 1, 2006"

#### %a

Abbreviated weekday name

#### %A

Full weekday name

#### %b

<b>%B</b>	Abbreviated month name
<b>%C</b>	Full month name
<b>%d</b>	Date and time representation appropriate for locale
<b>%d</b>	Day of month as decimal number (01 – 31)
<b>%H</b>	Hour in 24-hour format (00 – 23)
<b>%I</b>	Hour in 12-hour format (01 – 12)
<b>%j</b>	Day of year as decimal number (001 – 366)
<b>%m</b>	Month as decimal number (01 – 12)
<b>%M</b>	Minute as decimal number (00 – 59)
<b>%p</b>	Current locale's A.M./P.M. indicator for 12-hour clock
<b>%S</b>	Second as decimal number (00 – 59)
<b>%U</b>	Week of year as decimal number, with Sunday as first day of week (00 – 53)
<b>%w</b>	Weekday as decimal number (0 – 6; Sunday is 0)
<b>%W</b>	Week of year as decimal number, with Monday as first day of week (00 – 53)
<b>%x</b>	Date representation for current locale
<b>%X</b>	Time representation for current locale
<b>%y</b>	Year without century, as decimal number (00 – 99)
<b>%Y</b>	Year with century, as decimal number
<b>%z, %Z</b>	Either the time-zone name or time zone abbreviation, depending on registry settings; no characters if time zone is unknown
<b>%%</b>	Percent sign

As in the previous format function, the # flag may prefix any formatting code. In that case, the meaning of the format code is changed as follows.

#### Format code and meaning

**%#a, %#A, %#b, %#B, %#p, %#X, %#z, %#Z, %#%:** # flag is ignored.

**%#c:** Long date and time representation, appropriate for current locale. For example: "Tuesday, March 14, 1995, 12:41:29".

**%#x:** Long date representation, appropriate to current locale. For example: "Tuesday, March 14, 1995".

**%#d, %#H, %#I, %#j, %#m, %#M, %#S, %#U, %#w, %#W, %#y, %#Y:** Remove leading zeros (if any).

## 3.16 Using var declaration

Since V23.0.104, and V22.1.108, Tsl supports the use of "auto" as keyword. It can be used to declare a variable with type inference. Tsl is a strongly typed language, meaning that the value that is stored against a variable needs to correspond to the type of the variable. Also methods and functions expect arguments to be of a certain type. The use of the keyword "auto" does not change that concept. The type of the variable is just inferred from the expression in the declaration.

[Example showing var keyword:

```
auto bb = 10; // integer
double dd = bb + 3;
auto cc = dd + 10; // double
auto ff = 12.3; // double
auto gg(ff); // double

auto ee[] = { 10, 20}; // array of integers
auto sum = ee[0] + ee[1]; // integer
for(int i=0; i<ee.length(); i++)
    reportMessage("\n" + ee[i]);

int anint = 12;
auto ass = anint = 4; // inferred from assignment as int
ass += sum;

auto arnd(anint); // integer
auto arnd2(anint=ass);
reportMessage("\n" + "arnd2=" + arnd2);

auto allnames = PainterDefinition().getAllEntryNames(); // array of
strings
for(int i=0; i<allnames.length(); i++)
    reportMessage("\n" + allnames[i]);

auto stree[] = { "str1", "str2"}; // array of strings
auto streee = stree; // array of strings
for(int i=0; i<streee.length(); i++)
    reportMessage("\n" + streee[i]);

Map pp;
auto mp2 = pp; // Map
mp2.setInt("int", 3);

auto map3 = Map();
map3.setDouble("dbl", 4, _kNoUnit);
```

—end sample]

---

## 3.17 Functions

Since V26.0.8 Tsl supports functions. It is possible to define a function in the tsl, and call it later on.

Functions have a name, similar to variables. The name should be different from any other function name within the same scope. If not a compile error appears. But inside an embedded scope a function can be redefined.

A function has a return value. The return values can be of type void indicating there is no return value. Pointers and references cannot be returned. But arrays can be returned, although not efficient. The return value is always copied back to the caller, so array cloning would be involved.

A function can have 0, 1 or multiple arguments. An argument can be passed by reference or by value. For arrays the efficient way is to pass by reference. Arguments cannot be functions.

From within the body of the function, global variables are visible. Global here means variables defined at the root level of the script, and not inside another function. So closure is possible, but limited to global variables.

*[Example showing functions:*

*—end sample]*

**Part**



**IV**

## 4 Geometric calculations

### 4.1 Point and Vector manipulation

The type `Point3d` represents a point in 3D space. `Vector3d` represents a direction, with a length in 3D space. Points are automatically casted into vectors, and vice versa. When a point is interpreted as a vector, the vector is the vector from the origin.

Points and vectors have a rich set of operators: points and vectors can be added, subtracted, both with other points and vectors. Also the `+=` and `-=` operators are applicable. Points and vectors can be multiplied and divided by scalars. Therefor the `*=` and `/=` with a double are also implemented.

---

```
class Point3d {
    Point3d(double x, double y, double z);
    Point3d(Point3d pt);
    Point3d(Vector3d vec); // automatic conversion of vector into point

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    normalize(); // normalize as vector
    setToAverage(Point3d[] arPnts); // set this point to the average point of the points in the array

    double length() const; // length of vector
    Vector3d normal() const; // return normalized vector with point seen as vector from the origin
    Point3d projectVector(Plane plProjectionPlane) const; // project as vector
    Point3d projectPoint(Plane plProjectionPlane, double dDistanceToPlane) const;
    Point3d projectPoint(Plane plProjectionPlane, double dDistanceToPlane, Vector
        vecDirection) const; // added v21.0.51 and v20.3.19

    double dotProduct(Point3d ptOther) const;
    Vector3d crossProduct(Point3d ptOther) const;

    Vector3d alignCoordSysX(Point3d ptOther) const;
    Vector3d alignCoordSysY(Point3d ptOther) const;
    Vector3d alignCoordSysZ(Point3d ptOther) const;

    // normal set of operations: +,-,*/,+=-, *=,/==
    double operator*(double dScalar);
    double operator/(double dScalar);
    Point3d operator*(double dScalar) const;
    Point3d operator/(double dScalar) const;
    double operator*(Point3d ptOther) const; // same as dotProduct
    Point3d operator+=(Point3d ptOther);
    Point3d operator-=(Point3d ptOther);
    Point3d operator+(Point3d ptOther) const;
```

---

```

Point3d operator-(Point3d ptOther) const;

// logical operations ==, !=
int operator==(Point3d ptOther) const;
int operator!=(Point3d ptOther) const;

// individual component access
double X() const;
double Y() const;
double Z() const;
double setX(double dComponentX);
double setY(double dComponentY);
double setZ(double dComponentZ);

};


```

---

```

class Vector3d {

    Vector3d(double x, double y, double z);
    Vector3d(Point3d pt); // automatic conversion of point into vector
    Vector3d(Vector3d vec);

    visualize(Point3d ptStart, int indColor = -1) const;
    vis(Point3d ptStart, int indColor = -1) const;

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    normalize(); // normalize as vector

    double length() const; // length of vector
    int blsZeroLength() const; // return TRUE if length is zero, using hsb tolerance value
    Vector3d normal() const; // return normalized vector, does not change this vector
    Vector3d projectVector(Plane plProjectionPlane) const; // project as vector
    Vector3d projectPoint(Plane plProjectionPlane, double dDistanceToPlane) const;

    double dotProduct(Vector3d vecOther) const; // see DotProduct revealed
    Vector3d crossProduct(Vector3d vecOther) const; // see CrossProduct revealed

    Vector3d alignCoordSysX(Vector3d vecOther) const;
    Vector3d alignCoordSysY(Vector3d vecOther) const;
    Vector3d alignCoordSysZ(Vector3d vecOther) const;

    double angleTo(Vector3d vecOther) const;
    double angleTo(Vector3d vecOther, Vector3d vecRef) const;

    Vector3d rotateBy(double dAngle, Vector3d vecAxis) const;

    int isPerpendicularTo(Vector3d vecOther) const;
    int isParallelTo(Vector3d vecOther) const;
    int isCodirectionalTo(Vector3d vecOther) const;
}


```

---

```

// normal set of operations: +,-,*/,+=,-=,*=/=
double operator*=(double dScalar);
double operator/=(double dScalar);
Vector3d operator*(double dScalar) const;
Vector3d operator/(double dScalar) const;
double operator*(Vector3d vecOther) const; // same as dotProduct
Vector3d operator+=(Vector3d vecOther);
Vector3d operator-=(Vector3d vecOther);
Vector3d operator+(Vector3d vecOther) const;
Vector3d operator-(Vector3d vecOther) const;

// logical operations ==, !=
int operator==(Vector3d vecOther) const;
int operator!=(Vector3d vecOther) const;

// individual component access
double X() const;
double Y() const;
double Z() const;
double setX(double dComponentX);
double setY(double dComponentY);
double setZ(double dComponentZ);

// filter an array of beams
Beam[] filterBeamsParallel(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsParallelUnique(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicular(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicularSort(Beam[] arBeamsToCheck) const;

};


```

---

**double dotProduct(Point3d ptOther) const;**

The dotProduct of 2 vectors.

**double angleTo(Vector3d vecOther) const;**  
**double angleTo(Vector3d vecOther, Vector3d vecRef) const;**

The first one returns the angle between this vector and the vector vecOther in the range [0,180] (in degrees).

The second variant returns the angle between this vector and the vector vecOther in the range [0,360].

If (vecRef.dotProduct(vecThis.crossProduct(vecOther)) >= 0.0), then the return value coincides with the return value of the function angleTo(vecOther). Otherwise the return value is 360 minus the return value of the function angleTo(vecOther).

---

**Vector3d rotateBy(double dAngle, Vector3d vecAxis) const;**

Returns 3D vector which is the result of rotation of this vector around the line with axis vecAxis, and passing through the origin. Rotation angle is given by the argument dAngle, where positive direction of rotation is defined by the right-hand rule.

```
int isPerpendicularTo(Vector3d vecOther) const;
int isParallelTo(Vector3d vecOther) const;
int isCodirectionalTo(Vector3d vecOther) const;
```

These functions can be used to test the direction of a vector and compared to another vector. Each of the functions above returns a 1 or a 0, a TRUE or a FALSE. The testing is done with a tolerance which is set, and also used by hsbCad internally.

*[Example X-type, testing the direction of beams:*

```
String strReport;

if (_X0.isPerpendicularTo(_X1)) strReport += "\n Beams are perpendicular";
else strReport += "\n Beams are NOT perpendicular";

if (_X0.isParallelTo(_X1)) strReport += "\n Beams are parallel";
else strReport += "\n Beams are NOT parallel";

if (_X0.isCodirectionalTo(_X1)) strReport += "\n Beams are codirectional";
else strReport += "\n Beams are NOT codirectional";

reportMessage(strReport);

—end example]
```

```
Beam[] filterBeamsParallel(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsParallelUnique(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicular(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicularSort(Beam[] arBeamsToCheck) const;
```

These routines have as argument an array of beams, and also as return value an array of beams. The array returned is a subset of the array passed in as argument. Each function will filter out those beams that have certain properties, in relation with the vector that the function is called upon.

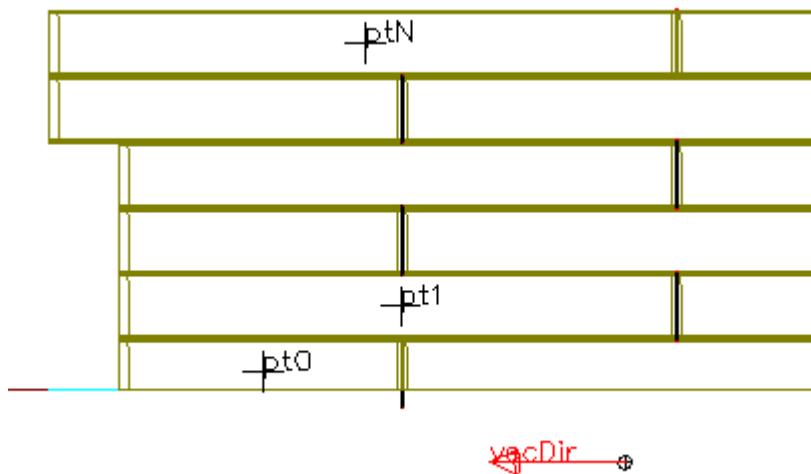
**filterBeamsParallel** returns the beams that are parallel with this vector.

**filterBeamsParallelUnique** returns the beams that are parallel with this vector, but remove beams that have their axes coinciding. The beam will be kept with the origin point most in the direction of this vector. Eg.: imagine the array of beams of a log wall, where the beams are split, calling this **filterBeamsParallelUnique**, will return a list of beams, with for each log, only one beam.

**filterBeamsPerpendicular** returns the beams that are perpendicular with this vector.

**filterBeamsPerpendicularSort** returns the beams that are perpendicular with this vector, and sort the list in the direction of this vector

*[Example O-type, ordering some beams in an element*



```

Unit(1,"mm");

if (_bOnInsert) {
    Element elSs = getElement();
    CoordSys csElSs = elSs.coordSys();
    _Pt0 = csElSs.ptOrg();
    _Element.append(elSs);
    return;
}

// check execute conditions
if (_Element.length()==0) return; // _Element[0] exists
Element el = _Element[0];
if (!el.blsValid()) return;

// coordsys, and envelope of the element
CoordSys csEl= el.coordSys();
Beam arBm[] = el.beam();

Vector3d vecDir = csEl.vecX();
arBm = vecDir.filterBeamsParallelUnique(arBm);
vecDir.vis(_Pt0,1);
vecDir = csEl.vecY();
arBm = vecDir.filterBeamsPerpendicularSort(arBm);

if (arBm.length()>1) { // 0,1 are valid index
    Point3d pt0 = arBm[0].ptCen(); pt0.vis();
    Point3d pt1 = arBm[1].ptCen(); pt1.vis();
    Point3d ptN = arBm[ arBm.length()-1 ].ptCen(); ptN.vis();
}
--end example]

setToAverage(Point3d[] arPnts);

```

Sets this point to the average point of the points in the array. The coordinates of this point are set to the normal average of the coordinates of the points in the array. If the array is empty, the point is set to the origin.

## 4.2 LineSeg

The LineSeg type represents a bounded straight line segment between 2 points in space. A LineSeg has a start and an end point.

---

```
class LineSeg {

    LineSeg(Point3d ptStart, Point3d ptEnd);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    Point3d closestPointTo(Point3d ptOther) const;
    Point3d closestPointTo(LineSeg segOther) const; // (added 15.1.31, 14.4.11)
    double distanceTo(Point3d ptOther) const; // (added 15.1.38, 14.4.11)
    double distanceTo(LineSeg segOther) const; // (added 15.1.38, 14.4.11)

    Point3d ptStart() const;
    Point3d ptEnd() const;
    Point3d ptMid() const;

    double length() const; // length of segment (added 15.1.39, 14.4.11)

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    static int[] findCapsuleIntersections(LineSeg[] arSeg, double dRadiusCapsule); // (added
    15.1.39, 14.4.11)
};
```

---

**Point3d closestPointTo(Point3d ptOther) const;**

This function can be applied on a line segment. It is used to find the closest point from another point to a line segment.

```
visualize(int indColor = -1) const;
vis(int indColor = -1) const;
```

In debug mode, the line segment can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

```
static int[] findCapsuleIntersections(LineSeg[] arSeg, double dRadiusCapsule); // (added 15.1.39,
14.4.11)
```

The method returns a list of indexes into the array arSeg. The indexes appear in couples. Each couple expresses the intersection of 2 capsules. These intersections were found by comparing the distance from each segment to each other segment in the list. If an intersection is found, the couple of indexes is added to the list.

The method is static, that means that you do not need an instance set in order to call it.

The list of indexes that the findCapsuleIntersections returns is calculated in the following way:

[*Sample*:

```
int arRet[0];
for (int i=0; i<arSeg.length(); i++) {
    for (int j=i+1; j<arSeg.length(); j++) {
        if (arSeg[i].distanceTo(arSeg[j])<=dRadiusCapsule) {
            arRet.append(i);
            arRet.append(j);
        }
    }
}
—end sample]
```

[*Example O-type*:

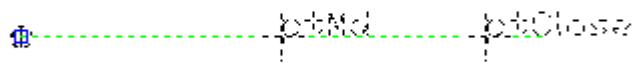
```
U(1,"mm");
Vector3d vecX = U(1000)*_XU;

// make a new LineSeg
LineSeg lseg(_Pt0, _Pt0+vecX);
lseg.vis(3); // just show in debug

// find closest point to a grippoint
Point3d ptClose = lseg.closestPointTo(_PtG[0]);
ptClose.vis();

// also show the midpoint
Point3d ptMd = lseg.ptMid();
ptMd.vis();
```

□



—end example]

## 4.3 Line and Plane

---

```
class Line {
    Line(Point3d ptOnLine, Vector3d vecDirection);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    int hasIntersection(Plane plIntersect) const; // return TRUE if intersect can be found
    int hasIntersection(Plane plIntersect, Point3d& ptIntersect) const; // return TRUE if intersect is
    // found, and set in ptIntersect.
    Point3d intersect(Plane plIntersect, double dDistanceNormalToPlane) const;

    Point3d closestPointTo(Point3d ptOther) const;
    Point3d closestPointTo(Line lnOther) const;

    Point3d ptOrg() const;
    Vector3d vecX() const;

    Point3d[] projectPoints(Point3d[] ptList) const;
    Point3d[] orderPoints(Point3d[] ptList) const;

    Point3d[] orderPoints(Point3d[] ptList, double dTol) const;

    Point3d[] filterClosePoints(Point3d[] ptList) const;
    Point3d[] filterClosePoints(Point3d[] ptList, double dDistMax) const;

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;
};
```

---

```

class Plane {

    Plane(Point3d ptOnPlane, Vector3d vecNormalToPlane);
    Plane(Point3d ptOnPlane, Point3d ptOnPlane, Point3d ptOnPlane);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    int hasIntersection(Plane plIntersect) const; // return TRUE if intersect can be found
    Line intersect(Plane plIntersect) const;

    Point3d ptOrg() const;
    Vector3d vecX() const;
    Vector3d vecY() const;
    Vector3d vecZ() const;
    Vector3d normal() const; // alias for vecZ

    Point3d closestPointTo(Point3d ptOther) const;

    Point3d[] projectPoints(Point3d[] ptList) const;
    LineSeg[] projectLineSegs(LineSeg[] segList) const; // similar to projectPoints

    Point3d[] filterClosePoints(Point3d[] ptList) const;
    Point3d[] filterClosePoints(Point3d[] ptList, double dDistMax) const;

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;
};


```

```

Point3d closestPointTo(Point3d ptOther) const;
Point3d closestPointTo(Line lnOther) const;

```

This function can be applied on a line. It is used to find the closest point from another point to a line, or the closest point from another line.

In the example below, the grip-point `_PtG[0]` is moved to its closest point on the axis of the beam `_Beam0`.

*[Example:*

```

// this script needs one extra grip/insert point: _PtG[0]
Line ln(_Pt0,_X0); // for this definition of ln we could have used the _L0 predefined
Point3d pt = ln.closestPointTo(_PtG[0]);

// move the choosen grip-point onto the axis of the beam:
_PtG[0] = pt;

--end example]

```

So the above example could have been written more compact as:

*[Example:*

```
_PtG[0] = _L0.closestPointTo(_PtG[0]);
—end example]
```

```
Point3d ptOrg() const;
Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;
```

Retrieve individual components. For the line, the point and the vecX are returned that were used in the constructor. For the plane, the vecZ is the normal to the plane, used in the constructor. For the Plane, the vecX and vecY will return a set of perpendicular vector in the plane.

```
visualize(int indColor = -1) const;
vis(int indColor = -1) const;
```

In debug mode, the line and the plane can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

```
Point3d[] Line::projectPoints(Point3d[] ptList) const;
Point3d[] Plane::projectPoints(Point3d[] ptList) const;
```

The projectPoints returns a new list of points. The returned points lie on the line (or plane). In case of the plane, the points are parallel projected normal to the plane. In case of the line, the points are projected onto the line, perpendicular to the line. The order of points is not changed and the length of the array that is returned is the same the length of the argument array.

```
Point3d[] Line::orderPoints(Point3d[] ptList) const;
Point3d[] Line::orderPoints(Point3d[] ptList, double dTol) const;
```

The orderPoints returns a new list of points. The returned points are ordered with respect to the line direction from beginning to end. Also duplicate points are removed so the size of the array returned, might be smaller than the array size of the argument. The second optional argument is the distance/tolerance that is used in the point comparison. The distance calculation of the points is done with the original points, not with the points projected onto the line.

```
Point3d[] Line::filterClosePoints(Point3d[] ptList) const;
Point3d[] Line::filterClosePoints(Point3d[] ptList, double dDistMax) const;
Point3d[] Plane::filterClosePoints(Point3d[] ptList) const;
Point3d[] Plane::filterClosePoints(Point3d[] ptList, double dDistMax) const;
```

The filterClosePoints returns a new list of points. The returned points lie at a distance equal or closer to line (or plane) than the given dDistMax. The points are not sorted.

```
int Line::hasIntersection(Plane plIntersect, Point3d& ptIntersect) const;
```

The hasIntersection on Line will return TRUE if the intersection is found, otherwise it will return FALSE. In case TRUE is returned, then ptIntersect is set to the intersection point.

---

[Example O-type:

```

Vector3d vec1 = _XU;
Point3d pt1 = _Pt0;
Line ln1(_Pt0, _XU);
Plane pl1(_Pt0, _XU);
pl1.vis();

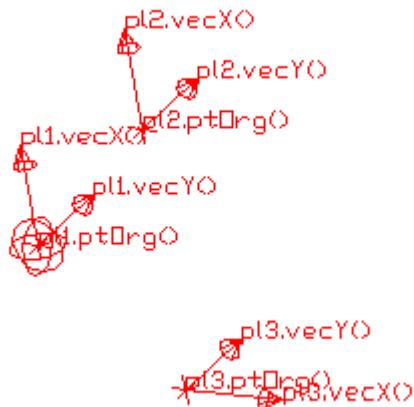
CoordSys cs1(_Pt0, _XU, _YU, _ZU);
CoordSys cs2, cs3;
cs3.setToRotation(-90, _ZU, _Pt0);

Point3d pt2 = pt1;
Vector3d vec2 = vec1;
Line ln2 = ln1;
Plane pl2 = pl1;
Vector3d vecTranslation = 100* _XU + 100* _YU;
if (FALSE) { // 2 equivalent alternatives
    cs2.setToTranslation(vecTranslation);
    pt2.transformBy(cs2);
    vec2.transformBy(cs2);
    ln2.transformBy(cs2);
    pl2.transformBy(cs2);
}
else {
    pt2.transformBy(vecTranslation);
    vec2.transformBy(vecTranslation);
    ln2.transformBy(vecTranslation);
    pl2.transformBy(vecTranslation);
}
pl2.vis();

Point3d pt3 = pt2;
Vector3d vec3 = vec2;
Line ln3 = ln2;
Plane pl3 = pl2;
pt3.transformBy(cs3);
vec3.transformBy(cs3);
ln3.transformBy(cs3);
pl3.transformBy(cs3);
pl3.vis();

```

---



*—end example]*

## 4.4 PLine

A PLine represents the type that describes the 3D planar polyline. A polyline connects 3d points by either line or arc segments. All points of the polyline must lie in the same plane. Points can be added to an existing polyline. When adding a point, the bulge factor can be specified, that defines, when different from zero, the arc segment. In order to build a closed polyline, the first point should be appended as last point.

---

```
class PLine {

    PLine(Vector3d vecNormal);
    PLine(Point3d pt1, Point3d pt2);
    PLine(Point3d pt1, Point3d pt2, Point3d pt3);
    PLine(Point3d pt1, Point3d pt2, Point3d pt3, Point3d pt4);

    addVertex(Point3d ptToAdd);
    addVertex(Point3d ptToAdd, double dBulge);
    addVertex(Point3d ptToAdd, double dRadius, int bClockWise);
    addVertex(Point3d ptToAdd, double dRadius, int bClockWise, int bSmall);
    addVertex(Point3d ptToAdd, Point3d ptOnArc);
    addVertex(double dBulge);

    Point3d closestPointTo(Point3d pt) const;

    Point3d[] vertexPoints(int bOnlyOnce) const;
    int isOn(Point3d pt) const;

    double getDistAtPoint(Point3d pt) const;
}
```

---

```
Point3d getPointAtDist(double dDist) const;
Vector3d getTangentAtPoint(Point3d pt) const; // added since v21.4.14

void transformBy(CoordSys csTransformationMatrix);
void transformBy(Vector3d vecTranslate);

void convertToLineApprox(double dAccuracy);
void projectPointsToPlane(Plane plane, Vector vecDir);
void reverse();
void close(double dBulge);
void close();
void setNormal(Vector3d vecNormal); // only changes normal, use flipNormal to correct bulges
as well.
void flipNormal(); // change normal, and adjust bulges (added v21.1.46)
void simplify(); // removes colinear segments (added V24.1.91, V25.1.64)
int reconstructArcs(double dMaxDeviationAtEndPoints, double
dMaxAngleBetweenSmoothConsecutiveArcSegments); // (added V25.1.72) returns
success of the operation

int offset(double dOffset); // positive offset on left side (added v21.4.65, v22.0.105), fillet corner
type. Return TRUE if successful.
int offset(double dOffset, int bFillet); // positive offset on left side (added v21.4.66, v22.0.106).
Return TRUE if successful.

double area() const; // returns the area in drawing units squared, divide by U(1)*U(1) to have it in
script units
double length() const; // returns the length in drawing units

Point3d[] intersectPoints(Point3d ptLine, Vector vecLine) const;
Point3d[] intersectPoints(Line line) const;
Point3d[] intersectPoints(Plane plane) const;

Point3d[] intersectPLine(PLine plOther) const; // PLine intersections (added hsbCAD2010 build
15.3.15 and hsbCAD2011 build 16.0.31)
double[] intersectPLineAsDistances(PLine plOther) const; // PLine intersections, but returned
as distances from the start of this PLine. Result is sorted from start to end on this
PLine..(added V24.1.90, V25.1.62)
int extend(double dExtendLength, int bSecondOrder); // return success of the operation (added
hsbCAD2010 build 15.3.15 and hsbCAD2011 build 16.0.31)
int append(PLine plOther); // append plOther to this pline. A missing straight line segment
might be added. If plOther is not in the same plane, the points are projected into the
plane. (added v21.4.66, v22.0.106). Return TRUE if successful.

visualize(int indColor = -1) const;
vis(int indColor = -1) const;

Point3d ptStart() const;
Point3d ptMid() const;
Point3d ptEnd() const;

CoordSys coordSys() const;
```

```

int trim(double dTrimDistance, int bAtEnd); // added since 22.1.54 and 23.0.36. Change this
    PLine by cutting of part of it, at the end or at the start. Return success of the
        operation

int isClosed() const; // added since 24.1.49. Return TRUE if PLine is closed, else FALSE.
int isSelfIntersecting() const; // added since 24.1.49. Return TRUE if PLine is self
    intersecting, else FALSE.
int isCircle(Point3d& ptCenter, Vector3d& vecNormal, double& radius) const;// added since
    26.5.5. Return TRUE if it is a circle in which case the arguments are set.
int isCircle() const;// added since 26.5.5. Return TRUE if it is a circle.

void createCircle(Point3d ptCen, Vector3d vecNormal, double dRadius);
void createRectangle(LineSeg segDiagonal, Vector3d vecX, Vector3d vecY);
void createConvexHull(Plane plane, Point3d arPnts[]); // (added hsbCAD2009+ build 14.0.30)
void createSmoothArcsApproximation(Plane plane, Point3d arPnts[], double
    dMaximumDeviation); // (added hsbCAD2015 build 20.0.6)
};

```

---

**PLine(Vector3d vecNormal);**

Constructing a new polyline with a given normal. The normal is needed if arc segments are specified by bulge factors.

```

PLine(Point3d pt1, Point3d pt2);
PLine(Point3d pt1, Point3d pt2, Point3d pt3);
PLine(Point3d pt1, Point3d pt2, Point3d pt3, Point3d pt4);

```

Constructing a new polyline with line segments given 2, 3 or 4 points.

```
Point3d[] vertexPoints(int bOnlyOnce) const;
```

Return the list of vertex points as an array of points. If the flag bOnlyOnce is TRUE or 1, the list will contain each point only once.  
If the bOnlyOnce is FALSE or 0, the last vertex is added, independent if it is equal to the first vertex.

```
int isOn(Point3d pt) const;
```

Find out if a point lies on the polyline.

```

addVertex(Point3d ptToAdd);
addVertex(Point3d ptToAdd, double dBulge);
addVertex(Point3d ptToAdd, double dRadius, int bClockWise);
addVertex(Point3d ptToAdd, double dRadius, int bClockWise, int bSmall);
addVertex(Point3d ptToAdd, Point3d ptOnArc);
addVertex(double dBulge);

```

Add another point to the polyline. If a bulge is specified that is different from zero, an arc segment is added. To close the polyline with an arcsegment, the addVertex(bulge) must be used. The bulge at a vertex is the tangent of 1/4 of the included angle for the arc between the newly added vertex and the previous vertex in the polyline's vertex list. A negative bulge value indicates that the arc goes clockwise from the previous vertex to the newly added vertex. Clockwise or counter clockwise is interpreted looking down to the polyline from the vecNormal direction.

If the vertex is added with a dRadius, there are actual 4 possibilities: clockwise or counter clockwise, but also with a big circle or a small circle. In case the bSmall is not specified, it has a default value of TRUE, taking the small circle solution.

In order for the bulges to make sense, the PLine must have a normal defined.

If the vertex is added with another point ptOnArc, the bulge is calculated such that the arc will run through the ptOnArc. So the arc is defined by 3 points: the previous added vertex, the new vertex and the point on the arc.

For the bClockWise, the following predefines can be used: `_kCWise` (=TRUE) and `_kCCWise` (=FALSE).

```
close();
close(double dBulge);
```

Closing a polyline is the same as adding the first point as last point. If you close the PLine with a bulge different from zero, the last segment is curved.

```
convertToLineApprox(double dAccuracy);
```

This routine changes the PLine itself.

Convert the polyline into a polyline that only contains straight line segments. If the original polyline contains curved segments, the curved segments are approximated by a set of straight line segments, with a deviation of the original curve not bigger than dAccuracy. If the pline already consists of only straight lines, nothing is done.

```
projectPointsToPlane(Plane plane, Vector vecDir);
```

This routine changes the PLine itself.

Parallel project the points of the polyline into the plane. The direction of projection is defined by vector vecDir. Also the normal of the PLine will be set to be parallel with the normal of the plane. The bulge factors however will not be adjusted.

```
reverse();
```

This routine changes the PLine itself.

Reverse the direction of the polyline. The start point will become the end point.

```
Point3d[] intersectPoints(Point3d ptLine, Vector vecLine) const;
Point3d[] intersectPoints(Line line) const;
Point3d[] intersectPoints(Plane plane) const;
```

The intersectPoints return an array of points that lie on the PLine. The points also lie on a plane. The plane is either given or calculated from a line. When the line is given as argument, the plane contains both that line, and the normal of the PLine. The line can also be given through a point and a vector vecLine. In this case the plane that defines the intersection points contains the point ptLine, the vector vecLine and the normal of the PLine. The PLine must have a normal set for this function to work with a line.

The points that are returned, lie on the PLine, but are ordered along the intersection line. If more than 2 intersection points are found, the first and the last point will be the extreme points along the line.

For example see [Element](#).

```
visualize(int indColor = -1) const;
vis(int indColor = -1) const;
```

In debug mode, the PLine can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

```
Point3d ptStart() const;
Point3d ptMid() const;
Point3d ptEnd() const;
```

Return the start, middle and end point of the polyline.

```
CoordSys coordSys() const;
```

The coordSys of a PLine returns the internal coordinate transformation from the XY plane to the WCS. Therefor, the vecX and vecY of the coordSys define the plane in which the PLine has its points. The vecZ of the coordSys is also the normal of the PLine.

```
int extend(double dExtendLength, int bSecondOrder); // return success of the operation (added  
hsbCAD2010 build 15.3.15 and hsbCAD2011 build 16.0.31)
```

Extend the current PLine with the dExtendLength at start and end. The extensions are added tangential to start and end segment of the current pline. If bSecondOrder is TRUE, and the existing end segment is an arc, then the extension is also an arc. Returns success of the operation.

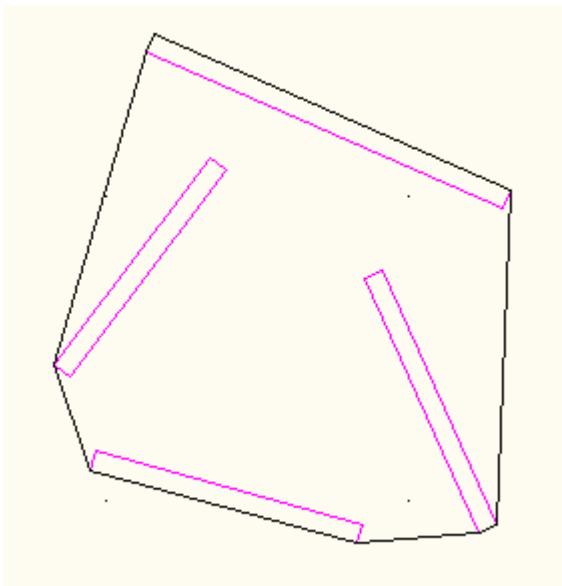
```
void createRectangle(LineSeg segDiagonal, Vector3d vecX, Vector3d vecY)
```

Create a rectangular PLine. The corner point from which is started is the start point of the segment. Then the segment is projected onto the vecX direction to find the second point. The third point is found by projecting the segment on the vecY. So the segment does not need to be in the plane of vecX, vecY.

```
void createConvexHull(Plane plane, Point3d arPnts[]) // (added hsbCAD2009+ build 14.0.30)
```

Create a PLine with straight line segments of which the vertices are projected points of the arPnts onto the plane, and that contains all other vertices.

*[Example O-type to illustrate the convex hull calculation:*



```
if (_bOnInsert) {
    PrEntity ssE(T("|Select a set of beams|"), Beam());
    if (ssE.go()) {
        Beam ssBeams[] = ssE.beamSet();
        reportMessage (T("\n|Number of beams selected:| ") +
ssBeams.length());
        _Beam = ssBeams;
    }
    _Pt0 = getPoint();
    return;
}

// make sure the tsl fires when the length of some beams changes
for (int b=0; b<_Beam.length(); b++) {
    Beam bm = _Beam[b];
    setDependencyOnBeamLength(bm);
}

// collect a list of all points of the beams
Point3d pntsAll[0];
for (int b=0; b<_Beam.length(); b++) {
    Beam bm = _Beam[b];
    Point3d pnts[] = bm.envelopeBody(FALSE, TRUE).allVertices();
    for (int p=0; p<pnts.length(); p++) {
        pntsAll.append(pnts[p]);
    }
}

// calculate the pline of the convex hull
Plane plane(_Pt0,_ZU);
PLine pline;
pline.createConvexHull(plane,pntsAll);
```

```
// display the pline
Display dp(-1);
dp.draw(pline);

—end example]
```

**void** createSmoothArcsApproximation(**Plane** plane, **Point3d** arPnts[], **double** dMaximumDeviation); // (added hsbCAD2015 build 20.0.6)

Create a PLine with arc segments of which the tangent is continuous in its vertices, and that does not deviate more than dMaximumDeviation from the given set of points arPnts.

[Example O-type to illustrate the smooth arc approximation calculation. Set the "number of extra grips to eg 3, and not insert done in script"

```
Unit(1, "mm");

PropDouble pMaxDev(0, U(5), T("|Maximum deviation|"));

Point3d pts[0];
pts.append(_Pt0);
for (int i=0; i<_PtG.length(); i++) pts.append(_PtG[i]);

Plane plane(_Pt0, _ZW);

PLine pline;
pline.createSmoothArcsApproximation(plane, pts, pMaxDev);

Point3d ptVertices[] = pline.vertexPoints(FALSE);
reportMessage("\n"+scriptName()+"": number of vertices =
"+ptVertices.length());
for(int p=0;p<ptVertices.length(); p++)
{
    Point3d pt = ptVertices[p];
    pt.vis();
}

Display dp(-1);
dp.draw(pline);

String strChangeEntity = T("|Append pline|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    EntPLine() .dbCreate(pline);
}
```

—end example]

---

[Example O-type:

**U**(1,"mm");

```

// select 2 elements
if (_bOnInsert){
    _Element.append(getElement());
    _Element.append(getElement());
    return;
}

// do nothing if only one element available
if (_Element.length()<2) return; // 0 and 1 are valid indices

// get outline in planview of wall 0 and 1
PLine pl0 = _Element[0].plOutlineWall();
PLine pl1 = _Element[1].plOutlineWall();

// collect points of polyline 0
Point3d pntsPl0[] = pl0.vertexPoints(TRUE);

// collect the points of pl2 which are on pl1
Point3d pntsCommon[0];
for (int i=0; i<pntsPl0.length(); i++) {
    if (pl1.isOn(pntsPl0[i])) pntsCommon.append(pntsPl0[i]);
}

reportMessage("\nNumber of points found: " + pntsCommon.length());

```

—end example]

[Example O-type with 1 extra grippoint:

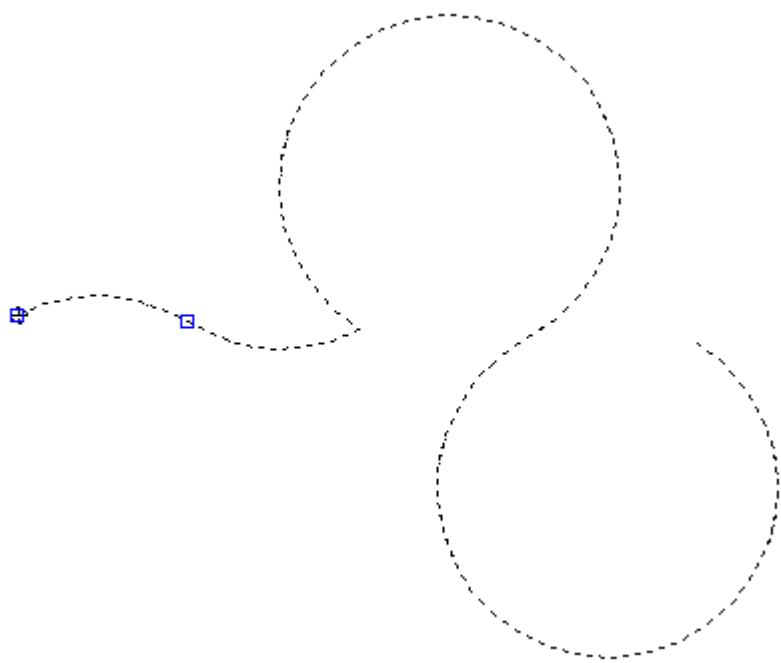
```

Point3d pt1 = _Pt0;
Point3d pt2 = _PtG[0];
Vector3d vec = pt2-pt1;
Point3d pt3 = pt2+vec;
Point3d pt4 = pt3+vec;
Point3d pt5 = pt4+vec;

double dDist = vec.length();
double dRadius = dDist;
int bBig = 0;

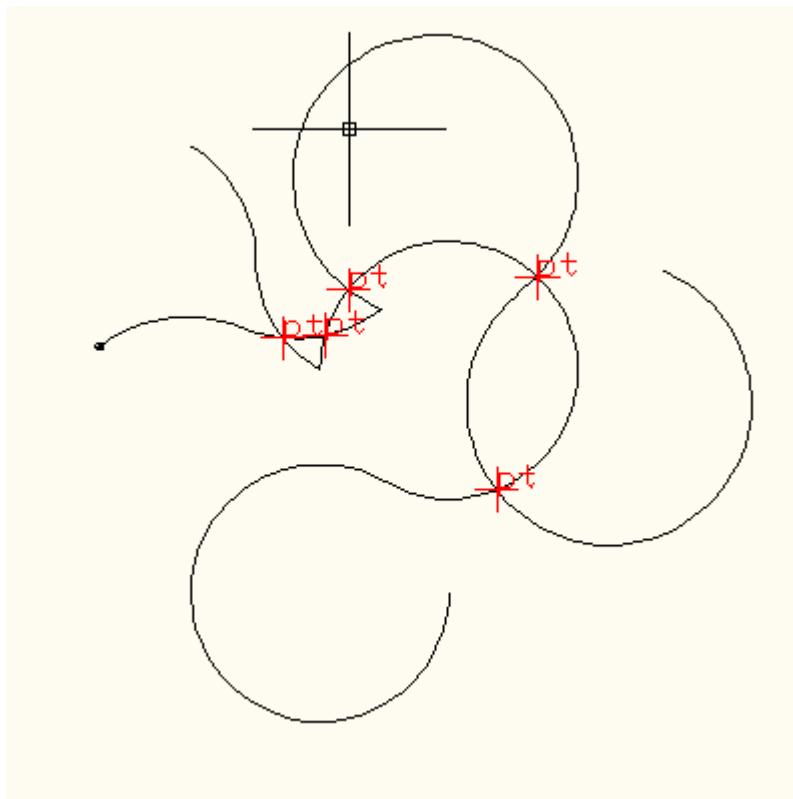
PLine pline(_ZW); // define PLine normal
pline.addVertex(pt1); // add first point
pline.addVertex(pt2,dRadius ,_kCWise);
pline.addVertex(pt3,dRadius ,_kCCWise);
pline.addVertex(pt4,dRadius ,_kCWise,bBig);
pline.addVertex(pt5,dRadius ,_kCCWise,bBig);
pline.vis();

```



*—end example]*

*[Example O-type with 3 extra grip points running in debug:*



```
PLine poly1, poly2;
{
    Point3d pt1 = _Pt0;
    Point3d pt2 = _PtG[0];
    Vector3d vec = pt2-pt1;
    Point3d pt3 = pt2+vec;
    Point3d pt4 = pt3+vec;
    Point3d pt5 = pt4+vec;

    double dDist = vec.length();
    double dRadius = dDist;
    int bBig = 0;

    PLine pline(_ZW); // define PLine normal
    pline.addVertex(pt1); // add first point
    pline.addVertex(pt2,dRadius ,_kCWise);
    pline.addVertex(pt3,dRadius ,_kCCWise);
    pline.addVertex(pt4,dRadius ,_kCWise,bBig);
    pline.addVertex(pt5,dRadius ,_kCCWise,bBig);
    pline.vis();
    poly1 = pline;
}

////////////////////

{
    Point3d pt1 = _PtG[1];
    Point3d pt2 = _PtG[2];
    Vector3d vec = pt2-pt1;
    Point3d pt3 = pt2+vec;
    Point3d pt4 = pt3+vec;
    Point3d pt5 = pt4+vec;

    double dDist = vec.length();
    double dRadius = dDist;
    int bBig = 0;

    PLine pline(_ZW); // define PLine normal
    pline.addVertex(pt1); // add first point
    pline.addVertex(pt2,dRadius ,_kCWise);
    pline.addVertex(pt3,dRadius ,_kCCWise);
    pline.addVertex(pt4,dRadius ,_kCWise,bBig);
    pline.addVertex(pt5,dRadius ,_kCCWise,bBig);
    pline.vis();
    poly2 = pline;
}

////////////////////

Point3d pnts[] = poly1.intersectPLine(poly2);
```

```

reportMessage("\nNumber of intersections found: "+pnts.length());
for (int p=0; p<pnts.length(); p++) {
    Point3d pt = pnts[p];
    pt.vis(1);
}

```

*—end example]*

## 4.5 PlaneProfile

A PlaneProfile represents the type that describes a list of closed 3D planar poly lines. These closed poly lines are often called rings. Any ring of the collection of rings, is not intersecting with any of the other rings. When adding a ring to an existing PlaneProfile, the rings are joined. If 2 or more rings are part of the definition of the Profile, one ring does not have to be contained within the other. The PlaneProfile is defined in the 3d space, that is, it is a 2 dimensional shape defined in on a plane in space. At time of construction, the plane is defined. Rings that are added later are transformed, and the defining points are projected onto the plane. If the normal's are not co directional, curved PLines are not projected correctly. Circles become ellipses, which cannot be represented by bulges.

When checking for the position of a point the return type can be one of the following:

<u>_kPointInProfile</u> == 0: <u>_kPointOutsideProfile</u> == 1: <u>_kPointOnRing</u> == 2:	point is clearly inside the profile, but not inside one of the openings/voids point is outside of the profile, or inside one of the openings/voids point is on one of the rings, outer or opening ring
---	--

---

```

class PlaneProfile {

    PlaneProfile();
    PlaneProfile(PLine plOuter);
    PlaneProfile(Plane plane);
    PlaneProfile(CoordSys csToDefinePlane);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    void removeAllRings();
    void removeAllOpeningRings(); // (added in build 21.3.98 and v22.0.55)

    int blsValid() const; // will return FALSE if the PlaneProfile is self intersecting. (added in build 24.1.68 and 25.1.20)

    int joinRing(PLine plRing, int bSubtract); // automatically repairs self intersecting
    int joinRing(PLine plRing, int bSubtract, int bRepairSelfIntersecting); // (added in build 24.1.68 and 25.1.20)
}

```

---

```

int shrink(double dShrinkDistance);
int unionWith(PlaneProfile prof);
int intersectWith(PlaneProfile prof);
int subtractProfile(PlaneProfile prof);

int project(Plane planeToProjectOn, Vector3d vecProjectDirection, double dAccuracy); //  

(added hsbCAD2009+ build 14.0.73)

int numRings() const;
int[] ringsOpening() const;
PLine[] allRings() const;
PLine[] allRings(int bIncludeNoneOpenings, int bIncludeOpenings) const; // (added in build  

21.3.98 and v22.0.55)

Point3d closestPointTo(Point3d pt) const;
int pointInProfile(Point3d pt) const;

CoordSys coordSys() const;

LineSeg extentInDir(Vector3d vecDir) const;

double area() const; // returns the area in drawing units squared, divide by U(1)*U(1) to have it in  

script units

Point3d[] getGripEdgeMidPoints() const;
Point3d[] getGripVertexPoints() const;
int moveGripEdgeMidPointAt(int nIndex, Vector3d vecMove);
int moveGripVertexPointAt(int nIndex, Vector3d vecMove);

LineSeg[] splitSegments(LineSeg segToSplit, int bKeepInside) const; // (added in build  

18.2.15, and 19.0.23) (see example in Element::noNailProfile)
LineSeg[] splitSegments(LineSeg[] segsToSplit, int bKeepInside) const; // (added in build  

18.2.15, and 19.0.23)

PLine[] splitPLine(PLine plToSplit, int bKeepInside, int bKeepOn) const; // (added V24.1.90,  

V25.1.62)

void simplify(); // removes colinear segments (added V24.1.91, V25.1.64)

void createRectangle(LineSeg segDiagonal, Vector3d vecX, Vector3d vecY); // (added in  

build 21.3.98 and v22.0.55) normal is always vecX.crossProduct(vecY)  

direction.

// following 4 methods added in build 22.1.70 and v23.0.43
Point3d[] intersectPoints(Plane plane, int bIncludeNoneOpenings, int bIncludeOpenings)  

const;
Point3d ptMid() const; // returns the center point of the enclosing rectangle in the  

coordSys.vecX and vecY directions.
double dX() const; // returns the dimension of the enclosing rectangle in vecX direction
double dY() const; // returns the dimension of the enclosing rectangle in vecY direction
};

```

```
PlaneProfile();
PlaneProfile(PLine plOuter);
PlaneProfile(Plane plane);
PlaneProfile(CoordSys csToDefinePlane);
```

Constructing a new PlaneProfile from an outer PLine. Internally the ring is stored with a 2d representation and a coordinate transformation. This results in a planar geometry, whatever operations are done upon it. The coordinate transformation is taken from the PLine, transforming the plane of the PLine to the world X-Y plane.

When the default constructor is used, the transformation matrix is the identity matrix.

The transformation matrix can also be calculated from the plane, or directly given by csToDefinePlane.

```
void removeAllRings();
```

To remove all the rings, the removeAllRings routine can be used. This function is handy if you want to keep the coordinate system of an existing plane profile, but want to redefine the shape.

```
int joinRing(PLine plRing, int bSubtract);
```

To add an opening/ring to the PlaneProfile, the joinRing routine should be used. If the bSubtract is TRUE, the plRing is added as an opening. If the bSubtract is FALSE, the plRing is added to the PlaneProfile as a full ring.

If the plRing is not closed, it will be closed to form a ring. If the ring is self intersecting, the routine will try to change the plRing such that it is not self intersecting anymore, before adding it to the PlaneProfile. If this is not possible, or if the ring is not valid for some other reason, it will not be joined (and FALSE will be returned).

The points of the plRing will be projected automatically onto the plane that originally defined the PlaneProfile. If the normal's are not parallel, curved PLines are not projected correctly. Circles become ellipses, which cannot be represented by bulges.

The success of the operation, TRUE or FALSE is returned.

```
int shrink(double dShrinkDistance);
```

To offset the PlaneProfile with in a uniform manner, one can use the shrink routine. Shrinking with a positive value, will decrease the area of the PlaneProfile, shrinking with a negative dShrinkDistance, will increase the area. The success of the operation, TRUE or FALSE is returned.

```
int unionWith(PlaneProfile prof);
int intersectWith(PlaneProfile prof);
int subtractProfile(PlaneProfile prof);
```

The unionWith, intersectWith and subtractProfile use as argument another PlaneProfile. The argument is not modified. To perform the operation, unite, intersect or subtract, the PlaneProfile passed in is first transformed to the correct location in space, being the plane of this PlaneProfile. If the PlaneProfile argument is not in the same plane, it is projected into this plane.

Beware, curved profiles will not always be projected correctly since the projection of a circle segment is not always a circle segment. On the projected profile, the operation is performed. The success of the operation, TRUE or FALSE is returned.

```
visualize(int indColor = -1) const;
vis(int indColor = -1) const;
```

In debug mode, the PlaneProfile can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

```
PLine[] allRings() const;
int numRings() const;
```

Return a list of all the rings with the routine allRings. To only get the number of rings, just call the numRings routine.

```
int[] ringIsOpening() const;
```

This routine returns an array of integers, with value either TRUE (1) or FALSE (0). If array length is equal to numRings(). The value TRUE means that the corresponding PLine in the allRings array is subtracted in the generation process of the PlaneProfile. So recomposing a PlaneProfile from the rings given by allRings, one needs to know for each ring whether to subtract or add it to the new PlaneProfile. The ringIsOpening returns this information.

```
Point3d closestPointTo(Point3d pt) const;
int pointInProfile(Point3d pt) const;
```

For both the closestPointTo and the pointInProfile, the given point pt is first projected into the plane of the PlaneProfile. Then either the closest point is searched for, or the check is done if the point is inside or outside the profile. So even if the point is completely outside the plane of the PlaneProfile, the pointInProfile routine might return \_kPointOnRing. The pointInProfile will return one of the following:

- |   |  |
|---|--|
| <u>_kPointInProfile</u> == 0:<br><u>_kPointOutsideProfile</u> == 1:<br><u>_kPointOnRing</u> == 2: | point is inside the profile, but not inside one of the openings/voids<br>point is outside of the outer ring, or inside one of the inner rings<br>point is on one of the rings, outer or inner ring |
|---|--|

```
CoordSys coordSys() const;
```

The coordSys of a PlaneProfile returns the internal coordinate transformation from the XY plane to the WCS. Therefore, the vecX and vecY of the coordSys define the plane in which the PlaneProfile has its points. The vecZ of the coordSys is also the normal of the PlaneProfile.

```
LineSeg extentInDir(Vector3d vecDir) const;
```

The extentInDir of a PlaneProfile returns a diagonal of the bounding box. The bounding box is taken in the vecDir direction. That is, the vecDir is first projected onto the plane of the PlaneProfile, then with that direction the upwards direction, the bounding box is taken. The extent-length in the vecDir direction, can easily be calculated from this LineSeg, and is given by abs(vecDir.dotProduct(ls.ptStart()-ls.ptEnd())).

```
Point3d[] getGripEdgeMidPoints() const;
Point3d[] getGripVertexPoints() const;
int moveGripEdgeMidPointAt(int nIndex, Vector3d vecMove);
int moveGripVertexPointAt(int nIndex, Vector3d vecMove);
```

The above routines allow to query and modify the vertex points and the edge mid points of a PlaneProfile. The moveGripEdgeMidPointAt only works on straight line segments currently.

```
int project(Plane planeToProjectOn, Vector vecProjectDirection, double dAccuracy); // (added hsbCAD2009+ build 14.0.73)
```

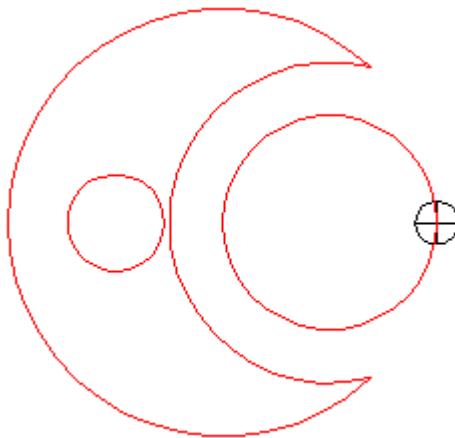
Allows to project the PlaneProfile onto a plane. The resulting PlaneProfile has a new internal plane as well. When the PLines of the profile contain bulges they are first converted into line approximations with the method PLine::convertToLineApprox. For this method, it is important to know to what accuracy the line approximation needs to be done.

```
LineSeg[] splitSegments(LineSeg segToSplit, int bKeepInside) const; // (added in build 18.2.15, and 19.0.23) (see example in Element::noNailProfile)
LineSeg[] splitSegments(LineSeg[] segsToSplit, int bKeepInside) const; // (added in build 18.2.15, and 19.0.23)
```

Return the list of segments that are formed by clipping the given segments by the PlaneProfile. The segments inside the plane profile are kept if bKeepInside is set to TRUE. Otherwise the segment parts outside the plane profile are kept.

---

*[Example O-type with 1 extra grip point. It constructs the following profile. By changing the property, you can alter the position of the smallest circle. With the grip point, the pointInProfile routine can be tested:*



```

(1, "mm");
PropDouble dMoveFactor(0,2);
Vector3d vec = U(100)*_xu;
double dRefLength = vec.length();

PLine pCirBig;
pCirBig.createCircle(_Pt0,_zu,dRefLength*2);
//pCirBig.vis();
PlaneProfile prof(pCirBig); // define profile

PLine pCirSmall;
pCirSmall.createCircle(_Pt0+vec,_zu,dRefLength*1.5);
//pCirSmall.vis();
prof.joinRing(pCirSmall,TRUE); // add an opening (subtract ring)

PLine pCirSmaller;
pCirSmaller.createCircle(_Pt0+vec,_zu,dRefLength);
//pCirSmaller.vis();
prof.joinRing(pCirSmaller,FALSE); // add a ring to it

prof.transformBy(-dMoveFactor*vec); // move the PlaneProfile

PLine pCirSmallest;
pCirSmallest.createCircle(_Pt0-3*vec,_zu,dRefLength*0.45);
//pCirSmallest.vis();
prof.joinRing(pCirSmallest,TRUE); // add an opening (subtract ring)

prof.vis(1); // visualize in debug

// find out if a point is located inside or outside the PlaneProfile
int nPointInProf = prof.pointInProfile(_PtG[0]);

// translate the enum into a string
String strPointInProf;
if (nPointInProf==_kPointInProfile) strPointInProf =
"_kPointInProfile";
if (nPointInProf==_kPointOutsideProfile) strPointInProf =
"_kPointOutsideProfile";

```

```

if (nPPointInProf==_kPointOnRing) strPointInProf = "_kPointOnRing";

reportNotice("\nNumber of rings:" + prof.numRings());
reportNotice("\n_PtG[0] is " + strPointInProf );

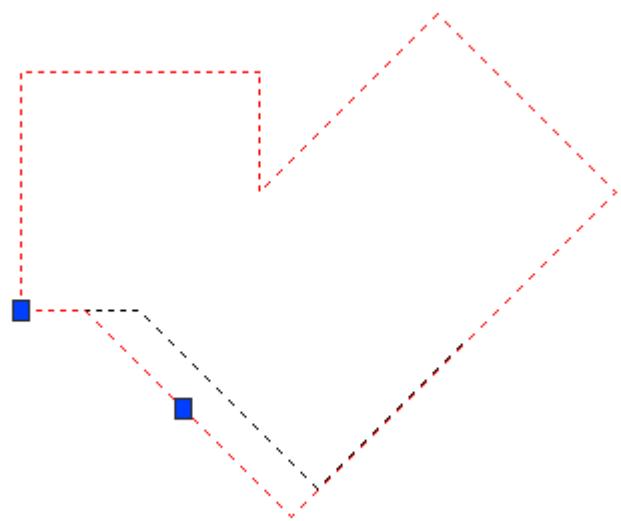
// recompose a planeprofile from the rings
PlaneProfile profRecomposed;
PLine rings[] = prof.allRings();
int subtract[] = prof.ringIsOpening();
if (rings.length()==subtract.length()) {
    for (int p=0; p<rings.length(); p++) {
        profRecomposed.joinRing(rings[p], subtract[p]);
    }
}
profRecomposed.transformBy(vec);
profRecomposed.vis(2);

```

—end example]

---

[Example O-type with 1 extra grip point. It constructs the following profile. The Tsl illustrates the getGrip and moveGrip routines.



```

U(1,"mm");
Vector3d vecX = U(100)*_XU;
Vector3d vecY = U(100)*_YU;

// compose a profile consisting of 2 joined rectangles
PLine pRect;
LineSeg segDiag(_Pt0,_Pt0+2*vecX+2*vecY);
pRect.createRectangle(segDiag,_XU,_YU);
PlaneProfile prof(pRect);

PLine pRect2;

```

```

LineSeg segDiag2(_Pt0+vecX,_Pt0+5*vecX+vecY);
pRect2.createRectangle(segDiag2,_XU+_YU,_YU-_XU);
prof.joinRing(pRect2, FALSE);

Display dp(-1);
dp.draw(prof); // draw profile

int bUseVertices = FALSE;

// get the grip points of the profile
PlaneProfile profMod = prof;
Point3d pnts[0];
if (bUseVertices)
    pnts = profMod.getGripVertexPoints(); // get all the vertex grip
points
else
    pnts = profMod.getGripEdgeMidPoints(); // get all the mid edge
grip points

// determine which point is closest to _PtG[0]
int nInd = -1;
double dDistMin = 0;
for (int p=0; p<pnts.length(); p++) {
    Point3d pt = pnts[p];
    pt.vis();
    double dDist = Vector3d(_PtG[0]-pt).length();
    if (p==0 || dDist<dDistMin) {
        dDistMin = dDist;
        nInd = p;
    }
}

// move a grip point with a vector towards _PtG[0]
if (nInd>=0) {
    if (bUseVertices)
        profMod.moveGripVertexPointAt(nInd, _PtG[0]-pnts[nInd]);
    else
        profMod.moveGripEdgeMidPointAt(nInd, _PtG[0]-pnts[nInd]);
}

dp.color(1); // change color to red
dp.draw(profMod); // draw modified profile

```

*—end example]*

## 4.6 Body

The 3d solid type is Body. This type represents any 3 dimensional shape. Internally the Autocad facet modeler, also called AModeler is used to represent the solid.

Although both the [MetalPart](#) type and the Body type represent a solid shape, there is a conceptual difference and also a different use. The solid that a variable of type Body refers to, is kept on the stack. This means that whenever such a variable goes out of scope, the solid shape is lost, and destructed. This is in contrast with a variable of type MetalPart. The MetalPart variable is actually a reference into a list of solids that belongs to the TSL instance.

When a MetalPart is created with one of the described constructors (see [MetalPart](#)), a new instance is added in the list of solids that is kept in the TSL. That solid will be displayed in the worldDraw of the TSL entity. Even if the variable goes out of scope, the solid will stay in the array that the TSL holds. An assignment of the MetalPart type, is just copying the reference, not the actual body. That is why a special copyPart function exists for the MetalPart.

---

```
class Body {

    Body(Point3d ptStart, Point3d ptEnd, double dRadius); // cylinder shaped body constructor
    Body(Point3d ptStart, Point3d ptEnd, double dRadiusStart, double dRadiusEnd); // cone
        shaped body constructor, added hsbCAD2011 16.0.59
    Body(PLine plShape, Vector3d vecExtrude, double dOffset = 1); // pline extrusion, see
        PLine

    Body(Quader quader); // box shaped, see Quader
    Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ); // box shaped
    Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double
        dLenY, double dLenZ); // box shaped
    Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double
        dLenY, double dLenZ, double dXFlag, double dYFlag, double dZFlag); // box
        shaped

    Body(PLine plStart, PLine plPath, int nApporx = 16); // (added V21.4.14) pline extrusion
        along path, see PLine
    Body(Point3d[] arPtStart, PLine plPath, Point3d[] arPtEnd, int nApporx = 16); // (added
        V21.4.14)

    transformBy(CoordSys csTransformationMatrix); // see CoordSys
    transformBy(Vector vecTranslate);

    int addTool(Tool tl); // returns success of operation since build 17.1.26 and 18.0.17
    int sliceBody(Point3d plPlane, Vector3d vecOuterNormal); // identical to
        addTool(Cut(ptPlane, vecOuterNormal)). Returns success of operation.
        Added since 23.8.0 and 24.0.

    int addPart(Body bdOther); // equivalent with += operator, resolves the intersecting edges
        and faces. Returns success of operation since build 17.1.26 and 18.0.17.
    operator+=(Body bdOther);
}
```

---

```

int subPart(Body bdOther); // equivalent with == operator. Returns success of operation
// since build 17.1.26 and 18.0.17.
operator-=(Body bdOther);

int copyPart(Body bdOther); // equivalent with assignment. Returns success of operation
// since build 17.1.26 and 18.0.17.
int combine(Body bdOther); // add without resolving the intersecting edges. Returns
// success of operation since build 17.1.26 and 18.0.17.
Body[] decomposeIntoLumps() const; // return an array of solid, as inverse operation of
// combine

visualize(int indColor = -1) const;
vis(int indColor = -1) const;

int hasIntersection(Body bdOther) const; // return TRUE if this body has intersection with
bdOther
int hasIntersection(Plane plane) const; // return TRUE if this body has parts on both side of
the plane, FALSE otherwise. Added since 23.8.0 and 24.0.
int intersectWith(Body bdOther); // modifies this body to the intersection with the bdOther

int resolveSelfIntersect(); // (added 24.1.91, 25.1.65) try to unite all the lumps, which might
have been collected by combining parts. Returns validity of body.

double area() const;
double volume() const;
Point3d ptCen() const;
int isNull() const; // (added V22.1.6 and V21.4.81)
int isValid() const; // (added V24.1.88 and V25.1.57)

double lengthInDirection(Vector3d vecDir) const;
Point3d[] extremeVertices(Vector3d vecDir) const;
Point3d[] intersectPoints(Line InToIntersectWith) const; // returns the ordered list of points,
// in direction of In.vecX()
int rayIntersection(Point3d ptLine, Vector3d vecDir, Point3d& ptLine) const; // half line
intersection.

Point3d[] allVertices() const;

// filter an array of beams
Beam[] filterGenBeamsIntersect(Beam[] arToCheck) const;
Sheet[] filterGenBeamsIntersect(Sheet[] arToCheck) const;
Sip[] filterGenBeamsIntersect(Sip[] arToCheck) const;
GenBeam[] filterGenBeamsIntersect(GenBeam[] arToCheck) const;

PlaneProfile extractContactFacelnPlane(Plane plane, double dToleranceDistance) const; //
see PlaneProfile
PlaneProfile getSlice(Plane plane) const; // return a PlaneProfile resulting from the
// slicing
PlaneProfile shadowProfile(Plane plane) const; // return a PlaneProfile resulting from the
// projection of the body on the plane

LineSeg[] hideDisplay(CoordSys csView, int bShowHiddenLines, int bShowOnlyHiddenLines,
// int bShowApproximatingEdges) const;

```

```

void dbCreateAsMassElement(CoordSys csEcs, int nColor) const; // appends the body as
entity. It is for debugging only.
Entity dbCreateAs3dSolid(int nColor) const; // appends the body as 3d solid entity, added to
hsbCAD2014 build 19.0.61

static void createSatFile(Body* arBody, String strFileName); // (added hsbCAD2010 build
15.3.12 and hsbCAD2011 build 16.0.23)
static String createStlFile(Body* arBody, String strFileName, int bAsciiFormat); // (added to
hsbCAD2014 build 19.0.61) Return string with error message on error. Return empty
string on success.

};

Body operator+(Body bd1, Body bd2);
Body operator- (Body bd1, Body bd2);

```

---

**Body**(**Point3d** ptStart, **Point3d** ptEnd, **double** dRadius);

Construct a new solid part with a cylindrical shape. Both ptStart and ptEnd are positioned at the axis line.

**Body**(**PLine** plShape, **Vector3d** vecExtrude, **double** dOffset = 1);

Construct a new solid part as an extrusion of a polyline. The third parameter is optional, and has a default value of 1. This parameter expresses the the position of the polyline, with regard to the final shape, in the vecExtrude direction. If dOffset==1, the plShape is at the beginning of the body. If dOffset==0, the plShape is at the middle. dOffset can have any value.  
The vecExtrude is a vector of which the length is used as extrusion length.

**Body**(**PLine** plStart, **PLine** plPath, **int** nApporx = 16); // (added V21.4.14) pline extrusion along
path, see [PLine](#)

Construct a new solid by extruding the plStart along the path plPath. If any of the PLines contains bulges, the nApprox is used to introduce arc points. the nApprox, default 16 will facet a full circle into 16 parts. If the arc is smaller than a full circle, less facet segment are introduced. The start point of the plPath should be in the plane of plStart, and the start point should lie inside the plStart as well.

Example use below.

**Body**(**Point3d**\* arPtStart, **PLine** plPath, **Point3d**\* arPtEnd, **int** nApporx = 16); // (added
V21.4.14)

Construct a new solid by extruding the polygon formed by arPtStart along the path plPath, and morphing into the polygon formed by arPtEnd.

The start point of the plPath should be in the plane of plStart, and the start point should lie inside the plStart as well.

Example use below.

```
Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double dLenY, double dLenZ);
Body(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double dLenY, double dLenZ, double dXFlag, double dYFlag, double dZFlag);
```

Construct a new body part with a box shape. To specify the dimensions, the length of the vector is multiplied with the length factor in that direction. Eg.: the height of the box is the same as **vecY.length()\*dLenY**. If **vecY** is a unit-length vector, then the **dLenY** corresponds with the height. If **dLenY** equals 1, then the length of the vector expresses the height.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -**vecX**, -**vecY** and -**vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

```
transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);
```

Apply a coordinate transformation to the solid.

```
addTool(Tool tl);
```

All tools that can be added to beams, can also be added to a solid body, eg.: Drill, Cut, BeamCut,... Whenever the **addTool** is called, the solid operation on the body itself is performed.

```
addPart(Body bdOther);
operator+=(Body bdOther);
```

To join different bodies together, the function **addPart** can be called. Doing this, the **bdOther** will not be changed. The call is completely equivalent with the **+=** operator. So with **bd1** and **bd2** of type **Body**, writing **bd1.addPart(bd2)** is the same as **bd1 += bd2**.

```
subPart(Body bdOther);
operator-=(Body bdOther);
```

To subtract **mpOther** from the current body, the function **subPart** can be called. Doing this, the **bdOther** will not be changed. The call is completely equivalent with the **-=** operator. So with **bd1** and **bd2** of type **Body**, writing **bd1.subPart(bd2)** is the same as **bd1 -= bd2**;

```
copyPart(Body bdOther);
```

Take a copy of an existing solid body. The **bdOther** solid will not be changed, and stay intact. This routine exists for compatibility with the **MetalPart** type. A normal assignment between 2 body types also works fine, and is more readable.

```
visualize(int indColor = -1) const;
vis(int indColor = -1) const;
```

In debug mode, the solid can be visualized. If no color is given, the color of the instance is used, indicated by value -1. Visualizing will not render the body. For normal displaying, see [Display](#).

```
Body operator+(Body bd1, Body bd2);
Body operator-(Body bd1, Body bd2);
```

Bodies can also be subtracted and added with normal arithmetics.  
e.g.: **Body** bd4 = bd1 + (bd3 - bd2);

```
int hasIntersection(Body bdOther) const;
int intersectWith(Body bdOther);
```

Both routines return if the bdOther intersects with this body. The hasIntersection routine, just checks if the other body intersects with this one, without changing this body, while the intersectWith routine actually calculates the intersection body.

```
Point3d ptCen() const;
```

Returns the centroid point of the body.

```
double lengthInDirection(Vector3d vecDir) const;
Point3d[] extremeVertices(Vector3d vecDir) const;
```

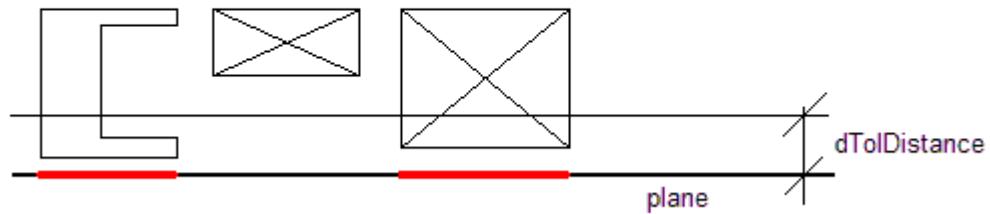
Calculate the length, or the extreme points in a certain direction. If the operation was not succesfull, no points are retruned, and the length of the point array will be zero.

```
Beam[] filterGenBeamsIntersect(Beam[] arToCheck) const;
Sheet[] filterGenBeamsIntersect(Sheet[] arToCheck) const;
Sip[] filterGenBeamsIntersect(Sip[] arToCheck) const;
GenBeam[] filterGenBeamsIntersect(GenBeam[] arToCheck) const;
```

These routines have as argument an array of genbeams, and also as return value an array of genbeams. The array returned is a subset of the array passed in as argument. Each function will filter out those genbeams that have intersection with this body. The body of the genbeam that is used in this intersection check is the envelopeBody.

```
PlaneProfile extractContactFacelnPlane(Plane plane, double dToleranceDistance) const;
```

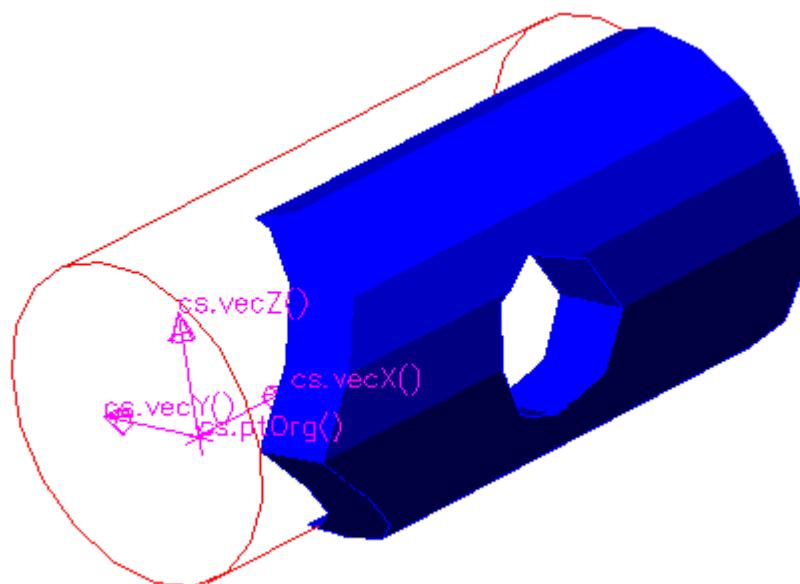
Returns the PlaneProfile representation of one of the surfaces of the Body. This surface is choosen as the surface closest to the given plane. Also the distance of the surface to the plane must be smaller than dToleranceDistance.



**LineSeg[] hideDisplay(CoordSys csView, int bShowHiddenLines, int bShowOnlyHiddenLines, int bShowApproximatingEdges) const**

Returns an array of line [LineSeg](#) that represent the body. Looking from the -vecZ direction towards the ptOrg of the csView, some edges might be hidden. Setting the arguments bShowHiddenLines, bShowOnlyHiddenLines and bShowApproximatingEdges, one can influence the array of segments that will be returned. All edges returned are still at the location of the body. One can use the projectLineSegs from [Plane](#) to transform these to a certain plane.

[Example O-type:



```
Unit(1, "mm"); // use mm as units

double dLength = U(400);
double dDiam = U(100);
```

```

CoordSys cs(_Pt0,_XU,_YU,_ZU); // define a cs as UCS at time of
insertion
cs.vis(6); // visualize the coordSys only in debug
Body bd1( Point3d(0,0,0), Point3d(dLength,0,0), dDiam ); // construct
a metalpart in the origin
bd1.transformBy(cs);

Body bd2 = bd1; // copy the body
bd2.transformBy(U(50)*_YU); // move the copied body
bd2.vis(1); // visualize the body only in debug

bd1 -= bd2; // subtract the second from the first

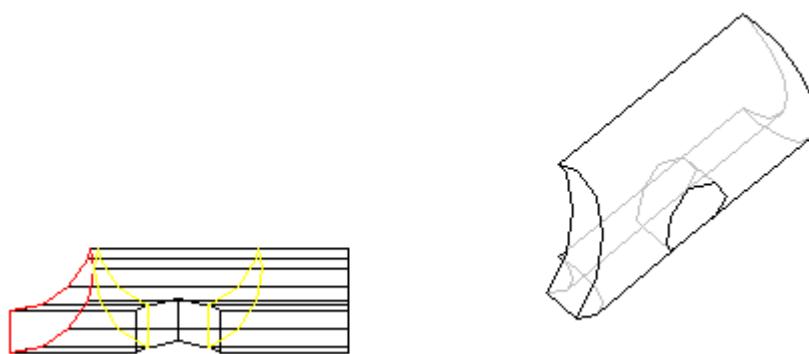
// Define a drill and apply
Point3d ptD1 = _Pt0 + 0.5*dLength*_XU - dLength*_YU;
Point3d ptD2 = _Pt0 + 0.5*dLength*_XU + dLength*_YU;
Drill drl( ptD1, ptD2, dDiam/2 );
bd1.addTool(drl); // drill the body

// Define some cuts and apply
Cut ct(_Pt0, -_XU+_ZU);
bd1.addTool(ct); // cut the body
Cut ct2(_Pt0, -_XU-_ZU);
Plane plCut2(_Pt0, -_XU-_ZU); // plane at location of cut
bd1.addTool(ct2); // cut the body
PlaneProfile profCut2 = bd1.extractContactFaceInPlane(plCut2,U(10));
//profCut2.vis(2);

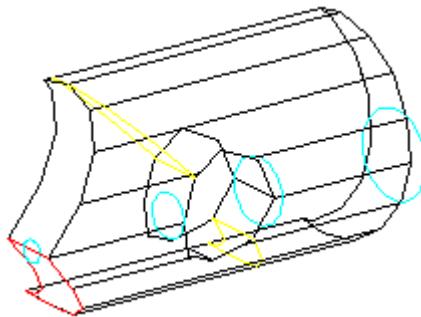
Plane pl(_Pt0 + 0.5*dLength*_XU, -_XU-_ZU);
PlaneProfile profSlice = bd1.getSlice(pl);
//profSlice.vis(2);

// display the solid, and collected profiles
Display dp(-1); // configure display for use of entity color
dp.draw(bd1); // show the body
dp.color(1); // red
dp.draw(profCut2);
dp.color(2); // yellow
dp.draw(profSlice);

```



```
//////////  
// Build a projection of the body  
  
// define projection coordsys SW isometric  
Vector3d vecX = _XW-_YW; vecX.normalize();  
Vector3d vecY = _XW+_YW+_ZW; vecY.normalize();  
Vector3d vecZ = vecX.crossProduct(vecY); vecZ.normalize();  
CoordSys csProj(_Pt0,vecX,vecY,vecZ); // fixed in world  
  
// collect the linesegments for the projected view: hidden line set,  
and visible line set  
int bShowHiddenEdges = TRUE;  
int bShowOnlyHiddenEdges = TRUE;  
int bShowApproxEdges = FALSE;  
LineSeg segsh[] = bd1.hideDisplay(csProj, bShowHiddenEdges,  
bShowOnlyHiddenEdges, bShowApproxEdges);  
  
bShowHiddenEdges = FALSE; // normal  
bShowOnlyHiddenEdges = FALSE; // normal  
LineSeg segsv[] = bd1.hideDisplay(csProj, bShowHiddenEdges,  
bShowOnlyHiddenEdges, bShowApproxEdges);  
  
// transform points from 3d space to projected space  
CoordSys csToProj = csProj; csToProj.invert();  
segsv = csToProj.transformLineSegs(segsv);  
segsh = csToProj.transformLineSegs(segsh);  
  
// project points onto XY plane  
Plane plProject(Point3d(0,0,0), _ZW);  
segsv = plProject.projectLineSegs(segsv);  
segsh = plProject.projectLineSegs(segsh);  
  
// move points to original location, but shifted  
CoordSys csMove; csMove.setToTranslation(Vector3d(_Pt0+U(600)*_XW));  
segsv = csMove.transformLineSegs(segsv);  
segsh = csMove.transformLineSegs(segsh);  
  
// display the arrays of line segments  
dp.color(9); // gray  
dp.draw(segsh);  
dp.color(7); // white  
dp.draw(segsv);
```



```
///////////
// find line body intersection
Line lnToIntersect(_Pt0-_U(80)*_YU, _XU); // direction is _XU
lnToIntersect.vis(1);
Point3d ptsIntersect[] = bd1.intersectPoints(lnToIntersect);
double dRadius = 0;
dp.color(4); // cyan
for (int p=0; p<ptsIntersect.length(); p++) {
    dRadius += U(10); // increase every step, so see the first one
    small
    PLine pCir; pCir.createCircle(ptsIntersect[p], _XU, dRadius);
    dp.draw(pCir);
}
```

*—end example]*

[Alternative transformation from `hideDisplay` to projected in WCS, just substitute in script above

```
///////////
// Build a projection of the body

// define projection coordsys SW isometric
Vector3d vecX = _XW- _YW; vecX.normalize();
Vector3d vecY = _XW+ _YW+ _ZW; vecY.normalize();
Vector3d vecZ = vecX.crossProduct(vecY); vecZ.normalize();
CoordSys csProj(_Pt0,vecX,vecY,vecZ); // fixed in world

// collect the linesegments for the projected view: hidden line set,
and visible line set
int bShowHiddenEdges = TRUE;
int bShowOnlyHiddenEdges = TRUE;
int bShowApproxEdges = FALSE;
LineSeg segsh[] = bd1.hideDisplay(csProj, bShowHiddenEdges,
    bShowOnlyHiddenEdges, bShowApproxEdges);

bShowHiddenEdges = FALSE; // normal
bShowOnlyHiddenEdges = FALSE; // normal
```

```

LineSeg segsv[] = bd1.hideDisplay(csProj, bShowHiddenEdges,
    bShowOnlyHiddenEdges, bShowApproxEdges);

// transformation from the ECS the WCS
CoordSys csToWcs; csToWcs.setToAlignCoordSys(_Pt0, vecX, vecY, vecZ,
    _Pt0, _XW, _YW, _ZW);
// transformation to project points in XY plane
CoordSys csToProj; csToProj.setToProjection(Plane(_Pt0, _ZW), _ZW);
csToWcs.transformBy(csToProj); // combine transformations
// transformation to move points
CoordSys csMove; csMove.setToTranslation(Vector3d(U(600)*_XW));
csToWcs.transformBy(csMove); // combine transformations

// transform segments
segsv = csToWcs.transformLineSegs(segsv);
segsh = csToWcs.transformLineSegs(segsh);

// display the arrays of line segments
dp.color(9); // gray
dp.draw(segsh);
dp.color(5); // blue
dp.draw(segsv);

```

—end example]

[Example O-type to illustrate the createSatFile

```

if (_bOnInsert) {

    _Pt0 = getPoint();
    PrEntity ssE(T("\n|Select a set of entities|"), Entity());
    if (ssE.go()) {
        _Entity = ssE.set();

        Body arBd[0];
        for (int e=0; e<_Entity.length(); e++) {
            Entity ent = _Entity[e];
            Body bd = ent.realBody();
            arBd.append(bd);
        }

        String strFile = _kPathDwg +"\\\"+ scriptName(); // extension
        will be added automatically
        Body().createSatFile(arBd,strFile);
    }
    eraseInstance();
    return;
}

```

—end example]

[Example O-type to illustrate the dbCreateAs3dSolid

```
if (_bOnInsert) {
    Entity ent = getEntity();
    _Pt0 = getPoint();

    Body bd = ent.realBody(_ZW);
    Point3d ptBody = bd.ptCen();
    bd.transformBy(_Pt0 - ptBody);

    Entity entNew = bd.dbCreateAs3dSolid(1);

    _Entity.append(entNew);

    eraseInstance();
    return;
}
```

—end example]

[Example O-type to illustrate the createStlFile

```
if (_bOnInsert) {

    _Pt0 = getPoint();
    PrEntity ssE(T("\n|Select a set of entities|"), Entity());
    if (ssE.go()) {
        _Entity = ssE.set();

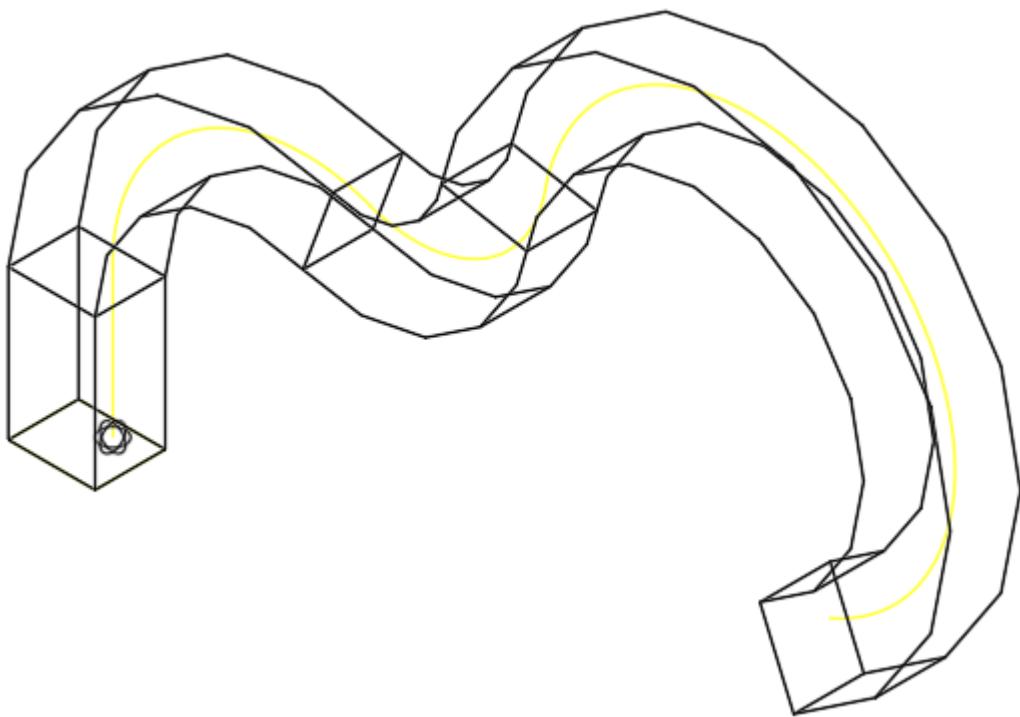
        Body arBd[0];
        for (int e=0; e<_Entity.length(); e++) {
            Entity ent = _Entity[e];
            Body bd = ent.realBody();
            arBd.append(bd);
        }

        String strFile = _kPathDwg +"\\"+ scriptName(); // extension
        will be added automatically
        String strError = Body().createStlFile(arBd,strFile,TRUE);
        if (strError.length() != 0)
            reportMessage("\n" + strError);
    }
    eraseInstance();
    return;
}
```

—end example]

---

[Example O-type illustrating Body constructor with extrude along path]



```
Unit(1, "mm");
if (_bOnInsert)
{
    _Entity.append(getEntPLine(T("Select pline curve")));
    _Entity.append(getEntPLine(T("Select pline path")));
    _Pt0 = getPoint();
    return;
}

if (_Entity.length() < 2)
{
    eraseInstance();
    return;
}

EntPLine entCurve= (EntPLine)_Entity[0];
EntPLine entPath = (EntPLine)_Entity[1];
if (!entCurve.bIsValid() || !entPath.bIsValid())
{
    eraseInstance();
    return;
}

setDependencyOnEntity(entCurve);
setDependencyOnEntity(entPath);
```

```

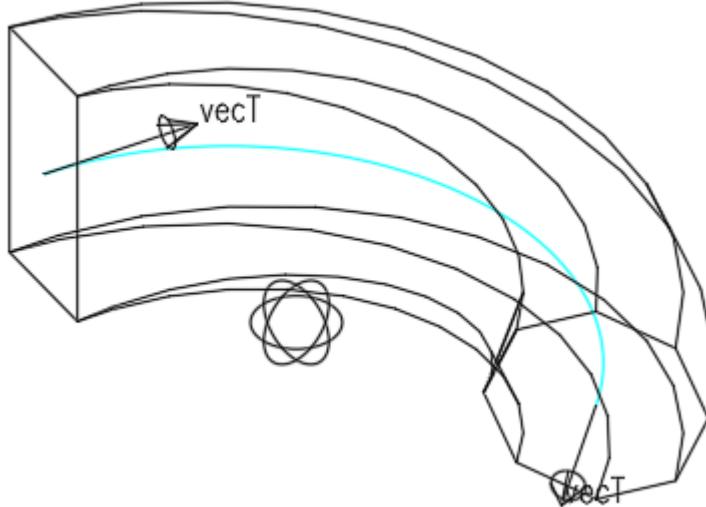
PLine plCurve = entCurve.getPLine();
PLine plPath = entPath.getPLine();
Body bd(plCurve, plPath);

bd.vis();

—end example]

```

[Example O-type illustrating Body constructor with extrude along path



```

Unit(1, "mm");
if (_bOnInsert)
{
    _Entity.append(getEntPLine(T(" | Select spline path")));
    _Pt0 = getPoint();
    return;
}

if (_Entity.length() < 1)
{
    eraseInstance();
    return;
}

double dSize = U(100);

EntPLine entPath = (EntPLine)_Entity[0];
if (!entPath.bIsValid())
{
    eraseInstance();
    return;
}

setDependencyOnEntity(entPath);

```

```

PLine plPath = entPath.getPLine();
Vector3d vecN = plPath.coordSys().vecZ();

Point3d arPtStart[0];
{
    Point3d pt = plPath.ptStart();
    Vector3d vectT = plPath.getTangentAtPoint(pt);
    vectT.vis(pt);
    Vector3d vecN2 = vectT.crossProduct(vecN).normal();

    double dS1 = 0.6 * dSize;
    double dS2 = 0.3 * dSize;
    arPtStart.append(pt + dS1 * vecN + dS2 * vecN2);
    arPtStart.append(arPtStart.last());

    arPtStart.append(pt + dS1 * vecN - dS2 * vecN2);
    arPtStart.append(arPtStart.last());

    arPtStart.append(pt - dS1 * vecN - dS2 * vecN2);
    arPtStart.append(arPtStart.last());
}

Point3d arPtEnd[0];
{
    Point3d pt = plPath.ptEnd();
    Vector3d vectT = plPath.getTangentAtPoint(pt);
    vectT.vis(pt);
    Vector3d vecN2 = vectT.crossProduct(vecN).normal();

    double dR = 0.5 * dSize;
    int nn = 8;
    for (int n=0; n<nn; n++)
    {
        double dAng = (n * 360.0) / nn;
        CoordSys cs;
        cs.setToRotation(dAng + 90, vectT, pt);
        Vector3d vecOffset = vecN;
        vecOffset.transformBy(cs);

        arPtEnd.append(pt + dR * vecOffset);
    }
}

int nApprox = 32;
Body bd(arPtStart, plPath, arPtEnd, nApprox);
bd.vis();

```

*—end example]*

## 4.7 LineBeamIntersect

To calculate the intersection of a line or a polyline with a beam, an instance of the LineBeamIntersect class must be made. An instance can be constructed with the following constructors:

```
LineBeamIntersect name(Point3d ptViewLine, Vector3d vecViewLine, Beam bm);
LineBeamIntersect name(Point3d ptViewLine, Vector3d vecViewLine, Beam bm, int
nDirection);
LineBeamIntersect name(PLine polyline, Beam bm, int nDirection);
```

The first constructor will find the intersection of the infinite beam bm, with an infinite line through the point ptView, and with direction vecViewDirection. The second constructor allows to use a half-infinite line. Depending on the nDirection, the vecViewLine side is taken or the oposite side is taken.

nDirection == 1: find intersection with half line from ptViewLine, in vecViewLine direction  
nDirection == -1: find intersection with half line from ptViewLine, in negative vecViewLine direction  
nDirection == 0: find intersection with beam, with an infinite line (same as first contructor)

The third contructor, uses a polyline and a direction specification, nDirection, to find the intersection between the bounded polyline, and the infinite beam bm.

The meaning of the nDirection parameter is:

nDirection == 1: find intersection from beginning of polyline to end of polyline (the view point is the start point of the polyline)  
nDirection == -1: find intersection from end to beginning of polyline  
nDirection == 0: find intersection with beam, independent of the polyline orientation

The member fuctions that are defined on LineBeamIntersect are

```
Point3d pt1 = lblIntersect.pt1(); // main intersection point
Vector3d vn1 = lblIntersect.vecNrm1(); // normal in point pt1, towards inside of beam
Point3d pt2 = lblIntersect.pt2(); // additional intersection point
Vector3d vn2 = lblIntersect.vecNrm2(); // normal in point pt2, towards inside of beam
int bCntct = lblIntersect.bHasContact(); // returns TRUE if pt1 is valid, and hits a surface of
the infinite extended beam.
int nNumPt = lblIntersect.nNumPoints(); // returns the number of valid intersection points
that are calculated.
```

To determine the two points, pt1 and pt2, a normal beam is seen as a collection of 4 infinite surfaces. Each of the 4 surfaces of the beam lies inside one of these infinite surfaces. Maximum two surfaces out of the 4 are considered for intersection calculations. The surfaces that are considered, are the ones that can be "seen" from the view-point, in the direction of the view-direction or polyline direction. If only one surface is "seen", the member function nNumPoints, will return 1. If the axis of the beam is parallel with the intersection line, nNumPoints will return 0. For an arbitrary intersection, and a view-point far away from the beam, the nNumPoints will return 2, and 2 valid points will be calculated.

If the line/polyline, hits the infinite extended beam, the bHasContact routine returns TRUE, otherwise it returns FALSE. If bHasContact is true, nNumPoints returns at least 1.

The following illustrates the above functions.

[Example E-type with 2 required beams:

```

// Find intersection of the axis of the primary beam, _Beam0, with the second beam
// Beam1.
// Look in the _X0 direction to the second beam.
LineBeamIntersect LBI(_Pt0,_X0,_Beam1,1);

if (LBI.nNumPoints()==1) { // only one point is calculated
    Point3d ptT1 = LBI.pt1();
    Vector3d vecT1 = LBI.vecNrm1();
    ptT1.vis(1);
    vecT1.vis(ptT1,1);

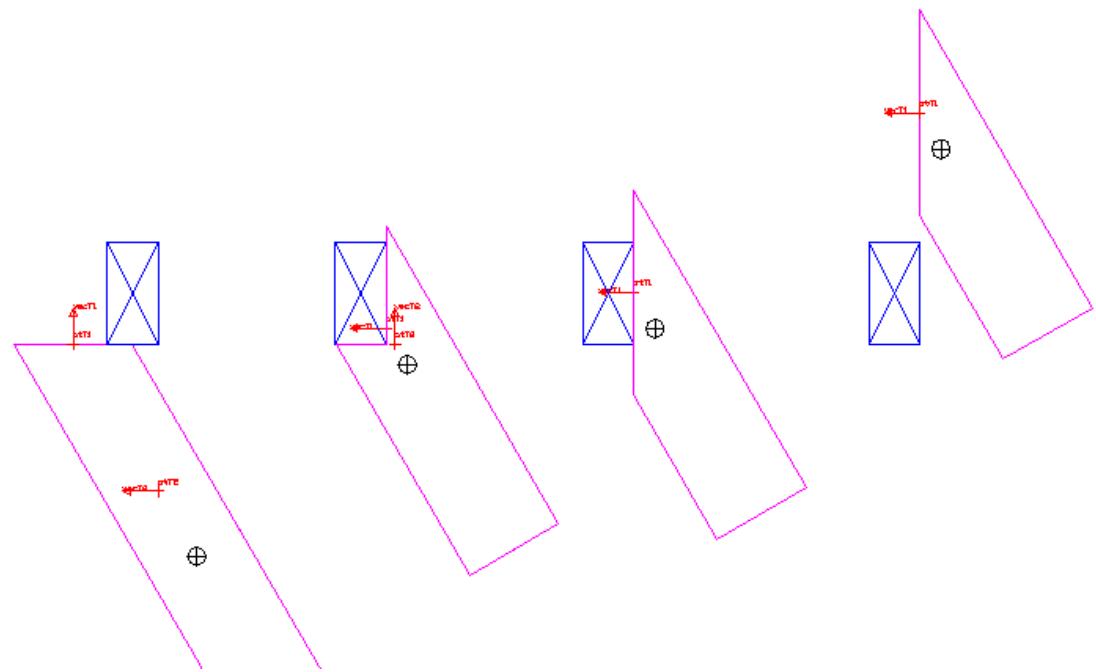
    Cut ct(ptT1,vecT1);
    _Beam0.addTool(ct,1);
}
else if (LBI.nNumPoints()==2) { // two points are calculated
    Point3d ptT1 = LBI.pt1();
    Vector3d vecT1 = LBI.vecNrm1();
    ptT1.vis(1);
    vecT1.vis(ptT1,1);

    Point3d ptT2 = LBI.pt2();
    Vector3d vecT2 = LBI.vecNrm2();
    ptT2.vis(1);
    vecT2.vis(ptT2,1);

    if (LBI.bHasContact()) { // the first point touches the beam
        DoubleCut dct(ptT1,vecT1,ptT2,vecT2);
        _Beam0.addTool(dct,1);
        reportMessage("\nHas Contact");
    }
    else {
        Cut ct(ptT1,vecT1);
        _Beam0.addTool(ct,1);
        reportMessage("\nHas NO Contact");
    }
}

```

—end example]



## 4.8 CoordSys, Matrix and transformation

The language contains a type to describe and hold a complete coordinate system: `CoordSys`. The coordinate system contains an origin point, an X vector, an Y vector and a Z vector. The `CoordSys` can be constructed with these arguments. There is also a set of functions available to get these values from an existing `CoordSys`.

```
CoordSys(Point3d ptOrig, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
```

Alternatively, the `CoordSys` type can be seen as representing the coordinate transformation from the world coordinate system to the given coordinate system. It represents the transformation from `_PtW(0,0,0)`, `_XW(1,0,0)`, `_YW(0,1,0)`, `_ZW(0,0,1)` to the coordinate system `ptOrig`, `vecX`, `vecY`, `vecZ`. Of course, a transformation matrix might be composed in a number of different ways, e.g.: specifying the `setToRotation`, `setToTranslation`,...

---

```
class CoordSys {

    CoordSys(); // default constructor sets the coordsys to world
    CoordSys(Point3d ptOrig, Vector3d vecX, Vector3d vecY, Vector3d vecZ);

    setToRotation(double dAngle, Vector3d vecAxis, Point3d ptCenter);
    setToTranslation(Vector3d vecTranslation);
    setToProjection(Plane plProject, Vector3d vecDirProject);
    setToAlignWorldToPlane(Plane plane);
    setToAlignPlaneToWorld(Plane plane);
    setToAlignCoordSys(Point3d ptOrigFrom, Vector3d vecXFrom, Vector3d vecYFrom, Vector3d
                      vecZFrom, Point3d ptOrigTo, Vector3d vecXTo, Vector3d vecYTo, Vector3d vecZTo);
```

---

```

setToMirroring(Point3d ptMirror);
setToMirroring(Plane plMirror);
setToMirroring(Line lnMirror);

invert();
double det() const;
double scale() const; // return the scaling if the transformation is a orthogonal transformation.

transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);

Point3d ptOrg() const;
Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;

Point3d[] transformPoints(Point3d[] arPnts) const; // just a loop over all the point, with a
// transformBy on each on them
LineSeg[] transformLineSegs(LineSeg[] arSegs) const;

visualize(int indColor = -1) const;
vis(int indColor = -1) const;

// routines for internal use only
Point3d getEulerAngles(CoordSys csOther) const; // the 3 euler angles are returned as X,Y and
// Z component of a point
Point3d getEulerAngles(CoordSys csOther, int nType) const; // default nType is 3, nType=1
// Hgg, nType=2 Alfa12, nType=3 Twist14.

};

```

**CoordSys**(**Point3d** ptOrig, **Vector3d** vecX, **Vector3d** vecY, **Vector3d** vecZ);

Constructing a new coordinate system with given origin, and set of vectors. It also represents the transformation from \_PtW(0,0,0), \_XW(1,0,0), \_YW(0,1,0), \_ZW(0,0,1) to the coordinate system ptOrig, vecX, vecY, vecZ.

**setToRotation**(**double** dAngle, **Vector3d** vecAxis, **Point3d** ptCenter);

Set the contents of the ptOrig, vecX, vecY, vecZ to the transformation representing a rotation over the angle dAngle, around the axis vecAxis, in the point ptCenter. The angle is given in degrees.

**setToTranslation**(**Vector3d** vecTranslation);

Set the contents of the ptOrig, vecX, vecY, vecZ to the transformation representing a translation with the vector vecTranslation.

```
setToAlignCoordSys(Point3d ptOrigFrom, Vector3d vecXFrom, Vector3d vecYFrom, Vector3d
vecZFrom, Point3d ptOrigTo, Vector3d vecXTo, Vector3d vecYTo, Vector3d
vecZTo);
```

Set the contents of the ptOrig, vecX, vecY, vecZ to the transformation from the coordinate system ptOrigFrom, vecXFrom, ... to the coordinate system ptOrigTo, vecXTo,...

```
setToMirroring(Point3d ptMirror);
setToMirroring(Plane plMirror);
setToMirroring(Line lnMirror);
```

Set the contents of the ptOrig, vecX, vecY, vecZ to the transformation representing a mirroring. It can either be a mirroring with respect to a point, or a plane or a line.

*[Example E-type:*

```
Unit(1, "mm");
BeamCut bc(_Pt0,_Y0,_X0,_Z0,_W0,U(40),_H0,0,0,1);
_Beam0.addTool(bc);
CoordSys mirror;
mirror.setToMirroring(Plane(_Beam0.ptCen(),_X0));
bc.transformBy(mirror);
_Beam0.addTool(bc);
—end example]
```

```
invert();
```

Set the contents of the ptOrig, vecX, vecY, vecZ to the inverted transformation matrix.

```
double det() const;
```

Calculate the determinant of the transformation matrix.

```
Point3d ptOrg() const;
Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;
```

Retrieve the individual components of the transformation matrix.

```
transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);
```

Calculate the new coordinate system as the transformation of the origin and vectors by the csTransformationMatrix. If the coordinate system, ptOrig, vecX, vecY, vecZ is equal to the world coordinate system, then after the transformBy call, it will be transformed into the coordinate system of csTransformationMatrix. A transformBy of a transformation matrix, can also be seen as a matrix premultiplication by csTransformationMatrix. The result is a transformation that

consists of the sequence "current transformation" before "csTransformationMatrix". So the call translation.transformBy(rotation) results in a transformation of a translation followed by a rotation.

```
visualize(int indColor = -1) const; vis(int indColor = -1) const;
```

In debug mode, the coordinate system can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

[Example any-type:

```
Vector3d vec1 = _XU;
Point3d pt1 = _Pt0;
_ZU.vis(_Pt0);
pt1.vis();
vec1.vis(pt1);

CoordSys cs1(_Pt0,_XU,_YU, _ZU);
CoordSys cs2, cs3;
cs3.setToRotation(45,_ZU,_Pt0);
cs2.setToTranslation(100*_XU + 100*_YU);

Point3d pt2 = pt1;
Vector3d vec2 = vec1;
pt2.transformBy(cs2);
vec2.transformBy(cs2);
pt2.vis();
vec2.vis(pt2);

Point3d pt3 = pt2;
Vector3d vec3 = vec2;
pt3.transformBy(cs3);
vec3.transformBy(cs3);
pt3.vis();
vec3.vis(pt3);
```

—end example]

## 4.9 Quader

The Tsl language contains a type to describe a box defined by one point and 3 vectors with their lengths. The Quader contains an origin point, an X vector, an Y vector and a Z vector.

The Quader has the following predefined sides. These are **int** values, and range from 0 to 5. (added since 24.1.87 and 25.1.55).

---

**\_kXP = 0, \_kXN, \_kYP, \_kYN, \_kZP, \_kZN = 5**

---

```

class Quader {

    Quader(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
    Quader(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX,
        double dLenY, double dLenZ);
    Quader(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX,
        double dLenY, double dLenZ, double dXFlag, double dYFlag, double dZFlag);

    Vector3d vecD(Vector3d vecDir) const; // returns the vector vecX,Y,Z,-X,-Y,-Z most aligned
        with vecDir
    double dD(Vector3d vecDir) const; // returns the dimension of the Quader in vecD(vecDir)
        direction

    CoordSys coordSys() const; // see CoordSys

    Point3d ptOrg() const; // returns ptOrg of coordSys
    Vector3d vecX() const; // returns the normalized vecX of coordSys
    Vector3d vecY() const; // returns the normalized vecY of coordSys
    Vector3d vecZ() const; // returns the normalized vecZ of coordSys
    double dX() const; // (added since 24.1.87 and 25.1.55)
    double dY() const; // (added since 24.1.87 and 25.1.55)
    double dZ() const; // (added since 24.1.87 and 25.1.55)

    double normalizeVectors() const; // Adjust dX and vecX such that vecX()*dX() does not change,
        but vecX is normalized. Do the same for the other vectors. (added 24.1.92 and 25.1.76)

    Point3d pointAt(double dXFlag, double dYFlag, double dZFlag) const; // returns a point
        with coordinates dXFlag,dYFlag, dZFlag, in a system where the directions are
        determined by the Quader, and the scaling is also determined by the Quader. The
        point (1,1,1) retuns the corner point of the Quader in the vecX, vecY, vecZ direction.

    Plane plFaceD(Vector3d vecFace) const; // calculate the plane along the face of the
        quader which has a normal which is most aligned with the given vecFace
        direction.
    Line lnEdgeD(Vector3d vecFace, Vector3d vecDirection) const; // calculate the line along
        the edge of the quader which is lying in the plane plFaceD(vecFace) and is
        found looking from the center point of the quader into the vecDirection.

    transformBy(CoordSys csTransformationMatrix); // see CoordSys
    transformBy(Vector vecTranslate);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    int modifyBoxUntilFaceCompletelyOut(Vector3d vecFace, Body bd, int
        bExtremeVerticesFromIntersection); // return success of operation. The result is an
        extended vecX, vecY or vecZ, and a moved origin, with one face completely outside the body
        bd. The face identified by vecFace is moved. If bExtremeVerticesFromIntersection is TRUE,
        then the new face will just touch the extreme vertex of the intersection with the beam. In

```

---

---

case it is FALSE, the extreme vertex is taken from the beam, without intersecting it with the extended box.

};

---

**Quader**(**Point3d** ptOrg, **Vector3d** vecX, **Vector3d** vecY, **Vector3d** vecZ);

Constructing a new coordinate system with given origin, and set of vectors. It also represents the transformation from \_PtW(0,0,0), \_XW(1,0,0), \_YW(0,1,0), \_ZW(0,0,1) to the coordinate system ptOrig, vecX, vecY, vecZ.

---

**visualize**(**int** indColor = -1) const; **vis**(**int** indColor = -1) const;

In debug mode, the quader can be visualized. If no color is given, the color of the instance is used, indicated by value -1.

---

[Example E-type:

```

Beam bm = _Beam0;
Point3d ptCen = bm.ptCen();
Vector3d vecX = bm.vecX();
Vector3d vecY = bm.vecY();
Vector3d vecZ = bm.vecZ();
double dLen = bm.dL();
double dWidth = bm.dW();
double dHeight = bm.dH();

Quader qdr(ptCen, dLen*vecX, dWidth*vecY, dHeight*vecZ);

double dX = qdr.dD(vecX); // get the dimension in a particular
direction
double dY = qdr.dD(vecY);
double dZ = qdr.dD(vecZ);
CoordSys csQ = qdr.coordSys();

Vector3d vecD = qdr.vecD(vecX); // get the vector most aligned with a
particular vector

// define a quader from
Quader qdr2(csQ.ptOrg(), csQ.vecX(), csQ.vecY(), csQ.vecZ(), dX, dY,
dZ);
Body bd(qdr2); // make a body from a quader
bd.vis(3);

```

---

—end example]

## 4.10 CrossProduct revealed

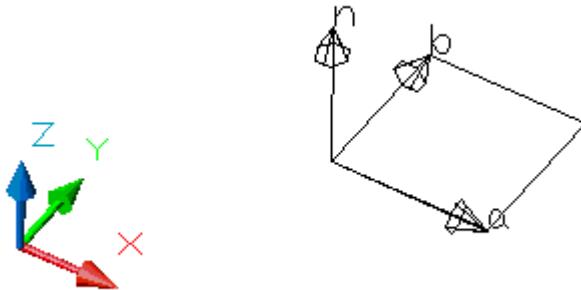
The cross product (also called the vector product) of two vectors is another vector. It has many useful properties, but the one we use most often is that it is perpendicular to both of the given vectors. The cross product is defined only for three-dimensional vectors. The cross product of two vectors  $a$  and  $b$  is often denoted by  $a \times b$ . It has the following useful properties:

1.  $a \times b$  is perpendicular (orthogonal) to both  $a$  and  $b$ .
2. The length of  $a \times b$  equals the area of the parallelogram determined by  $a$  and  $b$ . This area is equal to  $|a \times b| = |a||b|\sin(t)$  where  $t$  is the angle between  $a$  and  $b$ , measured from  $a$  to  $b$  or  $b$  to  $a$ , whichever produces an angle less than 180 degrees.
3. The sense of  $a \times b$  is given by the right-hand rule. For example, twist the fingers of your right hand from  $a$  to  $b$ ; then  $a \times b$  will point in the direction of your thumb.

From these properties, we can also conclude:

1.  $a \times b$  is equal to 0 if, and only if,  $a$  and  $b$  have the same or opposite directions or if either has zero length.
2. For perpendicular  $a$  and  $b$  vectors, the length of the cross product  $a \times b$  is given by  $|a \times b| = |a||b|$ .
3. If both vectors  $a$  and  $b$  are perpendicular and have a unit length, the cross product also has a unit length.
4.  $a \times b = -b \times a$

Sometimes you need to calculate the normal vector  $n$  to a plane. If the plane is known to pass through three specific points, the cross product provides the operation to accomplish this task. Any three points  $p_1$ ,  $p_2$  and  $p_3$  determine a unique plane, as long as the points do not lie in a straight line. To find the normal vector, build 2 vectors:  $a = p_2 - p_1$  and  $b = p_3 - p_1$ , and apply the cross product  $n = a \times b$ . (If the points do lie on a straight line, the cross product will be zero).

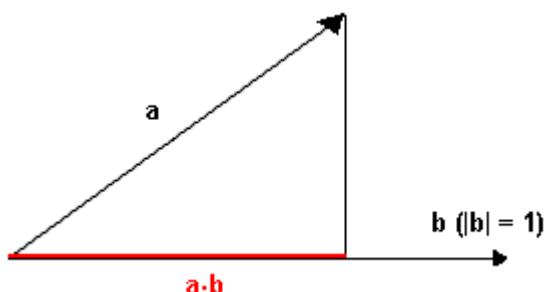


In TSL the cross product can be calculated by calling the function [crossProduct on a vector](#). So  $a \times b$  becomes `a.crossProduct(b)`.

## 4.11 DotProduct revealed

The dot product, also called scalar product, and sometimes called inner product or improduct, is a scalar. The dot product of two vectors  $a$  and  $b$  is often denoted by  $a \cdot b$ . The value of the scalar is the product of the lengths of each vector, multiplied with the cosine of the inner angle between the vectors. So  $a \cdot b = |a| \cdot |b| \cdot \cos(t)$  where  $t$  is the angle between  $a$  and  $b$ , measured from  $a$  to  $b$  or  $b$  to  $a$  (the cosine is not sensitive to that), and  $\cdot$  the multiplication operator.

For the two vectors  $a$  and  $b$ , with coordinates  $(ax, ay, az)$  and  $(bx, by, bz)$ , the scalar product can be easily computed and is given by the expression  $ax \cdot bx + ay \cdot by + az \cdot bz$ . From this expression it is easily seen that if  $b$  equals  $(1, 0, 0)$ , the result of the dot product is  $ax$ . If  $b$  equals  $(0, 1, 0)$ , the dot product equals  $ay$ , and so on. In fact, if  $b$  is a unit length vector, in any direction, the dot product equals the length of the projected  $a$  vector on the line, in the  $b$  direction (see figure).



From the above described properties, we can also conclude:

1. The dot product is positive for  $a$  and  $b$  pointing in the same direction, and negative for  $a$  and  $b$  pointing in the opposite direction. The vectors do not need to be parallel for this.
2. The dot product of perpendicular vectors is zero.
3. The projection of a vector,  $a$  on a line with direction unit vector  $d$ , is given by  $(a \cdot d)d$ .
4. To find the normal projection of a vector into a plane, it is sufficient to subtract the component in the direction normal to the plane. With  $a$  the vector, and  $n$  the normal vector to the plane, this component is given by  $a \cdot n$ .
5.  $a \cdot b = b \cdot a$
6. Given any orthogonal normalized coordinate system, with basic vectors  $u$ ,  $v$  and  $w$ , the coordinates of any vector  $a$  in this coordinate system are given by  $(a \cdot u, a \cdot v, a \cdot w)$ .

In TSL the dot product can be calculated by calling the function [dotProduct on a vector](#). So  $a \cdot b$  becomes  $a.\text{dotProduct}(b)$ .

## 4.12 PropellerSurface

A PropellerSurface represents a ribbon type of surface that is defined by 2 polylines, and a linear interpolation between them. The plines are approximated by line segments. The segment list for each pline contains an equal amount of points. As such the ribbon can be easily constructed. The maximum deviation of an hypothetical smooth surface and the linearisation is set by the `dMaximumDeviation`.

---

`class PropellerSurface // (added hsbCAD2015 build 20.0.6)`

---

```

{
    PropellerSurface();
    PropellerSurface(PLine pl1, PLine pl2, double dMaximumDeviation);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    PLine pline1() const; // PLine
    PLine pline2() const;

    PLine[] intersectWithPlane(Plane plane, int bReduce) const;
}

```

---

**PropellerSurface(PLine pl1, PLine pl2, double dMaximumDeviation);**

Constructing a new propeller ribbon surface. The pl1 is the defining polyline, while pl2 is responsible to define the bevel. The plane of the defining polyline (pl1) is used as reference for the tool. So the normal of the pl1 is used, but flipped towards the location of pl2.

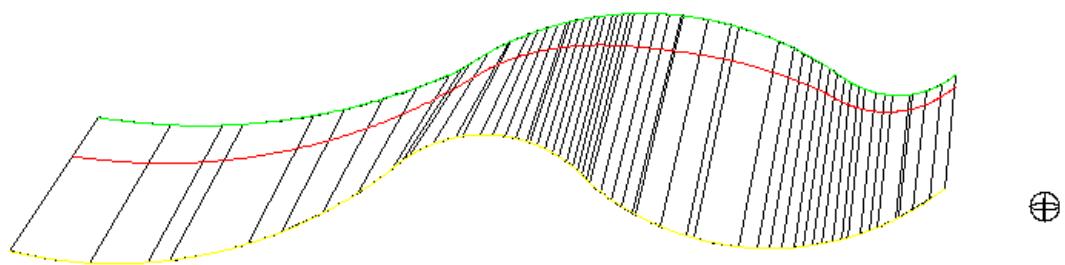
Bevels are defined from the reference plane. So the reference plane its normal is the pl1.coordSys().vecZ() or the negative one -pl1.coordSys().vecZ(). The normal is always pointing in the direction of the pl2.

**PLine[] intersectWithPlane(Plane plane, int bReduce) const;**

Return the list of intersection points of the list of quadruples. If the bReduce is set to FALSE, the intersection points are combined into a pline with straight line segments. If the bReduce is TRUE, the returned PLine is the one created with the set of intersection points, and calling createSmoothArcsApproximation on it (see [PLine](#)).

---

[Example O-type:



```

Unit(1, "mm");

PropDouble pMaxDev(0, U(5), T("Maximum deviation"));

if (_bOnInsert)
{
    _Map.setEntity("plEnt1", getEntPLine(T("Select first pline")));
    _Map.setEntity("plEnt2", getEntPLine(T("Select second pline")));
    _Pt0 = getPoint();
    return;
}

Entity ent1 = _Map.getEntity("plEnt1");
Entity ent2 = _Map.getEntity("plEnt2");
EntPLine plEnt1 = (EntPLine)ent1;
EntPLine plEnt2 = (EntPLine)ent2;

if (!plEnt1.bIsValid() || !plEnt2.bIsValid())
{
    eraseInstance();
    return;
}

PropellerSurface tt(plEnt1.getPLine(), plEnt2.getPLine(), pMaxDev);
tt.vis();
PLine pl1 = tt.pline1();
pl1.vis(3);
PLine pl2 = tt.pline2();
pl2.vis(2);

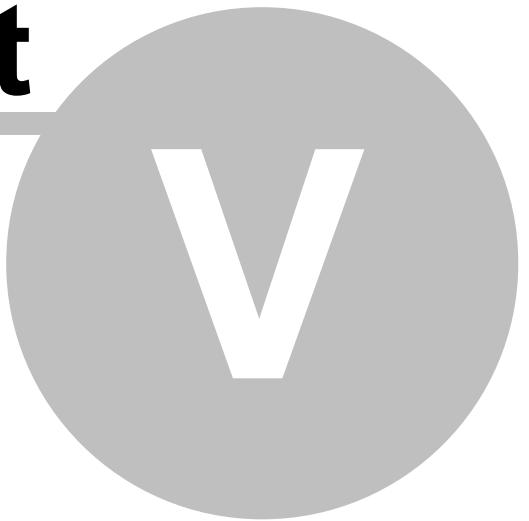
Plane plane(_Pt0, _xw);
int bReduce = TRUE;
PLine pls[] = tt.intersectWithPlane(plane, bReduce);
for (int p=0; p<pls.length(); p++) {
    PLine pl = pls[p];
    pl.vis(1);
}

```

```
String strChangeEntity = T("|Append intersection spline|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    for (int p=0; p<pls.length(); p++) {
        EntPLine().dbCreate(pls[p]);
    }
}

—end example]
```

**Part**



## 5 Basic operations

### 5.1 Global functions

The script language contains a lot of types: int, double, String, Beam, MetalPart,... On each type, a number of functions can be called, e.g. to find the xas of a beam bm: bm.vecX(). So the "." in the instruction bm.vecX() tells to look for the function of the beam, and execute it upon instances bm. These type of functions are called member functions.

Besides the member functions there exist a number of functions that can be called without any instance of a type, the so called global functions. Each global function can return a value of some type. As an example, the function `scriptName()` will return the name of the script as a String, so it could be used as "String str = `scriptName();`".

Here is a list of the global functions:

```
String scriptName(); // retruns the name of the script itself
String dwgName(); // returns the name of the dwg file, without the file path, and without
                     // the extension
String dwgFullName(); // returns the full file path

int arxVersion(); // returns the autocad arx version as an int: 2004, 2005, ...
int bits(); // returns 32 or 64 depending on how the acad exe is running (added
                  hsbCAD15.0.12)
String hsbOEVersion(); // returns the hsb version string that appears at the bottom of
                  hsb_Settings, reporting the version.

String projectName(); // returns the value set in the hsb_settings
String projectNumber(); // returns the value set in the hsb_settings
String projectStreet(); // returns the value set in the hsb_settings
String projectCity(); // returns the value set in the hsb_settings
String projectComment(); // returns the value set in the hsb_settings
String projectUser(); // returns the value set in the hsb_settings
String projectRevision(); // returns the value set in the hsb_settings
String projectSpecial(); // returns the value set in the hsb_settings
String hsbShareProjectId(); // (added V24.0.11) returns the value set in the hsb_settings

int getTickCount(); // returns the number of milliseconds that have elapsed since the
                      system was started, up to 49.7 days.
```

Combine individual red, green and blue values in the range of [0,255] to one int rgb value. To be used in trueColor method of [Display](#) and [Entity](#).

```
int rgb(int r, int g, int b); // (added V23.0.75) r, g and b should be in range [0,255]
int rgb(String strHexRgb); // (added V23.3.10) accepts rrggbb, aarrggbb, 0xrrggbb,
                     #rrggbb, 0xaarrggbb, #aarrggbb, all case insensitive eg rgb("60BB46")
int rgbR(int rgb); // (added V23.1.3) returns the red part [0,255] out of an rgb value
int rgbG(int rgb); // (added V23.1.3) returns the green part [0,255] out of an rgb value
int rgbB(int rgb); // (added V23.1.3) returns the blue part [0,255] out of an rgb value
```

Get the current background rgb color. Depending on being in model or 2d or 3d view mode, and being in layout tab (see [Viewport.inLayoutTab\(\)](#) method), this color might be different.

```
int getBackgroundTrueColor(); // (added V23.1.3), the color returned is in rgb format.
```

Get the current view direction. This is not a state of the drawing database, so the return is only valid for the moment of the call.

```
Vector3d getViewDirection(); // (added V23.1.3) view z direction
Vector3d getViewDirection(int nXYZ); // (added V23.1.3), for X direction, nXZY = 0; for Y,
nXYZ = 1; for Z, nXYZ = 2.
double getViewHeight(); // (added V23.4.4) view height dimension in drawing units
Point3d getViewCenter(); // (added V23.6.34)

int in3dGraphicsMode(); // (added V23.8.50 and V24.1.32) return FALSE if in 2dWireframe
mode, otherwise return TRUE.
```

Get the current UCS. This is not a state of the drawing database, so the return is only valid for the moment of the call.

```
CoordSys getUcs(); // (added V23.8.50 and V24.1.32) return the current ucs
```

These project values can also be set by the following global functions:

```
void setProjectName(String strVal);
void setProjectNumber(String strVal);
void setProjectStreet(String strVal);
void setProjectCity(String strVal);
void setProjectComment(String strVal);
void setProjectUser(String strVal);
void setProjectRevision(String strVal);
void setProjectSpecial(String strVal);
void setHsbShareProjectId(String strVal); // (added V24.0.11)
```

Since hsbCAD2012 17.1.23 and hsbCAD2013 18.0.13 MapX can also be added to ProjectInfo (also called docdata). This is done through the following global functions:

```
Map subMapXProject(String strKey) const; // see Map
void setSubMapXProject(String strKey, Map mapNew);
String[] subMapXKeysProject() const; // return list of available sub map keys
void removeSubMapXProject(String strKey);
```

The values can also be set with the lisp command Hsb\_setDclValue: (Hsb\_setDclValue "PROJECTNAME" "My project")

The following case sensitive keys can be used:

- "PROJECTNAME"
- "PROJECTNUMBER"
- "ADDRESS1" or "STREET"
- "ADDRESS2" or "CITY"
- "COMMENT"
- "USER"
- "REVISION"
- "HSBSPECIAL"

Also with the Hsb\_getDclValue, these values can be queried (among others): (Hsb\_getDclValue "PROJECTNAME")

Since V22.1.31 the posnum compare criteria can be retrieved and set by (also see example below):

```
Map posnumCriteria() const; // see Map
void setPosnumCriteria(Map mapNew);
```

There is also a number of global goniometrical functions sin, cos, atan,... (see section [Angle calculations](#)).

For the global insert functions like `getPoint`, `getElement`... see the section on [Global insert functions](#).

There are 2 global functions that refer to string manipulation:

```
String T(String strToTranslate); // translate a string, see String
String operator+(String strPart1, String strPart2); // concatenate 2 strings, see String
```

There are 2 global functions that refer to body manipulation:

```
Body operator+(Body bd1, Body bd2); // see body
Body operator-(Body bd1, Body bd2); // see body
```

Also the unit conversion function

```
double U(...); // convert to the current drawing units, see Unit
double Unit(...); // convert to the current drawing units, see Unit
```

Besides the global functions that query for something, there exist functions that set things for later use.

```
void setCompareKey(String strToSet); // sets a compare key for posnum assignment, see compare.
void setOPMKey(String strToSet); // sets a key for use by the OPM. See TslInst for retrieving it.
```

There are three functions to report what is happening inside the toolscript:

```
void reportError(String strError); // pops up a Error dialog box with message strError,
// and stops the execution of the script.
void reportWarning(String strWarning); // pops up a dialog box with message strWarning,
// and continues the execution of the script.
void reportMessage(String strMessage); // prints in acad standard output window the strMessage,
// and continues the execution of the script.
void reportNotice(String strMessage); // prints in a modeless popup window the strMessage,
// and continues the execution of the script.
```

There are some functions to influence the DXA output of the TSL instance

```
void exportWithElementDxa(Element el); // sets the element to export the TSL with, see DXA output.
void exportToDxi(int bSet); // sets the element to export to a dxi, see DXA output.
void material(String strToSetMaterial); // sets the material to a specific string, see DXA output.
void model(String strToSetMaterial); // sets the model to a specific string, see DXA output.
void dxaout(String strKey, String strValue); // sets the key, value couple in the DXA of the entity, see DXA output.
```

For a description on the `eraseInstance` function see section [Limited use](#). This function can also be used [during insert](#).

```
void eraseInstance();
```

There are also functions involved in assigning the Tsl instance to the group (see also [Change Instance group](#)).

```
assignToLayer(String strLayerName);
assignToGroups(Entity ent);
assignToElementGroup(...);
assignToElementFloorGroup(...);
```

To find the color index of a layer one can use the routine:

```
int colorFromLayer(String strLayerName);
```

To influence and query the number of [execution loops](#).

```
int bLastExecutionLoop();
setExecutionLoops(int nCount);
setDependencyOnBeamLength(Beam bm);

setDependencyOnEntity(Entity ent); // see Entity
setDependencyOnDictObject(DictObject object); // see DictObject and Execution
```

To set the diameter of the 3-circle graphic representation

```
setMarbleDiameter(double dDiam);
```

To fire/[spawn](#) the execution of another program

```
int spawn(String strLocale, String strExe, String strArg1, String strArg2);
int spawn_detach(String strLocale, String strExe, String strArg1, String strArg2);
```

Find a file

```
String findFile(String strFile); // (added hsbCAD15.0.12)
```

[S-type](#) specific routines

```
setSubAssemblyName(String strName);
String subAssemblyName() const;
```

To define custom [context menu](#) action. The `strTriggerKey` will appear as `_kExecuteKey` during execution.

```
addRecalcTrigger(int nTriggerType, String strTriggerKey);
```

To call a .Net function (see [dotNet](#) call):

```
String[] callDotNetFunction1(String strDIIPath, String strClassName, String strFunction);
String[] callDotNetFunction1(String strDIIPath, String strClassName, String strFunction, String[]
arArguments);
```

To add a [DimRequest](#) to the shop drawing framework:

```
void::addDimRequest(DimRequest dimRequest);
```

To [push](#) a command on the autocad command stack:

```
void pushCommandOnCommandStack(String strCommand); // (added hsbCAD16.0.11)
```

To create a new GUID

```
String createNewGuid(); // added hsbCAD v20.3.47 and v21.0.64
```

In [Viewport](#) there are a few static methods:

```
static int switchToModelSpace(); // return TRUE if successful, added build v21.3.50,  
v22.0.35  
static int switchToPaperSpace(); // return TRUE if successful, added build v21.3.50,  
v22.0.35  
static int inLayoutTab(); // return TRUE if in a layout tab, return FALSE if in a model tab,  
added build v21.4.41, v22.0.74  
// look also to Layout for currentLayout name.  
static int inPaperSpace(); // return TRUE if in a layout tab, and not inside a viewport,  
added build v21.4.41, v22.0.74
```

Similar to that we now have getVarInt, getVarDouble and getVarString:

```
int getVarInt(String strVar); // (added V27.5.1, V28.2.4)  
double getVarDouble(String strVar); // (added V27.5.1, V28.2.4)  
String getVarString(String strVar); // (added V27.5.1, V28.2.4)
```

---

[Example showing posnumCriteria:

```
Map mp = posnumCriteria();  
String keys[0];  
keys.append("");  
for (int i = 0; i < mp.length(); i++)  
{  
    keys.append(mp.keyAt(i));  
    reportMessage(TN("Value of " + mp.keyAt(i) + " is " +  
mp.getInt(i)));  
}  
  
PropString pName(0, keys, T("Variable name"));  
  
String arSYesNo[] = {T("|Yes|"), T("|No|")};  
int arNYesNo[] = {_kYes, _kNo};  
PropString pValue(1, arSYesNo, T("Use variable"));  
  
int nValNew = (pValue == arSYesNo[0]);  
int nValCurrent = mp.getInt(pName);  
  
if (pName != "")  
{  
    if (nValNew != nValCurrent)  
    {  
        Map mpNew;  
        mpNew.setInt(pName, nValNew);  
        setPosnumCriteria(mpNew);  
        reportMessage(TN("New value of " + pName + " set to " +  
nValNew));  
    }  
    else  
    {
```

```

        reportMessage(TN("Value of " + pName + " remains " +
nValCurrent));
    }
}

—end sample]

```

## 5.2 Metal parts declarations

The language contains a type to describe a metal part: MetalPart. Whenever constructed inside the script, the solid representation of the metal part is used in the graphical representation of the TSL instance in the drawing. This means, whenever a metal part is constructed, it is shown in the drawing. This is in contrast with the [Body](#) type.

Here is a comment to clarify the difference between Metalpart and Body. The main difference is that the solid that is represented by a Body is only there during the TSL execution, while the Metalpart will become the TSL. So Metalpart is persistent. Metalpart is the thing that

- will be used during the solid comparison during posnum assignment
- it is outputted by its volume,... to [dxa](#)

In contrast to that, Body is only a geometrical thing, like Line, Plane,...

There exist several different constructors:

- to build a cylindrical shape;
- to build a polyline extrusion;
- to build a box shape.

Metal parts can be numbered. Two metal parts will get the same number if their solids have the same shape.

The MetalPart type is actually a reference into a list of solids that belongs to the TSL instance. Therefor, whenever the MetalPart is copied (eg.: MetalPart mp1; MetalPart mp2 = mp1;) the solid itself is not copied, rather the reference is copied. To do a real solid copy, one should use the copyPart function.

```

class MetalPart {

    MetalPart(Point3d ptStart, Point3d ptEnd, double dRadius);
    MetalPart(PLine plShape, Vector3d vecExtrude, double dOffset = 1);

    MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
    MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX,
              double dLenY, double dLenZ);
    MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX,
              double dLenY, double dLenZ, double dXFlag, double dYFlag, double dZFlag);

    MetalPart(Body body);

    transformBy(CoordSys csTransformationMatrix);
}

```

```

    transformBy(Vector vecTranslate);

int addTool(Tool tl); // returns success of operation since build 17.1.26 and 18.0.17

addPart(MetalPart mpOther);
subPart(MetalPart mpOther);
copyPart(MetalPart mpOther);

Body body(); // added since V21.4.95, V22.1.72 and V23.0.45
};

```

---

**MetalPart**(**Point3d** ptStart, **Point3d** ptEnd, **double** dRadius);

Construct a new metal part with a cylindrical shape. Both ptStart and ptEnd are positioned at the axis line.

**MetalPart**(**PLine** plShape, **Vector3d** vecExtrude, **double** dOffset = 1);

Construct a new metal part as an extrusion of a polyline. The third parameter is optional, and has a default value of 1. This parameter expresses the the position of the polyline, with regard to the final shape, in the vecExtrude direction. If dOffset==1, the plShape is at the beginning of the body. If dOffset==0, the plShape is at the middle. dOffset can have any value.

The vecExtrude is a vector of which the length is used as extrusion length.

```

MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double
    dLenY, double dLenZ);
MetalPart(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dLenX, double
    dLenY, double dLenZ, double dXFlag, double dYFlag, double dZFlag);

```

Construct a new metal part with a box shape. To specify the dimensions, the length of the vector is multiplied with the length factor in that direction. Eg.: the height of the box is the same as **vecY.length()\*dLenY**. If **vecY** is a unit-length vector, then the **dLenY** corresponds with the height. If **dLenY** equals 1, then the length of the vector expresses the height.

The flags dXFlag, dYFlag and dZFlag, specify the position of the ptOrg inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point ptOrg is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -vecX, -vecY and -vecZ direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the vecX, vecY and vecZ coordinate system with origin ptOrg.

**MetalPart**(**Body** body);

Construct a new metal part from an existing solid of type Body (see [Body](#)).

**transformBy**(**CoordSys** csTransformationMatrix);

---

`transformBy(Vector vecTranslate);`

Apply a coordinate transformation to the solid shape.

`addTool(Tool tl);`

All tools that can be added to beams, can also be added to a solid shape, eg.: Drill, Cut, BeamCut,... Whenever the addTool is called, the solid operation on the metalpart itself is performed.

`addPart(MetalPart mpOther);`

To join different metalparts together, the function addPart can be called. Doing this, the mpOther will be destroyed, and set to NULL.

`subPart(MetalPart mpOther);`

To subtract mpOther from the current metalpart, the function subPart can be called. Doing this, the mpOther will be destroyed, and set to NULL.

`copyPart(MetalPart mpOther);`

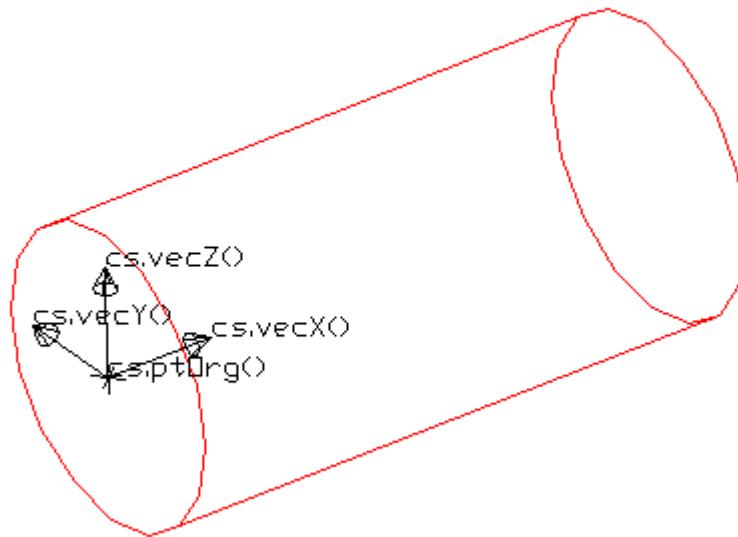
Take a copy of an existing metalpart. The mpOther solid will not be changed, and stay intact. In order to take a copy of a solid part, one needs to use this copyPart function. The assignment will not do this. The assignment is actually only a copy of the reference. After an assignment, both metalpart variables will refer to the same solid in the list of solids that belongs to the TSL instance. Calling the copyPart function, will make the reference refer to its own copy.

[Example O-type:

```
Unit(1, "mm"); // use mm as units

double dLength = U(400);
double dRadius = U(100);

CoordSys cs(_Pt0,_XU,_YU,_ZU); // define a cs as UCS at time of insertion
cs.vis(7); // visualize the coordSys only in debug
MetalPart mp2( Point3d(0,0,0), Point3d(dLength,0,0), dRadius ); // construct a metalpart in
the origin
mp2.transformBy(cs); // transformBy the metalpart
```



—end example]

---

[Example O-type:

```

Unit(1,"mm"); // use mm as units

double dLength = U(400);
double dRadius = U(100);

CoordSys cs(_Pt0,_XU,_YU,_ZU); // define a cs as UCS at time of insertion
cs.vis(7); // visualize the coordSys only in debug
MetalPart mp2( Point3d(0,0,0), Point3d(dLength,0,0), dRadius ); // construct a metalpart in
the origin
mp2.transformBy(cs); // transformBy the metalpart

MetalPart mp1;
mp1.copyPart(mp2);
// -- the following are two equivalent alternatives.
if (FALSE) {
    CoordSys csMove;
    csMove.setToTranslation(U(80)*_YU);
    mp2.transformBy(csMove); // move the metalpart
}
else {
    mp2.transformBy(U(80)*_YU); // move the metalpart
}

MetalPart mp3;
mp3.copyPart(mp1);
CoordSys csRot;
csRot.setToRotation(U(80),_ZU,_Pt0+0.5*dLength*_XU);

```

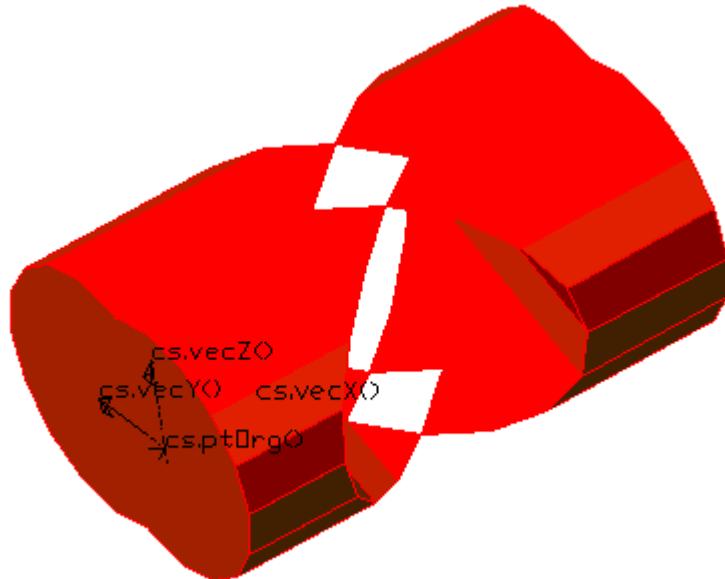
---

```

mp3.transformBy(csRot); // transformBy the metalpart

mp1.addPart(mp2); // make one
mp1.subPart(mp3);

```



—end example]

### 5.3 Extra Grip/Insert points

The toolscript instance can have a number of grippoints. These points can be accessed through the array `_PtG[]`:

This is a predefined array, declared as

**Point3d \_PtG[];**

The number of points that the user is asked for, during insertion is dependent on the type, and the number of extra grippoints set in the script definition.

For the T, X and G type, the `_Pt0` is defined by the intersection of the beams. This is not the case for a E and O type. The E and O type need the input from the user to determine the location of `_Pt0`. So for these 2 types, the first point that the user is asked for, is the `_Pt0`.

The name of the last grippoint that is moved is stored in a predefined variable called `_kNameLastChangedProp`. This value will also indicate the [last property changed](#). The name of the grippoint is `_PtG0`, `_PtG1...` or `_Pt0`.

#### How to draw temporary graphics while grip point dragging.

Since hsbcad V23.0.103, and V22.1.108, one can register to recalculate the tsl while dragging the grip point. While dragging a copy of the tsl instance is created. This copy does not become database resident. It only exists during one mouse move, but it can generate graphics to be shown

on that mouse move. This recalculation of this tsl should not try to modify the autocad database. Eg if you create an autocad layer (by assigning the tsl entity to a group), autocad might crash.

Normally the tsl will not be recalculated while dragging, unless you register for it, eg:

```
addRecalcTrigger(_kGripPointDrag, "_PtG2");
or
addRecalcTrigger(_kGripPointDrag, "_Pt0");
```

If registered, the tsl will execute with the state `_bOnGripPointDrag` set to TRUE. and

`_kExecuteKey` equal to the corresponding grippoint eg `"_PtG2"`.

```
if (_bOnGripPointDrag && _kExecuteKey=="_PtG2")
{
    ...
    return;
}
```

See example below.

---

[Sample of O-type and insert done in script (thanks to Roberto Hallo). It illustrates the use of `_kNameLastChangedProp`. It also illustrates the complexity involved in properties manipulating grip points, and grip points manipulating properties.

```
PropDouble dWidth (0, U(500,48), T("|View width|"));
PropDouble dHeight (1, U(1500,120), T("|View height|"));

if (_bOnInsert)
{
    showDialogOnce();
    _Pt0 = getPoint();

    //Adding Grips Points
    _PtG.append(_Pt0+_XU*dWidth*.5 + _YU*dHeight);
    _PtG.append(_Pt0+_XU*dWidth + _YU*dHeight*.5);
    _PtG.append(_Pt0+_XU*dWidth + _YU*dHeight);

    return;
}

//Control which grip point moves
if (_kNameLastChangedProp == "_PtG0")
{
    dHeight.set(_YU.dotProduct(_PtG[0] - _Pt0));
    _PtG[1] = _Pt0+ _YU*dHeight*.5 + _XU * dWidth;
    _PtG[2] = _Pt0+ _YU*dHeight + _XU * dWidth;
}

if (_kNameLastChangedProp == "_PtG1")
{
    dWidth.set(_XU.dotProduct(_PtG[1] - _Pt0));
    _PtG[0] = _Pt0+ _YU*dHeight + _XU * dWidth*.5;
```

```

    _PtG[2] = _Pt0+ _YU*dHeight +_XU * dWidth;

}

if (_kNameLastChangedProp == "_PtG2")
{
    dWidth.set( _XU.dotProduct(_PtG[2] - _Pt0));
    dHeight.set( _YU.dotProduct(_PtG[2] - _Pt0));

    _PtG[0] = _Pt0+ _YU*dHeight +_XU * dWidth*.5;
    _PtG[1] = _Pt0+ _YU*dHeight*.5 +_XU * dWidth;

}

//Control if the properties are modified
if (_kNameLastChangedProp == T("|View height|"))
{
    _PtG[0] = _Pt0+ _YU*dHeight +_XU * dWidth*.5;
    _PtG[1] = _Pt0+ _YU*dHeight*.5 +_XU * dWidth;
    _PtG[2] = _Pt0+ _YU*dHeight +_XU * dWidth;

}

if (_kNameLastChangedProp == T("|View width|"))
{
    _PtG[0] = _Pt0+ _YU*dHeight +_XU * dWidth*.5;
    _PtG[1] = _Pt0+ _YU*dHeight*.5 +_XU * dWidth;
    _PtG[2] = _Pt0+ _YU*dHeight +_XU * dWidth;

}

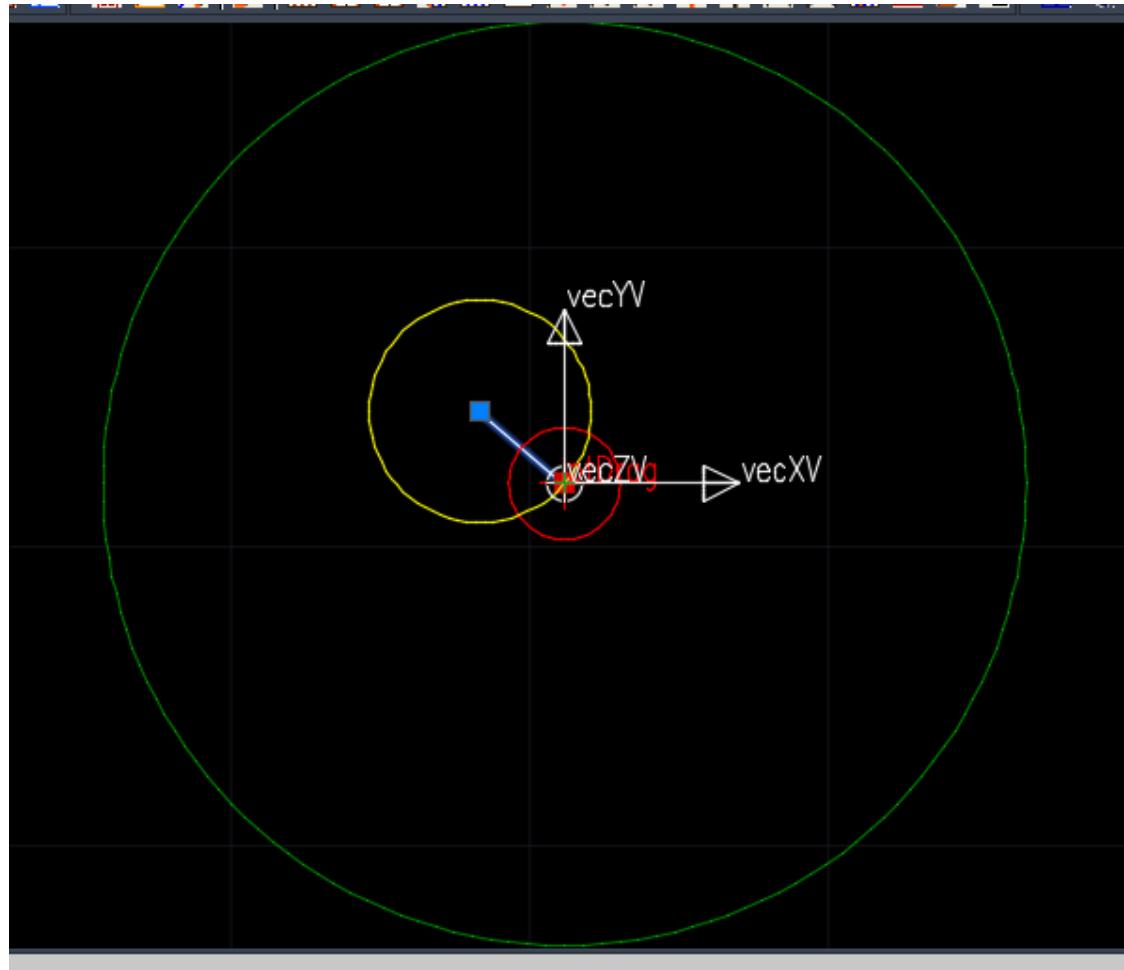
// draw rectangle
Display dp(-1);
dp.textHeight(U(10));
dp.draw(LineSeg(_Pt0, _Pt0+_XU*dWidth));
dp.draw(LineSeg(_Pt0+_XU*dWidth, _Pt0+_XU*dWidth+_YU*dHeight));
dp.draw(LineSeg(_Pt0+_YU*dHeight, _Pt0+_XU*dWidth+_YU*dHeight));
dp.draw(LineSeg(_Pt0+_YU*dHeight, _Pt0));

—end sample]

```

---

[Sample of O-type that shows temporary graphics while dragging a grip point.



```

Unit(1, "mm");
reportMessage("\n" + scriptName() + ", execute key: " + _kExecuteKey
+ ", bOnGripPointDrag: " + _bOnGripPointDrag);

if (_bOnGripPointDrag && (_kExecuteKey=="_PtG0" ||
_kExecuteKey=="_Pt0"))
{
    Point3d ptBase = _Pt0;
    Point3d ptDrag = _PtG[0];

    _ThisInst.setDebug(TRUE); // sets the debug state on the dragging
    TslInstance only, if you want to see the effect of vis methods
    ptDrag.vis(1);

    Vector3d vecXV = getViewDirection(0);
    vecXV.vis(ptDrag);
    Vector3d vecYV = getViewDirection(1);
    vecYV.vis(ptDrag);
    Vector3d vecZV = getViewDirection(2);
    vecZV.vis(ptDrag);

    double dRad = Vector3d(ptDrag - ptBase).length();
}

```

```

    Display dpJ(1);
    PLine plCir;
    plCir.createCircle(ptDrag, _ZU, dRad/2);
    dpJ.draw(plCir);

    dpJ.color(2);
    plCir.createCircle(ptBase, _ZU, dRad);
    dpJ.draw(plCir);

    dpJ.color(3);
    dpJ.transparency(50);
    plCir.createCircle(ptDrag, vecZV, 0.5*getViewHeight());
    dpJ.draw(plCir);

    return;
}

if (_bOnInsert)
{
    _Pt0 = getPoint(T("\n|Select start point|"));
    Point3d ptLast = _Pt0;

    PrPoint ssP2(T("|Select second point|"), ptLast);
    if (ssP2.go())
    {
        _PtG.append(ssP2.value());
    }

    return;
}

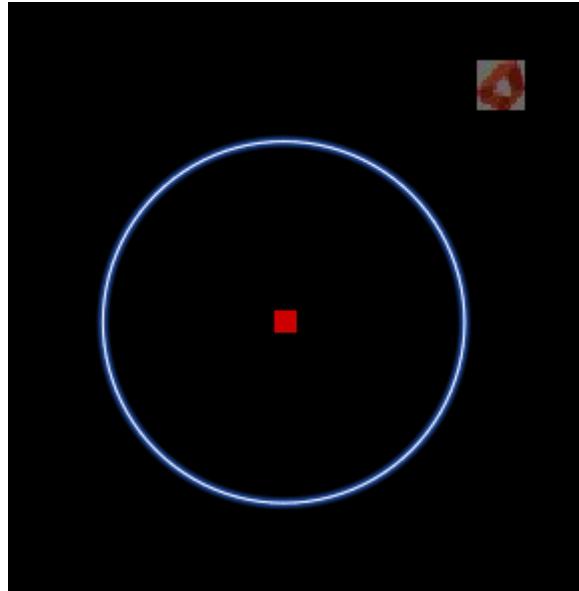
addRecalcTrigger(_kGripPointDrag, "_PtG0");
addRecalcTrigger(_kGripPointDrag, "_Pt0");

// normal database resident behavior, just draw something
Display dp(-1);
for (int i = 0; i < _PtG.length(); i++)
{
    dp.draw(PLine(_Pt0, _PtG[i]));
}
PLine plCir;
plCir.createCircle(_Pt0, _ZU, U(10));
dp.draw(plCir);
_Pt0.vis(1);

—end sample]

```

[Sample of O-type that shows temporary image graphics at cursor



```

Unit(1, "mm");

if (_bOnGripPointDrag && _kExecuteKey=="_Pt0")
{
    String strFile = _kPathHsbInstall + "\\Abbund\\Hsb.ico";
    String strImg = strFile.encodeImageFromFile();
    double dHOverW = strImg.encodedImageHeightOverWidth();

    Vector3d vecXV = getViewDirection(0);
    Vector3d vecYV = getViewDirection(1);

    double dSizeInX = 0.05 * getViewHeight();
    double dSizeInY = dHOverW * dSizeInX;

    Display dpJ(1);
    dpJ.transparency(50); // superimpose 50% transparency
    dpJ.drawImage(strImg, _Pt0, vecXV, vecYV, 1.5, 1.5, dSizeInX,
dSizeInY);

    return;
}

if (_bOnInsert)
{
    _Pt0 = getPoint();
    return;
}

addRecalcTrigger(_kGripPointDrag, "_Pt0");

// normal database resident behavior, just draw something
Display dp(-1);
PLine plCir;
plCir.createCircle(_Pt0, _ZU, U(10));

```

```
dp.draw(plCir);

—end sample]
```

## 5.4 Conversion to Hsb-units

To specify a value of a length, a diameter, a gap distance,... one often want to enter a value with a certain unit. In TSL this can be done using the Unit constructor. The Unit, or abbreviated to U, will convert the argument to the Hsb drawing units (specified in the Hsb\_Settings). The following constructors are available.

- 1 `Unit(String strValue, String strUnit, int LUNIT); // Unit("1' 6","inch",3);`  
`Unit(double dValue, String strUnit, int LUNIT); // Unit(2.34, "inch",3);`
- 2 `Unit(String strValue, String strUnit); // Unit("3 1/2","inch");`  
`Unit(double dValue, String strUnit); // Unit(1, "inch");`
- 3 `Unit(String strValue); // Unit("2");`  
`Unit(double dValue); // Unit(1);`
- 4 `Unit(double dValueMm, double dValueInch); // Unit(25,1);`  
`Unit(double dValueMm, String strValueInch); // Unit(50,"2");`

The type name Unit can be abbreviated to U, so the following are also valid constructors:

- 1 `U(String strValue, String strUnit, int LUNIT); // U("1' 6","inch",3);`  
`U(double dValue, String strUnit, int LUNIT); // U(2.34, "inch",3);`
- 2 `U(String strValue, String strUnit); // U("3 1/2","inch");`  
`U(double dValue, String strUnit); // U(1, "inch");`
- 3 `U(String strValue); // U("2");`  
`U(double dValue); // U(1);`
- 4 `U(double dValueMm, double dValueInch); // U(25,1);`  
`U(double dValueMm, String strValueInch); // U(50,"2");`

A value can be specified as a double dValue or as a string, strValue. In case the string is used, the string needs to be converted to a value.

The LUNIT value has the function of the AutoCad LUNIT value: how to convert the string into a number (see Autocad help for more information). If LUNIT is not specified in the Unit constructor (2,3,4), the previously used value is used. If LUNIT was not previously used in the script, the Autocad value is used.

If the strUnit is not specified in the Unit constructor (3,4), the previously used value is used. So it is important to specify at least once the units that are used in the script. The following values are allowed: "mm", "cm", "m", "inch", "feet".

The fourth constructor (4) always has a value in "mm", and a value in "inch", independent of the current strUnit value. Depending on the current Hsb drawing units, the value in "mm" or the value in "inch" will be used. For scripts that are applicable for drawings in mm and inches, it is convenient to specify a default value of for instance a property in both mm and inch.

---

[Example any-type:

```
double d1 = U(1,"mm",3); reportMessage(" U(1)=" + d1);
double d2 = U(500,"1 1/2"); reportMessage(" U(500,1 1/2)=" + d2);
double d3 = U(25,1); reportMessage(" U(25,1)=" + d3);

PropDouble pD(0, U(25,1), "Offset value");

double dList[] = { U(25,1), U(65,"2 1/2"), U(100,4) };
PropDouble pDL(1, dList, "List value");
```

The above example will output in a "mm" drawing:

*U(1)=1*

*U(500,1 1/2)=500*

*U(25,1)=25*

while the List value can have 25, 65, 100 as value.

In a "inch" drawing

*U(1)=0.0393701*

*U(500,1 1/2)=18*

*U(25,1)=1*

while the List value can have 1, 2.5, 4 as value.

—end example]

---

[Q&A 1: How do I process the square root of a surface value ?

When you have a surface area: 2 mm x 2 mm = 4 mm<sup>2</sup>, and you want to square-root it, you can either do U(sqrt(4)) or sqrt(U(2)\*U(2)).

So U(sqrt(4)) == sqrt(U(2)\*U(2)) == sqrt(U(4)\*U(1)) != sqrt(U(4))

Because U(1) == U(1,"mm") could be equal to 0.1 if your drawing unit is cm.

—Q&A]

---

## 5.5 Declaration of Shop drawing

```
View name(<origin>,<Xaxis>,<Yaxis>,<xpos shop drawing>,<ypos shop drawing>
, <xoffset dimension lines>,<yoffset dimension lines>,<int_dimtype>);
// if dimtype=0: no auto dimensions
// if dimtype=1: only max
// if dimtype=2: full dimensions
```

## 5.6 Compare instances and posnum generation

With the function called "setCompareKey" a string can be assigned that sets the equality condition during the posnum generation.

```
void setCompareKey(String strToSet); // sets a compare key for posnum assignment
```

With the function called "setCompareKey" a string can be assigned that sets the equality condition during the posnum generation.

```
void TslInst::setDependencyOnPosnumChanged(int bSet); // sets to TRUE to make the TslInst
recalculate on the event of its posnum being changed
```

The following are important statements:

- If the compare keys of 2 tsi instances are different they will have different posnum values.
- If no key is set, the geometry is taken as comparison condition. Only MetalPart geometry is taken into account.
- If there is no associated geometry (no metalpart defined) and no compareKey set, the pieces are considered different, and are given a unique posnum.

eg.:

```
setCompareKey("123abc");
setCompareKey(_Pt0); // any variable can be used, because it will be automatically
converted to a string.
setCompareKey(scriptName()); // use the scriptName as compare key
```

Since hsbCAD2011 (build 16.0.41) and hsbCAD2010 (build 15.3.22) posnum generation of Tsi's has been enhanced a bit.

- The TslInst now has a method to trigger recalculation if the posnum changes:  
setDependencyOnPosnumChanged.
- When GenBeams and Tsi instances are selected to assign posnums, the beams are processed first, then their reactions, and then the Tsi's.

If you want the Tsi to be dependent on the posnum of a Beam, you need to:

- Add the Beam to the \_Entity array. Otherwise the setDependencyOnEntity has no effect.
- Call the setDependencyOnEntity on the Beam.

If the Tsi is displaying its own posnum, you need to call the setDependencyOnPosnumChanged.

[Example of complex interaction of beam and tsl posnum: The posnum of the TsInst is dependent on the posnum of the beam.

```

if (_bOnInsert) {
    _Entity.append(getEntity());
    _Pt0 = getPoint();
    return;
}

if (_Entity.length()==0) return;
Entity ent = _Entity[0];

int nBeamPosnum = -2; // not a beam
Beam bm = (Beam)ent;
if (bm.bIsValid()) {
    // it is a beam
    nBeamPosnum= bm.posnum(); // will be -1 if posnum not set
}

String strText = scriptName() +"("+_ThisInst.posnum()+"":
"+ent.handle() + " with posnum "+nBeamPosnum;
reportNotice("\n"+strText);

// by setting the dependency on entity, which might be a beam, this
// tsl is recalculated if the beam's posnum is modified
setDependencyOnEntity(_Entity[0]);

// because this tsl outputs its own posnum, it should be recalculated
// if its own posnum changes
_ThisInst.setDependencyOnPosnumChanged(TRUE);

// make the posnum of this tsl dependent on the posnum of the beam
setCompareKey(nBeamPosnum);

// output my text
Display dp(-1);
dp.draw(strText,_Pt0,_XU,_YU,-1,0);

—end example]

```

## 5.7 DXA output

DXA or DXA !

The term DXA is often used as hsb export format. It can refer to the hsb export for Access, Excel or independent element export format (through hsb\_store3d command). Some of these files have "dxa" as extension, others have "dxx", "dxi", "axe", or "dat" as extension, but all have internally a DXA-structure as format. A DXA-structure is a key-value list. Each key and value is outputted on one line in the file (so '\n' is the separator).

Element-dxa refers to the file that is generated for a floor, wall or roof element, through the command hsb\_store3d.

A TSL instance has different data sets that can be exported: instance-data (metalpart or subassembly data), additional objects, graphics, hardware, and element tools including the construction directives. Each of these data sets is treated separately, with its own toggles and routines to allow or prevent appearance inside the export file.

Here we focus on the instance data and the additional objects. The instance data appears in the export file as object EMETAL, object METALPART or object SUBASSEMBLY. The object keys EMETAL and METALPART are used for E,T,G,X and O types, while the object key SUBASSEMBLY is used for the TSL S type.

The TSL script definition has a special toggle to allow/prevent TSL instances to appear in any export file (see [the editor window](#), field G). This toggle is only used for the instance data. The toggle is active for Excel, Access as well as element export. It is a global toggle for all instances of a particular script. For Excel and Access export, this is the only toggle that is checked before output is generated.

For element-dxa output, this is not the only toggle that needs to be set to allow export. The instance data is only exported to an element-dxa if the element is flagged with the routine

**void exportWithElementDxa(Element el); // sets the element to export the TSL with**  
This is also the default behaviour for the E,T,G,X and O types. The S type is exported with the element of the primary beam by default.

A tsl instance can only be exported with one element. Otherwise it would appear in the bill of material multiple times. So only the last call of exportWithElementDxa will set the element with which the instance data of the tsl is exported to the element-dxa. If the element is an invalid element, the reference is reset, and the tsl will not be exported with any of the elements. Also the tsl instance should be attached to the element. For a tsl to be attached to the element, the element should appear in the array \_Element of the tsl.

For dxi output, also the global script toggle needs to allow export. In addition, the instance data is only exported to a dxi if the tsl is flagged with the routine

**void exportToDxi(int bSet); // if bSet == TRUE, then output with dxi**  
Setting the value to TRUE, means that the tsl instance will be exported with the dxi of the floor that it belongs to. So this is the third condition in order for the TSL to appear in the DXI: the tsl instance must belong to the floor group of which the dxi is made.

The script can add key-value couples to the dxa-out of the metalpart. This can be done in 3 ways:

```
void material(String strValue); // sets the material to a specific string
void model(String strValue); // sets the model to a specific string
void dxaout(String strKey, String strValue); // sets the key, value couple in the DXA of the entity
```

**material(String strValue)**

will appear as

MATERIAL  
strValue

**model(String strValue)**

will appear as

MODEL  
strValue

**dxaout(String strKey, String strValue)**  
will appear as  
U\_STRKEY  
strValue

Both MATERIAL and MODEL keys will appear only once in the dxa-structure. The last strValue set will be used. There is no uniqueness condition on the user keys U\_STRKEY.

Together with the instance data, a list of additional objects can be exported to the file. This is meant for hsb-internal usage only. Objects can be collected by calling the routine

```
void dxaExportObject(String strObject, Map map);
void dxaExportObject(String strObject, Map map, Element elToExportWith, int
bExportToDxi);
```

The map needs to contain the complete and correct object description that is needed in the file. Be careful with the coordinate system that the object is specified in. By default the elToExportWith is not referring to any element, and the bExportToDxi is set to FALSE. One can do multiple calls to the dxaExportObject, resulting in multiple objects in the file. The objects will appear in the file if the element matches, or if the bExportToDxi is set, and exporting to dxi is active. The same conditions need to be fulfilled as with the [exportWithElementDxa](#) as [exportToDxi](#).

-----  
Other related issues are:

- Graphics is exported independent of the tsl instance to the element-dxa file, and not to the other export files.
- Elem tools (ElemDrill, ElemNail,...) are exported independent of the tsl instance to the element-dxa file, and not to the other export files.
- Construction directives and the ElemItems are exported to the floor-dxa file (dxi file), and not to the other export files.
- Hardware is exported independent of the tsl instance, but only to Access and Excel export.
- Excel output is numbered

## 5.8 Changing variables from inside the script

Whenever the script is executed, all predefined variables are reinitialized by the runtime environment. At the end of the script execution, there are however some variables that are read from the script, and stored back to the script entity, to be used later:

- All grip points: \_PtG[]
- The \_Pt0 for the O and E type
- The ECS coordinate system for internal comparison: \_XE, \_YE, \_ZE, \_PtE
- The entities that were added, or changed to the \_Element[], \_Beam[], \_Sheet[], \_Sip[], \_Viewport[], \_Opening[], \_Entity[] arrays.
- The \_Map key-value map.

The values of all other variables are lost at the end of the execution.

To have a script entity of E type that stays at  $\frac{3}{4}$  of the length of a beam, the following line can be used:

```
_Pt0 = _PtL0 + (0.25*_LMin0+0.75*_LMax0)*_XF0;
```

## 5.9 OPM functions

To define OPM (object property manager) available properties of the TSL instance, one of the following constructors can be used. A property is a variable that will be initialized during the first execution run, that can be changed by changing the value in the OPM, and that will keep this value during subsequent executions. So the initialValue will only be set during the first run.

A PropDouble can be used as a normal double in expressions. A PropInt can be used in expressions the same way as int type variables can. The same is true for the PropString type.

The Prop variable can also be constructed with an array of values, in which case, the property becomes an enumerated property. By default the first element of the array is the initial value of the enumerated property, unless the nInitialEnumIndex is indicating an other initial value.

The nPropertyIndexDouble is a unique index in the set of all PropDouble variables. All properties must have a unique index assigned. For the PropDouble, the index values 0,1,2,...,99 will be shown in the OPM. Higher values are also allowed, but will not be shown by default.

The index sequence determines the order in which they appear inside the OPM.

The nPropertyIndexInt, and the nPropertyIndexString in the range 0,1,...,7 define properties that are shown in the OPM.

When the value of 2 or more properties are dependent on each other, there is the difficulty of knowing which property was changed by the user. To overcome this difficulty, a predefined String variable is set with the name of the property that was last changed. This predefined is called `_kNameLastChangedProp`. This value will also indicate the [last grippoint moved](#). The name of the grippoint is `_PtG0,_PtG1,...` or `_Pt0`.

For the PropDouble type, the format in which the value is displayed is important. The format can be `_kLength` (default), `_kNoUnit`, `_kArea`, `_kVolume`, `_kAngle`.

Setting the read only state of a property, one can also set it to `_kHidden` (which is 2), or `_kReadOnly` (equivalent to 1 or TRUE). // added hsbCAD v23.3.10

### Remark

The OPM (object property manager) groups the selected entities by their name. This name is composed of the name of the TSL script, and extended with a string that can be set by the global function called: `setOPMKey`.

```
void setOPMKey(String strToSet)
```

A call of `setOPMKey("abc")` for a script with name "tube" generates the OPM name: "tube-abc".

It is recommended practice to make sure the OPM keys are different of TSL scripts that have a different set of OPM properties. This is due to a shortcoming in OPM: the set of parameters is not updated as often as is should.

**Controlling other properties**

```
void setControlsOtherProperties(int bSet);
    _bOnControlPropertyChanged
```

If a property is flagged setControlsOtherProperties(TRUE) there are a couple of scenarios:

- 1) The value changes in the property palette of autocad (OPM). In that case the tsl just recalculates, and updates the OPM.
  - 2) If such a property changes in the properties dialog, the tsl will fire with the event **\_bOnControlPropertyChanged** set to TRUE.
- 

```
class PropInt {

    PropInt name(int nPropertyIndexInt, int initialValue);
    PropInt name(int nPropertyIndexInt, int initialValue, String strOPMName);
    PropInt name(int nPropertyIndexInt, int[] strEnumArray);
    PropInt name(int nPropertyIndexInt, int[] strEnumArray, String strOPMName);
    PropInt name(int nPropertyIndexInt, int[] strEnumArray, String strOPMName, int
        nInitialEnumIndex);

    void set(int newValue);
    int setEnumValues(int[] strEnumArray); // added since V25.1.58 and V24.1.89. If empty array,
        property becomes none enumerated.

    void setReadOnly(int nReadOnly); // since v23.3.10 it also accepts _kHidden and _kReadOnly
    void setDescription(String strDesc);
    void setCategory(String strCat); // added hsbCAD v19.1.31 and hsbCAD v20.0.16
    void setControlsOtherProperties(int bSet); // since v27

    Map subMap(String strKey) const; // see Map (added V23.0.104)
    void setSubMap(String strKey, Map mapNew); // (added V23.0.104)
    String[] subMapKeys() const; // return list of available sub map keys (added V23.0.104)
    int removeSubMap(String strKey); // (added V23.0.104)

    (deprecated) setMap(Map map); // added hsbCAD v21.3.103 and hsbCAD v22
    (deprecated) Map getMap(); // added hsbCAD v21.3.103 and hsbCAD v22
};

class PropDouble {

    PropDouble name(int nPropertyIndexDouble, double initialValue);
    PropDouble name(int nPropertyIndexDouble, double initialValue, String strOPMName);
    PropDouble name(int nPropertyIndexDouble, double[] strEnumArray);
    PropDouble name(int nPropertyIndexDouble, double[] strEnumArray, String strOPMName);
    PropDouble name(int nPropertyIndexDouble, double[] strEnumArray, String strOPMName,
        int nInitialEnumIndex);

    void set(double newValue);
    double setEnumValues(double[] strEnumArray); // added since V25.1.58 and V24.1.89. If
        empty array, property becomes none enumerated.

    void setReadOnly(int nReadOnly); // since v23.3.10 it also accepts _kHidden and _kReadOnly
```

```

void setDescription(String strDesc);
void setCategory(String strCat); // added hsbCAD v19.1.31 and hsbCAD v20.0.16
void setControlsOtherProperties(int bSet); // since v26.5.5

Map subMap(String strKey) const; // see Map (added V23.0.104)
void setSubMap(String strKey, Map mapNew); // (added V23.0.104)
String[] subMapKeys() const; // return list of available sub map keys (added V23.0.104)
int removeSubMap(String strKey); // (added V23.0.104)

(deprecated) setMap(Map map); // added hsbCAD v21.3.103 and hsbCAD v22
(deprecated) Map getMap(); // added hsbCAD v21.3.103 and hsbCAD v22

void setFormat(int nFormat); // The format can be _kLength (default), _kNoUnit, _kArea,
                            _kVolume, _kAngle.
};

class PropString {

    PropString name(int nPropertyIndexString, String initialValue);
    PropString name(int nPropertyIndexString, String initialValue, String strOPMName);
    PropString name(int nPropertyIndexString, String[] strEnumArray);
    PropString name(int nPropertyIndexString, String[] strEnumArray, String strOPMName);
    PropString name(int nPropertyIndexString, String[] strEnumArray, String strOPMName, int
                    nInitialEnumIndex);

    void set(String newValue);
    String setEnumValues(String[] strEnumArray); // added since V25.1.58 and V24.1.89. If empty
                                                array, property becomes none enumerated.

    void setReadOnly(int nReadOnly); // since v23.3.10 it also accepts _kHidden and _kReadOnly
    void setDescription(String strDesc);
    void setCategory(String strCat); // added hsbCAD v19.1.31 and hsbCAD v20.0.16
    void setControlsOtherProperties(int bSet); // since v26.5.5

    void setDefinesFormatting(String strEntityType); // added hsbCAD v24.1.11, see also
                                                Entity::formatObject. The strEntityType (eg "Entity" or "GenBeam") is used to
                                                generate a sample during editing.
    void setDefinesFormatting(String strEntityType, Map mapAdditionalVariables); // added
                                                hsbCAD v24.1.11
    void setDefinesFormatting(Entity entExample); // added hsbCAD v24.1.11, see also
                                                Entity::formatObject. The entExample is used to generate a sample during editing.
    void setDefinesFormatting(Entity entExample, Map mapAdditionalVariables); // added hsbCAD
                                                v24.1.11

    Map subMap(String strKey) const; // see Map (added V23.0.104)
    void setSubMap(String strKey, Map mapNew); // (added V23.0.104)
    String[] subMapKeys() const; // return list of available sub map keys (added V23.0.104)
    int removeSubMap(String strKey); // (added V23.0.104)

(deprecated) setMap(Map map); // added hsbCAD v21.3.103 and hsbCAD v22
(deprecated) Map getMap(); // added hsbCAD v21.3.103 and hsbCAD v22
};

```

```
PropDouble::set(double newValue);
PropInt::set(int newValue);
PropString::set(String newValue);
```

Properties values can also be changed during execution of the script by calling the set method.

```
void setReadOnly(int nReadOnly); // since v23.3.10 it also accepts _kHidden and _kReadOnly
```

When setReadOnly is called with parameter TRUE or **\_kReadOnly**, the property will be read only, at least inside the OPM. The default value is FALSE. One can also set it to **\_kHidden** (which is 2).

```
void setDescription(String strDesc);
```

This routine allows to set the description that will appear at the bottom of the OPM, when the property is selected.

```
void setCategory(String strCat);
```

The OPM shows the properties per category. The default category is "General" which is equivalent to leaving the category not set, or setting it to "". It is strongly recommended to NOT assign the category to "General", as working on a none English copy of Autocad, will create a separate "General" category.

```
void setFormat(int nFormat);
```

For the PropDouble type, the format in which the value is displayed is important. The format can be **\_kLength** (default), **\_kNoUnit**, **\_kArea**, **\_kVolume**, **\_kAngle**. See example below.

---

[Example O-type with 2 properties. One property value will always be the tripple of the other property value:

```
Unit(1,"mm");
PropDouble pD1(0,U(1),"One");
PropDouble pD2(1,U(3),"Three");

if ( (pD1*3) != pD2 ) { // one of the props is changed
    if (_kNameLastChangedProp=="One") { // the name of the last one changed
        pD2.set(pD1*3); // change the value of the property pD2
        reportMessage("\nProp named Three auto adjusted");
    }
    else {
        pD1.set(pD2/3); // change the value of the property pD1
        reportMessage("\nProp named One auto adjusted");
    }
}
```

```

        }
    }
—end example]

```

[Example O-type with 1 properties

```

String strFlip[] = {T("Not flipped"), T("Flipped")};
PropString prStrFlip(0, strFlip, T("Flip a coin"), 1); // initial value has index 1
—end example]

```

[Example O-type, and images from ADT4

```

U(1, "inch");

```

```

PropDouble pDouble2(2, atan(1.1), T("My PropDouble Angle"));
pDouble2.setDescription(T("PropDouble index 2"));
pDouble2.setFormat(_kAngle);

```

```

PropDouble pDouble (0, 0.12345, T("My PropDouble NoUnit"));
pDouble.setDescription(T("PropDouble index 0"));
pDouble.setFormat(_kNoUnit);

```

```

PropDouble pDouble1(1, U(0.12345), T("My PropDouble Length"));
pDouble1.setDescription(T("PropDouble index 1"));
pDouble1.setFormat(_kLength);

```

```

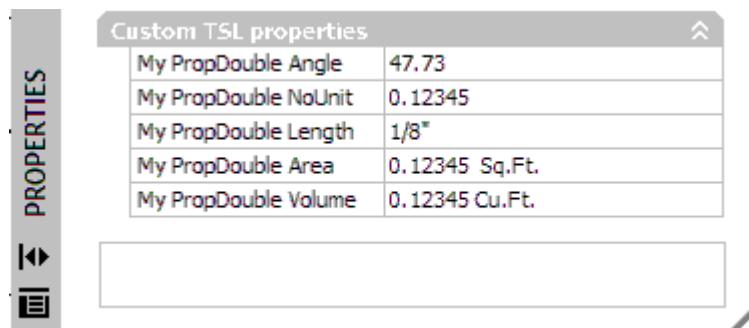
PropDouble pDouble3(3, U(0.12345*12)*U(12), T("My PropDouble Area"));
pDouble3.setDescription(T("PropDouble index 3"));
pDouble3.setFormat(_kArea);

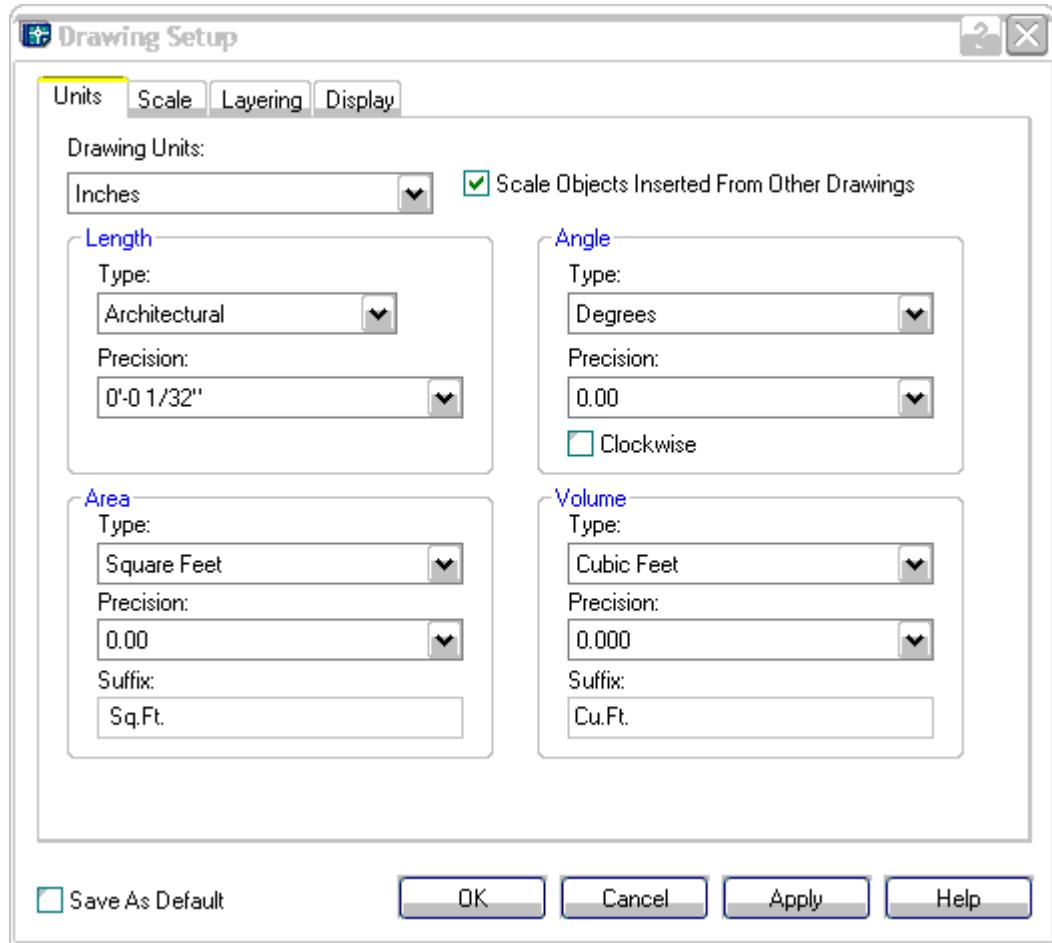
```

```

PropDouble pDouble4(4, U(0.12345*12)*U(12)*U(12), T("My PropDouble Volume"));
pDouble4.setDescription(T("PropDouble index 4"));
pDouble4.setFormat(_kVolume);

```





*—end example]*

*[Example O-type showing PropString with setDefinesFormatting*

```

if (_bOnInsert)
{
    _Pt0 = getPoint();
    _Entity.append(getEntity());
    return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}
Entity ent = _Entity[0];

Map mp();
mp.setString("akey", "aval");

String strLines[0];

```

```

Unit(1, "mm");
PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

PropInt ii(0, 2, T("|PropInt|"));

PropString ss2(1, "a format", T("|Format|"));
ss2.setDefinesFormatting(ent);
ss2.setDescription(T("|This is a format|"));
strLines.append("ss2: " + ent.formatObject(ss2));

PropString ss3(2, "a format", T("|Format2|"));
ss3.setDefinesFormatting("Sheet");
ss3.setDescription(T("|This is another format|"));
strLines.append("ss3: " + ent.formatObject(ss3));

PropString ss4(3, "a format", T("|Format|"));
ss4.setDefinesFormatting(ent, mp);
ss4.setDescription(T("|This is a format 3|"));
strLines.append("ss4: " + ent.formatObject(ss4, mp));

PropString ss5(4, "a format", T("|Format2|"));
ss5.setDefinesFormatting("Sheet", mp);
ss5.setDescription(T("|This is a format 4|"));
strLines.append("ss5: " + ent.formatObject(ss5, mp));

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1, 1);
}

```

*—end example]*

## 5.10 Limited use and automatic tool insertion

It is often the case that a particular script can only be applied if certain conditions are met. These conditions could be:

- The width or height of the timbers is in a certain range;
- The main axis (X-axis) of the timber has a certain direction;
- The orientation of the first beam (\_Beam0) with respect to the second beam (\_Beam1) has a certain angle;
- The location of the metal piece has a certain value, eg.: the height (z-coordinate) must be bigger than a certain value;
- ...

Properties of a beam can easily be queried, see [declaration of beam](#). The orientation of beams can be checked by applying the [direction test routines](#).

In the case these conditions are not met, one can do a number of actions:

- report a message in the Autocad text screen: using the [reportMessage](#) function;

- report a message in a pop up modeless dialog using the [reportNotice](#) function;
- report a message in a pop up dialog box: using the [reportWarning](#) function;
- report a message in a pop up dialog and end the execution of the script with the [reportError](#) function;
- just end the execution of the script using the [return](#) statement;
- erase the script entity from the drawing using the function [eraselInstance](#);

The function

[eraselInstance\(\)](#)

will fire an erase instruction from the autocad drawing database. The execution of the script will not stop however. The commands after the [eraselInstance](#) function will still be executed. To end the execution, the use of a return statement is recommended.

The [eraselInstance](#) function can also be used in combination with a huge selection set during insertion, or with the intelli-select for a T-type. Doing so, one could try to insert the script-entity in a lot of places, and let the script decide if all conditions are met for application of the script. If not, the script can be deleted using the [eraselInstance](#) function. The user would then see that the script is only inserted at these locations where the conditions are met. This is called automatic tool insertion.

[Example:

```
// only inserted if the beams are perpendicular
if (!\_X0.isPerpendicularTo\(\_X1\)) { // ! means NOT
    eraselInstance\(\);
    reportMessage("`n-> Tool automatic erased: beams not perpendicular");
    return;
}
```

—end example]

## 5.11 Change instance group/layer

When the TSL instance is inserted in the drawing, it becomes a drawing entity. Each drawing entity resides on a layer. As is the case with a normal ETenon, or an ESurfaceDrill, the layer of the TSL instance is set automatically to the layer of the primary beam ([\\_Beam0](#)). This is the case for E,G,T and X types. The O-type will not have its layer set automatically. The TSL can however overwrite this automatic behaviour, by explicitly assigning the instance to a layer/group using the [assignToLayer](#), [assignToGroups](#) or [assignToElementGroup](#) global function. For the O-type TSL, you need to call one of these functions in order to let the instance change its layer automatically.

The instance can be assigned to a particular layer with a given name. To do this the global function [assignToLayer](#) must be used. If the strLayerName is an invalid layer name (e.g. an empty string), the layer of the ts1 instance will not be changed, but the automatic layer assignment will be overwritten anyway. Doing so will give the cad user the control over the layer.

[assignToLayer\(String strLayerName\);](#)

The instance can be assigned to the same groups of an existing entity. To do this the global function [assignToGroups](#) must be used. The entity that is used can be a beam, sheet, element,...

[assignToGroups\(Entity ent\);](#)

[assignToGroups\(Entity ent, char cZoneCharacter\); // \(added since 18.1.45\)](#)

To assign the instance to the wall or roof element group, the function assignToElementGroup can be used. For an existing element (wall or roof), the element group might not exist yet. In that case, the instance will not be regrouped. The instance can also be added to the elements floor group with the assignToElementFloorGroup call.

```
assignToElementGroup(Element elem);
assignToElementGroup(Element elem, int bExclusive);
assignToElementGroup(Element elem, int bExclusive, int nZoneIndex, char
cZoneCharacter);
assignToElementFloorGroup(Element elem);
assignToElementFloorGroup(Element elem, int bExclusive);
assignToElementFloorGroup(Element elem, int bExclusive, int nZoneIndex, char
cZoneCharacter);
```

elem : the element that the instance will be assigned to  
bExclusive : normally an instance would only belong to one element: bExclusive is TRUE (is default)  
In some cases you want to specify if the instance belongs to multiple element groups.  
nZoneIndex : if specified, the zone index: (-5,-4,...4,5)  
cZoneCharacter : specifies the entity set; should be equal to 'Z' for general items, 'T' for beam tools, ('T' is default), 'E' for element tools

If the script adds different element tools to different zones, there is a difficulty, because the entity can only belong to one zone. Also, the entity must be put on a layer which is on, in order to be able to toggle the sub layers on which the element tools are visualized. Therefore it is needed that the tool is put on a layer that can be put on, but that essentially does not carry any visible items. The HSB-CAD software is designed that the E0 layer (cZoneCharacter 'E' with nZoneIndex 0) should be used for that. So if the script adds different element tools to different zones, the assignToElementGroup should be called with 'E' and 0.

One of the following situations is true

- nothing is specified in the script: For T,R,G and Xtypes, hsbCad is in control and will determine the layer depending on the beam of the tool.
- assignToLayer or assignToGroups or assignToElementGroup is called with a valid layer/entity/element: Now the Tsl-writer has control.
- assignToLayer is called with an empty layer name: The cad user has control, obviously the current layer plays an important role.

*[Example O-type:*

```
if (_bOnInsert){
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}
// check if TSL is attached to element, if not, do nothing
if (_Element.length() == 0) return; // 0 is a valid index now

// use the elToUse element reference further on
Element elToUse = _Element[0];
```

```
String arGroup[] = {"Same groups as element", "The special element group"};
PropString psGroup(0,arGroup,"Group");

if (psGroup==arGroup[0]) { // first option
    // assign to the same groups as the element belongs to
    assignToGroups(elToUse);
}
else {
    // assign to the element group if it exists
    assignToElementGroup(elToUse,TRUE,0,'E');
}

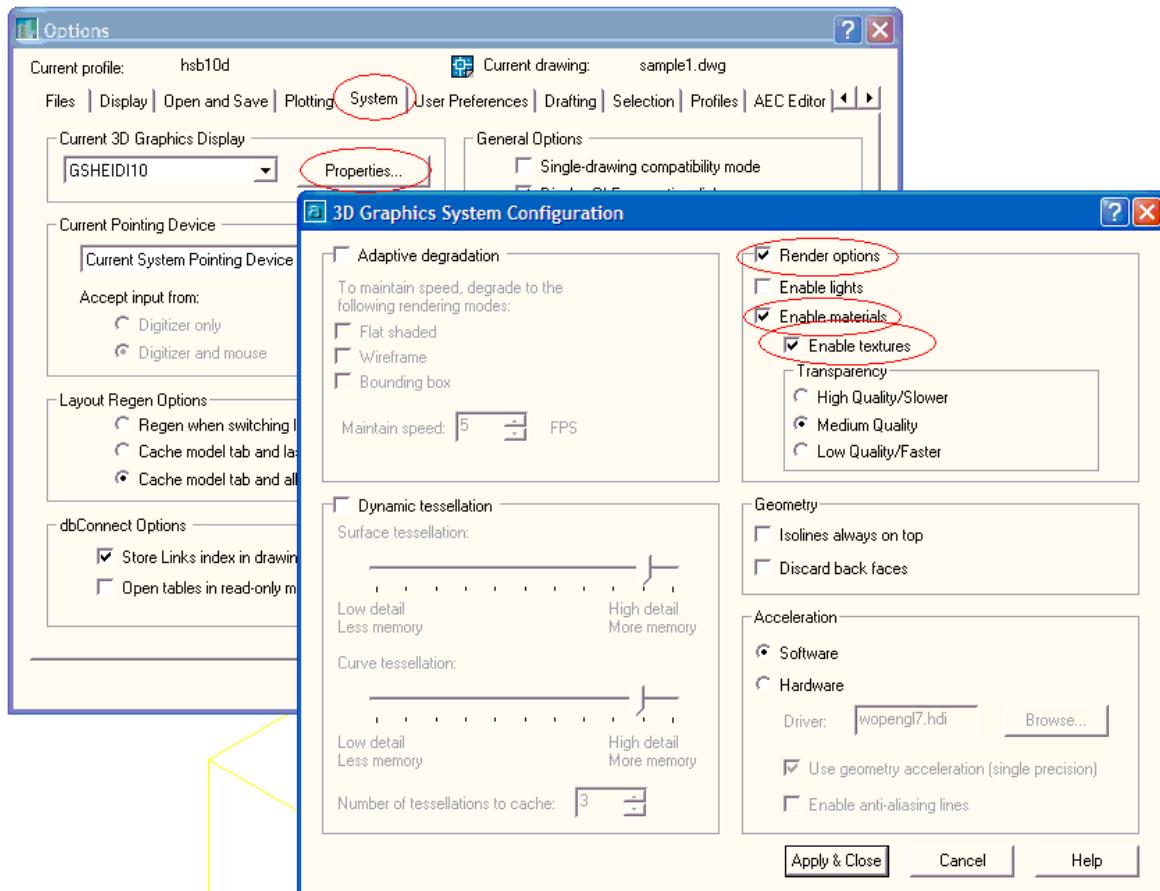
—end example]
```

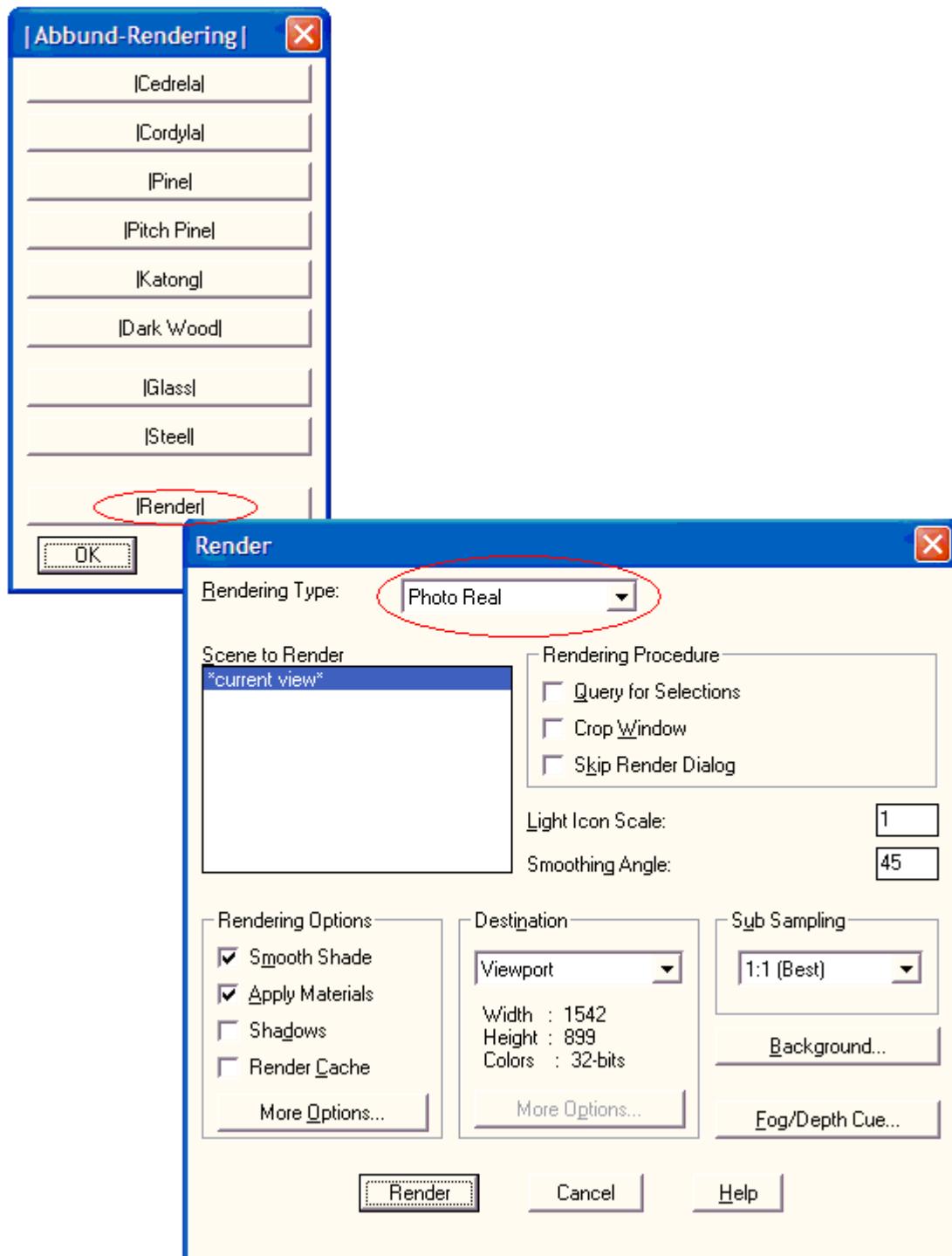
## 5.12 Thumbnails

In the TSL editor window, there is a button in the "Thumbnail bitmap" group to set the bitmap. When you press this button, the view from the preview window is captured, and stored as bitmap inside the definition. This bitmap is then later on used during TSL selection in the insert dialog, or in the file import dialog.

To use texture mapped beams inside the preview window, one should take the following steps:

- 1) Make sure the system settings of Autocad are such that textures are shown during rendered view (see figure 1).
- 2) Insert a TSL instance in the drawing.
- 3) Assign some textures to the beams of the TSL instance. To do that use the command Hsb\_MapWood, which is available through the menu, (see figure 2).
- 4) Then right click on the TSL instance, and choose "Edit TSL" from the context menu. After pressing "Preview" you can toggle the "Switch Render Mode" button until you see the textures appear (see figure 3).
- 5) If you press "Set" in the "Thumbnail bitmap" group, the preview window will get snapped.







## 5.13 Execution

Whenever something of the TSL instance changes, the TSL instance needs to be updated. This could be because the user changes an OPM value, or because a grippoint is moved, or because beams are added to the TSL instance,...

To update the TSL instance, the script is executed a number of times.

- Normally the script is only executed once.
- T,E,G,S and X types are executed twice, if certain variables or functions are used inside the script: \_LMin?, \_LMax?, \_Len?, dLMin, dLMax, ptCen, dL, setPtCtrl. The use of these variables suggests that the script is beam-length dependent. Because it is likely that the script adds an end cut to one of the beams, the script becomes dependent on its own execution. Therefor, it is executed twice.
- O type of script is not sensitive to the use of these variables.
- The number of executions can be set by calling the global function `setExecutionLoops`, for all types, O as well as T,E,G,S and X
- During insert, the script is always executed only once, whatever the value set with `setExecutionLoops`.

The following routines, and variable can be used, for instance to report messages during a certain run.

**\_kExecutionLoopCount:** is set to the internal loop counter. During the first run, the value is 0.

**int bLastExecutionLoop():** returns TRUE or FALSE, depending on being during the last execution run.

**setExecutionLoops(int nCount):** sets the number of execution loops that are required to update the TSL instance.

For T,E,G,S and X types, the TSL instance is assumed to be dependent on the beam length, if one of the above mentioned variables is used. If that is the case, the script will be updated if the length changes of one of its beams. For the O type, there is no such assumption. If this assumption is not satisfactory, or if it is an O type that needs to be dependent on the length of a certain beam, the global function `setDependencyOnBeamLength()` can be called for all the beams that the script needs to be dependent on.

The routine `setDependencyOnBeamLength` adds the beam `bm` to the list of beams that this TSL is dependent on. This dependency triggers the recalculation of the TSL if the length of the beam is changed.

`setDependencyOnBeamLength(Beam bm);`

To fire the execution of the Tsl instance whenever a specific entity in the drawing changes, one can use the routine `setDependencyOnEntity`. This reactor mechanism is not meant to be used on Beam, GenBeam, Sheet and Sip. It is actually meant to react on all entities but GenBeam derived ones. This is because the interaction between these GenBeam derived type of entities is much more close. E.g.: the tsl instance will typically add some tools to the GenBeam.

`setDependencyOnEntity(Entity ent);`

In order for the `setDependencyOnEntity` to work, one has to add the Entity `ent` to the `_Entity` array as well. Otherwise the reactor will not be added to the entity.

To fire the execution of the Tsl instance whenever a dictionary entry changes, one can use the routine `setDependencyOnDictObject`.

`setDependencyOnDictObject(DictObject object); // see DictObject`

Whenever a TSL is triggered for execution, the TSL will execute. During the first run, the internal value of "executionLoops" is set to 1. During the execution, the value of this "executionLoops" can be changed with the function `setExecutionLoops`.

## 5.14 Copy and erase with beams

The default behaviour of how the tsl instance reacts on erase and copy of beams, is depending on the type of the script. For the E and S type, the tsl instance is completely dependent on `_Beam0`. If this beam is copied, the tsl instance is copied. If the beam is erased, the tsl instance is erased.

For the T, X and G types, the tsl instance will be erased if either `_Beam0` or `_Beam1` is erased. During the copy, the tsl instance will only be added automatically to the selection if both beams are selected for copy. Btw, if the tsl instance is selected by the user, it stays in the selection set, even if only one of the beams is selected for copy.

The O type is independent of any beam. This means it will not be automatically erased with a beam, or copied with a beam.

The default behaviour of how the tsl instance copies and is erased with the beams can be overwritten. The behaviour can be overwritten with the function `setEraseAndCopyWithBeams`.

`setEraseAndCopyWithBeams(int nSet):`

The following predefines should be used.

- `_kNoBeams:` set to the default of O type
- `_kBeam0:` set to the default of E and S types
- `_kBeam01:` set to the default of T, X and G types

A word of caution is in order. The `_Beam0` (for E, T, X, and G) and the `_Beam1` (for T,X and G types) are needed for the calculation of the predefined values like `_X0`, `_Pt1`,...see [predefined variables](#). So scripts that use these predefined variables should not change the `eraseAndCopyWithBeams` status.

Besides the behaviour on what should happen when GenBeams that are referenced are erased or copied, there is also to decide on what should happen with the GenBeam references when this instance is copied. References to GenBeams that are also in the selection set are always kept. It is about the GenBeams which are not present in the copied set.

`setKeepReferenceToGenBeamDuringCopy(int nSet):` // added to hsbCAD20.0.68 and  
hsbCAD19.1.99

The following predefines should be used.

- `_kAuto:` behaviour is set by the `setEraseAndCopyWithBeams` method, default
- `_kNoBeams:` default for O type
- `_kBeam0:`
- `_kBeam01:`
- `_kAllBeams:` default of E, S, T, X and G types

Also on splitting of Beams, the behaviour can be set which corresponds to BeamCut behaviour.

```
setCloneDuringBeamSplit(int nSet); // added to hsbCAD22.1.61 and hsbCAD23
```

The following predefines should be used.

**\_kAuto:** behaviour is set by the default method

**\_kBeam0:** clone with Beam0 split, do not clone with any other beam. All other beam references are copied during split. Beam0 is substituted with split.

## 5.15 Tips using elements

When a TSL is attached to an element, the TSL often wants to do something with the beams or sheetings of the element.

Prior to hsbCAD2011, to get the beams or sheetings of an element, it was a rather expensive operation. Internally it involved the collection of all the layers that have element information, and a loop over all the entities in the Autocad database to find the entities that belong to one of these layers. The expensive functions were member functions of [Element](#), and are called beam, genBeam, sheet,... The performance of these methods has been drastically improved in hsbCAD2011, build 16.0.11.

```
GenBeam[] genBeam() const;
GenBeam[] genBeam(int nZoneIndex) const;
Beam[] beam() const;
Sheet[] sheet() const;
Sheet[] sheet(int nZoneIndex) const;
Sip[] sip() const;
```

Knowing that a TSL gets recalculated whenever one of its beams/sheets gets modified or erased, these functions should be used with care. Also each TSL might be executed at dwg open time, to restore its proper state (if the file state flag is not set).

So how to write the script to avoid, or at least limit the use, of these expensive function calls. The best practice is to cache/store the list of beams or sheets that the TSL uses in its own private arrays \_Beam, \_Sheet,... Of course whenever required, these arrays need to be updated to contain the correct list of beams and sheets. On particular [events](#), the arrays should be rebuilt. The following gives an example of how to do this for the list of beams that the element contains.

[Sample of O-type and insert done in script:

```
if (_bOnInsert){
    _Pt0 = getPoint();
    _Element.append(getElement());

    // assign the array of beams of the element to _Beam
    _Beam = _Element[0].beam();
    return;
}

// check if TSL is attached to element, if not, do nothing
if (_Element.length()==0) return; // 0 is a valid index now
// use the elToUse element reference further on
```

```

Element elToUse = _Element[0];

if (_bOnElementConstructed || _bOnElementRead || _bOnElementRecalc || _bOnRecalc ) {
    // assign the array of beams of the element to _Beam
    _Beam = elToUse.beam();
}

if (_bOnElementDeleted){
    _Beam.setLength(0);
}

// Use the stored array of beams instead of requery the element
Beam arBeam[0];
arBeam = _Beam;

—end sample]

```

---

It is faster to query the CoordSys of the element once, and then use this to query the vectors of the element then to query the element each time again. If certain vectors are used often, eg. the vecX vector of an Element, it is better to assign it to a variable separately.

*[Sample code:*

```

CoordSys csel = elToUse.coordSys();
Vector3d vecLX = csel.vecX();

—end sample]

```

---

If the script adds different element tools to different zones, there is a difficulty, because the entity can only belong to one zone. Also, the entity must be put on a layer which is on, in order to be able to toggle the sublayers on which the element tools are visualized. Therefor it is needed that the tool is put on a layer that can be put on, but that essentially does not carry any visible items. The HSB-CAD software is designed that the E0 layer (cZoneCharacter 'E' with nZoneIndex 0) should be used for that. So if the script adds different element tools to different zones, the assignToElementGroup (see also [Change Instance group](#)) should be called with 'E' and 0.

For such a TSL it is also important, that the displays set the element zone with the Display::elemZone routine (see [Display on Element](#)).

*[Sample code:*

```

assignToElementGroup(elToUse,TRUE,0,'E');

Display dp(6);
// specify the elemzone to draw on

```

---

```
dp.elemZone(elToUse,0,'T');

—end sample]
```

---

A number of commands can be called on the element. If these commands are used with the pushCommandOnCommandStack (added hsbCAD2011 build 16.0.11) method, the command line variations of these commands has to be used. These command line commands start with as '-' character. Examples are:

-Hsb\_Store3d (added hsbCAD2011 build 16.0.11)  
 -Hsb\_GenerateConstruction (added hsbCAD2011 build 16.0.11)

Both these commands accept a floor name or an element name. The following example illustrates their use.

**IMPORTANT:** Since 17.0.46 there is a more efficient way to trigger generate construction, see [Element::triggerGenerateConstruction](#).

*[Sample Tsl that illustrates the pushCommandOnCommandStack, and Hsb\_GenerateConstruction and Hsb\_Store3d:*

```
if (_bOnInsert) {
  _Pt0 = getPoint();
  _Element.append(getElement());

  return;
}

// check if TSL is attached to element, if not, return
if (_Element.length()==0) return; // 0 is a valid index now
// use the elToUse element reference further on
Element elToUse = _Element[0];

String strContext1 = T("GenerateConstruction");
addRecalcTrigger(_kContext, strContext1);
if (_bOnRecalc && _kExecuteKey==strContext1) {
  Group grp = elToUse.elementGroup();
  String strName = grp.name();
  pushCommandOnCommandStack("-Hsb_GenerateConstruction "+strName);
}

String strContext2 = T("Store3d");
addRecalcTrigger(_kContext, strContext2);
if (_bOnRecalc && _kExecuteKey==strContext2) {
  Group grp = elToUse.elementGroup();
  String strName = grp.name();
  pushCommandOnCommandStack("-Hsb_Store3d "+strName);
}
```

—end sample]

---

## 5.16 Map

The type Map represents a key-value coupled list of items. Each item in the map has a key, the value could be an integer value, or a double, or a Point3d, or a Vector3d, or a PLine, ... The map much works like a dictionary. Whenever you want to store a variable inside the map, you have to give it a name, the key. To retrieve the value from the map, you can use the key again.

The map is stored as a list. Each item in the list must have a key. The key is not case sensitive. So the following keys are identical: "KEY" and "Key". Each value can be retrieved by its key name, or by its index in the list. The map has been made tolerant for duplicate keys. So in contrast with earlier implementations, the map can also be used as an indexed list. If the key is used, the first appearance of that key is used.

Each Tsl instance has one persistent map, called `_Map`. A user can store values that needs to be remembered between Tsl execution runs inside this persistent map, `_Map`.

For each type that is allowed in the map, some "set" functions, some "append" functions, some "get" functions, and some "has" functions exist. A "set" function will add or replace the existing item in the list. Using the key to access an entry, will restrict the access to the first appearance of the key. The value returned of the set function, is the index in the list where the item is stored. Ignoring the key lookup, but just adding a new item to the list can be done with the "append" function. To retrieve the value, one can use its index or its key. If the value does not exist, a default value is returned. This might be the case, if the key is not in the map, or if the type at the key or index does not correspond with the type to be retrieved. In all these cases the default value will be returned. To check if the key or index and type correspond, one can use the has routines. Either TRUE or FALSE will be returned.

A double can be stored without any units, as a length, as an area or as a volume. This is determined by the nUnitType during the setDouble routine. The value itself, during set and get is always given in drawing units, drawing units squared,... The default value of nUnitType is `_kLength`. Other values are `_kNoUnit`, `_kArea`, `_kVolume`, `_kAngle`.

A point can be specified as an absolute point, or as a relative point. A relative one, will be transformed during the transformBy call. Also for the `_Map`, the relative points will be transformed with the transformBy of the tsl entity. The default value of the nTransType is `_kRelative`. Other value is `_kAbsolute`.

A vector can be specified as a scalable vector, or as a fixed size vector. Both types are transformed during the transformBy call, but the fized size will keep its length. The fixed size is typically used for unit vectors. The default value of the nScaleType is `_kFixedSize`. Other value is `_kScalable`.

When a sub-map is added to a map, it can be added with a flag to indicate that it is a special type of sub-map of type "Object" or of type "Tool". The nMapType can have values `_kList` (default), `_kObject`, `_kTool`.

Since version hsbCAD2011 build 16.0.23, all key usage supports mapPaths (subfolder expression using backslash as separator). This means "getXX", "appendXX", "setXX" and "hasXX" as well as the removeAt support keys that contain subfolders. Beware, because the \ character is also the escape character in a string, the \ must be entered as \\ in a string literal. An example would be getInt("aa\\bb\\myInt") or setMap("aa\\bb\\myMap", mp).

On the dbCreateBodyEntity there is an argument strEntityType which should be one of the following values:

`_kBTSUBDMESH, _kBTSURFACE, _kBTMASSELEMENT, _kBT3DSOLID`

---

```
class Map {

    Map();

    void setMapKey(String strKey); // Sets the key that will be used when this map is added
                                    // as a submap. The key should represent the type, and should describe the
                                    // contents of the map.
    String getMapKey() const; // returns the key that was used as lookup in the parent map,
                            // or set by setMapKey

    void setMapName(String strKey); // A map can have a name, similar to a variable name.
    String getMapName() const; // Returns the name set by setMapName.

    int writeToXmlFile(String strFileName) const; // see spawn executable for predefined paths.
    int readFromXmlFile(String strFileName);
    int writeToDxxFile(String strFileName) const; // (added hsbCAD14.0.78)
    int writeToDxxFile(String strFileName, int bCapitalizeKeys) const; // (added hsbCAD16.1.2)
    int readFromDxxFile(String strFileName); // (added hsbCAD14.0.78) Do not use during
                                            // ModelX import, handles will have been resolved. Use ModelX methods
                                            // instead.

    String getXmlContent() const; // returns the map content as an xml string
    int setXmlContent(String strContent); // returns success of operation = valid xml content

    String toJsonContent(int nConvertFromDrawingUnitsToMM) const; // returns the map
                                                                // content as a JSON string (added hsbCAD20.0.55)
    int setJsonContent(String strContent); // returns success of operation = valid JSON content
                                         // (added hsbCAD20.0.55)

    String getDxContent(int nConvertFromDrawingUnitsToMM) const; // returns the map content
                                                                // as an dx string, with \n as separator
    int setDxContent(String strContent, int nValuesAreInMMAndNotDU); // returns success of
                                                                // operation = valid dx content, with \n or ; as separator

    int length() const;
    String keyAt(int nIndex) const;
    int indexAt(String strKey) const;
    String nameAt(int nIndex) const; // added v21.0.85

    int removeAt(String strKey, int bKeepSequence); // if not found, return 0 else return 1
    int removeAt(int nIndex, int bKeepSequence); // if not found, return 0 else return 1

    int moveLastTo(int nIndex); // if not found, return 0 else return 1
    int swapLastWith(int nIndex); // if not found, return 0 else return 1

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    int copyMembersFrom(Map mapSource); // return amount of keys copied. (added v23.8.9)
```

---

```

int setInt(String strKey, int val); // returns the index where the value is stored
int appendInt(String strKey, int val); // returns the index where the value is stored
int getInt(String strKey) const; // if not found, return 0
int getInt(int nIndex) const; // if not the correct type at this index, return 0
int hasInt(String strKey) const;
int hasInt(int nIndex) const;

int setDouble(String strKey, double val, int nUnitType); // returns the index where the value
// is stored
int setDouble(String strKey, double val); // default _kLength
int appendDouble(String strKey, double val, int nUnitType); // returns the index where the
// value is stored
int appendDouble(String strKey, double val); // default _kLength
double getDouble(String strKey) const; // if not found, return 0
double getDouble(int nIndex) const; // if not the correct type at this index, return 0
int hasDouble(String strKey) const;
int hasDouble(int nIndex) const;

int setString(String strKey, String val); // returns the index where the value is stored
int appendString(String strKey, String val); // returns the index where the value is stored
String getString(String strKey) const; // if not found, return ""
String getString(int nIndex) const; // if not the correct type at this index, return ""
int hasString(String strKey) const;
int hasString(int nIndex) const;

int setPoint3d(String strKey, Point3d val, int nTransType); // returns the index where the
// value is stored
int setPoint3d(String strKey, Point3d val); // default _kRelative
int appendPoint3d(String strKey, Point3d val, int nTransType); // returns the index where
// the value is stored
int appendPoint3d(String strKey, Point3d val); // default _kRelative
Point3d getPoint3d(String strKey) const; // if not found, return Point3d(0,0,0)
Point3d getPoint3d(int nIndex) const; // if not the correct type at this index, return
// Point3d(0,0,0)
int hasPoint3d(String strKey) const;
int hasPoint3d(int nIndex) const;

int setVector3d(String strKey, Vector3d val, int nScaleType); // returns the index where the
// value is stored
int setVector3d(String strKey, Vector3d val); // default _kFixedSize
int appendVector3d(String strKey, Vector3d val, int nScaleType); // returns the index where
// the value is stored
int appendVector3d(String strKey, Vector3d val); // default _kFixedSize
Vector3d getVector3d(String strKey) const; // if not found, return Vector3d(0,0,0)
Vector3d getVector3d(int nIndex) const; // if not the correct type at this index, return
// Vector3d(0,0,0)
int hasVector3d(String strKey) const;
int hasVector3d(int nIndex) const;

// The nAddAsUnresolved should be used only to store an entity which cannot be resolved.
// Adding an entity to a Map

```

```

// just stores the objectId into the map. But the objectId needs to be resolvable. An objectId can
// be resolved if the objectId
// is from the host drawing, or from any of the xreffed drawings. In the special case that you want
// to store an objectId to
// an entity from the host into an xreffed drawing, then, and only then, you should use TRUE for
// nAddAsUnresolved.
int setEntity(String strKey, Entity val); // returns the index where the value is stored
int setEntity(String strKey, Entity val, int nAddAsUnresolved); // default nAddAsUnresolved is
// FALSE.
int appendEntity(String strKey, Entity val); // returns the index where the value is stored
int appendEntity(String strKey, Entity val, int nAddAsUnresolved); // default nAddAsUnresolved
// is FALSE.
Entity getEntity(String strKey) const; // if not found, return invalid entity Entity\(\)
Entity getEntity(int nIndex) const; // if not the correct type at this index, return invalid
// entity
int hasEntity(String strKey) const;
int hasEntity(int nIndex) const;

// the following setEntityArray and getEntityArray are added in hsbCAD2011, build
16.0.23. The strName can be an empty string.
int setEntityArray(<Entity>[] arEnts, int bAddIfEmpty, String strKey, String strName, String
strEntry); // returns the index where the array is stored
Entity[] getEntityArray(String strKey, String strName, String strEntry) const; // if not found,
// return empty array

int setPLine(String strKey, PLine val, int nTransType); // returns the index where the value
// is stored
int setPLine(String strKey, PLine val); // default _kRelative
int appendPLine(String strKey, PLine val, int nTransType); // returns the index where the
// value is stored
int appendPLine(String strKey, PLine val); // default _kRelative
PLine getPLine(String strKey) const; // if not found, return PLine()
PLine getPLine(int nIndex) const; // if not the correct type at this index, return PLine\(\)
int hasPLine(String strKey) const;
int hasPLine(int nIndex) const;

int setMap(String strKey, Map val, int nMapType); // returns the index where the value is
// stored. If strKey == "", the key set by setMapKey is used.
int setMap(String strKey, Map val); // default _kList
int appendMap(String strKey, Map val, int nMapType); // returns the index where the value
// is stored
int appendMap(String strKey, Map val); // default _kList
Map getMap(String strKey) const; // if not found, return empty Map()
Map getMap(int nIndex) const; // if not the correct type at this index, return empty Map\(\)
int hasMap(String strKey) const;
int hasMap(int nIndex) const;

// The Point3dArray represents the COORD3 in the dxa
int setPoint3dArray(String strKey, Point3d[] val); // always _kRelative
int setPoint3dArray(String strKey, Point3d[] val, double[] blgs); // always _kRelative points
int appendPoint3dArray(String strKey, Point3d[] val); // always _kRelative
int appendPoint3dArray(String strKey, Point3d[] val, double[] blgs); // always _kRelative
// points

```

```

Point3d[] getPoint3dArray(String strKey) const; // if not found, return null length array
    Point3d[0]
PLine getPoint3dPLine(String strKey) const; // if not found, will return PLine()
Point3d[] getPoint3dArray(int nIndex) const; // if not the correct type at this index, return
    Point3d[0]
int hasPoint3dArray(String strKey) const;
int hasPoint3dArray(int nIndex) const;

// The Point3dXYArray represents the COORD2 in the dxa
int setPoint3dXYArray(String strKey, Point3d[] val); // always _kAbsolute
int setPoint3dXYArray(String strKey, Point3d[] val, double[] blgs); // always _kAbsolute
    points
int appendPoint3dXYArray(String strKey, Point3d[] val); // always _kAbsolute
int appendPoint3dXYArray(String strKey, Point3d[] val, double[] blgs); // always _kAbsolute
    points
Point3d[] getPoint3dXYArray(String strKey) const; // if not found, return null length array
    Point3d[0]
PLine getPoint3dXYPLine(String strKey) const; // if not found, will return PLine()
Point3d[] getPoint3dXYArray(int nIndex) const; // if not the correct type at this index, return
    Point3d[0]
int hasPoint3dXYArray(String strKey) const;
int hasPoint3dXYArray(int nIndex) const;

// The PlaneProfile methods are available from v21 on
int setPlaneProfile(String strKey, PlaneProfile val, int nTransType); // returns the index
    where the value is stored
int setPlaneProfile(String strKey, PlaneProfile val); // default _kRelative
int appendPlaneProfile(String strKey, PlaneProfile val, int nTransType); // returns the index
    where the value is stored
int appendPlaneProfile(String strKey, PlaneProfile val); // default _kRelative
PlaneProfile getPlaneProfile(String strKey) const; // if not found, return PlaneProfile()
PlaneProfile getPlaneProfile(int nIndex) const; // if not the correct type at this index, return
    PlaneProfile()
int hasPlaneProfile(String strKey) const;
int hasPlaneProfile(int nIndex) const;

// The Body methods are available from v21 on
int setBody(String strKey, Body val); // returns the index where the value is stored
int appendBody(String strKey, Body val); // returns the index where the value is stored
Body getBody(String strKey) const; // if not found, return Body()
Body getBody(int nIndex) const; // if not the correct type at this index, return Body().
Body getBody(String strKey, int bAllowInvalid) const; // (added v21.3.117) Default
    bAllowInvalid is FALSE. If TRUE, accept invalid solids to be returned.
Body getBody(int nIndex int bAllowInvalid) const; // (added v21.3.117) Default bAllowInvalid
    is FALSE. If TRUE, accept invalid solids to be returned.
int hasBody(String strKey) const;
int hasBody(int nIndex) const;

Entity dbCreateBodyEntity(String strKey, int nColor, String strEntityType) const; // (added
    v21.3.117), see explanation below
PLine[] getBodyFaceLoops(String strKey) const; // added v22.1.2, see example below
PLine[] getBodyFaceLoops(String strKey, double vertexCorrectionTolerance) const; // added
    v24.1.42 and v23.8.56

```

```

PlaneProfile[] getBodyAsPlaneProfilesList(String strKey) const; // added v22.1.2, see
example below
PlaneProfile[] getBodyAsPlaneProfilesList(String strKey, double vertexCorrectionTolerance)
const; // added v24.1.42 and v23.8.56

int showAdd() const; // (v21, developer only) open MapExplorer with this appended to the
content
};

```

---

```

int writeXmlFile(String strFileName) const;
int readFromXmlFile(String strFileName);

```

With the Map routines writeXmlFile and readFromXmlFile it is possible to write/read the contents of a Map into/from an Xml file. The xml grammar is Hsb specific. Usefull predefines are **\_kPathCurrentDir**,... see topic "[Spawn executable](#)" for other predefines.  
An example is provided below.

```
int length() const;
```

Returns the length of the list of key-value couples.

```
String keyAt(int nIndex) const;
```

With the index as an argument, one can get the key of the item at that index location in the list. If the index is out of the index bounds, the empty string is returned.

```
int indexAt(String strKey) const;
```

With the key as an argument, one can get the index of the first appearance of an item with that key, independent of the type of the item. If the key does not appear in the list, the index value -1 is returned.

```

int setString(String strKey, String val); // returns the index where the value is stored
int appendString(String strKey, String val); // returns the index where the value is stored
String getString(String strKey) const; // if not found, return ""
String getString(int nIndex) const; // if not the correct type at this index, return ""
int hasString(String strKey) const;
int hasString(int nIndex) const;

```

For each type that is allowed in the map, some "set" functions, some "append" functions, some "get" functions, and some "has" functions exist. A "set" function will add or replace the existing item in the list. Using the key to access an entry, will restrict the access to the first appearance of the key. The value returned of the set function, is the index in the list where the item is stored. Ignoring the key lookup, but just adding a new item to the list can be done with the "append" function. To retrieve the value, one can use its index or its key. If the value does not exist, a default value is returned. This might be the case, if the key is not in the map, or if the type at the key or index does not correspond with the type to be retrieved. In all these cases the default

value will be returned. To check if the key or index and type correspond, one can use the has routines. Either TRUE or FALSE will be returned.

```
int setMap(String strKey, Map val);
```

A map can also be added to a map. In this case it could be called a sub-map.

```
int setDouble(String strKey, double val, int nUnitType);
int setDouble(String strKey, double val);
```

A double can be stored without any units, as a length, as an area or as a volume. This is determined by the nUnitType during the setDouble routine. The value itself, during set and get is always given in drawing units, drawing units squared,... The default value of nUnitType is `_kLength`. Other values are `_kNoUnit`, `_kArea`, `_kVolume`, `_kAngle`.

```
int setVector3d(String strKey, Vector3d val, int nScaleType);
int setVector3d(String strKey, Vector3d val);
int setPoint3d(String strKey, Point3d val, int nTransType);
int setPoint3d(String strKey, Point3d val);
int setPLine(String strKey, PLine val, int nTransType);
int setPLine(String strKey, PLine val); // default _kRelative
```

A point can be specified as an absolute point, or as a relative point. A relative one, will be transformed during the transformBy call. Also for the `_Map`, the relative points will be transformed with the transformBy of the ts1 entity. The default value of the nTransType is `_kRelative`. Other value is `_kAbsolute`.

A vector can be specified as a scalable vector, or as a fixed size vector. Both types are transformed during the transformBy call, but the fized size will keep its length. The fixed size is typically used for unit vectors. The default value of the nScaleType is `_kFixedSize`. Other value is `_kScalable`.

Although the coordinates of the points need to be specified in world coordinate system, the point could mean a location that needs to stay relative to the entity. That is the meaning of `_kRelative`.

```
int setPoint3dArray(String strKey, Point3d[] val); // always _kRelative
int setPoint3dArray(String strKey, Point3d[] val, double[] blgs);
Point3d[] getPoint3dArray(String strKey) const; // if not found, return null length array
Point3d[0]
PLine getPoint3dPLine(String strKey) const;
Point3d[] getPoint3dArray(int nIndex) const; // if not the correct type at this index, return
Point3d[0]
int hasPoint3dArray(String strKey) const;
int hasPoint3dArray(int nIndex) const;
```

When a setPoint3dArray is done with an empty strKey, the key is automatically replaced by the key "COORD3". There is also a call to set the array of points, accompanied with an array of bulges. This is added for compatibility only. The prefered storage is ofcourse setPLine. Also remember that when the getPoint3dPLine is called, which is the only way to retrieve the bulges back, the normal of the PLine is not initialized correctly. The reason is that the correct normal is not known.

```
int removeAt(String strKey, int bKeepSequence); // if not found, return 0 else return 1
```

```
int removeAt(int nIndex, int bKeepSequence);
```

If an entry in the map is removed, either by specifying its index, or its key, a decision has to be made about performance. If the value of bKeepSequence is set to TRUE, the entry will be removed, and all subsequent entries in the map will be moved one place. If the value of bKeepSequence is set to FALSE, the last entry is moved to the location of the entry which is removed. This approach is more efficient in time, but has a drawback that the sequence is lost. The performance gain is however small, since internally only pointers are moved, not the real contents of the entries.

```
int moveLastTo(int nIndex); // if not found, return 0 else return 1
int swapLastWith(int nIndex); // if not found, return 0 else return 1
```

Since only "append" functions are available for all types, there might be a need to move the last item around in the list of items. To move the last item of the list to a given valid index, one can use the moveLastTo function. If the nIndex is bigger than the available indices, it will remain at the end of the list. If the nIndex is smaller than 0, it is moved to the start. Moving the last item to an index location, results in increasing the index of all items with an index equal or bigger than nIndex.

The swapLastWith function allows to move the item with index nIndex with the last item.

```
Body getBody(String strKey) const; // if not found, return Body()
Body getBody(int nIndex) const; // if not the correct type at this index, return Body().
Body getBody(String strKey, int bAllowInvalid) const; // (added v21.3.117) Default bAllowInvalid is FALSE. If TRUE, accept invalid solids to be returned.
Body getBody(int nIndex int bAllowInvalid) const; // (added v21.3.117) Default bAllowInvalid is FALSE. If TRUE, accept invalid solids to be returned.
```

Create a [Body](#) from the vertices and face loops in the Map. When bAllowInvalid is FALSE, by default, and the generated Body is not valid, different simple Body reconstruction methods are tried. If bAllowInvalid is TRUE, these simple reconstruction methods are not fired.

If bAllowInvalid is 0, different simple Body reconstruction methods are tried

If bAllowInvalid is 1, no reconstruction is tried, and invalid body might be returned (on which Autocad might crash)

If bAllowInvalid is 2, the faceloop reconstruction which is also used in the dbCreateBodyEntity method is used.

```
Entity dbCreateBodyEntity(String strKey, int nColor, String strEntityType) const; // (added v21.3.117)
```

Create a body entity from the vertices and face loops in the Map. If this conversion is not successful, the blsValid method will return FALSE on the returned Entity.

The SubDMesh is one which is most likely to succeed. From a SubDMesh, one can almost always create a Surface mesh, but a 3dSolid might fail.

The MassElement is basically a Body to MassElement which might fail as well. MassElement is created without checking if the Body is correct internally.

The created entity will be given the nColor as colorIndex. If the nColor is -1, the TsInstance colorIndex is used.

*[Example sample:*

```

// trigger create solid entity
String strBodyTypes[] = { _kBTSUBDMESH, _KBTSURFACE, _KBTMASSELEMENT,
_kBT3dSolid };
for (int s = 0; s < strBodyTypes.length(); s++)
{
    String strType = strBodyTypes[s];
    String sTriggerCreateType = "Create entity - " + strType;
    addRecalcTrigger(_kContext, sTriggerCreateType );
    if (_bOnRecalc && _kExecuteKey == sTriggerCreateType)
    {
        Entity ent = _Map.dbCreateBodyEntity(strMapKey, -1, strType);
    }
}
—end example]

```

---

[Example O-type:

```

Unit (1, "mm");

Map map;
map.setInt("int", 3);
map.setDouble("dbl", 4, _kNoUnit);
map.setDouble("len", U(5));
map.setDouble("Area", U(5)*U(6), _kArea);
map.setDouble("Vol", U(5)*U(6)*U(1), _kVolume);
map.setString("key", "value");
map.setPoint3d("pt0", _Pt0);
map.setVector3d("vecXY", _XU+ _YU);

map.removeAt("len", TRUE);

for (int i=0; i<map.length(); i++) {
    String strKey = map.keyAt(i);
    int nIndex = map.indexAt(strKey);
    reportNotice("\n"+nIndex+" "+strKey);
}
—end example]

```

[Example E-type, with insert done inside script:

```

Unit (1, "mm");

if (_bOnInsert) {
    _Beam.append(getBeam());
    _Pt0 = getPoint();

    PLine pl(_Beam[0].vecZ());
    pl.addVertex(_Pt0);
    pl.addVertex(_Pt0+U(200)*_Beam[0].vecX(), 1);
    pl.addVertex(_Pt0+U(400)*_Beam[0].vecX(), -1);

    Map map;

```

```

    map.setInt("int",3);
    map.setDouble("dbl",4,_kNoUnit);
    map.setDouble("len",U(5));
    map.setDouble("Area",U(5)*U(6),_kArea);
    map.setDouble("Vol",U(5)*U(6)*U(1), _kVolume);
    map.setString("key","value");
    map.setPoint3d("pt0",_Pt0);
    map.setVector3d("vecXY",_XU+_YU);
    map.setEntity("beam0",_Beam[0]);
    map.setPLine("poly",pl,_kAbsolute);

    _Map.setMap("submap",map);

    return;
}

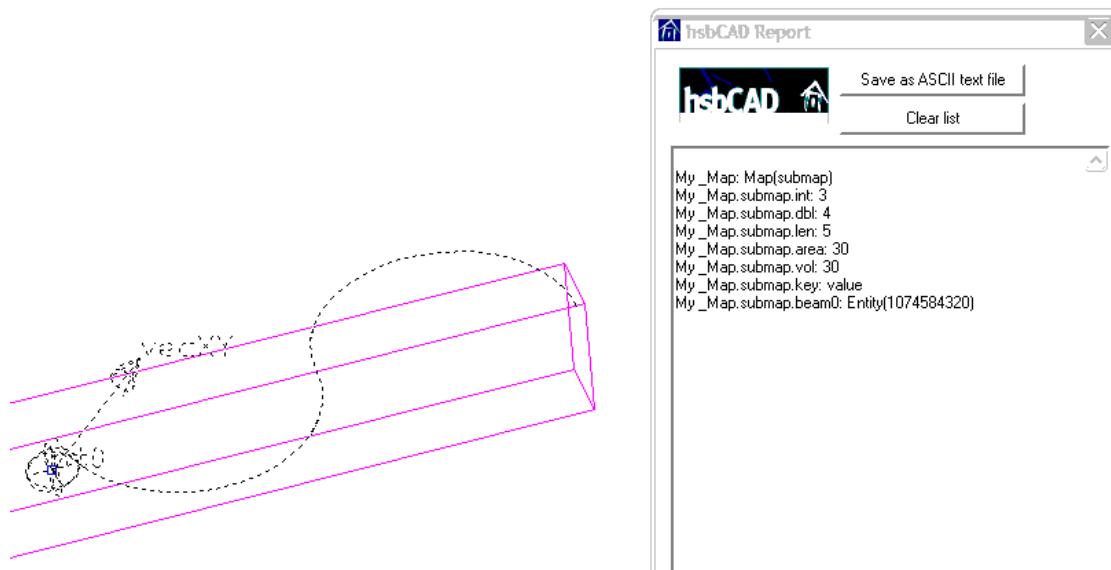
if (0) {
    Map map;
    map.setInt("int",3);
    _ThisInst.setMap(map); // should not call setMap on _ThisInst
because will be overwritten by _Map
    Map mapMe = _ThisInst.map();
    reportNotice("\nMy map(): "+mapMe);
}

reportNotice("\nMy _Map: "+_Map);
reportNotice("\nMy _Map.submap.int:
"+_Map.getMap("submap").getInt("int"));

Map submap = _Map.getMap("submap");
reportNotice("\nMy _Map.submap.dbl: "+submap.getDouble("dbl"));
reportNotice("\nMy _Map.submap.len: "+submap.getDouble("len"));
reportNotice("\nMy _Map.submap.area: "+submap.getDouble("area"));
reportNotice("\nMy _Map.submap.vol: "+submap.getDouble("vol"));
reportNotice("\nMy _Map.submap.key: "+submap.getString("key"));
Point3d pt0 = submap.getPoint3d("pt0");
pt0.vis();
Vector3d vecXY = submap.getVector3d("vecXY");
vecXY.vis(pt0);
PLine pl = submap.getPLine("poly");
pl.vis();

Entity ent = submap.getEntity("beam0");
reportNotice("\nMy _Map.submap.beam0: "+ent);
Beam bm0 = (Beam)ent;
if (bm0!=_Beam[0]) reportNotice("\nBeams not equal");

```



—end example]

[Example E-type that illustrates the writing and reading from xml file, with insert done inside script:

```

Unit (1,"inch");

if (_bOnInsert) {
    _Beam.append(getBeam());
    _Pt0 = getPoint();
    return;
}

// construct a map with some variables in
Map map;
map.setInt("int",3);
map.setDouble("dbl",4,_kNoUnit);
map.setDouble("len",U(4),_kLength);
map.setDouble("area",U(4)*U(1),_kArea);
map.setString("key","value");
map.setPoint3d("pt0",_Pt0,_kRelative);
map.setVector3d("vecXY",_XU+_YU,_kScalable);

PLine pl(_Beam[0].vecZ());
pl.addVertex(_Pt0);
pl.addVertex(_Pt0+U(200)*_Beam[0].vecX(),1);
pl.addVertex(_Pt0+U(400)*_Beam[0].vecX(),-1);

map.setEntity("beam0",_Beam[0]);
map.setPLine("poly",pl);

// create a map with uninitialized variables

```

```
Map mapD;
String strD; mapD.setString("strD",strD);
int iD; mapD.setInt("iD",iD);
double dD; mapD.setDouble("dD",dD);
Entity hD; mapD.setEntity("hD",hD);
Point3d ptD; mapD.setPoint3d("ptD",ptD);
Vector3d vecD; mapD.setVector3d("vecD",vecD);
PLine plD; mapD.setPLine("plD",plD);
Map mD; mapD.setMap("mD",mD);

// create an output map, and write the map to the file
Map mapOut;
mapOut.setDouble("aaa",U(4)*U(1)*U(1),_kVolume);
mapOut.setInt("int",3);
mapOut.setString("myKey","with the first value");
mapOut.setMap("submap",map);
mapOut.setMap("noInit",mapD);
mapOut.writeXmlFile("c:\\temp\\aa.xml");

// read the map back in again, and write it to another file
Map mapRW;
mapRW.readXmlFile("c:\\temp\\aa.xml");
mapRW.writeXmlFile("c:\\temp\\aarw.xml");

// this will result in the following c:\\temp\\aa.xml content
```

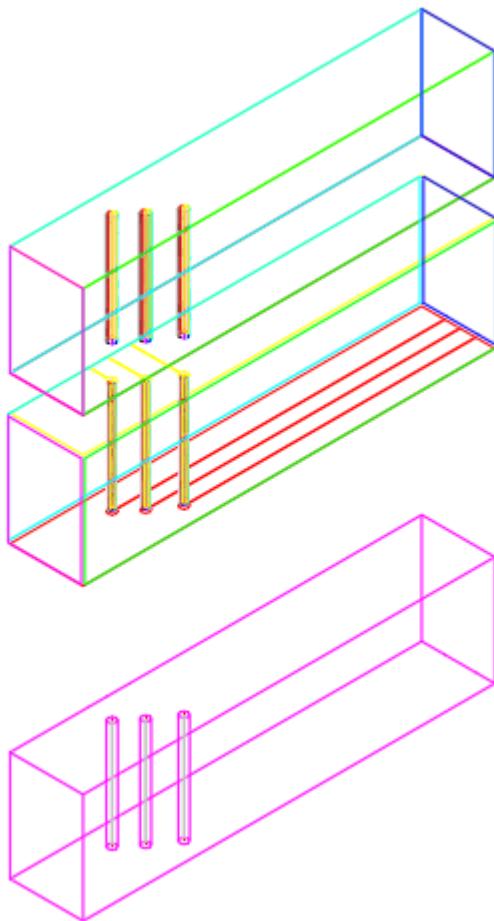
```

- <Hsb_Map>
  <dbl nm="aaa" ut="V" vl="65548.3" />
  <int nm="int" vl="3" />
  <str nm="myKey" vl="with the first value" />
- <lst nm="submap">
  <int nm="int" vl="3" />
  <dbl nm="dbl" ut="N" vl="4" />
  <dbl nm="len" ut="L" vl="101.6" />
  <dbl nm="area" ut="S" vl="2580.64" />
  <str nm="key" vl="value" />
  <pt nm="pt0" trm="R" vl="6758.22,3740.23,-25.0031" />
  <vc nm="vecXY" trm="A" vl="1,1,0" />
  <hdl nm="beam0" vl="210" />
- <crv nm="poly" trm="R" nrm="0,0,1">
  <pt vl="6758.22,3740.23,-25.0031" bg="1" />
  <pt vl="9210.63,-708.606,-25.0031" bg="-1" />
  <pt vl="11663,-5157.44,-25.0031" bg="0" />
</crv>
</lst>
- <lst nm="noInit">
  <str nm="strD" vl="" />
  <int nm="iD" vl="0" />
  <dbl nm="dD" ut="L" vl="0" />
  <hdl nm="hD" vl="" />
  <pt nm="ptD" trm="R" vl="0,0,0" />
  <vc nm="vecD" trm="R" vl="0,0,0" />
  <crv nm="pID" trm="R" nrm="0,0,1" />
  <lst nm="mD" />
</lst>
</Hsb_Map>

```

—end example]

[Example of E tsl attached to a Beam, illustrating the getBodyFaceLoops



```

Unit (1, "mm");
PropDouble dOffset(0, U(200), T("|offset|"));
PropDouble dExplode(1, U(5), T("|explode|"));

if (_bOnInsert) {
    _Beam.append(getBeam());
    _Pt0 = getPoint();
    return;
}

String strMapKey = "submap\\body";
//strMapKey = "submap\\SimpleBody";

String strChangeEntity0 = T("|reset submap|");
addRecalcTrigger(_kContext, strChangeEntity0 );
if (_bOnRecalc && _kExecuteKey==strChangeEntity0)
{
    Map map;
    _Map.setMap("submap", map);
}

String strChangeEntity2 = T("|setBody body|");
addRecalcTrigger(_kContext, strChangeEntity2 );

```

```

if (_bOnRecalc && _kExecuteKey==strChangeEntity2)
{
    _Map.setBody(strMapKey, _Beam0.realBody());
}

String strChangeEntity4 = T("appendBody body|");
addRecalcTrigger(_kContext, strChangeEntity4 );
if (_bOnRecalc && _kExecuteKey==strChangeEntity4 )
{
    _Map.appendBody(strMapKey, _Beam0.realBody());
}
Body bd = _Map.getBody(strMapKey);
bd.transformBy(_XW + _YW + _ZW) * dOffset);
bd.vis(1);

// _Map.showAdd();

PLine plFaces[] = _Map.getBodyFaceLoops(strMapKey);
Display dp(-1);
for(int i=0; i<plFaces.length(); i++)
{
    int nColor = i%8 + 1;
    PLine plFace = plFaces[i];
    PlaneProfile profFace(plFace);
    dp.color(nColor);

    // draw the PLine, a bit offset in the normal direction (using
dExplode)
    {
        plFace.transformBy(_XW + _YW + _ZW) * 2 * dOffset);

        Vector3d vecN = plFace.coordSys().vecZ();
        plFace.transformBy(dExplode * vecN);

        Point3d pnts[] = plFace.vertexPoints(TRUE);
        Point3d pt; pt.setToAverage(pnts);
        vecN.vis(pt, nColor);
        dp.draw(plFace);
    }

    // creating a PlaneProfile from the PLine will remove the bridge
edges
    {
        profFace.transformBy(_XW + _YW + _ZW) * 3 * dOffset);
        dp.draw(profFace, _kDrawAsCurves);
    }
}

```

*—end example]*

## 5.17 File management

The following predefines define some paths which might be helpful when working with files.

<u><b>_kPathCurrentDir:</b></u>	is set to the current directory, the directory with which autocad was started, often set in the shortcut
<u><b>_kPathHsbInstall:</b></u>	is set to the install location of hsbCad.
<u><b>_kPathWindowsInstall:</b></u>	is set to the windows install location.
<u><b>_kPathHsbCompany:</b></u>	is set to the active company location for the drawing.
<u><b>_kPathHsbWallDetail:</b></u>	is set to the active company wall detail.
<u><b>_kPathHsbRoofDetail:</b></u>	is set to the active company roof detail.
<u><b>_kPathPersonal:</b></u> Acad2004 and higher).	is set to the personal folder of the current user (only available in Acad2004 and higher).
<u><b>_kPathProgramFiles:</b></u> Acad2004 and higher).	is set to the program files folder (only available in Acad2004 and higher).
<u><b>_kPathRoamableRoot:</b></u> Acad2004 and higher).	is set to the autocad roamable root folder (only available in Acad2004 and higher).
<u><b>_kPathDwg:</b></u>	is set to the folder of the current Dwg.
<u><b>_kPathPersonalTemp:</b></u> hsbCAD15.1.16).	is set to the temp folder of the current user (added hsbCAD15.1.16).
<u><b>_kPathAppData:</b></u>	is set to the users app data folder (added hsbCAD15.1.16).

It is also possible to locate the file through the method `findFile` which returns the full path of the given `strFile` argument.

Since added hsbCAD2009+ build 14.1.43, hsbCAD2010 build 15.0.18, the paths that are searched with the `findFile` method are extended. If the path is specified, the `findFile` will return the path if the file can be found at that location. If the path is not specified, but only the filename with extension, then the file is searched for:

- In the auto search folders of the `hsbInstall`. Not all subfolders of `hsbInstall` are searched but most of them are.
- If the file is not present in any of these locations, the file is searched in the `<Company>\Tsl + subfolders`.
- If the file is not found, it is searched in the `<hsbInstall>\Content + subfolders`.

**String** `findFile(String strFile); // (added hsbCAD15.0.12 and hsbCAD14.1.27)`

To manipulate folders the following can be used:

```
int makeFolder(String strFolder); // (added hsbCAD15.1.16) return if the folder exists after the call
int removeEmptyFolder(String strFolder); // (added hsbCAD21.0.64) return if the folder existed and was removed
int deleteFile(String strFile); // (added hsbCAD21.0.64) return if the file existed and was deleted
String[] readTextFile(String strFile); // (added hsbCAD21.0.64) return content of text file, popular encodings are supported
int writeTextFile(String strFile, String[] arLines); // (added hsbCAD21.0.64) return if the writing was successful
```

---

```

void cleanFolder(String strFolder, String strWildCard); // (added hsbCAD21.0.64) delete
    files from given folder that match strWildCard expression
void cleanFolder(String strFolder); // (added hsbCAD21.0.64) delete all files from given
    folder
String[] getFilesInFolder(String strFolder, String strWildCard); // (added hsbCAD21.0.64)
    return list of files in given folder that match strWildCard expression
String[] getFilesInFolder(String strFolder); // (added hsbCAD21.0.64) return list of all files in
    given folder
String[] getFoldersInFolder(String strFolder, String strWildCard); // (added hsbCAD21.0.64)
    return list of folders in given folder that match strWildCard expression
String[] getFoldersInFolder(String strFolder); // (added hsbCAD21.0.64) return list of all sub
    folders in given folder

String expandEnvironmentVariables(String str); // (added hsbCAD21.4.51 and
    hsbCAD22.0.98)

```

---

[Example O-type:

```

String strTempFolder = _kPathPersonalTemp;
reportNotice("\\n\\ntempfolder: " + strTempFolder);
String strFolderName = "AFolder";
String strFolder = strTempFolder + "\\\" + strFolderName;

String strAction1 = T("|makeFolder|");
addRecalcTrigger(_kContext, strAction1 );
if (_bOnRecalc && _kExecuteKey==strAction1)
{
    makeFolder(strFolder);
}

String strAction2 = T("|removeEmptyFolder|");
addRecalcTrigger(_kContext, strAction2 );
if (_bOnRecalc && _kExecuteKey==strAction2)
{
    int bRemoved = removeEmptyFolder(strFolder);
    reportNotice("\\nFolder: " + strFolder + (bRemoved ? " removed"
: " not removed."));
}

String strAction3 = T("|writeTextFile|");
addRecalcTrigger(_kContext, strAction3 );
if (_bOnRecalc && _kExecuteKey==strAction3)
{
    String lines[] = { "line 1", "line 2", "line 3"};
    writeTextFile(strFolder + "\\file1.txt", lines);
    writeTextFile(strFolder + "\\file2.txt", lines);
    writeTextFile(strFolder + "\\file1.bat", lines);
}

String strAction4 = T("|delete file1.bat|");
addRecalcTrigger(_kContext, strAction4 );

```

---

```

if (_bOnRecalc && _kExecuteKey==strAction4)
{
    String strFile = strFolder + "\\file1.bat";
    int bRemoved = deleteFile(strFile);
    reportNotice("\nFile: " + strFile + " " + bRemoved);
}

String strAction5 = T("|cleanFolder *.txt|");
addRecalcTrigger(_kContext, strAction5 );
if (_bOnRecalc && _kExecuteKey==strAction5)
{
    cleanFolder(strFolder, "*.txt");
}

String strAction6 = T("|read file1.bat|");
addRecalcTrigger(_kContext, strAction6 );
if (_bOnRecalc && _kExecuteKey==strAction6)
{
    String strFile = strFolder + "\\file1.bat";
    String lines[] = readTextFile(strFile);
    reportNotice("\nFile: " + strFile + " has following
content:");
    for(int i=0; i<lines.length(); i++)
        reportNotice("\n - " + lines[i]);
}

// report state
{
    int bExists = (getFoldersInFolder(strTempFolder,
strFolderName).length() == 1);
    String files[] = getFilesInFolder(strFolder);
    String folders[] = getFoldersInFolder(strFolder);
    int nCountFilesInside = files.length();
    int nCountFoldersInside = folders.length();
    reportNotice("\nfolder: " + strFolder + (bExists ? " exists" : "
does not exist") +
        ", and has " + nCountFilesInside + " files and " +
nCountFoldersInside + " sub folders.");
    for(int i=0; i<files.length(); i++)
        reportNotice("\n   " + files[i]);
    for(int i=0; i<folders.length(); i++)
        reportNotice("\n   " + folders[i]);
}

```

*—end example]*

## 5.18 Spawn executable

While a tsl script is run, it is possible to fire the execution of an external program.

```
int spawn(String strLocale, String strExe, String strArg1, String strArg2);
int spawn_detach(String strLocale, String strExe, String strArg1, String strArg2);
```

- strLocale: allowed values are "", "C", "English", "German",... It specifies the localized settings. Use "C" for ansi C language.
- strExe: the name of the executable program
- strArg1: the first argument of the exe program. Can also be an empty string.
- strArg2: the second argument of the exe program. Can also be an empty string.

The spawn\_detach will not wait until the executable has finished, while the spawn call, will wait until the executable has finished.

The error code of the operation is returned. However, the actual spawning will only be done, when autocad is in a normal state. For instance during the dwgin, the spawning will not be done. In such a case, -1 is also returned.

If more arguments need to be passed to the executable program, one can use a file, and read/write a Map to it. With the [Map](#) routines writeToFile and readFromFile it is possible to write/read the contents of a Map into/from an Xml file.

The \ symbol in the strExe, but also in other strings, needs to be escaped: \\. So \\ actually counts as one \ after interpretation of the string.

Spaces embedded in strArg1 or strArg2 may cause unexpected behavior; for example, passing spawn the string "hi there" will result in the new process getting two arguments, "hi" and "there". If the intent was to have the new process open a file named "hi there", the process would fail. You can avoid this by quoting the string: "\"hi there\"".

See [file management](#) for other predefined paths which might be helpful when working with files.

- \_kPathCurrentDir: is set to the current directory, the directory with which autocad was started, often set in the shortcut
- \_kPathHsbInstall: is set to the install location of hsbCad.
- \_kPathWindowsInstall: is set to the windows install location.
- \_kPathHsbCompany: is set to the active company location for the drawing.
- \_kPathHsbWallDetail: is set to the active company wall detail.
- \_kPathHsbRoofDetail: is set to the active company roof detail.
- \_kPathPersonal: is set to the personal folder of the current user (only available in Acad2004 and higher).
- \_kPathProgramFiles: is set to the program files folder (only available in Acad2004 and higher).
- \_kPathRoamableRoot: is set to the autocad roamable root folder (only available in Acad2004 and higher).
- \_kPathDwg: is set to the folder of the current Dwg.
- \_kPathPersonalTemp: is set to the temp folder of the current user (added hsbCAD15.1.16).
- \_kPathAppData: is set to the users app data folder (added hsbCAD15.1.16).

[Sample of O-type:

```

String strHsbInstall = _kPathHsbInstall;
String strRoamFldr = _kPathRoamableRoot;
String strWinInstall = _kPathWindowsInstall;
String strCur = _kPathCurrentDir;
String strProgFiles = _kPathProgramFiles;
String strPers = _kPathPersonal;

int nn = 3;
if (nn==0) {
    String strExe = "C:\\temp\\commtest.exe";
    spawn("",strExe , "", "");
}
else if (nn==1) {
    int nRet = spawn("", "c:\\windows\\regedit.exe", "", "");
}
else if (nn==2) {
    int nRet = spawn("", strWinInstall + "\\system32\\cmd.exe", "/cregedit","");
}
else if (nn==3) {
    int nRet = spawn_detach("", strWinInstall + "\\regedit","", "");
}

```

—end sample]

## 5.19 Lisp ScriptInsert

The lisp interface for Tsl consists of two commands: "**Hsb\_ScriptInsert**" and "**Hsb\_ScriptInsertNoPrompt**".

The lisp command "**Hsb\_ScriptInsert**" can be used to customize a toolbar. When executing the autocad command

```

(Hsb_ScriptInsert "aa" 0 "key")
(Hsb_ScriptInsert "aa" )
(Hsb_ScriptInsert "aa" "key")
(Hsb_ScriptInsert "aa" 0)

```

an instance of the TSL named "aa" is added with the option 0. The option 0 means multiple instances can be inserted. If the option is 1, only one instance will be inserted at a time. The 0/1 is optional.

The second string value in the argument list is recognized and will be send to the script as \_kExecuteKey. The "key" is optional.

If the script named "aa" does not exist at that time, an attempt is made to load the script from the file system. If the complete file path was specified as argument, that file will be loaded. If not found, the install folders of hsbCAD are searched. Then the Company/Tsl folder is searched, including its sub folders.

The tsl script will be inserted with `_bOnInsert` set to TRUE.

The following are examples of the "`Hsb_ScriptInsertNoPrompt`". During this insert, the value of `_bOnInsert` is set to FALSE since all data are already specified.

When the MAP, or MAPMM key is specified, the contents will be interpreted and inserted into the `_Map` variable. The MAP (MAPMM) value is a ';' -separated list. The keys and values are alternating. If you want to add a ';' character in the value, it needs to be escaped by the '\' escape character. Also the '\' character needs to be escaped. The list can be closed with a ';', but this is optional. So far for interpreting the ';' -separated string. Of course Lisp has itself its escape sequences when the string is literal, so the instruction could look like (cons "MAP" "BC;this \\; is embedded")

You can also use the keyword "NAME" as an alias for "SCRIPTNAME", and the keyword "ARGUMENT" for the keyword "MAPMM".

The following are recognized keys in the list of key value couples passed as argument to the `Hsb_ScriptInsertNoPrompt` lisp command:

SCRIPTNAME or NAME	* required; only 1 allowed; name of the script as string.
20ELEMID	* optional; can appear many times; the value must be of ent type; the value will be appended to the <code>_Entity</code> array, and if the value is of the correct type it will populate the <code>_Element</code> , <code>_Opening</code> , <code>_Viewport</code> , ... array. The value could be a variable <code>ent1</code> , eg set by (setq <code>ent1</code> ( <code>entlast</code> )).
20BEAMID	* optional; can appear many times; the value must be of GenBeam type (Beam, Sheet, Sip); the value will be appended to the <code>_GenBeam</code> array, and if the value is of the correct type it will populate the <code>_Beam</code> , <code>_Sheet</code> or <code>_Sip</code> array. The value could be a variable <code>bm1</code> , eg set by (setq <code>bm1</code> ( <code>entlast</code> )).
11PT	* optional; can appear many times; the value must be a Point type, specified in world coordinate system (WCS). For the E, O, and S types the first key value will become <code>_Pt0</code> . All other values are appended to the <code>_PtG</code> array. If you have a point in UCS, you can transform it into WCS using (trans pt 1 0).
13UCSX	* optional; only 1 allowed; the value must be a vector in WCS. The value will return in the Tsl as <code>_XU</code> . If not specified, the current ucs x value will be used. It could also be the specified by (getvar "UCSXDIR").
13UCSY	* optional; only 1 allowed; the value must be a vector in WCS. The value will return in the Tsl as <code>_YU</code> . If not specified, the current ucs y value will be used. It could also be the specified by (getvar "UCSYDIR").
MAP, ARGUMENT or MAPMM	* optional; only 1 allowed; The content of the value is on its own a key value list which will be interpreted, and will populate the <code>_Map</code> persistent variable. If the

70PROPINT0

key is MAPMM or ARGUMENT, the value is a list in mm, while the MAP key is a list in drawing units. The value is a semi column separated list of key values. In this list, the keys can start with an integer specifying the map type. A 70 is always an int. A 40 is a length value. 11 is an indicator for a point.

\* optional; only 1 allowed; the value must be a number which will be assigned to the PropInt with index 0.

70PROPINT1...70PROPINT99

\* see 70PROPINT0, with the only difference that the value will be assigned to the property with the index equal to the number specified after 70PROPINT.

40PROPDDOUBLE0

\* optional; only 1 allowed; the value must be a double which will be assigned to the PropDouble with index 0.

40PROPDDOUBLE1...40PROPDDOUBLE99

\* see 40PROPDDOUBLE0, with the only difference that the value will be assigned to the property with the index equal to the number specified after 40PROPDDOUBLE.

PROPSTRING0

\* optional; only 1 allowed; the value must be a string which will be assigned to the PropString with index 0.

PROPSTRING1...PROPSTRING99

\* see PROPSTRING0, with the only difference that the value will be assigned to the property with the index equal to the number specified after PROPSTRING.

*[Lisp code sample to insert a TSL called \_ElemNail:*

```
(Hsb_ScriptInsertNoPrompt (list
  (cons "SCRIPTNAME" "_ELEMNAIL") ; script starts with underscore for HSB buildin tsis
  (cons "20ELEMID" ent1)
  (cons "20BEAMID" bm1)
  (cons "20BEAMID" bm2)
  (cons "11PT" (trans pt1 1 0)) ; transform from ucs to wcs
  (cons "11PT" (trans pt2 1 0))
  (cons "13UCSX" (getvar "UCSXDIR")) ; make sure to add the ucs in the list of args
  (cons "13UCSY" (getvar "UCSYDIR"))
  (cons "40PROPDDOUBLE1" 4.5) ; PropDouble with index 1
  (cons "70PROPINT0" zoneindex) ; PropInt with index 0
  (cons "70PROPINT3" toolindex) ; PropInt with index 3
  (cons "PROPSTRING1" "Just a string") ; PropString with index 1
))
```

*—end sample]*

*[Lisp code sample to insert a TSL called insertWithMap and accompanying TSL*

```
(Hsb_ScriptInsertNoPrompt (list
  (cons "SCRIPTNAME" "insertWithMap")
  (cons "13UCSX" (getvar "UCSXDIR")) ; make sure to add the ucs in the list of args
```

```

  (cons "13UCSY" (getvar "UCSYDIR"))
  (cons "MAP" "70Count;3;40Len;1;11PTX;10;11PTY;20;11PTZ;30") ; Map that is interpreted
in drawing units
  (cons "MAPMM" "70Count2;3;40Len2;25.4;11PT2X;10;11PT2Y;20;11PT2Z;30;") ; Map that
is interpreted with values in mm
))

int nCount = 0;
if (_Map.hasInt("COUNT")) nCount = _Map.getInt("COUNT");
PropInt pCount(5,nCount,"Counter");

double dLen = 0;
if (_Map.hasDouble("LEN")) dLen = _Map.getDouble("LEN");
PropDouble pLen(5,dLen,"Len");

double dLen2 = 0;
if (_Map.hasDouble("LEN2")) dLen2 = _Map.getDouble("LEN2");
PropDouble pLen2(6,dLen2,"Len2");

if (_Map.hasPoint3d("PT")) _Pt0 = _Map.getPoint3d("PT");

_Map = Map(); // reset contents of map

—end sample]

```

*[Lisp code illustrating the \_kExecuteKey, for a more extensive example look for [setPropValuesFromCatalog](#)*

```

if (_bOnInsert) {
  reportMessage("\n_kExecuteKey is " + _kExecuteKey);
  _Pt0 = getPoint();
  return;
}

—end sample]

```

## 5.20 S-type

The S-type is used to describe the hsbCad subassembly. A subassembly is a set of GenBeams (at least one), and a collection of other entities that are identified as one part. Typically they are preassembled before they are processed by the factory, and as such identified by a name or article. A subassembly is an item of which multiple instances can exist. A subassembly instance has a name, which identifies the subassembly type. If the name is not set, the tsl instance is not recognised as a subassembly.

An S-type tsl instance is exported with the element dx, if the subAssemblyName is not an empty string, and if the routine exportWithElementDxa is called. In the example, the element with which it is exported is the element of the primary beam. Also the instance is assigned to the group of the primary beam. If this is the desired behaviour, and also if the tsl instance should react on beam group/element changes, the following should be added:

```
if (_GenBeam.length()==0) return;
Element el = _GenBeam[0].element();
exportWithElementDxa(el);
assignToElementGroup(el);
```

The following global routines are specific for an S-type. In order to recognise the instance as a real subassembly, the name must be set.

```
setSubAssemblyName(String strName);
String subAssemblyName() const;
```

After executing the insert procedure, two special executions are done during which the predefined variables `_bOnCheckSubAssembly0` and `_bOnCheckSubAssembly1` have a value TRUE.

---

*[Sample of S-type for 2 beams:*

```
_st_dTol = Unit(1, "mm");
_st_Len0 = U(2667.2);
_st_WF0 = U(38);
_st_HF0 = U(90);
_st_arBoxDim.setLength(0);
_st_arBoxDim.append(Vector3d(0, U(-38), 0)); // ptCen
_st_arBoxDim.append(Vector3d(U(2667.2), 0, 0)); // dL*vecX
_st_arBoxDim.append(Vector3d(0, U(38), 0)); // dW*vecY
_st_arBoxDim.append(Vector3d(0, 0, U(90))); // dH*vecZ
_st_dYZ0Range = U(38);

//{{STIL: subassembly template insert location} }

if (_bOnCheckSubAssembly0) {
    int bKeep = _Beam0.hasDimensions(_st_Len0, _st_WF0, _st_HF0
, _st_dTol);
    if (bKeep) {
        TslInst subAss = _Beam0.subAssembly();
        if (subAss.bIsValid() ) bKeep = FALSE; // the _Beam0 already
belongs to a subassembly
    }
    if (!bKeep) eraseInstance();
    return;
}

if (_bOnCheckSubAssembly1) {

    Beam bm0 = _Beam0;
```

```

int nNumBeam1 = _st_arBoxDim.length()/4; // the length is a
multiple of 4, each set of 4 represents one beam
Beam arBm1[0];
Beam bmPot[0];

int bKeep = TRUE;
if (bKeep) {
    // first do an efficient filter step
    bmPot =
bm0.filterBeamsCenterDistanceYZRange(_Beam,_st_dYZ0Range,_st_dTol);
    bmPot = bm0.filterGenBeamsNotInSubAssembly(bmPot); // keep only
non subass beams
    bmPot = bm0.filterGenBeamsNotThis(bmPot); // keep beams not
equal to bm0
    bmPot =
bm0.filterBeamsBoxLocation(bmPot,_st_WF0,_st_arBoxDim,_st_dTol);

    // only take the first set
    if (bmPot.length()>=nNumBeam1) {
        arBm1.setLength(nNumBeam1+1);
        arBm1[0] = bm0;
        for (int b=0; b<nNumBeam1; b++) {
            arBm1[b+1] = bmPot[b];
        }
    }
    else bKeep = FALSE;
}

// We know now if this set of beams matches the configuration that
is described.
if (!bKeep) {
    eraseInstance();
    return;
}

bmPot.setLength(nNumBeam1); // since we keep, the bmPot.length()
>=nNumBeam1
CoordSys arTrans[] =
bm0.coordSysBeamsBoxLocation(bmPot,_st_WF0,_st_arBoxDim,_st_dTol);
if (arTrans.length()>0) {
    CoordSys tr = arTrans[0];
    // It is essential to set the ECS to the correct transformation
for beam recording
    _PtE = tr.ptOrg();
    _XE = tr.vecX();
    _YE = tr.vecY();
    _ZE = tr.vecZ();
}
else {
    eraseInstance();
    return;
}

```

```

}

_Beam = arBm1; // take over array of beams
_Sheet.setLength(0);
_Element.setLength(0);
_Element.append(bm0.element());
return;
}

setSubAssemblyName(scriptName());
String str = subAssemblyName();

if (_Element.length() == 0) return;
Element el = _Element[0];
exportWithElementDxa(el);
assignToElementGroup(el);

—end sample]

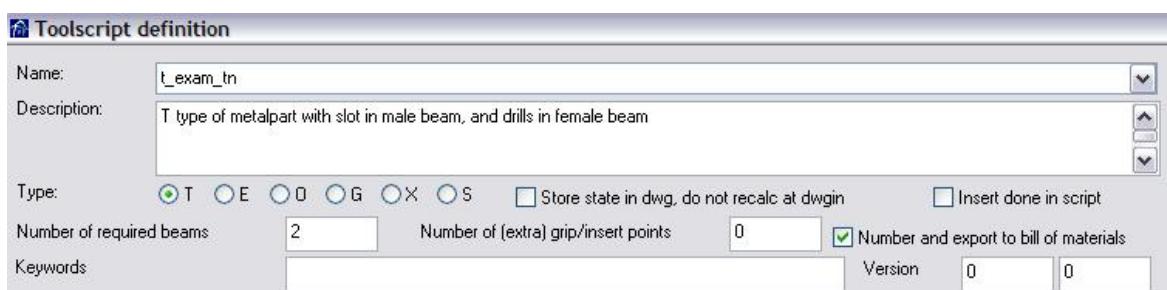
```

## 5.21 Store state in dwg

There is a toggle in the Tsl editor header that is called: "Store state in dwg, do not recalc at dwgin". When this toggle is selected, instances of this tsl will store there state in the drawing when the drawing is saved as dwg. The state of a tsl instance includes of course all persistent data (\_Map, \_Beam, \_Element,...), but also the calculated data like tools, graphics,... When the toggle is turned off, the calculated data (tools, graphics, ...) is not stored inside the dwg file. The persistent data is of course always stored. When the calculated data is not stored in the dwg file, the size of the file is smaller, but on the other hand, the tsl instance will need to recalculate before it can show itself, after the file has been opened.

For most Tsl's it is recommended that the state is stored inside the dwg. Only if the tsl uses heavy graphics, including complicated solids, the Tsl author might decide not to store the graphical state, but to recalculate it after file open.

Before this toggle was available Tsl instances used not to store their state in the dwg, but to recalculate at dwgin. Now the default for a new tsl definition has the toggle turned on.



To change the toggle of all loaded Tsl's in the drawing, there is a special hsbCAD command added:

**Hsb\_StoreTslStateInDwg**

When the command is fired, the user can turn on the toggle of all Tsl's in the drawing.

## 5.22 ShedWizard

When the command HSB\_LOGSHEDWIZARD is fired, the dialog allows to specify a number of Tsl's which will be attached to the walls. Typical functionality that can be added through a Tsl is the shape of the log wall, close to the roofplane. To assist in this functionality, the Tsl's that are appended will have a value set in their map \_Map.

In the persistent [Map](#) called \_Map there is the subMap "ShedWizard" which contains the double "RoofOverhang".

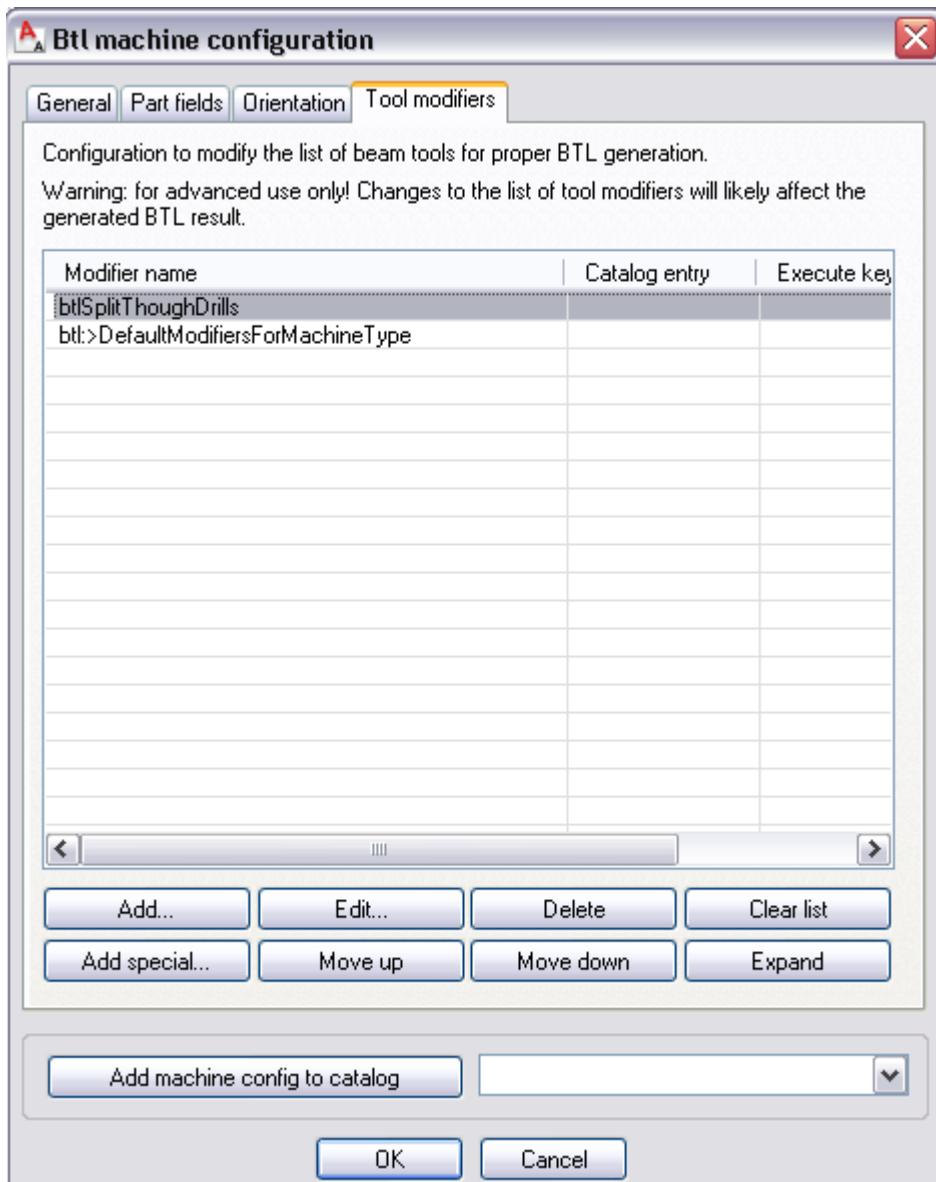
## 5.23 Btl generation

Among other cnc output, hsbCAD allows to output numbered beams into Btl format. For more info on Btl see <http://www.design2machine.com>. The Btl output in hsbCAD is based on the [AnalysedTools](#).

In the dialog of the HSB\_BTL command, it is possible to add a list of tool modifiers (since hsbCAD13.2.122). Tool modifiers change the list of tool operations that are defined inside hsbCAD on a beam. The tools are changed before they are actually translated into BTL machine instructions. Tool modifiers are either build in or added as tsl. There is already a list of build in tool modifiers: "SplitThroughDrillForSplinterFreeBeam", "AddEndCutForEndMortise", "ChangeHeadBeamCutIntoHouse",... This list of build in tool modifiers is automatically composed and applied, depending on the machine type you have selected for the BTL generation.

Tool modifiers can play a valuable role in customization of the BTL output. To explain you, here is a small example. Two few weeks ago, I was at a Canadian customer. He has a BTL machine. The machine is equipped with a special drill. The drill head has already the sink mill integrated. Now to control such a device, normally the machine software should handle that. That means if in the btl file it finds 2 drills that combined match the geometry of the tool, the machine should only do one action. But in case of the machine software of this BTL machine, such a solution was not possible for the software company to develop in short notice. So hsbCAD developed a "tool modifier" as a custom Tsl, which does the detection of such drills, and replaces them with just one drill.

In the image below the btlSplitThroughDrills is added. The contents of this Tsl is found below.



During the examination of the list of AnalysedTools, it is often required to report log messages. If the log messages are appended to a sub map of \_Map, and keyed with one of the logging predefines, they will appear in the logger output. The name of the sub map of \_Map must be: **\_kLogMessages** (which is of type String, with value "LogMessages"). The following map keys of type **String** are recognized in the logger map:

**\_kLLNoLog, \_kLLFatal, \_kLLError, \_kLLWarn, \_kLLInfo, \_kLLDebug, \_kLLTrace**

*[Example of O-type that can function as a btl tool modifier, and illustrates the use of logging.*

```
// find the submap with the analysed tools
AnalysedTool tls[] = AnalysedTool().convertFromSubMap(_Map,
_kAnalysedTools);
```

```

Map mpLogger; // the logger map
// add some info that will be shown if logging is set to Debug level
mpLogger.appendString(_kLLDebug, "List of recognized tools:");

// loop over all the analysed tools, and log some info of them
for (int i=0; i<tls.length(); i++) {
    AnalysedTool at = tls[i];
    String strTool = at; // convert the analysed tool into a string
    representation
    mpLogger.appendString(_kLLDebug, "Tool "+i+": "+strTool);
    Map mpTool = at.mapInternal(); // get the internal map of the tool
    mpLogger.appendString(_kLLTrace, mpTool);
}

// append the logger map to the _Map
.setMap(_kLogMessages, mpLogger);

eraseInstance(); // fire eraseInstance to prevent the analysed tools
to be reread

```

—end sample]

[Example **btlSplitThroughDrill**: O-type that can function as a btl tool modifier. This modifier will split each through drill in 2 drills.

```

Unit(1, "mm");

if (_bOnInsert) { // For debugging only, allow to append the tsl to a
beam in the drawing.
    _Beam.append(getBeam());
    _Pt0 = getPoint("Select point");
    return;
}

// Compose a _Map from the beam, in case the Tsl is appended to the
database.
// The contents of the Map should be equal to the _Map generated by
the btl framework.
if (_ThisInst.handle()!="") { // if handle is not empty, then
instance is database resident
    reportMessage("\nRecompose map from beam entity");
    Beam bm = _Beam0;

    // get a relevant tool, and store it inside the _Map->ToolList
    AnalysedTool tools[] = bm.analysedTools(1); // 1 means verbose
    reportMessage
    Map mpToolList = AnalysedTool().convertToMap(tools);
    _Map.setMap(_kAnalysedTools, mpToolList);
}

```

```

// find the submap with the analysed tools
AnalysedTool tls[] = AnalysedTool().convertFromSubMap(_Map,
_kAnalysedTools);

Map mpLogger; // the logger map

// loop over all tools, and take out the through drills
// loop backwards so the swap can be used
AnalysedDrill anDrills[0];
for (int t=tls.length()-1; t>=0; t--) {

    AnalysedDrill anDrill = (AnalysedDrill)tls[t];
    if (!anDrill.bIsValid())
        continue; // no a drill
    if (!anDrill.bThrough())
        continue; // not a through drill
    if (anDrill.bUseThisDirection())
        continue; // do not split drill if drill is flagged with
bUseThisDirection

    // remove the tool t from the list tls
    tls.swap(t,tls.length()-1);
    tls.removeAt(tls.length()-1);
    mpLogger.appendString(_kLLTrace, anDrill+" at index "+t+
removed);

    anDrills.append(anDrill);
}

// loop over all the drills, and split them in 2
for (int d=0; d<anDrills.length(); d++) {
    AnalysedDrill anDrill = anDrills[d];
    double dRadius = anDrill.dRadius();

    mpLogger.appendString(_kLlDebug, "Drill split "+ anDrill);

    // compose 2 maps of the new drills to be appended
    Map mpDrillsS = anDrill.mapInternal();
    Map mpDrillE = mpDrillsS; // copy

    Vector3d vecDir = anDrill.ptEndExtreme() -
anDrill.ptStartExtreme();
    Point3d ptMid = anDrill.ptStartExtreme() + 0.5*vecDir;
    vecDir.normalize();
    double dOverlap = U(0.5);

    Point3d ptStart1 = anDrill.ptStartExtreme();
    Point3d ptEnd1 = ptMid+0.5*dOverlap*vecDir;
    ptStart1.vis(1);
    ptEnd1.vis(1);
}

```

```

if (_bOnDebug) { // can only be if database resident
    Body bdDrill(ptStart1,ptEnd1,dRadius);
    bdDrill.vis(1);
}
mpDrills.setPoint3d("ptStart",ptStart1);
mpDrills.setPoint3d("ptEnd",ptEnd1);
mpDrills.setInt("bZNFree",FALSE);

Point3d ptStart2 = anDrill.ptEndExtreme();
Point3d ptEnd2 = ptMid-0.5*dOverlap*vecDir;
ptStart2.vis(3);
ptEnd2.vis(3);
if (_bOnDebug) { // can only be if database resident
    Body bdDrill(ptStart2,ptEnd2,dRadius);
    bdDrill.vis(3);
}
mpDrillE.setPoint3d("ptStart",ptStart2 );
mpDrillE.setPoint3d("ptEnd",ptEnd2 );
mpDrills.setInt("bZNFree",FALSE);

// mpLogger.appendString(_kLLTrace, "mpDrills:
"+mpDrills.getDxContent(FALSE));
// mpLogger.appendString(_kLLTrace, "mpDrillE:
"+mpDrillE.getDxContent(FALSE));

AnalysedDrill anDrills(mpDrills); // construct from map contents
if (!anDrills.bIsValid()) {
    mpLogger.appendString(_kLLDebug, "Invalid new drill 1 not
appended");
}
else { // tool is valid
    tls.append(anDrills);
    mpLogger.appendString(_kLLTrace, "New drill 1 appended");
}

AnalysedDrill anDrillE(mpDrillE); // construct from map contents
if (!anDrillE.bIsValid()) {
    mpLogger.appendString(_kLLDebug, "Invalid new drill 2 not
appended");
}
else { // tool is valid
    tls.append(anDrillE);
    mpLogger.appendString(_kLLTrace, "New drill 2 appended");
}
}

// substitute the modified list of tools in the _Map
Map mpModifiedTools = AnalysedTool().convertToMap(tls);
_Map.setMap(_kAnalysedTools, mpModifiedTools);

```

```

// append the logger map to the _Map
_map.setMap(_kLogMessages, mpLogger);

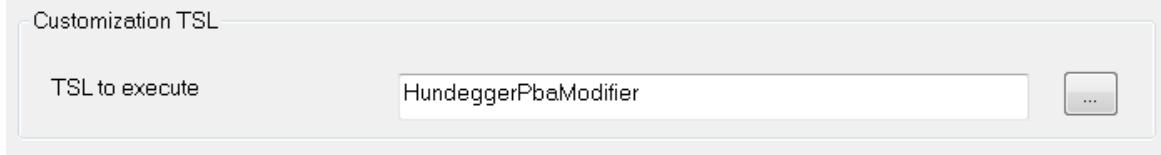
// report logging as notice
if (_bOnDebug) { // can only be if database resident
    for (int l=0; l<mpLogger.length(); l++) {
        reportNotice("\n"+mpLogger.keyAt(l)+":
"+mpLogger.getString(l));
    }
}

—end sample]

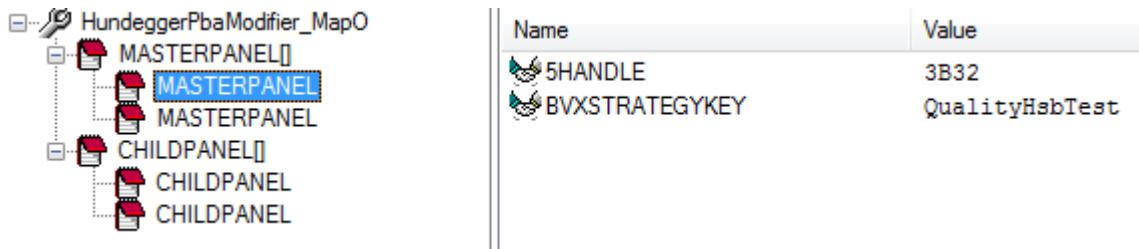
```

## 5.24 HundeggerPba generation

From hsbCAD 18.1.21, during the command Hsb\_HundeggerPba, it is possible to specify a tsl that will execute during the export.



The Tsl that is specified is called for a MapIO event during the Hsb\_HundeggerPba command. The input and output map have both the same structure.



The output map can have the following additional string keys:

For MasterPanel:

- BvxStrategyKey

For ChildPanel:

- BvxUserAttribute1
- BvxUserAttribute2
- BvxUserAttribute3

---

*[Example of O-type that can function as an example for the HundeggerPba that sets the BvxStrategyKey for some master panels.]*

```

if (_bOnMapIO) {

    String strNameTrigger = "special";

    reportNotice("\nTsl execution: " +scriptName() + ":" + +
_kExecuteKey + " " + _bOnMapIO);
    reportNotice("\nMaster panels with name set to: '" +
strNameTrigger + "' are modified");

    // for debugging export map
    _Map.writeToDxxFile(_kPathDwg+"\\\"+scriptName()+"_MapI.dxx");

    // read from _Map
    Map mpMasterPanels = _Map.getMap("MasterPanel[]");
    Map mpChildPanels = _Map.getMap("ChildPanel[]");

    Map mpMasterPanelsNew;
    Map mpChildPanelsNew;

    // modify the individual master panels
    for (int i=0; i<mpMasterPanels.length(); i++)
    {
        if (mpMasterPanels.keyAt(i)!="MasterPanel") {
            reportNotice("\nKey not recognized in MasterPanel[] at
index: "+i);
            continue;
        }

        Map mpI = mpMasterPanels.getMap(i);
        Entity ent =mpI.getEntity("Handle");
        MasterPanel entMP = (MasterPanel)ent;
        if (!entMP.bIsValid()) {
            reportNotice("\nCould not read masterpanel in MasterPanel[]
at index: "+i);
            continue;
        }

        if (entMP.name()==strNameTrigger) {
            mpI.setString("BvxStrategyKey", "QualityHsbTest");
        }

        mpMasterPanelsNew.appendMap("MasterPanel",mpI);
    }

    // do nothing with child panels now
    mpChildPanelsNew = mpChildPanels ;

    // write back into _Map
    _Map.setMap("MasterPanel[]", mpMasterPanelsNew);
    _Map.setMap("ChildPanel[]", mpChildPanelsNew);
}

```

```
// for debugging export map
.writeToDxxFile(_kPathDwg+"\\"+scriptName()+"_MapO.dxx");

reportNotice("\nTsl execution: " +scriptName() + " done\n");
}

—end sample]
```

*[Example of O-type that can function as an example for the HundeggerPba that sets the BvxUserAttribute1 of each child panel.*

```
if (_bOnMapIO) {

    int bDebug = FALSE;
    reportNotice("\nTsl execution: " +scriptName() + ":" + 
_kExecuteKey + " " + _bOnMapIO);

    // for debugging export map
    if (bDebug)
        _Map.writeToDxxFile(_kPathDwg+"\\"+scriptName()+"_MapI.dxx");

    // read from _Map
    Map mpChildPanels = _Map.getMap("ChildPanel[]");
    Map mpChildPanelsNew; // = mpChildPanels;

    // modify the individual child panels
    for (int i=0; i<mpChildPanels.length(); i++)
    {
        String strKey = mpChildPanels.keyAt(i);
        Map mpI = mpChildPanels.getMap(i);

        if (strKey != "ChildPanel") {
            reportNotice("\nKey not recognized in ChildPanel[] at index: "+i);
            mpChildPanelsNew.appendMap(strKey ,mpI);
            continue;
        }

        Entity ent =mpI.getEntity("Handle");
        ChildPanel entCP = (ChildPanel)ent;
        if (!entCP.bIsValid()) {
            reportNotice("\nCould not read child in ChildPanel[] at index: "+i);
            mpChildPanelsNew.appendMap(strKey ,mpI);
            continue;
        }

        Sip entSip = entCP.sipEntity();
        if (entSip.bIsValid())
        {
```

```

        String strStyle = entSip.style();
        mpI.setString("BvxUserAttribute1", strStyle);
        reportNotice("\nSip " + entSip.posnum() + "
BvxUserAttribute1 set to: " + strStyle);
    }

    mpChildPanelsNew.appendMap(strKey ,mpI);
}

// write back into _Map
_map.setMap("ChildPanel[]", mpChildPanelsNew);

// for debugging export map
if (bDebug)
    _Map.writeToDxxFile(_kPathDwg+"\\\"+scriptName()+"_MapO.dxx");

reportNotice("\nTsl execution: " +scriptName() + " done\n");
}

```

*—end sample]*

## 5.25 Option system

In hsbCAD there is an option system (at least if you have purchased the proper hsbCAD module for it :). If you have the option system, then you have access to the drop down menu called hsbOptions, as well as the extra hsb console tab called options.

Generally said, the purpose of the option system is to **turn on and off** certain entities **depending on the value of certain variables**.

If we look closer to the above sentence, we can discover the building blocks of the option system:

- Each **variable** is called an option definition. A definition can be of type Boolean, Double or String. The console tab options shows the list of option definitions.
- Each variable / option definition has always a state or **value** assigned (the on or off state of the light bulb in the console tab for Boolean ones). A drawing is always in a certain state regarding the options. You can change the state of all the option definitions at once using the menu entry: "set options". As an alternative you can modify the state of a certain option definition by double clicking in the console option tree on the option.
- If an entity participates in the option behaviour, there is a rule set attached to the entity. The rule set **evaluates** to TRUE or FALSE, **visible or not visible**. The rule set of an entity can be accessed through the menu entry "Create option rules". Alternatively, an option rule is appended when the entity is added to the console tree option as show or hide.

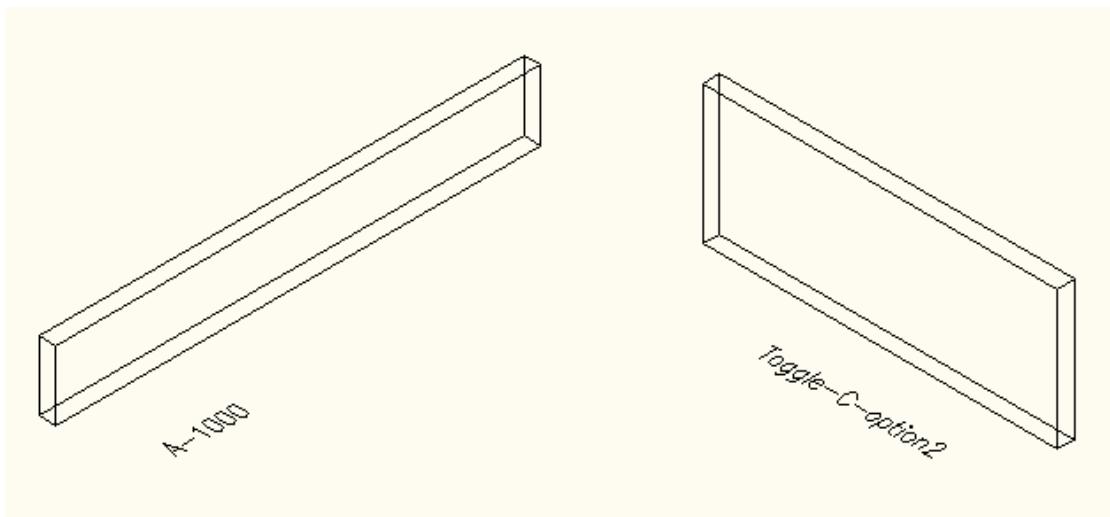
Since each option definition has always a certain value assigned, one can evaluate the state of the entity. This can be done by the method:

```
int Entity::optionEvaluate() const; // see Entity
```

When a Tsl instance has an option rule set attached (is added to an option definition), it will be fired automatically when the state of any option changes. During such an event the [predefined status variable](#) **\_bOnOptionChanged** is TRUE.

Here are the 2 scenario's how tsl's could be useful:

- 1) Upon settings an option (assigning a value to each of the option definitions), the tsl is triggered. Depending on the value of each of the option definitions, the tsl could do something. If the option definition name is a property of the tsl, the tsl itself could be option system independent.
- 2) Upon settings an option (assigning a value to each of the option definitions), the rule set evaluates to the option state: true or false. Depending on the tsl its state, the tsl could do something. In fact, depending on the state of also the entities in the entity set of the tsl, the tsl could do something.



[Example O-type TSL that illustrates the use of options scenario 1:

```

Unitl("mm"); //units in this script are in mm

PropString pName("A", "My name");
PropString pOptionDefinitionToTrackl("option1", "option definition to track");
PropDouble pWallHeightOnl(1000), "WallheightON");
PropDouble pWallHeightOffl(2000), "WallheightOFF");

if (_bOnInsert) {
    _Entity append(getEntity());
    _Pt0 = getPoint();
    return;
}

//check if this ts1 is connected to entities
if (_Entity.length() == 0)
    return;

//check if the firstentity is of type Wall
Wall wall = Wall_Entity[0];
if (!wall.bIsValid())
    return;

```

```

if (_bOnOptionChanged) {
    reportMessage ("n" + pName + "-event:_bOnOptionChanged.");
}

Map mOptionInfo = _Map.getMap("OptionsTriggerInfo"); //info from option change
mOptionInfo.writeToFile (_kPathDwg + "\\" + scriptName + ".xml"); //dump map in file

Map mOptionToTrack = mOptionInfo.getMap(optionDefinitionToTrack);
reportMessage ("n"+mOptionToTrack.getDxContent(FALSE)); //dump map on screen

String strVal= mOptionToTrack.getString("Value"); //get the state of the option
if (strVal=="") { //option data was present

    int bTunedOn = strVal.toInt();

    //do something
    if (bTunedOn) {
        wall.setBaseHeight(pWallHeightOn);
    }
    else {
        wall.setBaseHeight(pWallHeightOff);
    }
    reportMessage ("n" + pName + ": just changed baseheight using
"+bTunedOn );
    }
}

double dWallHeight= wall.baseHeight();
reportMessage ("n" + pName + ": wallbaseheight:" + dWallHeight);

//display a message in full name
Display dp(1);
String strM = "Toggle-" + pName + "-" + optionDefinitionToTrack;
dp.draw(strM,_Pt0,_XU,_YU,1,1);

—end example]

```

[Example O-type TSL that illustrates the use of options scenario 2:

```

Unit(,"mm"); //units in this script are in mm

PropString pName0,"A","My name");
PropDouble pWallHeight0,0,1000,"Wallheight to change to");

if (_bOnInsert) {
    _Entity.append(getEntity());
    _Pt0 = getPoint0;
    return;
}

```

```

//check if this ts1 is connected to entities
if (_Entity.length == 0)
    return;

//check if this Ts1Inst is turned on by the option
if (!_ThisInst.optionEvaluate()) {
    //this Ts1 is turned off at the moment
    _Map.setInt("Visible", FALSE);
    reportMessage ("\n" + pName + ": is not visible");
    return;
}

//check if the first entity is of type Wall
Wall wall = Wall_Entity[0];
if (!wall.IsValid())
    return;

if (_bOnOptionChanged) {
    //the ts1 is fired by changing an actual option state
    reportMessage ("\n" + pName + "-event: _bOnOptionChanged.");

    //check if the previous state was invisible
    int bWasVisible = _Map.getInt("Visible");
    if (bWasVisible) {
        //so, changed the state from invisible to visible
        reportMessage ("\n" + pName + ": is turned on"); //report it

        //keep track of my own state in my _Map, to be able to detect changes
        _Map.setInt("Visible", TRUE);

        //do something
        wall.setBaseHeight(pWallHeight);
        reportMessage ("\n" + pName + ": just changed baseheight");
    }
}

reportMessage ("\n" + pName + ": wallbaseheight:" + wall.baseHeight());

//display a mean info message
Display dp(-1);
String strM = pName + "-" + pWallHeight;
dp.draw(strM, _P0,_XU,_YU,1,1);

—end example]

```

## 5.26 ModelX

The ModelX type bundles the functionality to import and export as well as access ModelX data. ModelX is the term used in hsbCAD to refer to the hsbCAD specific data format. The ModelX data is an expression of any hsbCAD specific entity or object in the form of a [Map](#). All Map formats are supported, so it could be memory Map, dxx file format, xml file format,...

ModelX is the primary data exchange of hsbCAD with other applications. These applications could be either developed inside Hsb-Soft, or by 3rd party developers.

Typically on a ModelX instance a sequence of methods is called. This sequence is depending on whether the ModelX is being exported or imported.

For export the following sequence is used (as is illustrated in the example below):

1. `setEntities`: set the entities that will be expressed in the ModelX
2. `dbComposeMap`: compose the Map from the entities in the drawing into the member map().
3. `writeToDxxFile`: write the Map to a file.

For import the following sequence is used (as is illustrated in the example below):

1. `readFromDxxFile`: read a Map from file into the member map().
2. `dblInterpretMap`: interpret the map content as ModelX changing the entities in the drawing
3. `entity`: possibly do additional actions on the changed/imported entities.

```
class ModelX { // (added hsbCAD14.0.75 and renamed in V25.1.120 and V26.2.10)

    ModelX();

    void setEntities(<Entity>[] arEntities); // set the entities to be added to the map
    Entity[] entity() const; // list of entities set or interpreted

    Map& map(); // return type is Map&, a reference to the internal map member of ModelX,
    void setMap(Map mapNew); // overwrite the internal Map. Is equivalent to writing map() = mapNew.

    int dbComposeMap(ModelXComposeSettings flags); // see ModelXComposeSettings
    int dblInterpretMap(ModelXInterpretSettings flags); // see ModelXInterpretSettings

    int writeToDxxFile(String strFileName) const; // see spawn executable for predefined
    paths.
    int writeToDxxFile(String strFileName, int bCapitalizeKeys) const; // (added hsbCAD16.1.2)
    int writeToDxxFile(String strFileName, int bCapitalizeKeys, int bAddUniquelsAsHandle)
    const; // (added 24.0.11)

    int readFromDxxFile(String strFileName); // This method will not resolve handles until
    dblInterpretMap is called. If you need to resolve use Map method.
    int writeXmlFile(String strFileName) const; // (added hsbCAD14.0.78)
    int readXmlFile(String strFileName); // (added hsbCAD14.0.78)

    // the entities in entity() of specific types can be accessed directly
    GenBeam[] genBeam() const;
    Beam[] beam() const;
    Sheet[] sheet() const;
    Sip[] sip() const;
    TsInst[] tsInst() const;
```

```
// static methods to fire the cnc exporter from within Tsl. Should only be used in
// command context.
static String[] exporterGroups(); // added build v16.4.3, v17.1.9
static String[] exporterShortcuts(); // added build v20.0.41
static int callExporter(ModelXComposeSettings flags, Map mapProjectInfoOverwrite, String[]
    arFloorLevelNames, String strExporterGroup, String strDestinationFolder); //
    added build v16.4.3, v17.1.9
static int callExporter(ModelXComposeSettings flags, Map mapProjectInfoOverwrite,
    <Entity>[] arEntities, String strExporterGroup, String strDestinationFolder); //
    added build v16.4.8, v17.1.15
};
```

---

**Map& map();**

The ModelX has internally a member of type [Map](#). This member is exposed through the method `map()`. The return value of this method is of type `Map` reference, denoted `Map&`. Unlike most other methods in `Tsl` which return a copy of the value returned, the `map()` method returns a reference. A reference can be used as a normal value, both in an expression or on the left side of an assignment. The following are valid statements:

*[Sample to illustrate the use of `Map&`*

```
// create an Map mapOut, and add some keys to it
Map mapOut;
mapOut.setDouble("aaa", 4 * 1 * 1, kVolume);
mapOut.setInt("int", 3);
mapOut.setString("myKey", "with the first value");

ModelX mm; // instantiate a ModelX
reportMessage("\nModelXs map1: " + mm.map() + "\n"); // use mm.map()
in an expression, here in an automatic cast into String.

// takes a copy of MapOut, and overwrite the content of mm.map()
with it.
mm.map() = mapOut;

—end sample]
```

---

*[Example O-type illustrating the `export` of a ModelX into a dxx file selecting entities.*

```
Unit(1, "mm");

if (_bOnInsert) {
    PrEntity ssE(T("|Select entities to be exported|"));
}
```

---

```

if (!ssE.go ()) {
    eraseInstance ();
    return;
}
Entity ents[] = ssE.set ();
reportMessage (T("\n|Number of entities selected:| ") +
ents.length ());

// set some export flags
ModelXComposeSettings mmFlags;
mmFlags.addSolidInfo (TRUE); // default FALSE
mmFlags.addAnalysedToolInfo (TRUE); // default FALSE
mmFlags.addElemToolInfo (TRUE); // default FALSE
mmFlags.addConstructionToolInfo (TRUE); // default FALSE
mmFlags.addHardwareInfo (TRUE); // default FALSE
mmFlags.addRoofplanesAboveWallsAndRoofSectionsForRoofs (TRUE); //
default FALSE
mmFlags.addCollectionDefinitions (TRUE); // default FALSE

// compose ModelX
ModelX mm;
mm.setEntities (ents);
mm.dbComposeMap (mmFlags);

// write ModelX to dxx file
String strFileName = _kPathDwg + "\\mm.dxx";
mm.writeToDxxFile (strFileName);

// launch viewer
String strViewerExe = _kPathHsbInstall + "\\Utilities\\
\\DxxExplorer\\DXExplorer.exe";
spawn ("", strViewerExe, strFileName, "");

eraseInstance ();
return;
}

```

*—end example]*

[Example O-type illustrating the **export** of a ModelX into a dxx file selecting elements.

```

Unit(1, "mm");

if (_bOnInsert) {

PrEntity ssE(T("|Select elements to be exported|"), Element());
if (!ssE.go ()) {
    eraseInstance ();
    return;
}

```

```

Entity elems[] = ssE.set();
reportMessage(T("\n|Number of elements selected:| ") +
elems.length());

// find entities from elements
Entity ents[] = ssE.set(); // add elements as well

for (int i=0; i<elems.length(); i++) {
    Element el = (Element)elems[i]; // cast the entity to a
element
    if (!el.bIsValid())
        continue;
    Group grp = el.elementGroup(); // get group from element

    // get entities from group
    Entity elEnts[] = grp.collectEntities(FALSE, Entity(),
_kModelSpace);
    reportMessage("\nFor elem " +el.number() + " found " +
elEnts.length() + " entities.");

    ents.append(elEnts); // append to collection
}
reportMessage(T("\n|Number of entities collected:| ") +
ents.length());

// set some export flags
ModelXComposeSettings mmFlags;
mmFlags.addSolidInfo(TRUE); // default FALSE
mmFlags.addAnalysedToolInfo(TRUE); // default FALSE
mmFlags.addElemToolInfo(TRUE); // default FALSE
mmFlags.addConstructionToolInfo(TRUE); // default FALSE
mmFlags.addHardwareInfo(TRUE); // default FALSE
mmFlags.addRoofplanesAboveWallsAndRoofSectionsForRoofs(TRUE); //
default FALSE
mmFlags.addCollectionDefinitions(TRUE); // default FALSE

// compose ModelX
ModelX mm;
mm.setEntities(ents);
mm.dbComposeMap(mmFlags);

// write ModelX to dxx file
String strFileName = _kPathDwg + "\\\\" + mm.dxx";
mm.writeToDxxFile(strFileName);

// launch viewer
String strViewerExe = _kPathHsbInstall + "\\Utilities\\
\\DxxExplorer\\DXExplorer.exe";
spawn("", strViewerExe, strFileName, "");

eraseInstance();

```

```

        return;
    }
}

```

—end example]

[Example O-type illustrating the **import** of a ModelX from a dxx file.

```

Unit(1, "mm");

if (_bOnInsert) {

    String strFileName = _kPathDwg + "\\mm.dxx";
    PropString pFileName(0,strFileName,T("|Filename to import|"));
    PropDouble pDistToMove(0,U(1000),T("|Distance to move after
import|"));
    showDialog(); // present property dialog to user

    // read ModelX dxx file into mm.map
    ModelX mm;
    mm.readFromDxxFile(pFileName);

    // set some import flags
    ModelXInterpretSettings mmFlags;
    mmFlags.resolveEntitiesByHandle(TRUE); // default FALSE
    mmFlags.resolveElementsByNumber(TRUE); // default FALSE
    mmFlags.setBeamTypeNameAndColorFromHsbId(TRUE); // default FALSE

    // interpret ModelX
    mm.dbInterpretMap(mmFlags);

    // report the entities imported/updated/modified
    Entity ents[] = mm.entity();
    reportMessage (T("\n|Number of entities imported:| ") +
ents.length());

    // move these entities
    Vector3d vecMove(pDistToMove,0,0);
    for (int e=0; e<ents.length(); e++) {
        Entity ent = ents[e];
        ent.transformBy(vecMove);
    }

    eraseInstance();
    return;
}

```

—end example]

[Example O-type illustrating the callExporter.

```

if (_bOnInsert) {

    String arGroups[] = ModelX().exporterGroups();
    arGroups.insertAt(0, ""); // add empty string
    PropString pGroup(0,arGroups,T("|Exporter group|"));

    String arShortcuts[] = ModelX().exporterShortcuts();
    arShortcuts.insertAt(0, ""); // add empty string
    PropString pShortcut(1,arShortcuts,T("|Exporter shortcut|"));

    PrEntity ssE;
    if (!ssE.go()) {
        return;
    }

    showDialog();

    // set some export flags
    ModelXComposeSettings mmFlags;
    mmFlags.addSolidInfo(TRUE); // default FALSE
    mmFlags.addAnalysedToolInfo(TRUE); // default FALSE
    mmFlags.addElemToolInfo(TRUE); // default FALSE
    mmFlags.addConstructionToolInfo(TRUE); // default FALSE
    mmFlags.addHardwareInfo(TRUE); // default FALSE
    mmFlags.addRoofplanesAboveWallsAndRoofSectionsForRoofs(TRUE); // default FALSE
    mmFlags.addCollectionDefinitions(TRUE); // default FALSE

    String strDestinationFolder = _kPathDwg;

    // Map that contains the keys that need to be overwritten in the ProjectInfo
    Map mpProjectInfoOverwrite;
    mpProjectInfoOverwrite.appendString("ProjectInfo\ProjectRevision","10.11.12");
    mpProjectInfoOverwrite.appendString("ProjectInfo\\PathDwg", "aa");

    String strExportGroup = pGroup;
    if (strExportGroup == "")
        strExportGroup = pShortcut;

    // call the exporter
    int bOk = ModelX().callExporter(mmFlags, mpProjectInfoOverwrite,
ssE.set(), strExportGroup, strDestinationFolder);
    if (!bOk)
        reportMessage("\nTsl::callExporter failed.");

    eraseInstance();
    return;
}

```

—end example]

## 5.26.1 ModelXComposeSettings

The ModelXComposeSettings type represents a set of options that can be used during the composing of a [ModelX](#):

The ModelXComposeSettings is typically used in the dbComposeMap method of [ModelX](#):  
**ModelX::dbComposeMap(ModelXComposeSettings flags);**

---

```
class ModelXComposeSettings { // (added hsbCAD14.0.75 and renamed in V25.1.120 and
V26.2.10)

    void addSolidInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE.
    void addAnalysedToolInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE.
    void addOldToolInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE (added
        v20.1.44 and v21.0.39)
    void addElemToolInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE.
    void addConstructionToolInfo(int bSet); // set to TRUE or FALSE. The default value is
        FALSE.
    void addHardwareInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE.
    void addRoofplanesAboveWallsAndRoofSectionsForRoofs(int bSet); // set to TRUE or FALSE.
        The default value is FALSE.
    void addCollectionAndBlockDefinitions(int bSet); // set to TRUE or FALSE. The default value
        is FALSE.
    void addAllGroups(int bSet); // set to TRUE or FALSE. The default value is FALSE. (added
        hsbCAD20.1.3 and hsbCAD21.0.6)
    void addUniqueIdsAsHandle(int bSet); // set to TRUE or FALSE. The default value is TRUE.
        (added 24.0.11). Has only effect on the callExporter
};
```

---

[Sample.

```
ModelXComposeSettings mmFlags;
mmFlags.addSolidInfo(TRUE); // default FALSE
mmFlags.addAnalysedToolInfo(TRUE); // default FALSE
mmFlags.addElemToolInfo(TRUE); // default FALSE
mmFlags.addConstructionToolInfo(TRUE); // default FALSE
mmFlags.addHardwareInfo(TRUE); // default FALSE
mmFlags.addRoofplanesAboveWallsAndRoofSectionsForRoofs(TRUE); //
default FALSE
mmFlags.addCollectionAndBlockDefinitions(TRUE); // default FALSE
```

---

```
mmFlags.addAllGroups(TRUE); // default FALSE
mmFlags.addUniqueIdsAsHandle(TRUE); // default FALSE

—end sample]
```

## 5.26.2 ModelXInterpretSettings

The ModelXInterpretSettings type represents a set of options that can be used during the import of a [ModelX](#) into the drawing.

The ModelXInterpretSettings is typically used in the dblInterpretMap method of [ModelX](#):  
**ModelX::dblInterpretMap(ModelXInterpretSettings flags);**

---

```
class ModelXInterpretSettings { // (added hsbCAD14.0.75 and renamed in V25.1.120 and
V26.2.10)

    void resolveEntitiesByHandle(int bSet); // set to TRUE or FALSE. The default value is
        FALSE.
    void resolveElementsByNumber(int bSet); // set to TRUE or FALSE. The default value is
        FALSE.
    void setBeamTypeNameAndColorFromHsblId(int bSet); // set to TRUE or FALSE. The default
        value is FALSE.
    void importProjectInfo(int bSet); // set to TRUE or FALSE. The default value is FALSE
        (added v21.0.125)
    void allowXrefContentToBeModified(int bSet); // set to TRUE or FALSE. The default value is
        FALSE. (added v22.0.106)

    // The following 2 methods are obsolete and effectively do-nothing-methods since V24.0.11.
    // They now get automatically set with resolveEntitiesByHandle.
    // void resolveHandlesAsUniquelds(int bSet); // set to TRUE or FALSE. The default value is
        FALSE. (added v22.0.106)
    // void storeUniqueldsOfCreatedEntities(int bSet); // set to TRUE or FALSE. The default
        value is FALSE. (added v22.0.106)

};
```

---

[Sample.

```
ModelXInterpretSettings mmFlags;
mmFlags.resolveEntitiesByHandle(TRUE); // default FALSE
mmFlags.resolveElementsByNumber(TRUE); // default FALSE
mmFlags.setBeamTypeNameAndColorFromHsblId(TRUE); // default FALSE
mmFlags.importProjectInfo(TRUE); // default FALSE

—end sample]
```

## 5.27 Performance profiling

When tsl's become more and more complex, performance might degrade. To investigate this, hsbCAD (since hsbCAD2011 build 16.0.9) has now a set of commands to start and end collection of performance data.

The commands are:

```
HSB_TIMETRACKSTART  
HSB_TIMETRACKSTARTPROFILING  
HSB_TIMETRACKEND  
HSB_TIMETRACKENDANDFILE
```

Some commands, like the new generate construction and the refresh of a multipage, allow to turn on time tracking by pressing the left shift key on the keyboard while firing the command. Profiling can be started by pressing the right shift key. Time tracking is automatically turned off at the end of the command in case you pressed shift. You should not mix the commands to start time tracking with pressing the shift during the command start. This will give you invalid readings.

There is a small performance cost when running with time tracking, but the overhead is minimal. When profiling, the overhead is considerable bigger.

When time tracking stops (eg during command HSB\_TIMETRACKEND and HSB\_TIMETRACKENDANDFILE), the tracking results are output either on the report dialog or into a file. For each tsl the time spend in executing it is reported in milliseconds. Also the amount of calls is also reported.

When profiling is done, timers are added for each function call. When 2 counters run simultaneous, both timers will accumulate. So when one parent tsl calls a scriptInsert on a child tsl, the timetracker of the parent one also takes the time executed in the child tsl into account.

When the output is imported in excel, it is easier to analyse. The result looks like the following.

A	B	C	D
name	readable	time(ms)	count
1 GenerateConstruction:000\4216-1	46917 ms for 1 calls	46917	1
2 Tsl:TIS_DET_SHEATHING	42566 ms for 6 calls	42566	6
3 TIS_DET_SHEATHING:Beam::realBody	2300 ms for 1300 calls	2301	1300
4 TIS_DET_SHEATHING:Beam::envelopeBody	1395 ms for 466 calls	1395	466
5 TIS_DET_SHEATHING:PlaneProfile::shrink	1269 ms for 130 calls	1269	130
6 Tsl:TIS_DET_NAILING	853 ms for 5 calls	854	5
7 TIS_DET_SHEATHING:Body::hasIntersection	697 ms for 4488 calls	698	4488
8 TIS_DET_SHEATHING:Body::getSlice	549 ms for 76 calls	550	76
9 TIS_DET_SHEATHING:PLine::projectPointsToPlane	398 ms for 2964 calls	399	2964
10 TIS_DET_SHEATHING:PLine::vertexPoints	393 ms for 2992 calls	393	2992
11 TIS_DET_SHEATHING:Sheet::realBody	345 ms for 316 calls	345	316
12 TIS_DET_SHEATHING:Vector3d::dotProduct	291 ms for 4098 calls	291	4098
13 TIS_DET_SHEATHING:Element::addTool	285 ms for 110 calls	286	110
14 Tsl:TIS_DET_GRADING	221 ms for 2 calls	222	2
15 TIS_DET_SHEATHING:PlaneProfile::allRings	150 ms for 128 calls	151	128
16 TIS_WAF_COMPONENT_DETAILS:Element::vecX	150 ms for 3384 calls	150	3384
17 TIS_WAF_COMPONENT_DETAILS:Element::ptOrg	142 ms for 3060 calls	143	3060
18 TIS_DET_SHEATHING:Beam::solidLength	134 ms for 168 calls	134	168
19 Tsl:TIS_DET_LADDERS_TMP	132 ms for 1 calls	132	1
20 TIS_DET_LADDERS_TMP:TslInst::dbCreate	126 ms for 1 calls	126	1
21 TIS_DET_SHEATHING:Body::transformBy	115 ms for 292 calls	115	292
22 TIS_WAF_COMPONENT_DETAILS:Beam::ptCen	115 ms for 3114 calls	115	3114
23 TIS_DET_SHEATHING:Sheet::dbSplit	105 ms for 30 calls	105	30
24 TIS_DET_NAILING:Display::draw	101 ms for 640 calls	101	640
25 TIS_DET_SHEATHING:Beam::ptCen	92 ms for 656 calls	92	656
26 TIS_WAF_COMPONENT_DETAILS::eraseInstance	73 ms for 19 calls	74	19
27 TIS_WAF_COMPONENT_DETAILS::eraseInstance	69 ms for 200 calls	69	200

## 5.28 Push command on stack

To push a command on the command stack of the current document use the `pushCommandOnCommandStack`. The current Tsl will not wait for the command to execute. Instead, the command will be pushed on the command stack. Whenever the current command has ended, the command stack interpreter kicks in, executing whatever is left. As you might know, lisp commands can also be pushed on the command stack. Lisp is required whenever you need to pass arguments to the command.

The autocad command stack has a limited capacity.

Also, some commands require user input through the command line. Such commands will interfere with whatever is found on the command stack.

```
void pushCommandOnCommandStack(String strCommand); // (added hsbCAD16.0.11)
```

*[Sample of O-type that illustrates the `pushCommandOnCommandStack`.*

```
String strContext = T("|Insert tsl|");
addRecalcTrigger(_kContext, strContext );
if (_bOnRecalc && _kExecuteKey==strContext ) {
    pushCommandOnCommandStack("_HSB_MACROSELECT");
}
```

---

—end sample]

## 5.29 TestReport

The TestReport class is part of the testing framework build in hsbCAD. The purpose is to add a test to the framework. When running in debug mode, tests output to the screen. But the main purpose is to report its results during the command Hsb\_RunTests.

When the command Hsb\_RunTests runs, the state variable `_bOnRunTests` is TRUE.

To participate in the Hsb\_RunTests command, the tsl instance needs to set the recalc trigger. The recalc trigger is automatically added when a TestReport is instantiated with a test name. It can also be added explicitly however.

```
addRecalcTrigger(_kRunTestsEvent, "");
```

---

```
class TestReport { // added build 18.0.5

    TestReport(String strTestName);

    int countFailed(); // return the number of tests failed (added 23.5.22)
    int countSuccess(); // return the number of tests that worked (added 23.5.22)

    // the assert methods return TRUE or FALSE. TRUE if the test was successful (added
    // 23.5.22)
    int assertEquals(int n1, int n2);
    int assertEquals(double d1, double d2); // compare with tolerance
    int assertEquals(Point3d pt1, Point3d pt2); // compare with tolerance
    int assertEquals(Vector3d vec1, Vector3d vec2); // compare with tolerance
    int assertEquals(Point3d[] arPnts1, Point3d[] arPnts2); // compare list of points but
        independent of sequence

    int assertEqualsCase(String str1, String str2); // compare respecting case sensitivity
    int assertEqualsNoCase(String str1, String str2); // compare not case sensitive

    int assertTrue(int bVal);
    int assertFalse(int bVal);
}
```

---

[Example O-type TSL:

```
U(1, "mm");
```

---

```

if (_bOnInsert) {
    _Pt0 = getPoint();
    _GenBeam.append(getGenBeam());
    return;
}

GenBeam bm = _GenBeam[0];

{
    TestReport trep("testGenBeamGeometry");
    trep.assertEqual(bm.dL(), 1000);
    trep.assertEqual(bm.dW(), 100);
    trep.assertEqual(bm.dH(), 200);
    trep.assertEqual(bm.ptCen(), Point3d(500,0,-100));
    trep.assertEqual(bm.vecX(), Vector3d(1,0,0));
    trep.assertEqual(bm.vecY(), Vector3d(0,1,0));
    trep.assertEqual(bm.vecZ(), Vector3d(0,0,1));

    Body bd(bm.ptCen(), bm.vecX(), bm.vecY(), bm.vecZ(), bm.dL(),
            bm.dW(), bm.dH());
    Body bdBeamEnv = bm.envelopeBody();
    trep.assertEqual(bd.allVertices(), bdBeamEnv.allVertices());
}

{
    TestReport trep("testGenBeamPosnum");
    trep.assertEqual(bm.posnum(), -1);
}

```

—end example]

[Example O-type TSL:

```

reportNotice("\nRunning "+scriptName()+" at: "+_Pt0+
_bOnRunTests=_bOnRunTests;
addRecalcTrigger(_kRunTestsEvent, "");

```

—end example]

## 5.30 ParentChildGrouping

Since hsbcad v20.0.80 the hsbCAD Hundegger BVX2 exporter interface contains the possibility to export nested sheeting. How the sheeting is nested, by which methods or tools is basically open, But in order for the exporter to pick up the nesting info, it needs to be formatted in a certain way. The following explains the data structure of this grouping info, and how this data structure could be created through the use of Tsl's.

Below is an image of the required data. Nesting child and nesting parent entities are discovered by the presence of MapX data. The grouping child and grouping parent can be both Tsl's.



Parent child grouping could be of type Nesting, Stacking... The MapX name for nesting is Hsb\_NestingChild and Hsb\_NestingParent.

For the grouping child, the

- Entity: refers to the entity being grouped. So for sheet nesting, this would refer to the sheet.
- ParentUid: is the unique identifier of the parent. It matches the MyUid of the parent.
- ptRelOrg, ptVecX, ptVecY, ptVecZ: refer to a relative coordinate system, stored using 4 points. It expresses the relative position of the child within the parent. Because of this, they should be stored as absolute points.

For the grouping parent, the

- MyUid: refers to the unique identifier of the master. Here in this Tsl it uses the handle of the parent ts1 entity as string.
- ptOrg, vecX, vecY and vecZ: express a quader positioned in space. The vectors express the dimensions of the quader, so they are scalable.

[Example O-type TSL: NestingChild

```
U(1, "mm");

if (_bOnInsert)
{
    _Sheet.append(getSheet(T("|Select the sheet to be nested|")));
    _Entity.append(getTslInst(T("|Select the parent sheet ts1|")));
    _Pt0 = getPoint(T("|Select the location to place the sheet|"));
    return;
}

if (_Entity.length() == 0 || _Sheet.length() == 0)
{
    eraseInstance();
    return;
}

String strChangeEntity = T("|Change sheet entity|");
addRecalcTrigger(_kContext, strChangeEntity);
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    _Sheet[0] = getSheet(T("|Select the sheet to be nested|"));
}

Entity parentEnt = _Entity[0];
Sheet sheetEnt = _Sheet[0];
if (!parentEnt.bIsValid() || !sheetEnt.bIsValid())
{
```

```

        eraseInstance();
        return;
    }

    setDependencyOnEntity(parentEnt);
    setDependencyOnEntity(sheetEnt);

    CoordSys csThis = ThisInst.coordSys();
    CoordSys csSheet(sheetEnt.ptCenSolid(), sheetEnt.vecX(),
    sheetEnt.vecY(), sheetEnt.vecZ());
    CoordSys csParent = parentEnt.coordSys();

    csThis.vis(1);
    csSheet.vis(3);
    csParent.vis(4);

    CoordSys csModel2Parent = csSheet;
    csModel2Parent.invert();
    csModel2Parent.transformBy(csThis);

    CoordSys csParentInv = csParent;
    csParentInv.invert();
    CoordSys csChildRel = csThis;
    csChildRel.transformBy(csParentInv);
    //csChildRel.vis(5);

    Display dp(-1);

    // draw body of child
    Body bdSheet = sheetEnt.realBody();
    bdSheet.transformBy(csModel2Parent);
    dp.draw(bdSheet);

    // use the csChildRel to construct the transformation from Model to
    Parent
    Display dp2(5);
    Body bdSheet2 = sheetEnt.realBody();
    CoordSys csModel2Parent2 = csSheetInv;
    csModel2Parent2.transformBy(csChildRel);
    csModel2Parent2.transformBy(csParent);
    bdSheet2.transformBy(csModel2Parent2);
    dp2.draw(bdSheet2);

    if (_bOnDebug)
    {
        String str = String(sheetEnt.posnum()) + " " +
        csChildRel.ptOrg();
        dp.draw(str, csThis.ptOrg(), csThis.vecX(), csThis.vecY(), 0, 1);
    }

    // mark sheet that it is nested by drawing circle
    PLine plCirSheet;
    plCirSheet.createCircle(csSheet.ptOrg(), csSheet.vecZ(), U(100));

```

```

PLine plDiagSheet(csSheet.ptOrg() + U(100) * csSheet.vecX(),
csSheet.ptOrg() - U(100) * csSheet.vecX());
dp.draw(plCirSheet);
dp.draw(plDiagSheet);

// compose mapX with Nesting info
Map mapX;
mapX.setEntity("Entity", sheetEnt);
mapX.setString("ParentUid", parentEnt.handle());
mapX.setPoint3d("ptRelOrg", csChildRel.ptOrg(), _kAbsolute);
mapX.setPoint3d("ptVecX", csChildRel.ptOrg() + csChildRel.vecX(),
_kAbsolute);
mapX.setPoint3d("ptVecY", csChildRel.ptOrg() + csChildRel.vecY(),
_kAbsolute);
mapX.setPoint3d("ptVecZ", csChildRel.ptOrg() + csChildRel.vecZ(),
_kAbsolute);

// (over)write submapX
_ThisInst.setSubMapX("Hsb_NestingChild", mapX);

```

—end example]

[Example O-type TSL: NestingParent

```

U(1, "mm");
PropDouble dSizeX(0, U(4000));
PropDouble dSizeY(1, U(2000));
PropDouble dSizeZ(2, U(100));

CoordSys csThis = _ThisInst.coordSys();
csThis.vis(1);

CoordSys cs = CoordSys(csThis.ptOrg(), dSizeX * csThis.vecX(),
dSizeY * csThis.vecY(), dSizeZ * csThis.vecZ());
Body bdBox = Body(cs.ptOrg(), cs.vecX(), cs.vecY(), cs.vecZ(),
1,1,1);

// draw rectangle
Display dp(-1);
dp.draw(bdBox);

// compose mapX with Nesting info
Map mapX;
mapX.setString("MyUid", _ThisInst.handle());
mapX.setPoint3d("ptOrg", cs.ptOrg(), _kRelative);
mapX.setVector3d("vecX", cs.vecX(), _kScalable); // coordsys carries
size
mapX.setVector3d("vecY", cs.vecY(), _kScalable);
mapX.setVector3d("vecZ", cs.vecZ(), _kScalable);

// (over)write submapX
_ThisInst.setSubMapX("Hsb_NestingParent", mapX);

```

—end example]

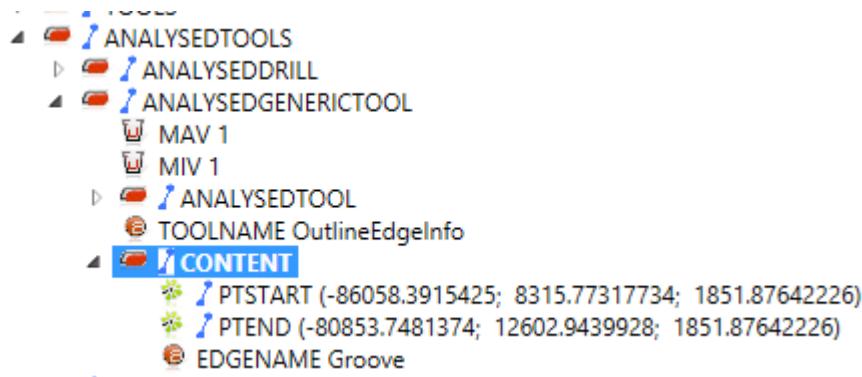
## 5.31 OutlineEdgeInfo

Bvx export of Sheeting supports EdgeName to be set per segment of the outline profile. Tsl can set this info using a [CncExport](#) tool.

The ptStart and ptEnd, and optionally a ptMid for arc segments, identifies a segment on which the start, end en mid points of the plEnvelope segment must lie, in order for the edge to get assigned the EdgeName value in Bvx. The segment defined by the CncExport tool can be longer than the plEnvelope segment, but not shorter.

If there are multiple EdgeNames for the plEnvelope segment defined, then only one will be taken.

The presence of CncExport data is not taken into account for Posnum generation. So it is important that the Tsl adds additional body manipulations or other Posnum sensitive data in order for the Posnum to be correct.



[Example O-type TSL: OutlineEdgeInfo

```
U(1, "mm");

PropString pEdgeName(0, "Groove", T("|Edge name|"));

if (_bOnInsert) {

    _Pt0 = getPoint(T("|Select first point|")); // select point
    _PtG.append(getPoint(T("|Select second point|")));
    _Sheet.append(getSheet());

    return;
}

string strChangeEntity = T("|Add more sheeting entities|");
addRecalcTrigger(_kContext, strChangeEntity);
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
```

```

PrEntity ssE(T("|Select a set of sheets|"), Sheet());
if (ssE.go()) {
    Sheet ssSheets[] = ssE.sheetSet();
    Sheet.append(ssSheets);
}
}

if (Sheet.length() == 0 || PtG.length() == 0) {
    eraseInstance();
    return;
}

Display dp(-1);
// draw segment
dp.color(2); // yellow
dp.draw(PLine(Pt0, PtG[0]));

// export tool info as CncExport
for (int b=0; b<Sheet.length(); b++)
{
    Sheet sht = Sheet[b];

    Map map;
    map.setPoint3d("ptStart", Pt0);
    map.setPoint3d("ptEnd", PtG[0]);
    map.setString("EdgeName", pEdgeName);

    CncExport tl("OutlineEdgeInfo", map);
    sht.addTool(tl);
}

```

—end example]

## 5.32 XrefLocker

The XrefLocker is a helper class to require a lock on an external database. A lock is required to edit or modify any of the external database its entities or objects.

The lock on the database will be released automatically whenever this XrefLocker goes out of scope, or when the Tsl ends.

---

```

class XrefLocker // added hsbcad v22.0.106
{
    XrefLocker();

    int lockDatabase(AcadDatabase db); // if db is an AcadDatabase, get a lock on it. Return the
                                         // error code, 0 for success.
    int lockDatabaseOf(Entity ent); // get a lock on the database of the entity

```

---

```
};
```

---

[Sample of illustrating XrefLocker:

```
Unit(1, "mm");
if (_bOnInsert)
{
    int bAllowNested = TRUE;
    _Entity.append(getEntity(T("|Select entity|"), bAllowNested));
    _Pt0 = getPoint(); // select a point
    return;
}

Entity ent = _Entity[0];

PropString strMapXName(0, "MyMapX", T("|MapX name|"));
PropString strMapKeyString(1, "MyKey", T("|Map key string|"));
PropString strMapViewValue(2, "MyValue", T("|Map string value|"));

String strAddMapMapX = T("|add MapX|");
addRecalcTrigger(_kContext, strAddMapMapX );
if (_bOnRecalc && _kExecuteKey==strAddMapMapX)
{
    XrefLocker locker;
    int nLockErr = locker.lockDatabaseOf(_Entity[0]);
    reportMessage("\\nLock err: "+ nLockErr);

    // compose a map
    Map mapw;
    mapw.setString(strMapKeyString, strMapViewValue);

    // (over)write submap of entity
    _Entity[0].setSubMapX(strMapXName, mapw);
    reportMessage("\\n subMap written: " + strMapXName);
}

String strRemoveMapX = T("|remove MapX|");
addRecalcTrigger(_kContext, strRemoveMapX );
if (_bOnRecalc && _kExecuteKey == strRemoveMapX)
{
    XrefLocker locker;
    int nLockErr = locker.lockDatabaseOf(_Entity[0]);
    reportMessage("\\nLock err: "+ nLockErr);

    int bRemoved = _Entity[0].removeSubMapX(strMapXName);
    reportMessage("\\n removed subMap" + strMapXName + ":" + bRemoved);
}

// Now interrogate map
```

---

```

String arKeys[] = ent.subMapXKeys();
reportMessage("\\n number of keys: "+arKeys.length());
for (int k=0; k<arKeys.length(); k++) {
    reportMessage("\\n MapXKey "+k+": "+arKeys[k]);
}
Map mapr = ent.subMapX(strMapXName);
reportMessage("\\n" + strMapKeyString + ":" + 
mapr.getString(strMapKeyString));

```

*—end sample]*

## 5.33 Grip

The Grip class allows to manage enhanced grip points. In contrast to the `_PtG`, the new grip points have tool tips, shape and color.

`TsInst` has a method to get and set its grip points.

```

Grip[] grips() const; // get the list
void setGrips(Grip[] arGrip); // set the list. BEWARE!! _ThisInst.setGrips will not work if
                                // _Grip is used in the script. Modifications to _Grip have preference.

```

There is also the predefined `_Grip` which actually is initialized at the beginning of the ts1 its execution with `_Grip = _ThisInst.grips()`. And at the end of the ts1 its execution, the `_ThisInst.setGrips(_Grip)` is called to set the grips from the modifications of `_Grip`.

---

```

class Grip // added hsbcad v25.1.83
{
    Grip(Point3d ptLoc); // initialize with the location of the grip point

    int shapeType() const;
    void setShapeType(int val);

    String name() const;
    void setName(String val);
    String toolTip() const;
    void setToolTip(String val);

    Point3d ptLoc() const; // the location of the grip point
    void setPtLoc(Point3d val);

    Vector3d vecX() const;
    void setVecX(Vector3d val);
    Vector3d vecY() const;
    void setVecY(Vector3d val);

    int color() const;
    void setColor(int val); // if not set, the default grip color is used

```

```

double scale() const;
void setScale(double val); // the default grip size is scaled with this value. Default 1.

int isRelativeToEcs() const;
void setsRelativeToEcs(int val); // Grips are normally relative to the coordSys of the
                                // TsInstance. But can be stored absolute, aka relative to WCS. Default TRUE.

int isStretchPoint() const;
void setsStretchPoint(int val); // Grips can also react to stretch command. Default FALSE.

void addViewDirection(Vector vecDirectionTowardsViewer);
void addHideDirection(Vector vecDirectionTowardsViewer);
Vector[] viewDirections() const;
Vector[] hideDirections() const;

Vector3d vecOffsetApplied() const; // if this grip is moved, the offset is applied to the ptLoc, but
                                // also the offset is returned with this method.

int isMoved() const; // if this grip is moved, this returns TRUE, else FALSE.

static int indexOfMovedGrip(Grip[] arGrip); // returns the index in the arGrip which is moved, if
                                                // not found it returns -1.

};


```

Values of shapeType are:

```

_kGSTAcad
_kGSTArrow
_kGSTCircle
_kGSTDiamond
_kGSTLine
_kGSTMinus
_kGSTPlus
_kGSTRotate
_kGSTSquare
_kGSTSquare45
_kGSTStar
_kGSTTriangle
_kGSTTriangleIso

```

---

[Sample of illustrating Grip:

```

U(1, "mm");

String arShapeTypes[] = { T("|Acad|"), T("|Arrow|"), T("|Circle|"),
T("|Diamond|"), T("|Line|"),
T("|Minus|"), T("|Plus|"), T("|Rotate|"), T("|Square|"), T("|"
Square45|"),
T("|Star|"), T("|Triangle|"), T("|TriangleIso|") };
PropString pShapeType(0, arShapeTypes, T("|Shape type|"));
int arShapeTypeInts[] = { _kGSTAcad, _kGSTArrow, _kGSTCircle,
_kGSTDiamond, _kGSTLine,

```

---

```

_kGSTMinus, _kGSTPlus, _kGSTRotate, _kGSTSquare, _kGSTSquare45,
_kGSTStar, _kGSTTriangle, _kGSTTriangleIso};
int nShapeType = arShapeTypeInts[arShapeTypes.find(pShapeType, 0)];

PropString pName(1, "my name", T("|Name|"));
PropString pToolTip(2, "my tooltip", T("|Tooltip|"));
PropInt pColor(0, 3, T("|Color index|"));
PropDouble pScale(0, 1.5, T("|Scale|"));

String arSYesNo[] = {T("|Yes|"), T("|No|")};
int arNYesNo[] = {_kYes, _kNo};
PropString pIsRelativeToEcs(3, arSYesNo, T("|IsRelativeToEcs|"));
PropString pIsStretchPoint(4, arSYesNo, T("|IsStretchPoint|"));

String arWorldUcsNone[] = {T("|World|"), T("|UCS|"), T("|None|")};
PropString pUseWorld(5, arWorldUcsNone, T("|Use world, or ucs, or
none|"));

String arDirections[] = {T("|None|"), T("|WorldZ|"), T("|WorldX|")};
PropString pViewDirections(6, arDirections, T("|Assign view
directions|"));
PropString pHideDirections(7, arDirections, T("|Assign hide
directions|"));

_ThisInst.setAllowGripAtPt0(FALSE);

String strChangeEntity0 = T("|Redefine grip 0|");
String strChangeEntity1 = T("|Redefine grip 1|");
addRecalcTrigger(_kContext, strChangeEntity0);
addRecalcTrigger(_kContext, strChangeEntity1);
if (_bOnRecalc && (_kExecuteKey==strChangeEntity0 ||
_kExecuteKey==strChangeEntity1))
{
    Point3d ptG = _Pt0 + U(100) * _XU;
    Vector3d vX = _XU;
    Vector3d vY = _YU;
    if (pUseWorld == arWorldUcsNone[0])
    {
        vX = _XW;
        vY = _YW;
    }
    else if (pUseWorld == arWorldUcsNone[2])
    {
        vX = Vector3d(0, 0, 0);
        vY = Vector3d(0, 0, 0);
    }
}

Grip grip(ptG);
grip.setShapeType(nShapeType);
grip.setName(pName);
grip.setToolTip(pToolTip);
grip.setVecX(vX);
grip.setVecY(vY);

```

```

        grip.setColor(pColor);
        grip.setScale(pScale);
        grip.setIsRelativeToEcs(pIsRelativeToEcs == arSYesNo[0]);
        grip.setIsStretchPoint(pIsStretchPoint== arSYesNo[0]);

        if (pViewDirections == arDirections[1])
            grip.addViewDirection(_ZW);
        if (pViewDirections == arDirections[2])
            grip.addViewDirection(_XW);
        if (pHideDirections == arDirections[1])
            grip.addHideDirection(_ZW);
        if (pHideDirections == arDirections[2])
            grip.addHideDirection(_XW);

        if (_Grip.length() < 2) _Grip.append(grip);
        if (_Grip.length() < 2) _Grip.append(grip);
        if (_kExecuteKey==strChangeEntity0)
            _Grip[0] = grip;
        else
            _Grip[1] = grip;
    }

    if (_bOnGripPointDrag && _kExecuteKey=="_Grip")
    {
        String strIndices = "";
        for (int i = 0; i < _Grip.length(); i++)
        {
            if (_Grip[i].isMoved())
            {
                if (strIndices.length() != 0) strIndices += " - ";
                strIndices += i;
            }
        }
        reportMessage("\n" + scriptName()
            + ", execute key: " + _kExecuteKey
            + ", bOnGripPointDrag: " + _bOnGripPointDrag
            + " kNameLastChangedProp: " + _kNameLastChangedProp
            + "strIndices: " + strIndices);

        int iGrip = Grip().indexOfMovedGrip(_Grip);
        if (iGrip < 0)
            return;

        Point3d ptBase = _Pt0;
        Point3d ptDrag = _Grip[iGrip].ptLoc();

        _ThisInst.setDebug(TRUE); // sets the debug state on the dragging
        TslInstance only, if you want to see the effect of vis methods
        ptDrag.vis(1);

        Vector3d vecXV = getViewDirection(0);
        vecXV.vis(ptDrag);
        Vector3d vecYV = getViewDirection(1);
    }
}

```

```

vecYV.vis(ptDrag);
Vector3d vecZV = getViewDirection(2);
vecZV.vis(ptDrag);

double dRad = Vector3d(ptDrag - ptBase).length();

Display dpJ(1);
PLine plCir;
plCir.createCircle(ptDrag, _ZU, dRad/2);
dpJ.draw(plCir);

dpJ.color(2);
plCir.createCircle(ptBase, _ZU, dRad);
dpJ.draw(plCir);

dpJ.color(3);
dpJ.transparency(50);
plCir.createCircle(ptDrag, vecZV, 0.5*getViewHeight());
dpJ.draw(plCir);

return;
}
addRecalcTrigger(_kGripPointDrag, "_Grip");

if (_kNameLastChangedProp == "_Grip")
{
    String strIndices = "";
    for (int i = 0; i < _Grip.length(); i++)
    {
        if (_Grip[i].isMoved())
        {
            if (strIndices.length() != 0) strIndices += " - ";
            strIndices += i;
        }
    }
    reportMessage("\n" + scriptName()
        + ", execute key: " + _kExecuteKey
        + ", bOnGripPointDrag: " + _bOnGripPointDrag
        + " kNameLastChangedProp: " + _kNameLastChangedProp
        + " strIndices: " + strIndices);
}

int iGrip = Grip().indexOfMovedGrip(_Grip);
reportMessage("\n"+ "Last grip index moved: " + iGrip);

if (iGrip == 0)
{
    Point3d ptNew = _Grip[iGrip].ptLoc();
    Vector3d vecDelta = _Grip[iGrip].vecOffsetApplied();
    Point3d ptOld = ptNew - vecDelta;
    vecDelta = vecDelta.dotProduct(_XU) * _XU;
    ptNew = Line(ptOld, _XU).closestPointTo(ptNew);

    _Grip[iGrip].setPtLoc(ptNew);
}

```

```

        }

        if (iGrip == 1)
        {
            Vector3d vecRad = Grip[iGrip].ptLoc() - Pt0;
            Vector3d vecPerp = ZE.crossProduct(vecRad);
            Grip[iGrip].setVecX(vecRad.normal());
            Grip[iGrip].setVecY(vecPerp.normal());
        }
    }

CoordSys csEcs(PtE, XE, YE, ZE);
csEcs = ThisInst.coordSys();
csEcs.vis(1);
//Pt0.vis(1);

Display dp(-1);
dp.textHeight(U(10));
dp.draw(scriptName(), Pt0, XU, YU, 1, 1);

for (int i = 0; i < Grip.length(); i++)
{
    Display dpI(Grip[i].color());
    Vector3d arV[] = Grip[i].viewDirections();
    for (int v = 0; v < arV.length(); v++)
    {
        dpI.addViewDirection(arV[v]);
    }
    Vector3d arH[] = Grip[i].hideDirections();
    for (int v = 0; v < arH.length(); v++)
    {
        dpI.addHideDirection(arH[v]);
    }
    dpI.draw(PLine(Pt0, Grip[i].ptLoc()));
}
PLine plCir;
plCir.createCircle(Pt0, ZU, U(10));
dp.draw(plCir);

```

*—end sample]*

## 5.34 Tag

Entities and Tools can carry MapX data. One special MapX is the "Hsb\_Tag" one. All entries in this map with name "Tag" are considered a tag.

*[Sample illustrating how to add some tags and report existing:*

```

if (bOnInsert)
{
    Entity ent = getEntity();

    // read the existing tags
    String strApp = "Hsb_Tag";

```

```

Map mpX = ent.subMapX(strApp);
String tags[0];
for (int i = 0; i < mpX.length(); i++)
{
    if (mpX.keyAt(i).makeUpper() == "TAG" && mpX.hasString(i))
        tags.append(mpX.getString(i));
}

// report the tags read
{
    String strTags = "\nTags read: ";
    for (int t = 0; t < tags.length(); t++)
        strTags += (t == 0) ? tags[t] : ", " + tags[t];;
    reportNotice(strTags);
}

// add some new tags
{
    String newTag = "abc";
    if (tags.findNoCase(newTag) == -1)
        tags.append(newTag);
}
{
    String newTag = "abcd";
    if (tags.findNoCase(newTag) == -1)
        tags.append(newTag);
}

// write the tags
mpX = Map();
for (int t = 0; t < tags.length(); t++)
    mpX.appendString("Tag", tags[t]);
ent.setSubMapX(strApp, mpX);

// report the tags written
{
    String strTags = "\nTags written: ";
    for (int t = 0; t < tags.length(); t++)
        strTags += (t == 0) ? tags[t] : ", " + tags[t];;
    reportNotice(strTags);
}

return;
}

```

*—end sample]*

## 5.35 OnDbErase

Since V27.3.6 and V28 a TsInstance can react on being erased from the Autocad database. The erase event happens during:

- the erase command
- the undo (Ctrl-Z) and redo (Ctrl-Y) commands of database adding
- undo of block exploding
- clipcopy (Ctrl-C and Ctrl-V) or clipcut (Ctrl-X and Ctrl-V)
- delete construction
- beam manipulations such as stretch, if the tsl controls the end tools
- block editor, and block in place editor
- ... and many other events and commands.

To subscribe to the erase event, write:

```
addRecalcTrigger(_kErase, TRUE);
```

The erase can also be undone eg during a redo (Ctrl-Y), which we call un-erase. Typically the un-erase will just restore the state of the erased entities by the filing process. If you modify the database during that event, further undo's will be prevented. So, it is not recommended to subscribe to the un-erase event.

```
addRecalcTrigger(_kErase, FALSE); // un-erase not recommended
```

When subscribed, when the TsInstance is erased from the database, the tsl gets executed with the flag `_bOnDbErase`, or `_bOnDbUnErase` set to TRUE.

---

[Sample of registering for `_kErase`:

```
addRecalcTrigger(_kErase, TRUE); // erase
addRecalcTrigger(_kErase, FALSE); // un-erase not recommended

if (_bOnDbCreated)
    _Map.setInt("cnt", 10);

if (_bOnDbErase)
    _Map.setInt("cnt", _Map.getInt("cnt") - 1);
if (_bOnDbUnErase)
    _Map.setInt("cnt", _Map.getInt("cnt") + 1);

reportMessage("\nTslName: " + "!" + scriptName() + "!"
+ ", handle: " + "!" + _ThisInst.handle() + "!"
+ ", execute key: " + "!" + _kExecuteKey + "!"
+ ", _bOnDbCreated: " + _bOnDbCreated
+ ", _bOnDbErase: " + _bOnDbErase
+ ", _bOnDbUnErase: " + _bOnDbUnErase
+ ", _bOnInsert: " + _bOnInsert
+ ", cnt: " + _Map.getInt("cnt")
);

if (_bOnInsert)
{
    _Pt0 = getPoint();
```

```

    return;
}

—end sample]

```

---

## 5.36 Fastener graphics

Fasteners are graphically represented by using the displays of a tsIs. In this case the tsl serves as a recipe to display a certain fastener component.

The graphics is generated for the combination of FastenerComponentData and FastenerArticleData. It is stored in a cache against the article number, assuming the graphics for a certain article is unique.

There is a ruleset which defines the assignment of which Tsl is executed to define the graphical representation of a component. The ruleset is stored distributed in the following files:

```

<hsbInstall>\FastenerAssembly\*.grs
<hsbCompany>\FastenerAssembly\*.grs

```

The following autocad commands help here to edit and view the rule set:

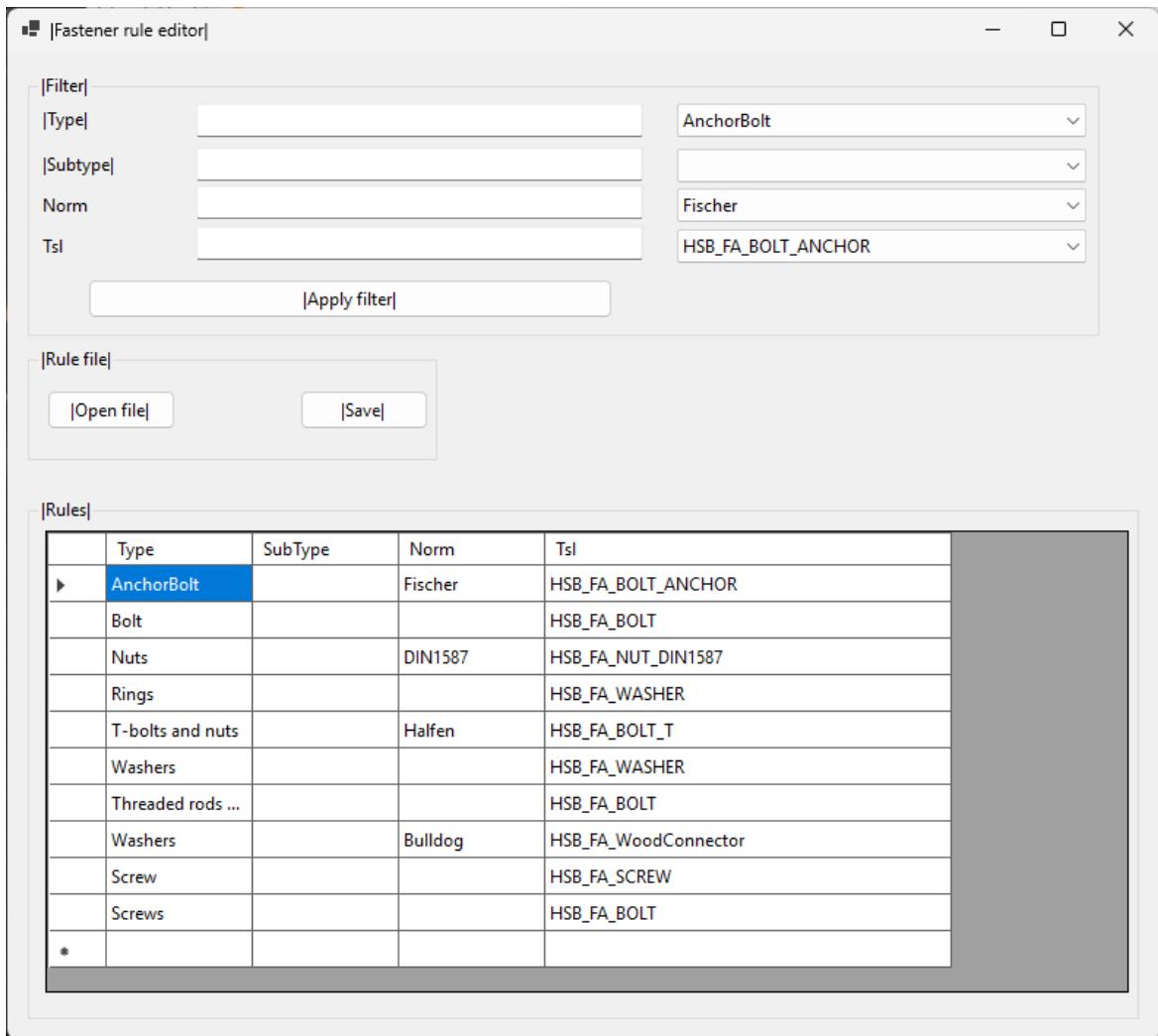
HSB\_FA\_DUMPGRAPHICSRULESET => to show the active rule set, dumped to the autocad console

HSB\_FA\_IMPORTGRAPHICSRULESET => delete the current rule set, and reload the \*.grs files from the locations above.

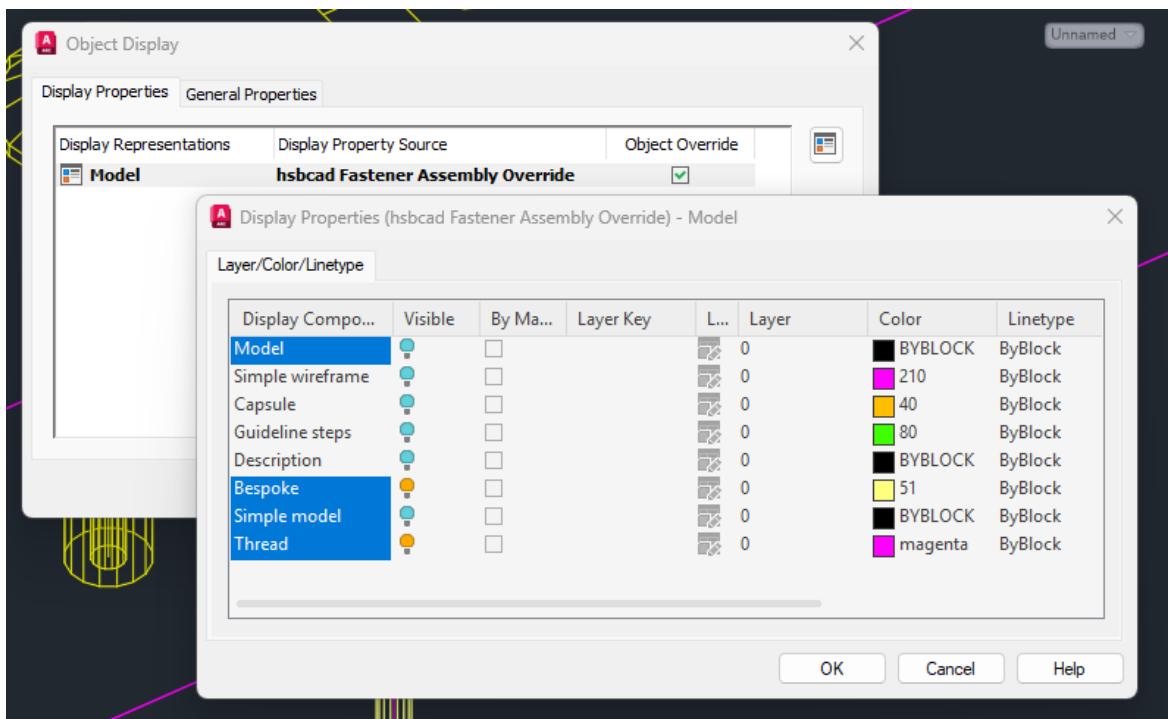
After loading in a new ruleset, graphics will need to be regenerated => RegenAll.

There is a small app that allows to edit grs files. It can be found in:

```
<hsbInstall>\Utilities\FastenerGraphicalRuleEditor\hsbFastenerGraphicalRuleEditor.exe
```



The HSBCAD\_FASTENERASSEMBLY has a few display components. What is shown for "Model", "Bespoke", "Simple Model" and "Thread" is controlled by the Tsl. "Simple Model" and "Thread" have been added since V28.0.9.



The Tsl needs to specify a Display which sets its state to either `_kSimpleModel`, `_kBespoke`, `_kThread`, or empty "" meaning the default model component by using `showInDispComp` of [Display](#).

```
Display dpS(-1);
dpS.showInDispComp(_kThread);
```

If the display specifies color -1, the color of the display component is used. If the display component refers to "ByBlock", the entity its color is used.

#### [Example O-type TSL that could be used

```
/// <summary Lang=en>
/// This Tsl creates the solid for the designated type of an fastener
/// assembly component and
/// it is not designed to be inserted into the drawing.
/// </summary>

/// <insert=en>
/// This TSL is not designed to be inserted in the drawing. It will
/// be used automatically by the fastener assembly framework.
/// If it is inserted in the drawing it will only be used for debug
/// purposes.
/// </insert>

// basics and props
Unit(1, "mm");
double dEps = U(.1);
```

```

    int bInDebug = false;

    // debug
    if (bInDebug) {
        reportMessage("\nTsl execution: " + scriptName() + ":" + 
_kExecuteKey); // + " " + _bOnDrawFastenerComponent);
        // export map
        //_Map.writeToDxxFile(_kPathDwg + "\\"+ scriptName() + 
".dxx", FALSE);
    }

    // get values from the FastenerAssembly through the _Map
    String strMapPath = "FastenerAssembly\\SimpleComponent\\";
    double dStackThickness = _Map.getDouble(strMapPath +
"ComponentData\\dStackThickness");
    double dSinkDiameter= _Map.getDouble(strMapPath +"ComponentData\ 
dSinkDiameter");
    double dMainDiameter= _Map.getDouble(strMapPath +"ComponentData\ 
dMainDiameter");
    double dFastenerLength= _Map.getDouble(strMapPath +"ArticleData\ 
dFastenerLength");

    reportNotice("\n" + scriptName() + " " + dMainDiameter);

    // in case the tsl is not fired from the bOnDrawFastenerComponent it
    // is probably added to the db.
    if (_ThisInst.handle()!="") { // is db resident
        PropDouble pStackThickness(0, U(10), "StackThickness");
        dStackThickness = pStackThickness;
        PropDouble pSinkDiameter(1, U(32), "SinkDiameter");
        dSinkDiameter= pSinkDiameter;
        PropDouble pMainDiameter(2, U(12), "MainDiameter");
        dMainDiameter= pMainDiameter;
        PropDouble pFastenerLength(3, U(80), "FastenerLength");
        dFastenerLength= pFastenerLength;
    }

    // entity coordinate system
    CoordSys cs(_Pt0, _XE, _YE, _ZE);

    // define a hexagon in 0 XY plane, with dimension dSinkDiameter
    PLine plHex;
    for (int i = 0; i < 6; i++) {
        double dRadius, dAngle;
        dRadius = 0.5*dSinkDiameter;
        dAngle = i*60;
        Point3d pt (dRadius*cos(dAngle), dRadius*sin(dAngle), 0);
        plHex.addVertex(pt);
    }
}

```

```

        }

    plHex.close();

    // the main body
    Body bdMain(Point3d(0,0,0), Point3d(0,0,U(2)),
0.5*dSinkDiameter);
    bdMain.addPart(Body(Point3d(0,0,U(2)),
Point3d(0,0,dStackThickness), 0.5*dSinkDiameter,0.1*dSinkDiameter));

    // square nut height
    double dF = U(3.5);
    if (dMainDiameter==U(6)) {dF=U(4);}
    else if (dMainDiameter==U(8)) {dF=U(5);}
    else if (dMainDiameter==U(10)) {dF=U(6);}
    else if (dMainDiameter==U(12)) {dF=U(8);}
    else if (dMainDiameter==U(16)) {dF=U(12);}
    else if (dMainDiameter>U(16)) {dF=U(12);}

    // add square
    bdMain.addPart
(Body
(Point3d
(
0
,0,0),cs.vecX(),cs.vecY(),cs.vecZ(),dMainDiameter,dMainDiameter,dF));
    bdMain.transformBy(cs);

    // inner radius of the hex
    double dInnerRadius = (dSinkDiameter/2*sqrt(3))/2;

    // apply cuts to the nut
    for (int i = 0; i < 12; i++){
        double dAngle = i*30;

        Point3d pt (dInnerRadius *cos(dAngle), dInnerRadius
*sin(dAngle),0);
        pt.transformBy(cs);
        //pt.vis(5);

        Vector3d vecC(cos(dAngle), sin(dAngle),1);
        vecC.transformBy(cs);
        //vecC.vis(pt);
        Cut ct(pt+cs.vecZ()*dStackThickness,vecC);
        //bdMain.addTool(ct);
    }

    // body of bolt
    if (dFastenerLength>dEps) {
        Body bdCyl(Point3d(0,0,0), Point3d(0,0,-dFastenerLength),
0.5*dMainDiameter);
        bdCyl.transformBy(cs);
        //dp.draw(bdCyl);
        bdMain.addPart(bdCyl);
    }
}

```

```

        }

Display dp(-1);
dp.draw(bdMain);
dp.draw(plHex);

if (_ThisInst.handle() != "") // is db resident
{
// draw solid in side view
CoordSys csAlign;
csAlign.setToAlignCoordSys(_Pt0,_XE, _YE, _ZE, _Pt0-
_XE*1.5*dSinkDiameter,-_ZE, _YE, _XE) ;
bdMain.transformBy(csAlign);
dp.draw(bdMain);

// draw scriptName
dp.textHeight(U(20));
dp.draw(scriptName(),_Pt0 + _XE*dSinkDiameter, _XE, _YE,1,0);
}

if (dFastenerLength>dEps)
{
    Display dpS(2);
    dpS.showInDispComp(_kSimpleModel);
    Body bd2(Point3d(0,0,0), Point3d(0,0,-dFastenerLength),
0.2*dMainDiameter);
    bd2.transformBy(cs);
    dpS.draw(bd2);
}

if (dFastenerLength>dEps)
{
    Display dpS(-1);
    dpS.showInDispComp(_kBespoke);
    Body bd2(Point3d(0,0,0), Point3d(0,0,-0.5 * dFastenerLength),
0.3*dMainDiameter);
    bd2.transformBy(cs);
    dpS.draw(bd2);
}

if (dFastenerLength>dEps)
{
    Display dpS(-1);
    dpS.showInDispComp(_kThread);
    PLine plL(Point3d(0, 0, 0), Point3d(0, 0 ,-
dFastenerLength));
    plL.transformBy(cs);
    dpS.draw(plL);
}
—end example]

```

**Part**

**VI**

## 6 Objects

### 6.1 AcadDatabase

The AcadDatabase wraps the internal autocad database pointer.

There is one special AcadDatabase, which is the current one:

**\_kCurrentDatabase:**     Return the current AcadDatabase.

A valid database can be used in [Group](#) queries.

---

```
class AcadDatabase // added hsbcad v21.0.66
{
    AcadDatabase();
    AcadDatabase(String strFullDwgName); // find the open database with the matching dwgName,
                                         (remember to use double backslash in literal filenames)

    String dwgName() const;

    static String[] allDwgNames();
    static AcadDatabase[] allDatabases();

    int blsValid() const; // return TRUE if the database exists and is still open

    AcadDatabase openFromFile(String strFullDwgName); // open a database from file. A file can
                                                       be simultaneously open more than once.
    AcadDatabase openNew(); // open a new database

    void close(); // Beware! You should only close databases that you have opened. In any case the
                  current database, or xref databases cannot be closed.

    AcadDatabase[] xrefDatabases() const; // list of xref databases attached to this database
                                         (added V23.5.9)

    Map getDrawingPropertiesMap() const; // all drawing properties are added to the Map
                                         which is returned. (added 26.4.5).
    int setDrawingPropertiesFromMap(Map mapWithValuesToBeChanged); // properties found in
                                                               the map are set. If a custom property is present, all other custom properties
                                                               are removed. (added 26.4.5)

};
```

---

[Sample of illustrating AcadDatabase:

```
PropString pDwgName(0, "c:\\aa.dwg", "full path of dwg to work with");

String strActionOpen = T("|openFromFile|");
```

---

```

addRecalcTrigger(_kContext, strActionOpen);
if (_bOnRecalc && _kExecuteKey==strActionOpen)
{
    String strDwg = pDwgName;
    AcadDatabase db;
    int bOpened = db.openFromFile(strDwg);
    reportMessage("\n dwg: "+ strDwg + (bOpened ? " opened so" : " not opened so") + (db.bIsValid() ? " is valid." : " is not valid."));
}

String strActionClose = T("|close|");
addRecalcTrigger(_kContext, strActionClose);
if (_bOnRecalc && _kExecuteKey==strActionClose)
{
    String strDwg = pDwgName;
    AcadDatabase dbE(strDwg);
    int bClosed = dbE.close();
    reportMessage("\n dwg: "+ strDwg + (bClosed ? " closed so" : " not closed so") + (dbE.bIsValid() ? " is valid." : " is not valid."));
}

String strAction2 = T("|open close new|");
addRecalcTrigger(_kContext, strAction2);
if (_bOnRecalc && _kExecuteKey==strAction2)
{
    String strDwg = "new";
    AcadDatabase db;
    int bOpened = db.openNew();
    if (bOpened)
    {
        int bClosed = db.close();
        reportMessage("\n dwg: "+ strDwg + (bClosed ? " closed so" : " not closed so") + (db.bIsValid() ? " is valid." : " is not valid."));
    }
}

String strAction3 = T("|list groups|");
addRecalcTrigger(_kContext, strAction3);
if (_bOnRecalc && _kExecuteKey==strAction3)
{
    AcadDatabase db(pDwgName);
    reportNotice("\n===== ");

    Group grHouses[] = Group(db, "").subGroups(FALSE); // if called
from Group() with arg FALSE, returns all house level groups
    for (int h=0; h<grHouses.length(); h++) {
        reportNotice("\n" + grHouses[h].name());
    }

    Group grFloors[] = grHouses[h].subGroups(FALSE); // returns
all floors of this level
    for (int f=0; f<grFloors.length(); f++) {
}
}

```

```

        reportNotice("\n      "+grFloors[f].name() );
        reportNotice(" * "+grFloors[f].dFloorHeight());

    Group grElem[] = grFloors[f].subGroups(FALSE); // returns
all third level groups of this level
    for (int e=0; e<grElem.length(); e++) {
        reportNotice("\n      "+grElem[e].namePart(2) );
        if (grElem[e].elementLinked().bIsValid()) {
            // it is an element
            reportNotice(" *** ");
        }
    }
}

String arDwgNames[] = AcadDatabase().allDwgNames();
reportMessage("\n\nNumber of dwg's: "+ arDwgNames.length());

AcadDatabase arDwgs[] = AcadDatabase().allDatabases();
for(int d=0; d<arDwgs.length(); d++)
{
    reportMessage("\nDwg: "+ arDwgs[d].dwgName());
}

reportMessage("\ncurrent dwg: "+ _kCurrentDatabase +
(_kCurrentDatabase.bIsValid() ? " is valid." : " is not valid"));

AcadDatabase dbInvestigate(pDwgName);
reportMessage("\ndwg: "+ dbInvestigate + (dbInvestigate.bIsValid() ?
" is valid." : " is not valid"));
reportMessage("\n");

```

—end sample]

[Sample of illustrating AcadDatabase, get all entities from all databases, including all block ref paths:

```

// get all databases
AcadDatabase arDwgs[] = AcadDatabase().allDatabases();

String strActionListEntities = T("|list entities|");
addRecalcTrigger(_kContext, strActionListEntities);
if (_bOnRecalc && _kExecuteKey == strActionListEntities)
{
    for (int d = 0; d < arDwgs.length(); d++)
    {
        AcadDatabase db = arDwgs[d];
        reportNotice("\nDatabase name: " + db.dwgName());
    }
}

```

```

Group grAll = Group(db, "");
int bAlsoInSubGroups = TRUE;

// get all entities from a database
Entity arEnt[] = grAll.collectEntities( bAlsoInSubGroups,
Entity(), _kModelSpace);
reportNotice("\\n\\t" + arEnt.length() + " entities found in
database.");

// get all elements from a database
Entity arElem[] = grAll.collectEntities( bAlsoInSubGroups,
Element(), _kModelSpace);
reportNotice("\\n\\t" + arElem.length() + " elements found in
database.");

// loop over all the elements
if (arElem.length() != 0)
{
    reportNotice("\\n\\t" + "Element numbers:");
    for (int e = 0; e < arElem.length(); e++)
    {
        // build all available blockref paths for each entity
        int bLookAtXrefsOnUnfrozenLayersOnly = FALSE;
        Entity entBrp[] =
arElem[e].allBlockRefPaths(bLookAtXrefsOnUnfrozenLayersOnly);

        // report the element number, and location
        for (int b = 0; b < entBrp.length(); b++)
        {
            Element el = (Element)entBrp[b];
            if ( ! el.bIsValid())
                continue;

            Point3d ptLoc = el.coordSys().ptOrg();
            reportNotice("\\n\\t\\t" + el.number() + "(" + ptLoc
+ ")");
        }
    }
}

String strActionExportCadModel = T("|export cad model|");
addRecalcTrigger(_kContext, strActionExportCadModel);
if (_bOnRecalc && _kExecuteKey==strActionExportCadModel)
{
    reportNotice("\\n\\n"+Executing" + _kExecuteKey);
    Entity entsToExport[0];

    for (int d = 0; d < arDwgs.length(); d++)
{

```

```

AcadDatabase db = arDwgs[d];
Group grAll = Group(db, "");
Entity arElem[] = grAll.collectEntities( TRUE, Element() ,
_kModelSpace);

reportNotice("\\n\\t" + arElem.length() + " elements found in
database.");
for (int e = 0; e < arElem.length(); e++)
{
    Entity entBrp[] = arElem[e].allBlockRefPaths();
    entsToExport.append(entBrp);
}
reportNotice("\\n"+entsToExport.length()+" entities exported.");

// set some export flags
ModelXComposeSettings mmFlags;
mmFlags.addSolidInfo(TRUE); // default FALSE
mmFlags.addAnalysedToolInfo(TRUE); // default FALSE
mmFlags.addElemToolInfo(TRUE); // default FALSE
mmFlags.addConstructionToolInfo(TRUE); // default FALSE
mmFlags.addHardwareInfo(TRUE); // default FALSE
mmFlags.addRoofplanesAboveWallsAndRoofSectionsForRoofs(TRUE); // 
default FALSE
mmFlags.addCollectionDefinitions(TRUE); // default FALSE

// compose ModelX
ModelX mm;
mm.setEntities(entsToExport);
mm.dbComposeMap(mmFlags);

//mm.map().showAdd();
String strFileName = _kPathDwg + "\\mm.dxx";
mm.writeToDxxFile(strFileName);
reportNotice("\\n"+"File written: " + strFileName);

String strViewerExe = _kPathHsbInstall + "\\Utilities\
\hsbMapExplorer\\hsbMapExplorer.exe";
spawn("", strViewerExe, "\"" + strFileName + "\" , "" );
}

```

*—end sample]*

[Sample of illustrating AcadDatabase drawing properties access:

```

if (_bOnInsert)
{
    _Pt0 = getPoint();
    return;
}

PropString pDimStyle(0,_DimStyles ,T("|Dim style|"));
PropDouble pTextHeight(0, U(20), T("|Text height|"));

```

```

void reportMap(String strTopic, Map& mp, String& strLines[])
{
    strLines.append(strTopic);
    for (int i=0; i<mp.length(); i++)
    {
        String strKey = mp.keyAt(i);
        if (mp.hasInt(i))
            strLines.append(i+".I" " "+strKey+": "+mp.getInt(i));
        else if (mp.hasDouble(i))
            strLines.append(i+".D" " "+strKey+": "+mp.getDouble(i));

        else if (mp.hasString(i))
            strLines.append(i+".S" " "+strKey+": "+mp.getString(i));
    }
}

String strChangeSummary = T("|Change summary 'Subject'|");
addRecalcTrigger(_kContext, strChangeSummary);
if (_bOnRecalc && _kExecuteKey==strChangeSummary)
{
    String strNew = getString(T("|Enter new value|"));
    Map mpSub;
    mpSub.setString("Subject", strNew);
    Map mpToSet;
    mpToSet.appendMap("Summary", mpSub);

    int bOk = _kCurrentDatabase.setDrawingPropertiesFromMap(mpToSet);
    reportMessage("\nProperty saved: " + bOk);
}

String strChangeCustom = T("|Remove all custom drawing properties and
add 'MyProp'|");
addRecalcTrigger(_kContext, strChangeCustom);
if (_bOnRecalc && _kExecuteKey==strChangeCustom)
{
    String strNew = getString(T("|Enter new value|"));
    Map mpSub;
    mpSub.setString("MyProp", strNew);
    Map mpToSet;
    mpToSet.appendMap("Custom", mpSub);

    int bOk = _kCurrentDatabase.setDrawingPropertiesFromMap(mpToSet);
    reportMessage("\nProperty saved: " + bOk);
}

Map mapProp = _kCurrentDatabase.getDrawingPropertiesMap();

String strLines[0];
strLines.append(scriptName());
Map mp = mapProp.getMap("Summary");

```

```

reportMap ("Summary", mp, strLines);
mp = mapProp.getMap ("Custom");
reportMap ("Custom", mp, strLines);

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end sample]*

## 6.2 Block

Block definitions can be inserted into the drawing. Block drawings can be visualized by the script. So the Tsl entity will function as a block reference. To do that, the block needs to be defined.

```

Block name(String strBlockName); // bAutoLoad default TRUE, but recommended FALSE.
Block name(String strBlockName, int bAutoLoad);

```

The strBlockName can be the name of the block in the drawing, or the full path name of the block to be used. In case of the full pathname, the block name is taken to be the filename, without the extension. The default extension used is ".dwg".

When a Block-drawing is used by the script, the script will first look in the table of loaded blocks to find a matching block name.

If no such block was found, and bAutoLoad is TRUE (default), the file system is checked if there exists a block with the specified name. The file system is checked at the last used block load-path. If the script execution finds that block drawing, it is inserted into the drawing automatically.

It is also possible to specify a specific block file location path in the name of the block. Remember to escape the '\' character: substitute the each '\' with '\\' in the filename.

If bAutoLoad is TRUE (default), and when the block drawing was not found on the file system, a file dialog is automatically shown to load the block drawing inside the current drawing. This will happen on any method of the Block. So it is recommended to use FALSE as bAutoLoad.

If bAutoLoad is FALSE, the Tsl author is in control if and when a block is loaded.

The predefined **\_BlockNames** is declared as String array, and contains all the blocks currently in the drawing (excluding the xref blocks). The array does not change during execution of the Tsl. The method getAllEntryNames does the same, but in a dynamic way, always returning the actual situation.

Possible valid definitions are:

```

Block blk1("box", FALSE);
Block blk2 = blk1;
Block blk3(blk1, FALSE);
Block blk4("c:\\Hsb-Company\\Block\\box");
Block blk5("c:\\Hsb-Company\\Block\\box.dwg");

```

To show a block, one should use the draw methods of a [Display](#):

```
Display::draw(Block block, Point3d ptLocation);
Display::draw(Block block, Point3d ptLocation, Vector3d vecX, Vector3d vecY, Vector3d
vecZ); // Block
```

For debugging purpose one can visualize a block. To visualize an insertion point is needed. One can also specify the coordinate system to which the block needs to be transformed. In that case the lengths of the vectors define the scaling factors. If the coordinate system is not specified, the UCS (at time of insertion) is taken.

---

```
class Block
{
    Block(String strBlockName); // blocks are not resolved in XRefs (as it used to be prior to
    16.3.17)
    Block(String strBlockName, int bAutoLoad); // (added 23.8.11) bAutoLoad default TRUE,
    suggested FALSE

    void visualize(Point3d ptLocation, [ int nColorIndex ]) const;
    void visualize(Point3d ptLocation, Vector3d vecX, Vector3d vecY, Vector3d vecZ, [ int
    nColorIndex ]) const;

    LineSeg getExtents() const; // returns a line segment that spans the space diagonal of
    the box enclosing the block definition (added 17.2.24 and 18.1.28)
    LineSeg getExtents(Vector3d vecX, Vector3d vecY) const; // returns a line segment that
    spans the space diagonal of the box enclosing the block definition, with box
    axis directions vecX, vecY, and vecX.crossProduct(vecY) (added 23.8.49 and
    24.1.30)

    // (added 17.2.32 and 18.1.37)
    TslInst[] tsInst() const; // return list of tsI's that belong to the block definition
    GenBeam[] genBeam() const;
    Beam[] beam() const;
    Sheet[] sheet() const;
    Sip[] sip() const;
    Entity[] entity() const;

    int blsValid() const; // (added 23.8.11) returns TRUE if the Block can be resolved in the
    drawing.
    String entryName() const; // (added 23.8.11), the resolved name of the block, without the
    path, and without invalid characters.
    String originalName() const; // (added 23.8.11), the path name stored in the block
    definition
    String[] getAllEntryNames() const; // (added 23.8.11) dynamic alternative for predefined
    _BlockNames
    int dbCreate(); // (added 23.8.11) creates a new block definition with the name of the
    block. Will overwrite existing block. Return error code, so 0 if success, and
    >0 for error.
```

```

int dbErase(); // (added 23.8.11) erase the block definition with the name of the block.
    Return error code, so 0 if success, and >0 for error.
int dbRename(String strNewBlockName); // (added 23.8.11) renames the existing block.
    Return error code, so 0 if success, and >0 for error.
Entity addEntity(Entity ent); // (added 23.8.11) adds a copy of the entity from model to the
    block definition, and returns the cloned entity.
Entity addEntity(Entity ent, CoordSys csRef); // (added 23.8.11) csRef is the location of the
    block its coordinate system in model space.

int dbCreateFromDxf(String strFileName); // (added 17.2.32 and 18.1.37) return 0 if all fine.
    Make sure the block name was set.
int dbCreateFromDxf(String strFileName, int bOverwrite); // (added 23.8.11) bOverwrite
    default TRUE

int dbCreateFromDwg(String strFileName); // (added 23.8.11) return 0 if all fine. Make sure
    the block name was set.
int dbCreateFromDwg(String strFileName, int bOverwrite); // (added 23.8.11) bOverwrite
    default TRUE

int writeToDwg(String strFileName); // (added 23.8.13) Return error code, so 0 if success,
    and >0 for error.
static int writeEntitySetIntoDwg(String strFileName, Entity[] arEnts, Point3d ptOrigin); //  

    added 23.5.9, similar to the wblock command, see example below, and as  

    wblock command, dependent on current ucs
};

```

---

[Example of O type illustrating the getExtents.

```

Block blck("aa", FALSE);
//blck.vis(_Pt0);

Display dp(-1);
dp.draw(blck,_Pt0);

LineSeg seg = blck.getExtents();
seg.transformBy(Vector3d(_Pt0)); // transform to the location where
the block is visualized
dp.draw(seg);

—end example]

```

[Example of O type illustrating the dbCreateFromDxf.

```

PropString pBlockName(1,"",T("|block name|"));
PropString pDxfName(2,"",T("|dxf file name|"));

if (_bOnInsert) {

```

```

PropString pDefBlockNames(0, BlockNames, T("|block names
present|"));
showDialog();

Pt0 = getPoint(T("|Select insert location|"));

String strBlock = pBlockName;
if (BlockNames.find(strBlock)<0) { // block not present yet, load
from dxf
    Block blkNew(strBlock, FALSE);
    int nErr = blkNew.dbCreateFromDxf(pDxfName);

    String strErr;
    if (nErr==1) strErr = T("|No name set|");
    else if (nErr==2) strErr = T("|Not possible to read dxf|");
    else if (nErr==3) strErr = T("|Not possible to create block|");
    else if (nErr==0) strErr = T("|Dbx file read fine and block
created|");
    reportMessage("\n"+strErr);
}
return;
}

// redefine propstring such that newly created block is in the list
PropString pDefBlockNames(0, BlockNames, T("|block names present|"));

String strBlock = pBlockName;
Block blck(strBlock, FALSE);
//blck.vis(_Pt0);

Display dp(-1);
dp.draw(blck, Pt0);

// look into block definition
Entity arEnt[] = blck.entity();
reportMessage("\nNumber of entities returned: "+arEnt.length());

for (int i=0; i<arEnt.length(); i++)
{
    Entity ent = arEnt[i];
    PLine pl = ent.getPLine();
    reportMessage("\n"+i+": is a "+ent.typeName() + ", len:
"+pl.length());
    if (ent.typeName() == "AcDbLine")
        dp.color(3);
    else
        dp.color(1);

    pl.transformBy(Vector3d(Pt0)); // apply the same transformation
    as drawing the block
    dp.draw(pl);
}

```

—end example]

[Example of O type illustrating the writeEntitySetIntoDwg.

```

if (_bOnInsert)
{
    PrEntity ssE(T("Select a set of entities"), Entity());
    if (ssE.go()) {
        _Entity = ssE.set();
    }
    _Pt0 = getPoint();
    return;
}

Entity ents[0];
ents = _Entity;

String strCreateFile = T("|create file from selected entities");
addRecalcTrigger(_kContext, strCreateFile );
if (_bOnRecalc && _kExecuteKey==strCreateFile )
{
    reportMessage("\nNumber of entities found: "+ents.length());
    int errCode = Block().writeEntitySetIntoDwg("c:\\temp\\
\\bb\\ablock.dwg", ents, _PtW);
    if (errCode != 0)
        reportMessage("\nError code: "+errCode);
}

String strFromBlockRef = T("|create file from first blockref
content");
addRecalcTrigger(_kContext, strFromBlockRef );
if (_bOnRecalc && _kExecuteKey==strFromBlockRef )
{
    Entity entsBl[0];
    for (int i = 0; i < ents.length(); i++)
    {
        BlockRef br = (BlockRef)ents[i];
        if ( ! br.bIsValid())
            continue;

        Block block(br.definition(), FALSE);
        entsBl.append(block.entity());
        break;
    }
    reportMessage("\nNumber of entities found: "+entsBl.length());
    int errCode = Block().writeEntitySetIntoDwg("c:\\temp\\
\\bb\\ablockFromBlock.dwg", entsBl, _PtW);
    if (errCode != 0)
        reportMessage("\nError code: "+errCode);
}

```

—end example]

[Example of O type illustrating Block creation and addEntity

```

if (_bOnInsert)
{
    _Pt0 = getPoint();
    return;
}

String strCreateNewEmptyBlock = T("|Create empty block|");
addRecalcTrigger(_kContext, strCreateNewEmptyBlock );
if (_bOnRecalc && _kExecuteKey==strCreateNewEmptyBlock)
{
    String strNewName = getString(T("\\n|Enter new block name|"));
    Block blockNew(strNewName, FALSE);
    int nErr = blockNew.dbCreate();
    reportMessage("\\ndbCreate error code: " + nErr + " and bIsValid:" +
+ blockNew.bIsValid());
}

String strEraseExisting = T("|Erase existing block|");
addRecalcTrigger(_kContext, strEraseExisting );
if (_bOnRecalc && _kExecuteKey==strEraseExisting)
{
    String strBlockName = getString(T("\\n|Enter existing block
name|"));
    Block block(strBlockName, FALSE);
    int bWasValid = block.bIsValid();
    int nErr = block.dbErase();
    reportMessage("\\ndbErase error code: " + nErr + " was bIsValid:" +
+ bWasValid+ " and bIsValid:" + block.bIsValid());
}

String strAutoReadNewBlock = T("|read dwg block and prompt user if
missing|");
addRecalcTrigger(_kContext, strAutoReadNewBlock );
if (_bOnRecalc && _kExecuteKey==strAutoReadNewBlock)
{
    String strNewName = getString(T("\\n|Enter block name to load|"));
    String strFilename = _kPathDwg + "\\\" + strNewName;
    Block block(strFilename, TRUE); // TRUE will prompt user on
bIsValid
    // autoload TRUE will prompt user on bIsValid
    reportMessage("\\nbIsValid:" + block.bIsValid());
}

String strReadNewBlock = T("|read dwg block if missing|");
addRecalcTrigger(_kContext, strReadNewBlock );
if (_bOnRecalc && _kExecuteKey==strReadNewBlock)
{
    String strNewName = getString(T("\\n|Enter block name|"));
    Block block(strNewName, FALSE);
}

```

```

    if ( ! block.bIsValid() )
    {
        String strFilename = _kPathDwg + "\\" + strNewName + ".dwg";
        int nErr = block.dbCreateFromDwg(strFilename, TRUE);
        reportMessage("\ndbCreate error code: " + nErr + " and
bIsValid:" + block.bIsValid());
    }
}

String strReadDxfBlock = T("|read dxf block if missing|");
addRecalcTrigger(_kContext, strReadDxfBlock );
if (_bOnRecalc && _kExecuteKey==strReadDxfBlock)
{
    String strNewName = getString(T("\n|Enter block name|"));
    Block block(strNewName, FALSE);
    if ( ! block.bIsValid())
    {
        String strFilename = _kPathDwg + "\\" + strNewName + ".dxf";
        int nErr = block.dbCreateFromDxf(strFilename, TRUE);
        reportMessage("\ndbCreate error code: " + nErr + " and
bIsValid:" + block.bIsValid());
    }
}

String blockNames[] = Block().getAllEntryNames();
blockNames.append("none existant block");

PropString blockName(0, blockNames, T("|Block name|"));
Block blck(blockName, FALSE);

String strAddEntity = T("|add entity to block|");
addRecalcTrigger(_kContext, strAddEntity );
if (_bOnRecalc && _kExecuteKey==strAddEntity)
{
    CoordSys csRel(_Pt0, _XE, _YE, _ZE);
    if (blck.bIsValid())
    {
        Entity ent = getEntity(T("|Select entity to add to block|"),
TRUE);
        Entity entClone = blck.addEntity(ent, csRel);
        if (entClone.bIsValid())
            entClone.setColor(1);
    }
}

String strRemoveEntity = T("|remove last entity of block|");
addRecalcTrigger(_kContext, strRemoveEntity );
if (_bOnRecalc && _kExecuteKey==strRemoveEntity)
{
    if (blck.bIsValid())
    {
        Entity ents[] = blck.entity();
        if (ents.length() > 0)

```

```

        ents[ents.length() - 1].dbErase();
    }

}

String strRename = T("|rename block|");
addRecalcTrigger(_kContext, strRename );
if (_bOnRecalc && _kExecuteKey==strRename)
{
    if (blk.bIsValid())
    {
        blk.dbRename(getString("\nNew block name: "));
    }
}

String strWriteBlock = T("|write block|");
addRecalcTrigger(_kContext, strWriteBlock );
if (_bOnRecalc && _kExecuteKey==strWriteBlock)
{
    if (blk.bIsValid())
    {
        String strNewName = getString("\nNew block name: ");
        String strFilename = _kPathDwg + "\\" + strNewName + ".dwg";
        int nErr = blk.writeToDwg(strFilename);
        reportMessage("\nwriteToDwg error code: " + nErr + " and
bIsValid:" + blk.bIsValid());

        Block blockNew(strNewName, FALSE);
        nErr = blockNew.dbCreateFromDwg(strFilename, TRUE);
        reportMessage("\ndbCreate error code: " + nErr + " and
bIsValid:" + blockNew.bIsValid());
    }
}

blk.vis(_Pt0+U(100)*(_XW+_YW+_ZW), 2);

reportMessage("\nOriginalName: " + blk.originalName());
reportMessage("\nentryName: " + blk.entryName());
reportMessage("\nEntity count: " + blk.entity().length());

Display dp(-1);
dp.draw(blk, _Pt0);

LineSeg seg = blk.getExtents();
seg.transformBy(Vector3d(_Pt0));
dp.draw(seg);

```

*—end example]*

## 6.3 Group

The Group class is a reference to the hsb-group, exposed in the hsb-console. A group reference is defined by a string consisting of 3 parts, the top level name (level 0), the middle level name (level 1), and the last level name (level 2). Changing the value of any of these parts, will change the reference. This means the Group will refer to another hsb-group in the hsb-console.

There is one special group, which is made with the default constructor, or by specifying an empty name. This group refers to the complete drawing, or to everything that does not belong to a group.

Collecting entities from a group will work for most TSL entity types, but not all. The collectEntities will NOT work for none Aec entities even though Entity type is allowed to be specified in the call. If Entity is the type requested, all Aec entities are collected in the drawing. For hsbCAD versions prior to 18.1.63, Entity was not accepted as argument.

To select the space (means block table record in Autocad database) to collect the entities from, one of the following values should be used

- \_kAllSpaces:** 0, includes model, all paper spaces and blocks
- \_kModelSpace:** 1
- \_kMySpace:** 2, means the model or paper space that the TSL entity was added to.  
During insert, the MySpace is not known yet, so the current active space is taken.

The groupVisibility is controlled by layers being on or off. A layer refers to the group if it contains a group key in its layer name. But not all those layers contribute to the visibility state. Only the layers that matter. Eg if zone 1 is turned off in the console settings, the zone 1 layers will not contribute to the visibility value. The groupVisibility method returns one of the following:

- \_kIsOn:** 1, all layers of the group that matter are on
- \_kIsOff:** 0, all layers of the group that matter are off
- \_kIsGray:** -1, some layers that matter are on, and some layers are off
- \_kIsNotDefined:** -2, there are no layers that matter

During insert, one can use the current group set int the hsbConsole:

**\_kCurrentGroup:** Is set to point to the current group, when this group is set in the console.

A group name cannot contain any of the following characters: \/:\*?"<>| ; , ='

Prior to version 16, this set of forbidden characters was: \/:\*?"<>|

---

```
class Group {
    Group();
    Group(String strCompositeName); // name eg "house\\floor\\gr", remember to use double
                                    // backslash
    Group(String strlevel0, String strlevel1, String strlevel2);

    Group(AcadDatabase db, String strCompositeName); // AcadDatabase added v21.0.66, get
                                                    // group from other loaded dwg

    String name() const;
```

---

```

void setName(String strComp);

String namePart(int nLevel) const; // nLevel is 0 based, possible values 0, 1, 2
void setNamePart(int nLevel, String str);

static Group[] allExistingGroups() const;
static Group[] allElementGroups() const;

Group[] subGroups(int bFullTree) const;

Element elementLinked() const;

double dFloorHeight() const;
int setDFloorHeight(double dValue); // works only on second level groups. Returns success of
the operation.

CoordSys getUcs() const; // (added V23.4.23) return the ucs associated with the group
void setUcs(CoordSys csUcs); // (added V23.4.23)

int bExists() const;

Entity[] collectEntities(int bAlsoInSubGroups, Entity type, int nSpace) const; //
bOnlyAecEntities default to TRUE
Entity[] collectEntities(int bAlsoInSubGroups, Entity type, int nSpace, int bOnlyAecEntities)
const; // (added hsbCAD v20.3.47 and v21.0.64)

void addEntity(Entity ent); // add the Entity to the group
void addEntity(Entity ent, int bExclusive);
void addEntity(Entity ent, int bExclusive, int nZoneIndex, char cZoneCharacter);

void dbCreate();
int dbRename(Group groupNew); // return success of the operation
int dbErase(); // return success of the operation

Grid grid() const; // returns a Grid

int blsDeliverableContainer() const; // (added hsbCAD13.2.125)
int setBlsDeliverableContainer(int bSet); // works only on second level groups. Returns success
of the operation.

int nEquivalentStory() const; // (added hsbCAD16.0.25)
int setNEquivalentStory(int nSet); // works only on second level groups. Returns success of the
operation.

void setFromHandle(String strHandle); // translates the string handle into a reference.
(added 22.0.26 and 21.1.25)

void turnGroupVisibilityOn(int bAlsoSubGroups); // added V23.4.23 (Viewport has similar
methods)
void turnGroupVisibilityOff(int bAlsoSubGroups); // added V23.4.23
int groupVisibility(int bAlsoSubGroups) const; // added V23.4.23, returns \_kIsOn, \_kIsOff,
\_kIsGray or \_kIsNotDefined

```

```
};
```

---

```
Group();
Group(String strCompositeName);
Group(String strlevel0, String strlevel1, String strlevel2);
```

A group reference is defined by a string consisting of 3 parts, the top level name (level 0), the middle level name (level 1), and the last level name (level 2). Changing the value of any of these parts, will change the reference. This means the Group will refer to another hsb-group in the hsb-console. The Group instance in TSL can be constructed with either a composite name, eg. Group("house\\floor\\aa"), or 3 separate parts Group("house", "floor", "aa"). In case the composite name is specified, the different parts need to be separated by double backslashes.

```
Group[] subGroups(int bFullTree) const;
```

The subGroups routine allows to query the group tree structure information. By using the subGroups on the Group with an empty name, one can retrieve the list of the level 0 groups. The argument bFullTree, specifies if the reported subgroups should go one level deep, or all the levels deep. See example below.

```
Entity[] collectEntities(int bAlsoInSubGroups, Entity type, int nSpace) const; // bOnlyAecEntities default to TRUE
Entity[] collectEntities(int bAlsoInSubGroups, Entity type, int nSpace, int bOnlyAecEntities) const; // (added hsbCAD v20.3.47 and v21.0.64)
```

The collectEntities is the routine to get the entities that belong to a group. One has to specify the space: \_kAllSpaces, \_kModelSpace, or the space of the Tsl instance \_kMySpace. The parameter bAlsoInSubGroups allows to browse the subgroups. Even if the entity belongs to multiple groups specified, it will only appear once in the returned list. Collecting entities from a group will work for most TSL entity types, but not all (unless bOnlyAecEntities is set to FALSE). If bOnlyAecEntities is TRUE (which is the default), the collectEntities will NOT work for none Aec entities even though Entity type is allowed to be specified in the call. If Entity() is the type requested, all Aec entities are collected in the drawing. For hsbCAD versions prior to 18.1.63, Entity() was not accepted as argument.  
If bOnlyAecEntities is FALSE, all entities are considered, but the call is less performant. Do not use if you are looking for only Aec entities.

```
void addEntity(Entity ent);
void addEntity(Entity ent, int bExclusive);
void addEntity(Entity ent, int bExclusive, int nZoneIndex, char cZoneCharacter);
```

Adds the entity to the group. If the groups does not exist yet, the group is created automatically.  
bExclusive: an entity can belong to multiple groups. bExclusive is FALSE (is default). In some cases you want to specify if the instance belongs to only one group.  
Although a group might not have a link with an element, the group information that is stored for the entity has a zoneindex and a zone character.  
nZoneIndex: if specified, the zone index: (-5,-4,...4,5), default 0.

---

cZoneCharacter: specifies the entity set; should be equal to 'Z' for general items, 'T' for beam tools, 'E' for element tools. The default character is '', and means the character is chosen automatically depending on the entity type.

**Grid** grid() const; // returns a [Grid](#)

Return the grid that would be considered active for this group. If the group contains a Grid, it is taken. If the group does not contain a Grid, the parent group is investigated. If the parent group does contain a Grid, that one is returned. This is repeated until a Grid is found or until no Grid is found that belongs to the Groups at hand. If that is the case, the complete database is searched for a Grid that does not belong to a Group. The first Grid found is returned.

**int** dbRename(**Group** groupNew);

The group is renamed to the new group specified. Returns success of the operation. Beware, a group cannot be renamed to an already existing database group. Also the groups internally are not stored as a hierarchical structure, but rather as a flat list of all groups as retrieved by [allExistingGroups](#). This means renaming a parent group will not rename all its children.

*[Sample of any-type with insert done in script. Illustrates the subGroups*

```
if (_bOnInsert) {
    reportNotice ("\n=====");
    Group grAll[] = Group() .subGroups (TRUE); // if called from Group()
with arg TRUE, returns all groups
    for (int a=0; a<grAll.length(); a++) {
        reportNotice ("\n" + grAll[a].name() );
    }
    reportNotice ("\n");

    Group grHouses[] = Group() .subGroups (FALSE); // if called from
Group() with arg FALSE, returns all house level groups
    for (int h=0; h<grHouses.length(); h++) {
        reportNotice ("\n" + grHouses[h].name() );

        Group grFloors[] = grHouses[h].subGroups (FALSE); // returns all
floors of this level
        for (int f=0; f<grFloors.length(); f++) {
            reportNotice ("\n      "+grFloors[f].name());
            reportNotice (" -- "+grFloors[f].dFloorHeight());
        }

        Group grElem[] = grFloors[f].subGroups (FALSE); // returns
all third level groups of this level
        for (int e=0; e<grElem.length(); e++) {
            reportNotice ("\n          "+grElem[e].namePart(2) );
        }
    }
}
```

```

        if (grElem[e].elementLinked().bIsValid()) {
            // it is an element
            reportNotice("****");
        }
    }
}

eraseInstance(); // do not add this on to the DB
return;
}

```

—end sample]

[Sample of O-type with insert done in script. Illustrates the collectEntities.

```

if (_bOnInsert) {
    _Pt0 = getPoint();
    return;
}

{
    Group gr("Blokhut\\wanden");
    int bAlsoInSubGroups = TRUE;
    Entity arEnt[] = gr.collectEntities( bAlsoInSubGroups, Beam(),
    _kModelSpace);
    reportNotice("\n"+arEnt.length()+" entities collected of
"+gr.name());
}

{
    Group gr; // default constructor, or empty groupname means
complete drawing
    int bAlsoInSubGroups = FALSE;
    Entity arEnt[] = gr.collectEntities( bAlsoInSubGroups, Beam(),
    _kModelSpace);
    reportNotice("\n"+arEnt.length()+" entities collected of complete
drawing.");
}

```

—end sample]

[Sample of O-type with insert done in script. Illustrates the addEntity.

```

if (_bOnInsert) {
    Entity ent= getEntity();
    Group gr("Blokhut\\special");
    gr.addEntity(ent, FALSE, 1, 'D');

    Group grList[] = ent.groups();
    reportNotice("\nEntity belongs now to the following groups: ");
}

```

```

        for (int g=0; g<grList.length(); g++) {
            if (g!=0) reportNotice(", ");
            reportNotice(grList[g].name());
        }

        eraseInstance();
        return;
    }

—end sample]

```

[Sample of O-type with insert done in script. Illustrates the list of groups, current group, dbCreate, and setBIsDeliverableContainer

```

if (_bOnInsert) {

    PropString sGrpNm1(0,"house");
    PropString sGrpNm2(1,"floor");
    showDialog();

    Group gr(sGrpNm1,sGrpNm2,"");
    gr.dbCreate();
    gr.setBIsDeliverableContainer(TRUE);

    // report what the group state is
    Group grGroups[] = Group().allExistingGroups();
    String strCurrent = _kCurrentGroup.name();

    for (int h=0; h<grGroups.length(); h++) {
        reportNotice("\n" + grGroups[h].name() );
        if (grGroups[h].name() == strCurrent)
            reportNotice(" cccccccccccccccccccccccc");
        if (grGroups[h].bIsDeliverableContainer())
            reportNotice(" dddddddddd");
    }

    eraseInstance(); // do not add this on to the DB
    return;
}

```

—end sample]

[Sample of O-type that illustrates the visibility and ucs access

```

Unit(1,"mm");
double dEps = U(0.1);

PropString pDimStyle(0,_DimStyles ,T("|Dim style|"));
PropDouble pTextHeight(0,U(20),T("|Text height|"));

```

```
String strCreateGroup = T("|Create group|");
addRecalcTrigger(_kContext, strCreateGroup );
if (_bOnRecalc && _kExecuteKey==strCreateGroup)
{
    Group grNew("huis","Vloer","jow");
    grNew.dbCreate();
}

Group grGroups[] = Group().allExistingGroups();
String grGroupNames[0];
for(int g=0; g<grGroups.length(); g++)
    grGroupNames.append(grGroups[g].name());

PropString pGroup(1, grGroupNames, T("|Groups|"),
grGroupNames.find(_kCurrentGroup.name(), 0));
Group grp((String)pGroup);

String strTurnVisOnSub = T("|turnGroupVisibilityOn + sub|");
addRecalcTrigger(_kContext, strTurnVisOnSub );
if (_bOnRecalc && _kExecuteKey==strTurnVisOnSub)
{
    grp.turnGroupVisibilityOn(TRUE);
}

String strTurnVisOn = T("|turnGroupVisibilityOn|");
addRecalcTrigger(_kContext, strTurnVisOn );
if (_bOnRecalc && _kExecuteKey==strTurnVisOn)
{
    grp.turnGroupVisibilityOn(FALSE);
}

String strTurnVisOffSub = T("|turnGroupVisibilityOff + sub|");
addRecalcTrigger(_kContext, strTurnVisOffSub );
if (_bOnRecalc && _kExecuteKey==strTurnVisOffSub)
{
    grp.turnGroupVisibilityOff(TRUE);
}

String strTurnVisOff = T("|turnGroupVisibilityOff|");
addRecalcTrigger(_kContext, strTurnVisOff );
if (_bOnRecalc && _kExecuteKey==strTurnVisOff)
{
    grp.turnGroupVisibilityOff(FALSE);
}

String strSetUcs = T("|setUcs|");
addRecalcTrigger(_kContext, strSetUcs );
if (_bOnRecalc && _kExecuteKey==strSetUcs)
{
    CoordSys cs(_PtU, _XU, _YU, _ZU);
    grp.setUcs(cs);
}
```

```

CoordSys cs = grp.getUcs();
Vector3d vecX = cs.vecX();
Vector3d vecY = cs.vecY();
vecX.vis(_Pt0,1);
vecY.vis(_Pt0,2);

String strLines[0];
strLines.append("selected group: "+pGroup);
strLines.append("_kCurrentGroup: "+_kCurrentGroup.name());
strLines.append("groupVisibility + sub: "+grp.groupVisibility(TRUE));
strLines.append("groupVisibility - sub:
"+grp.groupVisibility(FALSE));
strLines.append("_kIsOn: "+_kIsOn);
strLines.append("_kIsOff: "+_kIsOff);
strLines.append("_kIsGray: "+_kIsGray);
strLines.append("_kIsNotDefined: "+_kIsNotDefined);
strLines.append("ucs set: "+(vecX != vecY));

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

—end sample]

```

## 6.4 Layout

The Layout refers to the autocad layout tabs, all but Model.

---

```

class Layout { // (added hsbCAD23.1.3)

    Layout(String strEntry); // constructor

    String entryName() const;
    int blsValid() const;

    String handle() const; // return the handle (=text representation of objectId) of the object
    String typeName() const; // return the name of the object type eg: AcDbLayout

    int plotRotation(); // (added 24.1.91, 25.1.64) returns 0, 90, 180 or 270
    double paperSizeX(); // (added 24.1.91, 25.1.64)

```

---

```

double paperSizeY(); // (added 24.1.91, 25.1.64)
Point3d paperLowerLeft(); // (added 24.1.91, 25.1.64)

Map subMapX(String strKey) const; // see Map. Since V23.6.35 the strKey can also be a
path into the map, returning the submap directly eg "subMapXKey\
\subMapA\\subsubMapB".
void setSubMapX(String strKey, Map mapNew);
String[] subMapXKeys() const; // return list of available sub map keys
void removeSubMapX(String strKey);

static Layout[] getAllEntries(); // since V23.6.35 returns the sorted list based on the layout
tab index
static String[] getAllEntryNames(); // since V23.6.35 returns the sorted list based on the
layout tab index

Viewport[] viewport(); // added hsbCAD25.1.110. Returns all viewports in layout, including
the base one. Also look to Viewport for inLayoutTab method.
TslInst[] tsInst(); // added hsbCAD25.1.110. Returns a tsInst which are part of the layout.

static String currentLayout(); // added hsbCAD23.6.35.
int setCurrentLayout(); // added hsbCAD25.1.110. Return TRUE if successful. Method is only
available in command mode: on insert and on custom context commands.
static int setCurrentLayout(String strLayoutName); // added hsbCAD25.1.110. Return TRUE if
successful. Method is only available in command mode.
};

```

---

[Example O-type TSL:

```

Unit(1, "mm");

String strAllDefinitions[] = Layout().getAllEntryNames(); // list of
all available Layout
PropString pDefinition(0, strAllDefinitions, T("|Layouts|")); // make
property

if (_bOnInsert)
{
    showDialog(); // allow the user to change the definition
    _Pt0 = getPoint();
    return;
}

PropString pDimStyle(3, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

// compose a Layout from the String
Layout layout(pDefinition);
String strLines[0];

```

---

```

strLines.append("Layout");
strLines.append("entryName: "+layout.entryName());
strLines.append("handle: "+layout.handle());
strLines.append("typeName: "+layout.typeName());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*--end example]*

## 6.5 DictObject

A DictObject is an Autocad database resident object that is stored in some kind of named object dictionary. It servers as a base class to a number of objects, each of which are identified in the dictionary with a name, called the entryName. Some entries have a special status, they are phantom entries that are always accessible, but do not necesarily real database resident. An example of such a special entry is the "Rectangle" extrusion profile. It will appear in the list of entries, but is not a real object.

Tsl's can become dependent on the existence of such a DictObject, in which case they should call the global function:

`setDependencyOnDictObject(DictObject object); // see Global functions and Execution`

```
class DictObject {  
  
    String entryName() const;  
  
    int blsValid() const;  
    int blsSpecial() const;  
  
    int blsA(DictObject ent) const;  
    int blsKindOf(DictObject ent) const;  
  
    String handle() const; // return the handle (=text representation of objectId) of the object  
    String hostId() const; // equivalent to handle (=text representation of objectId) of the  
                         // object (added 25.1.57 and 24.1.88).  
    String typeName() const; // return the name of the object type eg: Hsb_DbProfile  
    AcadDatabase database() const; // (added V23.4.12 and V22.1.130) returns the  
                                 // AcadDatabase that this object belongs to  
  
    String styleDescription() const; // (added hsbCAD18.2.6) return the description of the  
                                  // DictObject
```

```

void setStyleDescription(String strDesc); // (added hsbCAD18.2.6)

Entity[] getReferencesToMe(); // return the entities that are dependent on this DictObject. It
    is essential that they have called setDependencyOnDictObject.
dbErase(); // should only be called if you are sure that getReferencesToMe returns a
    zero length array.

Map getClassificationMap() const; // (added hsbCAD16.0.21).

String[] attachedPropSetNames() const; // return list of attached property set names
    (added hsbCAD16.0.57)
Map getAttachedPropSetMap(String strPropSetName) const; // all properties found in the
    attached property set with the given name are added to the Map which is
    returned. (added hsbCAD16.0.57)
Map getAttachedPropSetMap(String strPropSetName, String[] arPropertyNames) const; // 
    only the properties specified (case insensitive match) are added to the map
    which is returned. (added hsbCAD16.0.57)

// The subMapX set of methods are available for all DictObjects. (added hsbCad17.0.45).
Map subMapX(String strKey) const; // see Map. Since V23.6.35 the strKey can also be a
    path into the map, returning the submap directly eg "subMapXKey\
    \subMapA\subsubMapB".
void setSubMapX(String strKey, Map mapNew);
String[] subMapXKeys() const; // return list of available sub map keys
void removeSubMapX(String strKey);
}

```

---

**Map** getClassificationMap() const; // (added hsbCAD16.0.21).

If the DictObject is an ACA style, it can have a classification (see example at [MvBlockDef](#)). To get the classification values, one can use this method. The [Entity](#) also has a getClassificationMap method.

```

String[] attachedPropSetNames() const; // return list of attached property set names (added
    hsbCAD16.0.57)
Map getAttachedPropSetMap(String strPropSetName) const; // all properties found in the
    attached property set with the given name are added to the Map which is
    returned. (added hsbCAD16.0.57)
Map getAttachedPropSetMap(String strPropSetName, String[] arPropertyNames) const; // 
    only the properties specified (case insensitive match) are added to the map
    which is returned. (added hsbCAD16.0.57)

```

If the DictObject is an ACA style, it can have property sets attached (see example at [MvBlockDef](#)).

## 6.5.1 CncCurveStyle

The CncCurveStyle, which is derived from the [DictObject](#), refers to a [CncCurveEnt](#) style description.

An DictObject can be casted into an CncCurveStyle. However when the casting is not allowed, the resulting CncCurveStyle will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because CncCurveStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The CncCurveStyle from a CncCurveEnt can be constructed with the string returned by  
`CncCurveEnt::style\(\)`;

---

```
class CncCurveStyle : DictObject { // see DictObject for base functions (added
    hsbCAD17.0.44)

    CncCurveStyle(String strEntry); // constructor

    static CncCurveStyle[] getAllEntries();
    static String[] getAllEntryNames();
}
```

---

[Example O-type TSL:

```
Unit(1, "mm");

if (_bOnInsert) {

    // select the Panel and insertion point
    CncCurveEnt sp = getCncCurveEnt();
    _Entity.append(sp);
    _Pt0 = getPoint();

    String strStyle = sp.style(); // get the style from the selected
    CncCurveEnt

    String strAllStyles[] = CncCurveStyle().getAllEntryNames(); // list of all available CncCurveStyles
    PropString pStyle(0, strAllStyles, "CncCurveEnt style",
    strAllStyles.find(strStyle)); // make property
    showDialog(); // allow the user to change the style
```

---

```

// change the CncCurveStyle of the CncCurveEnt, to match the
property value
sp.setStyle(pStyle);

return;
}

if (_Entity.length()==0) return;
CncCurveEnt sp = (CncCurveEnt) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strStyle = sp.style();
PropString pStyle(0, strStyle , "CncCurveEnt style"); // make
property
pStyle.set(strStyle);
pStyle.setReadOnly(TRUE);

PropString pDimStyle(1, _DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

// compose a CncCurveStyle from the String
CncCurveStyle cncCurveStyle(strStyle);

String strLines[0];
strLines.append("strStyle : "+strStyle );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO ,_XU, _YU, 1,1);
}

```

*—end example]*

## 6.5.2 CollectionDefinition

The CollectionDefinition, which is derived from the [DictObject](#), is referenced by the [CollectionEntity](#).

An DictObject can be casted into an CollectionDefinition. However when the casting is not allowed, the resulting CollectionDefinition will become invalid. This can be checked with the bIsValid() function of DictObject.

Because CollectionDefinition is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The CollectionDefinition from a CollectionEntity can be constructed with the string returned by [CollectionEntity::definition\(\)](#);

Each CollectionDefinition has an mirrorPlane. The type should be one of the following (added in hsbCAD22.1.40 and hsbCAD23.0.23)

[\\_kMPDoNotAllowMirror](#) (0), [\\_kMPUseYZPlane](#) (1), [\\_kMPUseZXPlane](#) (2), [\\_kMPUseXYPlane](#) (3)

```
class CollectionDefinition : DictObject { // see DictObject for base functions (added
hsbCAD14.0.73)

    CollectionDefinition(String strEntry); // constructor

    static CollectionDefinition[] getAllEntries();
    static String[] getAllEntryNames();

    TslInst[] tslInst() const; // return list of tsl's that belong to the collection definition
    GenBeam[] genBeam() const;
    Beam[] beam() const;
    Sheet[] sheet() const;
    Sip[] sip() const;
    Entity[] entity() const;

    // the following methods are added in hsbCAD20.0.68 and hsbCAD19.1.99
    string dbCreate(); // creates a new entry, make sure the strEntry that was set in the
                      // constructor does not exist yet. An error string is returned which is empty if
                      // all ok.
    int setContent(CoordSys cs, <Entity>[] ents, int bEraseEntities); // redefine the content of
                      // the underlying anonymous block. Return success of operation.
    int clearContent(); // clear the content of the underlying anonymous block. Return
                      // success of operation.

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    static String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from
                    // dwg. Only use on special events.
    static String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return
                    // error code as string. Empty string means all ok, see example below.
    static String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return
                    // error code as string. Empty string means all ok, see example below.

    // following methods are added in hsbCAD22.1.43 and hsbCAD23.0.23
    void transformContent(CoordSys csTrans); // transform the content of the internal block
    int mirrorPlane() const; // one of the mirrorPlane values \_kMPDoNotAllowMirror,
                           // \_kMPUseYZPlane, \_kMPUseZXPlane, \_kMPUseXYPlane
    void setMirrorPlane(int val);
    CollectionDefinition mirrorStyle() const; // returns CollectionDefinition, TrussDefinition or
                                              // MetalPartCollectionDef depending on the type
    void setMirrorStyle(CollectionDefinition val); // val should be of the appropriate type
                                              // CollectionDefinition, TrussDefinition or MetalPartCollectionDef
```

```
};

---


```

[Example O-type TSL:

```
Unit(1, "mm");
String strAllDefinitions[] =
CollectionDefinition().getAllEntryNames(); // list of all available
CollectionDefinitions

if (_bOnInsert) {

    PropString pDefinition(0, strAllDefinitions, "CollectionEntity
definition");

    showDialog(); // allow the user to change the definition

    // select insertion point
    _Pt0 = getPoint();

    if (strAllDefinitions.find(pDefinition) < 0)
    {
        // not found, create one
        String strNewName = pDefinition;
        CollectionDefinition anew(strNewName);
        String strError = anew.dbCreate();
        if (strError == "")
        {
            // all fine
            reportMessage("\nCollectionDefinition created with name:
"+anew.entryName());
            pDefinition.set(anew.entryName());
        }
        else
        {
            reportMessage("\nCould not create CollectionDefinition
with name: "+strNewName);
            reportMessage("\n"+strError );
            eraseInstance();
            return;
        }
    }

    // create one
    CoordSys csCE(_Pt0, _XU, _YU, _ZU);
    String strDefinition = pDefinition;
    CollectionEntity sp;
    sp.dbCreate(csCE, strDefinition);

    _Entity.append(sp);
```

```

        return;
    }

    if (_Entity.length()==0) return;
    CollectionEntity sp = (CollectionEntity) _Entity[0];
    if (!sp.bIsValid()) {
        eraseInstance(); // just erase from DB
        return;
    }

    String strDefinition = sp.definition();
    PropString pDefinition(0, strDefinition, "CollectionEntity
definition"); // make property
pDefinition.set(strDefinition);
pDefinition.setReadOnly(TRUE);

    int arOCT[] = {_kMPDoNotAllowMirror, _kMPUseYZPlane, _kMPUseZXPlane,
_kMPUseXYPlane};
    String arOCTStr[] = {"_kMPDoNotAllowMirror", "_kMPUseYZPlane",
"_kMPUseZXPlane", "_kMPUseXYPlane"};
    PropString pMirrorPlane(1, arOCTStr, "mirror plane");
    int nMirrorPlane = arOCT[arOCTStr.find(pMirrorPlane, 0)];

    PropString pMirrorDefinition(2, strAllDefinitions, "CollectionEntity
definition");

    PropString pDimStyle(3,_DimStyles , "Dim style");
    PropDouble pTextHeight(0,U(20), "Text height");

    // compose a CollectionDefinition from the String
    CollectionDefinition colDef(strDefinition);

    String strRedefine = T("|Redefine content of definition|");
    addRecalcTrigger(_kContext, strRedefine );
    if (_bOnRecalc && _kExecuteKey==strRedefine )
    {
        PrEntity SSE(T("|Select a set of entities|"), Entity());
        if (sse.go()) {
            Entity ents[] =sse.set();
            Point3d ptRef = getPoint();
            CoordSys cs(ptRef, _XU, _YU, _ZU);
            colDef.setContent(cs, ents, FALSE);
        }
    }

    String strRedefineErase = T("|Redefine content of definition and
erase|");
    addRecalcTrigger(_kContext, strRedefineErase );
    if (_bOnRecalc && _kExecuteKey==strRedefineErase )
    {
        PrEntity SSE(T("|Select a set of entities|"), Entity());
        if (sse.go()) {
            Entity ents[] =sse.set();

```

```

        Point3d ptRef = getPoint();
        CoordSys cs(ptRef, _XU, _YU, _ZU);
        colDef.setContent(cs, ents, TRUE);
    }

}

String strClear = T("|Clear content of definition|");
addRecalcTrigger(_kContext, strClear );
if (_bOnRecalc && _kExecuteKey==strClear )
{
    colDef.clearContent();
}

String strDbCreate = T("|Create a new definition|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    String strNewName = getString("new name:");
    CollectionDefinition anew(strNewName);
    String strError = anew.dbCreate();
    if (strError == "")
    {
        // all fine
        colDef = anew;
        sp.setDefinitionObject(anew);
        pDefinition.set(sp.definition());
        reportMessage("\nCollectionDefinition created with name:
"+colDef.entryName());
    }
    else
    {
        reportMessage("\nCould not create CollectionDefinition with
name: "+strNewName);
        reportMessage("\n"+strError );
    }
}

String strMirrorEntity = T("|Mirror entity|");
addRecalcTrigger(_kContext, strMirrorEntity );
if (_bOnRecalc && _kExecuteKey==strMirrorEntity )
{
    CoordSys cs = sp.coordSys();
    CoordSys csMirror;
    csMirror.setToMirroring(Plane(cs.ptOrg(), cs.vecX()));
    sp.transformBy(csMirror);
}

String strSetMirrorStyle = T("|Set mirror of style|");
addRecalcTrigger(_kContext, strSetMirrorStyle );
if (_bOnRecalc && _kExecuteKey==strSetMirrorStyle )
{
    CollectionDefinition mirrorStyleProp(pMirrorDefinition);
    colDef.setMirrorStyle(mirrorStyleProp);
}

```

```

        colDef.setMirrorPlane(nMirrorPlane);
    }

    String strMirrorContent = T("|Mirror content|");
    addRecalcTrigger(_kContext, strMirrorContent );
    if (_bOnRecalc && _kExecuteKey==strMirrorContent )
    {
        int mp = colDef.mirrorPlane();
        CoordSys csMirror;
        if (mp == 1) csMirror.setToMirroring(Plane(_PtW, _XW));
        else if (mp == 2) csMirror.setToMirroring(Plane(_PtW, _YW));
        else if (mp == 3) csMirror.setToMirroring(Plane(_PtW, _ZW));
        colDef.transformContent(csMirror);
    }

    CollectionDefinition mirrorStyle = colDef.mirrorStyle();

    String strLines[0];
    strLines.append("CollectionDefinition");
    strLines.append("entryName: "+colDef.entryName());
    strLines.append("entity: "+colDef.entity().length());
    strLines.append("genBeam: "+colDef.genBeam().length());
    strLines.append("beam: "+colDef.beam().length());
    strLines.append("sheet: "+colDef.sheet().length());
    strLines.append("sip: "+colDef.sip().length());
    strLines.append("tslInst: "+colDef.tslInst().length());
    strLines.append("mirrorPlane: "+colDef.mirrorPlane());
    strLines.append("mirrorStyle: "+mirrorStyle.entryName());

    // display the lines
    Display dp(-1);
    dp.dimStyle(pDimStyle);
    dp.textHeight(pTextHeight);
    for (int l=0; l<strLines.length(); l++) {
        Vector3d vecO = -l*1.2*pTextHeight*_YU;
        dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
    }
}

```

*—end example]*

---

[Example O-type TSL illustrating getAllEntryNamesFromDwg and importFromDwg:

```

Unit(1, "mm");

if (_bOnInsert) {

    String arListCur[] = CollectionDefinition().getAllEntryNames();
    PropString pListCur(0,arListCur,"Current list");
}

```

---

```

String strDwg = _kPathDwg + "\Source.dwg";
PropString pDwg(1, strDwg, "Dwg file to read from");
ShowDialog(T("|_Default|"));
strDwg = pDwg;
pDwg.setReadOnly(TRUE);

String arList[] =
CollectionDefinition().getAllEntryNamesFromDwg(strDwg);
PropString pListFromDwg(2, arList, T("|list from dwg|"));

PropString pEntry1(3, "", T("|Entry 1 to import|"));
PropString pEntry2(4, "", T("|Entry 2 to import|"));
PropInt pOverwrite(5, 0, T("|Overwrite|"));
ShowDialog();

String arToImport[0];
if (pEntry1 != "") arToImport.append(pEntry1);
if (pEntry2 != "") arToImport.append(pEntry2);
int bOverwrite = pOverwrite;

String strErr;
if (arToImport.length() == 1)
    strErr = CollectionDefinition().importFromDwg(strDwg,
arToImport[0], bOverwrite);
else
    strErr = CollectionDefinition().importFromDwg(strDwg,
arToImport, bOverwrite);

if (strErr != "")
{
    String strToLoad;
    for(int r=0; r<arToImport.length(); r++)
        strToLoad += " " + arToImport[r] + "";
    reportWarning("Could not load some of" + strToLoad + " from "
+ strDwg + ": " + strErr);
}

String arListCurNew[] = CollectionDefinition().getAllEntryNames();
PropString pListCurNew(6, arListCurNew, "New list");
ShowDialog(T("|_Default|"));

EraseInstance();
return;
}

```

*—end example]*

### 6.5.2.1 MetalPartCollectionDef

The MetalPartCollectionDef, which is derived from the [CollectionDefinition](#), is referenced by the [MetalPartCollectionEnt](#).

An DictObject can be casted into an MetalPartCollectionDef. However when the casting is not allowed, the resulting MetalPartCollectionDef will become invalid. This can be checked with the `bIsValid()` function of DictObject.

Because MetalPartCollectionDef is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The MetalPartCollectionDef from a MetalPartCollectionEnt can be constructed with the string returned by [CollectionEntity::definition\(\)](#):

---

```
class MetalPartCollectionDef : CollectionDefinition { // see CollectionDefinition for base
    functions (added hsbCAD2010 build 15.0.3)

    MetalPartCollectionDef(String strEntry); // constructor

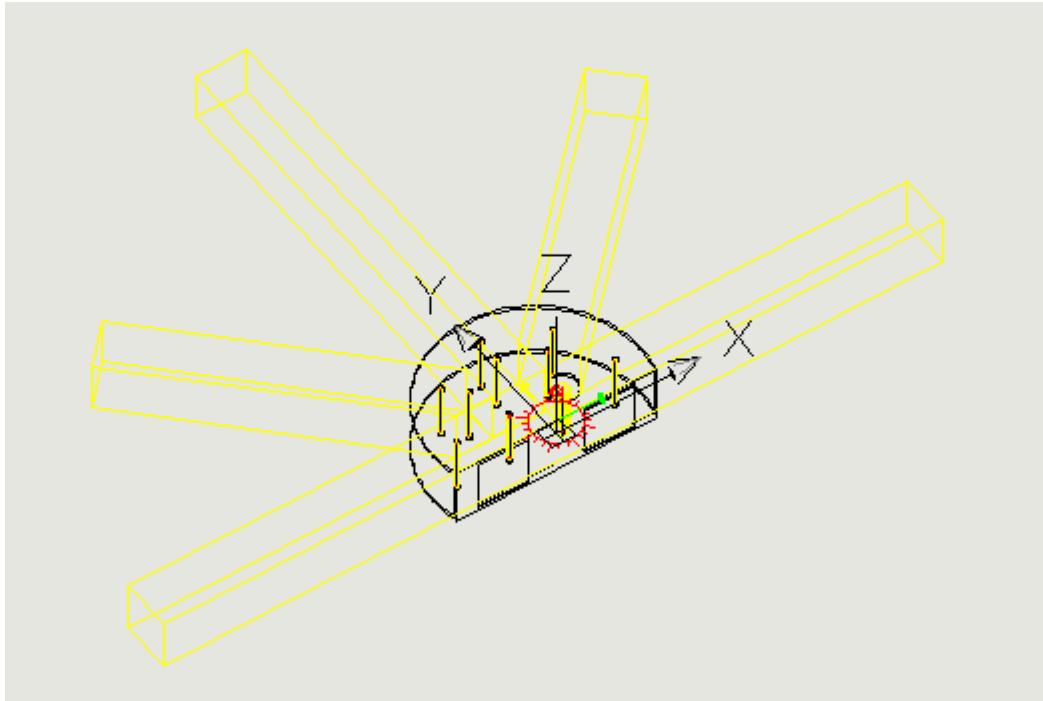
    static MetalPartCollectionDef[] getAllEntries();
    static String[] getAllEntryNames();

    // the following methods are added in hsbCAD20.0.68 and hsbCAD19.1.99
    string dbCreate(); // creates a new entry, make sure the strEntry that was set in the
                      // constructor does not exist yet. An error string is returned which is empty if
                      // all ok.

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    static String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from
                                                               dwg. Only use on special events.
    static String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return
                                                               error code as string. Empty string means all ok, see example below.
    static String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return
                                                               error code as string. Empty string means all ok, see example below
};
```

---

*[Example O-type TSL:*



```
Unit(1, "mm");

if (_bOnInsert) {

    String strAllDefinitions[] =
MetalPartCollectionDef().getAllEntryNames(); // list of all available
MetalPartCollectionDefs
    PropString pDefinition(0, strAllDefinitions,
"MetalPartCollectionEnt definition"); // make property
    showDialog(); // allow the user to change the definition

    // select insertion point
    _Pt0 = getPoint();

    if (strAllDefinitions.find(pDefinition) < 0)
    {
        // not found, create one
        String strNewName = pDefinition;
        MetalPartCollectionDef anew(strNewName);
        String strError = anew.dbCreate();
        if (strError == "")
        {
            // all fine
            reportMessage("\nMetalPartCollectionDef created with name:
"+anew.entryName());
            pDefinition.set(anew.entryName());
        }
        else
        {
```

```

        reportMessage("\nCould not create MetalPartCollectionDef
with name: "+strNewName);
        reportMessage("\n"+strError );
        eraseInstance();
        return;
    }
}

// create one
CoordSys csCE(_Pt0, _XU, _YU, _ZU);
String strDefinition = pDefinition;
MetalPartCollectionEnt sp;
sp.dbCreate(csCE, strDefinition);

_Entity.append(sp);

return;
}

if (_Entity.length()==0) return;
MetalPartCollectionEnt sp = (MetalPartCollectionEnt) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strDefinition = sp.definition();
PropString pDefinition(0, strDefinition , "MetalPartCollectionEnt
definition"); // make property
pDefinition.set(strDefinition);
pDefinition.setReadOnly(TRUE);

// compose a MetalPartCollectionDef from the String
MetalPartCollectionDef mpColDef(strDefinition);

String strRedefine = T("|Redefine content of definition|");
addRecalcTrigger(_kContext, strRedefine );
if (_bOnRecalc && _kExecuteKey==strRedefine )
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go())
        Entity ents[] =ssE.set();
        Point3d ptRef = getPoint();
        CoordSys cs(ptRef, _XU, _YU, _ZU);
        mpColDef.setContent(cs, ents, FALSE);
}
}

String strRedefineErase = T("|Redefine content of definition and
erase|");
addRecalcTrigger(_kContext, strRedefineErase );
if (_bOnRecalc && _kExecuteKey==strRedefineErase )

```

```

{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        Entity ents[] = ssE.set();
        Point3d ptRef = getPoint();
        CoordSys cs(ptRef, _XU, _YU, _ZU);
        mpColDef.setContent(cs, ents, TRUE);
    }
}

String strClear = T("|Clear content of definition|");
addRecalcTrigger(_kContext, strClear );
if (_bOnRecalc && _kExecuteKey==strClear )
{
    mpColDef.clearContent();
}

String strDbCreate = T("|Create a new definition|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    String strNewName = getString("new name:");
    MetalPartCollectionDef anew(strNewName);
    String strError = anew.dbCreate();
    if (strError == "")
    {
        // all fine
        mpColDef = anew;
        sp.setDefinitionObject(anew);
        pDefinition.set(sp.definition());
        reportMessage("\nMetalPartCollectionDef created with name:
"+mpColDef.entryName());
    }
    else
    {
        reportMessage("\nCould not create MetalPartCollectionDef with
name: "+strNewName);
        reportMessage("\n"+strError );
    }
}

String strMoveEntity = T("|Move entity|");
addRecalcTrigger(_kContext, strMoveEntity );
if (_bOnRecalc && _kExecuteKey==strMoveEntity )
{
    CoordSys cs = sp.coordSys();
    Vector3d vecMove = _Pt0 - cs.ptOrg();
    sp.transformBy(vecMove);
}

```

```

String strLines[0];
strLines.append("MetalPartCollectionDef");
strLines.append("entryName: "+mpColDef.entryName());
strLines.append("entity: "+mpColDef.entity().length());
strLines.append("genBeam: "+mpColDef.genBeam().length());
strLines.append("beam: "+mpColDef.beam().length());
strLines.append("sheet: "+mpColDef.sheet().length());
strLines.append("sip: "+mpColDef.sip().length());
strLines.append("tslInst: "+mpColDef.tslInst().length());

PropString pDimStyle(1,_DimStyles,"Dim style");
PropDouble pTextHeight(0,U(20),"Text height");

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

—end example]

```

---

[Example O-type TSL illustrating getAllEntryNamesFromDwg and importFromDwg:

```

Unit(1,"mm");

if (_bOnInsert) {

    String arListCur[] = MetalPartCollectionDef().getAllEntryNames();
    PropString pListCur(0,arListCur,"Current list");

    String strDwg = _kPathDwg + "\\Source.dwg";
    PropString pDwg(1, strDwg, "Dwg file to read from");
    showDialog(T("|_Default|"));
    strDwg = pDwg;
    pDwg.setReadOnly(TRUE);

    String arList[] =
MetalPartCollectionDef().getAllEntryNamesFromDwg(strDwg);
    PropString pListFromDwg(2,arList, T("|list from dwg|"));

    PropString pEntry1(3,"",T("|Entry 1 to import|"));
    PropString pEntry2(4,"",T("|Entry 2 to import|"));
    PropInt pOverwrite(5,0,T("|Overwrite|"));
    showDialog();
}

```

---

```

String arToImport[0];
if (pEntry1 != "") arToImport.append(pEntry1);
if (pEntry2 != "") arToImport.append(pEntry2);
int bOverwrite = pOverwrite;

String strErr;
if (arToImport.length() == 1)
    strErr = MetalPartCollectionDef().importFromDwg(strDwg,
arToImport[0], bOverwrite);
else
    strErr = MetalPartCollectionDef().importFromDwg(strDwg,
arToImport, bOverwrite);

if (strErr != "")
{
    String strToLoad;
    for(int r=0; r<arToImport.length(); r++)
        strToLoad += " " + arToImport[r] + "!";
    reportWarning("Could not load some of" + strToLoad + " from "
+ strDwg + ": " + strErr);
}

String arListCurNew[] =
MetalPartCollectionDef().getAllEntryNames();
PropString pListCurNew(6,arListCurNew,"New list");
showDialog(T("|_Default|"));

eraseInstance();
return;
}

```

*—end example]*

#### 6.5.2.2 TrussDefinition

The TrussDefinition, which is derived from the [CollectionDefinition](#), is referenced by the [TrussEntity](#)

An DictObject can be casted into an TrussDefinition. However when the casting is not allowed, the resulting TrussDefinition will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because TrussDefinition is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The TrussDefinition from a TrussEntity can be constructed with the string returned by [CollectionEntity::definition\(\)](#);

```

class TrussDefinition : CollectionDefinition { // see CollectionDefinition for base functions
    (added hsbCAD14.0.73)

    TrussDefinition(String strEntry); // constructor

    static TrussDefinition[] getAllEntries();
    static String[] getAllEntryNames();

    // the following methods are added in hsbCAD20.0.68 and hsbCAD19.1.99
    string dbCreate(); // creates a new entry, make sure the strEntry that was set in the
                      // constructor does not exist yet. An error string is returned which is empty if
                      // all ok.

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    static String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from
                                                                dwg. Only use on special events.
    static String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return
                                                                error code as string. Empty string means all ok, see example below.
    static String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return
                                                                error code as string. Empty string means all ok, see example below
}

```

---

[Example O-type TSL:

```

Unit(1, "mm");

if (_bOnInsert) {

    String strAllDefinitions[] =
TrussDefinition().getAllEntryNames(); // list of all available
TrussDefinitions
    PropString pDefinition(0, strAllDefinitions, "TrussEntity
definition"); // make property
    showDialog(); // allow the user to change the definition

    // select insertion point
    _Pt0 = getPoint();

    // create one
    CoordSys csCE(_Pt0, _XU, _YU, _ZU);
    String strDefinition = pDefinition;
    TrussEntity sp;
    sp.dbCreate(csCE, strDefinition);

    _Entity.append(sp);
}

```

---

```

        return;
    }

    if (_Entity.length()==0) return;
    TrussEntity sp = (TrussEntity) _Entity[0];
    if (!sp.bIsValid()) {
        eraseInstance(); // just erase from DB
        return;
    }

    String strDefinition = sp.definition();
    PropString pDefinition(0, strDefinition, "TrussEntity
definition"); // make property
pDefinition.set(strDefinition);
pDefinition.setReadOnly(TRUE);

// compose a TrussDefinition from the String
TrussDefinition trussDefinition(strDefinition);

String strRedefine = T("|Redefine content of definition|");
addRecalcTrigger(_kContext, strRedefine );
if (_bOnRecalc && _kExecuteKey==strRedefine )
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        Entity ents[] =ssE.set();
        Point3d ptRef = getPoint();
        CoordSys cs(ptRef, _XU, _YU, _ZU);
        trussDefinition.setContent(cs, ents, FALSE);
    }
}

String strRedefineErase = T("|Redefine content of definition and
erase|");
addRecalcTrigger(_kContext, strRedefineErase );
if (_bOnRecalc && _kExecuteKey==strRedefineErase )
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        Entity ents[] =ssE.set();
        Point3d ptRef = getPoint();
        CoordSys cs(ptRef, _XU, _YU, _ZU);
        trussDefinition.setContent(cs, ents, TRUE);
    }
}

String strClear = T("|Clear content of definition|");
addRecalcTrigger(_kContext, strClear );
if (_bOnRecalc && _kExecuteKey==strClear )
{
    trussDefinition.clearContent();
}

```

```

String strDbCreate = T("|Create a new definition|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    String strNewName = getString("new name:");
    TrussDefinition anew(strNewName);
    String strError = anew.dbCreate();
    if (strError == "")
    {
        // all fine
        trussDefinition = anew;
        sp.setDefinitionObject(anew);
        pDefinition.set(sp.definition());
        reportMessage("\nTrussDefinition created with name:
"+trussDefinition.entryName());
    }
    else
    {
        reportMessage("\nCould not create TrussDefinition with name:
"+strNewName);
        reportMessage("\n"+strError );
    }
}

String strMoveEntity = T("|Move entity|");
addRecalcTrigger(_kContext, strMoveEntity );
if (_bOnRecalc && _kExecuteKey==strMoveEntity )
{
    CoordSys cs = sp.coordSys();
    Vector3d vecMove = _Pt0 - cs.ptOrg();
    sp.transformBy(vecMove);
}

String strLines[0];
strLines.append("TrussDefinition");
strLines.append("entryName: "+trussDefinition.entryName());
strLines.append("entity: "+trussDefinition.entity().length());
strLines.append("genBeam: "+trussDefinition.genBeam().length());
strLines.append("beam: "+trussDefinition.beam().length());
strLines.append("sheet: "+trussDefinition.sheet().length());
strLines.append("sip: "+trussDefinition.sip().length());
strLines.append("tslInst: "+trussDefinition.tslInst().length());

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);

```

```

    for (int l=0; l<strLines.length(); l++) {
        Vector3d vecO = -l*1.2*pTextHeight*_YU;
        dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
    }

—end example]

```

---

[Example O-type TSL illustrating getAllEntryNamesFromDwg and importFromDwg:

```

Unit(1, "mm");

if (_bOnInsert) {

    String arListCur[] = TrussDefinition().getAllEntryNames();
    PropString pListCur(0,arListCur,"Current list");

    String strDwg = _kPathDwg + "\\Source.dwg";
    PropString pDwg(1, strDwg, "Dwg file to read from");
    showDialog(T("|_Default|"));
    strDwg = pDwg;
    pDwg.setReadOnly(TRUE);

    String arList[] =
TrussDefinition().getAllEntryNamesFromDwg(strDwg);
    PropString pListFromDwg(2,arList, T("list from dwg"));

    PropString pEntry1(3,"",T("|Entry 1 to import|"));
    PropString pEntry2(4,"",T("|Entry 2 to import|"));
    PropInt pOverwrite(5,0,T("|Overwrite|"));
    showDialog();

    String arToImport[0];
    if (pEntry1 != "") arToImport.append(pEntry1);
    if (pEntry2 != "") arToImport.append(pEntry2);
    int bOverwrite = pOverwrite;

    String strErr;
    if (arToImport.length() == 1)
        strErr = TrussDefinition().importFromDwg(strDwg, arToImport[0],
bOverwrite);
    else
        strErr = TrussDefinition().importFromDwg(strDwg, arToImport,
bOverwrite);

    if (strErr != "")
{
```

```

        String strToLoad;
        for(int r=0; r<arToImport.length(); r++)
            strToLoad += " " + arToImport[r] + "!";
        reportWarning("Could not load some of" + strToLoad + " from "
+ strDwg + ":" + strErr);
    }

    String arListCurNew[] = TrussDefinition().getAllEntryNames();
    PropString pListCurNew(6, arListCurNew, "New list");
    showDialog(T("|_Default|"));

    eraseInstance();
    return;
}

```

*—end example]*

### 6.5.3 CurvedStyle

The CurvedStyle, which is derived from the [DictObject](#), refers to a curved style description used in eg laminated timber.

An DictObject can be casted into an CurvedStyle. However when the casting is not allowed, the resulting CurvedStyle will become invalid. This can be checked with the blsValid() function of DictObject.

Because CurvedStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

There is one predefined curved style name. This predefined is of type String: [\\_kStraight](#).

To get the CurvedStyle from a [Beam](#), one can use curvedStyle method.

- The CurvedStyle and all its 2D geometry has an origin point which defines the CurvedStyle insertion point.
- The CurveStyle insertion point, (0,0) is mapped to Beam.ptRef() whenever a Beam its curvedStyle is set.
- The 2D XY plane of the CurvedStyle is mapped to the plane Beam.vecX(), Beam.vecZ()

```

class CurvedStyle : DictObject { // see DictObject for base functions

    CurvedStyle(String strEntry); // constructor

    void dbCreate(PLine plBase, PLine plClosed, Vector3d vecDir, Point3d ptRef, double
dWidth);
    void dbCreate(PLine plBase, PLine plClosed, Vector3d vecXPlane, Point3d ptRef, double
dWidth, Point3d ptOrgPlane, Vector3d vecYPlane); // (added from hsbCAD
18.2.17)

    double beamWidth() const;
}

```

```
void setBeamWidth(double dNew);

String description() const;
void setDescription(String strDesc);

PLine baseCurve() const;
PLine baseCurve(int bExtend) const;

PLine topCurve() const; // (added from hsbCAD 18.2.17)
PLine topCurve(int bExtend) const; // (added from hsbCAD 18.2.17)

PLine closedCurve() const;

PLine midCurve() const;
PLine midCurve(int bExtend) const;

static CurvedStyle[] getAllEntries();
static String[] getAllEntryNames();

// following methods are added from hsbCAD 18.2.0 on

double lamThickness() const;
void setLamThickness(double dNew);
double cuttingCurveOffset() const;
void setCuttingCurveOffset(double dNew);
double lamMinimumLength() const;
void setLamMinimumLength(double dNew);
double lamExtraLength() const;
void setLamExtraLength(double dNew);
double lamExtraHeightRaw() const;
void setLamExtraHeightRaw(double dNew);
double lamExtraWidthRaw() const;
void setLamExtraWidthRaw(double dNew);

double glueDensity() const;
void setGlueDensity(double dNew);

String lamGroups() const;
void setLamGroups(String strNew);
String woodClass() const;
void setWoodClass(String strNew);
String lamGrading() const;
void setLamGrading(String strNew);
String woodKind() const;
void setWoodKind(String strNew);

int dryJointLamIndex() const;
void setDryJointLamIndex(int nNew);
int dryJointLamColor() const;
void setDryJointLamColor(int nNew);

double dDisplacementRefPt2() const; // displacement at lam interface of lam with index
nLamIndexRefPt2
```

```

int nLamIndexRefPt2() const; // 0 based index of the first lam that uses the RefPt2. If value
    is 0, RefPt2 is not used.

int numLams() const;
String woodClassAt(int nlnd) const;
double lamSizeAt(int nlnd, int bLeft) const;

Point3d ptRef() const; // the reference point is the location where the straight stack of
    lams is build from.

void setPtInsert(double dX, double dY); // added hsbCAD 21.1.29 set the new insertion
    point
};

```

---

**void** setPtInsert(**double** dX, **double** dY); // added hsbCAD 21.1.29

The insertion point is always at (0,0), so it is always the origin. When a new insertion point is set, the internal geometry is moved with a vector (-dX,-dY) resulting in a new origin point for the geometry.

---

[Example O-type TSL:

```

String strStraight = _kStraight;

CurvedStyle cs; // the default constructor creates a straight one

// collect list of curved styles
CurvedStyle csList[] = CurvedStyle().getAllEntries();
reportNotice("\n ======");
for (int e=0; e<csList.length(); e++) {
    reportNotice("\n entryName: "+csList[e].entryName());
}

// declare a property with the names of the curved styles
PropString pS(0, CurvedStyle().getAllEntryNames(), T("|Curved style
name|"));

CurvedStyle csInvest(pS); // constructor with a curved style name
reportNotice("\n ----");
reportNotice("\n entryName: "+csInvest.entryName());
reportNotice("\n isValid: "+csInvest.bIsValid());
reportNotice("\n bIsSpecial: "+csInvest.bIsSpecial());
reportNotice("\n handle: "+csInvest.handle());
reportNotice("\n typeName: "+csInvest.typeName());

```

—end example]

[Example O-type TSL, that dbCreates a new curvedStyle:

```
U(1, "mm");
if (_bOnInsert) {
    PLine plBase = getEntPLine(T(" | Select base curve|")).getPLine();
    PLine plClosed = getEntPLine(T(" | Select closed
curve|")).getPLine();
    Pt0 = getPoint(T(" | Select reference point|"));
    double dWidth = U(200);
    Vector3d vecDir = _xu;

    CurvedStyle curvedStyle("tlsCreatedStyle");
    curvedStyle.dbCreate(plBase, plClosed, vecDir, Pt0, dWidth);
    if (curvedStyle.bIsValid()) {
        reportMessage(T("\n|Successful created| "));
        +curvedStyle.entryName();
    }
    else {
        reportMessage(T("\n|NOT created| ")) +curvedStyle.entryName();
    }

    eraseInstance();
    return;
}
```

—end example]

[Example O-type TSL, that shows curvedStyle of [CurvedDescription](#):

```
Unit(1, "mm");

PropString pDimStyle(1, DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_bOnInsert) {
    Pt0 = getPoint();

    // get the CurvedDescription
    CurvedDescription mp = getCurvedDescription();
    _Entity.append(mp);
}

// check entity link
CurvedDescription ent;
if (_Entity.length()>0) ent = (CurvedDescription)_Entity[0];
if (!ent.bIsValid()) {
    eraseInstance();
    return;
}
```

```

CoordSys css =ent.coordSys();
css.vis();

CurvedStyle myCurvedStyle = CurvedStyle(ent.style());

PLine plClosed = myCurvedStyle.closedCurve();
plClosed .transformBy(css); // transform to curved description
location
plClosed .transformBy(_Pt0-css.ptOrg()); // move to location of tsl
plClosed .vis(3);

PLine plBase= myCurvedStyle.baseCurve();
plBase.transformBy(css); // transform to curved description location
plBase.transformBy(_Pt0-css.ptOrg()); // move to location of tsl
plBase.vis(2);

PLine plMid = myCurvedStyle.midCurve();
plMid .transformBy(css); // transform to curved description location
plMid .transformBy(_Pt0-css.ptOrg()); // move to location of tsl
plMid .vis(1);

String strChangeEntity = T("Change CurvedDescription");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity )
{
    myCurvedStyle.setBeamWidth(myCurvedStyle.beamWidth() + U(1));
    myCurvedStyle.setDescription(myCurvedStyle.description() + "-A");
    myCurvedStyle.setLamThickness(myCurvedStyle.lamThickness() +
U(2));

    myCurvedStyle.setCuttingCurveOffset(myCurvedStyle.cuttingCurveOffset(
) + U(3));
    myCurvedStyle.setLamMinimumLength(myCurvedStyle.lamMinimumLength()
+ U(4));
    myCurvedStyle.setLamExtraLength(myCurvedStyle.lamExtraLength() +
U(5));

    myCurvedStyle.setLamExtraHeightRaw(myCurvedStyle.lamExtraHeightRaw()
+ U(6));
    myCurvedStyle.setLamExtraWidthRaw(myCurvedStyle.lamExtraWidthRaw()
+ U(7));
    myCurvedStyle.setGlueDensity(myCurvedStyle.glueDensity() + U(8));
    myCurvedStyle.setLamGroups(myCurvedStyle.lamGroups() + ",4");
    myCurvedStyle.setWoodClass(myCurvedStyle.woodClass() + ";");
    myCurvedStyle.setLamGrading(myCurvedStyle.lamGrading() + "G");
    myCurvedStyle.setWoodKind(myCurvedStyle.woodKind() + "K");
    myCurvedStyle.setDryJointLamColor(myCurvedStyle.dryJointLamColor()
+ 1);
    myCurvedStyle.setDryJointLamIndex(myCurvedStyle.dryJointLamIndex()
+ 2);
}

```

```

String strSetPtInsert = T("Set insertion point");
addRecalcTrigger(_kContext, strSetPtInsert );
if (_bOnRecalc && _kExecuteKey == strSetPtInsert )
{
    Point3d ptNew = getPoint();
    CoordSys csToLocal = csS;
    csToLocal.invert();
    ptNew.transformBy(csToLocal);
    myCurvedStyle.setPtInsert(ptNew.X() , ptNew.Y());
}

String strLines[0];
strLines.append("Curved Style");

strLines.append("beamWidth:"+ myCurvedStyle.beamWidth());
strLines.append("description:"+ myCurvedStyle.description());
strLines.append("lamThickness:"+ myCurvedStyle.lamThickness());
strLines.append("cuttingCurveOffset:"+
myCurvedStyle.cuttingCurveOffset());
strLines.append("lamMinimumLength:"+
myCurvedStyle.lamMinimumLength());
strLines.append("lamExtraLength:"+ myCurvedStyle.lamExtraLength());
strLines.append("lamExtraHeightRaw:"+
myCurvedStyle.lamExtraHeightRaw());
strLines.append("lamExtraWidthRaw:"+
myCurvedStyle.lamExtraWidthRaw());
strLines.append("glueDensity:"+ myCurvedStyle.glueDensity());
strLines.append("lamGroups:"+ myCurvedStyle.lamGroups());
strLines.append("woodClass:"+ myCurvedStyle.woodClass());
strLines.append("lamGrading:"+ myCurvedStyle.lamGrading());
strLines.append("woodKind:"+ myCurvedStyle.woodKind());
strLines.append("dryJointLamColor:"+
myCurvedStyle.dryJointLamColor());
strLines.append("dryJointLamIndex:"+
myCurvedStyle.dryJointLamIndex());

strLines.append("dDisplacementRefPt2:"+
myCurvedStyle.dDisplacementRefPt2());
strLines.append("nLamIndexRefPt2:"+ myCurvedStyle.nLamIndexRefPt2());

// Lams
int nNumLams= myCurvedStyle.numLams();
strLines.append("num lams: "+nNumLams);
for (int p=0; p<nNumLams; p++) {
    double dSizeL = myCurvedStyle.lamSizeAt(p,TRUE);
    double dSizeR = myCurvedStyle.lamSizeAt(p,FALSE);
    String strWoodClass = myCurvedStyle.woodClassAt(p);
    strLines.append(p + " wood class: "+strWoodClass + " lam left:
"+dSizeL + " lam right: "+dSizeR );
}

```

```
// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}
```

—end example]

[Example E-type TSL, that creates a new curved description, but with different orientation of the curves internally. It also adjusts the beam coodsys to compensate the new internal definition

```
double dEps = Unit(0.01, "mm");

if ( _Beam.length() == 0 ) {
    reportWarning(TN("|No beam selected.|"));
    eraseInstance();
    return;
}

Beam bm = _Beam[0];
if (bm.curvedStyle() == _kStraight) {
    reportMessage("Beam is straight.");
    return;
}

String strChangeEntity = T("Change CurvedDescription");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity )
{
    CoordSys csBm = bm.coordSys();
    Point3d ptBmRef = bm.ptRef();
    Vector3d vxBm = csBm.vecX();
    Vector3d vyBm = csBm.vecY();
    Vector3d vzBm = csBm.vecZ();
    Point3d ptBmOrg = csBm.ptOrg();

    CurvedStyle thisCurvedStyle(bm.curvedStyle());
    double dWidth = thisCurvedStyle.beamWidth();
    PLine plBaseCurve = thisCurvedStyle.baseCurve();
    PLine plClosedCurve = thisCurvedStyle.closedCurve();

    CoordSys csToBm;
    csToBm.setToAlignCoordSys(_PtW, _XW, _YW, _ZW, ptBmRef, vxBm,
    vzBm, -vyBm);
    plBaseCurve.transformBy(csToBm);
    plClosedCurve.transformBy(csToBm);

    double dAngle = 5;
```

```

CoordSys csRotP;  csRotP.setToRotation(dAngle, vyBm, ptBmOrg);
csBm.transformBy(csRotP);

CoordSys csBmNew(_Pt0, csBm.vecX(), csBm.vecY(), csBm.vecZ());
Point3d ptRef = csBmNew.ptOrg();
Point3d ptOrgPlane = csBmNew.ptOrg();
Vector3d vecXPlane = csBmNew.vecX();
Vector3d vecYPlane = csBmNew.vecZ();

// find a new name that does not exist yet
String strNewName = thisCurvedStyle.entryName()+"N";
while(CurvedStyle().getAllEntryNames().find(strNewName)>=0)
    strNewName = strNewName +"N";

CurvedStyle curvedStyleNew(strNewName);
curvedStyleNew.dbCreate(plBaseCurve, plClosedCurve, vecXPlane,
ptRef, dWidth, ptOrgPlane, vecYPlane);
if (curvedStyleNew.bIsValid()) {
    reportMessage(T("\n|Successful created| "))
+curvedStyleNew.entryName());
    bm.setCurvedStyle(curvedStyleNew.entryName());
    bm.setCoordSys(csBmNew);
}
else {
    reportMessage(T("\n|NOT created| "))
+curvedStyleNew.entryName());
}
}

CurvedStyle thisCurvedStyle(bm.curvedStyle());
CoordSys csBm = bm.coordSys();
Point3d ptBmRef = bm.ptRef();
Vector3d vxBm = csBm.vecX();
Vector3d vyBm = csBm.vecY();
Vector3d vzBm = csBm.vecZ();

PLine plMidCurve = thisCurvedStyle.midCurve();
PLine plBaseCurve = thisCurvedStyle.baseCurve();
PLine plClosedCurve = thisCurvedStyle.closedCurve();

CoordSys csToBm;
csToBm.setToAlignCoordSys(_PtW, _XW, _YW, _ZW, ptBmRef, vxBm, vzBm, -
vyBm);

plMidCurve.transformBy(csToBm);
//plMidCurve.vis(1);

plBaseCurve.transformBy(csToBm);
plBaseCurve.vis(3);
Point3d ptS = plBaseCurve.ptStart();
ptS.vis(3);
CoordSys csPl = plBaseCurve.coordSys();

```

```

csPl.vis(3);

plClosedCurve.transformBy(csToBm);
plClosedCurve.vis(4);
ptS = plClosedCurve.ptStart();
ptS.vis(4);
csPl = plClosedCurve.coordSys();
csPl.vis(4);

csBm.vis(1);
ptBmRef.vis(1);
//vxBm.vis(ptBm, 1);
//vyBm.vis(ptBm, 1);

Vector3d vecDir = vxBm;
double dAngle = 5;
vecDir.rotateBy(dAngle, vyBm);
vecDir.vis(_Pt0, 3);

—end example]

```

#### 6.5.4 ExtrProfile

The ExtrProfile, which is derived from the [DictObject](#), refers to an extrusion profile description.

An DictObject can be casted into an ExtrProfile. However when the casting is not allowed, the resulting ExtrProfile will become invalid. This can be checked with the `bIsValid()` function of DictObject.

Because ExtrProfile is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

There are 2 predefined extrusion profile names. These predefines are of type String:  
`_kExtrProfRectangular` and `_kExtrProfRound`.

To define a tool with the ExtrProfile, one should use the [ExtrProfileCut](#) tool.

```

class ExtrProfile : DictObject { // see DictObject for base functions

    ExtrProfile(String strEntry); // constructor

    PlaneProfile planeProfile() const; // return the PlaneProfile that describes the extrusion
                                     // profile
    PlaneProfile planeProfile(double dSizeX, double dSizeY) const; // return the scaled
                                     // PlaneProfile that fits inside a rectangle with sizes dSizeX and dSizeY

    // the componentProfiles method is added in hsbCad17.2.38, hsbCad18.1.51
    PlaneProfile[] componentProfiles() const; // return the PlaneProfiles that express the
                                     // different component profiles
    PlaneProfile[] componentProfiles(double dSizeX, double dSizeY) const; // return the scaled
                                     // PlaneProfiles that fits inside a rectangle with sizes dSizeX and dSizeY

    String[] componentMaterials() const; // (added in build 24.1.68 and 25.1.20) return the
                                     // materials of the different components. The length of the array is identical to
                                     // componentProfiles array.

    int blsScalable() const; // returns whether the profile is defined scalable

    static ExtrProfile[] getAllEntries();
    static String[] getAllEntryNames();

    String timberName() const; // (added hsbCad17.1.4).
    String timberMaterial() const; // (added hsbCad17.1.4).
    String timberGrade() const; // (added hsbCad17.1.4).

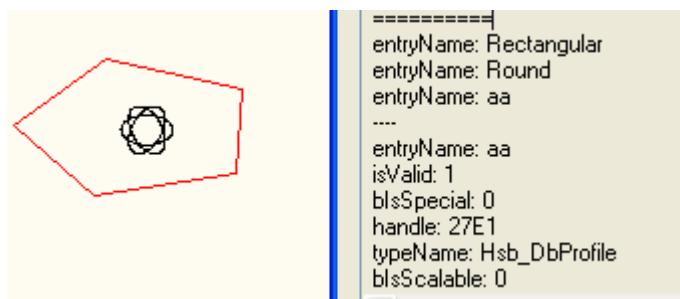
    double dTongueHeight() const; // (added hsbCad17.1.4).

};

```

---

[Example O-type TSL:



```

String strRec = \_kExtrProfRectangular;
String strRnd = \_kExtrProfRound;

ExtrProfile ep; // the default constructor creates a rectangular one

```

---

```

// collect list of extrusion profiles
ExtrProfile epList[] = ExtrProfile().getAllEntries();
reportNotice("\n =====");
for (int e=0; e<epList.length(); e++) {
    reportNotice("\n entryName: "+epList[e].entryName());
}

// declare a property with the names of the extrusion profiles
PropString pS(0, ExtrProfile().getAllEntryNames(), T("|Extrusion
profile name|"));

ExtrProfile epInvest(pS); // constructor with a extrusion profile
name
reportNotice("\n ----");
reportNotice("\n entryName: "+epInvest.entryName());
reportNotice("\n isValid: "+epInvest.bIsValid());
reportNotice("\n bIsSpecial: "+epInvest.bIsSpecial());
reportNotice("\n handle: "+epInvest.handle());
reportNotice("\n typeName: "+epInvest.typeName());
reportNotice("\n bIsScalable: "+epInvest.bIsScalable());

// profile not scaled, and located around origin
PlaneProfile profNotScaled = epInvest.planeProfile();

Display dp(-1);
dp.textHeight(U(10));

// profile scaled
PlaneProfile prof = epInvest.planeProfile(U(200),U(100));
// transform towards a specific location
prof.transformBy(CoordSys(_Pt0,_XU,_YU,_ZU));
dp.draw(prof);

// components scaled
PlaneProfile comps[] = epInvest.componentProfiles(U(200),U(100));
String compMat[] = epInvest.componentMaterials();
for (int c=0; c<comps.length(); c++)
{
    PlaneProfile prof = comps[c];
    String mat = compMat[c];
    prof.transformBy(CoordSys(_Pt0,_XU,_YU,_ZU));
    dp.color(3+c);
    dp.draw(prof);
    dp.draw(mat, prof.ptMid(), _XU,_YU, 0, 0);
}

```

*—end example]*

## 6.5.5 FastenerAssemblyDef

The FastenerAssemblyDef, which is derived from the [DictObject](#), is referred to by a [FastenerAssemblyEnt](#) entity.

An DictObject can be casted into an FastenerAssemblyDef. However when the casting is not allowed, the resulting FastenerAssemblyDef will become invalid. This can be checked with the `bIsValid()` function of DictObject.

Because FastenerAssemblyDef is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The FastenerAssemblyDef from a FastenerAssemblyEnt can be constructed with the string returned by `FastenerAssemblyEnt::style()`:

---

```

class FastenerAssemblyDef : DictObject { // see DictObject for base functions (added
  hsbCAD2012 17.0.21)

  FastenerAssemblyDef(String strEntry); // constructor

  void dbCreate(); // added in V26. Use setListComponent and other setters to set
    components and description.

  FastenerListComponent listComponent() const; // see FastenerListComponent
  void setListComponent(FastenerListComponent comp);

  FastenerSimpleComponent mainComponent(String strArticleNumber) const; // find the
    articleData in the listComponent that matches the strArticleNumber

  FastenerSimpleComponent[] headComponents() const; // see FastenerSimpleComponent
  void setHeadComponents(FastenerSimpleComponent\[\] arSimpleComp);

  FastenerSimpleComponent[] tailComponents() const; // see FastenerSimpleComponent
  void setTailComponents(FastenerSimpleComponent\[\] arSimpleComp);

  static FastenerAssemblyDef[] getAllEntries() const;
  static String[] getAllEntryNames() const;

  String description() const;
  void setDescription(String str); // can contain <DESC> to be substituted by the
    mainComponent its description

// added in hsbCAD20.1.69 and hsbCAD21.0.43
String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
  Only use on special events.
String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
  code as string. Empty string means all ok, see SipStyle example.

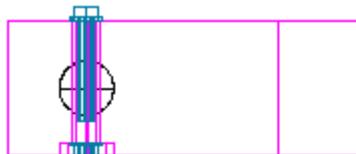
```

---

```
String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
code as string. Empty string means all ok, see SipStyle example.
};
```

---

[Example O-type TSL:



```
-----
flc norm: DIn931
flc type: Bolt
flc subType:
0: fad articleNumber: 10b13ddf-b1f1
    fad fastenerLength: 65
    fad description: PBø12*65
1: fad articleNumber: 8f504abd-cd5
    fad fastenerLength: 70
    fad description: PBø12*70
2: fad articleNumber: 0af42b87-b0f
    fad fastenerLength: 75
    fad description: PBø12*75
3: fad articleNumber: fb7c0d18-6c9
    fad fastenerLength: 80
    fad description: PBø12*80
4: fad articleNumber: 9991n32n-7rv

Unit(1,"mm");

if (_bOnInsert) {

    // select the entity and insertion point
    FastenerAssemblyEnt sp = getFastenerAssemblyEnt();
    _Entity.append(sp);
    _Pt0 = getPoint();

    String strStyle = sp.definition(); // get the definition from the
selected entity

    String strAllStyles[] =
FastenerAssemblyDef().getAllEntryNames(); // list of all available
FastenerAssemblyDefs
    PropString pStyle(0, strAllStyles, "Fastener assembly def",
strAllStyles.find(strStyle)); // make property
    showDialog(); // allow the user to change the style
}
```

---

```

// change the FastenerAssemblyDef of the entity, to match the
property value
sp.setDefinition(pStyle);
sp.setLengthSelectionIsAutomatic(FALSE);

FastenerAssemblyDef fadef(pStyle);
FastenerListComponent flc = fadef.listComponent();
FastenerArticleData arFad[] = flc.articleDataSet();
sp.setArticleNumber(arFad[arFad.length()-1].articleNumber());

return;
}

if (_Entity.length()==0) return;
FastenerAssemblyEnt sp = (FastenerAssemblyEnt) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strStyle = sp.definition();
PropString pStyle(0, strStyle, "Fastener assembly def"); // make
property
pStyle.set(strStyle);
pStyle.setReadOnly(TRUE);

PropString pDimStyle(1,_DimStyles,"Dim style");
PropDouble pTextHeight(0,U(20),"Text height");
String strLines[0];

// compose a FastenerAssemblyDef from the String
FastenerAssemblyDef fadef(strStyle);
FastenerListComponent flc = fadef.listComponent();

FastenerComponentData fcd = flc.componentData();
strLines.append("-----");
strLines.append("flc norm: "+fcd.norm());
strLines.append("flc type: "+fcd.type());
strLines.append("flc subType: "+fcd.subType());

FastenerArticleData arFad[] = flc.articleDataSet();
for (int a=0; a<arFad.length(); a++) {
    FastenerArticleData fad = arFad[a];
    strLines.append(a+": fad articleNumber: "+fad.articleNumber());
    strLines.append("      fad fastenerLength: "+fad.fastenerLength());
    strLines.append("      fad description: "+fad.description());
}

FastenerSimpleComponent arHead[] = fadef.headComponents();
strLines.append("Head components: "+arHead.length());
for (int c=0; c<arHead.length(); c++) {

```

```

FastenerSimpleComponent fsc = arHead[c];
FastenerArticleData fad = fsc.articleData();
strLines.append(c+": fad description: "+fad.description());
}

FastenerSimpleComponent arTail[] = faDef.tailComponents();
strLines.append("Tail components: "+arTail.length());
for (int c=0; c<arTail.length(); c++) {
    FastenerSimpleComponent fsc = arTail[c];
    FastenerArticleData fad = fsc.articleData();
    strLines.append(c+": fad description: "+fad.description());
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
};

—end example]

```

[Example O-type, illustrating dbErase and dbCreate:

```

Unit(1, "mm");

if (_bOnInsert) {

    // select the entity and insertion point
    FastenerAssemblyEnt sp = getFastenerAssemblyEnt();
    _Entity.append(sp);
    _Pt0 = getPoint();

    return;
}

if (_Entity.length() == 0) return;
FastenerAssemblyEnt sp = (FastenerAssemblyEnt) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strStyleOrig = sp.definition(); // get the definition from the
selected entity
String strStyleCloned = strStyleOrig + "_clone";

String strRemoveClonedStyle = T("|dbErased cloned style|");
addRecalcTrigger(_kContext, strRemoveClonedStyle );
if (_bOnRecalc && _kExecuteKey==strRemoveClonedStyle)

```

```

{
    FastenerAssemblyDef def(strStyleCloned);
    if (def.bIsValid())
    {
        def.dbErase();

        if (def.bIsValid())
        {
            reportMessage("\n" + strStyleCloned + " not dbErased!");
        }
        else
        {
            reportMessage("\n" + strStyleCloned + " dbErased.");
        }
    }
    else
    {
        reportMessage("\n" + strStyleCloned + " not valid!");
    }
}

String strCloneEmptyStyle = T("|dbCreate empty cloned style|");
addRecalcTrigger(_kContext, strCloneEmptyStyle );
if (_bOnRecalc && _kExecuteKey==strCloneEmptyStyle)
{
    FastenerAssemblyDef defOrig(strStyleOrig);
    FastenerAssemblyDef defCloned(strStyleCloned);
    if (!defCloned.bIsValid() && defOrig.bIsValid())
    {
        defCloned.dbCreate();
        if (defCloned.bIsValid())
        {
            reportMessage("\n"+strStyleOrig +" cloned into " +
strStyleCloned);
        }
        else
        {
            reportMessage("\nFailed to clone "+strStyleOrig +" into "
+ strStyleCloned);
        }
    }
    else
    {
        reportMessage("\n" + strStyleCloned + " not created!");
    }
}

String strSetClonedStyle = T("|set cloned style data|");
addRecalcTrigger(_kContext, strSetClonedStyle );
if (_bOnRecalc && _kExecuteKey==strSetClonedStyle)
{
    FastenerAssemblyDef defOrig(strStyleOrig);
}

```

```

FastenerAssemblyDef defCloned(strStyleCloned);
if (defCloned.bIsValid() && defOrig.bIsValid())
{
    reportMessage("\n"+strStyleOrig +" data copied into " +
strStyleCloned);

    defCloned.setDescription(defOrig.description());
    defCloned.setListComponent(defOrig.listComponent());
    defCloned.setHeadComponents(defOrig.headComponents());
    defCloned.setTailComponents(defOrig.tailComponents());
}
else
{
    reportMessage("\n" + strStyleOrig + " or " + strStyleCloned
+ " not valid!");
}
}

```

*—end example]*

#### 6.5.5.1 FastenerComponentData

The FastenerAssemblyDef is build using FastenerComponentData.

---

```

class FastenerComponentData { // added in hsbCAD2012, build 17.0.21.

    FastenerComponentData();

    String name() const;
    void setName(String str);
    String type() const;
    void setType(String str);
    String subType() const;
    void setSubtype(String str);

    String manufacturer() const;
    void setManufacturer(String str);
    String model() const;
    void setModel(String str);
    String material() const;
    void setMaterial(String str);
    String category() const;
    void setCategory(String str);
    String group() const;
    void setGroup(String str);
    String coating() const;
    void setCoating(String str);
    String grade() const;
    void setGrade(String str);
    String norm() const;
}

```

---

```
void setNorm(String str);

double stackThickness() const;
void setStackThickness(double dVal);
double sinkDiameter() const;
void setSinkDiameter(double dVal);
double mainDiameter() const;
void setMainDiameter(double dVal);

};
```

---

#### 6.5.5.2 FastenerArticleData

The FastenerAssemblyDef is build using FastenerArticleData.

---

```
class FastenerArticleData { // added in hsbCAD2012, build 17.0.21.

    FastenerArticleData();

    String articleNumber() const;
    void setArticleNumber(String str);
    String description() const;
    void setDescription(String str);
    String notes() const;
    void setNotes(String str);

    double fastenerLength() const;
    void setFastenerLength(double dVal);
    double threadLength() const;
    void setThreadLength(double dVal);

    // projection length is the remaining free length of the bolt with nut assembled
    double minProjectionLength() const;
    void setMinProjectionLength(double dVal);
    double maxProjectionLength() const;
    void setMaxProjectionLength(double dVal);

    int hasLengthInfo() const;
    void setHasLengthInfo(int nSet);
    int isBespoke() const;
    void setIsBespoke(int nSet);

    Map map() const;
    void setMap(Map mapNew);
```

```
};
```

#### 6.5.5.3 FastenerSimpleComponent

The FastenerAssemblyDef is build using FastenerSimpleComponent. A simple component represents one fastener item eg one specific nut, or one specific bolt or washer. The article number and description is stored in its articleData, while the type, name,... is stored in the componentData.

---

```
class FastenerSimpleComponent { // added in hsbCAD2012, build 17.0.21.

    FastenerSimpleComponent();

    FastenerComponentData componentData() const; // see FastenerComponentData
    void setComponentData(FastenerComponentData compdata);

    FastenerArticleData articleData() const; // see FastenerArticleData
    void setArticleData(FastenerArticleData articleData);

};
```

#### 6.5.5.4 FastenerListComponent

The FastenerAssemblyDef is build using FastenerListComponent. A list component represents a set of fastener items which shares the same component data, but have different lengths. eg one set of bolts that differ only by length. The article number and description is stored in its articleData for each length, while the type, name,... is stored in the common componentData.

---

```
class FastenerListComponent { // added in hsbCAD2012, build 17.0.21.

    FastenerListComponent();

    FastenerComponentData componentData() const; // see FastenerComponentData
    void setComponentData(FastenerComponentData compdata);

    FastenerArticleData[] articleDataSet() const; // see FastenerArticleData
    void setArticleDataSet(FastenerArticleData[] arArticleData);

};
```

## 6.5.6 GroupStates

The GroupStates, which is derived from the [DictObject](#), refers to a collection of hsbConsole group state information.

An DictObject can be casted into an GroupStates. However when the casting is not allowed, the resulting GroupStates will become invalid. This can be checked with the bIsValid() function of DictObject.

Because GroupStates is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

---

**class GroupStates : DictObject { // added since 26.3.35. See [DictObject](#) for base functions.**

```
    GroupStates(String strEntry); // constructor

    void dbCreate(); // automatically stores the current visibility state of all the groups in the
                    console into a newly created GroupStates object
    void restore(); // applies the content which is stored in this GroupStates, to the
                    hsbConsole.

    static GroupStates[] getAllEntries();
    static String[] getAllEntryNames();
};
```

---

[Example O-type TSL:

```
GroupStates cs; // the default constructor creates a straight one

// collect list of curved styles
GroupStates csList[] = GroupStates().getAllEntries();
reportNotice("\n ======");
for (int e=0; e<csList.length(); e++) {
    reportNotice("\n entryName: "+csList[e].entryName());
}

// declare a property with the names of the curved styles
PropString pS(0, GroupStates().getAllEntryNames(), T("|Current stored
group states|"));

GroupStates csInvest(pS); // constructor with a curved style name
reportNotice("\n ----");
reportNotice("\n entryName: "+csInvest.entryName());
reportNotice("\n isValid: "+csInvest.bIsValid());
reportNotice("\n bIsSpecial: "+csInvest.bIsSpecial());
reportNotice("\n handle: "+csInvest.handle());
reportNotice("\n typeName: "+csInvest.typeName());
```

---

—end example]

### 6.5.7 MapObject

The MapObject, which is derived from the [DictObject](#), refers to an entry in an own defined dictionary.

An DictObject can be casted into an MapObject. However when the casting is not allowed, the resulting MapObject will become invalid. This can be checked with the `blsValid()` function of DictObject. Because MapObject is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

If a TslInst is dependent on the contents of a DictObject, the Tsl script should contain a call:

```
setDependencyOnDictObject(DictObject object); // see Global functions and Execution
```

A MapObject is in fact a reference to a dictionary object inside an Autocad Named Object Dictionary. The reference is established by the entry name in the dictionary. The entry name and dictionary name are set at time of the MapObject constructor. The advantage of this approach is that the TSL author can use the readable entry name as reference to the object. But there is also a disadvantage to this approach in that the name of the entry in the dictionary could change. There is no mechanism to notify the Tsl that a particular entry has changed its name.

Consider the case where a Tsl has a enumerated property with the available entries in a dictionary. Whatever changes are done to the dictionary, the list of available entries is likely to change. To the one that changes the dictionary, must trigger the appropriate Tsl's for recalculation. Since such a recalculation will take place at the command ended, not more then one type of dictionary action should be performed before the command ends.

Beware, if an dictionary entry is renamed, and some other entries are created or deleted, it is impossible for existing Tsl instances to recover their correct dictionary entry.

---

```
class MapObject : DictObject { // see DictObject for base functions

    MapObject(String strDictionaryName, String strEntryName); // constructor

    String dictionaryName() const;

    Map map() const; // retrieve the Map stored inside the MapObject
    void setMap(Map mapNew); // overwrite the Map inside the MapObject
```

---

```

void dbCreate(Map mapNew); // append a new MapObject to the Acad database. The
    opposite dbErase is a member function of the DictObject.

int dbRename(String strEntryNew); // rename the entry inside the Acad database
    dictionary.

static MapObject[] getAllEntries(String strDictionaryName);
static String[] getAllEntryNames(String strDictionaryName);

static void recalcAllReferencesToDictionary(String strDictionaryName);

// added in hsbCAD20.1.69 and hsbCAD21.0.43
String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
    Only use on special events.
String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
    code as string. Empty string means all ok, see example below.
String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
    code as string. Empty string means all ok, see example below.

};


```

---

*[Example O-type TSL and automatic insert: This one uses a MapObject which contains a text, and a text height. That text is displayed by this Tsl.*

```

String strDictionary = "MYDICT";

// make a property with the list of available styles
String arEntries[] = MapObject() .getAllEntryNames(strDictionary);
PropString pStyle(0,arEntries,"Style to use");

// create a new object
MapObject mo(strDictionary ,pStyle); // will do lookup automatically
if (!mo.bIsValid()) {
    reportNotice("\nPlease select a valid style.");
    return;
}

// make sure the style is protected, and that this Tsl is updated
// when the style changes.
setDependencyOnDictObject(mo);

// now use the style
Map moMap = mo.map();
String strText = moMap.getString("text");
double dHeight = moMap.getDouble("height");

if (strText == "") strText = "no key";

```

---

```

if (dHeight==0) dHeight = U(10, "mm");

Display dp(-1);
dp.textHeight(dHeight);
dp.draw(strText ,_Pt0,_XU,_YU,0,0);

—end example]

```

---

*[Example O-type TSL, with insert done in script. It contains a simple Style manager to work with a style that can be used by the Tsl above.*

```

Unit(1, "mm");
String strDictionary = "MYDICT";
int bDataBaseChanged = FALSE;

if (_bOnInsert) {

    // === phase 1 ===

    String arAction[]={"New", "Edit", "Copy", "Rename", "Delete"};
    PropString pAction(0,arAction,"Action");

    // make a property with the list of available styles
    String arEntries[] = MapObject().getAllEntryNames(strDictionary);
    if (arEntries.length()==0) arEntries.append("-- No entries
available --");
    PropString pStyle(2,arEntries,"Style to modify");

    setCatalogFromPropValues("Step1"); // will save the default
values
    showDialog("Step1"); // use the default values, not the previous
entered values.

    // if the showDialog appears a second time later on, these
properties are not suppose to be changed.
    pAction.setReadOnly(TRUE);
    pStyle.setReadOnly(TRUE);

    // === phase 2 ===

    // Action New
    if (pAction=="New") {

        // read the map from the selected style, to use as default for
the new style
        String strText = "a new text";

```

---

```

        double dHeight = U(20);
        MapObject mol(strDictionary ,pStyle); // will lookup
automatically
        if (mol.bIsValid()) {
            Map mp1 = mol.map();
            strText = mp1.getString("text");
            dHeight = mp1.getDouble("height");
        }

        PropString pNewStyle(1,"","New style name");
        pNewStyle.setDescription("If new style name is given, a new
style will be created.");

        PropString pText(3, strText , "Text to show");
        PropDouble pHeight(0, dHeight , "Height of the text");
        setCatalogFromPropValues("Step2"); // will save the default
values
        showDialog("Step2"); // use the default values, not the
previous entered values.

        // create a new object
        String strEntry = pNewStyle;
        MapObject mo(strDictionary ,strEntry); // will lookup
automatically
        if (!mo.bIsValid()) { // if object is found, bIsValid is TRUE
            // add the properties to a map, and the map to the MapObject
            Map map;
            map.setDouble("height",pHeight);
            map.setString("text",pText);
            mo.dbCreate(map); // make sure the dictionary and entry name
are set in the constructor before calling dbCreate
            bDataBaseChanged = TRUE;
        }
        else {
            // The style already exists.
            reportNotice("\nUnable to create new MapObject.
"+mo.entryName()+" in dictionary " +mo.dictionaryName() + " already
exists.");
        }

        if (!mo.bIsValid()) { // should not happen.
            reportNotice("\nSomething is wrong with MapObject name
"+mo.entryName()+" for dictionary " +mo.dictionaryName());
        }
    }

    // Action Edit
    if (pAction=="Edit") {

```

```

// read the map from the selected style
String strText = "a new text";
double dHeight = U(20);
MapObject mo(strDictionary ,pStyle); // will lookup
automatically
if (mo.bIsValid()) {

    Map map= mo.map();
    strText = map.getString("text");
    dHeight = map.getDouble("height");

    PropString pText(3, strText , "Text to show");
    PropDouble pHeight(0, dHeight , "Height of the text");
    setCatalogFromPropValues("Step2"); // will save the default
values
    showDialog("Step2"); // use the default values, not the
previous entered values.

    map.setDouble("height",pHeight);
    map.setString("text",pText);
    mo.setMap(map);

}
else {
    reportNotice("\nUnable to edit MapObject. "+mo.entryName()+""
in dictionary " +mo.dictionaryName() + ".");
}
}

// Action Copy
if (pAction=="Copy") {

    // read the map from the selected style, to use for the new
style
    Map mapOld;
    MapObject mol(strDictionary ,pStyle); // will lookup
automatically
    if (mol.bIsValid()) {
        mapOld = mol.map(); // copy map
    }

    PropString pNewStyle(1,"","New style name");
    pNewStyle.setDescription("If new style name is given, a new
style will be created.");

    setCatalogFromPropValues("Step2"); // will save the default
values
    showDialog("Step2"); // use the default values, not the
previous entered values.
}

```

```

        // create a new object
        String strEntry = pNewStyle;
        MapObject mo(strDictionary ,strEntry); // will lookup
automatically
        if (!mo.bIsValid()) { // if object is found, bIsValid is TRUE
            // add the properties to a map, and the map to the MapObject
            mo.dbCreate(mapOld); // make sure the dictionary and entry
name are set in the constructor before calling dbCreate
            bDataBaseChanged = TRUE;
        }
        else {
            // The style already exists.
            reportNotice("\nUnable to create new MapObject.
"+mo.entryName()+" in dictionary " +mo.dictionaryName() + " already
exists.");
        }

        if (!mo.bIsValid()) { // should not happen.
            reportNotice("\nSomething is wrong with MapObject name
"+mo.entryName()+" for dictionary " +mo.dictionaryName());
        }
    }

    // Action Rename
    if (pAction=="Rename") {

        PropString pNewStyle(1,"","New style name");
        pNewStyle.setDescription("If new style name is given, the style
will be renamed.");

        setCatalogFromPropValues("Step2"); // will save the default
values
        showDialog("Step2"); // use the default values, not the
previous entered values.

        // create a new object
        int bOk = FALSE;
        MapObject mo(strDictionary ,pStyle); // will lookup
automatically
        if (mo.bIsValid()) { // if object is found, bIsValid is TRUE
            mo.dbRename(pNewStyle);
            if (mo.entryName()==pNewStyle) bOk = TRUE;
            bDataBaseChanged = TRUE;
        }
        if (!bOk) {
            reportNotice("\nUnable to rename new MapObject.
"+mo.entryName()+" in dictionary " +mo.dictionaryName() + " to " +
pNewStyle);
        }
    }
}

```

```

}

// Action Delete
if (pAction=="Delete") {

    MapObject mo(strDictionary ,pStyle); // will lookup
    automatically
    if (mo.bIsValid()) { // if object is found, bIsValid is TRUE
        Entity ents[] = mo.getReferencesToMe(); // check if there is
        anybody referring to this style
        if (ents.length()==0) {
            mo.dbErase(); // erase from the database
            reportNotice("\nMapObject "+mo.entryName()+" in
            dictionary " +mo.dictionaryName() + " erased.");
            bDataBaseChanged = TRUE;
        }
        else {
            // when there are still entities referring to the style,
            it should not be erased.
            reportNotice("\nMapObject "+mo.entryName()+" in
            dictionary " +mo.dictionaryName() + " could not be erased because it
            has " +
            ents.length() + " references.");
        }
    }

    if (bDataBaseChanged) {
        // When a style is erased, added,..., the list of styles is
        changed, so all Tsl's that do refer to a style, have most likely a
        combo property with a list of styles.
        // So these need to be triggered for recalculation.
        MapObject().recalcAllReferencesToDictionary(strDictionary);
    }

    eraseInstance();
    return;
}

```

*—end example]*

[Example O-type TSL illustrating getAllEntryNamesFromDwg and importFromDwg:

```

Unit(1,"mm");
String strDictionary = "MYDICT";
String strEntry = "temp";

```

```

if (_bOnInsert) {

    String arListCur[] = MapObject().getAllEntryNames(strDictionary);
    PropString pListCur(0,arListCur,"Current list");

    String strDwg = _kPathDwg + "\\Source.dwg";
    PropString pDwg(1, strDwg, "Dwg file to read from");
    showDialog(T("|_Default|"));
    strDwg = pDwg;
    pDwg.setReadOnly(TRUE);

    String arList[] = MapObject(strDictionary,
strEntry).getAllEntryNamesFromDwg(strDwg);
    PropString pListFromDwg(2,arList, T("list from dwg"));

    PropString pEntry1(3,"",T("|Entry 1 to import|"));
    PropString pEntry2(4,"",T("|Entry 2 to import|"));
    PropInt pOverwrite(5,0,T("|Overwrite|"));
    showDialog();

    String arToImport[0];
    if (pEntry1 != "") arToImport.append(pEntry1);
    if (pEntry2 != "") arToImport.append(pEntry2);
    int bOverwrite = pOverwrite;

    String strErr;
    if (arToImport.length() == 1)
        strErr = MapObject(strDictionary,
strEntry).importFromDwg(strDwg, arToImport[0], bOverwrite);
    else
        strErr = MapObject(strDictionary,
strEntry).importFromDwg(strDwg, arToImport, bOverwrite);

    if (strErr != "")
    {
        String strToLoad;
        for(int r=0; r<arToImport.length(); r++)
            strToLoad += " " + arToImport[r] + "!";
        reportWarning("Could not load some of" + strToLoad + " from "
+ strDwg + ": " + strErr);
    }

    String arListCurNew[] =
MapObject().getAllEntryNames(strDictionary);
    PropString pListCurNew(6,arListCurNew,"New list");
    showDialog(T("|_Default|"));

    eraseInstance();
    return;
}

```

—end example]

### 6.5.8 MasterPanelStyle

The MasterPanelStyle, which is derived from the [DictObject](#), refers to a [MasterPanel](#) style description.

An DictObject can be casted into an MasterPanelStyle. However when the casting is not allowed, the resulting MasterPanelStyle will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because MasterPanelStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The MasterPanelStyle from a MasterPanel can be constructed with the string returned by [MasterPanel::style\(\)](#):

```
class MasterPanelStyle : DictObject { // see DictObject for base functions (added
hsbCAD14.0.68)

    MasterPanelStyle(String strEntry); // constructor

    void dbCreate(double dThickness); // (added from hsbCAD 22.1.6 and 21.4.81)

    double dThickness() const;
    void setThickness(double dVal); // (added from hsbCAD 22.1.6 and 21.4.81)

    static MasterPanelStyle[] getAllEntries();
    static String[] getAllEntryNames();

    String surfaceQualityTop() const; // see SurfaceQualityStyle (added hsbCAD17.1.2)
    String surfaceQualityBottom() const; // example found in MasterPanel

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
    Only use on special events.
    String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
    code as string. Empty string means all ok, see SipStyle example.
    String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
    code as string. Empty string means all ok, see SipStyle example.
};
```

[Example O-type TSL:

```
Unit(1, "mm");

if (_bOnInsert) {
```

```

    // select the Panel and insertion point
    MasterPanel sp = getMasterPanel();
    _Entity.append(sp);
    _Pt0 = getPoint();

    String strStyle = sp.style(); // get the style from the selected
MasterPanel

    String strAllStyles[] = MasterPanelStyle().getAllEntryNames(); // 
list of all available MasterPanelStyles
    PropString pStyle(0, strAllStyles, "MasterPanel style",
strAllStyles.find(strStyle)); // make property
    showDialog(); // allow the user to change the style

    // change the MasterPanelStyle of the MasterPanel, to match the
property value
    sp.setStyle(pStyle);

    return;
}

if (_Entity.length()==0) return;
MasterPanel sp = (MasterPanel) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strDbCreate = T("|Create a MasterPanelStyle|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    String strNewName = getString("new name:");
    MasterPanelStyle anew(strNewName);
    String strError = anew.dbCreate(U(100));
    if (strError == "")
    {
        sp.setStyle(anew.entryName());
        reportMessage("\\nMasterPanelStyle created with name:
"+anew.entryName());
    }
    else
    {
        reportMessage("\\nCould not create MasterPanelStyle with name:
"+strNewName);
        reportMessage("\\n"+strError );
    }
}

String strStyle = sp.style();
PropString pStyle(0, strStyle , "MasterPanel style"); // make
property

```

```

pStyle.set(strStyle);
pStyle.setReadOnly(TRUE);

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

// compose a MasterPanelStyle from the String
MasterPanelStyle masterPanelStyle(strStyle);

String strLines[0];
strLines.append("dThickness: "+masterPanelStyle.dThickness());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO ,_XU, _YU, 1,1);
}

```

*—end example]*

## 6.5.9 MultiPageStyle

The MultiPageStyle, which is derived from the [DictObject](#), refers to a [MultiPage](#) entity its style description.

An DictObject can be casted into an MultiPageStyle. However when the casting is not allowed, the resulting MultiPageStyle will become invalid. This can be checked with the blsValid() function of DictObject.

Because MultiPageStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

Each MultiPageStyle has an objectCollectiontype. The type should be one of the following

[\\_kOCTIndividualShopDrawing](#), [\\_kOCTModuleElementDrawing](#), [\\_kOCTGroup](#),  
[\\_kOCTSubassembly](#), [\\_kOCTTrussEntity](#), [\\_kOCTCollectionEntity](#), [\\_kOCTElement](#),  
[\\_kOCTMetalPartCollectionEnt](#), [\\_kOCTTslDefined](#), [\\_kOCTMasterPanel](#), [\\_kOCTChildPanel](#),  
[\\_kOCTMassElement](#)

```

class MultiPageStyle : DictObject { // see DictObject for base functions (added
hsbCAD17.0.47)

    MultiPageStyle(String strEntry); // constructor

    int objectCollectionType() const;

    static MultiPageStyle[] getAllEntries();
    static String[] getAllEntryNames();

```

```

static String[] getAllEntryNames(int nObjectCollectionType);

// added in hsbCAD20.1.69 and hsbCAD21.0.43
String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
    Only use on special events.
String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
    code as string. Empty string means all ok, see SipStyle example.
String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
    code as string. Empty string means all ok, see SipStyle example.

String[] getListOfStereotypeOverrides(); // (added 22.1.107 and 23.0.75)
String[] getListOfStereotypeOverridesChainDim(); // (added 22.1.107 and 23.0.75)
String[] getListOfStereotypeOverridesHatch(); // (added 22.1.107 and 23.0.75)
};

```

---

[Example O-type TSL:

```

Unit(1, "mm");

int arOCT[] = {_kOCTIndividualShopDrawing, _kOCTModuleElementDrawing,
_kOCTGroup,
_kOCTSubassembly, _kOCTTrussEntity, _kOCTCollectionEntity,
_kOCTElement,
_kOCTMetalPartCollectionEnt, _kOCTTslDefined, _kOCTMasterPanel,
_kOCTChildPanel, _kOCTMassElement};

String arOCTStr[] = {"_kOCTIndividualShopDrawing",
"_kOCTModuleElementDrawing", "_kOCTGroup",
"_kOCTSubassembly", "_kOCTTrussEntity", "_kOCTCollectionEntity",
"_kOCTElement",
"_kOCTMetalPartCollectionEnt", "_kOCTTslDefined",
"_kOCTMasterPanel", "_kOCTChildPanel", "_kOCTMassElement"};

PropString pStyle(0, arOCTStr, "Shopdraw type");
PropString pDimStyle(1, DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

int nOCT = arOCT[arOCTStr.find(pStyle, 0)];
String strAllStyles[] = MultiPageStyle().getAllEntryNames(nOCT ); // 
list of all available MultiPageStyles

String strLines[0];
strLines.append(strAllStyles.length() + " styles found of type
"+pStyle);
for (int s=0; s<strAllStyles.length(); s++) {
    MultiPageStyle mps(strAllStyles[s]);
    strLines.append("strStyle: "+mps.entryName() + " oct:
"+mps.objectCollectionType() );
}

```

---

```

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

—end example]

```

### 6.5.10 MvBlockDef

The MvBlockDef, which is derived from the [DictObject](#), refers to a multi view block definition in the style manager.

An DictObject can be casted into an MvBlockDef. However when the casting is not allowed, the resulting MvBlockDef will become invalid. This can be checked with the blsValid() function of DictObject.

Because MvBlockDef is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The MvBlockDef from [MvBlockRef](#) can be constructed with the string returned by  
**MvBlockRef::definition();**

```

class MvBlockDef : DictObject { // see DictObject for base functions (added hsbCAD2011 build
16.0.21)

    MvBlockDef(String strEntry); // constructor

    static MvBlockDef[] getAllEntries();
    static String[] getAllEntryNames();

    static String[] getAllBlocksReferenced();

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
        Only use on special events.
    String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
        code as string. Empty string means all ok, see SipStyle example.
    String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
        code as string. Empty string means all ok, see SipStyle example.

};

```

*[Example O-type TSL:*

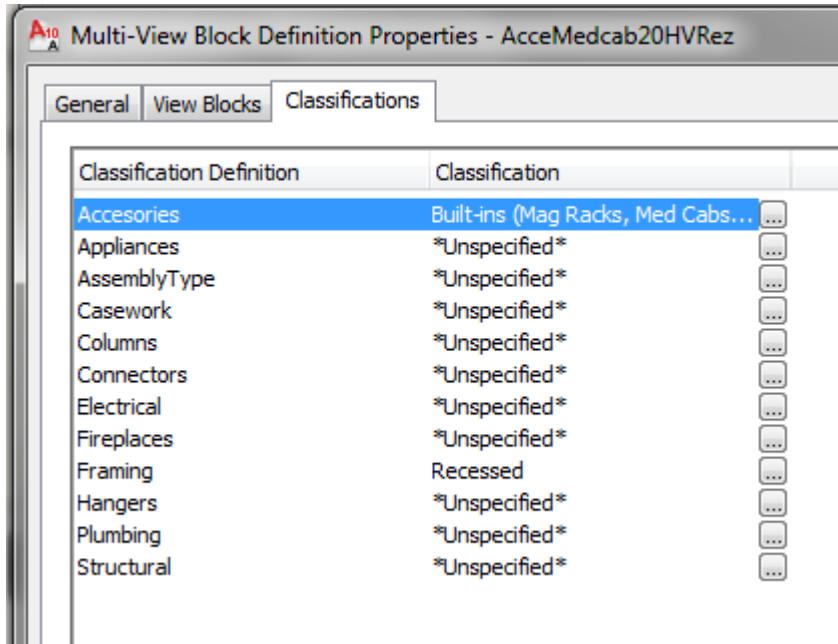
```

==== List of MvBlock entries ====
entryName: Aec8_Room_Tag
entryName: CaseBaseDr27W30H21DVRez
entryName: CaseWall33W12H12DVRez
entryName: AcceMedcab20HVRez
entryName: GeneralTxtASVRez
entryName: CaseCornerBaseSquareER42W36H42DVRez
entryName: PlumbWcTankStandardVRez
entryName: Refrig Top-Bot 31x28VRez
entryName: PlumbIceMkrConnVRez_P
entryName: ElecPowerDuplexRecptASVRez_P
entryName: ElecPowerCountertopRecptVRez_M
entryName: CaseCornerWall24W24H24DVRez
entryName: FireplPreFab36VRez
entryName: Hanger_HGUS3.25-10_SST
entryName: mvcircle
-----
entryName: AcceMedcab20HVRez
isValid: 1
blsSpecial: 0
handle: 5053
typeName: AecDbMvBlockDef
==== references the following blocks ===
blockName: VRezAcceMedcab20H_P
blockName: VRezAcceMedcab20H_F
blockName: VRezAcceMedcab20H_L
blockName: VRezAcceMedcab20H_R
blockName: VRezAcceMedcab20H_M
blockName: VRezAcceMedcab20H_MI
-- 2 classifications found
0) Accessories: Built-ins
1) Framing: Recessed

```

#### A<sub>10</sub> Multi-View Block Definition Properties - AcceMedcab20HVRez

General	View Blocks	Classifications
<p><b>Display Representations</b></p> <ul style="list-style-type: none"> <li><input checked="" type="checkbox"/> General</li> <li><input checked="" type="checkbox"/> Model</li> <li><input type="checkbox"/> Model to GenElev</li> <li><input type="checkbox"/> Plan</li> <li><input type="checkbox"/> Plan High Detail</li> <li><input type="checkbox"/> Plan Low Detail</li> <li><input type="checkbox"/> Plan Screened</li> <li><input type="checkbox"/> Reflected</li> <li><input type="checkbox"/> Render</li> </ul>	<p><b>View Blocks</b></p> <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"> <input checked="" type="checkbox"/> VRezAcceMedcab20H_M  <input checked="" type="checkbox"/> VRezAcceMedcab20H_MI         </div> <p style="text-align: center;"><b>Add...</b>    <b>Remove</b></p>	<p><b>View Directions</b></p> <p><input checked="" type="checkbox"/> Top  <input checked="" type="checkbox"/> Bottom  <input checked="" type="checkbox"/> Front  <input checked="" type="checkbox"/> Back  <input checked="" type="checkbox"/> Left  <input checked="" type="checkbox"/> Right  <input checked="" type="checkbox"/> Other</p>
<p><b>Set Interference Block...</b>    *NONE*</p>		



```

MvBlockDef mvdList[] = MvBlockDef().getAllEntries();
reportNotice("\n === List of MvBlock entries ===");
for (int e=0; e<mvdList.length(); e++) {
    reportNotice("\n entryName: "+mvdList[e].entryName());
}

// declare a property with the names of the entries
PropString pS(0, MvBlockDef().getAllEntryNames(), T("|MvBlockDef
name|"));

MvBlockDef mvdInvest(pS); // constructor with a MvBlockDef name
reportNotice("\n ----");
reportNotice("\n entryName: "+mvdInvest.entryName());
reportNotice("\n isValid: "+mvdInvest.bIsValid());
reportNotice("\n bIsSpecial: "+mvdInvest.bIsSpecial());
reportNotice("\n handle: "+mvdInvest.handle());
reportNotice("\n typeName: "+mvdInvest.typeName());

// The MvBlockDef refers to some Blocks.
String arStr[] = mvdInvest.getAllBlocksReferenced();
reportNotice("\n === references the following blocks ===");
for (int b=0; b<arStr.length(); b++) {
    reportNotice("\n\t blockName: "+arStr[b]);
}

// Since the MvBlockDef is an ACA style, there could be a
// classification associated with it.
Map mapEnt = mvdInvest.getClassificationMap();

reportNotice("\n--- "+mapEnt.length()+" classifications found");

```

```

for (int i=0; i<mapEnt.length(); i++) {
    String strKey = mapEnt.keyAt(i);
    if (mapEnt.hasString(i))
        reportNotice("\n"+i+": "+strKey+": "+mapEnt.getString(i));
}

```

—end example]

[Example O-type TSL:

```

U(1, "mm");
if (_bOnInsert) {

    // loop over all MvBlockDef's and insert for each of them a
    // MvBlockRef in the drawing.

    Point3d ptO = getPoint("Insertion point");

    PropDouble dOffset(0,U(100), "spacing");
    PropInt pPerRow(0,10, "number of items per row");
    showDialog();

    MvBlockDef mvdList[] = MvBlockDef().getAllEntries();

    for (int e=0; e<mvdList.length(); e++) {

        int nPerRow = pPerRow;
        int nX = e/nPerRow ;
        int nY = e%nPerRow ;
        Point3d ptInsert = ptO + nX*dOffset*_XU + nY*dOffset*_YU;

        CoordSys cs(ptInsert, _XU, _YU, _ZU);

        MvBlockRef mvBlockRef;
        String strDef = mvdList[e].entryName();
        mvBlockRef.dbCreate(cs, 1, 1, 1, strDef);
    }

    eraseInstance();
    return;
}

```

—end example]

[Example O-type TSL illustrating reading attached property sets to MvBlockDef:

```

    === Firing: MvBlockDef
--- 16 MvBlockDef's
MvBlockDef: Aec8_Room_Tag
MvBlockDef: CaseBaseDr27W30H21DVRez
MvBlockDef: CaseWall133W12H12DVRez
MvBlockDef: AcceMedcab20HVRez
MvBlockDef: GeneralTxt-ASVRez
MvBlockDef: CaseCornerBaseSquareER42W36H42DVRez
MvBlockDef: PlumbWcTankStandardVRez
MvBlockDef: Refrig Top-Bot 31x28VRez
MvBlockDef: PlumbIceMkrConnVRez_P
MvBlockDef: ElecPowerDuplexRecptASVRez_P
MvBlockDef: ElecPowerCountertopRecptVRez_M
MvBlockDef: CaseCornerWall24W24H24DVRez
MvBlockDef: FireplPreFab36VRez
MvBlockDef: FireplClassic36VRez
MvBlockDef: Hanger_HGUS3.25-10_SST
MvBlockDef: mvcircle
--- Investigating FireplPreFab36VRez
isValid: 1
bIsSpecial: 0
handle: 52B3
typeName: AecDbMvBlockDef
--- 6 Block references
blockName: VRezFireplPreFab36_P
blockName: VRezFireplPreFab36_F
blockName: VRezFireplPreFab36_L
blockName: VRezFireplPreFab36_R
blockName: VRezFireplPreFab36_M
blockName: VRezFireplPreFab36_MI
--- 1 classifications found
0) Fireplaces: Fireplaces
--- 2 attachedPropSetName
name: Areas
name: VRezUsageStyle
investigating property set: Areas
--- 0 properties found
investigating property set: VRezUsageStyle
--- 5 properties found
0.S) Usage: use it
1.S) ItemConfig: config it
2.S) ItemCategory: categorize it
3.S) ItemName: name it
4.S) ItemSize: size it

reportMessage("\n\n === Firing: "+scriptName());

// collect list
MvBlockDef mvList[] = MvBlockDef().getAllEntries();
reportMessage("\n --- "+mvList.length()+" MvBlockDef's");
for (int e=0; e<mvList.length(); e++) {
    reportMessage("\n MvBlockDef: "+mvList[e].entryName());
}

```

```

}

// declare a property with the names of the entries
PropString pS(0, MvBlockDef().getAllEntryNames(), T(" |MvBlockDef
name|"));

MvBlockDef mvdInvest(pS); // constructor with a MvBlockDef name
reportMessage("\n --- Investigating "+mvdInvest.entryName());
reportMessage("\n isValid: "+mvdInvest.bIsValid());
reportMessage("\n bIsSpecial: "+mvdInvest.bIsSpecial());
reportMessage("\n handle: "+mvdInvest.handle());
reportMessage("\n typeName: "+mvdInvest.typeName());

// The MvBlockDef refers to some Blocks.
String arStr[] = mvdInvest.getAllBlocksReferenced();
reportMessage("\n --- "+arStr.length()+" Block references");
for (int b=0; b<arStr.length(); b++) {
    reportMessage("\n blockName: "+arStr[b]);
}

// Since the MvBlockDef is an ACA style, there could be a
classification associated with it.
Map mapEnt = mvdInvest.getClassificationMap();

reportMessage("\n --- "+mapEnt.length()+" classifications found");
for (int i=0; i<mapEnt.length(); i++) {
    String strKey = mapEnt.keyAt(i);
    if (mapEnt.hasString(i))
        reportMessage("\n"+i+": "+strKey+": "+mapEnt.getString(i));
}

String arNames[] = mvdInvest.attachedPropSetName();
reportMessage("\n --- "+arNames.length()+" attachedPropSetName");
for (int n=0; n<arNames.length(); n++) {
    reportMessage("\n name: "+arNames[n]);
}

for (int n=0; n<arNames.length(); n++) {
    String strName = arNames[n];
    Map mapProp = mvdInvest.getAttachedPropSetMap(strName);

    reportMessage("\n investigating property set: "+strName);
    reportMessage("\n --- "+mapProp.length()+" properties found");
    for (int i=0; i<mapProp.length(); i++) {
        String strKey = mapProp.keyAt(i);
        if (mapProp.hasInt(i))
            reportMessage("\n"+i+".I) "+strKey+": "+mapProp.getInt(i));
        else if (mapProp.hasDouble(i))
            reportMessage("\n"+i+".D) "+strKey+": "+mapProp.getDouble(i));
        else if (mapProp.hasString(i))
    }
}

```

```

        reportMessage("\n"+i+".S) "+strKey+":
        "+mapProp.getString(i));
    }
}

—end example]

```

### 6.5.11 PainterDefinition

The PainterDefinition, which is derived from the [DictObject](#), refers to the entries in the Painter tab in the hsb\_Console.

A DictObject can be casted into an PainterDefinition. However when the casting is not allowed, the resulting PainterDefinition will become invalid. This can be checked with the blsValid() function of DictObject.

Because PainterDefinition is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The [Entity](#) has methods like **formatObject** and **acceptObject** which accept a format and a filter strings respectively.

---

```

class PainterDefinition : DictObject { // see DictObject for base functions (added
hsbCAD23.0.42)

PainterDefinition(String strEntry); // constructor

void dbCreate();

String name() const;
String type() const;
String filter() const;
String format() const;
String formatToResolve() const; // string that is used in the formatObject call

// setter methods can only be called on database resident objects
void setName(String str);
void setType(String str);
void setFilter(String str);
void setFormat(String str);

static PainterDefinition[] getAllEntries();
static String[] getAllEntryNames();

String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
Only use on special events.
String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
code as string. Empty string means all ok, see SipStyle example.
String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
code as string. Empty string means all ok, see SipStyle example.

```

```
<Entity>[] filterAcceptedEntities(<Entity>[] arEntities); // (added 23.0.75) return the list of
entities that are accepted by the filter
};
```

---

[Example O-type TSL:

```
Unit(1, "mm");

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

String strAllStyles[] = PainterDefinition().getAllEntryNames(); // list of all available PainterDefinitions

String strChange = T("|Change Painter|");
addRecalcTrigger(_kContext, strChange );
if (_bOnRecalc && _kExecuteKey==strChange && strAllStyles.length() > 0)
{
    PainterDefinition mps(strAllStyles[0]);
    mps.setFilter("!isDummy");
    mps.setFormat("@(Width)");
    mps.setName("Beam width not dummy");
    mps.setType("Beam");
}

PainterDefinition defs[] = PainterDefinition().getAllEntries();

String strDelete = T("|Delete Painter|");
addRecalcTrigger(_kContext, strDelete );
if (_bOnRecalc && _kExecuteKey==strDelete && defs.length() > 0)
{
    defs[0].dbErase();
}

String strCreate = T("|Create Painter|");
addRecalcTrigger(_kContext, strCreate );
if (_bOnRecalc && _kExecuteKey==strCreate)
{
    PainterDefinition mps("Beam is dummy");
    mps.dbCreate();
    mps.setFilter("isDummy");
    mps.setFormat("@(isDummy)");
    mps.setType("Beam");
}

defs = PainterDefinition().getAllEntries();
```

---

```

String strLines[0];
strLines.append(defs.length() + " styles found");
for (int s=0; s<defs.length(); s++)
{
    strLines.append("strStyle: "+defs[s].entryName());
    strLines.append(" name: "+defs[s].name());
    strLines.append(" type: "+defs[s].type());
    strLines.append(" filter: "+defs[s].filter());
    strLines.append(" format: "+defs[s].format());
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vec0 = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vec0, _XU, _YU, 1,1);
}

```

*—end example]*

## 6.5.12 RenderMaterial

The RenderMaterial, which is derived from the [DictObject](#), refers to a material that can be assigned to an autocad entity, but also to a hsbCAD beam.

A DictObject can be casted into an RenderMaterial. However when the casting is not allowed, the resulting RenderMaterial will become invalid. This can be checked with the `bIsValid()` function of DictObject.

Because RenderMaterial is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The [Entity](#) has a render material.

```

String Entity::renderMaterial() const; // see example below (added build 18.1.32)
void Entity::setRenderMaterial(String strVal); // added build 18.1.32

void Entity::setRenderMaterialMapping(Quader qdr, int bMapFront, double dScaleU, double
    dScaleV, [double dRandomLocation, double dRandomDirection]); // added build
    18.1.34

```

The `setRenderMaterialMapping` allows to specify the mapping of a render material to an entity in a special beam-box way. This beam-box way is the way materials are mapped to beams by the `HSB_MAPTEXTURE` command. The `dRandomLocation` (default 0.8) is a parameter to express the randomness by which the origin is altered. There is also a randomness for the mapping direction. Default is 0.03.

---

```

class RenderMaterial : DictObject { // see DictObject for base functions (added
    hsbCAD18.1.32)

    RenderMaterial(String strEntry); // constructor

    static RenderMaterial[] getAllEntries();
    static String[] getAllEntryNames();
}

```

---

[Example O-type TSL:

```

RenderMaterial csList[] = RenderMaterial ().getAllEntries ();
reportNotice ("\n =====");
for (int e=0; e<csList.length(); e++) {
    reportNotice ("\n entryName: "+csList[e].entryName ());
}

// declare a property with the names of the curved styles
PropString pS(0, RenderMaterial ().getAllEntryNames(), T("|
RenderMaterial name|"));

RenderMaterial csInvest(pS); // constructor with a curved style name
if (!csInvest.bIsValid()) {
    reportNotice (T("\n|RenderMaterial not valid|: ") +pS);
}

reportNotice ("\n ----");
reportNotice ("\n entryName: "+csInvest.entryName ());
reportNotice ("\n isValid: "+csInvest.bIsValid ());
reportNotice ("\n bIsSpecial: "+csInvest.bIsSpecial ());
reportNotice ("\n handle: "+csInvest.handle ());
reportNotice ("\n typeName: "+csInvest.typeName ());

```

—end example]

[Example O-type TSL:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint (); // select point
    Entity.append(getEntity ());

    return;
}

PropString pDimStyle(1, DimStyles, "Dim style");

```

---

```

PropDouble pTextHeight(0, U(20), "Text height");

PropString
pRM(2, RenderMaterial()).getAllEntryNames(), "RenderMaterial");

PropInt pMapFront(0, 1, "Map front");
PropDouble pScaleU(1, 1, "Scale U");
PropDouble pScaleV(2, 1, "Scale V");
PropDouble pRandomLocation(3, 0.8, "RandomLocation");
PropDouble pRandomDirection(4, 0.03, "RandomDirection");

if (_Entity.length() == 0) return;
Entity sp = _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strSetOverrides= T("|Apply render material to entity|");
addRecalcTrigger(_kContext, strSetOverrides);
if (_bOnRecalc && _kExecuteKey==strSetOverrides) {
    sp.setRenderMaterial(pRM);
}

String strApplyMapping= T("|Apply UV mapping|");
addRecalcTrigger(_kContext, strApplyMapping);
if (_bOnRecalc && _kExecuteKey==strApplyMapping)
{
    Quader qdr;
    GenBeam gb = (GenBeam)sp;
    if (gb.bIsValid()) {
        qdr = Quader(gb.ptCen(), gb.vecX(), gb.vecY(), gb.vecZ(),
gb.dL(), gb.dW(), gb.dH());
    }
    sp.setRenderMaterialMapping(qdr, pMapFront, pScaleU, pScaleV,
pRandomLocation, pRandomDirection);
}

CoordSys cs =sp.coordSys();
cs.vis();

RenderMaterial spStyle(sp.renderMaterial());

String strLines[0];
strLines.append("Entity");
strLines.append("renderMaterial: "+sp.renderMaterial());
strLines.append(spStyle);

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {

```

```

Vector3d vecO = -1*1.2*pTextHeight*_YU;
dp.draw(strLines[1],_Pt0+vecO,_XU,_YU, 1,1);
}

—end example]

```

### 6.5.13 SipStyle

The SipStyle, which is derived from the [DictObject](#), refers to a panel or [Sip](#) style description.

An DictObject can be casted into an SipStyle. However when the casting is not allowed, the resulting SipStyle will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because SipStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The SipStyle from a Sip can be constructed with the string returned by `Sip::style()`:

```

class SipStyle : DictObject { // see DictObject for base functions

    SipStyle(String strEntry); // constructor

    void dbCreate(double dThickness); // (added from hsbCAD 22.1.6 and 21.4.81)

    int numSipComponents() const; // the number of components
    SipComponent[] sipComponents() const; //
    void setSipComponents(SipComponent[] ar, int nAxisIndex); //
    SipComponent sipComponentAt(int nIndex) const; // return the SipComponent with a
                                                // specific index

    int axisIndex() const; // this is the index of the component on which the traditional Sip
                          // tools (e.g. PanelStop) do their job.
    double dThickness() const; // returns the sum of all components

    String surfaceQualityTop() const; // see SurfaceQualityStyle (added hsbCAD17.0.48)
    String surfaceQualityBottom() const; // example found in Sip

    static SipStyle[] getAllEntries();
    static String[] getAllEntryNames();

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    static String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from
                                                               dwg. Only use on special events.
    static String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return
                                                               error code as string. Empty string means all ok, see example below.
    static String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return
                                                               error code as string. Empty string means all ok, see example below.

    // following methods added in V24.1.28 and V23.8.40

```

```

int allowFlippingForComparison() const;
int allowRotationForComparison() const;
int showChildPanelFlipMark() const;
int markFlippedChildPanels() const;
String childPanelFlipPrefix() const;
int componentToExcludeFromOpenings() const;
double toleranceLumber() const;

void setAllowFlippingForComparison(int val);
void setAllowRotationForComparison(int val);
void setShowChildPanelFlipMark(int val);
void setMarkFlippedChildPanels(int val);
void setChildPanelFlipPrefix(String val);
void setComponentToExcludeFromOpenings(int val);
void setToleranceLumber(double val);
};

```

---

[Example O-type TSL:

```

Unit(1, "mm");

if (_bOnInsert) {

    // select the Panel and insertion point
    Sip sp = getSip();
    _Sip.append(sp);
    _Pt0 = getPoint();

    String strStyle = sp.style(); // get the style from the selected
    sip

    String strAllStyles[] = SipStyle().getAllEntryNames(); // list of
    all available sipstyles
    PropString pStyle(0, strAllStyles, "Sip style",
    strAllStyles.find(strStyle)); // make property
    showDialog(); // allow the user to change the style

    // change the sipstyle of the sip, to match the property value
    sp.setStyle(pStyle);

    return;
}

// check running conditions
if (_Sip.length()==0) return;
Sip sp = _Sip[0];

String strDbCreate = T("|Create a SipStyle|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )

```

---

```

{
    String strNewName = getString("new name:");
    SipStyle anew(strNewName);
    String strError = anew.dbCreate(U(100));
    if (strError == "")
    {
        // all fine
        sp.setStyle(anew.entryName());
        reportMessage("\nSipStyle created with name:
"+anew.entryName());
    }
    else
    {
        reportMessage("\nCould not create SipStyle with name:
"+strNewName);
        reportMessage("\n"+strError );
    }
}

String strStyle = sp.style();
PropString pStyle(0, strStyle , "Sip style"); // make property
pStyle.set(strStyle);
pStyle.setReadOnly(TRUE);

// compose a SipStyle from the String
SipStyle sipStyle(strStyle);

// get a list of components for this style
SipComponent sipComps[] = sipStyle.sipComponents();
int nAxis = sipStyle.axisIndex();

SipComponent spComp = sipStyle.sipComponentAt(nAxis);

sipStyle.setSipComponents(sipComps,nAxis); // sets the components

```

*—end example]*

---

[Example O-type TSL illustrating getAllEntryNamesFromDwg and importFromDwg:

```

Unit(1, "mm");

if (_bOnInsert) {

    String arListCur[] = SipStyle().getAllEntryNames();
    PropString pListCur(0,arListCur,"Current list");

    String strDwg = _kPathDwg + "\\Source.dwg";
    PropString pDwg(1, strDwg, "Dwg file to read from");
    showDialog(T("|_Default|"));
    strDwg = pDwg;
    pDwg.setReadOnly(TRUE);
}

```

---

```

String arList[] = SipStyle().getAllEntryNamesFromDwg(strDwg);
PropString pListFromDwg(2,arList, T("list from dwg"));

PropString pEntry1(3,"",T("|Entry 1 to import|"));
PropString pEntry2(4,"",T("|Entry 2 to import|"));
PropInt pOverwrite(5,0,T("|Overwrite|"));
showDialog();

String arToImport[0];
if (pEntry1 != "") arToImport.append(pEntry1);
if (pEntry2 != "") arToImport.append(pEntry2);
int bOverwrite = pOverwrite;

String strErr;
if (arToImport.length() == 1)
    strErr = SipStyle().importFromDwg(strDwg, arToImport[0],
bOverwrite);
else
    strErr = SipStyle().importFromDwg(strDwg, arToImport,
bOverwrite);

if (strErr != "")
{
    String strToLoad;
    for(int r=0; r<arToImport.length(); r++)
        strToLoad += " " + arToImport[r] + "!";
    reportWarning("Could not load some of" + strToLoad + " from "
+ strDwg + ": " + strErr);
}

String arListCurNew[] = SipStyle().getAllEntryNames();
PropString pListCurNew(6,arListCurNew,"New list");
showDialog(T("|_Default|"));

eraseInstance();
return;
}

```

*—end example]*

#### 6.5.13.1 SipComponent

The SipComponent describes one components of the SipStyle.

---

```

class SipComponent {

    SipComponent(String strName, String strMaterial, double dThickness);

```

---

```

String name() const;
void setName(String str);
String material() const;
void setMaterial(String str);

double dThickness() const;
void setDThickness(double dH);
int bAllowBevels() const;
void setBAllowBevels(int nSet);

int dimColor() const; // (added hsbCad17.1.4).
void setDimColor(int nSet); // (added hsbCad17.1.4).
};

```

### 6.5.14 SurfaceQualityStyle

The SurfaceQualityStyle, which is derived from the [DictObject](#), refers to a Sip entity its surfaceQuality top and bottom style description.

An DictObject can be casted into an SurfaceQualityStyle. However when the casting is not allowed, the resulting SurfaceQualityStyle will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because SurfaceQualityStyle is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The [SipStyle](#) has a surface quality for the top and the bottom. The [Sip](#) itself has an override for both.

```

class SurfaceQualityStyle : DictObject { // see DictObject for base functions (added
hsbCAD17.0.48)

    SurfaceQualityStyle(String strEntry); // constructor

    int quality() const; // (added hsbCAD17.1.14)

    static SurfaceQualityStyle[] getAllEntries();
    static String[] getAllEntryNames();

    // added in hsbCAD20.1.69 and hsbCAD21.0.43
    String[] getAllEntryNamesFromDwg(String strDwgName); // return list of entries from dwg.
        Only use on special events.
    String importFromDwg(String strDwgName, String strEntry, int bOverwrite); // return error
        code as string. Empty string means all ok, see SipStyle example.
    String importFromDwg(String strDwgName, String[] strEntries, int bOverwrite); // return error
        code as string. Empty string means all ok, see SipStyle example.
};

```

[Example O-type TSL:

```
Unit(1, "mm");

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

String strAllStyles[] = SurfaceQualityStyle() .getAllEntryNames(); // list of all available SurfaceQualityStyles

String strLines[0];
strLines.append(strAllStyles.length() + " styles found");
for (int s=0; s<strAllStyles.length(); s++) {
    SurfaceQualityStyle mps(strAllStyles[s]);
    strLines.append("strStyle: "+mps.entryName());
    strLines.append("quality: "+mps.quality());
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}
```

—end example]

### 6.5.15 TslScript

The TslScript, which is derived from the [DictObject](#), refers to a [TslInst](#) style description.

An DictObject can be casted into an TslScript. However when the casting is not allowed, the resulting TslScript will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because TslScript is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

The TslScript from a [TslInst](#) can be constructed with the string returned by [TslInst::scriptName\(\)](#);

---

```
class TslScript : DictObject { // see DictObject for base functions (added hsbCAD21.2.9)
```

```
TslScript(String strEntry); // constructor

static TslScript[] getAllEntries();
static String[] getAllEntryNames();
};
```

---

[Example O-type TSL:

```
Unit(1, "mm");

if (_bOnInsert) {

    // select the Panel and insertion point
    TslInst sp = getTslInst();
    _Entity.append(sp);
    _Pt0 = getPoint();

    return;
}

if (_Entity.length() == 0)
{
    eraseInstance(); // just erase from DB
    return;
}
TslInst sp = (TslInst) _Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strTslScript = sp.scriptName(); // get the scriptName from the
selected TslInst
String allTslScripts[] = TslScript().getAllEntryNames(); // list of
all available TslScripts
PropString pTslScript(0, allTslScripts, "TslScript",
allTslScripts.find(strTslScript)); // make property
pTslScript.setReadOnly(TRUE);

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

String strChangeEntity = T("|Update TslScript|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    pTslScript.setReadOnly(FALSE);
    showDialog(); // allow the user to change the style
    sp.setScriptName(pTslScript);
    pTslScript.setReadOnly(TRUE);
```

```

}

// compose a TslScript from the String
TslScript cncTslScript(pTslScript);

String strLines[0];
strLines.append("tslScript : "+pTslScript );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vec0 = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vec0,_XU, _YU, 1,1);
}

—end example]

```

### 6.5.16 TruckDefinition

The TruckDefinition, which is derived from the [DictObject](#), refers to a Truck style description.

An DictObject can be casted into an TruckDefinition. However when the casting is not allowed, the resulting TruckDefinition will become invalid. This can be checked with the `blsValid()` function of DictObject.

Because TruckDefinition is derived from DictObject, it inherits all the member functions of DictObject as well. It can also be casted into an DictObject.

---

```

class TruckDefinition : DictObject { // see DictObject for base functions (added in V26)

    TruckDefinition(String strEntry); // constructor

    string dbCreate(); // creates a new entry, make sure the strEntry that was set in the constructor
                        // does not exist yet. An error string is returned which is empty if all ok.

    double length() const;
    void setLength(double dVal);
    double width() const;
    void setWidth(double dVal);
    double height() const;
    void setHeight(double dVal);
    Quader quader() const;

    double allowedOversize(int side) const; // side should be _kXP = 0, _kXN, _kYP, _kYN, _kZP,
                                              // _kZN = 5 see also Quader
    void setAllowedOversize(int side, double dVal);
    Quader oversizeQuader() const;
}

```

---

```

PlaneProfile[] loadProfiles() const; // see PlaneProfile
void setLoadProfiles(PlaneProfile[] profs);

void setDisplay(Display display); // see Display::draw(TruckDefinition,...) method to draw it.

static TruckDefinition[] getAllEntries();
static String[] getAllEntryNames();
};

```

---

*[Example O-type TSL, to create a TruckDefinition:*

```

Unit(1, "mm");

String strNewNameSet = "";
String strDbCreate = T("|Create a new truck definition|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    String strNewName = getString(T("|enter new name|"));
    TruckDefinition anew(strNewName);
    String strError = anew.dbCreate();
    if (strError == "")
    {
        reportMessage ("\ndefinition created with name:
"+anew.entryName());
        strNewNameSet = anew.entryName();
    }
    else
    {
        reportMessage ("\nCould not create definition with name:
"+strNewName);
        reportMessage ("\n"+strError );
    }
}

String allTruckDefs[] = TruckDefinition().getAllEntryNames(); // list
of all available
PropString pTruckDef(0, allTruckDefs, "TruckDefinition"); // make
property
if (strNewNameSet.length() > 0)
    pTruckDef.set(strNewNameSet);

// compose a TruckDefinition from the String
TruckDefinition truckDef(pTruckDef);

String strTruckBox = T("|Define truck box from origin|");
addRecalcTrigger(_kContext, strTruckBox );
if (_bOnRecalc && _kExecuteKey == strTruckBox )
{

```

---

```

    Point3d ptCorner = getPoint(T("|Select corner point at flatbed
end of truck|"));
    double dHeight = getDouble(T("|Enter height of box|"));
    double dOversizeX = getDouble(T("|Enter allowed oversize X of
box|"));

    double dLength = abs(_XW.dotProduct(ptCorner - _PtW));
    double dWidth = 2 * abs(_YW.dotProduct(ptCorner - _PtW));

    truckDef.setLength(dLength);
    truckDef.setWidth(dWidth);
    truckDef.setHeight(dHeight);
    truckDef.setAllowedOversize(_kXP, dOversizeX);
}

String strAddProfile = T("|Add load profile|");
addRecalcTrigger(_kContext, strAddProfile );
if (_bOnRecalc && _kExecuteKey == strAddProfile )
{
    PLine pl = getEntPLine().getPLine();
    if (pl.area() > U(1) * U(1))
    {
        PlaneProfile profs[] = truckDef.loadProfiles();
        profs.append(PlaneProfile(pl));
        truckDef.setLoadProfiles(profs);
    }
}

String strModProfile = T("|Modify load profile|");
addRecalcTrigger(_kContext, strModProfile );
if (_bOnRecalc && _kExecuteKey == strModProfile )
{
    int index = getInt(T("|enter index of load profile to modify|"));
    int subtract = getInt(T("|subtract 1 or add 0|"));
    PLine pl = getEntPLine().getPLine();
    if (pl.area() > U(1) * U(1))
    {
        PlaneProfile profs[] = truckDef.loadProfiles();
        if (index >=0 && index < profs.length())
        {
            PlaneProfile prof = profs[index];
            prof.joinRing(pl, subtract);
            profs[index] = prof;
        }
        truckDef.setLoadProfiles(profs);
    }
}

String strRemProfile = T("|Remove load profile|");
addRecalcTrigger(_kContext, strRemProfile );
if (_bOnRecalc && _kExecuteKey == strRemProfile )
{
}

```

```

int index = getInt(T("|enter index of load profile to remove|"));
PlaneProfile profs[] = truckDef.loadProfiles();
if (index >= 0 && index < profs.length())
{
    profs.removeAt(index);
}
truckDef.setLoadProfiles(profs);
}

String strDefDisplay = T("|Define truck display|");
addRecalcTrigger(_kContext, strDefDisplay );
if (_bOnRecalc && _kExecuteKey == strDefDisplay )
{
    Display adp(-1);
    adp.showInTslInst(0);
    adp.draw("Test tekst", _Pt0, _XU, _YU, 0, 0);

    PrEntity ssE(T("|Select a set of body entities that define the
graphical rep of the truck|"), Entity());
    if (ssE.go())
    {
        Entity ents[] = ssE.set();
        for (int e = 0; e < ents.length(); e++)
        {
            Body body = ents[e].realBody();
            if (body.isNull())
                continue;
            adp.draw(body);
        }
    }

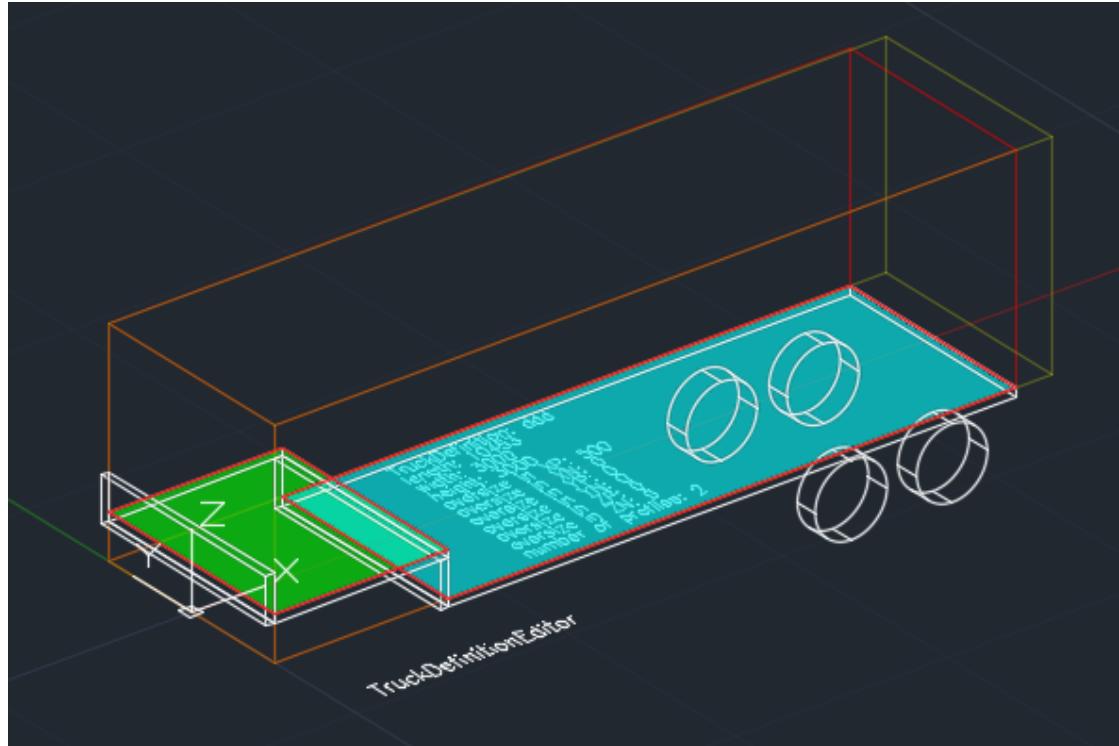
    truckDef.setDisplay(adp);
}

Display dp(-1);
dp.draw(scriptName(), _Pt0, _XU, _YU, 0, 0);

```

*—end example]*

*[Example O-type TSL, to visualize a TruckDefinition:*



```

Unit(1, "mm");

String allTruckDefs[] = TruckDefinition() .getAllEntryNames(); // list
of all available
PropString pTruckDef(0, allTruckDefs, "TruckDefinition"); // make
property

PropString pDimStyle(1, _DimStyles, T("|Dim style|"));
PropDouble pTextHeight(0, U(20), T("|Text height|"));
PropInt pDrawTruckRep(0, TRUE, T("|Draw truck representation|"));

// compose a TruckDefinition from the String
TruckDefinition truckDef(pTruckDef);
setDependencyOnDictObject(truckDef);

String strLines[0];
strLines.append("TruckDefinition: " + truckDef.entryName());
strLines.append("length: " + truckDef.length());
strLines.append("width: " + truckDef.width());
strLines.append("height: " + truckDef.height());

String strSides[] = { "XP", "XN", "YP", "YN", "ZP", "ZN" };
int arSideInts[] = { _kXP, _kXN, _kYP, _kYN, _kZP, _kZN };
for (int i = 0; i < 6; i++)
{
    int side = arSideInts[i];
    String strSide = strSides[i];
    strLines.append("allowedOversize in "+strSide+": " +
truckDef.allowedOversize(side));
}

```

```

}

PlaneProfile profs[] = truckDef.loadProfiles();
strLines.append("number of profiles: " + profs.length());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _PtG[0]+vecO, _XU, _YU, 1, 1);
}

CoordSys csTruLoc(_Pt0, _XU, _YU, _ZU);

Quader qdr = truckDef.quader();
qdr.transformBy(csTruLoc);
if (qdr.dX() > U(1) && qdr.dY() > U(1) && qdr.dZ() > U(1))
{
    dp.color(1);
    dp.transparency(40);
    dp.draw(Body(qdr));
}

Quader qdrOversize = truckDef.oversizeQuader();
qdrOversize.transformBy(csTruLoc);
if (qdrOversize.dX() > U(1) && qdrOversize.dY() > U(1) &&
qdrOversize.dZ() > U(1))
{
    dp.transparency(70);
    dp.color(2);
    dp.draw(Body(qdrOversize));
}

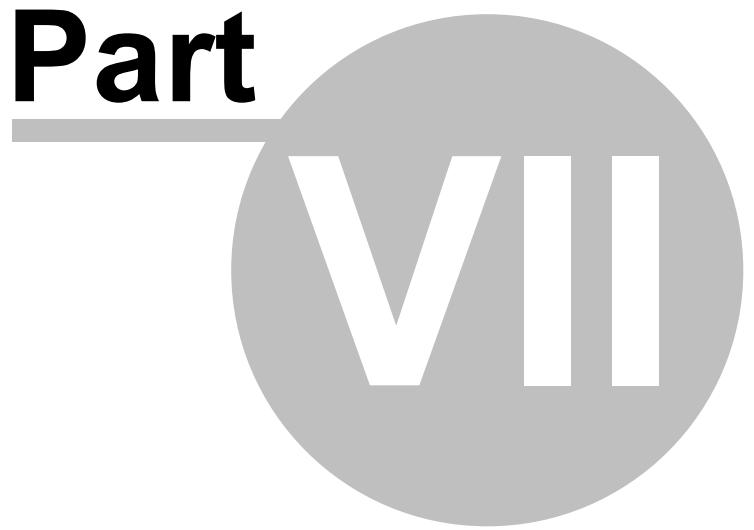
dp.transparency(40);
for (int p = 0; p < profs.length(); p++)
{
    PlaneProfile prf = profs[p];
    prf.transformBy(csTruLoc);
    dp.color(3+p);
    dp.draw(prf, _kDrawFilled);
}

if (pDrawTruckRep)
{
    dp.color(1);
    dp.transparency(0);
    dp.draw(truckDef, csTruLoc);
}

```

*—end example]*

**Part**



## 7 Entities

### 7.1 Entity

The Entity type is the basic type to reference an Autocad drawing entity. An entity could reference a beam, a wall element, a sheet,...

If the entity references a real Autocad entity, the `blsValid()` function will return true. An entity can be copied, transformed and erased.

Every toolscript instance in the drawing can hold zero, one or multiple references to entities. They are accessible through the predefined `_Entity` array.

This predefined is declared internally as:

```
Entity _Entity[];
```

The contents of the array is persistant over execution runs. This means that changes to the `_Entity` array, will be kept the next time the script is run.

There is however an important side effect. Entities of type Element, Opening and Viewport are internally also kept in the same Entity array. This has a number of consequences:

- All items stored in `_Opening`, `_Viewport` and `_Element` will always be items in the array `_Entity`.
- If an item is removed from eg. `_Opening`, the next execution run, it will not be an item in the `_Entity` array.
- If an item is added to the eg. `_Opening`, the next execution run, it will be an item in the `_Entity` array as well.
- If eg, an Opening is added to the `_Entity` array, it will appear inside the `_Opening` array, during the next execution run.

An entity can be casted into any type. However when the casting is not allowed, the resulting instance of that type will become invalid. This can be checked with the `blsValid()` function of Entity.

---

```
class Entity {
    int blsValid() const;
    int blsA(Entity ent) const;
    int blsKindOf(Entity ent) const;

    (GenBeam) ent;
    (Beam) ent;
    (Sheet) ent;
    (Sip) ent;
    (...) ent; // cast from any derived type. Substitute ... by the type name

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    void dbErase();
    Entity dbCopy() const;
    void dbCreate(String strObject, Map mapObject);

    PLine getPLine() const; // (added hsbCad13.2.132) // see also PLine and EntPLine
}
```

---

**CoordSys** coordSys() const; // returns the [CoordSys](#) of the entity (added v19.1.104 and v20.0.80). For the autocad entities that do not have a coordSys that follows the entity, the world coordinate system is returned.

**Quader** bodyExtents() const; // returns the extents [Quader](#) in the coordSys direction of the realBody (added 24.1.92 and 25.1.76)

**Quader** extentsWcs() const; // returns the extents [Quader](#) in the world direction (added 24.1.92 and 25.1.76)

**Element** element() const;  
**int** myZoneIndex() const;

**int** color() const; // color index of the entity  
**void** setColor(**int** nColorIndex); // sets the colorIndex as int

**int** hasTrueColor() const; // (added V23.0.75) return TRUE if the entity has a trueColor, instead of a colorIndex  
**int** trueColor() const; // (added V23.0.75) combined rgb value of the color of the entity. If !hasTrueColor, the colorIndex is converted to rgb.  
**void** setTrueColor(**int** rgb); // (added V23.0.75) sets the color by its rgb value. use [global method](#) [rgb](#)(**int** r, **int** g, **int** b)

**int** transparency() const; // (added hsbCad21.4.15) transparency percentage, 0 not transparent, 100 fully transparent  
**void** setTransparency(**int** nSet); // (added hsbCad21.4.15) set transparency.

**int** isVisible() const; // returns TRUE or FALSE (added build 16.3.5 and 17.0.40)  
**void** setIsVisible(**int** nSet); // sets the visibility, TRUE or FALSE (added build 16.3.5 and 17.0.40)

**int** lineWeight() const; // (added hsbCad14.0.82) linewidth value of the entity in 100th of a millimeter  
**void** setLineWeight(**int** nLineWeight); // sets the linewidth as int in 100th of a millimeter. Only specific values are available: see explanation below.

**String** notes() const; // notes shown in OPM, tab extended data. BEWARE: changing the notes in the OPM does not trigger the tsl to be recalculated.  
**void** setNotes(**String** strNote); // sets the notes.

**String** handle() const; // return the handle (=text representation of objectId) of the entity  
**String** hostId() const; // equivalent to handle (=text representation of objectId) of the object (added 25.1.57 and 24.1.88).  
**void** setFromHandle(**String** strHandle); // translates the string handle into an entity reference. (added hsbCAD2009+ build 14.2.8, hsbCAD2010 build 15.1.3)  
**String** uniqueId() const; // return the unique id aka Guid of the entity (added 24.0.10)  
**AcadDatabase** database() const; // (added V23.4.12 and V22.1.130) returns the [AcadDatabase](#) that this entity belongs to.

**String** layerName() const; // return the layername of the entity

**TslInst**[] tslInstAttached() const; // return list of attached tsl's to the entity. Tsl's are attached eg during appending an element tool to this Entity.

**assignToLayer**(**String** strLayer);

```

assignToGroups(Entity ent); // does not change the zone character
assignToGroups(Entity ent, char cZoneCharacter); // changes the zone character (added
build 18.2.49, 19.1.20 and 20.0.8)
assignToElementGroup(Element elem);
assignToElementGroup(Element elem, int bExclusive);
assignToElementGroup(Element elem, int bExclusive, int nZoneIndex, char cZoneCharacter);
assignToElementFloorGroup(Element elem);
assignToElementFloorGroup(Element elem, int bExclusive);
assignToElementFloorGroup(Element elem, int bExclusive, int nZoneIndex, char
cZoneCharacter);

Group[] groups() const; // return the Group list this entity belongs to
Grid grid() const; // return the relevant Grid for this entity. The group structure of the
entity is used to filter out the correct one.

Point3d[] gripPoints() const; // return list of grippoints of the entity

String typeName() const; // return the name of the entity type eg: AecDbWall
String typeDxfName() const; // return the dxf name of the entity type eg: AEC_WALL

// Body methods
Body realBody() const; // collect the body of the entity, if the entity has a body. If the
entity is an Acis solid, the AecFacetDev controls the facet approximation.
Reduce the value to obtain a more accurate approximation.
Body realBody(Vector vecDir) const; // specify the viewDir, because some entities have
different solid representations depending on the view direction
Body realBody(Vector vecDir, String strDisplaySetName) const; // specify the viewDir, and
display set name. The default name is empty, and refers to the model
display set.
Body realBodyTry() const; // (added V22.1.6 and V21.4.81) Try indicates that it will not pop
up an error if fails, but might return a body that returns true on isNull().
Body realBodyTry(Vector vecDir) const; // (added V22.1.6 and V21.4.81)
Body realBodyTry(Vector vecDir, String strDisplaySetName) const; // (added V22.1.6 and
V21.4.81)

static void createRealBodySatFile(Vector vecDir, String strDisplaySetName, Entity[] arEntity,
String strFileName); // (added hsbCAD2010 build 15.3.13 and hsbCAD2011
build 16.0.23)
static void createRealBodyStlFile(Vector vecDir, String strDisplaySetName, Entity[] arEntity,
String strFileName); // (added hsbCAD2014 build 19.0.62)

void setDrawOrderToFront(int bToFront); // if bToFront is TRUE, sets the draw order to front,
if FALSE, to back

Map getAutoPropertyMap() const; // all properties found are added to the Map which is
returned.
Map getAutoPropertyMap(String[] arPropertyNames) const; // only the properties specified
(case insensitive match) are added to the map which is returned

static String[] dispRepNames() const; // return list of display representation names as they
appear in the AecDisplayManager, that are registered to work for this entity's
type.

```

```

String[] attachedPropSetNames() const; // return list of attached property set names
                                         (added hsbCAD13.2.78)
Map getAttachedPropSetMap(String strPropSetName) const; // all properties found in the
                                         attached property set with the given name are added to the Map which is
                                         returned. (added hsbCAD13.2.78). BEWARE getting bigger propset maps
                                         seems to be a resource intensive action. Consider to just call specific
                                         properties.
Map getAttachedPropSetMap(String strPropSetName, String[] arPropertyNames) const; //
                                         only the properties specified (case insensitive match) are added to the map
                                         which is returned. (added hsbCAD13.2.78)

String[] availablePropSetNames() const; // return list of property set names that can be
                                         attached to the entity (added hsbCAD13.2.80)
static String[] availablePropSetNames(); // return list of property set names that can be
                                         attached entity type (added hsbCAD17.0.49)
int attachPropSet(String strPropSetName); // check if the propset was already attached, if
                                         not attached a new set (added hsbCAD13.2.80)
int removePropSet(String strPropSetName); // check if the propset is attached, if it is
                                         attached remove the set (added hsbCAD13.2.80)

int setAttachedPropSetFromMap(String strPropSetName, Map
                                         mapWithValuesToBeChanged); // all properties found in the map that match
                                         property names and types from the property set are changed to the value
                                         from the map. (added hsbCAD13.2.80)
int setAttachedPropSetFromMap(String strPropSetName, Map mapWithValuesToBeChanged,
                                         String[] arPropertyNames); // the properties found in the arPropertyNames
                                         array and found in the map that match property names and types from the
                                         property set are changed to the value from the map. (added hsbCAD13.2.80)
Beware: The units of the values in the map must match the units of the properties to set.
Length, Area, Volume, Angle are all supported in the Map, and will be
converted automatically to the drawing units. All the other Aec unit types
must be add to the map with _kNoUnit parameter.

int createPropSetDefinition(String strPropSetName, Map mapWithKeysAsProperties, int
                                         bOverwriteExistingKeys); // (added hsbCAD19.1.81 and hsbCAD20.0.50)
int renamePropertiesInSet(String strPropSetName, String[] arOldNames, String arNewNames,
                                         int bWriteGlobalName); // (added hsbCAD21.2.10)

// The subMap and setSubMap were previously members of GenBeam only. Since
// hsbCad 13.2.124 moved down to Entity. But they remain only available for
// some hsbCAD entities.
Map subMap(String strKey) const; // see Map
void setSubMap(String strKey, Map mapNew);
int hasSubMapContainer() const; // return TRUE if a sub map can be set (added
                                         hsbCad13.2.124). Returns only TRUE for those entities on which it can work.
String[] subMapKeys() const; // return list of available sub map keys (added
                                         hsbCad13.2.124)
int removeSubMap(String strKey); // (added hsbCad13.2.124)

// The subMapX set of methods are available for all entities, including none hsbCAD
// entities eg EntPLine. (added hsbCad15.0.13).
Map subMapX(String strKey) const; // see Map

```

```

Map subMapX(String strKey, CoordSys csRef-modified) const; // (added V23.7.4) retrieve
    the data, transformed from csRef-original to the new csRef-modified.
void setSubMapX(String strKey, Map mapNew);
void setSubMapX(String strKey, Map mapNew, CoordSys csRef-original); // (added V23.7.4)
    store also a reference coordinate system with the data, to work in
    combination with subMapX(strKey, csRef)
String[] subMapXKeys() const; // return list of available sub map keys
void removeSubMapX(String strKey);

int optionEvaluate() const; // (added hsbCad14.0.30) The hsbCAD option system evaluates
    each entity into a TRUE or FALSE state, visible or not visible. If the entity is
    not part of any options, the optionEvaluate is TRUE (visible).
void optionAssignSameRuleSet(Entity ent); // (added hsbCad14.0.34) The options rule set of
    ent is also added to this entity.

Map getClassificationMap(int bAddClassificationFromStyle) const; // all classifications are
    added to the Map which is returned (added hsbCAD16.0.21).

static EntityCollection getEntitiesWithPosnum(int nPosnumToSearchFor); // (added build
    16.1.18) return a list of all the entities with the given posnum. It returns the
    entities of the current database, not the entities embedded in xrefs.

String hyperlink() const; // added 17.1.8: hyperlink shown in OPM, tab extended data.
    BEWARE: changing the hyperlink in the OPM does not trigger the tsl to be
    recalculated.
void setHyperlink(String strLink); // sets the hyperlink.
void setHyperlink(String strLink, String strVisibleName); // sets the hyperlink.

// for an example of xrefName and allBlockRefPaths see Element
Entity[] allBlockRefPaths() const; // (added build 18.1.15) returns the list of a appearances
    of this entity. Multiple exist if the xref is inserted multiple times.
Entity[] allBlockRefPaths(int bLookAtXrefsOnUnfrozenLayersOnly) const; // (added build
    18.1.15)
String xrefName() const; // added 18.1.15
Entity[] blockRefs() const; // (added build 21.1.39) returns the list of block references in
    one block ref path. Multiple exist if the entity is in an nested xref. None exist
    for an entity in the host drawing.
Entity blockRef() const; // (added build 21.1.39) returns blockRefs[0] if it exists

String renderMaterial() const; // see example in RenderMaterial (added build 18.1.32)
void setRenderMaterial(String strVal); // added build 18.1.32
void setRenderMaterialMapping(Quader qdr, int bMapFront, double dScaleU, double
    dScaleV, [double dRandomLocation, double dRandomDirection]); // added build
    18.1.34, see RenderMaterial

void removeHsbData(); // (added hsbCAD19.1.101 and hsbCAD20.0.68) The hsbData is the
    OpeningSF part of the Opening, or the ElementSF part of the Wall.

// The refDocs api was added build 21.0.48. An example of its use is found in example
below.
String[] refDocs() const; // added build 21.0.48
int setRefDocs(String[] arDocs); // return TRUE if all ok (added build 21.0.48)

```

```

int setRefDocs(String[] arDocs, String[] arDescs); // return TRUE if all ok (added build
21.0.48)
int numRefDocs() const; // (added build 21.0.48)
int appendRefDoc(String doc, String desc); // return TRUE if all ok (added build 21.0.48)
String getRefDocAt(int nIndex) const; // return doc (added build 21.0.48)
String getRefDocDescAt(int nIndex) const; // return description (added build 21.0.48)

String formatObject(String strFormat); // (added 21.3.10)
String formatObject(String strFormat, Map mapAdditionalVariables); // (added 21.3.10)
String[] formatObjectVariables(); // (added 21.3.10) returns a list of all variables exposed on this
entity
String[] formatObjectVariables(String strType); // (added 22.1.87) returns a list of all variables
exposed on this entity as type strType (eg "Entity" or "GenBeam")

int acceptObject(String strFilter); // (added 23.0.43)
int acceptObject(String strFilter, Map mapAdditionalVariables); // (added 23.0.43)
String errorMessage(String strFilter); // (added 23.0.43)
String errorMessage(String strFilter, Map mapAdditionalVariables); // (added 23.0.43)

static <Entity>[] filterAcceptedEntities(<Entity>[] arEntities, String strFilter); // (added 23.0.75)
    return the list of entities that are accepted by the filter
static <Entity>[] filterAcceptedEntities(<Entity>[] arEntities, String strFilter, Map
    mapAdditionalVariables); // (added 23.0.75)
static <Entity>[] filterEntitiesOfType(<Entity>[] arEntities, String strType); // (added 27.5.1 and
    28.2.4) return the list of entities that match the type specified. The type is the
    PainterDefinition string representation of the type.
}

```

---

```
int blsValid() const;
```

Check if the instance references a valid/real entity in the Autocad database. This also validates the corresponding TSL type. A Panel entity from Autocad, will not be a valid Beam in TSL. But a Beam entity in Autocad is indeed a valid Beam in TSL.

```
int blsKindOf(Entity ent) const;
```

Find out if the autocad entity can be casted to the corresponding specified TSL type. If the ent argument is of type Beam, then this routine checks if the autocad entity can be casted to a Beam. Returns true if "this" object is of either the class represented by ent, or a class derived from ent.

*[Example O-type:*

```

if (_bOnInsert) {

    Entity ent = getEntity(); // select an entity
    if ( ent.blsKindOf(TslInst()) ) { // check if it is a TSL
        TslInst tsl = (TslInst) ent; // should be a valid cast now
        // ... tsl can be used now
    }

    return;
}
```

-- end example]

```
int blsA(Entity ent) const;
```

Find out if the autocad entity is of the corresponding specified TSL type. If the ent argument is of type Beam, then this routine checks if the autocad entity is a Beam. Returns true if "this" object is of the class represented by ent. For the general/base types Element, TsInst, Opening,... this call will most likely return FALSE. If you want to check if a cast operation is allowed, better use the blsKindOf routine.

```
(GenBeam) ent;
(Beam) ent;
(Sheet) ent;
(Sip) ent;
(...) ent;
```

Return the ent, but casted into the specified type.

```
dbErase();
```

Erase the entity from the autocad database. It is an immediate action.

```
Entity dbCopy() const;
```

Return a copy of the current entity. A copy is made similar to the copy command inside autocad.

Dependent entities might be copied with it (surface drill). Beams might stay connected with certain tools (roofplane).

For entities derived from GenBeam, there is a special copy function: [dbCopyShape](#).

```
void dbCreate(String strObject, Map mapObject);
```

The dbCreate allows to create an object by its [Map](#) description.

```
transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);
```

Do a transformBy of the entity inside the autocad database.

```
Element element() const;
```

Return the element that the entity belongs to. If the entity does not belong to an element, the element returned will not be valid.

**int myZoneIndex()** const;

Return the zoneIndex that the entity belongs to. If the entity does not belong to an element, the zone index returned is set to 0.

**assignToLayer(**String** strLayerName);**

The entity can be assigned to a particular layer with a given name. If you want to assign the current tsllnst: \_Tsllnst to a particular layer, please use the [global routine](#) instead of this one. The global routine will also prevent automatic assignment to other layers.

*[Example T-type:*

```
Cut ct(_Pt0,_Z1);
    _Beam0.addTool(ct,1);
    assignToLayer("aa");
    _Beam0.assignToLayer("bb");
-- end example]
```

**assignToGroups(**Entity** ent);**

The entity can be assigned to the same groups of an existing entity. To do this the function assignToGroups must be used. The entity that is used can be a beam, sheet, element,...

```
assignToElementGroup(Element elem); // see also Element
assignToElementGroup(Element elem, int bExclusive);
assignToElementGroup(Element elem, int bExclusive, int nZoneIndex, char cZoneCharacter);
assignToElementFloorGroup(Element elem);
assignToElementFloorGroup(Element elem, int bExclusive);
assignToElementFloorGroup(Element elem, int bExclusive, int nZoneIndex, char cZoneCharacter);
```

To assign the entity to the wall or roof element group, the function assignToElementGroup can be used. For an existing element (wall or roof), the element group might not exist yet. In that case, the entity will not be regrouped. The entity can also be added to the elements floor group with the assignToElementFloorGroup call.

elem: the element that the instance will be assigned to

bExclusive: normally an instance would only belong to one element: bExclusive is TRUE (is default). In some cases you want to specify if the instance belongs to multiple element groups.

nZoneIndex: if specified, the zone index: (-5,-4,...4,5)

cZoneCharacter: specifies the entity set; should be equal to 'Z' for general items, 'T' for beam tools, ('T' is default), 'E' for element tools

**Map getAutoPropertyMap()** const; // see also [Map](#)

**Map getAutoPropertyMap(**String[]** arPropertyName) const;**

Each autocad entity has a set of automatic properties defined. One way to see a list of the automatic properties, is to go into the AecStyleManager -> DocumentationObjects -> PropertySetDefinition, and create a dummy PropertySetDefinition. Set it to work with a single

object of the type that you want to investigate in the Applies to tab. Then go to the tab Definition, and try to add an automatic property. The full list is shown.

With this function, the list of automatic properties is made available in TSL. Only int, double and string entries are recognized currently.

*[Example O-type, with insert done in script. The script will list all the automatic properties of the selected entity.*

```

String arAS[]={T("|All|"),T("|Only selected|")};
PropString pShow(0,arAS,T("|What properties to query|"));

if (bOnInsert) {

    showDialogOnce(); // ask user the question

    Entity ent = getEntity();

    String arKeys[]={T("|Length|"),T("|Volume|"),T("|Color|"),T("|Layer|")};

    Map mapProp;
    if (pShow==T("|All|"))
        mapProp = ent.getAutoPropertyMap();
    else
        mapProp = ent.getAutoPropertyMap(arKeys);

    reportMessage("\n--- "+mapProp.length()+" automatic properties found");
    for (int i=0; i<mapProp.length(); i++) {
        String strKey = mapProp.keyAt(i);
        if (mapProp.hasInt(i))
            reportMessage("\n"+i+".I) "+strKey+":
"+mapProp.getInt(i));
        else if (mapProp.hasDouble(i))
            reportMessage("\n"+i+".D) "+strKey+":
"+mapProp.getDouble(i));
        else if (mapProp.hasString(i))
            reportMessage("\n"+i+".S) "+strKey+":
"+mapProp.getString(i));
    }

    return;
}
eraseInstance();

-- end example]
```

**String[]** dispRepNames() const;

Each entity type has a zero, one or multiple display representations registered in the display system of Adt. The list of display representation names can be retrieved for a particular Entity type. The names can be used in the showInDispRep method of [Display](#).

[Example O-type, with insert done in script. The script will list all the dispreps of the selected entity, of \_ThisInst and of the Beam type.

```

if (_bOnInsert) {

    // list of dispRep names of a selected entity
    Entity ent = getEntity(); // select entity
    String arNames[] = ent.dispRepNames();

    // report list of dispRep names
    reportMessage("\n\nList of dispRepNames for selected entity");
    for (int i=0; i<arNames.length(); i++) {
        reportMessage("\n"+arNames[i]);
    }

    // list of dispRep names of _ThisInst
    String arNames2[] = _ThisInst.dispRepNames();

    // report list of dispRep names
    reportMessage("\n\nList of dispRepNames of _ThisInst");
    for (int i=0; i<arNames2.length(); i++) {
        reportMessage("\n"+arNames2[i]);
    }

    // list of dispRep names of a specific type
    String arNames3[] = Beam().dispRepNames();

    // report list of dispRep names
    reportMessage("\n\nList of dispRepNames of Beam");
    for (int i=0; i<arNames3.length(); i++) {
        reportMessage("\n"+arNames3[i]);
    }

    eraseInstance();
}

-- end example]

```

```
String[] attachedPropSetNames() const; // return list of attached property set names that are  
// actually attached  
String[] availablePropSetNames() const; // return list of available property set names that could  
// be attached  
Map getAttachedPropSetMap(String strPropSetName) const; // all properties found in the  
// attached property set with the given name are added to the Map which is  
// returned.
```

**Map** getAttachedPropSetMap(**String** strPropSetName, **String[]** arPropertyNames) const; // only the properties specified (case insensitive match) are added to the map which is returned

*[Example O-type, with insert done in script. The script will list the properties in a particular propertyset.*

```

if (_bOnInsert) {

    Entity ent = getEntity();

    String arNamesAv[] = ent.availablePropSetNames();
    if (arNamesAv.length()==0) arNamesAv.insertAt(0,"");
    PropString pPropSetNameAv(2,arNamesAv,T("|Property sets
available|"));

    String arNames[] = ent.attachedPropSetNames();
    if (arNames.length()==0) arNames.insertAt(0,"");
    PropString pPropSetName(0,arNames,T("|Property set|"));

    String arAS[]={T("|All|"),T("|Only selected|")};
    PropString pShow(1,arAS,T("|What properties to query|"));

    showDialog(" --- "); // ask user the question, show default
properties

    String arKeys[]={ "oneprop", "Color" };

    Map mapProp;
    if (pShow==T("|All|"))
        mapProp = ent.getAttachedPropSetMap(pPropSetName);
    else
        mapProp = ent.getAttachedPropSetMap(pPropSetName,arKeys);

    reportMessage("\n--- "+mapProp.length()+" properties found");
    for (int i=0; i<mapProp.length(); i++) {
        String strKey = mapProp.keyAt(i);
        if (mapProp.hasInt(i))
            reportMessage("\n"+i+".I) "+strKey+": "+mapProp.getInt(i));
        else if (mapProp.hasDouble(i))
            reportMessage("\n"+i+".D) "+strKey+": "
+mapProp.getDouble(i));
        else if (mapProp.hasString(i))
            reportMessage("\n"+i+".S) "+strKey+": "
+mapProp.getString(i));
    }

-- end example]

```

```
int attachPropSet(String strPropSetName); // check if the propset was already attached, if not
                                         // attached a new set (added hsbCAD13.2.80)
int removePropSet(String strPropSetName); // check if the propset is attached, if it is attached
                                         // remove the set (added hsbCAD13.2.80)
```

[Example O-type, with insert done in script. The script will attach a propset chosen.

```
if (_bOnInsert) {

    Entity ent = getEntity();

    String arNames[] = ent.availablePropSetNames();
    if (arNames.length()==0) arNames.insertAt(0,"");
    PropString pPropSetName(0,arNames,T("|Property sets available|"));

    showDialog(" --- "); // ask user the question, show default
properties

    int bAdded = ent.attachPropSet(pPropSetName);

    reportMessage("\n--- Propset added successfully: "+bAdded );
}

-- end example]
```

```
int renamePropertiesInSet(String strPropSetName, String[] arOldNames, String arNewNames, int
                           bWriteGlobalName); // (added hsbCAD21.2.10)
```

Inside the property, there is a name and a globalName. The lookup (using the oldName) is  
always first trying to find the name, if not found, the globalName is tried.  
Which of the 2 is actually set, is determined by the bWriteGlobalName value.

[Example O-type to illustrate the renaming of a property in a property set.

```
String strRenamePropSetForGenBeam = T("|Rename properties in a set
for genbeam|");
addRecalcTrigger(_kContext, strRenamePropSetForGenBeam );
if (_bOnRecalc && _kExecuteKey==strRenamePropSetForGenBeam )
{
    String arNames[] = GenBeam().availablePropSetNames();
    if (arNames.length()==0) arNames.insertAt(0,"");
    PropString pPropSetName(0,arNames,T("|GenBeam property sets
available|"));
    PropString pPropertyNameOld(1,"",T("|Property name old|"));
    PropString pPropertyNameNew(2,"",T("|Property name new|"));
    PropInt pUseGlobal(0,0,T("|Use global|"));
}
```

```

showDialog("---"); // ask user the question, show default
properties

String strPropSetName = pPropSetName;
if (strPropSetName == "")
    return;

String arOld[] = { pPropNameOld };
String arNew[] = { pPropNameNew };
int bSuccess = GenBeam().renamePropertiesInSet(strPropSetName,
arOld, arNew, pUseGlobal );

reportMessage("\n--- GenBeam propset props renamed: "+bSuccess);

}

-- end example]

```

```
int createPropSetDefinition(String strPropSetName, Map mapWithKeysAsProperties, int
bOverwriteExistingKeys); // (added hsbCAD19.1.81 and hsbCAD20.0.50)
```

Each key in the given map represents one property. The type of the map entry is used to define the type of the property: double, text, integer.

If a more fine grained access to the property set definition is required, please use the managed interface of ACA:

<Program Files>\Autodesk\AutoCAD 2015\ACA\AecPropDataMgd.dll

see for example: <http://adndevblog.typepad.com/aec/2012/09/defining-a-property-set-definition-in-.html>

*[Example O-type. The script will create a new propset definition on context menu action. The propset can be created on any Entity type or on selected entity.*

```

String strCreatePropSetForSelection = T("|Create new property set
definition for entity|");
addRecalcTrigger(_kContext, strCreatePropSetForSelection );
if (_bOnRecalc && _kExecuteKey==strCreatePropSetForSelection )
{
    Entity ent = getEntity();

    String arNames[] = ent.availablePropSetNames();
    if (arNames.length()==0) arNames.insertAt(0,"");
    PropString pPropSetName(0,arNames,T("|Property sets
available|"));
    PropString pPropSetNew(1,"",T("|New property set name (leave
empty to adjust existing)|"));

    showDialog("---"); // ask user the question, show default
properties

    String strPropSetName = pPropSetNew;
    if (strPropSetName == "")
        strPropSetName = pPropSetName;
}

```

```

        if (strPropSetName == "")
            return;

    Map mpPropDef;
    mpPropDef.appendString("strString","boe");
    mpPropDef.appendInt("myInt",1);
    mpPropDef.appendDouble("ddd", 0.1);
    mpPropDef.appendDouble("dLength", 0.2, _kLength);
    mpPropDef.appendDouble("dAngle", 0.3, _kAngle);
    mpPropDef.appendDouble("dArea", 0.4, _kArea);
    mpPropDef.appendDouble("dVolume" ,0.5, _kVolume);
    mpPropDef.appendDouble("dNoUnit", 0.6, _kNoUnit);

    int bOverwriteExistingProperties = FALSE;

    int bExists = ent.createPropSetDefinition(strPropSetName,
    mpPropDef, bOverwriteExistingProperties );

    reportMessage("\n--- Propset added exists: "+bExists );
}

String strCreatePropSetForGenBeam = T("|Create new property set
definition for genbeam|");
addRecalcTrigger(_kContext, strCreatePropSetForGenBeam );
if (_bOnRecalc && _kExecuteKey==strCreatePropSetForGenBeam )
{
    String arNames[] = GenBeam().availablePropSetNames();
    if (arNames.length()==0) arNames.insertAt(0,"");
    PropString pPropSetName(0,arNames,T("|GenBeam property sets
available|"));
    PropString pPropSetNew(1,"",T("|New property set name (leave
empty to adjust existing)|"));

    showDialog("----"); // ask user the question, show default
properties

    String strPropSetName = pPropSetNew;
    if (strPropSetName == "")
        strPropSetName = pPropSetName;
    if (strPropSetName == "")
        return;

    Map mpPropDef;
    mpPropDef.appendString("strString","boe");
    mpPropDef.appendInt("myInt2",1);
    mpPropDef.appendString("ddd", "0.234 str");

    mpPropDef.appendDouble("dLength", 0.22, _kLength);
    mpPropDef.appendDouble("dAngle", 0.33, _kAngle);
    mpPropDef.appendDouble("dArea", 0.44, _kArea);
    mpPropDef.appendDouble("dVolume" ,0.55, _kVolume);
}

```

```

    mpPropDef.appendDouble("dNoUnit", 0.66, _kNoUnit);

    int bOverwriteExistingProperties = TRUE;

    int bExists = GenBeam().createPropSetDefinition(strPropSetName,
mpPropDef, bOverwriteExistingProperties );

    reportMessage("\n--- GenBeam propset added exists: "+bExists );

}

-- end example]

```

```

Map subMap(String strKey) const; // since hsbCAD13.2.124 moved from GenBeam to Entity.
void setSubMap(String strKey, Map mapNew);
int hasSubMapContainer() const; // return TRUE if a sub map can be set (added
hsbCad13.2.124)
String[] subMapKeys() const; // return list of available sub map keys (added hsbCad13.2.124)
int removeSubMap(String strKey); // (added hsbCad13.2.124)

```

Some entity types have an internal instance map, which serves as a submap container. The method hasSubMapContainer returns TRUE in this case, otherwise it returns FALSE. GenBeam and ToolEnt have such a submap container, but many other types without a Tsl wrapper as well, including the ShopdrawMultiPage entity. The Tsl can add a submap to this instance map.

*[Example O-type:*

```

if (_bOnInsert) {
    Entity.append(getEntity()); // select an entity
    _Pt0 = getPoint(); // select a point

    // compose a map
    Map mapw;
    mapw.setString("key", "myVal3");

    // (over)write submap of entity
    _Entity[0].setSubMap("subMap2", mapw);

    // also remove another submap if present
    int bRemoved = _Entity[0].removeSubMap("subMap1");
    reportMessage("\n removed subMap: "+bRemoved);

    return;
}

Entity ent = _Entity[0];

// Now interrogate map
int bHas = ent.hasSubMapContainer();
reportMessage("\n hasSubMapContainer: "+bHas);
if (bHas) {
    String arKeys[] = ent.subMapKeys();
}

```

```

        reportMessage("\n number of keys: "+arKeys.length());

        for (int k=0; k<arKeys.length(); k++) {
            reportMessage("\n key "+k+": "+arKeys[k]);
        }
    }

    Map mapr = ent.subMap("subMap2");
    reportMessage("\n"+mapr.getString("key"));

—end sample]

```

**PLine getPLine() const;** // (added hsbCad13.2.132) // see also [PLine](#) and [EntPLine](#)

If the entity has a Pline or if it can be converted to it, then the pline is returned here. So Line, Arc, even Wall entities can be converted to a pline.

*[Example O-type, illustrating the getPLine on Entity.*

```

if (_bOnInsert) {
    _Entity.append(getEntity());
    _Pt0 = getPoint();
    return;
}

if (_Entity.length()==0 || !_Entity[0].bIsValid()) {
    return;
    eraseInstance();
}

Entity ent = _Entity[0];
PLine pline = ent.getPLine(); // get the pline from the entity if
it has one

Display dp(2); // color yellow
dp.draw(pline);

```

—end sample]

**int lineWeight() const;** // (added hsbCad14.0.82)  
**void setLineWeight(int nLineWeight);**

The line weight of an entity can be queried and changed. The value is expressed in 100th of a millimeter, and is only one of the following: 0, 5, 9, 13, 15, 18, 20, 25, 30, 35, 40, 50, 53, 60, 70, 80, 90, 100, 106, 120, 140, 158, 200, 211, or -1, -2, -3.

The -1 represents the by layer, the -2 is by block, and the -3 is by default.

**Map getClassificationMap(int bAddClassificationFromStyle) const;** // (added hsbCAD16.0.21).

If the entity is an ACA entity, it can have a classification. To get the classification values, one can use this method. If the entity also has a style with a classification set, the style could contribute to the classification. Setting the argument to TRUE will first add the style values to the map, and then add or possibly overwrite the entity values to the map. The [DictObject](#) also has a getClassificationMap method.

*[Example O-type, illustrating the getClassificationMap on Entity.]*

```

String arAS[]={T("|Entity|"),T("|Entity and style|")};
PropString pShow(0,arAS,T("|What classifications to get|"));

if (_bOnInsert) {

    showDialogOnce(); // ask user the question

    Entity ent = getEntity();

    int bAddClassificationFromStyle = TRUE;
    if (pShow==T("|Entity|"))
        bAddClassificationFromStyle = FALSE;

    Map mapEnt =
    ent.getClassificationMap(bAddClassificationFromStyle);

    reportMessage("\n--- "+mapEnt.length()+" classifications
found");
    for (int i=0; i<mapEnt.length(); i++) {
        String strKey = mapEnt.keyAt(i);
        if (mapEnt.hasString(i))
            reportMessage("\n"+i+" "+strKey+":
"+mapEnt.getString(i));
    }

    return;
}
eraseInstance();

```

*—end sample]*

```

static void createRealBodySatFile(Vector vecDir, String strDisplaySetName, Entity[] arEntity,
String strFileName); // (added hsbCAD2010 build 15.3.13 and hsbCAD2011 build
16.0.23)
static void createRealBodyStlFile(Vector vecDir, String strDisplaySetName, Entity[] arEntity,
String strFileName); // (added hsbCAD2014 build 19.0.62)

```

Creates for the given array of entities a sat file that contains the bodies. This method is similar to the [Body](#)::createSatFile ([Body](#)::createStlFile), but has the advantage that the entity might be streamed into an Acis solid if applicable. Also no bodies need to be generated for the individual entities. This now all happens internally. The vecDir and strDisplaySetName are not relevant for

GenBeam entities, but might be important for others. They have the identical meaning as for the realBody method.

*[Example O-type to illustrate the createRealBodySatFile or createRealBodyStlFile*

```
if (_bOnInsert) {

    PrEntity ssE(T("\n|Select a set of entities|"), Entity());
    if (ssE.go()) {
        Entity arEnt[] = ssE.set();
        Vector3d vec(1,1,1);
        String strDispRep = "";

        String strFile = _kPathDwg +"\\\"+ scriptName(); // extension
will be added automatically
        Entity().createRealBodySatFile(vec, strDispRep, arEnt,
strFile);
        // Entity().createRealBodyStlFile(vec, strDispRep, arEnt,
strFile);
    }
    eraseInstance();
    return;
}
```

*--end example]*

```
String[] refDocs() const; // added build 21.0.48
int setRefDocs(String[] arDocs); // return TRUE if all ok (added build 21.0.48)
int setRefDocs(String[] arDocs, String[] arDescs); // return TRUE if all ok (added build 21.0.48)
int numRefDocs() const; // (added build 21.0.48)
int appendRefDoc(String doc, String desc); // return TRUE if all ok (added build 21.0.48)
String getRefDocAt(int nIndex) const; // return doc (added build 21.0.48)
String getRefDocDescAt(int nIndex) const; // return description (added build 21.0.48)
```

Manipulates the reference documents which are attached to the entity. These can be found in the property manager, tab "extended data".

*[Example O-type to illustrate refDocs manipulation*

```
Unit(1, "mm");
PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_bOnInsert) {
    _Pt0 = getPoint();
    _Entity.append(getEntity());
    return;
}

if (_Entity.length() == 0)
```

```

{
    eraseInstance();
    return;
}

Entity ent = _Entity[0];
setDependencyOnEntity(ent);

String strChangeEntity = T("|Append refdoc getAll setAll|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    String arRefDocsToSet[] = ent.refDocs();
    arRefDocsToSet.append("cc");
    ent.setRefDocs(arRefDocsToSet);
}

String strChangeEntity2 = T("|Append refdoc|");
addRecalcTrigger(_kContext, strChangeEntity2 );
if (_bOnRecalc && _kExecuteKey==strChangeEntity2)
{
    ent.appendRefDoc("cc", "dd");
}

String strChangeEntity3 = T("|Append refdoc with desc setAll|");
addRecalcTrigger(_kContext, strChangeEntity3 );
if (_bOnRecalc && _kExecuteKey==strChangeEntity3)
{
    String arDoc[0];
    String arDesc[0];
    int nNum = ent.numRefDocs();
    for(int i=0; i<nNum; i++)
    {
        arDoc.append(ent.getRefDocAt(i));
        arDesc.append(ent.getRefDocDescAt(i));
    }
    arDoc.append("ee");
    arDesc.append("ff");

    ent.setRefDocs(arDoc, arDesc);
}

String strLines[0];
int nNum = ent.numRefDocs();
for(int i=0; i<nNum; i++)
{
    String strDoc = ent.getRefDocAt(i);
    String strDesc = ent.getRefDocDescAt(i);
    strLines.append(strDoc + " - " + strDesc);
}

// display the lines

```

```

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vec0 = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vec0,_XU,_YU, 1,1);
}

—end example]

```

### **String** xrefName() const; // added 18.1.15

Returns the name of the xref the entity belongs to.

### **Entity[]** blockRefs() const; // added build 21.1.39

For a given entity this method returns the list of block references that identify this particular entity. For an entity in the host drawing, the array is empty. But for an entity in an xref, this array contains the block reference to which the entity belongs. If the entity belongs to an xref inside an xref, the returned array will contain 2 references.

### **Entity** blockRef() const; // added build 21.1.39

For a given entity this method returns the top most block reference blockRefs()[0], if it exists.

### **Entity[]** allBlockRefPaths() const; // added build 18.1.15

### **Entity[]** allBlockRefPaths(**int** bLookAtXrefsOnUnfrozenLayersOnly) const; // added build 18.1.15

The method returns all the different block ref paths for a simple entity reference. If an xref is inserted twice in the same drawing, the length of the array will be 2 when you call this method on any entity inside the xref. For an entity in the host drawing, the array length is one, and the entity itself is in the array. But for entities inside an xref, the list contains the blockref path to each instance of the entity.

*[Example O-type to illustrate refDocs manipulation*

```

Unit(1, "mm");

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

PropInt pAllowInsideXref(1, 0, T("|Allow selection inside xref|"));

if (_bOnInsert)
{
    showDialog();
    _Pt0 = getPoint();
}

```

```

Entity entSel = getEntity(T("|Select entity|"),
pAllowInsideXref);
_Entity.append(entSel);
return;
}

// text to be displayed
String strLines[0];
strLines.append("number of entities selected: "+_Entity.length());

if (_Entity.length() > 0)
{
    Entity ent = _Entity[0];
    Element el = (Element)ent;
    if (el.bIsValid())
    {
        strLines.append("el.definition: "+ el.definition());
        strLines.append("el.code: "+ el.code());
    }
    strLines.append("ent.typeDxfName: "+ ent.typeDxfName());
    strLines.append("ent.xrefName: "+ ent.xrefName());
    strLines.append("ent.handle: "+ ent.handle());

    Entity entBlockRef = ent.blockRef();
    if (entBlockRef.bIsValid())
    {
        strLines.append("entBlockRef.handle: " +
entBlockRef.handle());
    }

    {
        Entity ents[] = ent.allBlockRefPaths();
        strLines.append("ent.allBlockRefPaths.length: " +
ents.length());
        for (int e = 0; e < ents.length(); e++)
        {
            Entity ee = ents[e];
            strLines.append("ent["+ + e + "].typeDxfName: " +
ee.typeDxfName());
            strLines.append("ent["+ + e + "].xrefName: " +
ee.xrefName());
            strLines.append("ent["+ + e + "].handle: " +
ee.handle());
        }
    }

    {
        Entity ents[] = ent.blockRefs();
        strLines.append("ent.blockRefs.length: " + ents.length());
        for (int e = 0; e < ents.length(); e++)
        {
            Entity ee = ents[e];

```

```

        strLines.append("br[" + e + "].typeDxfName: " +
ee.typeDxfName());
        strLines.append("br[" + e + "].xrefName: " +
ee.xrefName());
        strLines.append("br[" + e + "].handle: " +
ee.handle());
    }
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

```

—end example]

**String formatObject(String strFormat);** // (added 21.3.10)  
**String formatObject(String strFormat, Map mapAdditionalVariables);** // (added 21.3.10)

The strFormat that is passed in can contain variables in the @() notation that will be resolved in the context of the Entity eg @({Height}) for a GenBeam.

If mapAdditionalVariable is specified, the variable resolving will first use this map to try in the process, before it reverts to the normal resolving process. So using mapAdditionalVariable one can overwrite existing variables, or add additional variables.

```

int acceptObject(String strFilter);
int acceptObject(String strFilter, Map mapAdditionalVariables);
String errorMessage(String strFilter);
String errorMessage(String strFilter, Map mapAdditionalVariables);

```

The strFilter that is passed in can contain variables. These are the same variables that can be used in formatObject, but without the @() notation. An expression engine similar to [NCalc](#) is used which support logical and numerical operators.

If the object is rejected because of an error, the errorMessage return the reason why. If errorMessage returns an empty string, then the filter worked well, and the entity got rejected or accepted. But if there was an invalid filter, a strFilter expression syntax error or a runtime error during evaluation of the filter expression, the errorMessage will explain.

```

int transparency() const; // (added hsbCad21.4.15) transparency percentage, 0 not transparent,
                           100 fully transparent
void setTransparency(int nSet); // (added hsbCad21.4.15) set transparency

```

The methods to manipulate the transparency.

*[Example O-type to illustrate getting and setting of transparency]*

```

if (_bOnInsert) {
    _Entity.append(getEntity()); // select an entity
    _Pt0 = getPoint();
    return;
}
if (_Entity.length()==0) return;

Entity ent = _Entity[0];
if (!ent.bIsValid()) return;

String strChangeEntity = T(" |Change transparency|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    int nVal = getInt(T(" |Enter transparency value, -1=byLayer, -2=byblock. Enter value in range [-2, 100]|"));
    ent.setTransparency(nVal);
}

reportMessage("\nColor: " + ent.color());
reportMessage("\nTransparency: " + ent.transparency());
reportMessage("\nHandle: " + ent.handle());

—end example]

```

### 7.1.1 EntityCollection

The EntityCollection type represents an array of entities, optionally enriched with a coordinate system and a posnum.

An array of EntityCollection is for instance returned by the [Beam](#)::composeBeamPacks method  
**EntityCollection[] Beam::composeBeamPacks(Beam[] arBeamsToPack, String[] arDifferentiators);**

and by the Entity::getEntitiesWithPosnum

**EntityCollection Entity::getEntitiesWithPosnum(int nPosnumToSearchFor);**

---

class **EntityCollection** { // is not derived from Entity

```

GenBeam[] genBeam() const;
Beam[] beam() const;
Sheet[] sheet() const;
Sip[] sip() const;
TslInst[] tslInst() const;
Entity[] entity() const;

void setEntities(Entity[] arEntities);

CoordSys coordSys() const; // transformation from model space to world paper space
void setCoordSys(CoordSys cs);

int posnum() const; // the default value is -1, if not set
void setPosnum(int nVal);

visualize(int indColor = -1) const;
vis(int indColor = -1) const;

};

```

---

*[Example O-type, with insert done in script. The script illustrates the composeBeamPacks as well as the EntityCollection.:]*

```

U(1, "mm");

if (_bOnInsert) {
    _Pt0 = getPoint(); // first select the insertion point
    PrEntity ssE(T("|Select a set of elements|"), Element());
    if (ssE.go())
        _Element = ssE.elementSet();
    return;
}

String arStrColorBy[] = {T("|pack index|"), T("|by posnum|")};
PropString pColorBy(0, arStrColorBy, T("|Color beams by|"));
int bColorByPosnum = (pColorBy==arStrColorBy[1]);

Vector3d vecMove(0,0,0);

if (_Element.length()==0) return;

Beam arBmToPack[0];
String arStrDiffer[0];

for (int e=0; e<_Element.length(); e++) {
    Element el = _Element[e];
    String strEl = el.number();
}

```

---

```

CoordSys csEl = el.coordSys();
Vector3d vecSort = csEl.vecX();

Beam arBm[] = vecSort.filterBeamsPerpendicularSort(el.beam());

for (int b=0; b<arBm.length(); b++) {
    arBmToPack.append(arBm[b]);
    arStrDiffer.append(strEl);
}

if (e==0) {
    vecMove = _Pt0-csEl.ptOrg();
}
}

// EntityCollection arBeamPacks[] =
Beam().composeBeamPacks(arBmToPack);
EntityCollection arBeamPacks[] =
Beam().composeBeamPacks(arBmToPack,arStrDiffer);

Display dp(-1);

for (int i=0; i<arBeamPacks.length(); i++) {
    EntityCollection beamPack = arBeamPacks[i];
    Beam arBeam[] = beamPack.beam();
    reportMessage("\n"+i+": #beams - posnum: "+arBeam.length() + " - "
    "+beamPack.posnum());

    int nColor = i+1;
    if (bColorByPosnum)
        nColor = beamPack.posnum()+1;

    for (int b=0; b<arBeam.length(); b++) {
        Beam bm = arBeam[b];
        Body bd = bm.envelopeBody();
        bd.transformBy(vecMove);
        dp.color(nColor);
        dp.draw(bd);
    }
}

```

*—end example]*

## 7.2 ToolEnt

The term ToolEnt refers to an instance of an hsbCad tool entity, appended to the drawing. All hsbCad tools are derived from this base class. So tenon, drill, dovetail, ... entities are derived from it. You should not confuse with the tools that are added to the beams. A ToolEnt is an entity in the drawing, and has entity properties, can be copied, erased, ...

An entity can be casted into an ToolEnt. However when the casting is not allowed, the resulting ToolEnt will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ToolEnt is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted into an Entity.

During `_bOnInsert`, the `getToolEnt()` function can be used to query the user to select any ToolEnt or ToolEnt derived.

```
ToolEnt getToolEnt();
ToolEnt getToolEnt(String strPrompt);
```

---

```
class ToolEnt : Entity { // see Entity for base functions

    GenBeam[] genBeam() const;
    Beam[] beam() const;
    Sheet[] sheet() const;
    Sip[] sip() const;
    Entity[] entity() const;

    Point3d ptOrg() const;

    void removeGenBeamConnection(GenBeam bmToRemove); // crash fix added in v19.1.95
                                                       and v20.0.66

    // HardWrComp added in hsbCAD2011, build 16.0.18.
    int numHardWrComps() const; // return the number of HardWrComp
    HardWrComp hardWrCompAt(int nInd) const; // get the HardWrComp at index nInd
    HardWrComp[] hardWrComps() const; // get the list of HardWrComp
    void setHardWrComps(HardWrComp[] arHw); // set the list of HardWrComp

    void resetFastenerGuidelines();
    void addFastenerGuideline(FastenerGuideline fgl); // see FastenerGuideline
    FastenerGuideline[] fastenerGuidelines() const; // get the list of FastenerGuideline
    FastenerAssemblyEnt[] getAttachedFasteners() const; // returns the FastenerAssemblyEnt
                                                       attached to this ToolEnt. Added v20.1.14 and v21.0.21.

    Body cuttingBody() const; // returns the body that represents the part that is used for
                           // doing the solid operation, if it exists (added in v19.1.95 and v20.0.66).

    int frozen() const; // the frozen state, returns TRUE or FALSE, normally FALSE (added 23.7.15
                       and 24.0)
    void setFrozen(int bVal); // (added 23.7.15 and 24.0)
};
```

---

---

```
void removeGenBeamConnection(GenBeam bmToRemove);
```

When one of the entries of genBeam is given as argument to this function, the link from the ToolEnt to this GenBeam is removed.

---

[Example O-type TSL:

```
Unit(1, "mm");
if (_bOnInsert) {

    ToolEnt toolent = getToolEnt();
    _Entity.append(toolent);
    _Pt0 = getPoint();

    return;
}

if (_Entity.length() == 0) return;
ToolEnt tent = (ToolEnt)_Entity[0];
if (!tent.bIsValid()) return;

reportMessage("\nNumber of beams: " + tent.beam().length());
```

—end example]

[Example O-type TSL:

```
if (_bOnInsert) {
    _Entity.append(getToolEnt());
    return;
}

if (_Entity.length() == 0) {
    eraseInstance();
    return;
}

ToolEnt tent = (ToolEnt)_Entity[0];

_Pt0 = tent.ptOrg();
_Pt0.vis();
```

—end example]

---

## 7.2.1 HardWrComp

Each [ToolEnt](#) has a list of HardWrComp. The UI in hsbCAD only allows to manage this list of HardWrComp on TsInst as well as on Drill entities.

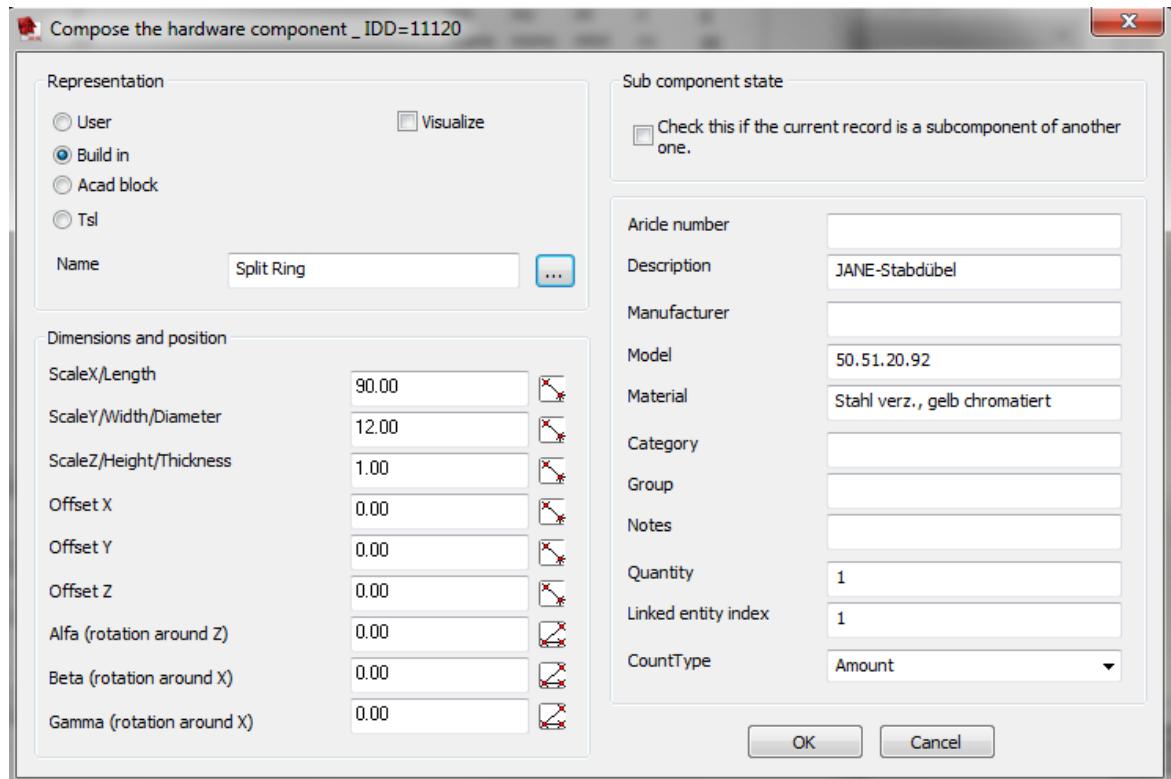
The following methods of ToolEnt access this data:

```
int numHardWrComps() const; // return the number of HardWrComp
HardWrComp hardWrCompAt(int nInd) const; // get the HardWrComp at index nInd
HardWrComp\[\] hardWrComps() const; // get the list of HardWrComp
void setHardWrComps(HardWrComp\[\] arHw); // set the list of HardWrComp
```

A HardWrComp represents a hardware item, possibly containing sub items. Its primary purpose is to export it (through hsbModelX to any external resource). The members are summarized in the following dialog inside hsbCAD.

The name identifies either the build in name (referring to the older Hardware), of a block name, or a Tsl name. In case of a block name or a Tsl name, these are the responsible for the hardware visualization.

The repType, or representation type is one of the following values: [\\_kRTUser](#), [\\_kRTBuildIn](#), [\\_kRTBlock](#) or [\\_kRTTsl](#).




---

class [HardWrComp](#) { // HardWrComp added in hsbCAD2011, build 16.0.18.

```
HardWrComp(String strArticleNumber, int nQuantity);
```

```

String repType() const;
void setRepType(String str);
String name() const;
void setName(String str);
int bVisualize() const;
void setBVisualize(int bSet);

String articleNumber() const;
void setArticleNumber(String str);
String description() const;
void setDescription(String str);
String manufacturer() const;
void setManufacturer(String str);
String model() const;
void setModel(String str);
String material() const;
void setMaterial(String str);
String category() const;
void setCategory(String str);
String group() const;
void setGroup(String str);
String notes() const;
void setNotes(String str);

int quantity() const;
void setQuantity(int nSet);

Entity linkedEntity() const;
void setLinkedEntity(Entity ent); // The ent needs to belong to _GenBeam or _Entity array
                                of the ToolEnt.
String countType() const;
void setCountType(String str);

double dScaleX() const;
void setDScaleX(double dVal);
double dScaleY() const;
void setDScaleY(double dVal);
double dScaleZ() const;
void setDScaleZ(double dVal);

double dOffsetX() const;
void setDOffsetX(double dVal);
double dOffsetY() const;
void setDOffsetY(double dVal);
double dOffsetZ() const;
void setDOffsetZ(double dVal);

double dAngleA() const;
void setDAngleA(double dVal);
double dAngleB() const;
void setDAngleB(double dVal);

```

```

double dAngleG() const;
void setDAngleG(double dVal);

HardWrComp[] subComponents() const;
void setSubComponents(HardWrComp[] arHw);

static String[] getListOfCatalogNames() const; // return the list of catalog entry names (added
// v20.1.59 and v21.0.39)
static HardWrComp[] getHardWrCompsFromCatalog(String strCatalogEntryName); // (added
// v20.1.59 and v21.0.39)

static HardWrComp[] showDialog(HardWrComp[] lstOfHardWrComponents); // (added
23.8.13) The edited list, and if canceled the original list is returned after prompting the
dialog to the user. Only works during insert enabled context.

// The subMapX set of methods added v28.0
Map subMapX(String strKey) const; // see Map
void setSubMapX(String strKey, Map mapNew);
String[] subMapXKeys() const; // return list of available sub map keys
void removeSubMapX(String strKey);
};
```

---

The following is the old (since hsbCAD2011, build 16.0.18), and less flexible way. In this old approach, the hardware is added automatically each time a Hardware is declared in the tsl script. So in order for this to function, before the first Hardware item is attached, the internal array is reset. The new approach works completely different. HardWrComp is a set of data, which can be defined, copied, modified... without changing the internal stored array of HardWrComp. The internal array of the ToolEnt is accessed through the ToolEnt::hardWrComps and related methods.

Beware: Since both Hardware and HardWrComp are internally stored in the same array, when calling the Hardware, the internal array is reset.

So the following is obsolete:

```

Hardware name(String strType, String strDescription, String strModel, double dLen, double
dDiam, int nNumber, String strMaterial);
Hardware name(String strType, String strDescription, String strModel, double dLen, double
dDiam, int nNumber, String strMaterial, String strNotes);
<Hardware>.visualize(<Point3d location>,<Vector3d direction>,[color]);
```

The following is also obsolete:

```

Bolt name(String strDescription, String strModel, double dLen, double dDiam, int nNumber,
String strMaterial);
<Bolt>.visualize(<Point3d location>,<Vector3d direction>,[color]);
```

For the Hardware and Bolt, all properties will appear in DXA file. The dLength and dDiam parameter are also used in visualize function call.

---

*[Example O-type, that reports the hardware attached to \_ThisInst.:*

```

int nLenHw = _ThisInst.numHardWrComps();
for (int i=0; i<nLenHw; i++) {
    HardWrComp hw = _ThisInst.hardWrCompAt(i);
    reportMessage("\nHardWrComp with index: "+i);

    reportMessage("\n      articleNumber: "+hw.articleNumber());
    reportMessage(", name: "+hw.name());
    reportMessage(", quantity: "+hw.quantity());
    reportMessage(", description: "+hw.description());
}

```

*—end example]*

[Example O-type, reads HardWrComp from catalog:

```

PropString pEntry(0,"","");
String strSetFromCatalog = T("|Set entries from catalog|");
addRecalcTrigger(_kContext, strSetFromCatalog );
if (_bOnRecalc && _kExecuteKey==strSetFromCatalog )
{
    String arr[] = HardWrComp().getListOfCatalogNames();
    arr.insertAt(0,"");
    PropString pList(0, arr, "entries in catalog");
    showDialog();

    String strEntry = pList;
    pEntry.set(strEntry);

    HardWrComp hwList[0];
    if (strEntry != "")
    {
        hwList = HardWrComp().getHardWrCompsFromCatalog(strEntry);
    }
    _ThisInst.setHardWrComps(hwList);
}

String strSetManual = T("|Set entries explicit|");
addRecalcTrigger(_kContext, strSetManual );
if (_bOnRecalc && _kExecuteKey==strSetManual )
{
    HardWrComp hwList[0];

    String strArt = "Art-999";
    int nCount = 2;

    HardWrComp hw1(strArt, nCount);
    hw1.setBVisualize(FALSE);
    hw1.setModel("Carriage Bolt");
    hw1.setDescription(scriptName());
    hw1.setName("");
}

```

```

        hwList.append(hw1);

        // assign this list of hardWrComp to this Tsl instance
        _ThisInst.setHardWrComps(hwList);
    }

    String strShowDialog = T("|Show hardware dialog");
    addRecalcTrigger(_kContext, strShowDialog );
    if (_bOnRecalc && _kExecuteKey == strShowDialog )
    {
        HardWrComp hwListCur[] = _ThisInst.hardWrComps();
        HardWrComp hwListNew[] = HardWrComp().showDialog(hwListCur);
        _ThisInst.setHardWrComps(hwListNew);
    }

    int nLenHw = _ThisInst.numHardWrComps();
    for (int i=0; i<nLenHw; i++) {
        HardWrComp hw = _ThisInst.hardWrCompAt(i);
        reportMessage("\nHardWrComp with index: "+i);

        reportMessage("\n      articleNumber: "+hw.articleNumber());
        reportMessage(", name: "+hw.name());
        reportMessage(", quantity: "+hw.quantity());
        reportMessage(", description: "+hw.description());
    }
}

```

*—end example]*

## 7.2.2 FastenerGuideline

FastenerGuideline is a class that allows to specify the guiding geometry to which a [FastenerAssemblyEnt](#) can be attached.

The following methods of ToolEnt access this data:

```

void resetFastenerGuidelines();
void addFastenerGuideline(FastenerGuideline fgl); // see FastenerGuideline
FastenerGuideline[] fastenerGuidelines() const; // get the list of FastenerGuideline

```

---

```

class FastenerGuideline { // added hsbCAD2012 17.0.21

    FastenerGuideline(Point3d ptStart, Point3d ptEnd, double dRadius);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;
}

```

---

```

transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);

void addStep(Point3d ptStart, Point3d ptEnd, double dRadius);
void addStep(Point3d ptStart, Point3d ptEnd, double dRadiusStart, double dRadiusEnd);

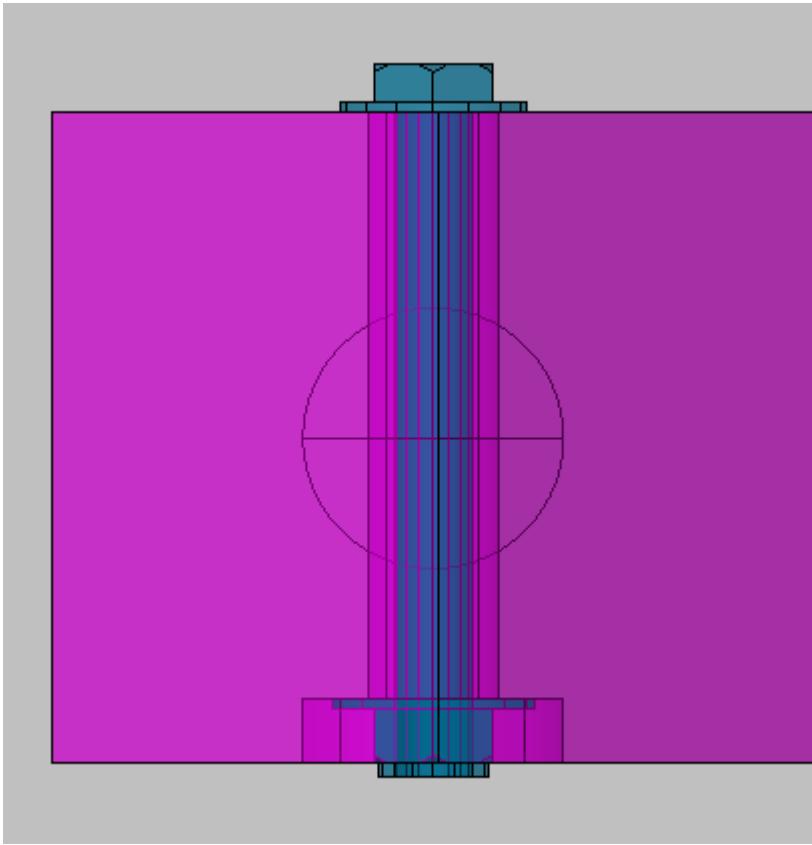
Point3d ptStart() const;
Point3d ptEnd() const;

};

```

---

*[Example E-type that shows the use of the FastenerGuideline. The image shows a Fastener assembly attached to the guide line.*



```

Unit(1, "mm");

PropDouble pDiameter(0, U(20), T("|Diameter|"));
PropDouble pDiameterSink(1, U(40), T("|Sink diameter|"));
PropDouble pDepthSink(2, U(10), T("|Depth sink|"));

double dRad = pDiameter*0.5;

```

---

```

Point3d pts = Pt0+0.5*_W0*_Y0;
Point3d ptE = Pt0-0.5*_W0*_Y0;
Drill dr(pts, ptE , dRad);
Beam0.addTool(dr);

double dRadS = pDiameterSink*0.5;
Vector3d vecD = ptE-pts;
vecD.normalize();
Point3d ptES = pts + vecD*pDepthSink;
Drill drs(pts, ptES , dRadS);
Beam0.addTool(drs);

FastenerGuideline fg(pts, ptE, dRad);
fg.addStep(pts, ptES, dRadS);
ThisInst.resetFastenerGuidelines();
ThisInst.addFastenerGuideline(fg);

—end example]

```

## 7.3 TsInst

The term TsInst refers to an instance of the Tsl type, appended to the drawing. A Tsl instance, also called macro instance, or script instance is an entity in the drawing that instantiates a script, written in this language.

An entity can be casted into a TsInst. However when the casting is not allowed, the resulting TsInst will become invalid. This can be checked with the blsValid() function of Entity.

Because TsInst is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity: eg.:

```
Entity ent = ThisInst;
```

The predefined ThisInst refers to this instance.

During bOnInsert, the getTsInst() function can be used to query the user to select a Tsl instance.

```

TsInst getTsInst();
TsInst getTsInst(String strPrompt);
```

---

```

class TsInst : ToolEnt { // see ToolEnt for base functions

    CoordSys coordSys() const;

    Point3d ptOrg() const;
    void setPtOrg(Point3d ptOrgNew);

    GenBeam[] genBeam();
    Beam[] beam();
    Sheet[] sheet();
```

```

Sip[] sip();
Entity[] entity();

void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, int[] lstPropInt, double[] lstPropDouble, String[] lstPropString);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, int[] lstPropInt, double[] lstPropDouble, String[] lstPropString, int bForceModelSpace);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, int[] lstPropInt, double[] lstPropDouble, String[] lstPropString, int bForceModelSpace, Map map);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, int[] lstPropInt, double[] lstPropDouble, String[] lstPropString, int bForceModelSpace, Map map, String strExecuteKey); // (added 26.5.5)
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, int[] lstPropInt, double[] lstPropDouble, String[] lstPropString, int bForceModelSpace, Map map, String strExecuteKey, int bOnCommandEnded); // bOnCommandEnded default value is FALSE (added 26.5.5)

void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, String strCatalogName, int bForceModelSpace, Map map, String strExecuteKey, String strEvent); // (added hsbCad21.3.23 and v22.0.32)
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[] lstGenBeams, Entity-derived[] lstEntities, Point3d[] lstPoints, String strCatalogName, int bForceModelSpace, Map map, String strExecuteKey, String strEvent, int bOnCommandEnded); // bOnCommandEnded default value is FALSE (added 26.5.5)

int version() const; // returns (major*10000+minor) of the tsl definition (added hsbCad13.2.89).

String[] getListOfPropNames() const; // return the list of property names (added hsbCad13.2.132)

String propIntName(int nIndex) const; // returns propName if the property exists with index nIndex (added hsbCad13.2.132)
int hasPropInt(int nIndex) const; // returns TRUE if the property exists with index nIndex (added hsbCad13.2.132)
int hasPropInt(String strName) const; // returns TRUE if the property exists with OPM name strName (added hsbCad13.2.132)
int propInt(int nIndex) const; // returns the value of the PropInt with index nIndex
int propInt(String strName) const; // returns the value of the PropInt with OPM name strName
void setPropInt(int nIndex, int nVal); // set the value of the PropInt with index nIndex
void setPropInt(String strName, int nVal); // set the value of the PropInt with OPM name strName

String propDoubleName(int nIndex) const; // (added hsbCad13.2.132)
int hasPropDouble(int nIndex) const; // (added hsbCad13.2.132)
int hasPropDouble(String strName) const; // (added hsbCad13.2.132)

```

```

double propDouble(int nIndex) const;
double propDouble(String strName) const;
void setPropDouble(int nIndex, double dVal);
void setPropDouble(String strName, double dVal);

String propStringName(int nIndex) const; // (added hsbCad13.2.132)
int hasPropString(int nIndex) const; // (added hsbCad13.2.132)
int hasPropString(String strName) const; // (added hsbCad13.2.132)
String propString(int nIndex) const;
String propString(String strName) const;
void setPropString(int nIndex, String strVal);
void setPropString(String strName, String strVal);

Point3d gripPoint(int nIndex) const; // returns the grippoint with index nIndex

String scriptName() const; // see also TslScript
void setScriptName() const; // (added hsbCad21.2.9), see also TslScript
String opmName() const; // scriptName, possebly extended with "-" and opmKey.

Map map() const;
void setMap(Map mapNew);

// (added V28) Following methods will return _kOk, _kUpdate. The _kCancel automatically
returns control internally.
// See example in global insert functions
int showDialog(); // see also global insert functions, and Master slave insert.
int showDialog(String strCatalogEntryNameToStartFrom); // returns FALSE which is _kCancel,
if cancel is pressed

void setCatalogFromPropValues(String strCatalogEntryName); // see also global insert
functions
int setPropValuesFromCatalog(String strCatalogEntryName); // return TRUE if successful
String[] getListOfCatalogNames() const; // return the list of catalog entry names for the
opmname of this instance
static String[] getListOfCatalogNames(String strScriptOpmName);
int removeCatalogEntry(String strCatalogEntryName); // return TRUE if successful (added
23.0.74)
int removeCatalogEntry(String strCatalogEntryName, String strScriptOpmName); // return
TRUE if successful (added 23.0.74)

static Map mapWithPropValuesFromCatalog(String strScriptOpmName, String
strCatalogEntryName) const; // added 22.1.63
Map mapWithPropValues(); // (added hsbCAD14.0.78) // see also global insert functions
Map mapWithPropValues(int bOnlyIndexAndValues); // (added V25.1.92) default FALSE, if
TRUE get back a compact map.
int setPropValuesFromMap(Map mapWithPropValues); // return TRUE if successful (added
hsbCAD14.0.78)

static int callMapIO(String strScriptName, Map& mapIO); // Map (added hsbCAD14.0.79)
static int callMapIO(String strScriptName, String strExecuteKey, Map& mapIO); // Map (added
hsbCAD15.1.19)

```

```

int sequenceNumber() const; // The sequence number is used to sort the list of Tsl's during
    // execution of eg OnElementConstructed (added 14.1.35, 15.0.12)
void setSequenceNumber(int nVal); // default 0, can be set to positive or negative value.

int showElementTools() const; // (added 23.8.9) default value is TRUE. If FALSE, the
    // elementTools are not shown as graphics by the tsl.
void setShowElementTools(int nVal);

int debug() const; // the debug state (added 23.0.75 and 22.1.108)
void setDebug(int bVal); // (added 23.0.75and 22.1.108)

// The originator, modelDescription, materialDescription, and additinalNotes are added in
// hsbCAD2011 16.0.31, and are shown as OPM properties.
String originator() const;
void setOriginator(String str);
String modelDescription() const; // Returns the value set with the global method model
void setModelDescription(String str);
String materialDescription() const; // Returns the value set with the global method material
void setMaterialDescription(String str);
String additionalNotes() const;
void setAdditionalNotes(String str);

// the following was added in hsbCAD2011 (build 16.0.41) and hsbCAD2010 (build 15.3.22)
void setDependencyOnPosnumChanged(int bSet); // (see compare) sets to TRUE to make
    // the TslInst recalculate on the event of its posnum being changed

void recalc(); // the following was added in hsbCAD2012 (build 17.2.25) and hsbCAD2013 (build
    // 18.1.29)
void recalcNow(); // the following was added in hsbCAD2013 (build 18.2.11) and hsbCAD2014
void recalcNow(String strExecuteKey); // the following was added in hsbCAD2013 (build
    // 18.2.39) and hsbCAD2014 (build 19.0.60)

int allowGripAtPt0() const; // For the E, O and S Tsl types the grip point at _Pt0 can be
    // suppressed by setting the value to FALSE (added 19.1.81, 20.0.50)
void setAllowGripAtPt0(int bVal); // default TRUE, can be set to FALSE or TRUE.

int posnum() const;
int releasePosnum(); // (added build 21.1.28) release the posnum. If TRUE is passed as
    // argument, the location is reset according to the settings. Return the posnum
    // of the entity (should be -1).
int assignPosnum(int nPosnumToStartSearchingFrom); // (added build 21.1.28) assign a
    // posnum value if the entity does not have one yet. Return the posnum of the
    // entity (should not be -1). The default for bLookForEqual is TRUE.
int assignPosnum(int nPosnumToStartSearchingFrom, int bLookForEqual); // (added build
    // 21.1.28)

void flipAlignZ(); // the following was added in 23.0.23 and build 22.1.39

Grip[] grips() const; // (added to V25.1.83) get the list of Grip
void setGrips(Grip[] arGrip); // (added to V25.1.83) set the list of Grip. BEWARE
    // _ThisInst.setGrips will not work if _Grip is used in the script. Modifications to
    // _Grip have preference.
};


```

**CoordSys** coordSys() const;

Build the coordinate system of the tsl instance. It corresponds with the ECS (entity coordinate system) of the entity (\_PtE, \_XE, \_YE, \_ZE).

**Point3d** ptOrg() const;

Return the \_Pt0.

**void** setPtOrg(**Point3d** ptOrgNew);

Sets the new location of \_Pt0 equal to ptOrgNew.

Remark: This function should not be called on the \_ThisInst. It will not have any effect, since the value will be overwritten at the end of the execution of the script by the contents of \_Pt0.

**Map** map() const;

Return the \_Map contents of that TSL instance.

**void** setMap(**Map** mapNew);

Sets the new contents of \_Map equal to mapNew.

Remark: This function should not be called on the \_ThisInst. It will not have any effect, since the value will be overwritten at the end of the execution of the script by the contents of \_Map.

```
int posnum() const;
int releasePosnum(); // (added build 21.1.28)
int assignPosnum(int nPosnumToStartSearchingFrom); // (added build 21.1.28)
int assignPosnum(int nPosnumToStartSearchingFrom, int bLookForEqual); // (added build
21.1.28)
```

The posnum method returns the posnum value. This value could be -1 if no posnum has been assigned.

Each TsInst can be assigned to a group of identical items. Each of these groups gets a unique positive number called the posnum. If a TsInst is not assigned to such a group, the posnum value of the TsInst is -1. Although identical TsInst can belong to different posnums, different beams can never have the same posnum value.

Whenever a posnum is assigned one can give a nPosnumToStartSearchingFrom. Posnums with a smaller value will not be considered for comparison. If the value of nPosnumToStartSearchingFrom is still available, this value is assigned. If not the first available value that is bigger then nPosnumToStartSearchingFrom is assigned.

Both the posnum, releasePosnum and assignPosnum return the posnum of the TsInst.

To get all the entities with a certain posnum use the method of [Entity::getEntitiesWithPosnum](#).

*[Example E-type:*

```

if (_bOnInsert)
{
    _Entity.append(getTslInst());
    _Pt0 = getPoint();
    return;
}

if (_Entity.length() < 0)
{
    eraseInstance();
    return;
}

TslInst tsl = (TslInst)_Entity[0];
if (!tsl.bIsValid())
{
    eraseInstance();
    return;
}

PropInt nNewVal(0,10,T("|new posnum value|"));

String strAssignPosnum = T("|Assign new posnum|");
addRecalcTrigger(_kContext, strAssignPosnum );
if (_bOnRecalc && _kExecuteKey==strAssignPosnum ) {
    tsl.assignPosnum(nNewVal);
}

String strAssignPosnum2 = T("|Assign new posnum but do NOT look for
equal|");
addRecalcTrigger(_kContext, strAssignPosnum2 );
if (_bOnRecalc && _kExecuteKey==strAssignPosnum2 )
{
    int bLookForEqual = FALSE;
    tsl.assignPosnum(nNewVal, bLookForEqual);
}

String strReleasePosnum = T("|Release posnum|");
addRecalcTrigger(_kContext, strReleasePosnum );
if (_bOnRecalc && _kExecuteKey==strReleasePosnum ) {
    tsl.releasePosnum();
}

String strListEntities = T("|List entity handles|");
addRecalcTrigger(_kContext, strListEntities );
if (_bOnRecalc && _kExecuteKey==strListEntities ) {
    int nPosnum = tsl.posnum();
    EntityCollection col = tsl.getEntitiesWithPosnum(nPosnum);
    TslInst arTsl[] = col.tslInst();
    reportMessage("\n"+arTsl.length()+" TslInst with posnum
"+col.posnum());
}

```

```

        for (int e=0; e<arTsl.length(); e++) {
            String strHan = arTsl[e].handle();
            reportMessage("\n\t"+strHan);
        }
    }

—end sample]

```

```

void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, int[] IstPropInt, double[]
    IstPropDouble, String[] IstPropString);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, int[] IstPropInt, double[]
    IstPropDouble, String[] IstPropString, int bForceModelSpace);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, int[] IstPropInt, double[]
    IstPropDouble, String[] IstPropString, int bForceModelSpace, Map map);
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, int[] IstPropInt, double[]
    IstPropDouble, String[] IstPropString, int bForceModelSpace, Map map, String
    strExecuteKey); // (added 26.5.5)
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, int[] IstPropInt, double[]
    IstPropDouble, String[] IstPropString, int bForceModelSpace, Map map, String
    strExecuteKey, int bOnCommandEnded); // bOnCommandEnded default value is FALSE
    (added 26.5.5)
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, String strCatalogName, int
    bForceModelSpace, Map map, String strExecuteKey, String strEvent); // (added
    hsbCad21.3.23 and v22.0.32)
void dbCreate(String strScriptName, Vector3d vecUcsX, Vector3d vecUcsY, GenBeam-derived[]
    IstGenBeams, Entity-derived[] IstEntities, Point3d[] IstPoints, String strCatalogName, int
    bForceModelSpace, Map map, String strExecuteKey, String strEvent, int
    bOnCommandEnded); // (added hsbCad26.5.5)

```

Create a new database resident instance, insert a new TSL into the Autocad drawing. The TSL will use the script with name strScriptName. Also the type is determined from the script. The \_XU and \_YU of the new instance are determined from vecUcsX and vecUcsY. The new instance has a number of beams, and elements attached to it. The IstPoints determines the \_Pt0 and all grippoints. For the O and E type, the first point of IstPoints is the \_Pt0. For the other types (T,G and X), the \_Pt0 is determined from the beam locations/intersections.

The values for the properties can be set with arrays IstPropInt, IstPropDouble and IstPropString. The other constructor allow to specify a catalog entry.

The script instance that is created will NOT have a \_bOnInsert set.

If the bForceModelSpace value is set to TRUE, the instance will always be created in the model space, even if the current space is paper space.

The passed in map will be put into \_Map of the newly created instance.

The execute key can also be specified.

The strEvent can be "" (empty string means default), "OnDbCreated" (which is default), "OnElementConstructed", "OnRecalc", "OnMapIO"...

When bOnCommandEnded is TRUE (default FALSE), the inserted tsl is only executed, for the first time, on command ended, after all other executing tsIs have done their work. The

TslInstance is available as Entity, and can be stored in the \_Entity array for instance, but should not be queried for its \_Map nor its properties nor should any other action be triggered on it.

*[Example O-type with name dbCreateTsl:*

*The following output is generated with the script below:*

*Handle: 448 ScriptName: 'dbCreateTsl' executeKey: 'aa' event: 'OnElementConstructed'*  
*Handle: 438 ScriptName: 'dbCreateTsl' executeKey: 'Create new instance with catalog values'*  
*event: 'OnRecalc'*

```

U(1, "mm");

if (_bOnInsert) {

    String strScriptName = scriptName(); // name of the script
    Vector3d vecUcsX(1,0,0);
    Vector3d vecUcsY(0,1,0);
    Beam lstBeams[0];
    Element lstElements[0];
    Point3d lstPoints[0];

    int lstPropInt[0];
    double lstPropDouble[0];
    String lstPropString[0];

    lstBeams.append(getBeam()); // will become _Beam0
    lstBeams.append(getBeam()); // will become _Beam1
    lstPoints.append(getPoint()); // will become _Pt0 because S4
E-type is an E type
    lstPoints.append(getPoint()); // will become _PtG[0]
    lstPropInt.append(11); // will become PropInt with index 0

    TslInst tsl;
    tsl.dbCreate(strScriptName, vecUcsX, vecUcsY, lstBeams,
lstElements, lstPoints,
        lstPropInt, lstPropDouble, lstPropString); // create new
instance

    eraseInstance(); // this instance will not stay in the DB, but
the dbCreated will!
    return;
}

String strCreateInstance = T("|Create new instance with catalog
values|");
addRecalcTrigger(_kContext, strCreateInstance );
if (_bOnRecalc && _kExecuteKey==strCreateInstance )
{
    String arNames[] =
TslInst().getListOfCatalogNames(scriptName());
    PropString pCatToRead(2,arNames , T("|Catalog name|"));
    showDialog();

    String strScriptName = scriptName(); // name of the script

```

```

Vector3d vecUcsX(1,0,0);
Vector3d vecUcsY(0,1,0);
Beam lstBeams[0];
Element lstElements[0];
Point3d lstPoints[0];
lstPoints.append(_Pt0 + U(100) * _XU); // locate at a
different location

TslInst tsl;
tsl.dbCreate(strScriptName, vecUcsX, vecUcsY, lstBeams,
lstElements, lstPoints,
    pCatToRead, FALSE, Map(), "aa", "OnElementConstructed"); // 
create new instance
}

PropInt pInt(0, 1, T("|My Prop Int|"));

String strEvent = "";
if (_bOnDbCreated) strEvent = "OnDbCreated"; // default on
dbCreate
if (_bOnElementConstructed) strEvent = "OnElementConstructed";
if (_bOnElementDeleted) strEvent = "OnElementDeleted";
if (_bOnMapIO) strEvent = "OnMapIO";
if (_bOnRecalc) strEvent = "OnRecalc";

reportMessage("\nHandle: " + _ThisInst.handle() + " ScriptName: "
+ scriptName()
    + "' executekey: '" + kExecuteKey + "' event: '" + strEvent +
"')");

```

*—end sample]*

```

int showDialog();
int showDialog(String strCatalogEntryNameToStartFrom);

```

The showDialog allow to the user to set the properties through a dialog during insert (when \_bOnInsert is TRUE) or special events. The showDialog will show each time the insert process takes place.

This routine will only do something during \_bOnInsert. When one TSL calls a dbCreate of another one, it can also call the showDialog of this other tsl, to prompt the user to set the other tsl its values. Can also be called with a catalog value of the other one. The return value is FALSE, if the cancel button was pressed. The return is TRUE if the user pressed the OK button.

When no argument or an empty string is passed as argument, the last inserted values are presented as initial values in the dialog. These values are the values that where present the last time the ok button was pressed. There are also default values defined in the script, during the definition of the properties. That set of values is called the "\_Default" set of values, and they are stored inside the catalog of values for that TSL script.

When the initial value of the properties needs to be retrieved from the catalog, one should use the catalog entry name as argument for the showDialog call. If the entry is not found in the catalog, the \_Default set of values is chosen.

So the following are possible choices:

```
showDialog(""); // empty string means T("_LastInserted")
showDialog(); // no argument means T("_LastInserted")
showDialog(T("_LastInserted"));
showDialog(T("|\_Default|")); // the entry called _Default is used
showDialog(" ---"); // if entry is not found, then T("|\_Default|")
is used
showDialog("myCatalogEntry"); // specific named entry
```

*[Example O-type TSL that inserts another TSL with a specific name:*

```
if (_bOnInsert) {

    String strScriptName = "S4 E-type"; // name of the script
    Vector3d vecUcsX(1,0,0);
    Vector3d vecUcsY(0,1,0);
    Beam lstBeams[0];
    Element lstElements[0];
    Point3d lstPoints[0];

    int lstPropInt[0];
    double lstPropDouble[0];
    String lstPropString[0];

    lstBeams.append(getBeam()); // will become _Beam0
    lstBeams.append(getBeam()); // will become _Beam1
    lstPoints.append(getPoint()); // will become _Pt0 because S4 E-
type is an E type
    lstPoints.append(getPoint()); // will become _PtG[0]

    TslInst tsl;
    tsl.dbCreate(strScriptName, vecUcsX, vecUcsY, lstBeams,
    lstElements, lstPoints,
        lstPropInt, lstPropDouble, lstPropString); // create new
    instance

    tsl.showDialog(); // allows the user to input the values of the
    newly created tsl

    eraseInstance(); // this instance will not stay in the DB
    return;
}
```

*—end example]*

```
int setPropValuesFromCatalog(String strCatalogEntryName);
```

This routine allows to set the property values to the values stored in the catalog for this TSL. This routine is only allowed when the hsbCAD application is loaded (so not in DBX mode). It works nicely together with the \_kExecuteKey value, passed during the Hsb\_ScriptInsert lisp call. Returns TRUE, if the entry was found in the catalog, and the reading was successful.

```
void setCatalogFromPropValues(String strCatalogEntryName);
```

This routine allows writing the property values of this instance to the catalog. This routine is only allowed when the hsbCAD application is loaded (so not in DBX mode).

*[Example O-type TSL that manipulates the properties of another TSL:*

```
PropString pChildCatSave(1, "", "Catalog name child save");
PropString pChildCatRead(2, "", "Catalog name child read");

if (_bOnInsert) {

    showDialog();
    TslInst tsl = getTslInst();

    tsl.setCatalogFromPropValues(pChildCatSave);
    tsl.setPropValuesFromCatalog(pChildCatRead);

    eraseInstance();
    return;
}
```

*—end example]*

```
String opmName() const
```

Return the opmName of this script. The opmName, visible in the selection set in the property manager, is composed of the scriptname, and possibly extended with "-" and an opmKey (see [global functions](#)).

```
String[] getListOfCatalogNames() const; // return the list of catalog entry names for the opmname of this instance
static String[] getListOfCatalogNames(String strScriptOpmName);
```

This routine allows to get the list of all catalog entries for a particular scriptname. There is a catalog for each script. In fact there is a catalog for each individual opmname. Since the opmName of a script is composed of the scriptname, possibly extended with "-" and an opmKey, there can even be different catalogs for a Tsl script with a particular name.

The first function returns the list of catalog entries for this instance. The second function is a static function (this instance is not used), and returns the list for a particular opm name.

[Example O-type TSL that manipulates the properties of another TSL:

```

if (_bOnInsert) {

    TslInst tsl = getTslInst(T("Select Tsl to change properties
from|"));

    PropString pChildCatSave(1,"","Save Catalog name");

    //String arNames[] = tsl.getListOfCatalogNames();
    String arNames[] = TslInst().getListOfCatalogNames(tsl.opmName());
    PropString pChildCatRead(2,arNames , "Read Catalog name");

    showDialog();

    tsl.setCatalogFromPropValues(pChildCatSave); // a blank catalog
name will not do anything
    tsl.setPropValuesFromCatalog(pChildCatRead);

    eraseInstance();
    return;
}

```

—end example]

```

String[] getListOfPropNames() const; // return the list of property names (added hsbCad13.2.132)
String propIntName(int nIndex) const; // returns propName if the property exists with index nIndex
                                         (added hsbCad13.2.132)
int hasPropInt(int nIndex) const; // returns TRUE if the property exists with index nIndex (added
                                         hsbCad13.2.132)
int hasPropInt(String strName) const; // returns TRUE if the property exists with OPM name
                                         strName (added hsbCad13.2.132)
int propInt(int nIndex) const; // returns the value of the PropInt with index nIndex
int propInt(String strName) const; // returns the value of the PropInt with OPM name strName
void setPropInt(int nIndex, int nVal); // set the value of the PropInt with index nIndex
void setPropInt(String strName, int nVal); // set the value of the PropInt with OPM name strName

```

The properties of a tsl are accessible. Each of the properties can be retrieved or set, both by index or by name. A list of all properties of the tsl is available through the getListOfPropNames. Individual methods need to be used for int, double and string properties.

[Example O-type TSL that manipulates the properties of another TSL similar to the quick select:

```

Unit(1, "mm");
double dEps = U(0.00001); // tolerance used in double comparison

if (_bOnInsert) {

    // first select all the Tsl's that you want to consider to change
    properties
}

```

```

TslInst arTsl[0];
PrEntity ssE(T("|Select the Tsls to investigate|"), TslInst());
if (ssE.go()) {
    Entity ents[] = ssE.set();
    for (int e=0; e<ents.length(); e++) {
        TslInst tsl = (TslInst) ents[e];
        if (tsl.bIsValid())
            arTsl.append(tsl);
    }
    reportMessage (T("\n|Number of Tsls selected:| ") +
arTsl.length());
}

// collect list of tsl names, and properties
String arTslName[0];
String arTslPropName[0];
for (int t=0; t<arTsl.length(); t++) {
    TslInst tsl = arTsl[t];

    // collect the tsl name
    String strName = tsl.opmName();
    if (arTslName.find(strName)<0)
        arTslName.append(strName);

    String arPropName[] = tsl.getListOfPropNames();
    for (int p=0; p<arPropName.length(); p++) {
        String strPropName = arPropName[p];
        if (arTslPropName.find(strPropName)<0)
            arTslPropName.append(strPropName);
    }
}

String arTypes[] = {T("|Integer|"), T("|Double|"), T("|String|")};

PropString pName(0,arTslName,T("|Select Tsl type|"));
PropString pPropName(1,arTslPropName,T("|Select property|"));
PropString pPropType(2,arTypes,T("|Select property type|"));
PropString pPropOldValue(3,"",T("|Prop old value|"));
PropString pPropNewValue(4,"",T("|Prop new value|"));
showDialog();
int nType = arTypes.find(pPropType,0); // nType = 0,1,2

// now loop over the properties, and change the values that match
int nNumValuesChanged = 0;
for (int t=0; t<arTsl.length(); t++) {
    TslInst tsl = arTsl[t];

    // check the tsl name
    String strName = tsl.opmName();

```

```

        if (strName != pName)
            continue; // jump to next t

        // check the property name and type
        if (nType==0 && tsl.hasPropInt(pPropertyName)) {
            int nOldVal = tsl.propInt(pPropertyName);
            if (nOldVal==pPropOldValue.Atoi()) { // check if the value
is the same
                tsl.setPropInt(pPropertyName,pPropnewValue.Atoi());
                nNumValuesChanged++;
            }
        }

        // check the property name and type
        else if (nType==1 && tsl.hasPropDouble(pPropertyName)) {
            double dOldVal = tsl.propDouble(pPropertyName);
            if (abs(dOldVal-pPropOldValue.Atof())<dEps) { // check if
the value is the same
                tsl.setPropDouble(pPropertyName,pPropnewValue.Atof());
                nNumValuesChanged++;
            }
        }

        // check the property name and type
        else if (nType==2 && tsl.hasPropString(pPropertyName)) {
            String strOldVal = tsl.propString(pPropertyName);
            if (strOldVal==pPropOldValue) { // check if the value is the
same
                tsl.setPropString(pPropertyName,pPropnewValue);
                nNumValuesChanged++;
            }
        }

    }

    reportMessage(T("\n|Number of properties changed:| "))
+nNumValuesChanged);

eraseInstance();
return;
}

```

*—end example]*

**Map** mapWithPropValuesFromCatalog(**String** strScriptOpmName, **String** strCatalogEntryName)  
const // (added 22.1.63)

Get the property values from a catalog of an Tsl specified by its opm name.

[Example O-type looks to the properties in the catalog of another tsl:]

```
Unit(1, "mm");
if (_bOnInsert) {

    TslInst tsl = getTslInst(); // select a tsl
    String strOpmName = tsl.opmName();
    String arCatNames[] = tsl.getListOfCatalogNames(strOpmName);

    PropString pCatalogName(0, arCatNames, T("|Catalog name|"));
    showDialog();

    // get the map filled with properties of the tsl
    Map mpProps =
    TslInst().mapWithPropValuesFromCatalog(strOpmName, pCatalogName);

    // write the map file as dxx
    String strFileMap = "c:\\Temp\\\" + strOpmName + "_" +
    pCatalogName + ".dxx";
    mpProps.writeToDxxFile(strFileMap);

    // call the viewer as property editor
    spawn("", _kPathHsbInstall+"\\Utilities\\hsbMapExplorer\
    \\hsbMapExplorer.exe", strFileMap, ""); // wait

    eraseInstance();
    return;
}
```

—end example]

```
Map mapWithPropValues(); // (added hsbCAD14.0.78) // see also global insert functions
int setPropValuesFromMap(Map mapWithPropValues); // return TRUE if successful (added
hsbCAD14.0.78)
```

These routines allow reading and writing of the property values of this instance to a [Map](#).

[Example O-type TSL that manipulates the properties of another TSL through the call of an external viewer:]

```
Unit(1, "mm");

if (_bOnInsert) {

    TslInst tsl = getTslInst(); // select a tsl

    // get the map filled with properties of the tsl
    Map mpProps = tsl.mapWithPropValues();

    // write the map file as dxx
    String strFileMap = "c:\\Temp\\\" + scriptName() + ".dxx";
    mpProps.writeToDxxFile(strFileMap);
```

```

    // call the viewer as property editor
    spawn("",_kPathHsbInstall+"\Utilities\hsbMapExplorer\
\hsbMapExplorer.exe", strFileMap, ""); // wait

    // read the map from the file
    mpProps = Map(); // reset content
    mpProps.readFromDxxFile(strFileMap);

    // set the properties back to the tsl
    tsl.setPropValuesFromMap(mpProps);

    eraseInstance();
    return;
}

```

*—end example]*

```

static int callMapIO(String strScriptName, Map& mapIO); // Map _bOnMapIO (added
hsbCAD14.0.79)
static int callMapIO(String strScriptName, String strExecuteKey, Map& mapIO); // Map (added
hsbCAD15.1.19)

```

This method allows to call another Tsl as worker Tsl. The argument mapIO is modified after the call. The other tsl, specified by strScriptName, is called with the \_bOnMapIO set to TRUE. The contents of mapIO is available as \_Map.

*[Example O-type TSL that illustrates the callMapIO:*

```

// compose map
Map mapIO;
mapIO.setDouble("dVal",10);
mapIO.setDouble("dVal2",11);

int bOk = TslInst().callMapIO("CallMapIO_Worker", mapIO);
if (!bOk) {
    reportMessage("\ncallMapIO returned false.");
}
else {
    reportMessage("\ncallMapIO returned true.");
}

if (_bOnDbCreated) {
    // for debug view the map file
    String strFileMap = "c:\\Temp\\" + scriptName() + ".dxx";
    mapIO.writeToDxxFile(strFileMap);
    spawn("",_kPathHsbInstall+"\Utilities\hsbMapExplorer\
\hsbMapExplorer.exe", strFileMap, ""); // wait
}
else {
    String strMap = mapIO.getDxContent(FALSE);
}

```

```

        reportMessage("\nmapIO contents begin:\n");
        reportMessage(strMap);
        reportMessage("\nmapIO contents end:");
    }
}

```

**—end example]**

[Example O-type TSL that illustrates the called Tsl in the example above. The Tsl should be available with name "CallMapIO\_Worker".

```

if (_bOnMapIO) {

    String strKey = "dVal"; // key in _Map
    if (_Map.hasDouble(strKey)) {
        reportNotice("\n_Map contains "+strKey);
    }
    else {
        reportNotice("\n_Map does not contain key: "+strKey);
    }

    // interpret the arguments
    double dVal = _Map.getDouble("dVal");
    double dVal2 = _Map.getDouble("dVal2");
    _Map = Map(); // reset map contents

    // do the work
    double dValNew = dVal+dVal2;

    // return the results
    _Map.setDouble("dValNew", dValNew);

    // do not call eraseInstance(), otherwise no _Map changes are
    accepted
    return;
}

—end example]

```

```

int sequenceNumber() const; // added hsbCAD2009+ (build 14.1.35), hsbCAD2010 (build 15.0.12)
void setSequenceNumber(int nVal);

```

The value of the sequenceNumber is used to sort the Tsl's during grouped execution. This grouped execution takes place for instance after generate construction during the event \_bOnElementConstructed. This grouped execution also takes place when an element is set on a viewport. The list of tsl's is sorted from low to high sequenceNumber. The default value of a TslInst is 0. Negative values will come before the 0. So the sequenceNumber can be positive and negative.

KR: Could you please update the TSL help for the TslInst class to state that the sequenceNumber will also be used for governing the execution order of TSLs when an element is set on a viewport.

```
void recalc(); // the following was added in hsbCAD2012 (build 17.2.25) and hsbCAD2013 (build 18.1.29)
```

The recalc method appends this tsl to the list of entities that needs to be recalculated on the command-ended-event of autocad. A tsl can be triggered for recalculation by calling recalc on it as many times as you like. Finally, it will only be appended once to the list of entities. As such it will only be fired once on command ended. Even during command ended execution a duplicate cannot be added. So circular loops are not possible.

The other alternative to trigger a tsl for recalculation is to call transformBy(Vector(0,0,0)) on it. The difference with the recalc method is that the transformBy is executed immediately. So the recalculation action is not postponed to the command-ended-event. The draw back of this transformBy is that you might end up in circular dependency loops.

```
void recalcNow(); // the following was added in hsbCAD2013 (build 18.2.11) and hsbCAD2014
```

The recalcNow method fires the recalculation at the moment of the call.

*[Example O-type TSL that illustrates the recalcNow in a scenario where the recalc would not work].*

```
Unit(1, "mm");

if (bOnInsert) {

    TslInst tsl;
    Entity set[] = PrEntity().getPickFirstSS();
    if (set.length()>0) {
        tsl = (TslInst)set[0];
    }

    Pt0 = getPoint();
    Beam.append(getBeam());

    if (!tsl.bIsValid() || !tsl.hasPropDouble("dW")) {

        return;
    }

    int nCase = 2;
    if (nCase == 0) {
        tsl.setPropDouble("dW", tsl.propDouble("dW")*1.2);
        //reportMessage("\nTsl selected propDouble dW:
        "+tsl.propDouble("dW"));
    }
    else if (nCase == 1) {
        Point3d ptNew = getPoint("\\nNew location");
        tsl.setPtOrg(ptNew);
    }
    else if (nCase == 2) {
        String str = Beam[0].label() + "A";
    }
}
```

```

        _Beam[0].setLabel(str);
        //tsl.recalc(); // is fired at command ended
        tsl.recalcNow(); // is fired during insert
    }

    Point3d ptDoNothing = getPoint("\nWaiting on point input");

    return;
}

if (!_Beam0.bIsValid()) {
    eraseInstance();
    return;
}

PropDouble dW(0,U(100));
PropDouble dH(1,U(50));

PLine pl(_Pt0, _Pt0+dW*_YU, _Pt0+dW*_YU + dH*_XU, _Pt0+dH*_XU);

Display dp(-1);
dp.textHeight(U(20));
dp.draw(pl);

String str = _Beam[0].label();
dp.draw(str, _Pt0, _XU, _YU, 1,1);

—end example]

```

**void flipAlignZ();** // the following was added in 23.0.23 and build 22.1.39

The flipAlignZ method is only active for T and E types. It flips the align Z property also shown in the property manager.

[sample]

```

String strFlip = T("|Flip Z alignment|");
addRecalcTrigger(_kContext, strFlip );
if (_bOnRecalc && _kExecuteKey==strFlip )
{
    _ThisInst.flipAlignZ();
    setExecutionLoops(2); // add recalc if calling on _ThisInst,
such that predefines are recalculated
    return;
}
—end sample]

```

---

[Example O-type TSL that illustrates the filling of a catalog with hardcoded defaults".

```

Unit(1,"mm");

```

```

PropInt pInt(0, 1, "my first int");
PropString pString(0, "Boe!", "my first string");
PropDouble pDouble(0, U(123), "my first double");

if (_bOnInsert)
{
    showDialog();
    _Pt0 = getPoint();
    return;
}

String strFillCatalogWithDefaults = T("|Fill catalog with
defaults|");
addRecalcTrigger(_kContext, strFillCatalogWithDefaults );
if (_bOnRecalc && _kExecuteKey==strFillCatalogWithDefaults )
{
    // store current state of properties in catalog
    setCatalogFromPropValues("_LastUsed");

    pInt.set(20);
    pString.set("Boe 20");
    pDouble.set(0.20);
    setCatalogFromPropValues("Boe20");

    pInt.set(30);
    pString.set("Boe 30");
    pDouble.set(0.30);
    setCatalogFromPropValues("Boe30");

    // restore current state of properties from catalog
    setPropValuesFromCatalog("_LastUsed");
}

```

*—end example]*

## 7.4 GenBeam

Beam, Sheet and Sip are all derived from the type GenBeam. A GenBeam is a structural Hsb entity, that has a center location in space, with a coordinate system, a width, height and length. So a GenBeam can be seen as a box, located in space. The term GenBeam comes from generic beam.

A GenBeam has also a large set of functions. All functions of GenBeam are available for Beam, Sip and Sheet types.

The GenBeam type is derived from Entity. This means that a GenBeam instance has all functions from the [Entity](#) type.

The GenBeam class serves as a base class for the real entities Beam, Sheet and Sip. You will not find a GenBeam entity as such, in the drawing. But you could find Beams, Sheets and Sips. Since these types are derived from GenBeam, they are GenBeams. It is the same kind of relationship as with Entity. E.g. each Autocad entity has a color, each GenBeam has a width.

The `GenBeam::isotropic()` returns one of the following predefined variables of type `int`:  
`_kUndefinedIsotropic`, `_kIsotropic`, `_kNonIsotropic`, `_kMultiPlexIsotropic`

---

```
class GenBeam : Entity { // see Entity for base functions

    Vector3d vecX() const; // corresponds to the _XF?
    Vector3d vecY() const; // corresponds to the _YF?
    Vector3d vecZ() const; // corresponds to the _ZF?
    Point3d ptCen() const; // center point on the beam axis

    double dL() const; // dimension in vecX direction
    double dW() const; // dimension in vecY direction, corresponds to the _WF?
    double dH() const; // dimension in vecZ direction, corresponds to the _HF?

    Point3d ptRef() const; // fixed point on the X axis

    CoordSys coordSys() const; // Gets a coordinate system with vecX, vecY, vecZ and ptRef.
                                // (added v19.1.104 and v20.0.80)

    Point3d ptCenSolid() const; // The center point of the solid. This point works together with
                                // the solidLength.
    double solidLength() const; // real length (in vecX direction) of the solid
    double solidWidth() const; // real width (in vecY direction) of the solid
    double solidHeight() const; // real height (in vecZ direction) of the solid
    Point3d ptCentreOfGravity() const; // center point on the beam axis (added v23.0.39)

    Quader quader() const; // return the quader defining the envelope box. The ptCen with
                            // the refOffset in case of Beam and dL are used. (see Quader).
    Quader quader(int bFromSolid) const; // return the quader defining the envelope box. If
                                            // the bFromSolid is true, the ptCenSolid and solidLength are used, otherwise
                                            // the ptCen and dL are used.

    double dLMin() const; // distance in vecX direction from ptRef, corresponds to the
                            // _LMin?
    double dLMax() const; // distance in vecX direction from ptRef, corresponds to the
                            // _LMax?

    Vector3d vecD(Vector3d vecDirection) const; // find the vecY or vecZ most aligned with
                                                // the vecDirection argument
    double dD(Vector3d vecDirection) const; // find the dWidth or dHeight in the direction
                                                // most aligned with the vecDirection argument

    String name("grade") const; // grade (OPM)
    String name("profile") const; // extrusion profile (OPM)
    String name("material") const; // material (OPM)
    String name("label") const; // label (OPM)
    String name("sublabel") const; // sublabel (OPM)
    String name("sublabel2") const; // sublabel2 (OPM)
    String name("beamcode") const; // hsb internally used beam code (OPM)
    String name("type") const; // hsb assigned type of beam: rafter, post, ... as string
    String name("information") const; // information (OPM)
```

---

```

String name("name") const; // name of beam assigned during creation (OPM)
String name("posnumandtext") const; // displayed posnum and text
String name("layer") const; // layer name on which the beam resides
String name("hsbld") const; // hsb assigned generation code
String name("module") const; // hsb assigned module name

double volume() const; // volume in du^3

int posnum() const; // posnum value, could be -1 if no posnum is assigned
int releasePosnum(int bResetLocationToo); // (added build 16.1.18) release the posnum. If
    // TRUE is passed as argument, the location is reset according to the settings.
    // Return the posnum of the entity (should be -1).
int assignPosnum(int nPosnumToStartSearchingFrom); // (added build 16.1.18) assign a
    // posnum value if the entity does not have one yet. Return the posnum of the
    // entity (should not be -1). The default for bLookForEqual is TRUE.
int assignPosnum(int nPosnumToStartSearchingFrom, int bLookForEqual); // (added
    // bLookForEqual build 19.1.47)

int blsDummy() const; // check if beam is a dummy beam
void setBlsDummy(int nSet); // sets the dummy state to TRUE or FALSE

int type() const; // hsb assigned type as int. This int can be used as index into
    // _BeamTypes
void setType(int nType); // sets the type as int

String label() const; // get the label
void setLabel(String strLabel); // sets the label
String subLabel() const; // get the sub label
void setSubLabel(String strLabel); // sets the sub label
String subLabel2() const; // get the sub label 2
void setSubLabel2(String strLabel); // sets the sub label 2
String grade() const; // get the value
void setGrade(String strValue); // sets the new value
String information() const; // get the value
void setInformation(String strValue); // sets the new value
String material() const; // get the value
void setMaterial(String strValue); // sets the new value
String beamCode() const; // get the value
void setBeamCode(String strValue); // sets the new value
String name() const; // get the value
void setName(String strValue); // sets the new value
String module() const; // get the value
void setModule(String strValue); // sets the new value
String hsbld() const; // get the value
void setHsbld(String strValue); // sets the new value

void setPtCtrl(Point3d ptCtrl, Vector3d vecDirection);

addTool(Tool tl);
addTool(Tool tl, int nStretch); // nStretch can be _kStretchNot, _kStretchOnInsert or
    // _kStretchOnToolChange
addToolStatic(Tool tl);

```

```

addToolStatic(Tool tl, int nStretch); // nStretch can be _kStretchNot, _kStretchOnInsert or
                                         _kStretchOnToolChange

copyToolsFrom(GenBeam genBeamSource); // added v23.7.4, see example below
copyToolsFrom(GenBeam genBeamSource, CoordSys transform);
copyToolsFrom(GenBeam genBeamSource, CoordSys transform, int nStretch);

int removeToolsStaticOfType(Tool tl); // all static tools of the same type as tl will be
                                         deleted from the GenBeam. This method was called removeToolsStatic
                                         renamed since v19.1.103 and v20.0.76. New method returns the amount of
                                         tools removed.
int removeToolStatic(Tool tl); // remove single static tools that matches given tl. It will be
                                         deleted from the GenBeam. Added v19.1.103 and v20.0.76. Method returns
                                         the amount of tools removed. The tool matching uses an internal differ
                                         method which currently is only supported by certain tools: Drill.

GenBeam dbCopyShape() const;

Point3d[] dimPoints(DimLine line, int nType) const;

// Body realBody() const; // Body now moved to the base class Entity
Body envelopeBody() const; // bUseExtProf default to FALSE, bApplyCuts default to
                           FALSE, blncludeExtrusionProfileTongueHeight default FALSE
Body envelopeBody(int bUseExtProfile, int bApplyCuts) const; //
                           blncludeExtrusionProfileTongueHeight default FALSE
Body envelopeBody(int bUseExtProfile, int bApplyCuts, int
                           blncludeExtrusionProfileTongueHeight) const; // (hsbcad 21.1.29) the extrusion
                                         profile tongue height needs to be set to see effect

Entity[] eToolsConnected() const; // return list of all etools operating on this genbeam. All
                                         etools (including TSL's) that add a tool to this genbeam.

TslInst subAssembly() const; // return the TslInst s-type instance that this genbeam
                                         belongs to.
TslInst[] subAssemblies() const; // return the list of TslInst s-type instances that this
                                         genbeam belongs to. (added hsbCAD2009 build 13.6.14 and hsbCAD2009+ build
                                         14.0.69)

GenBeam[] filterGenBeamsNotInSubAssembly(GenBeam[] arGenBeams) const; // return the
                                         list of genbeams that do not belong to any s-type Tsl. This Tsl is not
                                         considered.
GenBeam[] filterGenBeamsNotThis(GenBeam[] arGenBeams) const; // return the list of
                                         genbeams that are not this genbeam

Entity panhand() const; // retrieve the panhand of the genbeam. The panhand is used to
                                         identify the genbeam not to be removed with the element generation.
void setPanhand(Entity ent); // sets the panhand to the Entity. The panhand must be set to
                                         a valid entity for the beam to be protected from erasing during delete and
                                         generate construction.

CncExport[] getToolsOfTypeCncExport() const; // return the list of CncExport tools that
                                         belong to this GenBeam. The list is a copy of the internal tools . Changing
                                         the tools, will not change the GenBeam contents.

```

```

CncExportI getToolsOfTypeCncExport(String strToolName) const; // added v22.1.13

FreeProfileI getToolsOfTypeFreeProfile() const; // return the list of FreeProfile tools that
belong to this GenBeam. The list is a copy of the internal tools . Changing
the tools, will not change the GenBeam contents. (added v21.4.51 and
v22.0.98)
DrillI getToolsStaticOfTypeDrill() const; // return the list of Drill tools that belong to this
GenBeam. The list is a copy of the internal tools . Changing the tools, will
not change the GenBeam contents. (added v19.1.103 and v20.0.76)
CutI getToolsStaticOfTypeCut() const; // return the list of Cut tools that belong to this
GenBeam. The list is a copy of the internal tools . Changing the tools, will
not change the GenBeam contents. (added v20.0.114 and v21.0.5)
BeamCutI getToolsStaticOfTypeBeamCut() const; // (since v21.0.103)

AnalysedToolI analysedTools() const; // return the list of AnalysedTool tools that belong
to this GenBeam.
AnalysedToolI analysedTools(int nVhsb_runerboseMode) const; // return the list of
AnalysedTool tools that belong to this GenBeam. If nVerboseMode==1,
errors will be outputted with reportMessage. If nVerboseMode==2, then
errors of the analysedTool generation are outputted through reportNotice. If
nVerboseMode==0, which is the default, no reporting is done.
AnalysedToolI analysedToolsFromEnvelope(int nVerboseMode) const; // (method added
18.1.38) The envelope of Sip and Sheets is described using
AnalysedBeamCuts and AnalysedCuts, including the openings.

FastenerAssemblyEntI getAttachedFasteners() const; // returns the FastenerAssemblyEnt
of the eToolsConnected ToolEnt. Added v20.1.14 and v21.0.21.

Body cuttingBodyOfToolEnt(Entity eToolEnt) const; // (since v21.1.33) Find the cutting Body
of the tools added by the specified ToolEnt.

int isotropic() const; // (added 21.1.34) int value, corresponding to the predefines
                  _kUndefinedIsotropic, _kIsotropic, _kNonIsotropic, _kMultiPlexIsotropic
void setIsotropic(int nIsotropic); // (added 21.1.34) sets the isotropic as int

int loadBearing() const; // (added 26.1.6) int value TRUE or FALSE.
};

```

---

```

Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;
Point3d ptCen() const;

```

Return the axis system and the center point of the GenBeam.

```

Point3d ptRef() const;
double dLMin() const;
double dLMax() const;

```

The ptRef is a fixed reference point on the axis of the beam. It is NOT the center point of the beam, but it is somewhere on the axis of the beam. The dLMin and dLMax are measured from this point along the vecX on the beam. If you move the endcut of a beam, with the grip point, the ptRef will stay fixed, but the dLMin or dLMax might change, as well as the ptCen.

**double solidLength() const;**

The length of the solid might differ from the dL() length a little bit, because the dL() is calculated from the location of the tool, and solidLength is calculated from the actual solid with all its details. Use the routine solidLength with care. Tool calculations should be done based on dL. The dL is much cheaper to be calculated, especially when multiple tools are present. The reason is that, whenever the solidLength is called, the solid is recalculated (if it is not present). This is an expensive operation. Adding a tool to a beam, will make the solid invalid. So adding multiple tools that use the solidLength, will result in multiple recalculations of the solid. That is in contrast with the use of dL. The dL is an approximate solid length, that is calculated from the endtool cuts, and endtool locations. In all the build-in hsb tools we use the dL. The solidLength should only be used for outputting the real length of the beam, and not for tool calculations.

**GenBeam dbCopyShape() const;**

Because GenBeam is derived from Entity, the dbCopy function is also available. The difference with dbCopyShape is that tools, dynamic tool links will not be copied with the dbCopyShape. This call will only copy the shape of the GenBeam.

**int type() const;**

Returns the type of the beam as an enumerated value. Possible values of type are \_kRafter, ... The complete list of beamtypes is given in **\_BeamTypes**. The possible values for this type starts with value 0, and runs consecutive until the last value. Therefor, the value that this function returns can be used as an index into the **\_BeamTypes** array. To have a property with all the possible beamtypes:

**PropString** pTypes(0, **\_BeamTypes**, "Beam types");

To retrieve the type from a given string, one can do a find in the **\_BeamTypes** array:

**int type = \_BeamTypes.find(pTypes);**

*[Example O-type:*

**PropString** pTypes(0, **\_BeamTypes**, "Beam types");

```
int type = _BeamTypes.find(pTypes);
reportMessage("\n"+type);
```

```
int arPredefTypes[] = {
    _kRafters,
    _kTopPlate,
    _kMidPlate,
    _kRidge,
    _kValleyRafters,
    _kHipRafters,
    _kPost,
    _kSlopedRafters,
    _kTieBeam,
```

```
_kHiddenRafters,
_kBlocking,
_kBrace,
_kBeam,
_kValleyBoard,
_kDummyBeam,
_kSpline,
_kElementModule,
_kOpeningWrap,
_kHeader,
_kPanel,
_kSheeting,
_kTrimmerStud,
_kKingPost,
_kCrippleStud,
_kSFTopPlate,
_kSFBottomPlate,
_kSFBlocking,
_kStud,
_kSill,
_kPurlin,
_kKingStud,
_kLog,
_kPanelTopPlate,
_kPanelLateralPlate,
_kPanelBottomPlate,
_kPanelCapStrip,
_kPanelTrimmerStud,
_kPanelFillerLetIn,
_kPanelKingStud,
_kPanelCrippleStud,
_kPanelEmbeddedLumber,
_kPanelPressureTreatedPlate,
_kPanelRidgePerimeter,
_kPanelEavePerimeter,
_kPanelPerimeter,
_kPanelSplineLumber,
_kPanelHipPerimeter,
_kPanelValleyPerimeter,
_kFloorBeam,
_kDiagonalStud,
_kAnyType,
_kTypeNotSet,
_kSFStudLeft,
_kSFStudRight,
_kSFAngledTPLeft,
_kSFAngledTPRight,
_kSFSupportingBeam,
_kSFTransom,
_kSFJackOverOpening,
_kSFJackUnderOpening,
_kSFPacker,
_kSFFVent,
```

```
_kDakBackEdge,
_kDakFrontEdge,
_kDakLeftEdge,
_kDakRightEdge,
_kDakCenterJoist,
_kLath,
_kExtraRafter,
_kCantileverBlock,
_kSupportingCrossBeam,
_kExtraBlock,
_kValleyLayBoard,
_kEaveLath,
_kRoofPost,
_kThroughPost,
_kNormalPost,
_kBasementBeam,
_kDiagonalBasementBeam,
_kPillowBeam,
_kRisingBeam,
_kPedimentRafter,
_kPedimentGable,
_kRoofSupport,
_kLocatingPlate,
_kSFSolePlate,
_kDecking,
_kJoist,
_kLedger,
_kRimJoist,
_kRimBeam,
_kStringer,
_kTread,
_kRiser,
_kSFTopHeaderSill,
_kSFBottomHeaderSill,
_kSFTopShim,
_kSFBottomShim,
_kSFLeftShim,
_kSFRightShim,
_kSFVeryTopPlate,
_kSFVeryTopSlopedPlate,
_kSFVeryBottomPlate,
_kSFBottomSlopedPlate,
_kSFVeryBottomSlopedPlate,
_kSFVeryTopDropInPlate,
_kSFSubheader,
_kSFSubsill,
_kSFWindowBlocking,
_kTRTop_Chord,
_kTRBottom_Chord,
_kTRWeb,
_kTRWedge,
_kTRSilder,
_kTRCosmetic,
```

```
_kTRStrongbacks,
_kTRRibbons,
_kTRTruss_Bracing,
_kEWPBlocking_Panel,
_kEWPSquash_Block,
_kEWPPBacker_Block,
_kEWPPWeb_Stiffener,
_kEWPFFlush_Beam,
_kEWPDropped_Beam,
_kEWPAccessories,
_kEWPCantilever_Filler,
_kEWPFiller_Block,
kBatten,
_kBoardSheeting,
_kRoofSheeting
};

String arPredefTypesString[] =
{
    "_kRafters",
"_kTopPlate",
"_kMidPlate",
"_kRidge",
"_kValleyRafters",
"_kHipRafters",
"_kPost",
"_kSlopedRafters",
"_kTieBeam",
"_kHiddenRafters",
"_kBlocking",
"_kBrace",
"_kBeam",
"_kValleyBoard",
"_kDummyBeam",
"_kSpline",
"_kElementModule",
"_kOpeningWrap",
"_kHeader",
"_kPanel",
"_kSheeting",
"_kTrimmerStud",
"_kKingPost",
"_kCrippleStud",
"_kSFTopPlate",
"_kSFBottomPlate",
"_kSFBlocking",
"_kStud",
"_kSill",
"_kPurlin",
"_kKingStud",
"_kLog",
"_kPanelTopPlate",
"_kPanelLateralPlate",
```

```
"_kPanelBottomPlate",
"_kPanelCapStrip",
"_kPanelTrimmerStud",
"_kPanelFillerLetIn",
"_kPanelKingStud",
"_kPanelCrippleStud",
"_kPanelEmbeddedLumber",
"_kPanelPressureTreatedPlate",
"_kPanelRidgePerimeter",
"_kPanelEavePerimeter",
"_kPanelPerimeter",
"_kPanelSplineLumber",
"_kPanelHipPerimeter",
"_kPanelValleyPerimeter",
"_kFloorBeam",
"_kDiagonalStud",
"_kAnyType",
"_kTypeNotSet",
"_kSFStudLeft",
"_kSFStudRight",
"_kSFAngledTPLeft",
"_kSFAngledTPRight",
"_kSFSupportingBeam",
"_kSFTransom",
"_kSFJackOverOpening",
"_kSFJackUnderOpening",
"_kSFPacker",
"_kSFSFVent",
"_kDakBackEdge",
"_kDakFrontEdge",
"_kDakLeftEdge",
"_kDakRightEdge",
"_kDakCenterJoist",
"_kLath",
"_kExtraRafters",
"_kCantileverBlock",
"_kSupportingCrossBeam",
"_kExtraBlock",
"_kValleyLayBoard",
"_kEaveLath",
"_kRoofPost",
"_kThroughPost",
"_kNormalPost",
"_kBasementBeam",
"_kDiagonalBasementBeam",
"_kPillowBeam",
"_kRisingBeam",
"_kPedimentRafters",
"_kPedimentGable",
"_kRoofSupport",
"_kLocatingPlate",
"_kSFsolePlate",
"_kDecking",
```

```

        "_kJoist",
        "_kLedger",
        "_kRimJoist",
        "_kRimBeam",
        "_kStringer",
        "_kTread",
        "_kRiser",
        "_kSFTopHeaderSill",
        "_kSFBottomHeaderSill",
        "_kSFTopShim",
        "_kSFBottomShim",
        "_kSFLeftShim",
        "_kSFRightShim",
        "_kSFVeryTopPlate",
        "_kSFVeryTopSlopedPlate",
        "_kSFVeryBottomPlate",
        "_kSFBottomSlopedPlate",
        "_kSFVeryBottomSlopedPlate",
        "_kSFVeryTopDropInPlate",
        "_kSFSubheader",
        "_kSFSubsill",
        "_kSFWindowBlocking",
        "_kTRTop_Chord",
        "_kTRBottom_Chord",
        "_kTRWeb",
        "_kTRWedge",
        "_kTRSlider",
        "_kTRCosmetic",
        "_kTRStrongbacks",
        "_kTRRibbons",
        "_kTRTruss_Bracing",
        "_kEWBBlocking_Panel",
        "_kEWPSquash_Block",
        "_kEWPAccesBlock",
        "_kEWPCantilever_Filler",
        "_kEWPFiller_Block",
        "_kBatten",
        "_kBoardSheeting",
        "_kRoofSheeting"
    };

    for (int i=0; i<BeamTypes.length(); i++)
    {
        if (i<arPredefTypes.length() &&
        i<arPredefTypesString.length() )
            reportMessage("\n" + i + " - " + arPredefTypes[i] + " "
        " + arPredefTypesString[i] +" => "+BeamTypes[i]);
        else
            reportMessage("\n"+i+" NO PREDEF) "+BeamTypes[i]);
    }
}

```

```

    }
—end example]
```

**void setPtCtrl(**Point3d** ptCtrl, **Vector3d** vecDirection);**

The ptCtrl is the actual control point. To determine the side of the beam, the beamaxis is projected onto the vecDirection. So the control point is set in the direction of vecDirection.

[Example E-type with 2 required beams:

```

// Place the beam _Beam1 such that _Pt0 is on its axis
_Beam1.setPtCtrl(_Pt0,_X1);

// find intersection point of axis of _Beam1 with _Beam0
LineBeamIntersect LB(_Beam1.ptCen(), _X1, _Beam0);

if (LB.bHasContact()) {
    // Define cut, and cut _Beam1
    Cut ct(LB.pt1(), LB.vecNrm1());
    _Beam1.addTool(ct,1);
}
```

—end example]

[Example O-type with insert done in script:

```

// You can use this TSL in 2 ways:
// 1) add one TSL to a beam
// or
// 2) add one TSL to one side, and another TSL to the other side
// of the beam

if (_bOnInsert) {
    reportMessage("\nFirst draw a beam, then insert this TSL.");
    reportMessage("\nAfter the TSL has been inserted, select the
TSL, and move the grippoint.");
    reportMessage("\nThen insert a second TSL at the other side of
the beam.");

    _Beam.append(getBeam("Select the beam that will be
controlled"));
    _Pt0 = getPoint("Select the point through which the beam axis
will run");
    return;
}

// With one ptCtrl set, the vector to determine the control point
// side has no effect,
// because the beam will keep its direction.
// If two control points are set, the direction of each point is
important.
// There can only be one point at each direction in effect.
Vector3d vecDir = _Pt0-_Beam[0].ptCen();
vecDir.normalize();
```

```
_Beam[0].setPtCtrl(_Pt0, vecDir);
—end example]
```

```
addTool(Tool tl);
addTool(Tool tl, int nStretch);
```

There is a big (ever expanding) list of tools that can be added to a variable of type Beam, Sheet, Sip. These tools include: Drill, Cut, BeamCut,... Whenever the addTool is called on a beam, the tool is added as an operation to the list of tools that the beam is maintaining. A tool, in general, modifies the solid of the beam, and can be translated into one or more machine instructions. Some of the tools are endtools. An endtool, is an operation on a beam that does something at the end of a beam. Often it happens that adding an endtool to the beam for the first time, the other endtools in the same direction need to be removed. If this happens, the nStretch flag must be set to `_kStretchOnInsert` or 1. When the value of nStretch is set to 2 or `_kStretchOnToolChange`, the removing of the other endtools will be done, every time the end operation changes. The default value of nStretch is `_kStretchNot` or 0.

```
addToolStatic(Tool tl);
addToolStatic(Tool tl, int nStretch);
```

With the addToolStatic a tool can be added as a one time action. Typically this can be done when the `_bOnInsert` is TRUE. When `_bOnInsert` is TRUE the tsl instance is not database resident yet, and as such, cannot add dynamically maintained tools. Adding a tool as static has the drawback that the tool cannot be modified or replaced. The tool does not keep a reference to the TSL entity that has added it to the beam. So there is no link that this static tool was added by this tsl. Once added means always added. Of course if the tool is an endtool, it can be removed by adding other endtools. Also, the tool can still be removed by the console tooling tab, delete tool.

[Example any-type with insert done in script:

```
U(1, "mm");

if (_bOnInsert) {
    _Beam.append(getBeam());
    Vector3d vecX0 = _Beam[0].vecX();
    Vector3d vecY0 = _Beam[0].vecY();
    Vector3d vecZ0 = _Beam[0].vecZ();
    double dH0 = _Beam[0].dH();
    Point3d ptRef0 = _Beam[0].ptRef();

    _Pt0 = getPoint();
    _Pt0 = Line(ptRef0, vecX0).closestPointTo(_Pt0); // movepoint
    to axis of beam

    Vector3d vecN = vecX0 + vecY0 + vecZ0;
    Point3d ptC = _Pt0;
    Cut ct(ptC, vecN);
    _Beam[0].addToolStatic(ct, _kStretchOnInsert); // remove other
    end tools
```

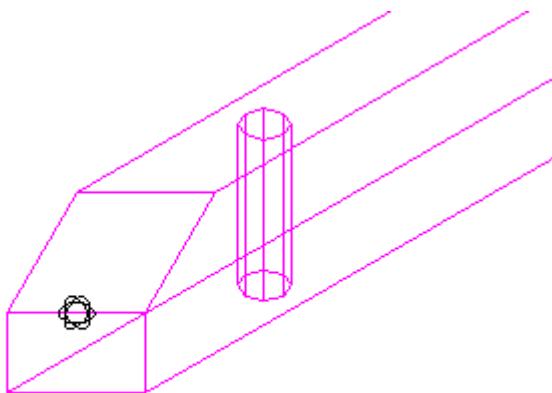
```

Vector3d vecN2 = vecX0 + vecY0;
Point3d ptC2 = Pt0;
Cut ct2(ptC2, vecN2);
Beam[0].addToolStatic(ct2, kStretchNot); // do not remove
other end tools

Point3d ptD1 = Pt0 - U(300)*vecX0 + dH0*vecZ0;
Point3d ptD2 = Pt0 - U(300)*vecX0 - dH0*vecZ0;
double dRad = U(30);
Drill drl(ptD1, ptD2, dRad);
Beam[0].addToolStatic(drl); // add the drill tool

return;
}

```



*—end example]*

**removeToolsStatic(Tool tl);**

With the **removeToolsStatic** all tools with the same type of tl will be removed from the beam.

*[Example any-type with insert done in script:*

```

if (bOnInsert) {
    Beam bm = getBeam();
    Drill drl;
    bm.removeToolsStatic(drl);

    eraseInstance();
    return;
}

```

*—end example]*

**Point3d[] dimPoints(DimLine line, int nType) const;**

Collect a list of dimension points for this entity.

Current values of nType supported:

**\_kLeft** only the point with the smallest X coordinate (along the vecX of the dimline) is appended.

**\_kRight** only the point with the biggest X coordinate (along the vecX of the dimline) is appended.

**\_kLeftAndRight** both the point with the smallest and the biggest X coordinate (along the vecX of the dimline) are appended.

**\_kCenter** the average point of \_kLeft and \_kRight is appended.

[Sample code :

```
{
    DimLine ln(pp, csEl.vecX(), csEl.vecY());

    Beam arBeams[] = elToUse.beam();
    Point3d pnts[0];
    pnts.append(csEl.ptOrg()); // append origin of element

    if (1) { // fast approach
        pnts.append(ln.collectDimPoints(arBeams, _kLeft));
    }
    else { // manual approach
        Beam arBeamsPerp[] = vecXX.filterBeamsPerpendicular(arBeams);
        for (int b=0; b<arBeamsPerp.length(); b++) {
            pnts.append(arBeamsPerp[b].dimPoints(ln, _kLeft));
        }
    }

    Dim dim(ln, pnts, "<>", "<>", _kDimCumm);
    dp.draw(dim);
}
—end sample]
```

**Body realBody() const;**

Return the solid that is in use at that moment by the Beam, Sheet,...

**Body envelopeBody() const;**

**Body envelopeBody(int bUseExtProfile, int bApplyCuts) const;**

Return an envelope solid, a solid that contains the realBody completely. This envelopBody will not have any toolings, eg. drill holes, beamcuts,.. The default value of bUseExtProfile is FALSE, and the default value of bApplyCuts is also FALSE.

[Sample code E-type:

```

    Unit(1, "mm");

    Body bd = _Beam0.realBody();
    bd.transformBy(Vector3d(220, 0, 0));

    Body bd2 = _Beam0.envelopeBody();
    bd2.transformBy(Vector3d(200, 0, 0));
```

```

// find out if bodies intersect
if (bd.hasIntersection(bd2)) { // bodies did not modify
    reportNotice("\nBodies intersect.");
    bd.intersectWith(bd2); // replace bd by intersection body
    bd.vis();
}
else {
    reportNotice("\nBodies do not intersect.");
    bd.vis();
    bd2.vis();
}
—end sample]

```

**Entity[] eToolsConnected() const;**

This routine allows to return a list of all the etools that operate on a GenBeam, Beam, Sheet, Sip. These entities can then be casted to eg ERoofPlane or TslnInst.

[Sample code:

```

...
Entity ents[] = Beam0.eToolsConnected();

// find among the etools the ERoofPlane
ERoofPlane roofplane;
for (int e=0; e<ents.length(); e++) {
    if (ents[e].bIsKindOf(ERoofPlane())) {
        roofplane = (ERoofPlane) ents[e];
        break; // out of this for loop
    }
}

// if this TSL is not attached to a Beam that belongs to a
// roofplane, do nothing
if (!roofplane.bIsValid()) {
    reportMessage("\n"+scriptName() +": the beam does not belong to
a roofplane --> erased.");
    eraseInstance();
    return;
}
...
—end sample]

```

**Body cuttingBodyOfToolEnt(Entity eToolEnt) const; // (since v21.1.33) Find the cutting Body of the tools added by the specified ToolEnt.**

This routine returns the combined cutting body of all the tools that belong to the given ToolEnt. The tools of the ToolEnt that do not implement cuttingBody, are discarded for body collection.  
[E type:

```
Entity ents[] = Beam0.eToolsConnected();
```

```

int nColor = 1;
for (int e=0; e<ents.length(); e++) {
    Body bd = _Beam0.cuttingBodyOfToolEnt(ents[e]);
    bd.vis(nColor++);
}

—end sample]

```

```

int posnum() const;
int releasePosnum(int bResetLocationToo); // (added build 16.1.18)
int assignPosnum(int nPosnumToStartSearchingFrom); // (added build 16.1.18)

```

Each GenBeam can be assigned to a group of identical items. Each of these groups gets a unique positive number called the posnum. If a GenBeam is not assigned to such a group, the posnum value of the GenBeam is -1. Although identical beams can belong to different posnums, different beams can never have the same posnum value.

Whenever a posnum is assigned one can give a nPosnumToStartSearchingFrom. Posnums with a smaller value will not be considered for comparison. If the value of nPosnumToStartSearchingFrom is still available, this value is assigned. If not the first available value that is bigger then nPosnumToStartSearchingFrom is assigned.

Both the posnum, releasePosnum and assignPosnum return the posnum of the GenBeam. To get all the entities with a certain posnum use the method of [Entity::getEntitiesWithPosnum](#).

[Example E-type:

```

PropInt nNewVal(0,10,T("|new posnum value|"));

String strAssignPosnum = T("|Assign new posnum|");
addRecalcTrigger(_kContext, strAssignPosnum );
if (_bOnRecalc && _kExecuteKey==strAssignPosnum ) {
    _Beam0.assignPosnum(nNewVal);
}

String strReleasePosnum = T("|Release posnum|");
addRecalcTrigger(_kContext, strReleasePosnum );
if (_bOnRecalc && _kExecuteKey==strReleasePosnum ) {
    int bResetLocation = FALSE;
    _Beam0.releasePosnum(bResetLocation);
}

String strReleasePosnumReset = T("|Release posnum and reset|");
addRecalcTrigger(_kContext, strReleasePosnumReset );
if (_bOnRecalc && _kExecuteKey==strReleasePosnumReset ) {
    int bResetLocation = TRUE;
    _Beam0.releasePosnum(bResetLocation);
}

String strListEntities = T("|List entity handles|");
addRecalcTrigger(_kContext, strListEntities );
if (_bOnRecalc && _kExecuteKey==strListEntities ) {
    int nPosnum = _Beam0.posnum();
    EntityCollection col = _Beam0.getEntitiesWithPosnum(nPosnum);
}

```

```

GenBeam arGb[] = col.genBeam();
reportMessage("\n"+arGb.length()+" genbeams with posnum
"+col.posnum());
for (int e=0; e<arGb.length(); e++) {
    String strHan = arGb[e].handle();
    reportMessage("\n\t"+strHan);
}
}

—end sample]

```

```

int isotropic() const; // (added 21.1.34) int value, corresponding to the predefines
    _kUndefinedIsotropic, _kIsotropic, _kNonIsotropic, _kMultiPlexIsotropic
void setIsotropic(int nIsotropic); // (added 21.1.34) sets the isotropic as int

```

[Example O-type:

```

Unit(1, "mm"); // from now on use the mm as unit in this script

if (_bOnInsert)
{
    _GenBeam.append(getGenBeam());
    _Pt0 = getPoint();

    return;
}

String strIsotropicTypes[] = { T("|Undefined|"), T("|Isotropic|"),
T("|NonIsotropic|"), T("|MultiPlexIsotropic|") };
int arIsotropicTypeInts[] = { _kUndefinedIsotropic, _kIsotropic,
_kNonIsotropic, _kMultiPlexIsotropic };
PropString pStrIsotropic(0,strIsotropicTypes, T("|Isotropic
type|"));
int nIsotropic =
arIsotropicTypeInts[strIsotropicTypes.find(pStrIsotropic,0)];

if (_GenBeam.length() < 1)
{
    eraseInstance();
    return;
}
GenBeam gb = _GenBeam[0];

String strChangeEntity = T("|Change isotropic state|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    gb.setIsotropic(nIsotropic);
}

int nIso = gb.isotropic();

```

```

String strIso =
strIsotropicTypes[arIsotropicTypeInts.find(nIso, 0)];
reportMessage("\\nIsotropic value is: " + strIso);
setDependencyOnEntity(gb);

—end example]

```

---

[Example O-type:

```

Unit(1, "mm");

if (_bOnInsert) {
    // get beam and point
    Beam bmOrig = getBeam();
    Point3d ptSelect = getPoint();

    // Beam bmNew = bmOrig.dbCopyShape();
    Beam bmNew = bmOrig.dbCopy();

    // keep parts, and point
    _Pt0 = ptSelect;
    _Beam.append(bmOrig);
    _Beam.append(bmNew);

    CoordSys csRot1; csRot1.setToRotation(30, _ZU, _Pt0);
    bmNew.transformBy(csRot1);

    return;
}

eraseInstance();

```

—end example]

---

[Example O-type illustrating copyToolsFrom:

```

if (_bOnInsert)
{
    Beam bmSource;
    while (true)
    {
        bmSource = getBeam(T("|Select beam flagged as 'Dummy with
tools'|"), true);
        if (!bmSource.bIsValid())
            continue;
        if (bmSource.bIsDummy() == 2)
            break; // is a dummy with tools
        reportMessage(T("|Not selected dummy with tools! Please
select again.|"));
    }
}

```

```

    _Entity.append(bmSource);

    Quader qdr = bmSource.quader();
    Beam gbTarget;
    gbTarget.dbCreate(qdr.ptOrg(),
        qdr.vecX(), qdr.vecY(), qdr.vecZ(),
        qdr.D(qdr.vecX()), qdr.D(qdr.vecY()), qdr.D(qdr.vecZ()));

    _GenBeam.append(gbTarget);

    return;
}

if (_GenBeam.length() == 0 || _Entity.length() == 0)
{
    eraseInstance();
    return;
}

GenBeam gbTarget = _GenBeam[0];
GenBeam gbSource = (_GenBeam)_Entity[0];
if (!gbTarget.bIsValid() || !gbSource.bIsValid())
{
    eraseInstance();
    return;
}

String strChangeEntity = T("Update beam location");
addRecalcTrigger(_kContext, strChangeEntity);
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    CoordSys csCorrect;;
    csCorrect.setToAlignCoordSys(
        gbTarget.ptRef(), gbTarget.vecX(), gbTarget.vecY(),
        gbTarget.vecZ(),
        gbSource.ptRef(), gbSource.vecX(), gbSource.vecY(),
        gbSource.vecZ());
    gbTarget.transformBy(csCorrect);
}

setDependencyOnEntity(gbSource);
Entity entRef = gbSource.blockRef();
if (entRef.bIsValid())
    setDependencyOnEntity(entRef);

_Pt0 = gbSource.ptCen();

CoordSys csTrans;
csTrans.setToAlignCoordSys(
    gbSource.ptRef(), gbSource.vecX(), gbSource.vecY(),
    gbSource.vecZ(),
    gbTarget.ptRef(), gbTarget.vecX(), gbTarget.vecY(),
    gbTarget.vecZ());

```

```
gbTarget.copyToolsFrom(gbSource, csTrans, _kStretchOnToolChange);  
—end example]
```

## 7.5 Beam

Beam is the type that represents a piece of timber. It is derived from GenBeam, so all routines from [GenBeam](#) and [Entity](#) are available for Beam.

Beams can be declared, assigned and put in an array.

```
Beam bmMale[0];
bmMale.append(_Beam0);
bmMale[0].addTool(Cut(_Pt0,_Z1),1);
```

The `bmMale` in the example above is an array of beams, initial length zero. Then the `_Beam0` is added to it, and a tool is added to the beam of the array. As an alternative for the first two lines, there could have been written:

```
Beam bmMale[1];  
bmMale[0] = _Beam0;
```

If a beam is declared inside the script, it does not reference a valid beam, until it is assigned a valid reference. Valid beams are the predefined ones, as long as the beam is not deleted in the drawing. To check the validity of a beam, one should use the `blsValid` function. A typical query would be:

```
if (_Beam[i].blsValid()) {  
    // use _Beam[i]  
}
```

A number of predeclared beams exist for each type of script:

Beam0, Beam1, Beam2, ... , Beam9

Also there is a predefined array of beams:

### Beam[]

This array is filled with the (only valid) references to the beams of the tool entity. So the following two instructions are equivalent for a T connection, right after insertion.

```
_Beam0.addTool(ct,1);  
_Beam[0].addTool(ct,1);
```

```
class Beam : GenBeam { // see GenBeam for base functions}
```

**Vector3d** vecCenterOffset() const; // (hsbcad 21.1.29) offset vector from ptRef by the extrusion profile offset

```

Vector3d vecCenterLogOffset() const; // (hsbcad 21.1.29) offset vector from ptRef by the
// extrusion profile log offset

int blsMyEnd(int bPositiveSide) const; // check if beam has an end tool on the side
// indicated
int blsCutStraight(int bPositiveSide) const; // check if beam is cut straight on the side
// indicated

// autocad database actions
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXLength, double dYWidth, double dZHeight);
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXLength, double dYWidth, double dZHeight, double dXFlag, double dYFlag,
    double dZFlag);

void dbCreate(Body bdToConvert); // convert a body into a beam
void dbCreate(Body bdToConvert, int bRound); // (added hsbCAD14.0.73)
void dbCreate(Body bdToConvert, int bRound, int bBoundingBody); // (added 21.3.107)
void dbCreate(Body bdToConvert, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
void dbCreate(Body bdToConvert, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    bRound); // (added hsbCAD14.0.73)
void dbCreate(Body bdToConvert, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int bRound,
    int bBoundingBody); // (added 21.3.107)
void dbCreate(Body bdToConvert, Point3d ptCen, Vector3d vecX, Vector3d vecY, Vector3d
    vecZ, double dYWidth, double dZHeight);

// the following dbCreate methods were added 22.1.69 and 23.0.43
void dbCreate(Entity 3dAcisSolidEntity); // convert an acis entity into a beam
void dbCreate(Entity 3dAcisSolidEntity, int bRound);
void dbCreate(Entity 3dAcisSolidEntity, int bRound, int bBoundingBody);
void dbCreate(Entity 3dAcisSolidEntity, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
void dbCreate(Entity 3dAcisSolidEntity, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    bRound);
void dbCreate(Entity 3dAcisSolidEntity, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    bRound, int bBoundingBody);
void dbCreate(Entity 3dAcisSolidEntity, Point3d ptCen, Vector3d vecX, Vector3d vecY,
    Vector3d vecZ, double dYWidth, double dZHeight);

Beam dbSplit(Point3d ptFrom, Point3d ptTo);
void dbJoin(Beam bmOther);

Entity stretchDynamicTo(GenBeam bm); // inserts a contact entity. Only use on insert or
// other events.
Entity stretchDynamicTo(GenBeam bm, double dGap); // stretchDynamicTo and setting the
// gap parameter of the contact entity. (added V22.1.6 and V21.4.81)
Entity stretchDynamicToMultiple(GenBeam bm1, GenBeam bm2); // inserts a hexcontact
// entity. Only use on insert or other events.
int stretchStaticTo(GenBeam bmFemale, int bStretch); // add static cut at the location of
// the T connection. Only use on insert or other events. (added 13.2.132)
int stretchStaticToMultiple(GenBeam bm1, GenBeam bm2, int bStretch); // insert a hex tool
// or 2 cut tools. Only use on insert or other events. (added 13.2.132)

```

```

Beam[] filterBeamsContactCut(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsContactCutHead(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsParallel(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicular(Beam[] arBeamsToCheck) const;

Plane findPlaneContactCut(Beam bmOther) const;
Plane findPlaneContactCutHead(Beam bmOther) const;

// The following 3 methods do NOT support beam connections over Xref's.
Beam[] filterBeamsTConnection(Beam[] arBeamsToCheck, double dRange, int
    bOverWriteOther) const;
Beam[] filterBeamsTConnection(Beam[] arBeamsToCheck, double dRange, int
    bOverWriteOther, double dRangeSecondBeam) const;
int hasTConnection(Beam beamToCheck, double dRange, int bOverWriteOther) const;

Beam[] filterBeamsCapsuleIntersect(Beam[] arBeamsToCheck) const;
static Beam[] filterBeamsCapsuleIntersect(Beam[] arBeamsToCheck, LineSeg seg, double
    dRadiusCapsule); // (added hsbCAD2012 build 17.0.21) static method that uses
                    the capsule defined by seg and radius

void setD(Vector3d vecDirection, double dDNew); // set the dWidth or dHeight in the
                    direction most aligned with the vecDirection argument
void setCoordSys(CoordSys csNewCoordSys); // set the reference point, X, Y and Z
                    vectors of the Beam. Make sure it is a positive coordinate system, or it will
                    be changed into one.

String extrProfile() const; // returns the entryname in the dictionary of extrusion profiles,
    see ExtrProfile
void setExtrProfile(String strVal);

String curvedStyle() const; // returns the entryname in the dictionary of curved styles, see
    CurvedStyle (added hsbCAD2009+ build 14.0.30)
void setCurvedStyle(String strVal);

String strCutN() const;
String strCutP() const;
String strCutNC() const;
String strCutPC() const;

double dCutRib1N() const;
double dCutRib2N() const;
double dCutRib3N() const;
double dCutRib4N() const;
double dCutRib1P() const;
double dCutRib2P() const;
double dCutRib3P() const;
double dCutRib4P() const;

int nCutsN() const;
int nCutsP() const;

int hasDimensions(double dLen, double dWidth, double dHeight, double dTolerance ) const;

```

```

Beam[] filterBeamsCenterDistanceYZRange(Beam[] arBeamsToCheck, double dRange,
double dTolerance) const;
Beam[] filterBeamsBoxLocation(Beam[] arBeamsToCheck, double dWidth, Vector3d[]
arBoxDims, double dTolerance) const;
CoordSys[] coordSysBeamsBoxLocation(Beam[] arBeamsToCheck, double dWidth,
Vector3d[] arBoxDims, double dTolerance) const;

static Beam[] filterBeamsHalfLineIntersectSort(Beam[] arBeamsToCheck, Point3d ptLine,
Vector3d vecLine);
static Beam[] filterBeamsHalfLineIntersectSort(Beam[] arBeamsToCheck, Point3d ptLine,
Vector3d vecLine, int bUseEnvelopeBodyWithCuts); // hsbCAD2017 build 21.3.52
and v22.0.39, Default bUseEnvelopeBodyWithCuts is FALSE, which uses
EnvelopeBody without additional cuts.

int cncSplinterFree() const;
void setCncSplinterFree(int bSet);

int hatchSection() const; // (added hsbCAD2009+ build 14.0.22)
void setHatchSection(int nSet);

int hatchHidden() const; // (added hsbCAD2009+ build 14.0.41)
void setHatchHidden(int nSet);

int isEqualComparingPosnumCriteria(Beam beamToCheck) const; // return TRUE if the
beams are considered equal regarding posnum criteria, otherwise return
FALSE.

static EntityCollection[] composeBeamPacks(Beam[] arBeamsToPack, String[]
arDifferentiators); // see EntityCollection

String texture() const; // (added hsbCAD2013 build 18.2.5)
void setTexture(String strVal);

Vector3d referenceFace() const; // (added v20.1.61 and v21.0.39)
void setReferenceFace(Vector3d vecRef); // (added v20.1.61 and v21.0.39)
};

```

---

**Vector3d** vecCenterOffset() const; // (hsbcad 21.1.29) offset vector from ptRef by the extrusion
profile offset  
**Vector3d** vecCenterLogOffset() const; // (hsbcad 21.1.29) offset vector from ptRef by the
extrusion profile log offset

If the beam does not refer to an [ExtrProfile](#) then the vecCenterOffset and vecCenterLogOffset
return a zero length vector. But if an extrusion profile is involved, they have a important meaning.
The vecCenterOffset is the offset from ptRef which is located on the bounding box center (in the
beam its YZ view). Extrusion profiles have an origin point, the origin point of the PlaneProfile
describing the extrusion profile. This origin is mapped to the line defined by ptRef, and vecX of
the beam. As a result the quader of the beam is not centered around the ptRef nor ptCen nor
ptCenSold. The quader of the beam is centered along the line through ptRef + vecCenterOffset.

The vecCenterOffset is the vector to be added to ptCen to find the center of the quader. An extrusion profile can also have a special reference point for log building. The line through this log reference point is used to locate log tools, this defines vecCenterLogOffset.

**Beam dbSplit(Point3d ptFrom, Point3d ptTo);**

To split a beam, 2 points need to be given, a ptFrom and a ptTo. Both points will be projected onto the axis line of the beam. The beam will then be split, resulting into 2 beams. The first beam runs from the beginning to ptTo. The second beam runs from ptFrom to the end. The beginning and end are seen relative to the vecX direction.

**void dbJoin(Beam bmOther);**

If bmOther is a valid beam, and the axis is parallel with this beam, then the beams will be joined, resulting in the modification of this beam, and the deletion of the bmOther. If this is the case the bmOther will not be valid after the execution.

*[Example any-type:*

```
if (_bOnInsert) {
    _Beam.append(getBeam());
    Beam bmOther = getBeam();
    _Beam[0].dbJoin(bmOther);

    reportNotice("\nbmOther valid:" +bmOther.bIsValid());
}
// will return: bmOther valid:0
—end example]
```

**void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);**  
**void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXLength,**  
**double dYWidth, double dZHeight);**  
**void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXLength,**  
**double dYWidth, double dZHeight, double dXFlag, double dYFlag, double**  
**dZFlag);**

Create a new database resident instance, insert a new Beam into the Autocad drawing. To specify the dimensions of the beam, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the beam is the same as **vecZ.length()\*dZheight**. If **vecY** is a unit-length vector, then the **dZheight** corresponds with the height. If **dZheight** equals 1, then the length of the vector expresses the height.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -**vecX**, -**vecY** and -**vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

*[Example any-type:*

```
Unit(1, "mm"); // script uses mm
if (_bOnInsert) {

    Point3d pt = getPoint(); // select point
    Vector3d vecX = _XU; // UCS X at time of insert
```

```

Vector3d vecY = _YU;
Vector3d vecZ = vecX.crossProduct(vecY);

Beam bm; bm.dbCreate(pt,vecX,vecY,vecZ,U(1000),U(100),U(200));

eraseInstance(); // this instance will not stay in the DB
return;
} // if (_bOnInsert)

—end example]

```

```

void dbCreate(Body bdToConvert); // convert a body into a beam
void dbCreate(Body bdToConvert, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
void dbCreate(Body bdToConvert, Point3d ptCen, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
    double dYWidth, double dZHeight);

```

Create a new database resident instance, insert a new Beam into the Autocad drawing. The beam is defined by the shape of a body. If only the body bdToConvert is given, then the axis system, as well as the dimensions of the beam are determined from the solid. Often, the tsl author might know the axis directions. The second dbCreate allows to specify them. In this case, the solid is not investigated to find the axis system. In the third dbCreate call, the center point as well as the width and height can be specified. In this case, only the tools are determined from the solid.

**BETTER:** This call might be calculation intensive to execute. Also the body should not be too complex.

[Example any-type:

```

Unit(1, "mm"); // script uses mm
if (_bOnInsert) {

    Beam bm = getBeam();
    Body bd = bm.realBody();
    Beam bmNew;
    bmNew.dbCreate(bd); // make a new beam from the body
description
    //bmNew.dbCreate(bd, bm.vecX(), bm.vecY(), bm.vecZ());
    //bmNew.dbCreate(bd, bd.ptCen(), bm.vecX(), bm.vecY(),
bm.vecZ(), 0.5*bd.dW(), 2*bd.dH());

    bmNew.transformBy(U(1000)*_XU);

    eraseInstance(); // this instance will not stay in the DB
    return;
} // if (_bOnInsert)

—end example]

```

```

Beam[] filterBeamsContactCut(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsContactCutHead(Beam[] arBeamsToCheck) const;

```

```
Beam[] filterBeamsParallel(Beam[] arBeamsToCheck) const;
Beam[] filterBeamsPerpendicular(Beam[] arBeamsToCheck) const;
```

These routines have as argument an array of beams, and also as return value an array of beams. The array returned is a subset of the array passed in as argument. Each function will filter out those beams that have certain properties, in relation with the properties of this beam. This beam is the beam that the function is called upon.

**filterBeamsContactCut** returns the beams that have an end tool of type Cut, that coincides with one of the side surfaces of this beam. The Cut contact touches this beam.

**filterBeamsContactCutHead** returns the beams that have an end tool of type Cut, that coincides with an end tool of type Cut of this beam. The Cut contact touches this beam.

**filterBeamsParallel** returns the beams that are parallel with this beam.

**filterBeamsPerpendicular** returns the beams that are perpendicular with this beam.

```
Beam[] filterBeamsCenterDistanceYZRange(Beam[] arBeamsToCheck, double dRange, double dTolerance) const;
```

This routine has as argument an array of beams, and also as return value an array of beams. The array returned contains only beams of which the projected center point lies inside a circle with radius dRange, centered around the center point of this beam. The projection is done in the plane through the center point of this beam, and parallel with the Y and Z axes of this beam.

```
Beam[] filterBeamsBoxLocation(Beam[] arBeamsToCheck, double dWidth, Vector3d[] arBoxDims, double dTolerance) const;
```

```
CoordSys[] coordSysBeamsBoxLocation(Beam[] arBeamsToCheck, double dWidth, Vector3d[] arBoxDims, double dTolerance) const;
```

The filterBeamsBoxLocation routine has as argument an array of beams, and also as return value an array of beams. The array returned contains 0, 1 or multiple times the amount of boxes specified. A box is specified in the array arBoxDims, by 4 vectors. So the argument arBoxDims must be an array of which the length is a multiple of 4. Each 4 vectors define a box: vecCen, vecX, vecY, vecZ, in the relative coordinate system of this beam. So vecCen is actually the offset of the center of the box. The length of the vectors vecX, vecY and vecZ determine the dimension of the box.

Since the length of the array is a multiple of 4, numM, multiple boxes can be defined.

The array that is returned is a list of beam sets. Each set of beams contains exactly numM beams. Each beam of each set has dimensions that match the dimensions of the corresponding box.

The coordinate system of this beam is defined by the dWidth value. The csY is the axis that corresponds with the dWidth dimension. If dW and dH of this beam are different, and dW matches the dWidth, 2 possible csY vectors can be taken, e.g. vecY and -vecY. If dW and dH of the beam are the same, and match the dWidth, 4 different values for the csY are possible: vecY, -vecY, vecZ and -vecZ. In the X direction both directions are considered: vecX and -vecX. This results in either 4 or 8 possible transformations.

The coordSysBeamsBoxLocation returns an array of coordinate transformations that can be used to map the returned set of beams to the real world location.

```
Plane findPlaneContactCut(Beam bmOther) const;
Plane findPlaneContactCutHead(Beam bmOther) const;
```

Find the contact plane with the other beam. If the beams do not have any contact, the world XY plane is returned. These functions are guaranteed to return a valid plane, if the beams is the result of the [filterBeamsContactCut](#) respectively [filterBeamsContactCutHead](#) function call.

```
Beam[] filterBeamsCapsuleIntersect(Beam[] arBeamsToCheck) const;
static Beam[] filterBeamsCapsuleIntersect(Beam[] arBeamsToCheck, LineSeg seg, double
dRadiusCapsule); // (added hsbCAD2012 build 17.0.21) static method that uses
the capsule defined by seg and radius
```

The argument is an array of beams, and also as return value an array of beams. The array returned is a subset of the array passed in as argument. It will contain the beams with which this beam has a "capsule" intersection. A capsule is a combination of a cylindrical shape and 2 half spheres, a rounded top and bottom. It is described by a line segment and a radius. The cylindrical part has the length of the beam. The capsule shape contains the complete beam. The intersection check is rather efficient. There is no interpretation done of the type of intersection. You should use this routine to collect all beams that can have a potential interaction with this beam, male or female. An example of the use of this routine is added below.

```
Beam[] filterBeamsTConnection(Beam[] arBeamsToCheck, double dRange, int bOverWriteOther)
const;
Beam[] filterBeamsTConnection(Beam[] arBeamsToCheck, double dRange, int bOverWriteOther,
double dRangeSecondBeam) const;
int hasTConnection(Beam beamToCheck, double dRange, int bOverWriteOther) const;
```

The first 2 routines have as argument an array of beams, and also as return value an array of beams. The array returned is a subset of the array passed in as argument. It will contain the beams with which this beam could make a T connection, this beam being the male beam. The hasTConnection routine will check this beam as male with a beamToCheck as female beam for a potential T connection. Both routines internally use the same procedures as the Intelli-Select-UI for T connections does. A range parameter is involved. This parameter, if different from zero, will express the distance between the end of this beam and the potential contact point on the surface of the female beam. Also beams can be filtered out if they have already an existing E-tool, using the bOverWriteOther boolean.

The filterBeamsTConnection can have a 4th arguments. If this 4th argument is given, the array returned will have 4 entries. Some of the entries might be null references (blsValid() will return FALSE for them). The potential 4 beams returned are 2 beams for the positive x direction of this beam, and 2 beams for the negative x direction. Indices 0 and 2 are for the positive direction, and 1 and 3 are for the negative one. These 2 pairs of beams could then be used in a stretchStaticToMultiple or stretchDynamicToMultiple.

*[Example any-type. It illustrates how the filterBeamsTConnection in conjunction with the stretchStaticToMultiple:*

```
Unit(1, "mm");

if (_bOnInsert) {

    String arYN[] = {T("|Yes|"), T("|No|")};
    PropString pStretch(0, arYN, T("|Remove other end tools|"));
    PropDouble pRangeForSecondBeam(1, U(200), T("|Range for second
beam detection|"));
}
```

```

showDialog();
int bStretch = (pStretch==arYN[0]);

Beam bm0 = getBeam(T("|Select male beam"));

Beam arBeam[0];
PrEntity ssE(T("|Select a set of beams|"), Beam());
if (ssE.go()) {
    arBeam = ssE.beamSet();
    reportMessage (T("\n|Number of beams selected:| ") +
arBeam.length());
}

// find out the matching other beams
double dRangeForDouble = pRangeForSecondBeam;
Beam arBeamT[] = bm0.filterBeamsTConnection(arBeam, U(10000),
TRUE, dRangeForDouble);
reportMessage (T("\n|Beams for T connection:| ") + arBeamT[0] +
", " + arBeamT[1]+", "
+ arBeamT[2] + ", " + arBeamT[3]);
bm0.stretchStaticToMultiple(arBeamT[0], arBeamT[2], bStretch);
bm0.stretchStaticToMultiple(arBeamT[1], arBeamT[3], bStretch);

eraseInstance();
return;
}

```

*—end example]*

```

static Beam[] filterBeamsHalfLineIntersectSort(Beam[] arBeamsToCheck, Point3d ptLine,
Vector3d vecLine);
static Beam[] filterBeamsHalfLineIntersectSort(Beam[] arBeamsToCheck, Point3d ptLine,
Vector3d vecLine, int bUseEnvelopeBodyWithCuts); // hsbCAD2017 build 21.3.52
and v22.0.39, Default bUseEnvelopeBodyWithCuts is FALSE, which uses
EnvelopeBody without additional cuts.

```

This routine has as argument an array of beams, and also as return value an array of beams. The array returned is a subset of the array passed in as argument. The function will return a list of beams of which the envelopeBody intersects the halfline from the point ptLine in the direction of the vector vecLine. The first item in the array returned is the one closest to the point.  
This routine is a static routine, which means it does not need an instance of beam to work on.

*[Example any-type. Code snippet that illustrates how the filterBeamsHalfLineIntersectSort can be used in conjunction with stretchDynamicTo:*

```

// for the vertical beams of my set, stretch them to the top and
bottom of the frame
Beam myVerticalBeams[] = ...;

if (_bOnElementConstructed || _bOnRecalc) {

```

```

// check out to where my beams need to be stretched to.
Beam arElementBeams[] = elFromBeams.beam();

// from the list of beam, remove my own beams
for (int b=0; b<myVerticalBeams.length(); b++) {
    arElementBeams=
myVerticalBeams[b].filterGenBeamsNotThis(arElementBeams);
}

// for the remaining beams, find the beam to stretch to
for (int b=0; b<myVerticalBeams.length(); b++) {
    for (int nSide=0; nSide<2; nSide++) { // loop for positive
and negative side

        Beam bm = myVerticalBeams[b];
        Point3d ptBm = bm.ptCen();
        Vector3d vecBm = bm.vecX();
        if (nSide==1) vecBm = -vecBm;

        Beam arBeamHit[] =
Beam().filterBeamsHalfLineIntersectSort(arElementBeams, ptBm
,vecBm );
        if (arBeamHit.length()>0) {
            Beam bmHit = arBeamHit[0]; // take first beam from
filtered list because it is closest.
            bm.stretchDynamicTo(bmHit);
        }

    }
}
}

—end example]

```

**String** curvedStyle() const; // (added hsbCAD2009+ build 14.0.30)  
**void** setCurvedStyle(**String** strVal);

The CurvedStyle of a Beam can be found by retrieving the entry name of the [CurvedStyle](#).

When setting the curved style, there are a few side effects to other beam properties:

- The beam width and height are adjusted to the width and height of the curved style.
- Resetting the curved style to [\\_kStraight](#) will not reset those values.
- The Iam distribution style is reset if the curved style is not set to [\\_kStraight](#).

**void** setD(**Vector3d** vecDirection, **double** dDNew);

Set the dimension of the beam in the direction of the vecDirection. The new dimension is dDNew. If the absolute value of the dotProduct of the vecY is bigger then the vecZ of the beam, then the width of the beam is changed, otherwise the height is changed. The function is typically applied only during insert, but will also work otherwise. When the dimension of the beam is changed, the

axis line will stay the same. If a surface of the beam needs to stay fixed in space, the setD needs to be combined with a transformBy call.

*[Example any-type. Changes the Y dimension (width) to 200mm of the selected beam, fixing the location of the Y surface:*

```
Unit(1,"mm"); // script uses mm
if (_bOnInsert) {

    Beam bm = getBeam(); // select beam
    Vector3d vecDir = bm.vecY(); // just take the Y axis of the beam, any other will also
do

    double dDNew = U(200); // new dimension
    double dDelta = dDNew - bm.dD(vecDir); // increase of beam dimension
    Vector3d vecD = bm.vecD(vecDir); // beam vector most aligned with vecDir
    Vector3d vecTrans = -0.5*dDelta*vecD; // move the beam in oposite direction, just
half of the inc.

    bm.setD(vecD,dDNew); // set the dimension
    bm.transformBy(vecTrans); // move the beam

    eraseInstance(); // this instance will not stay in the DB
    return;
} // if (_bOnInsert)

—end example]
```

**void setCoordSys(CoordSys csNewCoordSys);**

Set the coordinate system of the beam, without transforming any of the tools. The reference point ptRef, as well as the vecX, vecY and vecZ values will be changed by this. This action corresponds with flip X or flip Y and Z axis. Make sure the coordinate system is positive. If this is not the case, the coordinate system will be made positive anyway.

Be careful to use this in combination with the predefined variables: \_X0,\_Y0,... because the values of these predefines will not be valid anymore during that ts1 execution loop.

*[Example any-type. Changes the coordinate system of a beam:*

```
if (_bOnInsert) {

    Beam bm = getBeam(); // select beam
    Vector3d vecY = bm.vecY();
    Vector3d vecX = bm.vecX();
    Vector3d vecZ = bm.vecZ();
    Point3d ptRef = bm.ptRef();

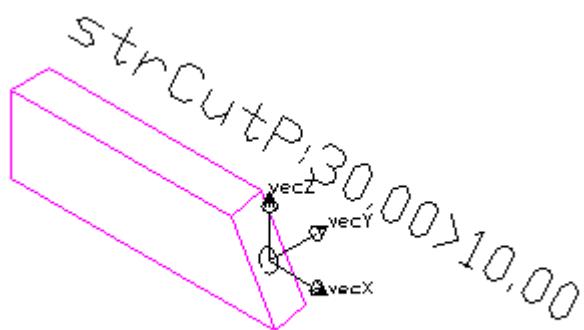
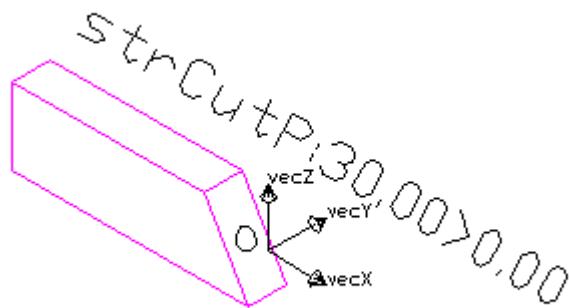
    CoordSys csNew(ptRef,-vecX,-vecY,vecZ);
    bm.setCoordSys(csNew);

    eraseInstance(); // this instance will not stay in the DB
    return;
} // if (_bOnInsert)
```

*[end example]*

```
String strCutN() const;
String strCutP() const;
String strCutNC() const;
String strCutPC() const;
```

A string representation of the angle (in degrees) of the straight cut at the end of the beam. In positive \_XF or vecX direction strCutP and strCutPC, and in negative vecX direction strCutN and strCutNC. The C and the end of the routine indicates the complementary angle. The strCutN equals a strCutP value for a coordinate system of the beam which is (-vecX,-vecY,-vecZ). The strCutPC equals a strCutP value for a coordinate system of the beam which is (vecX,vecZ,-vecY).



```
double dCutRib1N() const;
double dCutRib2N() const;
double dCutRib3N() const;
double dCutRib4N() const;
double dCutRib1P() const;
double dCutRib2P() const;
double dCutRib3P() const;
double dCutRib4P() const;
```

From the ptCenSolid of the beam, the distance along the vecX is measured of the intersection point of an edge with a relevant cut at the end of the beam. The following points are points on the cut plane:

```
Point3d pt1N = ptCenSolid() - dCutRib1N()*vecX() + dW()*vecY() + dH()*vecZ();
Point3d pt2N = ptCenSolid() - dCutRib2N()*vecX() - dW()*vecY() + dH()*vecZ();
Point3d pt3N = ptCenSolid() - dCutRib3N()*vecX() - dW()*vecY() - dH()*vecZ();
Point3d pt4N = ptCenSolid() - dCutRib4N()*vecX() + dW()*vecY() - dH()*vecZ();
Point3d pt1P = ptCenSolid() + dCutRib1P()*vecX() + dW()*vecY() + dH()*vecZ();
Point3d pt2P = ptCenSolid() + dCutRib2P()*vecX() - dW()*vecY() + dH()*vecZ();
Point3d pt3P = ptCenSolid() + dCutRib3P()*vecX() - dW()*vecY() - dH()*vecZ();
Point3d pt4P = ptCenSolid() + dCutRib4P()*vecX() + dW()*vecY() - dH()*vecZ();
```

```
int hasDimensions(double dLen, double dWidth, double dHeight, double dTollerance ) const;
```

Return TRUE or FALSE. Return TRUE if the dW() or dH() of the beam matches the dWidth and dHeight, and if the dL() matches the dLen. So either the dWidth matches the dW(), in which case the dHeight must match the dH(), or the dHeight matches the dW(), in which case the dWidth must match dH(). The length that is compared is not the solid length, but the approximate box length, which is available even if the solid representation is not up to date.

```
int cncSplinterFree() const;
void setCncSplinterFree(int bSet);
```

Return and set the cncSplinterFree parameter of beam. If bSet is set to TRUE, the cncSplinterFree is on.

**Entity** stretchDynamicToMultiple(**GenBeam** bm1, **GenBeam** bm2); // inserts a hexcontact entity.  
Only use on insert or other events.

The routine allows to insert a hexcontact eg during insert. Since the routine uses dbCreate internally to create a new ToolEnt instance, it should only be used during special events.

*[Example any-type. Illustrates the use of the stretchDynamicToMultiple.*

```
if (_bOnInsert) {

    Beam bm0 = getBeam("Select male beam");
    Beam bm1 = getBeam("Select first female beam");
    Beam bm2 = getBeam("Select second female beam");

    bm0.stretchDynamicToMultiple(bm1, bm2);

    eraseInstance();
    return;
}
```

*—end example]*

```
int stretchStaticTo(GenBeam bmFemale, int bStretch); // (added 13.2.132)
int stretchStaticToMultiple(GenBeam bm1, GenBeam bm2, int bStretch); // (added 13.2.132)
```

The stretchStaticTo routine adds a static cut located at the face of the bmFemale, at least if this beam has a T connection with the bmFemale. If bStretch is set to TRUE, all other endtools in that direction are removed. This routine should only be used on insert or other events.  
 The stretchStaticToMultiple will either add 2 cut tools or one hex tool, depending on the orientation or the beams. This routine should only be used on insert or other events.

*[Example any-type with insert done in script:*

```
if (_bOnInsert) {

    String arYN[] = {T("Yes"), T("No")};
    PropString pStretch(0, arYN, T("|Remove other end tools|"));
    showDialog();
    int bStretch = (pStretch==arYN[0]);

    Beam bm0 = getBeam(T("|Select male beam|"));
    Beam bm1 = getBeam(T("|Select female beam|"));

    bm0.stretchStaticTo(bm1, bStretch);

    eraseInstance();
    return;
}
```

*—end example]*

**int isEqualComparingPosnumCriteria(Beam beamToCheck) const**

The routine allows to verify if this beam can be considered as equal to beamToCheck using the posnum criteria. The posnum criteria can be set by the toggles in the Hsb\_settings, tab posnum. The toggle to assign each beam a unique posnum is not seen as a posnum criteria, and is not taken into account in this comparison.

**static EntityCollection[] composeBeamPacks(Beam[] arBeamsToPack, String[] arDifferentiators); // see [EntityCollection](#)**

The routine groups beams that are touching into different sets. Touching means that beams must have parallel axes to start with. Furthermore their envelope body must share a face.

---

Remember that the axis system of the beam, .vecX(), .vecY(), .vecZ(), does not react to any **OPM** properties, and is fixed to the internal axis system of the beam.

A typical loop over all the beams of a script entity, could be

```
for (int i=0; i<_Beam.length(); i++) {
    if (_Beam[i].blsValid()) {
        _Beam[i].addTool(ct,1);
    }
}
```

The **blsMyEnd** and **blsCutStraight** functions return a TRUE (1) or a FALSE (0). These functions need an argument to indicate the side of the beam that is queried. If the bPositiveSide is TRUE or not 0, the end in the positive vecX direction is considered. If the bPositiveSide is FALSE or 0, the negative vecX direction is taken. These functions must be handled with care. At time of insertion, the first execution of the script, no tools have been added to the beams yet. After the first execution, some end tools might have been added, depending on the contents of the script.

To query the value in a certain \_X0 direction one could use:

```
int bMySidePositive = (_Beam[i].vecX().dotProduct(_X0)>0);
int blsMyEnd = _Beam[i].blsMyEnd(bMySidePositive);
```

The **blsCutStraight** function will only return TRUE (or 1) when there is only one straight cut in that beam direction. If you add additional cuts, even if the cuts do not touch the beam, the function will return FALSE.

To pop up the information of a beam one could use the following example. Please notice the last line with the [eraseEntity](#) call. This line will erase the toolscript as soon as it is executed the first time, leaving no trace of the insert in the drawing.

*[Example any type:*

```
double duperm = Unit(1,"m"); // calculates how much 1 m is in drawing units
double duperm3 = duperm*duperm*duperm; // calculate the volume scale factor

Unit(1,"mm"); // from now on use the mm as unit in this script

for (int i =0; i<_Beam.length(); i++) {
    if (_Beam[i].blsValid()) {
        String strBeam = "Information beam "+i;
        strBeam += "\n volume:" + (_Beam[i].volume()/duperm3) + " m^3";
        strBeam += "\n length:" + (_Beam[i].solidLength()/duperm) + " m";
        strBeam += "\n grade:" + _Beam[i].name("grade");
        strBeam += "\n profile:" + _Beam[i].name("profile");
        strBeam += "\n material:" + _Beam[i].name("material");
        strBeam += "\n label:" + _Beam[i].name("label");
        strBeam += "\n sublabel:" + _Beam[i].name("sublabel");
        strBeam += "\n beamcode:" + _Beam[i].name("beamcode");
        strBeam += "\n type:" + _Beam[i].name("type");
        strBeam += "\n information:" + _Beam[i].name("information");
        strBeam += "\n name:" + _Beam[i].name("name");
        strBeam += "\n posnumandtext:" + _Beam[i].name("posnumandtext");
        strBeam += "\n posnum:" + _Beam[i].posnum();
        strBeam += "\n layer:" + _Beam[i].name("layer");
        strBeam += "\n blsDummy:" + _Beam[i].blsDummy();
        strBeam += "\n color:" + _Beam[i].color();
        strBeam += "\n blsCutStraight(0):" + _Beam[i].blsCutStraight(0);
        strBeam += "\n blsCutStraight(1):" + _Beam[i].blsCutStraight(1);
        strBeam += "\n blsMyEnd(0):" + _Beam[i].blsMyEnd(0);
        strBeam += "\n blsMyEnd(1):" + _Beam[i].blsMyEnd(1);
        reportWarning(strBeam);
    }
}
```

```

    }

    eraseEntity();

—end example]

```

[Example E-type with one grip point:

```

Vector3d vecDir = PtG[0]-Pt0;
double dD = Beam0.dD(vecDir); // find beam dimension in the vecDir direction
Vector3d vecD = Beam0.vecD(vecDir); // find beam vecY or vecZ in the vecDir direction

Point3d ptVis = Pt0+0.5*dD*vecD; // point on surface
vecD.vis(ptVis);

```

—end example]

[Example O-type to test filterBeamsCapsuleIntersect:

```

if (_bOnInsert) {
    Beam.append(getBeam("Select primary beam"));

    PrEntity ssE("select a set of other beams", Beam());
    if (ssE.go()) {
        Beam ssBeams[] = ssE.beamSet();
        for (int b=0; b<ssBeams.length(); b++)
            Beam.append(ssBeams[b]);
    }
    return;
}

if (_Beam.length()==0) return;

Beam bm0 = Beam[0];
Beam bmOther[0];
bmOther = Beam;
bmOther[0] = Beam(); // assing null reference to index 0
Pt0 = bm0.ptCen();

bmOther = bm0.filterBeamsCapsuleIntersect(bmOther);
reportMessage("\nNumber of capsule intersections:
"+bmOther.length());

// visualize result
Display dp(-1);

for (int b=0; b<bmOther.length(); b++) {
    Point3d pt = bmOther[b].ptCen();
    LineSeg seg(_Pt0,pt);
    dp.draw(seg);
}

```

—end example]

## 7.6 Sheet

Sheet is the type that represents a sheeting on an element, a board. It is derived from GenBeam, so all routines from [GenBeam](#) and [Entity](#) are available for Sheet.

Sheets can be declared, assigned and put in an array.

```
Sheet shList[0];
shList.append(_Sheet0);
shList[0].addTool(Cut(_Pt0,_Z1),1);
```

The shList in the example above is an array of sheets, initial length zero. Then the **\_Sheet0** is added to it, and a tool is added to the sheet of the array. As an alternative for the first two lines, there could have been written:

```
Sheet shList[1];
shList[0] = _Sheet0;
```

If a sheet is declared inside the script, it does not reference a valid drawing entity, until it is assigned a valid reference. Valid sheets are the predefined ones, as long as the sheet is not deleted in the drawing. To check the validity of a sheet, one should use the blsValid function. A typical query would be:

```
if (_Sheet0.blsValid()) {
    // use _Sheet0
}
```

A number of predeclared sheets exist for each type of script:

**\_Sheet0, \_Sheet1, \_Sheet2, ... , \_Sheet9**

Also there is a predefined array of sheets:

**\_Sheet[]**

```
class Sheet : GenBeam { // see GenBeam for base functions

    void dbCreate(PlaneProfile prof, double dThickness); // Sheet coordsys is set from prof
    coordsys
    void dbCreate(PlaneProfile prof, double dThickness, double dFlag);

    void setDH(double dDNew); // set the thickness, dDNew must be positive and not zero
    void setXAxisDirectionInXYPlane(Vector3d vecDir); // (added V22.1.137, V23.6.25, V24.0)

    void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
    void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
        dXLength, double dYWidth, double dZHeight);
```

```

void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXLength, double dYWidth, double dZHeight, double dXFlag, double dYFlag,
    double dZFlag);

// all dbCreate from Body are added v21.0.51
void dbCreate(Body bdToConvert); // convert a body into a sheet
void dbCreate(Body bdToConvert, int bRound, double dBlowUpAndShrinkProfileDistance); //
    default bRound is FALSE, and dBlowUpDist is U(100,"mm") by default
void dbCreate(Body bdToConvert, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int bRound,
    double dBlowUpAndShrinkProfileDistance);

PLine plEnvelope(Point3d ptInXY) const; // see PLine
PLine plEnvelope() const;
PLine[] plOpenings(Point3d ptInXY) const;
PLine[] plOpenings() const;

PlaneProfile profShape(Point3d ptInXY) const; // see PlaneProfile
PlaneProfile profShape() const;

void dbJoin(Sheet shOther);
Sheet[] dbSplit(Plane planeCut, double dGap);
Sheet[] dbSplit(PLine polyCut, double dGap);

Sheet[] joinRing(PLine polyRing, int bSubtract); // can return an array of sheets, if the
    polyRing is cutting the sheet in multiple pieces.

int setPIEnvelope(PLine polyRing); // basically redefines the sheet its shape by setting the
    plEnvelope and removing all plOpenings. The polyRing needs to be in XY
    plane, else it will fail. Return TRUE is operation is successful. (added
    v19.1.104 and v20.0.80)
};

```

---

**double** solidLength() const;

The length of the solid might differs from the dL() length, because the dL() is the initial length of the sheeting, and solidLength is calculated from the actual solid with all its details.

**double** solidWidth() const;

The width of the solid might differs from the dW() width, because the dW() is the initial width of the sheeting, and solidWidth is calculated from the actual solid with all its details.

```

void dbCreate(PlaneProfile prof, double dThickness);
void dbCreate(PlaneProfile prof, double dThickness, double dFlag);

```

---

Create a new database resident instance, insert a new Sheet into the Autocad drawing. The sheet is constructed by extruding a PlaneProfile, in its normal direction. If the dFlag is 0, the defining profile is in the center of the sheet. If the dFlag is 1, then the profile is at the base of the extrusion in the positive normal direction.

The length, width and height dimensions of the sheet are dependent on how the PlaneProfile is defined. The vecY of the coordSys of the PlaneProfile will become the vecX of the Sheet, and as such the axis along which the length is taken. Therefore it is essential that the CoordSys of the PlaneProfile is set correctly. The CoordSys of the PlaneProfile needs to be set in the constructor of the PlaneProfile.

```
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXLength,
              double dYWidth, double dZHeight);
void dbCreate(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXLength,
              double dYWidth, double dZHeight, double dXFlag, double dYFlag, double
              dZFlag);
```

Create a new database resident instance, insert a new Sheet into the Autocad drawing. To specify the dimensions of the sheet, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the sheet is the same as **vecZ.length()\*dZheight**. If **vecY** is a unit-length vector, then the **dZheight** corresponds with the height. If **dZheight** equals 1, then the length of the vector expresses the height.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -**vecX**, -**vecY** and -**vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

/Example any-type:

```
Unit(1, "mm");

if (_bOnInsert) {
    _Pt0 = getPoint();
    Sheet sh; sh.dbCreate(_Pt0, _XU, _YU, _ZU, U(300), U(200), U(10));
    _Sheet.append(sh);
    return;
}

if (_Sheet.length() == 0) return;
Sheet sh = _Sheet[0];

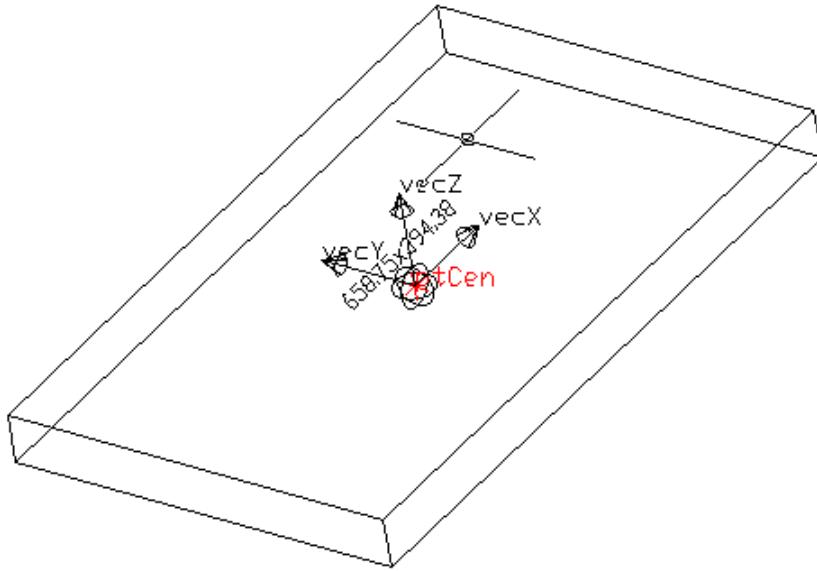
Point3d ptOrg = sh.ptCen();
Vector3d vecX = sh.vecX();
Vector3d vecY = sh.vecY();
Vector3d vecZ = sh.vecZ();

ptOrg.vis();
vecX.vis(ptOrg);
vecY.vis(ptOrg);
vecZ.vis(ptOrg);
```

```

reportNotice("\n\nsolidLength = " + sh.solidLength());
reportNotice("\nsolidWidth = " + sh.solidWidth());
reportNotice("\nLength = " + sh.dL());
reportNotice("\nheight = " + sh.dH());
reportNotice("\nWidth = " + sh.dW());

```



*—end example]*

```

PLine plEnvelope(Point3d ptInXY) const;
PLine plEnvelope() const;
PLine[] plOpenings(Point3d ptInXY) const;
PLine[] plOpenings() const;

```

These routines allow to collect the describing [PLine](#)s of the sheet profile. The sheet profile exists of an outer PLine, plEnvelope, and possibly some subtracted PLines, plOpenings. This profile is used in the construction of the body of the sheet. This profile is actually extruded. The tools that are added to the sheeting are considered as additional solid operations. These tools do not influence the profile of the sheet. The example illustrates the difference between the solid and the profile.

If the point ptInXY is given, the PLine is moved in the perpendicular (sheet.vecZ()) direction such that the PLine lies in the plane through the point ptInXY.

*[Example O-type:*

```

Unit(1,"mm");
if (_bOnInsert) {
    _Pt0 = getPoint();
    _Sheet.append(getSheet());
    return;
}

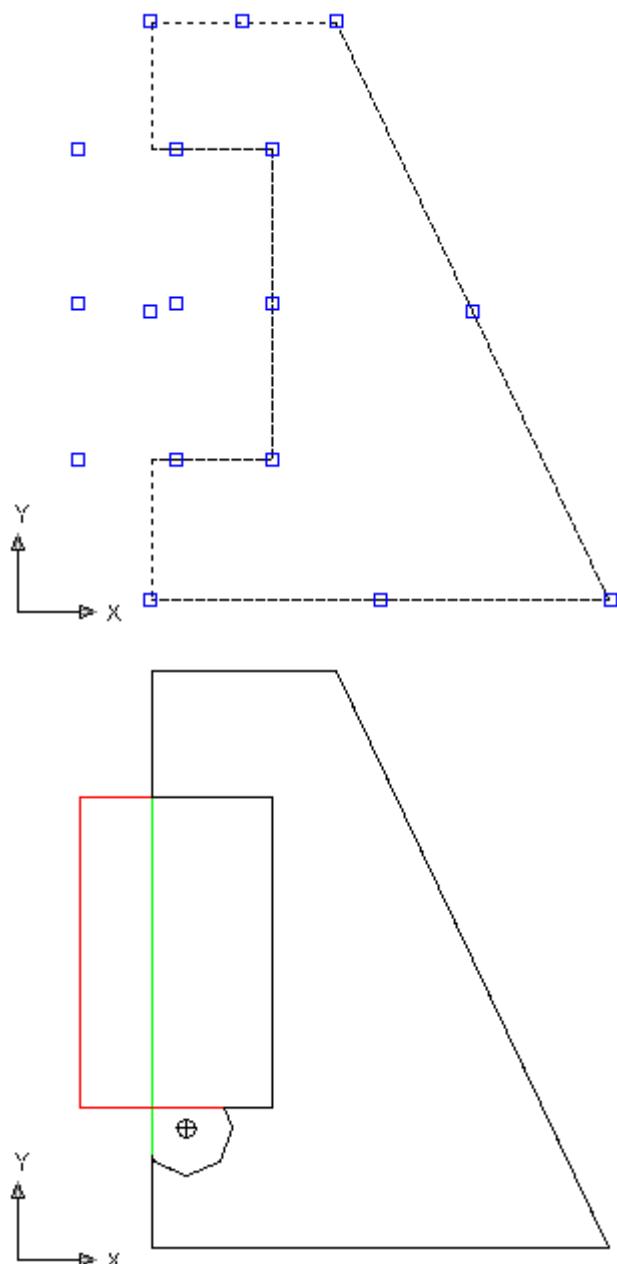
Sheet sh = _Sheet0;
CoordSys cs(_Pt0,sh.vecX(),sh.vecY(),sh.vecZ());

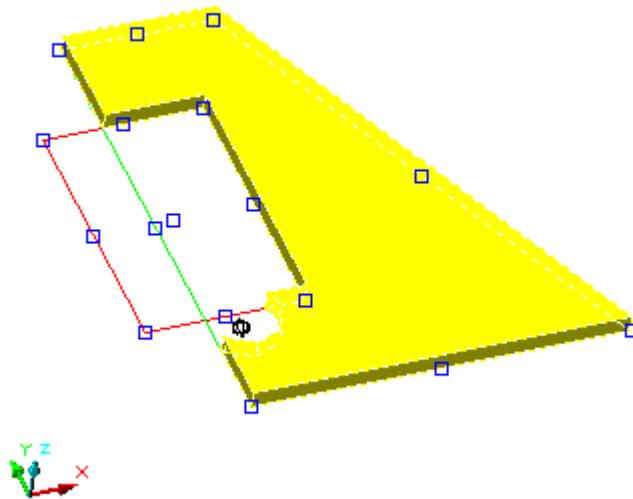
```

```
Point3d ptDrill1 = _Pt0-U(1000)*cs.vecZ();
Point3d ptDrill2 = _Pt0+U(1000)*cs.vecZ();
Drill drill(ptDrill1, ptDrill2, U(100));
sh.addTool(drill); // this tool will not influence the plEnvelope or the plOpenings

PLine plEnv = sh.plEnvelope();
plEnv.vis(3); // green

PLine plOpns[] = sh.plOpenings();
for (int o=0; o<plOpns.length(); o++) {
    plOpns[o].vis(1); // red
}
```





The first of the above images shows a sheet with a part subtracted through the `hsb` command `sheet subtract`. The next image shows the same sheet but with the above `TSL` attached. This `TSL` also adds an additional drill to the solid. Nevertheless, the `plEnvelope`, shown in green or the `plOpenings`, shown in red are not altered. The third image shows the solid.

—end example]

```
PlaneProfile profShape(Point3d ptInXY) const;
PlaneProfile profShape() const;
```

These routines allow to collect the describing `profShape` of the sheet as a [PlaneProfile](#). This profile is actually composed of the `plEnvelope` and `plOpenings`, but joined into a single `PlaneProfile`. The solid operations of the additional toolings that are applied on the sheeting will not be part of this `profShape`. If you also want these to appear, you better use the [Body::extractContactFacelnPlane](#) instead of this `profShape`.  
If the point `ptInXY` is given, the `PlaneProfile` is moved in the perpendicular (`sheet.vecZ()`) direction such that the `PlaneProfile` lies in the plane through the point `ptInXY`.

## 7.7 Sip

Sip is the type that represents a structural insulated panel. It is derived from `GenBeam`, so all routines from [GenBeam](#) and [Entity](#) are available for Sip.

Sips can be declared, assigned and put in an array.

```
Sip spList[0];
spList.append(_Sip0);
spList[0].addTool(Cut(_Pt0,_Z1),1);
```

The spList in the example above is an array of sips, initial length zero. Then the \_Sip0 is added to it, and a tool is added to the sip of the array. As an alternative for the first two lines, there could have been written:

```
Sip spList[1];
spList[0] = _Sip0;
```

If a sip is declared inside the script, it does not reference a valid drawing entity, until it is assigned a valid reference. Valid sips are the predefined ones, as long as the sip is not deleted in the drawing. To check the validity of a sip, one should use the blsValid function. A typical query would be:

```
if (_Sip0.blsValid()) {
    // use _Sip0
}
```

A number of predeclared sips exist for each type of script:

```
_Sip0, _Sip1, _Sip2, ... , _Sip9
```

Also there is a predefined array of sips:

```
_Sip[]
```

---

```
class Sip : GenBeam { // see GenBeam for base functions

    void dbJoin(Sip spOther);

    void dbCreate(PLine plShape, String strStyle);
    void dbCreate(PLine plShape, String strStyle, double dFlag);

    Sip[] dbSplit(Plane planeCut, double dGap);
    Sip[] dbSplit(PLine polyCut, double dGap);

    void setXAxisDirectionInXYPlane(Vector3d vecDir); // (added hsbCAD14.0.81)

    Plane findFoamRemovalPanelEdge(Beam bmLumber) const; // if no plane found,
        the normal of the plane has length 0

    String style() const;
    void setStyle(String strVal); // see sipStyle for more information

    SipEdge[] sipEdges() const; // return the edge information for each edge,
        see SipEdge

    int stretchEdgeTo(Point3d ptOnEdge, Plane plToStretchTo); // will stretch the edge
        of the panel identified with ptOnEdge to the specified plane
    int stretchEdgeTo(int nOpeningIndex, int nEdgelndex, Plane plToStretchTo); // (added
        v21.0.56 and v20.3.30) will stretch the edge of the panel
```

identified with nOpeningIndex and nEdgeIndex to the specified plane. For envelope use -1 as nOpeningsIndex.

```

Vector3d woodGrainDirection() const;
void setWoodGrainDirection(Vector3d vecDir);

Beam[] lumberInstallList() const; // return the list of lumber to be pre installed
void addLumberToInstallList(Beam bm); // added V26.1.5 and V25.1.106
void removeLumberFromInstallList(Beam bm); // added V26.1.5 and V25.1.106

PLine plEnvelope() const; // see example below (added hsbCAD14.0.69)
PLine[] plOpenings() const; // see example below (added hsbCAD14.0.69)

PLine plEnvelopeCnc() const; // see example below (added hsbCAD14.0.69)
PLine plShadowCnc() const; // added hsbCAD18.1.9
PLine plShadow() const; // added hsbCAD18.1.45

Sip[] addOpening(PLine polyRing, int bPlumb); // (added hsbCAD14.0.69) can return an array of sips, if the polyRing is cutting the sip in multiple pieces. If bPlumb is TRUE, the vertical (world Z) cutout is taken.
Sip[] addRing(PLine polyRing); // (added hsbCAD22.1.61) returns an array of Sips, the result of adding the ring.

LineSeg[] getWirechaseSegs() const; // return the list of wire chases (added hsbCAD15.3.14 and hsbCAD16.0.23)

PlaneProfile profCncNesting() const; // see example below (added hsbCAD17.0.48)
void setProfCncNesting(PlaneProfile profCnc); // see example below (added hsbCAD17.0.48)

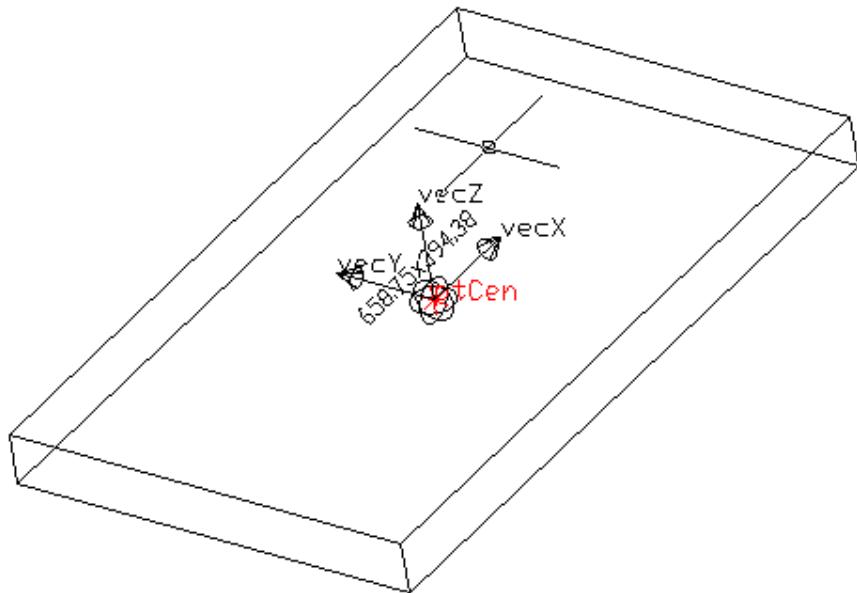
String surfaceQualityOverrideTop() const; // see example below (added hsbCAD17.0.48)
void setSurfaceQualityOverrideTop(String strVal); // see SurfaceQualityStyle for more information
String surfaceQualityOverrideBottom() const; // see example below (added hsbCAD17.0.48)
void setSurfaceQualityOverrideBottom(String strVal); // see SurfaceQualityStyle for more information

Body realBodyOfComponentAt(int nIndex); // (added hsbCAD17.2.24 and hsbCAD18.1.28)

Body basicFoamBody(); // (added hsbCAD20.0.40)
int foamComponentIndex(); // (added hsbCAD20.0.40)

int findClosestRingIndex(Point3d pt) // (added V26) returns -1 for the envelope, and >=0 for an opening.
int findClosestEdgeIndex(Point3d pt) // (added V26) returns the index into the sipEdges array.

};
```



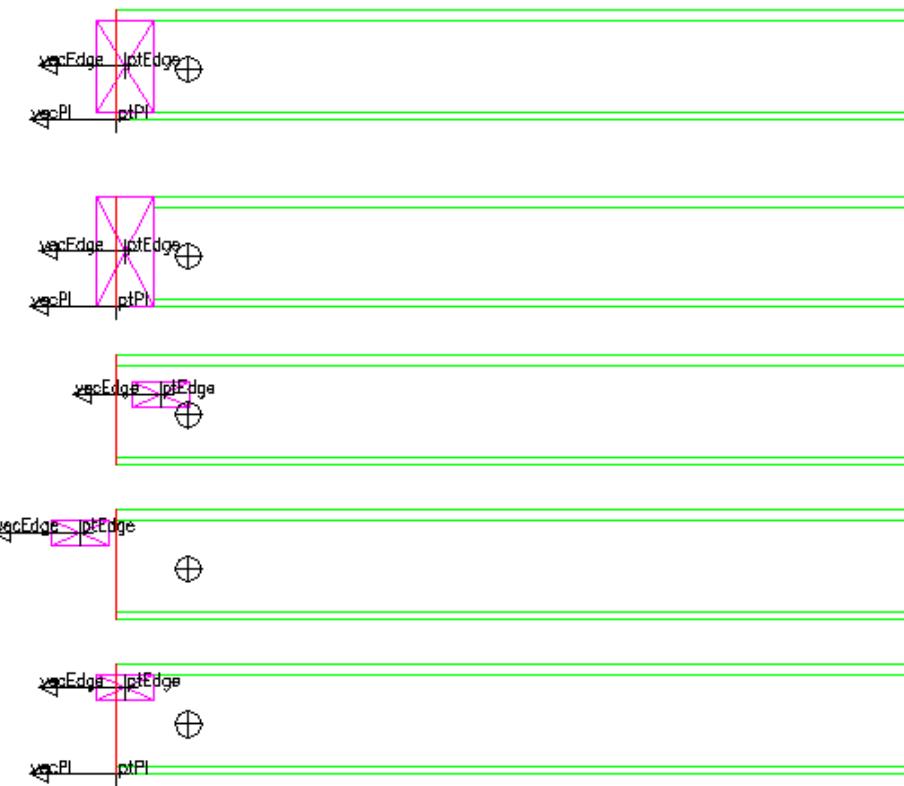

---

```
Plane findFoamRemovalPanelEdge(Beam bmLumber) const;
```

The plane that is returned contains a point on the panels basic Body ([GenBeam](#)::envelopeBody()), and a vector normal to the panel face, pointing outwards. The plane returned is a plane which is "shared" between the beam bmLumber and the panel.

*[Sample:*

```
    Plane pl = pnl.findFoamRemovalPanelEdge(bm);
    if ( !pl.normal().bIsZeroLength() ) {
        Point3d ptPl = pl.ptOrg();
        Vector3d vecPl = pl.normal();
        ptPl.vis();
        vecPl.vis(ptPl);
    }
    else {
        reportMessage ("\nNot findFoamRemovalPanelEdge!");
    }
—end example]
```



```
int stretchEdgeTo(Point3d ptOnEdge, Plane plToStretchTo);
int stretchEdgeTo(int nOpeningIndex, int nEdgeIndex, Plane plToStretchTo); (added v21.0.56 and v20.3.30)
```

This routine will stretch one edge of the panel to the specified plane. The edge of the panel is identified as the edge with the closest distance to the point `ptOnEdge`. Since the stretching is done each time the call is made, the `stretchEdgeTo` should be used with care, as it might prevent the user to interact with the panel through its normal UI.

The other method uses edge identification through `nOpeningIndex` and `nEdgeIndex`. The `nOpeningIndex` should be -1 for the envelope, or a 0 based index to identify the opening. The `nEdgeIndex` is also a zero based index to identify the edge.

*[Sample:*

```
Unit(1, "mm");

if (_bOnInsert) {

    // select the Panel and insertion point
    Sip sp = getSip();
    _sip.append(sp);
    _pt0 = getPoint(T("Select point to identify edge to stretch"));
}
```

```

        _PtG.append(getPoint(T("Select second point to identify plane
normal")));
}

return;
}

// check running conditions
if (_PtG.length()==0) return; // required one extra grip point
if (_Sip.length()==0) return; // required one Sip in the array
Sip sp = _Sip[0];

// the sip edge which is closest to the following point will be
stretched
Point3d ptToIdentifyEdge = _Pt0;

// calc plane normal of plane to stretch to
Vector3d vecNPlane = _PtG[0] - _Pt0;
vecNPlane.normalize();
Plane plToStretchTo(_Pt0, vecNPlane); // plane to stretch the sip
edge to.

sp.stretchEdgeTo(ptToIdentifyEdge, plToStretchTo);

Display dp(-1);
PLine plLine(_Pt0, _PtG[0]);
dp.draw(plLine);

—end example]

```

```

void dbCreate(PLine plShape, String strStyle);
void dbCreate(PLine plShape, String strStyle, double dFlag);

```

Create a new database resident instance, insert a new Sip into the Autocad drawing. The sip is constructed by extruding a plShape, in its normal direction. If the dFlag is 0, the defining profile is in the center of the sip. If the dFlag is 1, then the profile is at the base of the extrusion in the positive normal direction.

[Sample to illustrate the dbCreate

```

Unit(1, "mm");

if (_bOnInsert) {

    String strAllStyles[] = SipStyle().getAllEntryNames(); // list
of all available sipstyles
    PropString pStyle(0, strAllStyles, "Sip style"); // make
property
    showDialogOnce(); // allow the user to change the style

    PLine plSel = getEntPLine().getPLine();
}

```

```

Sip sp; sp.dbCreate(plSel,pStyle,1); // 1 means plSel is at the
base of the Sip, 0 means at the center

    Sip.append(sp); // keep reference in Sip array

    return;
}

if (_Sip.length()==0) return;
Sip sp = _Sip[0];

_Pt0 = sp.ptCen();
Point3d ptOrg = sp.ptCen();
Vector3d vecX = sp.vecX();
Vector3d vecY = sp.vecY();
Vector3d vecZ = sp.vecZ();

ptOrg.vis();
vecX.vis(ptOrg);
vecY.vis(ptOrg);
vecZ.vis(ptOrg);

reportNotice("\nLength = " + sp.dL());
reportNotice("\nheight = " + sp.dH());
reportNotice("\nWidth = " + sp.dW());

—end example]

```

### void setXAxisDirectionInXYPlane(**Vector3d** vecDir); // (added hsbCAD14.0.81)

The setXAxisDirectionInXYPlane allows to change the vecX of the coordSys of the Sip.

[Sample to illustrate the setXAxisDirectionInXYPlane

```

U(1,"mm");
if (_bOnInsert) {

    Sip sp = getSip();
    Sip.append(sp);

    Pt0 = getPoint(); // select point
    PtG.append(getPoint("Select point for X direction"));

    return;
}

if (_Sip.length()==0 || !_Sip[0].bIsValid() ) {
    eraseInstance(); // just erase from DB
    return;
}

Sip sp = _Sip[0];

```

```

Vector3d vecXNew = PtG[0] - Pt0;

String strChangeXDir = T("Change xdir");
addRecalcTrigger(_kContext, strChangeXDir );
if (_bOnRecalc && _kExecuteKey==strChangeXDir ) {
    sp.setXAxisDirectionInXYPlane(vecXNew);
}

—end example]

```

**PlaneProfile** profCncNesting() const; // see example below (added hsbCAD17.0.48)  
**void** setProfCncNesting(**PlaneProfile** profCnc); // see example below (added hsbCAD17.0.48)

[Sample to illustrate the profCncNesting and setProfCncNesting

```

if (_bOnInsert) {
    Pt0 = getPoint(); // select point
    Sip.append(getSip());
    return;
}

if (Sip.length()==0) {
    eraseInstance();
    return;
}

Sip sp = Sip[0];

String strSetNewProfNesting= T("|Set new profile for nesting|");
addRecalcTrigger(_kContext, strSetNewProfNesting);
if (_bOnRecalc && _kExecuteKey==strSetNewProfNesting) {
    PLine plN = getEntPLine().getPLine();
    PlaneProfile prof(plN);
    sp.setProfCncNesting(prof);
}

PlaneProfile prf = sp.profCncNesting();
prf.vis(1);

—end example]

```

**String** surfaceQualityOverrideTop() const; // see example below (added hsbCAD17.0.48)  
**void** setSurfaceQualityOverrideTop(**String** strVal); // see [SurfaceQualityStyle](#) for more information  
**String** surfaceQualityOverrideBottom() const; // see example below (added hsbCAD17.0.48)  
**void** setSurfaceQualityOverrideBottom(**String** strVal); // see [SurfaceQualityStyle](#) for more information

[Sample to illustrate the SurfaceQuality

```

U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    _Sip.append(getSip());

    return;
}

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

PropString
pTop(2,SurfaceQualityStyle() . getAllEntryNames(), "Surface Qual
Top");
PropString
pBot(3,SurfaceQualityStyle() . getAllEntryNames(), "Surface Qual
Bot");

if (_Sip.length() ==0) return;
Sip sp = _Sip[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strSetOverrides= T("Set top and bottom overrides on Sip");
addRecalcTrigger(_kContext, strSetOverrides);
if (_bOnRecalc && _kExecuteKey==strSetOverrides) {
    sp.setSurfaceQualityOverrideTop(pTop);
    sp.setSurfaceQualityOverrideBottom(pBot);
}

CoordSys cs =sp.coordSys();
cs.vis();

SipStyle spStyle(sp.style());

String strLines[0];
strLines.append("Sip");
strLines.append("style: "+sp.style());
strLines.append("surfaceQualityOverrideTop:
"+sp.surfaceQualityOverrideTop());
strLines.append("surfaceQualityOverrideBottom:
"+sp.surfaceQualityOverrideBottom());
strLines.append("style.surfaceQualityTop:
"+spStyle.surfaceQualityTop());
strLines.append("style.surfaceQualityBottom:
"+spStyle.surfaceQualityBottom());

// display the lines
Display dp(-1);

```

```

dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

```

*—end example]*

**Body** realBodyOfComponentAt(**int** nIndex); // (added hsbCAD17.2.24 and hsbCAD18.1.28)

*[Sample to illustrate the realBodyOfComponentAt*

```

(1,"mm");
if (_bOnInsert) {
    _Pt0 = getPoint(); // select point
    _Sip.append(getSip());
    return;
}

if (_Sip.length()==0) {
    eraseInstance();
    return;
}

Sip sp = _Sip[0];
int nNumComp = SipStyle(sp.style()).numSipComponents();
reportMessage("\nNumber of sip components: "+nNumComp);

Vector3d vecMove = u(1000)*_ZW;

Display dp(-1);

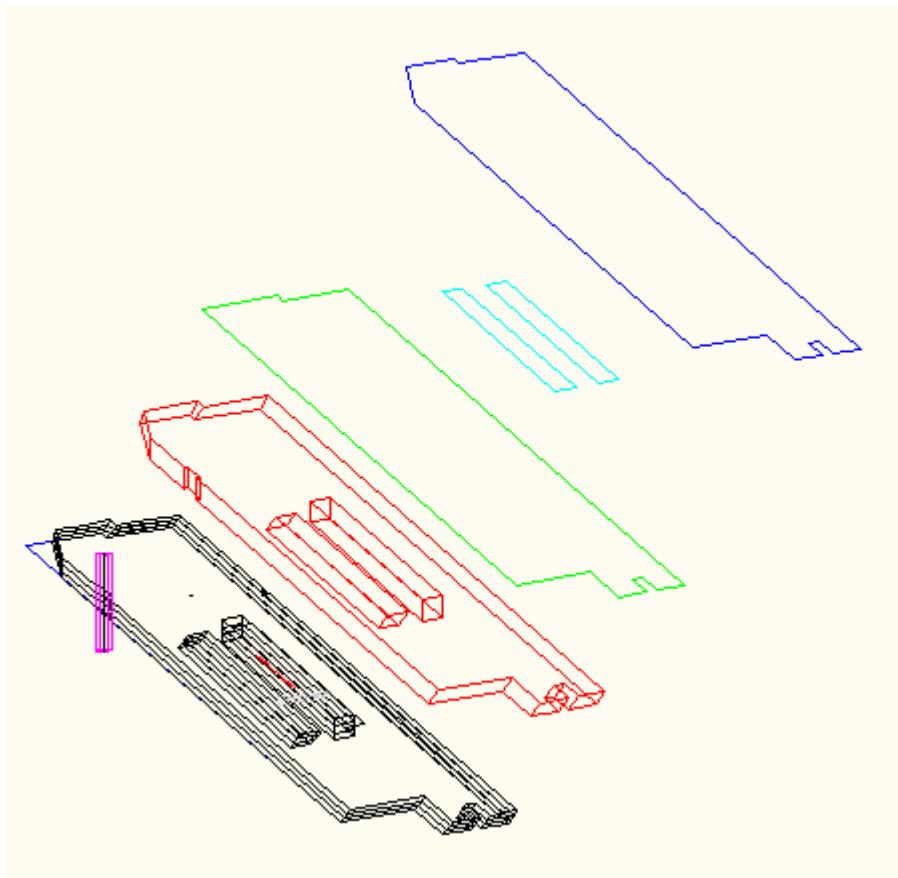
for (int s=0; s<nNumComp; s++) {
    Body bd = sp.realBodyOfComponentAt(s);
    bd.transformBy(vecMove);
    dp.color(1+s);
    dp.draw(bd);
}

```

*—end example]*

---

*[Example O-type, with insert done inside script to illustrate the plEnvelope and other PLines of the Sip*



```

U(1, "mm");

if (_bOnInsert) {
    _Pt0 = getPoint(); // select point
    _Sip.append(getSip());
    return;
}

PropDouble pMoveUp(0, U(333), "Blow up distance");

if (_Sip.length()==0) return;
Sip sp = (Sip)_Sip[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strChangeEntity = T("Add opening");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    EntPLine epl= getEntPLine();
    PLine pl = epl.getPLine();
    int bPlumb = TRUE;
    sp.addOpening(pl, bPlumb);
}

```

```

}

Vector3d vecZ = sp.vecZ();

Body bd = sp.realBody();
bd.transformBy(pMoveUp*vecZ);
bd.vis(1);

PLine plEnvelop = sp.plEnvelope();
plEnvelop.transformBy(2*pMoveUp*vecZ);
plEnvelop.vis(3);

PLine arOpenings[] = sp.plOpenings();
for (int o=0; o<arOpenings.length(); o++) {
    PLine plOpening = arOpenings[o];
    plOpening.transformBy(3*pMoveUp*vecZ);
    plOpening.vis(4);
}

PLine plEnvelopCnc = sp.plEnvelopeCnc();
plEnvelopCnc.transformBy(4*pMoveUp*vecZ);
plEnvelopCnc.vis(5);

```

*—end example*

### 7.7.1 SipEdge

The SipEdge describes one edge of the [Sip](#).

```
SipEdge edges[] = sip.sipEdges(); // query the sip edges from a sip
```

---

```

class SipEdge {

    Point3d ptStart() const;
    Point3d ptEnd() const;
    Point3d ptMid() const;

    Vector3d vecNormal() const;

    PLine plEdge() const;

    String detailCode() const;
    int recessType() const;
    double dRecessDepth() const;
}
```

---

```
};
```

Values of nRecessType are also exposed for the [PanelStop](#) tool:

```
_kNoEdgeRecess
_kSplineDouble
_kSplineJoist
_kSpline2xLumber
_kSplineSingleLumber
_kSplineSingleTopSkin
_kSplineBlock
_kSplineCustom
_kLetIn
_kHeaderRecess
```

---

[Example O-type TSL:

```
Unit(1, "mm");

if (_bOnInsert) {

    // select the Panel and insertion point
    Sip sp = getSip();
    _Sip.append(sp);
    _Pt0 = getPoint();

    return;
}

// check running conditions
if (_Sip.length()==0) return;
Sip sp = _Sip[0];
Vector3d vecNSip = sp.vecZ();

SipEdge spEdges[] = sp.sipEdges();

reportMessage("\n");
for (int i=0; i<spEdges.length(); i++) {

    SipEdge spEdge = spEdges[i];
    Point3d ptS = spEdge.ptStart(); ptS.vis(1);
    Point3d ptM = spEdge.ptMid(); ptM.vis(2);
    Point3d ptE = spEdge.ptEnd(); ptE.vis(3);

    reportMessage("\n -- Edge index: "+i);
    reportMessage("\ndetailCode: "+spEdge.detailCode());
}
```

---

```

    // calc bevel.
    Vector3d vecN = spEdge.vecNormal();
    Vector3d vecRef = vecNSip.crossProduct(vecN);
    double dBevel = vecNSip.angleTo(vecN, vecRef);
    reportMessage("\nBevel: "+dBevel);

    reportMessage("\n recessType: "+spEdge.recessType());

    PLine plE = spEdge.plEdge();
    plE.vis(1);

    reportMessage("\n dRecessDepth: "+spEdge.dRecessDepth());
}

—end example]

```

## 7.8 Element

The term element refers to a wall element, floor element or a roof element. The element is the manufacturable item, defined by a set of beams and sheetings that build up the element. An element refers to an entity in the drawing that holds all the element information. For a stickframe wall element, this entity is an AEC\_WALL. In the script, the Element type refers to the drawing entity that holds the element information.

Every toolscript instance in the drawing can hold zero, one or multiple references to elements. They are accessible through the predefined \_Element array, or the \_Element0, \_Element1,... These predefined are declared internally as:

```

Element _Element[];
Element _Element0, _Element1;

```

Because Element is derived from Entity, it inherits all the member functions of Entity as well. It can also be casted to an Entity: eg.:

```
Entity ent = _Element0;
```

An entity can be casted into an Element. However when the casting is not allowed, the resulting element will become invalid. This can be checked with the blsValid() function of Entity.

When the element is constructed, build for the first time, the condition **\_bOnElementConstructed** will be true.

Also when the element is deleted, the **\_bOnElementDeleted** will be true. When you call the Hsb Autocad command "Hsb\_recalc" (also available in the menu) of an element, the variable **\_bOnElementRecalc** will be true. If you select the TSL instance during the "Hsb\_recalc" command, the **\_bOnRecalc** will be true.

---

```

class Element : Entity { // see Entity for base functions

    int zoneIndex(int nZoneCount, Vector3d vecDirection ) const;
}

```

---

```

CoordSys coordSys(Point3d ptLocation) const;
CoordSys coordSys() const;

double dPosZOutlineFront() const;
void setDPosZOutlineFront(double dValue);
double dPosZOutlineBack() const;
void setDPosZOutlineBack(double dValue); // be careful not to set with a -dValue smaller than
                                         the dBeamWidth

double dBeamWidth() const;
void setDBeamWidth(double dValue);
double dBeamHeight() const;
void setDBeamHeight(double dValue);
String beamExtrProfile() const;
void setBeamExtrProfile(String strValue);

Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;
Point3d ptOrg() const;

PLine plOutlineWall() const; // see PLine

ElemZone zone(int nZoneIndex) const; // see ElemZone for more info
void setZone(int nZoneIndex, ElemZone zone);

GenBeam[] genBeam() const;
GenBeam[] genBeam(int nZoneIndex) const;
Beam[] beam() const;
Sheet[] sheet() const;
Sheet[] sheet(int nZoneIndex) const;
Sip[] sip() const;
Opening[] opening() const; // the openings that are part of an assembly are not included,
                           but the assemblies are.
TslInst[] tslInst() const; // return list of tsl's that belong to the elements group
NailLine[] nailLine() const; // return list of nailing lines that belong to element group (see
                           NailLine)
NailLine[] nailLine(int nZoneIndex) const; // return list of nailing lines that nail a particular
                                         zone
SawLine[] sawLine() const; // (added 23.7.12) return list of saw lines that belong to
                           element group (see SawLine)
SawLine[] sawLine(int nZoneIndex) const; // (added 23.7.12) return list of saw lines that
                                         saw a particular zone

String number() const; // can be used to initialize ElemNumber with.
void setNumber(String strNewNumber); // added in hsbCAD2012 17.0.45. For an illustration
                                         see ElemNumber which is used to compose strNewNumber.
void setFloorGroup(String strHouseName, String strFloorName); // added in V23.8.1 and
                                         24.0

String code() const;
String definition() const;

```

```

PLine plEnvelope(Point3d ptInXY) const;
PLine plEnvelope() const;

PlaneProfile profNetto(int nZoneIndex) const; // see PlaneProfile, includes only the
    openings, not the element connections
PlaneProfile profBrutto(int nZoneIndex) const; // is identical to calling profNetto with both
    bIncludeOpenings and bIncludeElementConnections set to FALSE.
void setProfNetto(int nZoneIndex, PlaneProfile prof); // will set both the contour and the
    openings, and not alter the element connections
PlaneProfile profNetto(int nZoneIndex, int bIncludeOpenings, int bIncludeElementConnections)
    const; // added in hsbCAD2014 19.0.24.
void setProfNetto(int nZoneIndex, PlaneProfile prof, int nRingTypeToReplace); // added in
    hsbCAD2014 19.0.24. The nRingTypeToReplace can be 0 for contour, 1 for
    openings and 2 for element connections. In case of contour, only the none
    voids are taken from the prof. In the other case, only the voids are taken.

LineSeg segmentMinMax() const; // return a segment that describes the diagonal of a box
    in the coordSys directions with the dimensions of the element.

String information() const; // get the value
void setInformation(String strValue); // sets the new value
String subType() const; // get the value
void setSubType(String strValue); // sets the new value

int quantity() const; // get the value of amount
void setQuantity(int nValue); // sets the new value of amount

int lock() const; // return TRUE if element is locked, else return FALSE
int setLock(int bValue); // sets the new value of lock

int numElemTexts() const; // return the number of ElemText
ElemText elemTextAt(int nInd) const; // get the ElemText at index nInd
ElemText[] elemTexts() const; // get the list of ElemText
void setElemTexts(ElemText[] elTxts); // set the list of ElemText

double dFloorGroupHeight() const;
Group elementGroup() const; // returns the group that is linked with this element

addTool(Tool tl);

// Construction directives for Stick frame
void setDistributionStudLocation(Point3d ptStudX); // place a stud at a particular location
void setRangeNoDistributionStud(Point3d ptFromX, Point3d ptToX); // define an area where
    no distribution studs are to be placed
void setAreaStud(PLine plArea); // define an area, normally inside an
    RangeNoDistributionStud, bounded by the polyline where studs need to be placed
    after all
void setAreaNoSheet(int nZoneIndex, PLine plArea); // define an area where the sheeting
    of a particular zone needs to be cut
void setSheetCutLocation(int nZoneIndex, Point3d ptLocationX); // define the location where
    the sheeting of a particular zone can be cut

```

```

void setRangeSheetGap(int nZoneIndex, Point3d ptLocation1X, Point3d ptLocation2X); // define the range where there should be no sheeting placed of a particular zone
void setOpeningOverwrite(Opening opening); // this opening is specified by tsl
void setOpeningOverwrite(Opening opening, int nZoneIndex); // this opening is specified for a particular zone by tsl
void setOpeningOverwrite(Opening opening, PLine plOpening); // use a different pline as opening for ElementWallPanel only
void setDistributionReferences(Opening opening, Point3d ptFromX, Point3d ptToX); // set reference points for the sheeting distribution, overwriting for a particular opening
void setBlockingRun(Point3d pt1, Point3d pt2, int nAmount, double dYEI, double dZEI, double dDeduct, double dTrench, double dMinLength, int bFront, int bBack, int bStaggered, int bFaceMounted, int bContinuous, Map subMap); // (added hsbCAD2013 build 18.1.8) see SetBlockingRun

// Construction directives Sip building
void setCutLocation(int nZoneIndex, Point3d ptLocationXY, Vector3d vecDirection); // define the location, in the element XY plane of the zone where the sip can be cut. The cut will run from the point in the vecDirection.
void setRoofEdgeDirective(int nZoneIndex, ElemRoofEdge edge); // define how the panel edge at that location should be generated using ElemRoofEdge.

void triggerGenerateConstruction(int bOnlyDuringDynamicFraming); // (added in build 17.0.46)
void triggerGenerateSheeting(int[] arZones); // (added in build 19.0.59)
void triggerGenerateSheeting(int[] arZones, int bGenerateNow); // (added in build 19.0.60)

PlaneProfile noNailProfile(int nZoneIndex) const; // (added in build 18.2.15, and 19.0.23) return list of no nailing areas joined into one PlaneProfile, for a particular zone

ElemNoNail[] getToolsOfTypeNoNail() const; // (added in build 21.1.1) return the list of ElemNoNail tools that belong to this Element. The list is a copy of the internal tools . Changing the tools, will not change the Element contents.
ElemSaw[] getToolsOfTypeSaw() const; // (added in build 23.5.9) return the list of ElemSaw tools that belong to this Element. The list is a copy of the internal tools . Changing the tools, will not change the Element contents.
ElemMill[] getToolsOfTypeMill() const; // (added in build 23.5.9) return the list of ElemMill tools that belong to this Element. The list is a copy of the internal tools . Changing the tools, will not change the Element contents.
ElemNail[] getToolsOfTypeNail() const; // (added in build 23.5.9) return the list of ElemNail tools that belong to this Element. The list is a copy of the internal tools . Changing the tools, will not change the Element contents.
ElemDrill[] getToolsOfTypeDrill() const; // (added in build 23.5.9) return the list of ElemDrill tools that belong to this Element. The list is a copy of the internal tools . Changing the tools, will not change the Element contents.

static String[] aca2HsbCatalogList(); // (added hsbCAD20.0.5) returns the list of aca2Hsb catalog entries
static int aca2HsbRunRuleSet(<Entity>[] arEnts, String strCatalogEntry, int bOverwriteExisting, int bUseWallGroups, int blconOnLeftSide); // (added hsbCAD20.0.5)

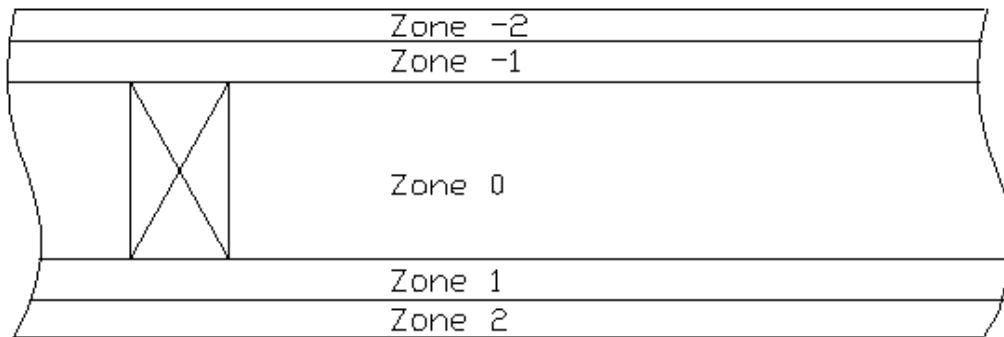
```

```
Entity[] dbCopyEntitiesFrom(Element elemFrom, <Entity>[] arEntsToConsider, <Entity>[]
arEntsToMapTo, CoordSys csTrans, int bKeepToolsFromNoneClonedToolEnts, int
bFreezeToolEnts); // (added hsbCAD21.0.67)
};
```

**int zoneIndex(int nZoneCount, Vector3d vecDirection );**

Find the zoneIndex for the n-th zone in a particular direction

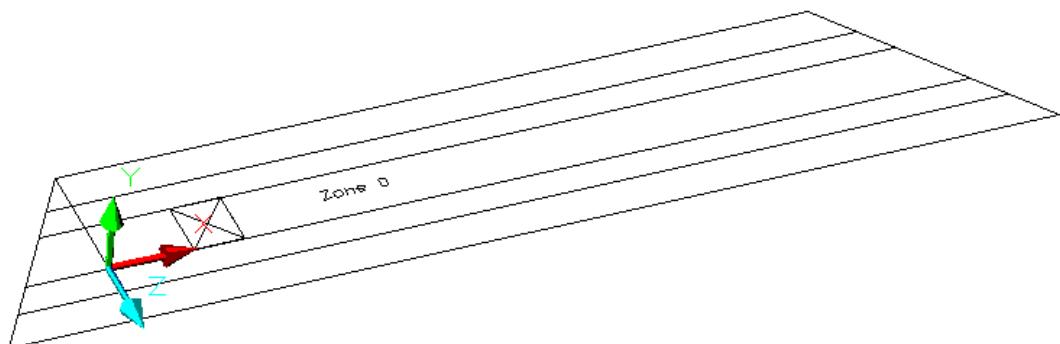
nZoneCount: zone counter -5,-4,-3,-2,-1,0,1,2,3,4,5, to be counted in a particular direction  
vecDirection: direction to count the zone in



**CoordSys** coordSys(**Point3d** ptLocation) const;  
**CoordSys** coordSys() const;

Build the coordinate system of the element, and position it in the ptLocation, projected onto the XY plane

ptLocation: the 3d point that will be projected into the XY plane of the coordSys, where the origin of the coordinate system is moved to. If the ptLocation is not given, the origin point of the element is taken.



```
double dPosZOutlineFront() const;
double dPosZOutlineBack() const;
```

Return the z-coordinate, in the element coordinate system of the lines that define the thickness of the element.

```
double dBeamWidth() const;
double dBeamHeight() const;
```

Return the dimensions of the beams stored inside the element. These dimensions are often the main stud dimensions.

```
Vector3d vecX() const;
Vector3d vecY() const;
Vector3d vecZ() const;
Point3d ptOrg() const;
```

Return the axis system and the origin point of the element.

*[Example O-type:*

```
if (_Element0.bIsValid()) {
    CoordSys cs = _Element0.coordSys();
    Point3d ptOrg = cs.ptOrg();
    Vector3d vecX = cs.vecX(); vecX.vis(ptOrg);
    Vector3d vecY = cs.vecY(); vecY.vis(ptOrg);
    Vector3d vecZ = cs.vecZ(); vecZ.vis(ptOrg);
}
```

*—end example]*

```
PLine plOutlineWall() const;
```

Retrun the polyline that defines the outline of the element in the XZ plane of the element. This function is most meaningfull for a wall element.

```
ElemZone zone(int nZoneIndex) const;
```

Construct an ElemZone instance corresponding with the zone of the element with index nZoneIndex.

nZoneIndex: The index -5,-4,-3,-2,-1,0,1,2,3,4,5 of the zone to use.

The ElemZone has of course the zone specific data: code, thickness, and the array of variables which contains also the material and the color.

The other member variables are calculated from the position of this zone inside the element.

```
void setZone(int nZoneIndex, ElemZone zone);
```

To change the zone specific data, this routine has to be used. Data that can be changed is: code, thickness, and the array of variables which contains also the material and the color.

```
GenBeam[] genBeam() const;
GenBeam[] genBeam(int nZoneIndex) const;
Beam[] beam() const;
Sheet[] sheet() const;
Sheet[] sheet(int nZoneIndex) const;
Sip[] sip() const;
NailLine[] nailLine() const;
NailLine[] nailLine(int nZoneIndex) const;
Tslnst[] tslnst() const;
```

Return a list of genbeams, beams, sheets or sips that belong to the element. These functions are expensive (take a lot of computer resources) to execute. Please use with care. One variant of the sheet(), nailLine and the genBeam exists to get a list of entities that belong to a certain zone.

The tslnst routine return the TSL's that belong to the elements group, in contrast with the Entity::tslnstAttached, which returns the TSL's that are linked with the element.

These routines used to return all the entities from a certain type that belong to the element. Now an additional filter is added that returns only those entities that belong to model space. This way, entities that belong to a block will not appear in the returned arrays.

**String number()** const;

Return the number of the element.

**String code()** const;

Return the code, also called type, element type, wall type,... of the element.

**String definition()** const;

Return the wall definition description.

```
PLine plEnvelope(Point3d ptInXY) const;
PLine plEnvelope() const;
```

The PLine resulting from cutting the element with a XY plane (in ECS of the element) through the point ptInXY. Example see below. If no point is given, the PLine is taken in the plane through the ptOrg. For roof and floor elements the PLine is the descriptive polyline, moved to the point ptInXY. So only a move is done in that case.

```
PlaneProfile profNetto(int nZoneIndex) const; // see PlaneProfile, includes only the openings,  
not the element connections
```

```

PlaneProfile profBrutto(int nZoneIndex) const; // is identical to calling profNetto with both
    bIncludeOpenings and bIncludeElementConnections set to FALSE.
void setProfNetto(int nZoneIndex, PlaneProfile prof); // will set both the contour and the
    openings, and not alter the element connections
PlaneProfile profNetto(int nZoneIndex, int bIncludeOpenings, int bIncludeElementConnections)
    const; // added in hsbCAD2014 19.0.24.
void setProfNetto(int nZoneIndex, PlaneProfile prof, int nRingTypeToReplace); // added in
    hsbCAD2014 19.0.24. The nRingTypeToReplace can be 0 for contour, 1 for
    openings and 2 for element connections. In case of contour, only the none
    voids are taken from the prof. In the other case, only the voids are taken.

```

These routines are currently only available for the stickframe wall, roof and floor elements. During the element construction, closed polylines are calculated as reference polylines for the different zones. The set of PLines that belong to one zone can be combined into a number of different PlaneProfiles: the one without the openings, profBrutto, and the one with the openings cut out and openings from element connections, profNetto.

It is also possible to set/adjust the profile.

*[Example O-type:*

```

Unit(1, "mm");

PropDouble dMoveOut(0, U(500), T("|Move out distance|"));
PropDouble dMoveDelta(1, U(30), T("|Move delta|"));

String arChoises[] = {T("|profbrutto|"), T("|default profnetto|"),
    T("|only openings|"), T("|only elem connections|"), T("|openings
    and elem conn|")};
PropString pChoice(0, arChoises, T("|What to show|"));

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint(T("\n|Select location|"));
    return;
}

// check execute conditions
if (_Element.length() == 0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

String strDefProfNet = T("|Remove with default setProfNetto|");
addRecalcTrigger(_kContext, strDefProfNet );
if (_bOnRecalc && _kExecuteKey==strDefProfNet ) {
    PlaneProfile profEmpty;
    for (int z=-5;z<6;z++) {
        el.setProfNetto(z, profEmpty);
    }
}

String str1Openings = T("|Remove zone 1 openings|");
addRecalcTrigger(_kContext, str1Openings );
if (_bOnRecalc && _kExecuteKey==str1Openings ) {

```

```

        int nZone = 1;
        // PlaneProfile prof = el.profNetto(nZone, TRUE, FALSE);
        PlaneProfile prof = el.profBrutto(nZone);
        int nRingTypeToReplace = 1; // openings
        el.setProfNetto(nZone, prof, nRingTypeToReplace);
    }

String str1WallCons = T(" | Remove zone 1 wall connection | ");
addRecalcTrigger(_kContext, str1WallCons );
if (_bOnRecalc && _kExecuteKey==str1WallCons ) {
    int nZone = 1;
    // PlaneProfile prof = el.profNetto(nZone, TRUE, FALSE);
    PlaneProfile prof = el.profBrutto(nZone);
    int nRingTypeToReplace = 2; // Wall connections
    el.setProfNetto(nZone, prof, nRingTypeToReplace);
}

// add the script entity to the element group itself
assignToElementGroup(el,TRUE,0,'E');

// get coordsys of element
CoordSys cs = el.coordSys(_Pt0);
_Pt0 = cs.ptOrg(); // set _Pt0 to XY plane
cs.vis();

Wall wall = (Wall)el;
if (wall.bIsValid()) {
    CoordSys csW = wall.coordSysHsb();
    csW.vis();
}

// show the profiles
Display dp(-1);
int bSomeShown = FALSE;

// loop over all the zones
for (int n=-5; n<6; n++)
{
    PlaneProfile prof;
    if (pChoise == arChoises[0])
        prof = el.profBrutto(n);
    else if (pChoise == arChoises[1])
        prof = el.profNetto(n);
    else if (pChoise == arChoises[2])
        prof = el.profNetto(n, TRUE, FALSE);
    else if (pChoise == arChoises[3])
        prof = el.profNetto(n, FALSE, TRUE);
    else if (pChoise == arChoises[4])
        prof = el.profNetto(n, TRUE, TRUE);

    prof.transformBy((dMoveOut+n*dMoveDelta)*cs.vecZ());
}

```

```

double dArea = prof.area();
reportMessage("\nArea for zone "+n+" is: "+dArea);

if (dArea>0.001) {
    dp.draw(prof);
    bSomeShown = TRUE;
}
}

if (!bSomeShown) {
    dp.draw(scriptName(),_Pt0,_XU,_YU,1,1);
}
—end example]

```

```

int lock() const;
int setLock(int bLock);

```

The function lock returns the locked status of the element. With the setLock function, this state can be changed.

*[Example O-type:*

```

U(1, "mm");

PropInt bLock(0,0,"Lock Element");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
Element el = _Element[0];

int bWasLocked = el.lock();
if (bWasLocked!=bLock) {
    el.setLock(bLock);
    reportMessage("\nElement lock status turned to: "+el.lock());
}
—end example]

```

```

ElemText[] elemTexts() const;
void setElemTexts(ElemText[] elTxts);

```

Get or set the list of [ElemText](#) that is stored inside the Element. This text was probably generated automatically during the generation of the element.

```

double dFloorGroupHeight() const;

```

The height of the floor to which the element group belongs is returned.

**void triggerGenerateConstruction(int bOnlyDuringDynamicFraming); // (added in build 17.0.46)**

The generate construction for this element is added to the list of elements to be generated on command ended. On the event of command ended, the set of elements is considered, and grouped into floor sets. For each floor set, the generate construction is fired with only these elements unlocked for which the generate construction was requested. The set of elements is joined with other sets of elements to be generated, coming from other requests.

If the flag bOnlyDuringDynamicFraming is TRUE, then the element generation will not happen if the console flag "Dynamic Framing" is turned off. Otherwise it happens.

*[Example O-type, and insert done in script, illustrating the generate construction]*

```
if (_bOnInsert)
{
    if (insertCycleCount ()>1) { eraseInstance (); return; }

    PrEntity ssE(T("|Select elements|"), Element ());
    Element arEl[0];
    if (ssE.go ())
        arEl = ssE.elementSet ();

    for (int e=0; e<arEl.length (); e++) {
        int bOnlyDuringDynamicFraming = TRUE;

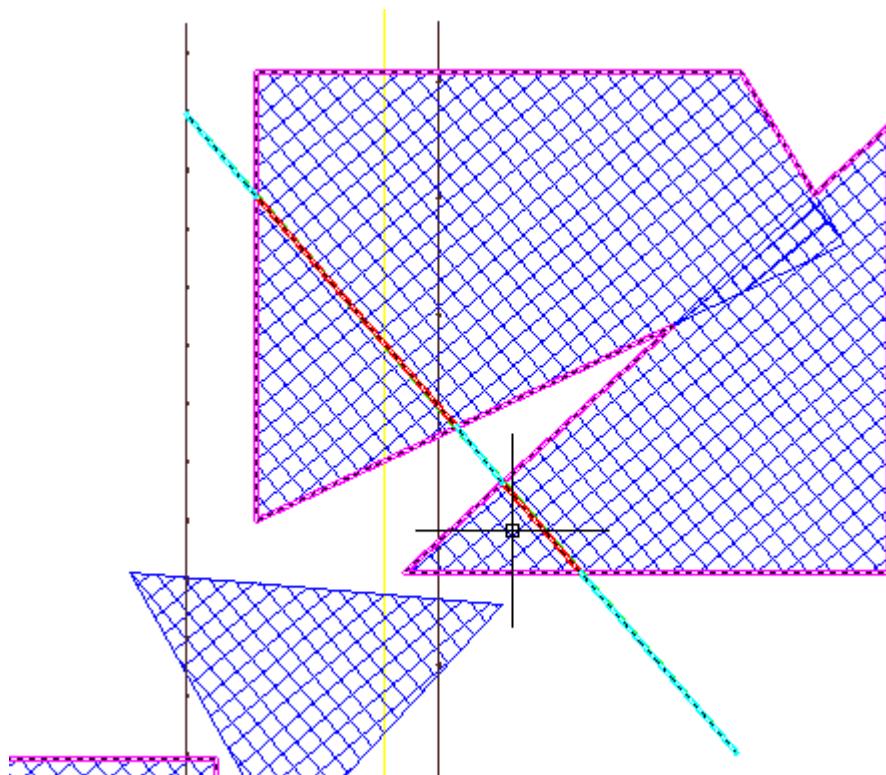
        arEl[e].triggerGenerateConstruction (bOnlyDuringDynamicFraming);
    }
}
```

*—end example]*

**PlaneProfile noNailProfile(int nZoneIndex) const; // (added in build 18.2.15, and 19.0.23) return list of no nailng areas joined into one [PlaneProfile](#), for a particular zone**

Returns the list of no nailng areas joined into one PlaneProfile. It works well together with PlaneProfile.splitSegments.

*[Example O-type, and insert done in script, illustrating the noNailProfile and splitSegments]*



```

int nArZone[]={-5,-4,-3,-2,-1,1,2,3,4,5};
PropInt nZone(0,nArZone,T("|Active zone|"),5); // default is zone
1

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint(T("|Select first point|"));
    _PtG.append(getPoint(T("|Select second point|")));
}

showDialog();
return;
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;
if (_PtG.length()==0) return;

CoordSys cs = el.coordSys();
PlaneProfile prof = el.noNailProfile(nZone);

Display dp(-1);
dp.draw(prof);

LineSeg seg(_Pt0, _PtG[0]);
dp.color(3);

```

```

dp.draw(seg);

dp.color(1);
int bKeepInternal = TRUE;
LineSeg segsIn[] = prof.splitSegments(seg, bKeepInternal);
dp.draw(segsIn);

dp.color(4);
bKeepInternal = FALSE;
LineSeg segsOut[] = prof.splitSegments(seg, bKeepInternal);
dp.draw(segsOut);

—end example]

```

**Entity[] dbCopyEntitiesFrom(Element elemFrom, Entity[] arEntsToConsider, Entity[]  
arEntsToMapTo, CoordSys csTrans, int bKeepToolsFromNoneClonedToolEnts, int  
bFreezeToolEnts); // (added hsbCAD21.0.67)**

The dbCopyEntitiesFrom allows to copy entities from one Element to another. The elemFrom is the source Element and can be part of another AcadDatabase. The Element on which dbCopyEntitiesFrom is called is the target Element.

You have to pass in the array of entities that are considered. This array contains both the entities for cloning, and for mapping to existing entities. The arEntsToMapTo contains the corresponding entities that can be used during cloning to translate existing references to. In principle the arEntsToMapTo has the same length of arEntsToConsider. But entities that are not valid in arEntsToMapTo will be cloned. Entities that are valid in arEntsToMapTo are not cloned but references to the corresponding arEntsToConsider are translated to the arEntsToMapTo during the call.

csTrans is the transformation that is applied to the cloned entities.

bKeepToolsFromNoneClonedToolEnts allows to toggle what happens with eg the Drill from an Hsb\_SurfaceDrill in case the Hsb\_SurfaceDrill is not part of the arEntsToConsider. The Drill tool can be kept, in which case it is converted into a static tool. If not kept, the Drill tool is removed from the copy.

Also, ToolEnt entities can be frozen during copy, which is practical if these ToolEnts refer to none cloned entities.

*[Example O-type, illustrating the dbCopyEntitiesFrom functionality]*

```

Unit(1, "mm");

PropInt pKeepToolsFromNoneClonedToolEnts(0, 0,
                                         "KeepToolsFromNoneClonedToolEnts");
PropInt bFreezeToolEnts(1, 0, "FreezeToolEnts");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length() == 0) return;
Element elem = (Element) _Element[0];

```

```

if (!elem.bIsValid()) return;

String strAction0 = T("|Clone|");
addRecalcTrigger(_kContext, strAction0 );
String strAction1 = T("|Clone and map beam|");
addRecalcTrigger(_kContext, strAction1 );

if (_bOnRecalc && (_kExecuteKey==strAction0 ||
_kExecuteKey==strAction1))
{
    Element elSource = getElement();

    Entity arEntsToConsider[0];
    Entity arEntsToMapTo[0];

    if (_kExecuteKey==strAction1)
    {
        Entity entSource = getEntity("Select entity to map source");
        Entity entTarget = getEntity("Select entity to map target");
        arEntsToConsider.append(entSource);
        arEntsToMapTo.append(entTarget);
    }

    PrEntity ssE(T("|Select the entities to be cloned|"), Entity());
    if (ssE.go())
        arEntsToConsider.append(ssE.set());
}

// make sure we are not copying the elSource itself
int nInd = arEntsToConsider.find(elSource);
if (nInd >=0) arEntsToConsider.removeAt(nInd);

// compose transformation from source element to target element
CoordSys csSource = elSource.coordSys();
CoordSys csTarget = elem.coordSys();
CoordSys csTrans = csSource;
csTrans.invert();
csTrans.transformBy(csTarget);

Entity entsCopied[] = elem.dbCopyEntitiesFrom(elSource,
arEntsToConsider, arEntsToMapTo, csTrans,
pKeepToolsFromNoneClonedToolEnts, bFreezeToolEnts);
reportMessage("\n" + entsCopied.length() + " entities copied.");

Vector3d vecMove(U(1000),0,0);
for(int e=0; e<entsCopied.length(); e++)
{
    entsCopied[e].transformBy(vecMove);
}
}

—end example]

```

[Example O-type, and insert done in script:

```

Unit(1, "mm"); // script uses mm

if (_bOnInsert) {
    _Pt0 = getPoint("Select a point somewhere inside the element");
    _Element.append(getElementFromEntity());

    // add the script entity to the element group itself
    assignToElementGroup(_Element[0], TRUE, 0, 'Z');

    return;
}

CoordSys cs = _Element0.coordSys(_Pt0);
_Pt0.vis();

PLine pl0 = _Element0.plEnvelope(_Pt0);
PLine pl1 = _Element0.plEnvelope(_Element0.zone(-1).ptOrg());
PLine plF = _Element0.plEnvelope(_Pt0+_Element0.dPosZOutlineFront()*cs.vecZ());
PLine plB = _Element0.plEnvelope(_Pt0+_Element0.dPosZOutlineBack()*cs.vecZ());

Display dp(-1);

// draw the different polylines
dp.draw(pl0);
dp.draw(pl1);
dp.draw(plF);
dp.draw(plB);

// define a horizontal line some space above the _Pt0, and project points onto the line.
Line ln(_Pt0+U(500)*cs.vecY(), cs.vecX()); // vecX of the element is horizontal
Point3d pts[] = pl0.vertexPoints(TRUE); // take points of pl0
pts = ln.projectPoints(pts); // project the points onto the line.
pts = ln.orderPoints(pts); // clean and order the points in the direction of the line being
                           cs.vecX
for (int i=0; i<pts.length(); i++) {
    if (i==0) { Point3d ptline0= pts[i]; ptline0.vis(1); }
    else { Point3d ptlinei= pts[i]; ptlinei.vis(1); }
}

// make from the collected points a polyline and display it
dp.color(1); // red
if (pts.length()>1) { // at least 2 points are available
    PLine plh(pts[0],pts[pts.length()-1]);
    dp.draw(plh);
}

// define a plane cutting the element in some sense, and project points onto the plane
Plane plane(_Pt0,cs.vecX()+cs.vecZ());
pts = pl0.vertexPoints(TRUE); // recollect the vertexPoints of pl0

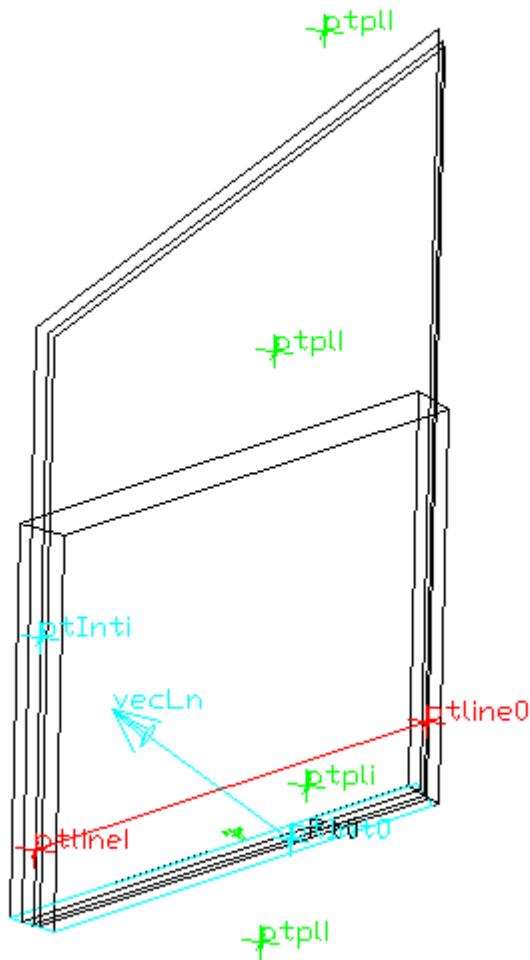
```

```

pts = plane.projectPoints(pts); // project the points onto a plane
for (int i=0; i<pts.length(); i++) {
    Point3d ptpli = pts[i]; ptpli.vis(3);
}

// find the intersection of a polyline and a plane
// The plane is defines by a point and 2 vectors inside the plane (vecLn and normal of
// polyline)
Vector3d vecLn = cs.vecX()+cs.vecY()+cs.vecZ(); vecLn.vis(_Pt0,4);
Point3d ptInt[] = pl0.intersectPoints(_Pt0,vecLn);
for (int i=0; i<ptInt.length(); i++) {
    if (i==0) { Point3d ptInt0= ptInt[i]; ptInt0.vis(4); }
    else { Point3d ptInti= ptInt[i]; ptInti.vis(4); }
}

```



*—end example]*

*[Example O-type, that should be used in the layout context to visualize xref name and count*

```
Unit(1, "mm"); // script uses mm
```

```
PropString pDimStyle(0,_DimStyles,"hsb-Bom");
pDimStyle.setDescription(T("|Select dimstyle with proper viewport
scaling and dimension format.|"));

String strWallDesc= T("|Wall Description|");
String strWallCode= T("|Wall Code|");
String strXrefName= T("|Xref Name|");
String strXrefCount= T("|Xref Count|");
String strXrefCountUnfrozen= T("|Xref Count unfrozen|");

String strTypes[] = {strWallDesc, strWallCode, strXrefName,
strXrefCount, strXrefCountUnfrozen};
PropString pType(1,strTypes,T("|Type of data|"));

if (_bOnInsert) {

    showDialog();
    _Pt0 = getPoint(); // select point
    Viewport vp = getViewport(T("|Select the viewport from which the
element is taken|")); // select viewport
    _Viewport.append(vp);

    return;
}

// set the diameter of the 3 circles, shown during dragging
setMarbleDiameter(U(4));

// do something for the last appended viewport only
if (_Viewport.length()==0) return; // _Viewport array has some
elements
Viewport vp = _Viewport[_Viewport.length()-1]; // take last element
of array
_Visual[0] = vp; // make sure the connection to the first one is
lost

// check if the viewport has hsb data
if (!vp.element().bIsValid()) return;
Element el = vp.element();

// text to be displayed
String strText = "...";

if (pType==strWallDesc) {
    strText = el.definition();
}
else if (pType==strWallCode) {
    strText=el.code();
}
else if (pType==strXrefName) {
    strText=el.xrefName();
}
```

```

else if (pType==strXrefCount) {
    Entity ents[] = el.allBlockRefPaths();
    strText=ents.length();
}
else if (pType==strXrefCountUnfrozen) {
    Entity ents[] = el.allBlockRefPaths(TRUE);
    strText=ents.length();
}

Display dp(-1); // use color of entity
dp.dimStyle(pDimStyle);

// draw the text at insert point
dp.draw(strText ,_Pt0,_XW,_YW,1,1);

—end example]

```

[Example O-type, illustrating the aca2Hsb functionality

```

Unit(1,"mm");

String arEntries[] = Element().aca2HsbCatalogList();

PropString pEntry(0, arEntries, T("|Aca2Hsb calatog to use|"));
PropInt pOverwriteExisting(0, 0, T("|Overwrite existing|"));
PropInt pUseWallGroups(1, 1, T("|Use wall groups|"));
PropInt pOnLeftSide(2, 0, T("|Place icon on left side|"));

if (_bOnInsert) {
    showDialog();
    _Pt0 = getPoint();
}

String strCreateModel = T("|Trigger run aca2hsb|");
addRecalcTrigger(_kContext, strCreateModel );
if (_bOnInsert || (_bOnRecalc && _kExecuteKey==strCreateModel) )
{
    Entity ents[0];

    PrEntity ssE(T("\n|Select a set of wall elements|"),wall());
    if (ssE.go()) {
        ents = ssE.set();
    }

    Element().aca2HsbRunRuleSet(ents, pEntry, pOverwriteExisting,
    pUseWallGroups, pOnLeftSide);
}

```

—end example]

### 7.8.1 ElemZone

Every element has a number of zones. One zone for each sheeting material. At this moment there is a maximum of 5 zones at the front of the element (for walls the arrow side), and 5 zones at the back of the element (for wall element the none-arrow side). A zone has a zoneindex into the element. A zone has a thickness, a position in the element, and a coordinate system. A zone can be constructed through querying the element.

```
ElemZone elzone = _Element0.zone(int nZoneIndex);
```

When parameters of the zone are changed by eg setDH, setColor, ... the values are only changed in the ElemZone variable itself. If the values also need to change in the original element, one has to use the setZone routine of the element.

Beware: there is no ElemZone for zone 0, which is the area where the beams go.

---

```
class ElemZone {

    int zoneIndex() const;

    CoordSys coordSys(Point3d ptLocation) const;
    CoordSys coordSys() const;

    double dH() const;
    void setDH(double dH);

    double dPosZ() const;
    Vector3d vecZ() const;
    Point3d ptOrg() const;

    String material() const;
    void setMaterial(String str);
    int color() const;
    void setColor(int colorIndex);
    String code() const;
    void setCode(String str);

    String distribution() const;

    double dVar(String strName) const;
    String strVar(String strName) const;
    int hasVar(String strName) const;

    double dVar(int nIndex) const;
    String strVar(int nIndex) const;
    int hasVar(int nIndex) const;

    void setDVar(String strName, double dVal);
```

```

void setDVar(int nIndex, double dVal);
void setStrVar(String strName, String strVal);
void setStrVar(int nIndex, String strVal);

};

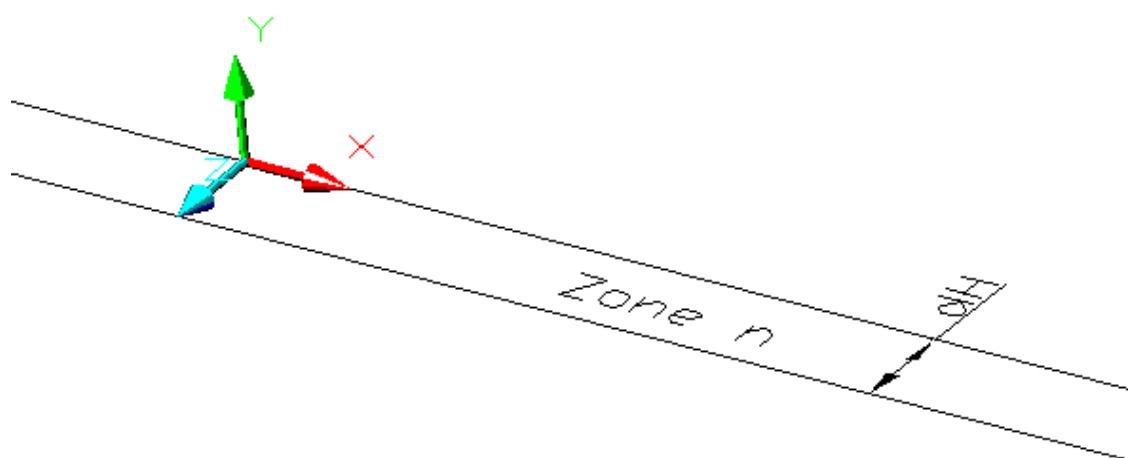

```

---

**CoordSys** coordSys(**Point3d** ptLocation) const;  
**CoordSys** coordSys() const;

Build the coordinate system of the zone, and position it in the ptLocation, projected onto the XY plane. The coordinate system is located with the thickness of the zone from 0 to dH, in the positive vecZ direction.

**ptLocation:** the 3d point that will be projected into the XY plane of the coordSys, where the origin of the coordinate system is moved to. If the ptLocation is not given, the origin point of the element is taken.



**double dH() const;**

Return the thickness in z direction of the zone.

**double dPosZ() const;**

Return the z-coordinate, in the element coordinate system of the XY plane of the zone.

**Vector3d vecZ() const;**  
**Point3d ptOrg() const;**

Return the origin and the vecZ of the coordinate system of the zone.

**String material()** const;

Return value of the material variable (also the variable with index 0).

**int color()** const;

Return value of the color variable (also the variable with index 1).

**String code()** const;

Return the distribution code. Possible values are HSB-PL01, HSB-PL02, ... It is actually this code that needs to be used to react on a certain distribution, rather than using the translated distribution name.

**String distribution()** const;

Return the translated name of the distribution.

**int zoneIndex()** const;

Recall the zone index of the zone in the element.

```
double dVar(String strName) const;
String strVar(String strName) const;
int hasVar(String strName) const;
double dVar(int nIndex) const;
String strVar(int nIndex) const;
int hasVar(int nIndex) const;
```

The routines dVar, strVar return the value of a variable. The variable can be identified with a name or an index. In case it is a name, the name is not case sensitive, and must not be translated.

For valid names look to the example below. Valid names are depending on the distribution, and could be: "material", "color", "width", "gap", "first sheet", "height sheet", "height", "module", "max. length", "height plate", "width opening"...

The variable can be returned as a string or as a double value. It is also possible to access the variable from its index. The index is zero based. The index 0 will return the material, and the index 1 will return the color. Other indexes that could return a useful value are 2,3,4,5,6,7. To find out if the distribution at hand has a certain variable, by name or by index, the hasVar routine can be used.

---

[Example O-type:

```
Unit(1, "mm");
int arZones[] = {1, 2, 3, 4, 5};
```

---

```

PropInt nZoneCount(0,arZones,T("|Zone|"),0); // zone 1 is
default (index 0)
PropString pDimStyle(0,_DimStyles,T("|Dim style|"));
PropDouble pTextHeight(0,U(20),T("|Text height|"));

String arCodes[0];
String arDistributions[0];
arCodes.append("HSB-PL01"); arDistributions.append(T("|
Full|"));
arCodes.append("HSB-PL02"); arDistributions.append(T("|
Vertical Sheets|"));
arCodes.append("HSB-PL03"); arDistributions.append(T("|
Continous Sheeting|"));
arCodes.append("HSB-PL04"); arDistributions.append(T("|
Horizontal Lap Siding|"));
arCodes.append("HSB-PL05"); arDistributions.append(T("|
Vertical Siding|"));
arCodes.append("HSB-PL06"); arDistributions.append(T("|
Vertical Siding 2|"));
arCodes.append("HSB-PL08"); arDistributions.append(T("|
Left - Right|"));
arCodes.append("HSB-PL09"); arDistributions.append(T("|
Vert. latwerk op stijlen|"));
arCodes.append("HSB-PL10"); arDistributions.append(T("|
Type 10|"));
arCodes.append("HSB-PL11"); arDistributions.append(T("|
Party Wall|"));
arCodes.append("HSB-PL12"); arDistributions.append(T("|
Overlap at window|"));
arCodes.append("HSB-PL13"); arDistributions.append(T("|
Horizontal Lath|"));
arCodes.append("HSB-PL14"); arDistributions.append(T("|
Log Siding|"));

PropString pZoneCode(1,arCodes,T("|Zone code|"),0);
pZoneCode.setReadOnly(TRUE);
PropString pDistribution(2,arDistributions,T("|Zone
distribution|"),0);
pDistribution.setReadOnly(TRUE);

if (_bOnInsert) {
    _Pt0 = getPoint();
    _Element.append(getElement());
    return;
}

// check running conditions
if (_Element.length()==0) return;

```

```
Element el = _Element[0];
if (!el.bIsValid())
    return;
CoordSys cs = el.coordSys();
int nZone = el.zoneIndex(nZoneCount, _Pt0-cs.ptOrg()); // find
zoneIndex of nZoneCount on side of _Pt0

String strChangeZoneCode = T("|Set distribution by zone code
property|");
addRecalcTrigger(_kContext, strChangeZoneCode );
if (_bOnRecalc && _kExecuteKey==strChangeZoneCode)
{
    pZoneCode.setReadOnly(FALSE);
    showDialog();
    ElemZone ezToSet = el.zone(nZone);
    ezToSet.setCode(pZoneCode);
    el.setZone(nZone, ezToSet);
}

String strChangeZoneDistri = T("|Set distribution by
property|");
addRecalcTrigger(_kContext, strChangeZoneDistri );
if (_bOnRecalc && _kExecuteKey==strChangeZoneDistri)
{
    pDistribution.setReadOnly(FALSE);
    showDialog();
    int ind = arDistributions.find(pDistribution, 0);
    String zoneCodeNew = arCodes[ind];

    ElemZone ezToSet = el.zone(nZone);
    ezToSet.setCode(zoneCodeNew);
    el.setZone(nZone, ezToSet);
}

String strSetSome = T("|Set some values|");
addRecalcTrigger(_kContext, strSetSome );
if (_bOnRecalc && _kExecuteKey==strSetSome)
{
    ElemZone ezToSet = el.zone(nZone);
    ezToSet.setStrVar("blocking", 1);
    ezToSet.setDVar("vertical gap", 11);
    el.setZone(nZone, ezToSet);
}

ElemZone ez = el.zone(nZone);
pZoneCode.set(ez.code());
pDistribution.set(ez.distribution());
```

```

Vector3d vecZ = ez.vecZ();
Point3d ptOrg = ez.ptOrg();
ptOrg.vis();
vecZ.vis(ptOrg);

String strLines[0];

strLines.append("zone index: " + ez.zoneIndex());
strLines.append("thickness: " + ez.dH());
strLines.append("posZ: " + ez.dPosZ());
strLines.append("material: " + ez.material());
strLines.append("color: " + ez.color());
strLines.append("distribution: " + ez.distribution());
strLines.append("code: " + ez.code());

String keys[] = { "material", "color", "width", "gap", "first
sheet", "height sheet", "blocking",
"vertical gap", "height", "module", "max. length", "height
plate", "control point name",
"control point type", "width first plate", "gap back",
"gap front", "width opening",
"horizontal gap", "max length", "distribution", "type",
"tongue height", "first log height"};
```

strLines.append("");  
**for**(**int** k=0; k<keys.length(); k++)  
{  
**String** key = keys[k];  
**int** bHasKey = ez.hasVar(key);  
**if** (bHasKey)  
 strLines.append("Key "+key+": " + ez.strVar(key));  
}  
  
strLines.append("");  
**for**(**int** k=0; k<keys.length(); k++)  
{  
**String** key = keys[k];  
**int** bHasKey = ez.hasVar(key);  
**if** (!bHasKey)  
 strLines.append("no key "+key+" found");  
}  
  
// display the lines  
**Display** dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
**for** (**int** l=0; l<strLines.length(); l++) {
 **Vector3d** vecO = -l\*1.2\*pTextHeight\*\_YU;
}

```

dp.draw(strLines[1], _Pt0+vecO , _XU, _YU, 1,1);
}

—end example]

```

## 7.8.2 ElemText

Each [element](#) (can) contain a list of text. Actually a list of ElemText. An ElemText is a text located somewhere around the element, to be displayed in the wall elevation layout. It is normally generated automatically during the element generation process.

This is not a tool. Every element has a number of ElemTexts as member variables. A list of ElemText can be constructed through querying the element. The complete list of ElemText can be overwritten by calling the setElemTexts.

```

ELEMTEXT eltext[] = _ELEMENT0.elemTexts();
_ELEMENT0.setElemTexts(ELEMTEXT eltext[]);

```

The following codes and subcodes are used by the generation process:

code	subcode	meaning
DETAIL	1	bottom
	2	left
	3	left-angled
	4	top
	5	right-angled
	6	right
	7	bottom-angled
WINDOW	WINDOWTOP	
	WINDOWBOTTOM	
	WINDOWLEFT	
	WINDOWRIGHT	
	HEADER	
	TOP (height top)	
DIM	DESCRIPTION	
	BOTTOM	

---

```

class ELEMTEXT {

    ELEMTEXT(Point3d ptOrg, Vector3d vecDir, String strTxt);

    Point3d ptOrg() const;
    void setPtOrg(Point3d pt);
    Point3d pt1() const;
    void setPt1(Point3d pt);
    Point3d pt2() const;
}

```

---

```

void setPt2(Point3d pt);

Vector3d vecDir() const;
void setVecDir(Vector3d vecDir);

String text() const;
String code() const;
String subCode() const;
void setText(String str);
void setCode(String str);
void setSubCode(String str);

double dH() const;
int zoneIndex() const;
void setDH(double dH);
void setZoneIndex(int nZoneIndex);

int autoErase() const;
int indexRef() const;
Entity idRef() const;
void setAutoErase(int nVal);
void setIndexRef(int nVal);
void setIdRef(Entity ent);

};

```

```

double dH() const;
void setDH(double dH);

```

Get and set the proposed height of the text.

```

Point3d ptOrg() const;
void setPtOrg(Point3d pt);
Vector3d vecDir() const;
void setVecDir(Vector3d vecDir);

```

Get and set the origin point and the offset direction of the text.

```

Point3d pt1() const;
void setPt1(Point3d pt);
Point3d pt2() const;
void setPt2(Point3d pt);

```

The points pt1 and pt2 define a line segment that is relevant for this elemtext, at least if it is relevant.

```
String text() const;
String code() const;
String subCode() const;
void setText(String str);
void setCode(String str);
void setSubCode(String str);
```

Get and set the actual text, a code and a subcode.

```
int zoneIndex() const;
void setZoneIndex(int nZoneIndex);
```

Get and set the zone index of the zone in the element.

```
int autoErase() const;
void setAutoErase(int nVal);
```

If true if the text will be erased when the element is regenerated.

```
int indexRef() const;
Entity idRef() const;
void setIndexRef(int nVal);
void setIdRef(Entity ent);
```

These two routines identify where this information is coming from. If this text needs to be maintained by another entity, the idRef can be set to reference that entity. The indexRef is an additional value of which the meaning is completely determined by the entity that is maintaining this text. If a particular TSL would set more than one text of an element, that TSL needs to be able to distinguish the different texts. The indexRef can be used for that.

---

*[Example O-type with insert implemented:*

```
U(1,"mm");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
Element el = _Element[0];
CoordSys csEl = el.coordSys();

ElemText elTexts[] = el.elemTexts();
```

---

```

for (int t=0; t<elTexts.length(); t++) {

    ElemText txt = elTexts[t];
    reportNotice("\n\n zone index: " + txt.zoneIndex());
    reportNotice("\n text height: " + txt.dH());
    reportNotice("\n ptOrg: " + txt.ptOrg());
    reportNotice("\n vecDir: " + txt.vecDir());
    reportNotice("\n text: " + txt.text());
    reportNotice("\n code: " + txt.code());
    reportNotice("\n subCode: " + txt.subCode());
    reportNotice("\n autoErase: " + txt.autoErase());
    reportNotice("\n indexRef: " + txt.indexRef());
    reportNotice("\n idRef: " + txt.idRef());

}

ElemText txt;
//ElemText txt2(csEl.ptOrg(),csEl.vecZ(),"This is me");
//txt = txt2;
txt.setDH(U(100));
txt.setPtOrg(csEl.ptOrg());
txt.setVecDir(csEl.vecY());
txt.setZoneIndex(3);
txt.setText("Who is who");
txt.setCode("what is my code");
txt.setSubCode("subsubsub");
txt.setAutoErase(TRUE);
txt.setIndexRef(1);
txt.setIdRef(_ThisInst);

elTexts.append(txt);
el.setElemTexts(elTexts);

```

*—end example]*

### 7.8.3 Construction directives

As declared in the [Element](#) class, there are a number of functions implemented to specify what can be called as construction directives. A construction directive is a directive on how to generate the construction, e.g. where to place a stud, where to cut sheeting.

The following functions are implemented for the stick framing:

```

void setDistributionStudLocation(Point3d ptStudX); // place a stud at a particular location
void setRangeNoDistributionStud(Point3d ptFromX, Point3d ptToX); // define an area where
    no distribution studs are to be placed
void setAreaStud(PLine plArea); // define an area, normally inside an
    RangeNoDistributionStud, bounded by the polyline where studs need to be placed
    after all
void setAreaNoSheet(int nZoneIndex, PLine plArea); // define an area where the sheeting
    of a particular zone needs to be cut

```

```

void setSheetCutLocation(int nZoneIndex, Point3d ptLocationX); // define the location where
    the sheeting of a particular zone can be cut
void setRangeSheetGap(int nZoneIndex, Point3d ptLocation1X, Point3d ptLocation2X); //
    define the range where there should be no sheeting placed of a particular zone
// void setSpecial(String strName, Point3d ptLocationX [, String strVal0, String
    strVal1, ...]); // define a special at a certain point. The number of arguments must be
    equal or bigger then 2. The arguments after the first 2 are interpreted as strings.
    (OBSOLETE since hsbCAD14.0.70)

void setOpeningOverwrite(Opening opening); // this opening is specified by tsl
void setOpeningOverwrite(Opening opening, PLine plOpening); // use a different pline as
    opening for ElementWallPanel only
void setOpeningOverwrite(Opening opening, int nZoneIndex); // this opening is specified for
    a particular zone by tsl
void setOpeningOverwrite(Opening opening, int bAllowModuleAssignmentOfDistributionBeams,
    int blInterruptBlockingRuns); // (added hsbCAD2013 build 18.1.35) default values of 2nd
    and 3rd parameter is TRUE.
void setOpeningOverwrite(Opening opening, PLine plOpening); // (added hsbCAD2013 build
    18.2.7) overwrite the opening pline for sip generation.

void setDistributionReferences(Opening opening, Point3d ptFromX, Point3d ptToX); // set
    reference points for the sheeting distribution, overwriting for a particular opening

void setBlockingRun(Point3d pt1, Point3d pt2, int nAmount, double dYEI, double dZEI,
    double dDeduct, double dTrench, double dMinLength, int bFront, int bBack, int
    bStaggered, int bFaceMounted, int bContinuous, Map subMap); // (added hsbCAD2013
    build 18.1.8) see SetBlockingRun

```

The following functions are implemented for the panel building (sips):

```

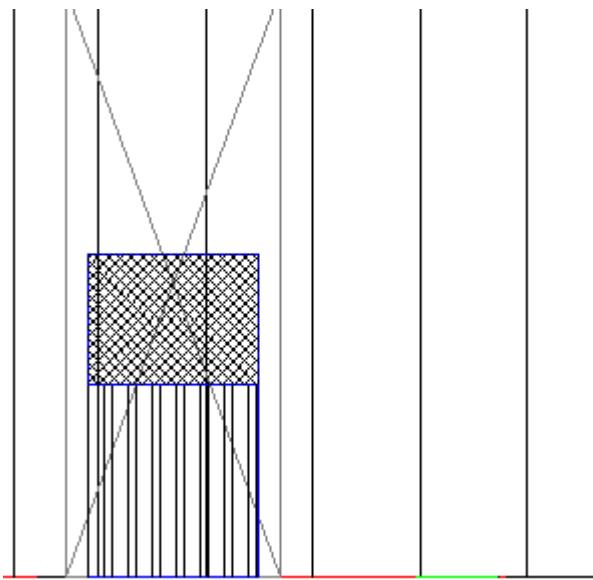
void setCutLocation(int nZoneIndex, Point3d ptLocationXY, Vector3d vecDirection); // define
    the location, in the element XY plane of the zone where the panel can be cut. The
    cut will run from the point in the vecDirection.
void setRoofEdgeDirective(int nZoneIndex, ElemRoofEdge edge); // define how the panel
    edge at that location should be generated using ElemRoofEdge.

```

When a location or range is specified, 3d points are used as argument, but only the X (and possibly Y) component in the elements coordinate system will be used.

---

*[Example O-type with insert implemented:*



```

Unit(1,"mm");

PropInt nZoneIndex(0,1,T("Zone index"));
PropDouble dStudDelta(0,U(500),T("Stud spacing"));

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

ElemZone ez = el.zone(nZoneIndex);
CoordSys cs = el.coordSys();
// move the _Pt0 point
_Pt0 -= cs.vecZ().dotProduct(_Pt0-cs.ptOrg())*cs.vecZ();

PLine plEl = el.plEnvelope();
Point3d pntsEl[] = plEl.intersectPoints(Line(cs.ptOrg(),cs.vecX()));
if (pntsEl.length()<2) return;
Point3d ptMin = pntsEl[0];
Point3d ptMax = pntsEl[pntsEl.length()-1];
double dPtY = cs.vecY().dotProduct(_Pt0-ptMin);
double dPtX = cs.vecX().dotProduct(_Pt0-ptMin);
Point3d ptBase = _Pt0 - dPtY*cs.vecY();

// add the script entity to the element group itself
assignToElementGroup(el,TRUE,0,'E');

```

```

// define the stud locations, and sheet cut location
double dElLen = cs.vecX().dotProduct(ptMax-ptMin);
for (double dLoc=0; dLoc<dElLen; dLoc+=abs(dStudDelta)) {
    Point3d ptCut = ptMin + dLoc*cs.vecX();
    el.setSheetCutLocation(nZoneIndex,ptCut);
    Point3d ptStud = ptCut;
    if (dLoc==0) ptStud += 0.5*el.dBeamWidth()*cs.vecX();
    el.setDistributionStudLocation(ptStud);
}

// define an area where no studs should be placed
double dDeltaNoStuds = U(1000);
Point3d ptFrom = ptBase - 0.5*dDeltaNoStuds*cs.vecX();
Point3d ptTo = ptBase + 0.5*dDeltaNoStuds*cs.vecX();
el.setRangeNoDistributionStud(ptFrom, ptTo);

// set area where studs can be placed anyway
double dDeltaAreaX = U(800);
double dDeltaAreaY = dPtY;
Point3d pt2 = ptBase + 0.5*dDeltaAreaY*cs.vecY(); // define center point for PLine
PLine plBeam(cs.vecZ());
plBeam.addVertex(pt2+0.5*dDeltaAreaX*cs.vecX()+0.5*dDeltaAreaY*cs.vecY());
plBeam.addVertex(pt2+0.5*dDeltaAreaX*cs.vecX()-0.5*dDeltaAreaY*cs.vecY());
plBeam.addVertex(pt2-0.5*dDeltaAreaX*cs.vecX()-0.5*dDeltaAreaY*cs.vecY());
plBeam.addVertex(pt2-0.5*dDeltaAreaX*cs.vecX()+0.5*dDeltaAreaY*cs.vecY());
plBeam.close();
plBeam.vis(2);
el.setAreaStud(plBeam);

// define area where there should not be any spacing
double dDeltaWX = U(800);
double dDeltaWY = U(600);
Point3d pt3 = ptBase +(0.5*dDeltaWY+dPtY)*cs.vecY(); // define center point for PLine
PLine plNoSheet(cs.vecZ());
plNoSheet.addVertex(pt3+0.5*dDeltaWX*cs.vecX()+0.5*dDeltaWY*cs.vecY());
plNoSheet.addVertex(pt3+0.5*dDeltaWX*cs.vecX()-0.5*dDeltaWY*cs.vecY());
plNoSheet.addVertex(pt3-0.5*dDeltaWX*cs.vecX()-0.5*dDeltaWY*cs.vecY());
plNoSheet.addVertex(pt3-0.5*dDeltaWX*cs.vecX()+0.5*dDeltaWY*cs.vecY());
plNoSheet.close();
plNoSheet.vis(1);
el.setAreaNoSheet(nZoneIndex,plNoSheet);

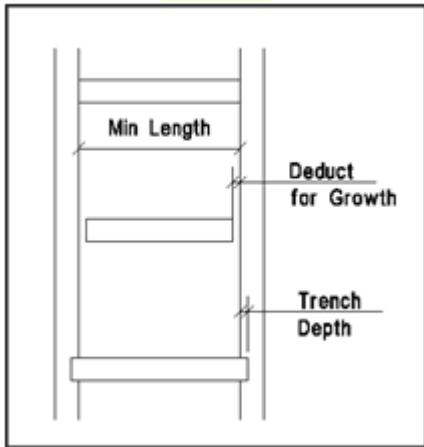
```

*—end example]*

#### 7.8.3.1 setBlockingRun

This construction directive, added in hsbCAD2013 build 18.1.8, specifies a line along the vecX direction of the element along which blocking pieces are added during generate construction.

```
void setBlockingRun(Point3d pt1, Point3d pt2, int nAmount, double dYEI, double dZEI,
double dDeduct, double dTrench, double dMinLength, int bFront, int bBack, int
bStaggered, int bFaceMounted, int bContinuous, Map subMap); // (added hsbCAD2013
build 18.1.8) set reference points and other parameters for blocking run
```



[Example O-type with insert implemented:

```
Unit(1, "mm");

PropInt nAmount(0,1,T("|Amount|"));
PropDouble dHeight(0,U(1000),T("|Height in element|"));
PropDouble dYEI(1,U(100),T("|Blocking dimension in element Y
direction|"));
PropDouble dZEI(2,U(30),T("|Blocking dimension in element z
direction|"));
PropDouble dDeduct(3,U(5),T("|Deduct (total gap left + right)|"));
PropDouble dTrench(4,U(10),T("|Trench depth into stud|"));
PropDouble dMinLength(5,U(0),T("|Minimum blocking length|"));
PropInt bFront(1,1,T("|Blocking on front of element (zone 1
side)|"));
PropInt bBack(2,0,T("|Blocking on back of element (zone -1 side)|"));
PropInt bStaggered(3,0,T("|Staggered|"));
PropInt bFaceMounted(4,0,T("|Face mounted|"));
PropInt bContinuous(5,0,T("|Continuous|"));

PropString strMaterial(0,"mymat",T("|Blocking material|"));
PropString strSubLabel2(1,"",T("|Blocking sub label 2|"));
PropString strName(2,"myname",T("|Blocking name|"));

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint(T("|Select first point|"));

    PrPoint ssP2(T("|Select second point|"),_Pt0);
}
```

```

    if (ssP2.go ()==_kOk)
        _PtG.append(ssP2.value ());

    showDialog ();
    return;
}

// check execute conditions
if (_Element.length ()==0) return;
Element el = _Element[0];
if (!el.bIsValid ()) return;
if (_PtG.length ()==0) return;

CoordSys cs = el.coordSys ();

// move the _Pt0 and _PtG[0] point to the element zone 0
double dx1 = cs.vecX ().dotProduct (_Pt0-cs.ptOrg ());
double dx2 = cs.vecX ().dotProduct (_PtG[0]-cs.ptOrg ());
_Pt0 = cs.ptOrg () + dx1*cs.vecX () + dHeight*cs.vecY ();
_PtG[0] = cs.ptOrg () + dx2*cs.vecX () + dHeight*cs.vecY ();

Point3d ptFrom = _Pt0;
Point3d ptTo = _PtG[0];

Map subMap;
subMap.setString ("material",strMaterial);
subMap.setString ("sublabel2",strSubLabel2);
subMap.setString ("name",strName);

// add the script entity to the element group itself
assignToElementGroup (el,TRUE,0,'E');

el.setBlockingRun (ptFrom, ptTo, nAmount, dYEl, dZEl, dDeduct,
dTrench, dMinLength,
bFront, bBack, bStaggered, bFaceMounted, bContinuous, subMap);

—end example]

```

#### 7.8.4 ElemNumber

The ElemNumber is a helper class to manipulate element numbers. It gets typically constructed with an [Element](#) its number.

```
ElemNumber elemnum(_Element0.number()); // construct with an elements his number
```

Which can then be used to change the element its number:

```
_Element0.setNumber(elemnum.composed());
```

```

class ElemNumber (added hsbCAD2012 build 17.0.45)
{
    ElemNumber(String strComposedNumber); // could come from Element.number(), and will set
                                              prefix, number, amountOfDigits and sequenceType at once

    String prefix() const;
    void setPrefix(String str);

    int number() const;
    void setNumber(int ii);

    int amountOfDigits() const;
    void setAmountOfDigits(int ii);

    int sequenceType() const;
    void setSequenceType(int nSequenceType);

    String composed() const;
    void setFromComposed(String strComposedNumber); // this will set prefix, number,
                                              amountOfDigits and sequenceType at once.
};

Value of nSequenceType should be one of the following
_kSTDigits
_kSTEExcelColumn
_kSTOddCharAABB

```

---

*[Example O-type TSL:*

```

if (_bOnInsert) {
    _Pt0 = getPoint();
    _Element.append(getElement());
    return;
}

reportMessage("\nElement code " + _Element0.code());
reportMessage("\nElement number " + _Element0.number());

ElemNumber elemNum(_Element0.number());
//elemNum.setFromComposed(_Element0.number()); // this is an
                                              alternative to using a constructor

reportMessage("\nElemNumber.prefix: " + elemNum.prefix());
reportMessage("\nElemNumber.number: " + elemNum.number());
reportMessage("\nElemNumber.amountOfDigits: " +
elemNum.amountOfDigits());
reportMessage("\nElemNumber.sequenceType: " +
elemNum.sequenceType());

```

---

```

    reportMessage("\nElemNumber.composed: " + elemNum.composed());

    String strIncreaseNumber = T("| Increase number, prefix and amount of
digits|");
    addRecalcTrigger(_kContext, strIncreaseNumber );
    if (_bOnRecalc && _kExecuteKey==strIncreaseNumber ) {
        elemNum.setPrefix(elemNum.prefix()+"A");
        elemNum.setNumber(elemNum.number()+1);
        elemNum.setAmountOfDigits(elemNum.amountOfDigits()+1);

        reportMessage("\nnew ElemNumber.composed: " + elemNum.composed());
        _Element0.setNumber(elemNum.composed());
    }

    String strChangeSequence = T("| Change sequence type|");
    addRecalcTrigger(_kContext, strChangeSequence );
    if (_bOnRecalc && _kExecuteKey==strChangeSequence ) {

        int ii = getInt(T("|Enter new sequence type (0/1/2)|")) % 3;
        elemNum.setSequenceType(ii);

        reportMessage("\nnew ElemNumber.composed: " + elemNum.composed());
        _Element0.setNumber(elemNum.composed());
    }

    String strDecreaseNumber = T("| Decrease number|");
    addRecalcTrigger(_kContext, strDecreaseNumber );
    if (_bOnRecalc && _kExecuteKey==strDecreaseNumber ) {
        elemNum.setNumber(elemNum.number()-1);

        reportMessage("\nnew ElemNumber.composed: " + elemNum.composed());
        _Element0.setNumber(elemNum.composed());
    }
}

```

*—end example]*

## 7.9 ElementWall

The term ElementWall refers to an instance of an ADT wall to which HSB wall element data is attached.

An entity can be casted into an ElementWall. However when the casting is not allowed, the resulting ElementWall will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ElementWall is derived from `Element`, it inherits all the member functions of Element as well. It can also be assigned to an Element.

When you link the TSL instance with an element, it will be automatically added to the predefined array `_Element`. To use the i-th element of this array as a valid ElementWall, the following can be used.

`ElementWall elWall = (ElementWall) _Element[i];`

```

if (elWall.bIsValid()) {
    // here use elWall as a valid ElementWall type
}

```

During `_bOnInsert`, the `getElement()` function can be used to query the user to select an element instance. With the above code sample, the casting to an `ElementWall` can then be done.

```

Element getElement();
Element getElement(String strPrompt);

```

The `cornerCleanupMode()` accepts one of the following predefined variables of type `int`:

```

_kCCNoCornerCleanup, _kCCStretchLast, _kCCStretchThis, _kCCMitre,
_kCCMitreOpenEnd, _kCCLogWallCrossing, _kCCSipOverlap, _kCCTConnection,
_kCCRRightHandShort, _kCCLeftHandShort

```

---

```

class ElementWall : Element { // see Element for base functions

    Element[] getConnectedElements() const;
    int setWallRoofLine(int bRemoveBevels) const;

    Point3d ptArrow() const; // returns the base point for the arrow representing the hsbCAD
                           // attached data
    Point3d setPtArrow(Point3d ptArrowNew); // sets the base point for the arrow representing
                                           // the hsbCAD attached data

    Point3d ptStartOutline() const;
    Point3d ptEndOutline() const;
    LineSeg segStartOutline() const;
    LineSeg segEndOutline() const;
    int setPtStartOutline(Point3d ptStart); // returns success of the operation
    int setPtEndOutline(Point3d ptEnd); // returns success of the operation
    int stretchOutlineTo(Plane planeVertical); // returns success of the operation

    int cuttingType(); // returns the cutting type: 2: back, 1: front, 0: both (added
                      // hsbCAD15.0.12 and hsbCAD14.1.27)
    void setCuttingType(int nCuttingType); // (added hsbCAD15.0.12 and hsbCAD14.1.27)

    int exposed(); // returns the exposed or external state: 0: FALSE, 1: TRUE (added
                  // hsbCAD17.0.41 and hsbCAD16.3.7)
    void setExposed(int nSet); // set to TRUE or FALSE (added hsbCAD17.0.41 and
                             // hsbCAD16.3.7)

    double maximumHeight() const; // added V22.1.119 and V23.3.3
    void setMaximumHeight(double dValue); // added V22.1.119 and V23.3.3, call
                                         // setWallRoofLine to activate
    String roofCodeExpression() const; // added V22.1.119 and V23.3.3
    void setRoofCodeExpression(String strValue); // added V22.1.119 and V23.3.3, call
                                                // setWallRoofLine to activate
}

```

---

```
static String cornerCleanupMode(int nCornerCleanupMode); // (added hsbCAD20.0.5) returns
the translated string of the corner cleanup mode
static void cornerCleanup(<Entity>[] arElements, int nCCModePerpendicular, int
nCCModeAngled, int bAdjustWallEntities, double dGap, double dSetBack,
double dLogCrossing, int bLogCrossingStretchFromCenterLine, int
bUseWallDirectionForThisLast); // added hsbCAD20.0.5
};
```

---

**Element[]** getConnectedElements() const;

This routine collects all wall elements from the drawing and filters out those that have a vertexpoint of their wall outline on the wall outline of this wall, or vice versa. So wall elements are considered to be connected if this occurs.

**int** setWallRoofLine(**int** bRemoveBevels) const;

This routine triggers the recalculation of the wall roof line. It is equivalent with calling the hsbCad command hsb\_wallroofline. The maximum height, as well as the roofplane codes from the wall description are used.

**void** cornerCleanup(**Element[]** arElements, **int** nCCModePerpendicular, **int** nCCModeAngled, **int**
bAdjustWallEntities, **double** dGap, **double** dSetBack, **double** dLogCrossing, **int**
bLogCrossingStretchFromCenterLine, **int** bUseWallDirectionForThisLast);

The static method cornerCleanup allows to trigger the corner cleanup functionality, similar to the corner cleanup present in the Aca2Hsb. Different modes use different parameters.

- bUseWallDirectionForThisLast: the normal corner cleanup inside hsbCAD looks at start and end point of the walls to determine which wall is "this" and which wall is "last". But using it from TSL one can impose its own sequence by setting this flag to FALSE. In this case, the arElements sequence is used.

---

[Example O-type, with insert done inside script. Illustration of the getConnectedElements.

```
Unit(1, "mm");
if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length() == 0) return;
```

---

```

ElementWall elWall = (ElementWall) Element[0];
if (!elWall.bIsValid()) return;

// get the polyline of this element
PLine plEl = elWall.plOutlineWall();

// get the other connected wall elements
Element arEl[] = elWall.getConnectedElements();

// visualize something
for (int e=0; e<arEl.length(); e++) {
    CoordSys csOther = arEl[e].coordSys();
    Line ln(csOther.ptOrg(), csOther.vecX());
    ln.vis();
    Point3d pnts[] = plEl.intersectPoints(ln);
    if (pnts.length()>0) { // at least one intersection point
        Point3d pt = pnts[0];
        pt.vis();
    }
}

```

*—end example]*

*[Example O-type, with insert done inside script. Illustration of the ptArrow and setPtArrow.*

```

Unit(1, "mm");
if (_bOnInsert) {
    Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
ElementWall elWall = (ElementWall) Element[0];
if (!elWall.bIsValid()) return;

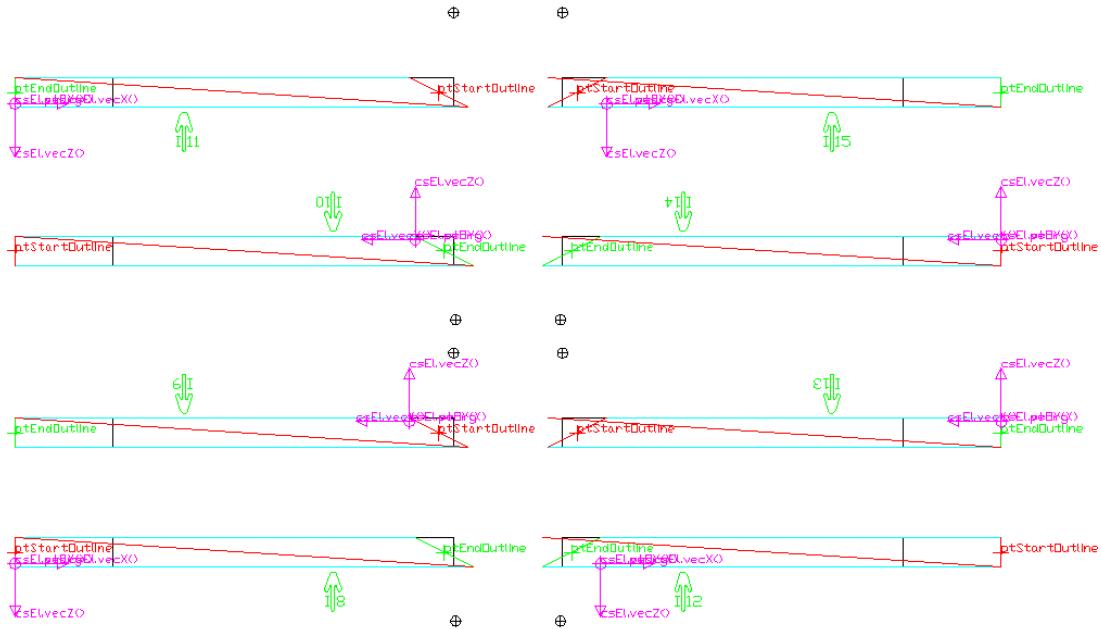
// set and get the arrow point of an ElementWall
elWall.setPtArrow(_Pt0);
Point3d ptArw = elWall.ptArrow();

Display dp(-1);
LineSeg seg(_Pt0, ptArw);
dp.draw(seg);

```

*—end example]*

*[Example O-type, with insert done inside script. Illustration of the get and set outline data.*



```

Unit(1,"mm");
if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
ElementWall elWall = (ElementWall) _Element[0];
if (!elWall.bIsValid()) return;

String strSetPtStart = T("setPtStartOutline");
addRecalcTrigger(_kContext, strSetPtStart );
if (_bOnRecalc && _kExecuteKey==strSetPtStart ) {
    elWall.setPtStartOutline(_Pt0);
}

String strSetPtEnd = T("setPtEndOutline");
addRecalcTrigger(_kContext, strSetPtEnd );
if (_bOnRecalc && _kExecuteKey==strSetPtEnd ) {
    elWall.setPtEndOutline(_Pt0);
}

String strStretchOutlineTo = T("stretchOutlineTo");
addRecalcTrigger(_kContext, strStretchOutlineTo );
if (_bOnRecalc && _kExecuteKey==strStretchOutlineTo ) {
    Vector3d vecN = _XU+2*_YU;
    vecN.normalize();
    Plane plVertical(_Pt0,vecN);
    elWall.stretchOutlineTo(plVertical);
}

```

```

}

Point3d ptStartOutline = elWall.ptStartOutline();
Point3d ptEndOutline = elWall.ptEndOutline();
ptStartOutline.vis(1);
ptEndOutline.vis(3);

LineSeg segStartOutline = elWall.segStartOutline();
LineSeg segEndOutline = elWall.segEndOutline();
segStartOutline.vis(1);
segEndOutline.vis(3);

LineSeg segmentMinMax= elWall.segmentMinMax();
segmentMinMax.vis(1);

CoordSys csEl = elWall.coordSys();
csEl.vis(6);

```

*—end example]*

*[Example O-type, with insert done inside script. Illustration cornerCleanup.*

```

Unit(1, "mm");

String arModes[0];
int arModesInt[] = {_kCCNoCornerCleanup, _kCCStretchLast,
_kCCStretchThis, _kCCMitre, _kCCMitreOpenEnd,
_kCCLogWallCrossing, _kCCSipOverlap, _kCCTConnection,
_kCCRRightHandShort, _kCCLeftHandShort};
for (int i=0; i<arModesInt.length(); i++)
    arModes.append(ElementWall() .cornerCleanupMode(arModesInt[i]));

PropString pStretchModePerpen(0, arModes, T("|Stretch mode
perpendicular|"));
PropString pStretchModeAngled(1, arModes, T("|Stretch mode
angled|"));
PropInt pAdjustWallEntity(2, 0, T("|Adjust wall entities|"));
PropDouble pGap(0, 0, T("|Gap|"));
PropDouble pSetBack(1, 0, T("|Setback|"));
PropDouble pLogCrossing(2, 0, T("|Log crossing|"));
PropInt pLogCrossingFrom(3, 0, T("|Log crossing from centerline|"));

int nStretchModePerpen = arModesInt[arModes.find(pStretchModePerpen,
0)];
int nStretchModeAngled= arModesInt[arModes.find(pStretchModeAngled,
0)];

if (_bOnInsert) {
    showDialog();
    _Pt0 = getPoint();
}

```

```

String strCreateModel = T("|Trigger corner cleanup|");
addRecalcTrigger(_kContext, strCreateModel );
if (_bOnInsert || (_bOnRecalc && _kExecuteKey==strCreateModel) )
{
    Entity ents[0];

    PrEntity ssE(T("\n|Select a set of wall elements|"),Wall());
    if (ssE.go()) {
        ents = ssE.set();
    }
    int bUseWallDirectionForThisLast = FALSE;
    ElementWall().cornerCleanup(ents, nStretchModePerpen,
nStretchModeAngled, pAdjustWallEntity, pGap,
    pSetBack, pLogCrossing, pLogCrossingFrom,
bUseWallDirectionForThisLast);
}

```

*—end example*

## 7.10 ElementLog

The term ElementLog refers to an instance of an ADT wall to which HSB log wall element data is attached.

An entity can be casted into an ElementLog. However when the casting is not allowed, the resulting ElementLog will become invalid. This can be checked with the blsValid() function of Entity.

Because ElementLog is derived from [ElementWall](#), so also from [Element](#), it inherits all the member functions of ElementWall and Element as well. It can also be assigned to an Element.

When you link the TSL instance with an element, it will be automatically added to the predefined array \_Element. To use the i-th element of this array as a valid ElementLog, the following can be used.

```

ElementLog elLog = (ElementLog) _Element[i];
if (elLog.blsValid()) {
    // here use elLog as a valid ElementLog type
}

```

During \_bOnInsert, the getElement() function can be used to query the user to select an element instance. With the above code sample, the casting to an ElementLog can then be done.

```

Element getElement();
Element getElement(String strPrompt);

```

---

```
class ElementLog : ElementWall { // see ElementWall and Element for base functions
```

```

double dFirstLog() const;
void setDFirstLog(double dValue);

double dTongue() const;
void setDTongue(double dValue);
double dGroove() const;
void setDGroove(double dValue);
double dGap() const;
void setDGap(double dValue);

String extrProfile() const;
void setExtrProfile(String strVal);

int bCutHalfLogTop() const;
void setBCutHalfLogTop(int nVal);
int bCutHalfLogBottom() const;
void setBCutHalfLogBottom(int nVal);

double dVisibleHeight() const;

double[] arLogYValues(double dWallHeight) const; // log axis Y location, assuming the
// first log dimension as any other log would be dVisibleHeight.
double dHeightFromCourseNr(int nCourseNr) const; // assuming the first log dimension as
// any other log would be dVisibleHeight.

void setContourPtsLeft(Point3d[] pnts, double dHeight);
void setContourPtsRight(Point3d[] pnts, double dHeight);

LogCourse[] logCourses() const; // see LogCourse // DOES NOT SUPPORT XREFS

void setOpeningOverwrite(Opening opening); // added 23.8.19 and 24.1.11. If overwrite set,
// log generation will ignore opening.
}

```

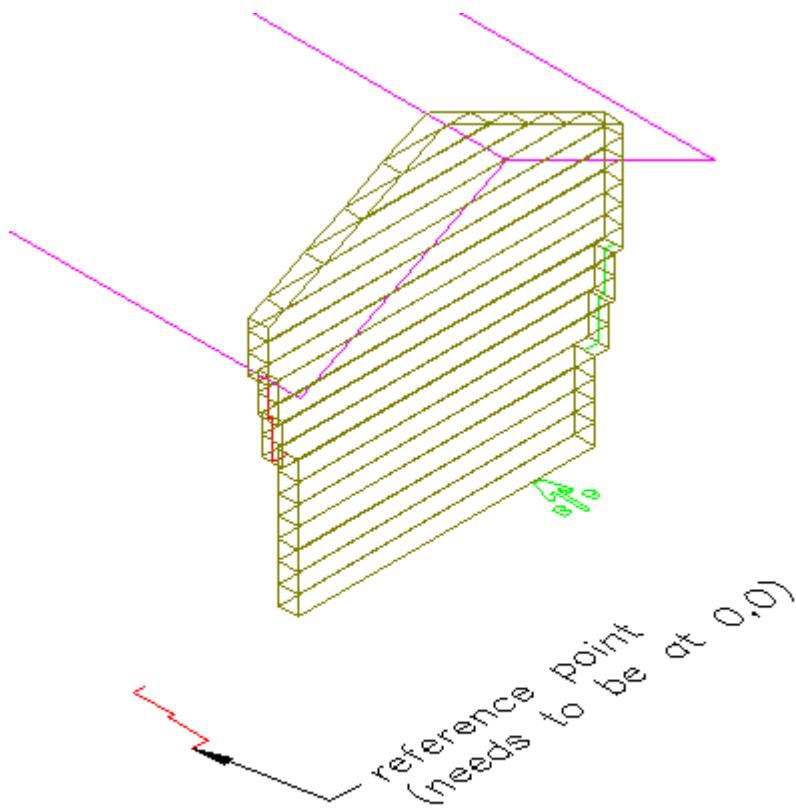
---

```

void setContourPtsLeft(Point3d[] pnts, double dHeight);
void setContourPtsRight(Point3d[] pnts, double dHeight);

```

Both of the above routines can be considered as log element modifiers. When called upon the log element, the routine will store inside the element a list of points to be used as part of the wall envelope, or contour. Doing so, allows the user to specify the contour by which the logs are cut, at log generated time. The points will influence the log generation process. The contour points do not have any effect after the logs are generated.



[Example O-type:

```

Unit(1, "mm");
PropInt iStartHgt(0, 7, "Start course");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint("Select the point which determines the side");
    return;
}

// check execute conditions
if (_Element.length()==0) return;
ElementLog elLog = (ElementLog) _Element[0];
if (!elLog.blsValid()) return;

// coordsys of the element, determination of side, and repositioning of point
CoordSys csEl= elLog.coordSys(_Pt0);
_Pt0 = csEl.ptOrg();
Point3d ptMid; ptMid.setToAverage(elLog.pOutlineWall().vertexPoints(TRUE));
Vector3d vecDir = _Pt0 - ptMid;
char cSide = 'L';
if (vecDir.dotProduct(csEl.vecX())>0) cSide = 'R';

```

```

// calculate a new set of points to serve as right contour
double dStep = elLog.dVisibleHeight() + elLog.dGap();
Point3d pnts[0]; // initial array length h
pnts.append(_PtW); // reference point is 0,0,0
pnts.append( Point3d(U(150),0,0)); // go to the right
pnts.append( Point3d(U(150),2*dStep ,0) ); // go up 2 logs
pnts.append( Point3d(U(200),2*dStep ,0) ); // go to the right
pnts.append( Point3d(U(200),4*dStep ,0) ); // go up another 1 logs
pnts.append( Point3d(U(300),4*dStep ,0) ); // go to the right

if (cSide=='L')
    elLog.setContourPtsLeft(pnts, elLog.dHeightFromCourseNr(iStartHgt));
else {
    elLog.setContourPtsRight(pnts, elLog.dHeightFromCourseNr(iStartHgt));
}

assignToGroups(elLog);

—end example]

```

---

[Example O-type, with insert done inside script:

```

Unit(1,"mm");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
ElementLog elLog = (ElementLog) _Element[0];
if (!elLog.bIsValid()) return;

reportNotice("\n\nElement code " +elLog.code() ); // access Element functions
reportNotice("\nElement number " + elLog.number());
reportNotice("\nElement dBeamWidth " + elLog.dBeamWidth());
reportNotice("\nElement dBeamHeight " + elLog.dBeamHeight());

reportNotice("\nElementLog dFirstLog " + elLog.dFirstLog());
reportNotice("\nElementLog dTongue " + elLog.dTongue());
reportNotice("\nElementLog dGroove " + elLog.dGroove());
reportNotice("\nElementLog dGap " + elLog.dGap());
reportNotice("\nElementLog extrProfile " + elLog.extrProfile());
reportNotice("\nElementLog bCutHalfLogTop " + elLog.bCutHalfLogTop());
reportNotice("\nElementLog bCutHalfLogBottom " + elLog.bCutHalfLogBottom());
reportNotice("\nElementLog dVisibleHeight " + elLog.dVisibleHeight());

```

---

```

// calculate the height of the wall
double dWallHeight = 0;
CoordSys elCs = elLog.coordSys();
PLine pl = elLog.plEnvelope();
Point3d pnts[] = pl.intersectPoints(elCs.ptOrg(), elCs.vecY());
if (pnts.length()==2) dWallHeight = Vector3d(pnts[0]-pnts[1]).length();

// calculate the center points of the logs
double arLogY[] = elLog.arLogYValues(dWallHeight);
for (int i=0; i<arLogY.length(); i++) {
    reportNotice("\nLog center " + i + " Y value " + arLogY[i]);
}

/* This script could output
Element code k
Element number 14
Element dBeamWidth 100
Element dBeamHeight 200
ElementLog dFirstLog 0
ElementLog dTongue 22
ElementLog dGroove 33
ElementLog dGap 10
ElementLog extrProfile Test
ElementLog bCutHalfLogTop 0
ElementLog bCutHalfLogBottom 1
ElementLog dVisibleHeight 178
Log center 0 Y value 100
Log center 1 Y value 310
Log center 2 Y value 520
Log center 3 Y value 730
Log center 4 Y value 940
Log center 5 Y value 1150
Log center 6 Y value 1360
Log center 7 Y value 1570
Log center 8 Y value 1780
Log center 9 Y value 1990
*/
—end example]

```

### 7.10.1 LogCourse

The set of beams, logs, of a logwall, can be sorted into log courses. Each LogCourse has a subset of the beams of the log wall.

```
LogCourse lgCourses[] = logEl.logCourses(); // query the log courses from a ElementLog
```

---

```

class LogCourse { // DOES NOT SUPPORT XREFS

    Beam[] beam() const; // logs are ordered from left to right, in the Element vecX direction

    double dYMin() const;
    double dYMax() const;

};

```

---

[Example O-type, with insert done inside script:

```

Unit(1, "mm");
PropInt pIndCourse(0, 0, "Log course index");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
ElementLog elLog = (ElementLog) _Element[0];
if (!elLog.bIsValid()) return;

// get the log courses from the ElementLog
LogCourse lgCourses[] = elLog.logCourses();

// report the list of log course Y min and max values
reportMessage("\n\nNumber of courses: "+lgCourses.length());
for (int l=0; l<lgCourses.length(); l++) {
    LogCourse lgCourse = lgCourses[l];
    reportMessage("\nLogCourse from "+lgCourse.dYMin()+" to "
    "+lgCourse.dYMax());
}

// visualize the logs in one specific LogCourse, determined by a
// property
if (pIndCourse>=0 && pIndCourse<lgCourses.length()) {

    Vector3d vecMove = U(1000)*elLog.vecZ();
    Display dp(-1);
    int nColor = 0;
    Beam beams[] = lgCourses[pIndCourse].beam();
    for (int b=0; b<beams.length(); b++) {
        Body bdShow = beams[b].envelopeBody(); // just take
        envelopeBody
}

```

---

```

        bdShow.transformBy(vecMove); // move away from the original
        location
        nColor++; // increment color for each so we can see the order
        dp.color(nColor);
        dp.draw(bdShow);
    }

    if (beams.length()==0) { // in case no beams found, display my
        name
        dp.draw(scriptName(),_Pt0,_XU,_YU,1,1);
    }
}

```

*—end example]*

## 7.11 ElementWallSF

The term ElementWallSF refers to an instance of an ADT wall to which HSB stick frame wall element data is attached.

An entity can be casted into an ElementWallSF. However when the casting is not allowed, the resulting ElementWallSF will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ElementWallSF is derived from [ElementWall](#), so also from [Element](#), it inherits all the member functions of ElementWall and Element as well. It can also be assigned to an Element.

When you link the TSL instance with an element, it will be automatically added to the predefined array `_Element`. To use the i-th element of this array as a valid ElementWallSF, the following can be used.

```

ElementWallSF elWallSF = (ElementWallSF) Element[i];
if (elWallSF.blsValid()) {
    // here use elWallSF as a valid ElementWallSF type
}

```

During `_bOnInsert`, the `getElement()` function can be used to query the user to select an element instance. With the above code sample, the casting to an ElementWallSF can then be done.

```

Element getElement();
Element getElement(String strPrompt);

```

---

```

class ElementWallSF : ElementWall { // see ElementWall and Element for base functions

    double spacingBeam() const;
    void setSpacingBeam(double dValue);
    double deltaDistribution() const;
    void setDeltaDistribution(double dValue);
}

```

---

```

int forceLevelDetail() const;
void setForceLevelDetail(int nVal);
int distributionType() const;
void setDistributionType(int nVal);
int distributionZone() const;
void setDistributionZone(int nVal);

String constrDetailTop() const;
String constrDetailBottom() const;
String constrDetailLeft() const;
String constrDetailRight() const;
String constrDetailTopLeft() const;
String constrDetailTopRight() const;
void setConstrDetailTop(String strVal);
void setConstrDetailBottom(String strVal);
void setConstrDetailLeft(String strVal);
void setConstrDetailRight(String strVal);
void setConstrDetailTopLeft(String strVal);
void setConstrDetailTopRight(String strVal);

int loadBearing() const;
void setLoadBearing(int nVal);

static String findDescriptionForDetCode(String strDetCode);
static Map detCodeMap(); // 5April2012: added 18.1.22, 17.2.18
};

```

---

```

double spacingBeam() const;
void setSpacingBeam(double dValue);

```

SpacingBeam refers to the spacing value of the distribution, entered in the Wall definition dialog.

```
static String findDescriptionForDetCode(String strDetCode);
```

The findDescriptionForDetCode method reads the <CompanyWall>\DetCode.dat file, and searches the strDetCode. If the value is found, it returns the found description. If the value is not found, an empty string is returned.

```
eg: reportMessage ("\n"+
ElementWallSF().findDescriptionForDetCode ("AS0") );
```

---

[Example O-type, with insert done inside script:

```
Unit(1, "mm");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// check execute conditions
if (_Element.length() == 0) return;
ElementWallSF elWallSF = (ElementWallSF) _Element[0];
if (!elWallSF.bIsValid()) return;

elWallSF.setForceLevelDetail(1);
elWallSF.setSpacingBeam(U(20));
elWallSF.setDeltaDistribution(U(30));

reportNotice("\n\nElement code " + elWallSF.code()); // access
Element functions
reportNotice("\nElement number " + elWallSF.number());
reportNotice("\nElement dBeamWidth " + elWallSF.dBeamWidth());
reportNotice("\nElement dBeamHeight " + elWallSF.dBeamHeight());

reportNotice("\nElementWallSF spacingBeam " +
elWallSF.spacingBeam());
reportNotice("\nElementWallSF deltaDistribution " +
elWallSF.deltaDistribution());
reportNotice("\nElementWallSF forceLevelDetail " +
elWallSF.forceLevelDetail());

/////
// get the zone from the element
ElemZone ez = elWallSF.zone(1);

// use the untranslated name, case not sensitive, to access the
variable
ez.setDVar("First sheet", U(20));
double dVal = ez.dVar("First sheet");

// set the zone to the element
elWallSF.setZone(1, ez);

—end example]
```

[Example O-type, with insert done inside script:

```
if (_bOnInsert) {

    Map mpDet = ElementWallSF().detCodeMap();
    reportMessage("\ndetCode map: " + mpDet.getDxContent(FALSE));
```

```

        eraseInstance();
        return;
    }

—end example]

```

## 7.12 ElementRoof

The term ElementRoof refers to an instance of the hsbCad entity roof/floor element.

An entity can be casted into an ElementRoof. However when the casting is not allowed, the resulting ElementRoof will become invalid. This can be checked with the blsValid() function of Entity.

Because ElementRoof is derived from [Element](#), it inherits all the member functions of Element as well. It can also be assigned to an Element.

When you link the TSL instance with an element, it will be automatically added to the predefined array \_Element. To use the i-th element of this array as a valid ElementRoof, the following can be used.

```

ElementRoof elRoof = (ElementRoof) _Element[i];
if (elRoof.blsValid()) {
    // here use elRoof as a valid ElementRoof type
}

```

During \_bOnInsert, the getElement() function can be used to query the user to select an element instance. With the above code sample, the casting to an ElementRoof can then be done. But also the getElementRoof() function is available.

```

ElementRoof getElementRoof();
ElementRoof getElementRoof(String strPrompt);

```

```

class ElementRoof : Element { // see Element for base functions

    CoordSys coordSysHsb() const; // mirrorable coordsys added since V27

    String beamDistribution() const;
    void setBeamDistribution(String strVal); // R,L,M

    double dBeamSpacing() const;
    void setDBeamSpacing(double dValue);

    // the base class Element, contains the dBeamWidth, dBeamHeight, dBeamExtrProfile
    double dBeamDirection() const; // value in degrees, is calculated from the vecY
}

```

```
void setVecZ(Vector3d vec); // normal to the ElementRoof
void setVecY(Vector3d vec); // main beam direction, will change the dBeamDirection

ElemRoofEdge[] elemRoofEdges() const; // get the list of ElemRoofEdge
void setElemRoofEdges(ElemRoofEdge[] edges); // set the list of ElemRoofEdge

void dbCreate(Group group, PLine pline);
void setVertexPoints(Point3d[] pts); // added v19.0.60, points will be projected to existing
plane

double dReferenceHeight() const;
void setDReferenceHeight(double dValue);

int bPurlin() const;
void setBPurlin(int bSet); // set to TRUE or FALSE

String insulation() const;
void setInsulation(String strVal);

String constrDetailRidge() const;
void setConstrDetailRidge(String strVal);
String constrDetailHip() const;
void setConstrDetailHip(String strVal);
String constrDetailLeft() const;
void setConstrDetailLeft(String strVal);
String constrDetailRight() const;
void setConstrDetailRight(String strVal);
String constrDetailValley() const;
void setConstrDetailValley(String strVal);
String constrDetailOverhang() const;
void setConstrDetailOverhang(String strVal);

double dKneeWallHeight() const;
void setDKneeWallHeight(double dValue);
String constrDetailKneeWall() const;
void setConstrDetailKneeWall(String strVal);
double dKneeWallHeight2() const;
void setDKneeWallHeight2(double dValue);
String constrDetailKneeWall2() const;
void setConstrDetailKneeWall2(String strVal);

double dWallPlateHeight() const;
void setDWallPlateHeight(double dValue);
String constrDetailWallPlate() const;
void setConstrDetailWallPlate(String strVal);
double dWallPlateHeight2() const;
void setDWallPlateHeight2(double dValue);
String constrDetailWallPlate2() const;
void setConstrDetailWallPlate2(String strVal);

double dStrutHeight() const;
void setDStrutHeight(double dValue);
```

```

String constrDetailStrut() const;
void setConstrDetailStrut(String strVal);
double dStrutHeight2() const;
void setDStrutHeight2(double dValue);
String constrDetailStrut2() const;
void setConstrDetailStrut2(String strVal);

String tileLathDistribution() const;
void setTileLathDistribution(String strVal);
String counterLathDistribution() const;
void setCounterLathDistribution(String strVal);
double dCounterLathSpacing() const;
void setDCounterLathSpacing(double dValue);
double dCounterLathHeight() const;
void setDCounterLathHeight(double dValue);
double dCounterLathWidth() const;
void setDCounterLathWidth(double dValue);

// the following methods were added hsbCAD2011 (build 16.0.52) and hsbCAD2010 (build
15.3.25)
String[] getListOfCatalogNames(int bFloorNotRoof) const; // return the list of catalog entry
names for the roof or floor catalog
int setValuesFromCatalog(String strCatalogEntryName, int bFloorNotRoof); // return TRUE if
successful

int blsAFloor() const; // return TRUE if this is a floor (added hsbCAD2011 build 16.0.52 and
hsbCAD2010 build 15.3.25)

};

```

---

*[Example O-type, with insert done inside script:*

```

Unit(1, "mm");
if (_bOnInsert) {
    _Element.append(getElementRoof());
    _Pt0 = getPoint();
    return;
}

if (_Element.length() == 0) return;
ElementRoof elRoof = (ElementRoof) _Element[0];
if (!elRoof.bIsValid()) return;

// get the polyline of this element
PLine plEl = elRoof.plEnvelope();

CoordSys cs = elRoof.coordSys();

```

```
Line ln(cs.ptOrg(), cs.vecX());
cs.vis();
plEl.vis();
```

—end example]

[Example O-type, with insert done inside script, illustrate the dbCreate of a new one.

```
Unit(1, "mm");
if (_bOnInsert) {

    Group gr = _kCurrentGroup;
    if ( (gr.namePart(0)== "") || (gr.namePart(1)== "") ) {
        reportMessage("\nMake a floor group current before inserting
this TSL.");
        eraseInstance(); return;
    }
    gr.setNamePart(2, "tsl1"); // set last part of group

    EntPLine plent = getEntPLine();
    PLine pl = plent.getPLine();

    Vector3d vecUcsY = _YU; // Y vector of UCS at time of insert

    ElementRoof er;
    er.dbCreate(gr, pl); // define geometry, and create element group
    er.setVecY(vecUcsY); // set main beam direction

    _Element.append(er); // add to persistent array for later use

    return;
}

if (_Element.length()==0) return;
ElementRoof elRoof = (ElementRoof) _Element[0];
if (!elRoof.bIsValid()) return;

_Pt0 = elRoof.ptOrg();
```

—end example]

[Example O-type, illustrating the use of the catalog reading:

```
Unit(1, "mm");
if (_bOnInsert) {
    _Element.append(getElementRoof());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
ElementRoof elRoof = (ElementRoof) _Element[0];
```

```

if (!elRoof.bIsValid()) return;

// get the polyline of this element
PLine plEl = elRoof.plEnvelope();

CoordSys cs = elRoof.coordSys();
Line ln(cs.ptOrg(),cs.vecX());
cs.vis();
plEl.vis();

reportMessage("\n ");
reportMessage("\n bIsAFloor: " + elRoof.bIsAFloor());

// collect all the catalog values, and report them
int bFloorNotRoof= elRoof.bIsAFloor();
String arEntries[] = elRoof.getListOfCatalogNames(bFloorNotRoof);
for (int e=0; e<arEntries.length();e++) {
    reportMessage("\n entry["+e+"]= " + arEntries[e]);
}

String strLoadCatalogGreen = T("Load catalog entry");
addRecalcTrigger(_kContext, strLoadCatalogGreen );
if (_bOnRecalc && _kExecuteKey==strLoadCatalogGreen)
{
    int nIndex = getInt("enter index of catalog to read");
    if (nIndex>=0 && nIndex<arEntries.length())
    {
        String strEntry = arEntries[nIndex];
        reportMessage("\nReading catalog called: "+strEntry);
        elRoof.setValuesFromCatalog(strEntry,bFloorNotRoof);
    }
}

reportMessage("\n detailRidge: " + elRoof.constrDetailRidge());
reportMessage("\n detailHip: " + elRoof.constrDetailHip());
reportMessage("\n detailLeft: " + elRoof.constrDetailLeft());
reportMessage("\n detailRight " + elRoof.constrDetailRight());
reportMessage("\n detailValley: " + elRoof.constrDetailValley());
reportMessage("\n detailOverhang: " + elRoof.constrDetailOverhang());
reportMessage("\n detailWallPlate: " +
elRoof.constrDetailWallPlate());
reportMessage("\n detailKneeWall: " + elRoof.constrDetailKneeWall());

```

*—end example*

### 7.12.1 ElemRoofEdge

Each roof / floor element consists of a closed ring of edges. Each edge has a starting point, and a holds the data for the construction generation of that edge. That edge data is stored in an array of ElemRoofEdges. An ElemRoofEdge has a node, the second/other node, which is actually the next edge node, a string containing the construction detail, and a flag if the construction info is the default for the roof element, or if it is overwritten for the edge. In addition to this, the ElemRoofEdge has also a direction, which is the vector along the edge, in the counter clockwise direction, either starting from the node, or starting from the other node.

This is not a tool. Every roof/ floor element has a number of ElemRoofEdge as member variables. A list of ElemRoofEdge can be constructed through querying the roof element. The complete list of ElemRoofEdge can be overwritten by calling the setElemRoofEdges. However, when doing that, only a subset of the information will be used: ptNode, blsDefault, and if not default, constrDetail. All the other data is determined through the collection of ElemRoofEdges, by the [ElementRoof](#).

```
ElemRoofEdge edges[] = roofElement.elemRoofEdges();
roofElement.setElemRoofEdges(ElemRoofEdge edges[]);
```

The ElemRoofEdge is also used in the [Element](#)::setRoofEdgeDirective.

---

```
class ElemRoofEdge {

    ElemRoofEdge(Point3d ptNode, int blsDefault, String strConstrDetail);

    Point3d ptNode() const;
    void setPtNode(Point3d pt);

    Point3d ptNodeOther() const;
    void setPtNodeOther(Point3d pt);

    Vector3d vecDir() const;
    void setVecDir(Vector3d vec);

    int blsDefault() const;
    void setBlsDefault(int bSet);

    String constrDetail() const;
    void setConstrDetail(String str);

    Vector3d vecBevelNormal() const; // in use in the Element::setRoofEdgeDirective.
    void setVecBevelNormal(Vector3d vec);

    Vector3d dGap() const; // in use in the Element::setRoofEdgeDirective.
    void setDGap(double dd);

};
```

[Example O-type with insert implemented:

```
U(1,"mm");

if (_bOnInsert) {
    _Element.append(getElementRoof());
    _Pt0 = getPoint();
    return;
}

if (_Element.length()==0) return;
ElementRoof elRoof = (ElementRoof) _Element[0];
if (!elRoof.blsValid()) return;

ElemRoofEdge edges[] = elRoof.elemRoofEdges();

for (int t=0; t<edges.length(); t++) {

    ElemRoofEdge edge = edges[t];
    reportNotice("\n\n ptNode: " + edge.ptNode());
    reportNotice("\n ptNodeOther: " + edge.ptNodeOther());
    reportNotice("\n vecDir: " + edge.vecDir());
    reportNotice("\n blsDefault: " + edge.blsDefault());
    reportNotice("\n constrDetail: " + edge.constrDetail());

    edges[t].setPtNode(edge.ptNode()+U(100)*_XU);
    edges[t].setPtNodeOther(edge.ptNodeOther()+U(100)*_XU);
    edges[t].setVecDir(edge.vecDir());
    edges[t].setBlsDefault(elRoof.blsDefault());
    edges[t].setConstrDetail("blabla;joehoe");

}

elRoof.setElemRoofEdges(edges);
```

—end example]

## 7.13 ElementMulti

The term ElementMulti refers to an instance of a multi element entity.

An entity can be casted into an ElementMulti. However when the casting is not allowed, the resulting ElementMulti will become invalid. This can be checked with the blsValid() function of Entity.

Because ElementMulti is derived from [Element](#), it inherits all the member functions of Element as well. It can also be assigned to an Element.

When you link the TSL instance with a multi element, it will be automatically added to the predefined array `_Element`. To use the i-th element of this array as a valid `ElementMulti`, the following can be used.

```
ElementMulti elMulti = (ElementMulti) _Element[i];
if (elMulti.bIsValid()) {
    // here use elMulti as a valid ElementMulti type
}
```

During `_bOnInsert`, the `getElement()` function can be used to query the user to select an element instance. With the above code sample, the casting to an `ElementMulti` can then be done. But also the `getElementMulti()` function is available.

```
ElementMulti getElementMulti();
ElementMulti getElementMulti(String strPrompt);
```

---

```
class ElementMulti : Element { // see Element for base functions

    SingleElementRef[] singleElementRefs() const; // Return the list of SingleElementRef objects.

    void dbCreate(Group group, CoordSys csEcs); // added v21.0.66, group name can contain
                                                new element number part

    double dXMax() const; // added v21.0.66
    void setDXMax(double dValue); // added v21.0.66
    double dYMax() const; // added v21.0.66
    void setDYMax(double dValue); // added v21.0.66

    int insertSingleElement(AcadDatabase db, String strElemNumber, CoordSys csSwlnMw, Map
                           mapWallSource); // added v21.0.66, see AcadDatabase
};
```

---

```
SingleElementRef[] singleElementRefs() const;
```

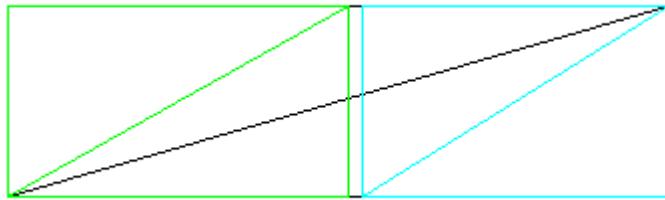
Return the list of [SingleElementRef](#) objects that belong to this `ElementMulti`.

---

*[Example O-type, with insert done inside script. Generated output for a particular `ElementMulti` selected:*

*Multi Element number: BF34  
 Element code: F  
 Element dBeamWidth: 120  
 Element dBeamHeight: 50  
 Single element references: 2  
 Single element number: 3*

*Single element number: 2*



```

U(1,"mm");

PropString pDwgName(0,"c:\\aa.dwg","full path of source dwg to work
with");
PropString pElemNr(1,"DG_GZ001","elem number in source dwg");
PropDouble pDXOffsetSW(0,U(1000), "X offset single element");
PropDouble pDYOffsetSW(1,U(100), "Y offset single element");
PropInt pRotate90(0,1, "rotate single element");

String strAction1 = T("|dbCreate ElementMulti|");
addRecalcTrigger(_kContext, strAction1 );
if (_bOnRecalc && _kExecuteKey==strAction1)
{
    CoordSys cs(_Pt0, _XU, _YU, _ZU);
    Group grp("aa\\bb\\aa12"); // group contains mw name
    ElementMulti elm;
    elm.dbCreate(grp, cs);
    _Element.append(elm);
}

String strAction2 = T("|shrink ElementMulti|");
addRecalcTrigger(_kContext, strAction2 );
if (_bOnRecalc && _kExecuteKey==strAction2)
{
    if (_Element.length() > 0)
    {
        int nInd = _Element.length() - 1;
        ElementMulti elMW = (ElementMulti) _Element[nInd];
        if (elMW.bIsValid())
        {
            elMW.setDXMax(0.8* elMW.dXMax());
            elMW.setDYMax(0.8* elMW.dYMax());
        }
    }
}

String strActionOpen = T("|openFromFile|");
addRecalcTrigger(_kContext, strActionOpen);
if (_bOnRecalc && _kExecuteKey==strActionOpen)
{
}

```

```

String strDwg = pDwgName;
AcadDatabase db;
int bOpened = db.openFromFile(strDwg);
reportMessage("\\n dwg: "+ strDwg + (bOpened ? " opened so" : " not opened so") + (db.bIsValid() ? " is valid." : " is not valid."));
}

String strAction3 = T("|add Single Element|");
addRecalcTrigger(_kContext, strAction3 );
if (_bOnRecalc && _kExecuteKey==strAction3)
{
    String strElemNr = pElemNr;
    AcadDatabase db(pDwgName);
    if (!db.bIsValid())
        reportMessage("First open database. At the end, do not forget to close it.");
    else if (_Element.length() > 0)
    {
        int nInd = _Element.length() - 1;
        ElementMulti elMW = (ElementMulti) _Element[nInd];
        if (elMW.bIsValid())
        {
            Vector3d vX = _XW;
            Vector3d vY = _YW;
            Vector3d vZ = _ZW;
            if (pRotate90)
            {
                vX = _YW;
                vY = -_XW;
            }
            CoordSys cs(Point3d(pDXOffsetSW, pDYOffsetSW, 0), vX, vY,
vZ);
            Map mp; // empty map
            elMW.insertSingleElement(db, strElemNr, cs, mp);
        }
    }
}

String strActionClose = T("|close|");
addRecalcTrigger(_kContext, strActionClose);
if (_bOnRecalc && _kExecuteKey==strActionClose)
{
    String strDwg = pDwgName;
    AcadDatabase dbE(strDwg);
    int bClosed = dbE.close();
    reportMessage("\\n dwg: "+ strDwg + (bClosed ? " closed" : " not closed."));
}

reportNotice("\\n\\nNumber of Elements: "+ _Element.length());
for(int el=0; el<_Element.length(); el++)
{
    ElementMulti elMW = (ElementMulti) _Element[el];
}

```

```

if (!elMW.bIsValid()) continue;

CoordSys csEl = elMW.coordSys();

reportNotice("\nMulti Element number: " + elMW.number()) ; // access Element functions
reportNotice("\nElement code: " + elMW.code());
reportNotice("\nElement dBeamWidth: " + elMW.dBeamWidth());
reportNotice("\nElement dBeamHeight: " + elMW.dBeamHeight());

Display dp(-1);
LineSeg segMM = elMW.segmentMinMax(); // get diagonal of multi element
dp.draw(segMM);

PLine rectAll; // create rectangle from diagonal
rectAll.createRectangle(segMM,csEl.vecX(),csEl.vecY());
PlaneProfile pp(rectAll);
pp.shrink(U(100));
dp.draw(pp);

// get the list of single element references
SingleElementRef arElemRef[] = elMW.singleElementRefs();

reportNotice("\nSingle element references: " +
arElemRef.length());

for (int e=0; e<arElemRef.length(); e++) {

    reportNotice("\n\tSingle element number: " +
arElemRef[e].number());

    int indColor = e+3; // start with color 3 (green)
    dp.color(indColor);

    CoordSys csElRef = arElemRef[e].coordSys(); // each elemRef has a coordSys
    LineSeg segMMRef = arElemRef[e].segmentMinMax(); // get diagonal of element ref
    dp.draw(segMMRef);

    PLine rectRef; // create rectangle from diagonal
    rectRef.createRectangle(segMMRef,csElRef.vecX(),csElRef.vecY());
    dp.draw(rectRef);
}
}

—end example]

```

### 7.13.1 SingleElementRef

Each multi element has an array of single element references. Such a object is of type SingleElementRef. Such a reference contains some data extracted from the elements when the multi element was made.

The complete list of SingleElementRef can be retrieved from a [ElementMulti](#).

```
SingleElementRef elRefs[] = multiElement.singleElementRefs();
```

For an example on how to use it, see [ElementMulti](#).

---

```
class SingleElementRef {
    String number() const;
    LineSeg segmentMinMax() const;
    CoordSys coordSys() const;

    Entity[] entitiesFromMultiElementBuild() const;

    // The subMapX set of methods added v22.1.71 and v23.0.45
    Map subMapX(String strKey) const; // see Map
    void setSubMapX(String strKey, Map mapNew);
    String[] subMapXKeys() const; // return list of available sub map keys
    void removeSubMapX(String strKey);
};
```

---

```
Entity[] entitiesFromMultiElementBuild() const;
```

Return the list of entities that still exist from the action Hsb\_MultiElementBuild. Of course after the multi element has been build, other entities could have been added, other entities could have been erased. Also entities could be modified, eg top plates of different single elements could be have been joined. So the list of entities that is returned by the function entitiesFromMultiElementBuild is most likely not an accurate list, but it is a list of entities that are probably somehow related to the SingleElementRef.

## 7.14 BlockRef

The term BlockRef refers to an instance of a block reference, sometimes called an insert. A BlockRef can also refer to a dynamic block.

An entity can be casted into a BlockRef. However when the casting is not allowed, the resulting BlockRef will become invalid. This can be checked with the `blsValid()` function of Entity.

---

Because BlockRef is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getBlockRef()` function can be used to query the user to select an instance.

```
BlockRef getBlockRef();
BlockRef getBlockRef(String strPrompt);
```

---

```
class BlockRef : Entity { // see Entity for base functions (added hsbCAD2012 build 17.0.44)

    CoordSys coordSys() const; // with normalized vectors
    CoordSys coordSysScaled() const; // with scaled vectors (added hsbCAD2017 build
        21.3.32)

    double dScaleX() const;
    double dScaleY() const;
    double dScaleZ() const;
    void setScale(double dScaleX, double dScaleY, double dScaleZ);

    String definition() const; // return name of Block
    void setDefinition(String strDefinition); // make the blockref refer to another block
        definition (added hsbCAD2017 build 21.3.32)

    int blsDynamic() const;
    String getResultBlock() const; // added since 24.1.49. In case of a dynamic block return
        name of the anonymous Block that contains the result, otherwise just return
        the definition.
    Map getDynBlockPropertyMap(int bAddOnlyShowable, int bIncludePoints) const; // returns a
        map filled with the properties defined in the dynamic block

    void dbCreate(CoordSys csEcsScaled, String strDefinition); // (added hsbCAD2017 build
        21.3.32)

    String[] getAttributeList() const; // returns list of attribute tags (added hsbCAD2017 build
        21.3.32)
    Map getAttributeMap() const; // returns tag value map (added hsbCAD2017 build 21.3.32)
    void setAttributesFromMap(Map mapAttributes); // (added hsbCAD2017 build 21.3.32)
    void createMissingAttributes(); // (added hsbCAD2017 build 21.3.32)
    void eraseAttributes(); // (added hsbCAD2017 build 21.3.32)
};
```

---

**CoordSys** coordSys() const;

Build the coordinate system of the BlockRef. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

---

[Example O-type, with insert done inside script:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    BlockRef blockRef = getBlockRef();
    _Entity.append(blockRef);

    return;
}

// check entity link
BlockRef blockRef;
if (_Entity.length()>0) blockRef = (BlockRef)_Entity[0];
if (!blockRef.bIsValid()) {
    eraseInstance();
    return;
}

int nIndFound = _BlockNames.find(blockRef.definition());
PropString strBlockName(0, _BlockNames, T("|Block name|"),
nIndFound);

blockRef.setDefinition(strBlockName); // will change the block
reference

CoordSys css =blockRef.coordSys();
css.vis();
Point3d pt0 = css.ptOrg();
Vector3d vecX = css.vecX();
Vector3d vecY = css.vecY();

int bOnlyShowable = TRUE;
int bIncludePoints = FALSE;
Map mpProps = blockRef.getDynBlockPropertyMap(bOnlyShowable,
bIncludePoints );
String strMpProps = mpProps.getDxContent(FALSE);

reportMessage("\nBlockRef definition: " +blockRef.definition() );
reportMessage("\nScale X: " +blockRef.dScaleX() );
reportMessage("\nScale Y: " +blockRef.dScaleY() );
reportMessage("\nScale Z: " +blockRef.dScaleZ() );
reportMessage("\nbIsDynamic: " +blockRef.bIsDynamic() );
reportMessage("\nstrMpProps: " +strMpProps );

// for debug view the map file
String strShowProps = T("|Show property map in explorer|");
addRecalcTrigger(_kContext, strShowProps );
if (_bOnRecalc && _kExecuteKey==strShowProps )
{

```

```

String strFileMap = _kPathPersonalTemp + "\\" + scriptName() +
".dxx";

int bOnlyShowable = getInt("Only showable (1/0): ");
int bIncludePoints = getInt("Include points (1/0): ");
Map mpPropsL= blockRef.getDynBlockPropertyMap(bOnlyShowable,
bIncludePoints);

mpPropsL.writeToDxxFile(strFileMap);
spawn_detach("",_kPathHsbInstall+"\Utilities\DXExplorer\
\DXExplorer.exe", strFileMap,"");
}

String strDbCreate = T("|dbCreate a new one|");
addRecalcTrigger(_kContext, strDbCreate );
if (_bOnRecalc && _kExecuteKey==strDbCreate )
{
    CoordSys csNew = blockRef.coordSysScaled();
    csNew.transformBy(_Pt0 - csNew.ptOrg()); // move to _Pt0 location
    String strDef = blockRef.definition();

    BlockRef refNew;
    refNew.dbCreate(csNew, strDef);
    if (refNew.bIsValid())
    {
        reportMessage("\n\nNew BlockRef created fine: " + strDef);

        refNew.createMissingAttributes(); // attributes are not
        automatically created
        Map attrMap = blockRef.getAttributeMap(); // get original
        attributes
        refNew.setAttributesFromMap(attrMap); // set attribute values
    }
}

String strDbCreateNoAtt = T("|dbCreate a new one no attributes|");
addRecalcTrigger(_kContext, strDbCreateNoAtt );
if (_bOnRecalc && _kExecuteKey==strDbCreateNoAtt )
{
    CoordSys csNew = blockRef.coordSysScaled();
    csNew.transformBy(_Pt0 - csNew.ptOrg()); // move to _Pt0 location
    String strDef = blockRef.definition();

    BlockRef refNew;
    refNew.dbCreate(csNew, strDef);
    if (refNew.bIsValid())
    {
        reportMessage("\n\nNew BlockRef created fine: " + strDef);
    }
}

String strChangeDef = T("|change def to xyz|");
addRecalcTrigger(_kContext, strChangeDef );

```

```

if (_bOnRecalc && _kExecuteKey==strChangeDef )
{
    blockRef.setDefinition("xyz"); // if block does not exist, the
blockref is not changed.
}

String strCreateAttr = T("|create missing attributes|");
addRecalcTrigger(_kContext, strCreateAttr );
if (_bOnRecalc && _kExecuteKey==strCreateAttr )
{
    blockRef.createMissingAttributes();
}

String strEraseAttr = T("|erase attributes|");
addRecalcTrigger(_kContext, strEraseAttr );
if (_bOnRecalc && _kExecuteKey==strEraseAttr )
{
    blockRef.eraseAttributes();
}

```

*—end example]*

## 7.15 ChildPanel

The term ChildPanel refers to an instance of a child panel entity used in the cnc output of [Sip](#) panels using master panels.

An entity can be casted into a ChildPanel. However when the casting is not allowed, the resulting ChildPanel will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ChildPanel is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getChildPanel()` function can be used to query the user to select a ChildPanel instance.

```

ChildPanel getChildPanel();
ChildPanel getChildPanel(String strPrompt);

```

---

```

class ChildPanel : Entity { // see Entity for base functions (added hsbCAD14.0.69)

    void dbCreate(Sip sip, CoordSys csEcs); // since 17.1.14 only ptOrg and vecX are used of
                                              the csEcs
    void dbCreate(Sip sip, Point3d ptInsert, Vector3d vecX); // added 17.1.14. ptInsert maps to
                                              the center of the Sip, vecX maps to the vecY of the Sip. VecZ of Sip is maps
                                              to the -WZ.

    CoordSys coordSys() const;

```

---

```

Sip sipEntity() const; // see Sip
CoordSys sipToMeTransformation() const;

int blsFlipped() const; // return flipped state
void setBlsFlipped(int nSet); // sets the flipped state to TRUE or FALSE

PLine plEnvelopeCnc() const; // cnc PLine of the Sip, but at the location of the ChildPanel

Vector3d woodGrainDirection(); // vector could be zero length if not defined (added
                                hsbcad20.1.6 and hsbcad21.0.9)

MasterPanel getMasterPanel() const; // added since 22.1.54 and 23.0.36. Returns the
                                    MasterPanel if any to which the ChildPanel belongs. Use bIsValid on the
                                    result.

};


```

---

**CoordSys** coordSys() const;

Build the coordinate system of the ChildPanel. It corresponds with the ECS (entity coordinate system) of the entity.

---

[Example O-type, with insert done inside script:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    ChildPanel cp = getChildPanel();
    _Entity.append(cp);

    return;
}

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_Entity.length() == 0) return;
ChildPanel cp = (ChildPanel) _Entity[0];
if (!cp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

```

---

```

String strChangeEntity = T("Change entity");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    cp.setBIsFlipped(!cp.bIsFlipped()); // reverse flipping
}

Sip sp = cp.sipEntity();
Body bd = sp.realBody();
CoordSys csTrans = cp.sipToMeTransformation();
bd.transformBy(csTrans);
bd.vis(1);

CoordSys cpS =cp.coordSys();
cpS.vis();
Vector3d vecZ = cpS.vecZ();
Vector3d vecX = cpS.vecX();
PLine plCnc = cp.plEnvelopeCnc();
plCnc.transformBy(U(333)*vecZ);
plCnc.vis(3);

String strLines[0];
strLines.append("Child panel ");
strLines.append("bIsFlipped: "+cp.bIsFlipped());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example]*

*[Example O-type, with insert done inside script to illustrate the dbCreate*

```

Unit(1,"mm");
if (_bOnInsert) {

    Sip sp = getSip();
    ChildPanel cp;
    Point3d ptLoc = getPoint();
    CoordSys csEcs(ptLoc,_XU,_YU,_ZU);
    cp.dbCreate(sp, csEcs);

    return;
}

```

—end example]

## 7.16 ClipVolume

The term ClipVolume refers to an instance of a section or elevation, aka AecCallOut in the drawing. It serves as the defining object for the [Section2d](#).

An entity can be casted into a ClipVolume. However when the casting is not allowed, the resulting ClipVolume will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ClipVolume is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getClipVolume()` function can be used to query the user to select a section clip volume instance.

```
ClipVolume getClipVolume();
ClipVolume getClipVolume(String strPrompt);
```

```
class ClipVolume : Entity { // see Entity for base functions (added hsbcad V21.4.36)
```

```
    CoordSys coordSys() const;
    Vector3d vecFromDir() const;

    String name() const;
    void setName(String strName);

    Point3d[] vertices() const;
    void setVertices(Point3d[] pnts); // points are projected on XY plane of coordSys.

    void dbCreate(Point3d[] vertices, Vector3d vecZ, int blsSectionLine);

    double lengthA() const;
    void setLengthA(double dVal);
    double lengthB() const;
    void setLengthB(double dVal);
    double angleA() const;
    void setAngleA(double dVal);
    double angleB() const;
    void setAngleB(double dVal);

    double lowerExtension() const; // not used if useModelExtentsForHeight is TRUE
    void setLowerExtension(double dVal);
    double height() const; // not used if useModelExtentsForHeight is TRUE
    void setHeight(double dVal);

    int isSectionLine() const; // its either a SectionLine or an ElevationLine in Autocad
    int useModelExtentsForHeight() const;
    void setUseModelExtentsForHeight(int bVal);

    Body clippingBody() const; // Body
    Body combinedClippedBodyOfEntities() const; // all bodies streamed by the viewSet of the
                                                // ClipVolume, combined into one, and clipped by the clippingBody.
```

```

String viewSet() const;

Entity[] includedEntities() const; // entities that are used in the Section2d entity.
Entity[] exclusionEntities() const;
Entity[] entitiesInClipVolume() const; // does not contain the exclusion entities. Default
    bExplodeXrefBlocks is FALSE.
Entity[] entitiesInClipVolume(int bExplodeXrefBlocks) const; // (added since V22.1.72 and
    V23.0.45) if bExplodeXrefBlocks is TRUE then return the entities inside the xrefs,
    instead of the BlockRef entities itself.

void setIncludedEntities(<Entity>[] ents); // this happens automatically whenever a Section2d is
    refreshed.
void setExclusionEntities(<Entity>[] ents);
};

```

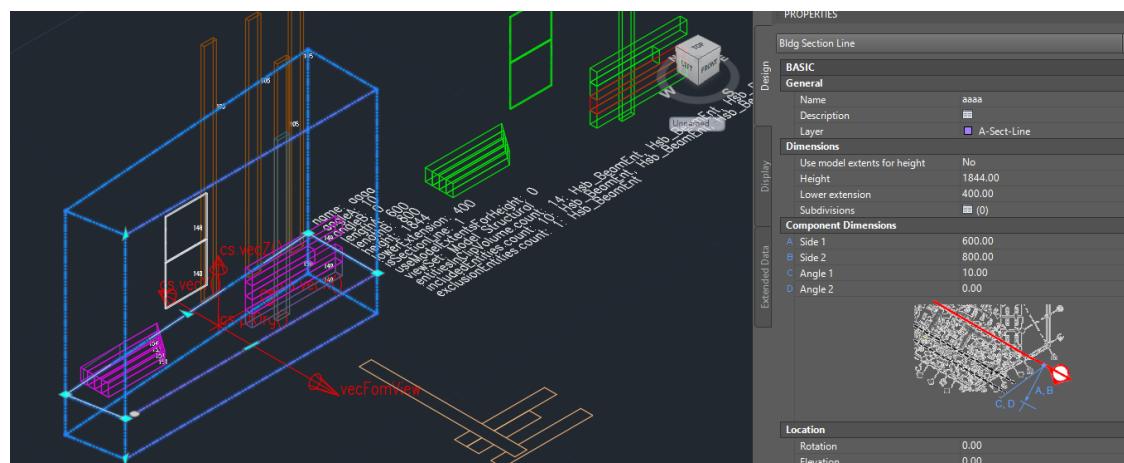
---

**CoordSys** coordSys() const;

Returns the coordsys of which the vecZ is the normal to the plane of the vertices, and ptOrg is a point in that plane.

---

[Example O-type, with insert done inside script:



```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    ClipVolume cv = getClipVolume();
    _Entity.append(cv);

    return;
}

PropInt nColor(0, 1, T("|color index|"));

```

```

PropString pDimStyle(0,_DimStyles ,T("|Dim style|"));
PropDouble pTextHeight(0,U(20),T("|Text height|"));
PropDouble pOffsetDistance(1,U(3000),T("|Offset distance|"));

if (_Entity.length() == 0 || !((ClipVolume)_Entity[0]).bIsValid())
{
    eraseInstance();
    return;
}
ClipVolume clipvol = (ClipVolume)_Entity[0];

String strChangeName = T("|Change name|");
addRecalcTrigger(_kContext, strChangeName );
if (_bOnRecalc && _kExecuteKey==strChangeName)
{
    String strNameNew = getString(T("|Enter new name|"));
    clipvol.setName(strNameNew);
}

String strChangeVertices = T("|Change vertices|");
addRecalcTrigger(_kContext, strChangeVertices );
if (_bOnRecalc && _kExecuteKey==strChangeVertices)
{
    EntPLine entpline = getEntPLine();
    Point3d pnts[] = entpline.getPLine().vertexPoints(FALSE);
    clipvol.setVertices(pnts);
}

String strCreateNew = T("|Create new one|");
addRecalcTrigger(_kContext, strCreateNew );
if (_bOnRecalc && _kExecuteKey==strCreateNew)
{
    Point3d pnts[] = clipvol.vertices();
    CoordSys cs = clipvol.coordSys();

    Vector3d vecMove(pOffsetDistance, 0, 0);
    for (int i = 0; i < pnts.length(); i++) {
        pnts[i].transformBy(vecMove);
    }

    int bIsSectionLine = clipvol.isSectionLine();

    ClipVolume cvNew;
    cvNew.dbCreate(pnts, cs.vecZ(), bIsSectionLine); // create new
one
    if (cvNew.bIsValid())
    {
        cvNew.setName(clipvol.name());
        cvNew.setAngleA(clipvol.angleA());
        cvNew.setAngleB(clipvol.angleB());
        cvNew.setLengthA(clipvol.lengthA());
        cvNew.setLengthB(clipvol.lengthB());
        cvNew.setHeight(clipvol.height());
    }
}

```

```

        cvNew.setLowerExtension(clipvol.lowerExtension());
        cvNew.setUseModelExtentsForHeight(clipvol.useModelExtentsForHeight());
    }

    //clipvol = cvNew;
    //_Entity[0] = cvNew;
}

}

String strIncludeEnts = T("|select new included entity set|");
addRecalcTrigger(_kContext, strIncludeEnts );
if (_bOnRecalc && _kExecuteKey==strIncludeEnts)
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        clipvol.setIncludedEntities(ssE.set());
    }
}

String strExcludeEnts = T("|select new exclusion entity set|");
addRecalcTrigger(_kContext, strExcludeEnts );
if (_bOnRecalc && _kExecuteKey==strExcludeEnts)
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        clipvol.setExclusionEntities(ssE.set());
    }
}

setDependencyOnEntity(clipvol);
Display dp(-1);

Point3d pnts[] = clipvol.vertices();
PLine pl;
for (int i = 0; i < pnts.length(); i++) {
    pl.addVertex(pnts[i]);
}
pl.vis(nColor);

CoordSys cs = clipvol.coordSys();
cs.vis(nColor);

Vector3d vecFomView = 2*clipvol.viewFromDir();
vecFomView.vis(cs.ptOrg(), 1);

String strLines[0];
strLines.append("name: "+clipvol.name());
strLines.append("angleA: "+clipvol.angleA());
strLines.append("angleB: "+clipvol.angleB());
strLines.append("lengthA: "+clipvol.lengthA());
strLines.append("lengthB: "+clipvol.lengthB());
strLines.append("height: "+clipvol.height());
strLines.append("lowerExtension: "+clipvol.lowerExtension());

```

```

strLines.append("isSectionLine: "+clipvol.isSectionLine());
strLines.append("useModelExtentsForHeight:
"+clipvol.useModelExtentsForHeight());
strLines.append("viewSet: "+clipvol.viewSet());

Vector3d vecMove(pOffsetDistance, 0, 0);
{
    Entity ents[] = clipvol.entitiesInClipVolume();
    String strEnts;
    for (int e = 0; e < ents.length(); e++)
    {
        if (e != 0) strEnts += ", ";
        strEnts += ents[e].typeName();
    }
    strLines.append("entitiesInClipVolume.count: " + ents.length() +
": " + strEnts);
}
{
    dp.color(3);
    Entity ents[] = clipvol.includedEntities();
    String strEnts;
    for (int e = 0; e < ents.length(); e++)
    {
        dp.color(3);
        if (e != 0) strEnts += ", ";
        strEnts += ents[e].typeName();
        Body bd = ents[e].realBody();
        bd.transformBy(vecMove);
        dp.draw(bd);
    }
    strLines.append("includedEntities.count: " + ents.length() + ":" +
+ strEnts);
}
{
    dp.color(1);
    Entity ents[] = clipvol.exclusionEntities();
    String strEnts;
    for (int e = 0; e < ents.length(); e++)
    {
        if (e != 0) strEnts += ", ";
        strEnts += ents[e].typeName();
        Body bd = ents[e].realBody();
        bd.transformBy(vecMove);
        dp.draw(bd);
    }
    strLines.append("exclusionEntities.count: " + ents.length() + ":" +
+ strEnts);
}

// display the lines
dp.color(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);

```

```

for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

if (_bOnDebug)
{
    dp.color(144);
    Body bd2 = clipvol.clippingBody();
    dp.draw(bd2);

    Body bd3 = clipvol.combinedClippedBodyOfEntities();
    bd3.vis(145);
}

```

*—end example]*

## 7.17 CollectionEntity

The term CollectionEntity refers to an instance of a entity that refers to a [CollectionDefinition](#).

An entity can be casted into a CollectionEntity. However when the casting is not allowed, the resulting CollectionEntity will become invalid. This can be checked with the `blsValid()` function of Entity.

Because CollectionEntity is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getCollectionEntity()` function can be used to query the user to select a CollectionEntity instance.

```

CollectionEntity getCollectionEntity();
CollectionEntity getCollectionEntity(String strPrompt);

```

---

```

class CollectionEntity : ToolEnt { // see ToolEnt for base functions (added hsbCAD14.0.73)

    CoordSys coordSys() const;

    String definition() const;
    void setDefinition(String strVal); // see CollectionDefinition for more information

    CollectionDefinition definitionObject() const; // (added hsbCAD18.1.24)
    void setDefinitionObject(CollectionDefinition val); // (added hsbCAD18.1.24)

```

---

```

PlaneProfile shadowProfile(Plane plane, double blowupShrinkDistance) const; // (added
V25.1.58)
PlaneProfile shadowProfile(Plane plane, double blowupShrinkDistance, Vector vecDir, String
strDisplaySetName) const; // (added V25.1.58)

void dbCreate(CoordSys csEcs, String strDefinition); // (added hsbCAD20.0.68 and
hsbCAD19.1.99)
};

```

---

**CoordSys** coordSys() const;

Build the coordinate system of the CollectionEntity. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

---

[Example O-type, with insert done inside script:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    CollectionEntity ce = getCollectionEntity();
    _Entity.append(ce);

    return;
}

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_Entity.length() == 0) return;
CollectionEntity ce = (CollectionEntity) _Entity[0];
if (!ce.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

CoordSys cs = ce.coordSys();
cs.vis();

CollectionDefinition cd = ce.definition();
Beam arBm[] = cd.beam(); // find beams of collection definition

```

---

```

    for (int e=0; e<arBm.length(); e++) {
        Beam bm = arBm[e];
        Body bd = bm.realBody();
        bd.transformBy(cs); // transform to entity coordsys
        bd.transformBy(U(500)*cs.vecX()); // additional move out
        bd.vis(1);
    }

    String strLines[0];
    strLines.append("CollectionEntity");
    strLines.append("definition: "+ce.definition());

    // display the lines
    Display dp(-1);
    dp.dimStyle(pDimStyle);
    dp.textHeight(pTextHeight);
    for (int l=0; l<strLines.length(); l++) {
        Vector3d vecO = -l*1.2*pTextHeight*_YU;
        dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
    }
}

```

*—end example*

## 7.18 CncCurveEnt

The term CncCurveEnt refers to an instance of a cnc curve used in the cnc output of [Sip](#) panels using child panels and master panels. The CncCurveEnt refers to a [CncCurveStyle](#).

An entity can be casted into a CncCurveEnt. However when the casting is not allowed, the resulting CncCurveEnt will become invalid. This can be checked with the `blsValid()` function of Entity.

Because CncCurveEnt is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getCncCurveEnt()` function can be used to query the user to select a CncCurveEnt instance.

```

CncCurveEnt getCncCurveEnt();
CncCurveEnt getCncCurveEnt(String strPrompt);

```

To find the CncCurveEnt entities that are attached to one [MasterPanel](#), use the method `myCncCurveEnts`.

---

```

class CncCurveEnt : Entity { // see Entity for base functions (added hsbCAD17.0.44)

    String style() const;
    void setStyle(String strVal); // see CncCurveStyle for more information

    void dbCreate(PLine pline, int nSideRight, String strStyle, MasterPanel mp);
}

```

---

---

```
};
```

---

[Example O-type, with insert done inside script:

```
Unit(1, "mm");

// get the CncCurveStyle
String strAllStyles[] = CncCurveStyle().getAllEntryNames(); // list
of all available CncCurveStyles
PropString pStyle(0, strAllStyles, "CncCurveEnt style"); // make
property

if (_bOnInsert) {

    _Pt0 = getPoint();

    // get the pline
    EntPLine plineEnt = getEntPLine();
    _Entity.append(plineEnt);

    // get the masterpanel
    MasterPanel mp = getMasterPanel();
    _Entity.append(mp);

    showDialog(); // allow the user to change the style
}

if (_Entity.length()<2) {
    eraseInstance();
    return;
}

String strCreateEntity = T("Create CncCurveEnt");
addRecalcTrigger(_kContext, strCreateEntity );
if (_bOnInsert || (_bOnRecalc && _kExecuteKey==strCreateEntity ))
{
    EntPLine plineEnt = (EntPLine)_Entity[0];
    MasterPanel mp = (MasterPanel)_Entity[1];

    if (!plineEnt.bIsValid() || !mp.bIsValid()) {
        eraseInstance();
        return;
    }

    PLine pline = plineEnt.getPLine();

    // create the CncCurveEnt
```

---

```

CncCurveEnt cce;
cce.dbCreate(pline, _kRight, pStyle, mp);

if (cce.bIsValid()) {
    reportMessage("\nNew created CncCurveEnt is valid.");
}
else {
    reportMessage("\nNew created CncCurveEnt is NOT valid.");
}
}

—end example]

```

*[Example O-type, with insert done inside script that illustrates the erase of all CncCurveEnt attached to one MasterPanel]*

```

if (_bOnInsert) {

    MasterPanel mp = getMasterPanel();
    CncCurveEnt ents[] = mp.myCncCurveEnts();
    reportMessage("\nNumber of initial cnc curves: "+ents.length());

    for(int c=0; c<ents.length(); c++) {
        CncCurveEnt cce = ents[c];
        cce.dbErase();
    }

    CncCurveEnt entsEmpty[] = mp.myCncCurveEnts();
    reportMessage("\nNumber of remaining cnc curves:
"+entsEmpty.length());

    eraseInstance();
    return;
}

```

—end example]

## 7.19 CurvedDescription

The term CurvedDescription refers to an entity that describes the [CurvedStyle](#).

An entity can be casted into a CurvedDescription. However when the casting is not allowed, the resulting CurvedDescription will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because CurvedDescription is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getCurvedDescription()` function can be used to query the user to select a CurvedDescription instance.

---

```
CurvedDescription getCurvedDescription();
CurvedDescription getCurvedDescription(String strPrompt);
```

---

```
class CurvedDescription : Entity { // see Entity for base functions (added hsbCAD 18.2.0)

    String style() const;
    void setStyle(String strVal); // see CurvedStyle for more information

    CoordSys coordSys() const;
    double dbCreate(CoordSys cs);

    PressEnt[] presses() const; // Return the list of PressEnt attached to this entity

    void dbCreate(String strStyle, CoordSys cs);
};
```

---

[Example O-type, with insert done inside script:

```
Unit(1, "mm");

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_bOnInsert) {
    _Pt0 = getPoint();
    // get the CurvedDescription
    CurvedDescription mp = getCurvedDescription();
    _Entity.append(mp);
}

// check entity link
CurvedDescription ent;
if (_Entity.length()>0) ent = (CurvedDescription)_Entity[0];
if (!ent.bIsValid()) {
    eraseInstance();
    return;
}

// reportNotice("\nent is fine CurvedDescription "+ent.style());

CoordSys css =ent.coordSys();
```

---

```

csS.vis();

String strCreateEntity = T("Create CurvedDescription");
addRecalcTrigger(_kContext, strCreateEntity );
if (_bOnRecalc && _kExecuteKey==strCreateEntity )
{
    Point3d ptN = getPoint(); // select point
    CoordSys csN(ptN, csS.vecX(), csS.vecY(), csS.vecZ());

    // create the CurvedDescription
    CurvedDescription cce;
    cce.dbCreate(ent.style(), csN);

    if (cce.bIsValid()) {
        reportMessage("\nNew created CurvedDescription is valid.");
    }
    else {
        reportMessage("\nNew created CurvedDescription is NOT valid.");
    }
}

CurvedStyle myCurvedStyle = CurvedStyle(ent.style());

PLine plBase = myCurvedStyle.baseCurve();
plBase.transformBy(csS);
plBase.vis(2);

String strLines[0];
strLines.append("CurvedDescription");

PressEnt prsss[] = ent.presses();
strLines.append("presses count: "+prsss.length());
for (int p=0; p<prsss.length(); p++) {
    strLines.append(prsss[p].label() + " pos "+prsss[p].curveParam());
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

—end example]

```

## 7.20 EcsMarker

The term EcsMarker refers to an instance of the hsb ecs entity. This entity can be added by the command: "HSB\_COLENTE\_ADD\_ECSMARKER".

An entity can be casted into a EcsMarker. However when the casting is not allowed, the resulting EcsMarker will become invalid. This can be checked with the blsValid() function of Entity.

Because EcsMarker is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During \_bOnInsert, the getEcsMarker() function can be used to query the user to select an instance.

```
EcsMarker getEcsMarker();
EcsMarker getEcsMarker(String strPrompt);
```

---

```
class EcsMarker : Entity { // see Entity for base functions (added hsbCAD2012 build 17.1.22
    and hsbCAD2013 build 18.0.10)

    CoordSys coordSys() const;
    double dbCreate(CoordSys cs);
};
```

---

**CoordSys** coordSys() const;

Build the coordinate system of the EcsMarker. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

---

[Example O-type, with insert done inside script:

```
U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    EcsMarker ent = getEcsMarker();
    Entity.append(ent);

    return;
}
```

---

```

}

// check entity link
EcsMarker ent;
if (_Entity.length()>0) ent = (EcsMarker)_Entity[0];
if (!ent.bIsValid()) {
    eraseInstance();
    return;
}

CoordSys csS =ent.coordSys();
csS.vis();

String strCreateEntity = T("Create new entity");
addRecalcTrigger(_kContext, strCreateEntity );
if (_bOnRecalc && _kExecuteKey==strCreateEntity ) {
    Point3d ptN = getPoint(); // select point
    CoordSys csN(ptN, csS.vecX(), csS.vecY(), csS.vecZ());
    EcsMarker ecsNew;
    ecsNew.dbCreate(csN);
    _Entity[0] = ecsNew;
}

```

*—end example]*

## 7.21 EntPLine

The term EntPLine refers to an instance of a AcDbPolyline in the drawing.

An entity can be casted into a EntPLine. However when the casting is not allowed, the resulting EntPLine will become invalid. This can be checked with the bIsValid() function of Entity.

Because EntPLine is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During [\\_bOnInsert](#), the [getEntPLine\(\)](#) function can be used to query the user to select a polyline instance.

```

EntPLine getEntPLine();
EntPLine getEntPLine(String strPrompt);

```

---

```

class EntPLine : Entity { // see Entity for base functions

    // PLine getPLine() const; // now moved to Entity base class (since hsbCAD13.2.132)
    void setPLine(PLine pl); // see PLine

    void dbCreate(PLine pl);
};

```

---

---

**PLine** getPLine() const;

Returns the PLine geometry from the EntPLine entity.

**void** setPLine(**PLine** pl);

With the setPLine routine, the geometry of an existing EntPLine can be changed.

**void** dbCreate(**PLine** pl);

Create a new EntPLine in the Autocad database. The given PLine will be used to construct the new EntPLine. The current instance will be set to the new created one.

---

*[Example O-type, with insert done inside script:*

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    EntPLine pl = getEntPLine();
    _Entity.append(pl);

    return;
}

for (int i=0; i<_Entity.length(); i++) {
    EntPLine epl = (EntPLine)_Entity[i];
    PLine pl = epl.getPLine();

    Point3d pnts[] = pl.vertexPoints(TRUE);
    if (pnts.length()>0) {
        Point3d ptN = pnts[pnts.length()-1] + U(1000)*_XW;
        pl.addVertex(ptN, 0);
        pl.vis(1);

        epl.setPLine(pl);
    }
}

```

*--end example]*

---

## 7.22 EntCircle

The term EntCircle refers to an instance of a AcDbCircle in the drawing.

An entity can be casted into a EntCircle. However when the casting is not allowed, the resulting EntCircle will become invalid. This can be checked with the blsValid() function of Entity.

Because EntCircle is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getEntCircle()` function can be used to query the user to select a circle instance.

```
EntCircle getEntCircle();
EntCircle getEntCircle(String strPrompt);
```

---

```
class EntCircle : Entity // Added since 24.1.49. See Entity for base functions.
{
    Point3d ptCen() const; // center point of circle
    void setPtCen(Point3d pt);
    Vector3d normal() const; // normal of plane of circle
    void setNormal(Vector3d vec);
    double radius() const; // radius of circle
    void setRadius(double rad);

    void dbCreate(Point3d ptCen, Vector3d vecNormal, double dRadius);
};
```

---

[Example O-type, with insert done inside script:

```
U(1, "mm");
if (_bOnInsert)
{
    _Pt0 = getPoint(); // select point
    EntCircle ent = getEntCircle();
    Entity.append(ent);

    return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}
EntCircle ent = (EntCircle)_Entity[0];
```

---

```

String strChangeCenter = T("|Change center point|");
addRecalcTrigger(_kContext, strChangeCenter );
if (_bOnRecalc && _kExecuteKey==strChangeCenter)
{
    Point3d ptNew = getPoint(T("|Select new center location|"));
    ent.setPtCen(ptNew);
}

String strAlignNormal = T("|Align normal|");
addRecalcTrigger(_kContext, strAlignNormal );
if (_bOnRecalc && _kExecuteKey==strAlignNormal)
{
    EntCircle entOther = getEntCircle(T("|Select other circle to get
normal from|"));
    Vector3d vecNew = entOther.normal();
    ent.setNormal(vecNew);
}

String strIncreaseRadius = T("|Increase radius|");
addRecalcTrigger(_kContext, strIncreaseRadius );
if (_bOnRecalc && _kExecuteKey==strIncreaseRadius)
{
    ent.setRadius(ent.radius() * 1.1);
}

String strCopyCircle = T("|Copy circle|");
addRecalcTrigger(_kContext, strCopyCircle );
if (_bOnRecalc && _kExecuteKey==strCopyCircle)
{
    Point3d ptNew = getPoint(T("|Select new center location|"));
    ent.dbCreate(ptNew, ent.normal(), ent.radius());
    _Entity[0] = ent;
}

PLine pl = ent.getPLine();
pl.vis(1);
Point3d ptC = ent.ptCen();
ptC.vis(1);
Vector3d vecN = ent.normal();
vecN.vis(ptC,1);

—end example]

```

## 7.23 ERoofPlane

The type ERoofPlane refers to an instance of the hsbCad roof plane entity type. The ERoofPlane is derived from Entity, so it is an entity in the autocad database/drawing.

An entity can be casted into an ERoofPlane. However when the casting is not allowed, the resulting ERoofPlane will become invalid. This can be checked with the blsValid() function of Entity.

Because ERoofPlane is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity.

During `_bOnInsert`, the `getERoofPlane()` function can be used to query the user to select a ERoofPlane instance.

```
ERoofPlane getERoofPlane();
ERoofPlane getERoofPlane(String strPrompt);
```

Beware !

An hsbERoofPlane entity in the autocad drawing can be/do many things.

- it is a part of a roof, generated by the roof generation command. As such, it has a 3d planar closed shape.
- it bounds the wall element during the wall element generation. For that purpose it has a roof code.
- it might control a number of timbers (rafters, plates,...). How the plate cuts the rafter, what the gap is at the top of the roofplane, ... For this purpose, the roofplane is actually a dynamic tool, updating cuts, beamcuts,... in the beams itself.
- it might be an element on its own. That is in the panel generation, the roofplane can have data attached that it is actually an Element. If you want to look at the ERoofPlane as an Element, within TSL, you will need to cast it to an Element. The Element will be valid if the hsbERoofPlane carries element generation data.

The `ERoofPlane::planeType()` returns one of the following predefined variables of type **int**:

`_kRPTRoof, _kRPTFloor, _kRPTCeiling`

---

```
class ERoofPlane : ToolEnt { // see ToolEnt for base functions

    PLine plEnvelope() const; // see PLine
    PLine plWallBoundary() const; // see PLine

    CoordSys coordSys() const; // see CoordSys

    String code() const; // returns the wall envelope code
    void setCode(String str); // sets the wall envelope code

    String roofNumber() const; // (added hsbCAD2010 build 15.3.8)
    void setRoofNumber(String str); // (added hsbCAD2010 build 15.3.8)

    void dbCreate(PLine pline); // create a new database resident ERoofPlane from PLine.

    Point3d[] vertices() const; // (added hsbCAD21 build 21.4.12)
    void setVertices(Point3d[] arVertices); // (added hsbCAD21 build 21.4.12)

    int planeType() const; // (added hsbCAD2013 build 18.0.6)
    void setPlaneType(int nType); // (added hsbCAD2013 build 18.0.6)

    int frontCutPerpToRoof() const; // returns TRUE or FALSE (added build 21.4.75 and 22.0.116)
    void setFrontCutPerpToRoof(int bVal); // (added build 21.4.75 and 22.0.116)

    static void runCorrector(); // (added v20.0.50 and v19.1.80)
```

```

static void runCorrector(<Entity>[] arRoofplanesToConsider); // (added v20.0.50 and v19.1.80)

ERoofPlane[] findContainingRoofplanes(); // (added v22.0.32)
ERoofPlane[] findContainingRoofplanes(ERoofPlane[] arRoofplanesToConsider); // (added
v22.0.32)

ERoofPlane findParentRoofplane(); // (added v22.0.39)
ERoofPlane findParentRoofplane(ERoofPlane[] arRoofplanesToConsider); // (added v22.0.39)

EdgeTileData[] edgeTileData(); // (added v22.0.39) get the EdgeTileData
EdgeTileData[] edgeTileTopology(); // (added v22.0.39) get the EdgeTileData

ERoofPlane[] getAllRoofPlanes(); // only ERoofPlanes from current model space (added
v22.0.39)
ERoofPlane[] getAllRoofPlanes(int bAlsoLookInBlockRefs); // if TRUE, all blockrefs including
xrefs are investigated, resulting in a possible performance issue. (added
v22.0.39)
};

```

---

**PLine** plEnvelope() const;

The PLine that describes the shape of the roofplane.

**PLine** plWallBoundary() const;

The PLine that describes the boundary to where the walls are stretched.

**CoordSys** coordSys() const;

Build the coordinate system of the ERoofPlane.

```

int planeType() const;
void setPlaneType(int nType); // (added hsbCAD2013 build 18.0.6)

```

The planeType is one of the following predefines: [\\_KRPTRoof](#), [\\_KRPTFloor](#), [\\_KRPTCeiling](#)

```

static void runCorrector() const; // Automatically select all ERoofPlanes in the drawing (added
v20.0.50 and v19.1.80)
static void runCorrector(Entity[] arRoofplanesToConsider); // (added v20.0.50 and v19.1.80)

```

Correct the vertex points of the given roofplanes such that they match up with each other.  
 Roofplanes generated with the build in hsbCAD functionality do not need this, but if the  
 roofplanes come from another source, eg ModelX, small deviations in the vertices might result in  
 slightly overlapping, or slightly decoupled vertices.

```
ERoofPlane[] findContainingRoofplanes(); // (added v22.0.32 and v21.3.13)
ERoofPlane[] findContainingRoofplanes(ERoofPlane[] arRoofplanesToConsider); // (added
v22.0.32 and v21.3.13)
```

Find all or filter the given set of ERoofPlanes to the ones that lay completely inside this ERoofPlane. From the given set arRoofplanesToConsider, or from all ERoofPlanes in this database (if no ERoofPlanes are given as argument), return the list of ERoofPlanes that:

- share the same plane with this ERoofPlane;
- all vertices lay on or inside this ERoofPlane

Example, see below.

```
ERoofPlane findParentRoofplane(); // (added v22.0.39)
ERoofPlane findParentRoofplane(ERoofPlane[] arRoofplanesToConsider); // (added v22.0.39)
```

Find one from all or a given set of ERoofPlanes that surrounds completely this ERoofPlane. Return the first ERoofPlanes that:

- share the same plane with this ERoofPlane;
  - all vertices lay on or outside this ERoofPlane
- 

[Example O-type TSL:

```
Unit(1, "mm");

if (_bOnInsert) {

    ERoofPlane roofplane = getERoofPlane();
    _Entity.append(roofplane);
    return;
}

if (_Entity.length()==0) return;
ERoofPlane roofpl = (ERoofPlane)_Entity[0];
if (!roofpl.bIsValid()) return;

PLine pline = roofpl.plEnvelope();
pline.vis(1);

PLine plWall= roofpl.plWallBoundary();
plWall.vis(3);

CoordSys cs = roofpl.coordSys();
cs.vis(1);
_Pt0 = cs.ptOrg();
```

—end example]

[Example O-type TSL:

```

Unit(1, "mm");
if (_bOnInsert)
{
    _Entity.append(getERoofPlane(T("|Select main roofplane|"),
TRUE));

PrEntity ssE(T("|Select a set of roofplanes|"), ERoofPlane());
ssE.allowNested(TRUE);
if (ssE.go()) {
    _Entity.append(ssE.set());
}

_Pt0 = getPoint();
return;
}

ERoofPlane erMain;
ERoofPlane arOther[0];
for(int e=0; e<_Entity.length(); e++)
{
    ERoofPlane rp = (ERoofPlane)_Entity[e];
    if (!rp.bIsValid())
        continue;
    if (e == 0)
        erMain = rp;
    else
        arOther.append(rp);
}

String strCorrect = T("|Correct roofplanes|");
addRecalcTrigger(_kContext, strCorrect );
if (_bOnRecalc && _kExecuteKey==strCorrect )
{
    if (arOther.length() == 0)
        ERoofPlane().runCorrector();
    else
    {
        Entity arEnt[0];
        for (int e = 0; e < arOther.length(); e++)
            arEnt.append(arOther[e]);
        ERoofPlane().runCorrector(arEnt);
    }
}

ERoofPlane arContaining[0];
if (arOther.length() == 0)
    arContaining = erMain.findContainingRoofplanes();
else
    arContaining = erMain.findContainingRoofplanes(arOther);

reportMessage ("\nFound " + arContaining.length() + "/" +
arOther.length());

```

```

Display dp(-1);
dp.textHeight(U(100));
if (arContaining.length() == 0)
    dp.draw(scriptName(), _Pt0, _XU, _YU, 0, 0);

for (int e = 0; e < arContaining.length(); e++)
{
    ERoofPlane rp = arContaining[e];
    PLine pl = rp.plEnvelope();
    PlaneProfile prof(pl);
    dp.draw(prof, _kDrawFilled, 50);
}

```

—end example]

### 7.23.1 EdgeTileData

EdgeTileData is a class exposing the tile info for the edges of the [ERoofPlane](#).

```

EdgeTileData tiles[] = roof.edgeTileData(); // query the roof plane edges
EdgeTileData tiles[] = roof.edgeTileTopology(); // query the roof plane edge topology

```

The [EdgeTileData::tileType\(\)](#) returns one of the following predefined variables of type [int](#):

```

_kTileEave = 0,
_kTileRidge,
_kTileRight,
_kTileLeft,
_kTileHip,

_kTileValley,
_kTileRightSloped,
_kTileLeftSloped,
_kTileRightOpening,
_kTileLeftOpening,

_kTileBottomOpening,
_kTileTopOpening,
_kTileHorizontalPlaneEdge,
_kTileOther,
_kTileEdgeTooSmall, (added to v22.0.38)

_kTileNotSet

```

---

```

class EdgeTileData { // (Do not use in versions before 22.0.38)

    EdgeTileData();

```

---

```

String tileName() const;
int tileType() const; // returns _kTileLast if not set

int tileTypePartner() const; // returns _kTileLast if not set
ERoofPlane partnerRoofplane() const;
ERoofPlane myRoofplane() const;

LineSeg segment() const; // return LineSeg
Map tileMap() const; // return Map

static String nameFromType(int type); // returns translated name of type (type should be:
_kTileEave, _kTileRidge...)
};

```

---

*[Example O-type TSL, insert done in script:*

```

Unit(1, "mm");

int arIntTypes[] =
{
    _kTileEave,
    _kTileRidge,
    _kTileRight,
    _kTileLeft,
    _kTileHip,
```

- \_kTileValley**,
- \_kTileRightSloped**,
- \_kTileLeftSloped**,
- \_kTileRightOpening**,
- \_kTileLeftOpening**,
- \_kTileBottomOpening**,
- \_kTileTopOpening**,
- \_kTileHorizontalPlaneEdge**,
- \_kTileOther**,
- \_kTileEdgeTooSmall**,
- \_kTileNotSet**
};

**String** arStrTypes[arIntTypes.length()];
**for** (**int** i=0; i<arIntTypes.length(); i++)
 arStrTypes[i] =
**EdgeTileData()**.nameFromType(arIntTypes[i]);

**PropString** pTypeToUse(0, arStrTypes, **T**("|Type to use|"));

---

```

int nTypeToUse = arIntTypes[arStrTypes.find(pTypeToUse, 0)];

if (_bOnInsert) {

    // select the Panel and insertion point
    ERoofPlane rp = getERoofPlane();
    _Entity.append(rp);
    _Pt0 = getPoint();

    return;
}

// check running conditions
ERoofPlane rp;
if (_Entity.length()!=0)
    rp = (ERoofPlane)_Entity[0];
if (!rp.bIsValid())
{
    eraseInstance();
    return;
}

```

```

EdgeTileData edges[] = rp.edgeTileTopology();

reportMessage("\n");
reportMessage("\n nTypeToUse: "+
EdgeTileData().nameFromType(nTypeToUse) );

for (int i=0; i<edges.length(); i++)
{
    EdgeTileData edge = edges[i];
    LineSeg seg = edge.segment(); seg.vis(1);

    reportMessage("\n tileName: "+edge.tileName() );
    reportMessage("\n tileType: "+edge.tileType() );
    reportMessage("\n tileTypePartner:
"+edge.tileTypePartner() );
    reportMessage("\n partnerRoofplane:
"+edge.partnerRoofplane() );
    reportMessage("\n myRoofplane: "+edge.myRoofplane() );
    reportMessage("\n tileMap: "+edge.tileMap() );
}

```

*—end example]*

## 7.24 ERoofPlaneOpening

The term ERoofPlaneOpening refers to an instance of the hsb opening entity inside the ERoofplane, typically coupled with a roof plane section.

An entity can be casted into an ERoofPlaneOpening. However when the casting is not allowed, the resulting ERoofPlaneOpening will become invalid. This can be checked with the `blsValid()` function of Entity.

Because ERoofPlaneOpening is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity.

When you link the TSL instance with an opening, you must add it to the entity array `_Entity`.

During `_bOnInsert`, the `getERoofPlaneOpening()` function can be used to query the user to select an opening instance.

```
ERoofPlaneOpening getERoofPlaneOpening();
ERoofPlaneOpening getERoofPlaneOpening(String strPrompt);
```

---

```
class ERoofPlaneOpening : Entity { // see Entity for base functions

    CoordSys coordSys() const;

    PLine plEnvelope() const;

    String name() const; // the name property
    void setName(String strVal);

    LineSeg[] lineSegments() const;

    // the realBody function from the base class is also available

};
```

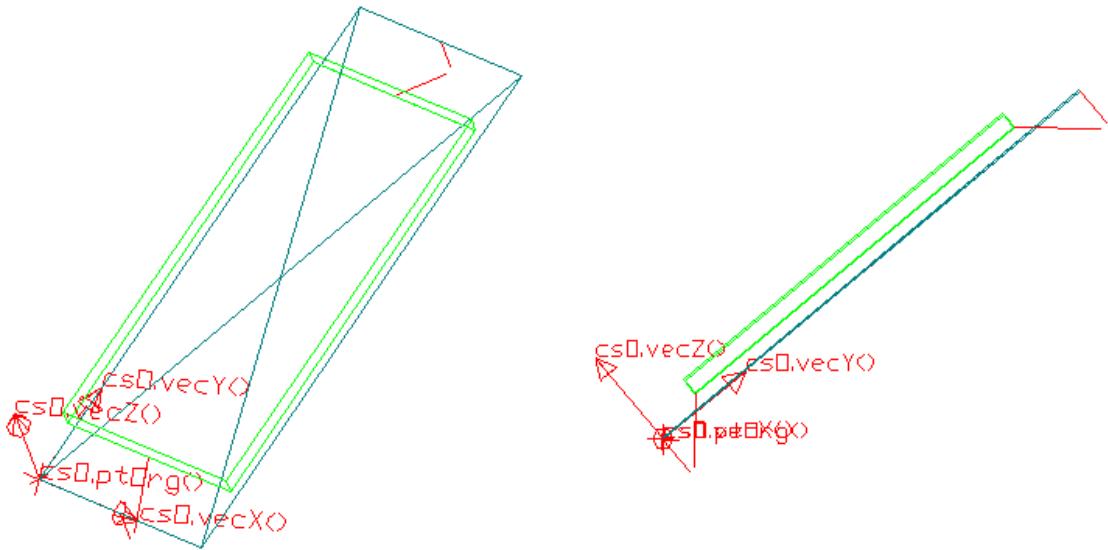
---

**LineSeg[]** lineSegments();

This function returns in 3D the linesegments that can be used in a section view of the roof. Typically there are 4 of them. The first one is perpendicular at the bottom, the next one is normally vertical located at the bottom. The third one is the horizontal one at the top, and the last one is the perpendicular at the top.

The `startPoint` of the segments is on or above the ERoofPlaneOpening. The `endPoint` is at the bottom.

[Example O-type, with insert done inside script:



```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    ERoofPlaneOpening op = getERoofPlaneOpening();
    _Entity.append(op);

    return;
}

Display dp(1); // color red

for (int i=0; i<_Entity.length(); i++) {

    ERoofPlaneOpening opr = (ERoofPlaneOpening)_Entity[i];
    if (!opr.bIsValid()) continue; // go to next i

    CoordSys cs0 = opr.coordSys();
    cs0.vis(1);

    opr.setName("myName");
    reportNotice("\n\n_ERoofPlaneOpening.name = " +opr.name() );

    PLine plop = opr.p1Envelope();
    dp.draw(plop);
}

```

```

//Vector3d vecNormal = csO.vecZ();
//vecNormal.vis(csO.ptOrg());

LineSeg segs[] = opr.lineSegments();
dp.draw(segs);

dp.color(3);
Body body = opr.realBody();
dp.draw(body);
}

```

*—end example]*

## 7.25 FastenerAssemblyEnt

The term **FastenerAssemblyEnt** refers to an instance of a entity that refers to a [FastenerAssemblyDef](#).

An entity can be casted into a **FastenerAssemblyEnt**. However when the casting is not allowed, the resulting **FastenerAssemblyEnt** will become invalid. This can be checked with the **blsValid()** function of Entity.

Because **FastenerAssemblyEnt** is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During **\_bOnInsert**, the **getFastenerAssemblyEnt()** function can be used to query the user to select a **FastenerAssemblyEnt** instance.

```

FastenerAssemblyEnt getFastenerAssemblyEnt();
FastenerAssemblyEnt getFastenerAssemblyEnt(String strPrompt);

```

```

class FastenerAssemblyEnt : ToolEnt { // see ToolEnt for base functions (added hsbCAD2012
    build 17.0.21)

    void dbCreate(String strDefinition, CoordSys csEcs);
    int anchorTo(ToolEnt toolEnt, Point3d ptLoc, double dVal); // return success

    CoordSys coordSys() const; // calculated automatically when anchored to a ToolEnt

    String definition() const;
    void setDefinition(String strVal); // see FastenerAssemblyDef for more information

    double holdingDistance() const;
}

```

```

void setHoldingDistance(double dVal); // calculated automatically when anchored to a
    ToolEnt
void setHoldingDistance(double dVal, int bRecalculateArticle); // default value of
    bRecalculateArticle is FALSE

String articleNumber() const;
void setArticleNumber(String strVal); // calculated automatically when
    lengthSelectionIsAutomatic is TRUE

int lengthSelectionIsAutomatic() const; // default TRUE
void setLengthSelectionIsAutomatic(int iVal);

String description() const; // read only value determined by FastenerAssemblyDef

ToolEnt guidelineToolEnt(); // FastenerGuideline ToolEnt carrier to which this one is
    attached (added v20.1.14 and v21.0.21.)

};

```

---

**CoordSys** coordSys() const;

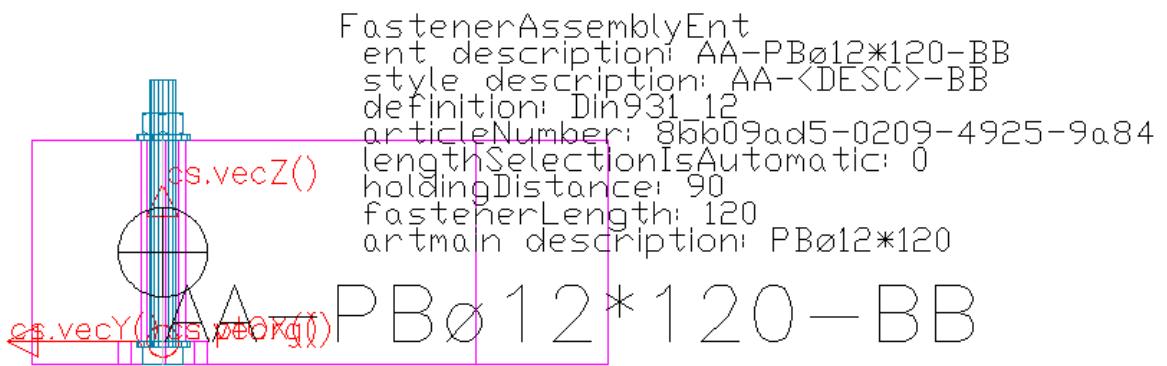
Build the coordinate system of the FastenerAssemblyEnt. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

**String** description() const;

The description is of the FastenerAssemblyEnt is the string that is displayed with the corresponding display component of the entity. It is basically the description of the style ([FastenerAssemblyDef](#) pointed to by definition). In case the description of the style is empty, then the dictionary entry name of the style is shown. In the description, the string <DESC> is substituted by the description of the list component to which the article number of this entity refers to (look for mainComponent).

---

*[Example O-type, with insert done inside script illustrating accessing the info of the FastenerAssemblyEnt as well as composition of the description:*



```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    FastenerAssemblyEnt ce = getFastenerAssemblyEnt();
    _Entity.append(ce);

    return;
}

PropString pDimStyle(1, DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_Entity.length() == 0) return;
FastenerAssemblyEnt ce = (FastenerAssemblyEnt) _Entity[0];
if (!ce.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

CoordSys cs = ce.coordSys();
cs.vis(1);

FastenerAssemblyDef cd(ce.definition());
FastenerSimpleComponent compMain =
cd.mainComponent(ce.articleNumber());
FastenerArticleData artMain = compMain.articleData();

String strLines[0];
strLines.append("FastenerAssemblyEnt");
strLines.append(" ent description: "+ce.description());
strLines.append(" style description: "+cd.description());
strLines.append(" definition: "+ce.definition());
strLines.append(" articleNumber: "+artMain.articleNumber());
strLines.append(" lengthSelectionIsAutomatic:
"+ce.lengthSelectionIsAutomatic());
strLines.append(" holdingDistance: "+ce.holdingDistance());
strLines.append(" fastenerLength: "+artMain.fastenerLength());
strLines.append(" artmain description: "+artMain.description());

```

```
// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}
```

—end example]

[Example O-type, illustrating FastenerAssemblyEnt dbCreate:

```
U(1, "mm");
if (_bOnInsert) {

    String strAllStyles[] =
FastenerAssemblyDef().getAllEntryNames(); // list of all available
FastenerAssemblyDefs
    PropString pStyle(0, strAllStyles, T("|Fastener assembly
def|"));
    // make property
    showDialog(); // allow the user to change the style

    _Pt0 = getPoint(); // select insertion point

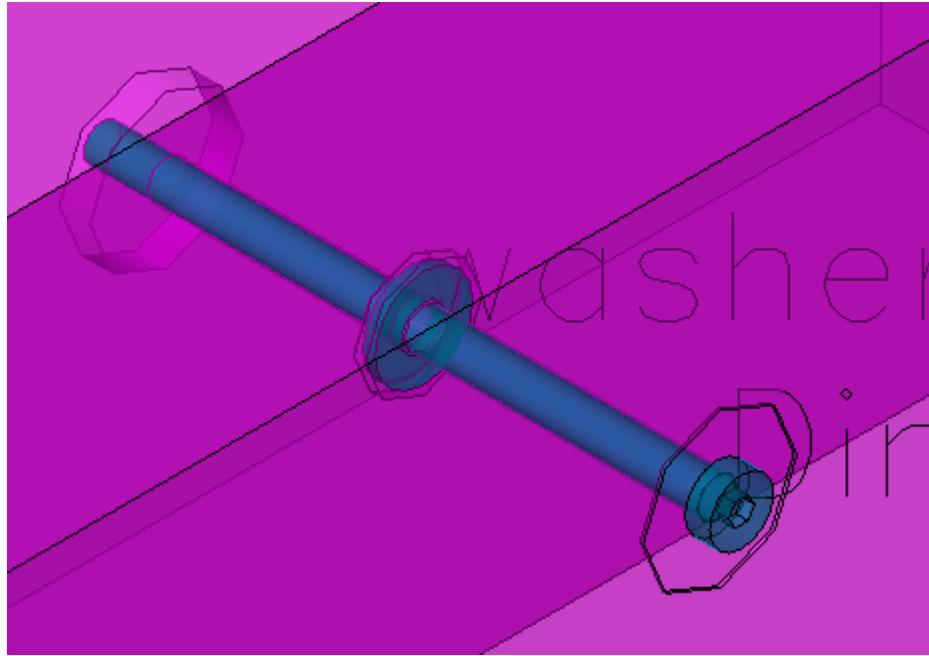
    // compose ecs
    CoordSys csEcs(_Pt0, _XU, _YU, _ZU);

    // create a new FA in the acad database
    FastenerAssemblyEnt fae;
    fae.dbCreate(pStyle, csEcs);

    eraseInstance(); // do not insert this Tsl in the database
    return;
}
```

—end example]

[Example O-type, illustrating FastenerAssemblyEnt that is created at an intersection of 2 beams along the axis line of an existing fastener:



```

U(1, "mm");

//PropDouble pDrillDepth(0,U(5),T("|Drill depth|"));
//PropDouble pDrillDiam(1,U(40),T("|Drill diameter|"));

String strAllStyles[] = FastenerAssemblyDef().getAllEntryNames(); // 
list of all available FastenerAssemblyDefs
PropString pStyleFAToInsert(0, strAllStyles, T("|Fastener assembly to
insert|")); // make property

if (_bOnInsert) {

    showDialogOnce(); // allow the user to change the properties

    FastenerAssemblyEnt fae = getFastenerAssemblyEnt();
    _Entity.append(fae);

    // go and find out if there are only 2 beams involved for this
fastener
    CoordSys cs =fae.coordSys();
    double dZ = fae.holdingDistance();
    LineSeg seg(cs.ptOrg(), cs.ptOrg()+dZ*cs.vecZ());

    // find all beams that this fastener touches
    Group grAll;
    Entity ents[] = grAll.collectEntities(TRUE,Beam(), _kModelSpace);
    Beam bms[0];
    for (int e=0; e<ents.length(); e++)
    {
        Beam bm = (Beam)ents[e];
        if (bm.bIsValid())

```

```

        bms.append(bm);
    }
Beam bmsSeg[] =
Beam() .filterBeamsCapsuleIntersect(bms, seg, U(0.1));
Beam bmsPerp[] = cs.vecZ() .filterBeamsPerpendicularSort(bmsSeg);

if (bmsPerp.length()==2) { // precisely 2
    Beam.append(bmsPerp[0]);
    Beam.append(bmsPerp[1]);
}
else {
    // ask user to select 2 beams
    Beam.append(getBeam(T("|Multiple beams are found. Please
select the first beams to connect.|")));
    Beam.append(getBeam(T("|Please select the second beams to
connect.|")));
}

return;
}

// check running conditions
if (_Entity.length()==0) {
    eraseInstance(); // just erase from DB
    return;
}
FastenerAssemblyEnt fae = (FastenerAssemblyEnt)_Entity[0];
if (!fae.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}
CoordSys cs = fae.coordSys();
Vector3d vecZ = cs.vecZ();

Beam = cs.vecZ() .filterBeamsPerpendicularSort(_Beam);
if (_Beam.length()<2) {
    eraseInstance(); // just erase from DB
    return;
}

setDependencyOnEntity(fae); // if fae changes, also change this
instance

// get data from selected fastener
FastenerAssemblyDef fad(pStyleFATOInsert);
FastenerSimpleComponent compMain =
fad.mainComponent(fae.articleNumber());
FastenerComponentData fcd = compMain.componentData();

double dDrillDepth; // = pDrillDepth; // from property
double dDrillDiam; // = pDrillDiam;
dDrillDepth = fcd.stackThickness();

```

```

dDrillDiam = U(6) + fcd.sinkDiameter();

// use the location of the fae to place the drills
Vector3d vecZ0 = _Beam[0].vecD(vecZ);
double dH0 = _Beam[0].dD(vecZ);
Plane pl0(_Beam[0].ptCen() + 0.5*dH0*vecZ0, vecZ0);

Vector3d vecZ1 = _Beam[1].vecD(vecZ);
double dH1 = _Beam[1].dD(vecZ);

Point3d pts0 = Line(cs.ptOrg(), vecZ).intersect(pl0, 0);
pts0.vis();
_Pt0 = pts0;
double dRadius = 0.5*dDrillDiam;

// draw something
Display dp(-1);
PLine pCir; pCir.createCircle(pts0, vecZ, 0.8*dRadius);
dp.draw(pCir);
PLine pCir2; pCir2.createCircle(pts0, vecZ, 0.76*dRadius);
dp.draw(pCir2);

// add tools to beam0 and beam1
Point3d pt02 = pts0-0.9*dDrillDepth*vecZ;
Drill dr0(pts0, pt02, dRadius);
_Beam[0].addTool(dr0);

Point3d pt12 = pts0+0.1*dDrillDepth*vecZ;
Drill dr1(pts0, pt12, dRadius);
_Beam[1].addTool(dr1);

// add a guideline for the drills added such that a new fastner
// assembly can be anchored to it.
FastenerGuideline fg(pt02, pt12, dRadius);
_ThisInst.resetFastenerGuidelines();
_ThisInst.addFastenerGuideline(fg);

// right after insert, on db created, a fastener assembly is added to
// the database as well.
String strChangeEntity = T("|Recreate fastener ring|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnDbCreated || (_bOnRecalc && _kExecuteKey==strChangeEntity))
{
    // compose ecs
    CoordSys csEcs(pt12, cs.vecX(), cs.vecY(), cs.vecZ());

    // create a new FA in the acad database
    FastenerAssemblyEnt faeNew;
    faeNew.dbCreate(pStyleFATOInsert, csEcs);

    // also anchor the new fastener to me.
    faeNew.anchorTo(_ThisInst, pt12, U(1));
}

```

*—end example]*

## 7.26 Grid

The Grid entity refers to an instance of an Hsb\_Grid in the drawing.

An entity can be casted into a Grid. However when the casting is not allowed, the resulting Grid will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because Grid is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getGrid()` function can be used to query the user to select a grid instance.

```
Grid getGrid();
Grid getGrid(String strPrompt);
```

It is also possible to find the grid that belongs to a group, see [Group](#).

```
Grid Group::grid();
```

This member function from Group returns the grid that would be considered active for the group. If the group contains a Grid, it is taken. If the group does not contain a Grid, the parent group is investigated. If the parent group does contain a Grid, that one is returned. This is repeated until a Grid is found or until no Grid is found that belongs to the Groups at hand. If that is the case, the complete database is searched for a Grid that does not belong to a Group. The first Grid found is returned.

It is also possible to find the grid that an entity is assumed to belong to, see [Entity](#).

```
Grid Entity::grid();
```

This member function from Entity returns the grid that has the best match of group names with the groups of the entity.

---

```
class Grid : Entity { // see Entity for base functions

    CoordSys coordSys() const;
    double dSizeX() const; // dimension in vecX direction of coordSys
    double dSizeY() const; // dimension in vecY direction of coordSys

    String coordinate(Point3d pt, Vector3d vecDir) const;
    String coordinate(Point3d pt, Vector3d vecDir, String strOfflineChar, double dTol) const;

    Point3d[] intersectPoints(LineSeg seg, Vector3d vecDir) const;
```

---

```

Point3d[] intersectPoints(LineSeg seg, Vector3d vecDir, double dTol) const; // tolerance in
    segment direction
Point3d[] intersectPoints(Line seg, Vector3d vecDir) const;

int hasClosestPointTo(Point3d pt, Vector3d vecDir, Point3d& ptClosest) const; // return TRUE
    if closest is found, and set in ptClosest.
int hasClosestPointTo(Point3d pt, Vector3d vecDir, Point3d& ptClosest, Point3d&
    ptOneButClosest) const; // return TRUE if closest is found, and set in ptClosest.

int blsOn(Point3d pt, Vector3d vecDir, double dTol) const; // return TRUE if closest point is
    closer to pt than dTol
int blsInside(Point3d pt) const; // return TRUE if point is inside grid rectangle

// added hsbCAD2011, build 16.0.40
String name() const;
void setName(String str);

CoordSys flipAxisToOptimalGridOrientation(CoordSys csBeam) const; // added to hsbCAD2011
    build 16.2.9

};

```

---

**Point3d[]** intersectPoints(**LineSeg** seg, **Vector3d** vecDir) const;  
**Point3d[]** intersectPoints(**Line** seg, **Vector3d** vecDir) const;

Returns an ordered list of points. The points lie on the Line or LineSeg. They are the intersections of the line with the planes that run through the gridlines, and along the normal vecZ of the grid. The gridlines that are most perpendicular with vecDir are considered. If vecDir does not have a component in the XY plane of the grid, all intersection points are considered. The points are ordered in the vecDir direction. If the vecDir component in the XY plane is null, then the direction of the segment or line is used.

**String** coordinate(**Point3d** pl, **Vector3d** vecDir) const;  
**String** coordinate(**Point3d** pl, **Vector3d** vecDir, **String** strOfflineChar, **double** dTol) const;

Returns a string that represents the coordinate in the grid. The vecDir is used to determine the coordinate, either the X or the Y coordinate. The coordinate of the gridlines that are most perpendicular with vecDir are considered. If vecDir does not have a component in the XY plane of the grid, the complete XY coordinate is returned.

The call with 4 arguments returns the Japanese grid coordinate, with a offline indicator prefix for the coordinate if the point lies more than dTol from the gridline.

```

int hasClosestPointTo(Point3d pt, Vector3d vecDir, Point3d& ptClosest) const; // return TRUE if
    closest is found, and set in ptClosest.
int hasClosestPointTo(Point3d pt, Vector3d vecDir, Point3d& ptClosest, Point3d&
    ptOneButClosest) const; // return TRUE if closest is found, and set in ptClosest.

```

---

Returns TRUE if the closest point is found. In this case the closest point is set to ptClosest. The candidates of closest points are calculated by calling the intersectPoints with a line through pt, and with direction vecDir. So make sure, the vecDir has a component in the XY plane of the grid. If not, FALSE will be returned.

**CoordSys** flipAxisToOptimalGridOrientation(**CoordSys** csBeam) const;

This method is added to hsbCAD2011 build 16.2.9. The coordinate system returned is the coordinate system passed in as argument, with the vecX, vecY and vecZ possibly flipped. The X axis for none vertical beams is flipped to align with most positive grid x or grid y direction. Z and Y of beam are flipped to most positive grid z.

If X axis of beam is vertical, then it is flipped to positive grid z, and x or y are aligned to negative grid axis.

[Example E-type to visualize

```
Unit(1, "mm");

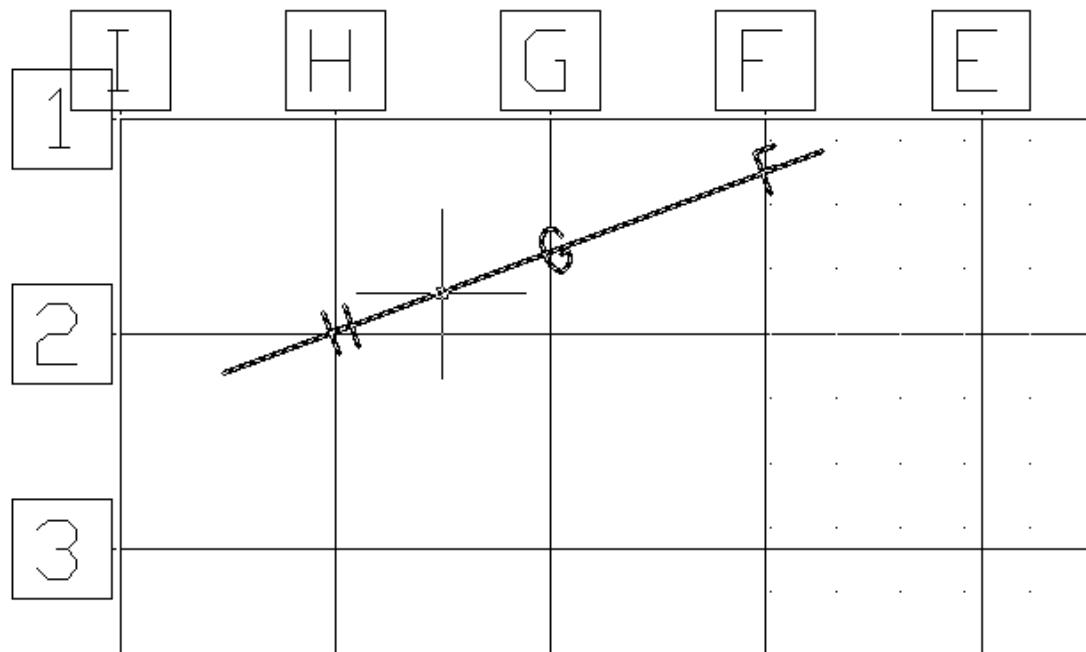
CoordSys csB = _Beam0.coordSys();
csB.vis(1);

Grid grd = _Beam0.grid();
CoordSys csBO = grd.flipAxisToOptimalGridOrientation(csB);
csBO.transformBy(U(200)*csB.vecX());
csBO.vis(3);
```

—end example]

---

[Example O-type, with insert done inside script, that illustrates the intersectPoints and coordinate methods.



```

Unit(1, "mm");

if (_bOnInsert) {
    Grid gr = getGrid();
    _Entity.append(gr);
    _Pt0 = getPoint("Select first point");
    _PtG.append(getPoint("Select second point"));
    return;
}

// check array contents
if (_Entity.length()==0) { eraseInstance(); return; }
if (_PtG.length()==0) { eraseInstance(); return; }

Grid gr = (Grid)_Entity[0];
Point3d pt1 = _Pt0;
Point3d pt2 = _PtG[0];

// coordinate system of grid
CoordSys csGr = gr.coordSys();
csGr.vis(1);

// determine the grid coordinate string representation
String strX = gr.coordinate(pt1, csGr.vecX());
reportMessage("\n\nGrid coordinate vecX is "+strX);
String strY = gr.coordinate(pt1, csGr.vecY());
reportMessage("\nGrid coordinate vecY is "+strY);
String strZ = gr.coordinate(pt1, csGr.vecZ());
reportMessage("\nGrid coordinate vecZ is "+strZ);

```

```

// construct a linesegment, and find intersection points on Grid
LineSeg seg(pt1,pt2);
Vector3d vecRead = pt2-pt1;
Vector3d vecUp = csGr.vecZ().crossProduct(vecRead);

Vector3d vecDir = pt2-pt1; // take the gridlines most perpendicular
to vecDir
// If vecZ is the direction, then the complete coordinate is given.
// Also all intersection points are returned.
// vecDir = csGr.vecZ(); // uncomment line to use it

Point3d pnts[] = gr.intersectPoints(seg, vecDir);

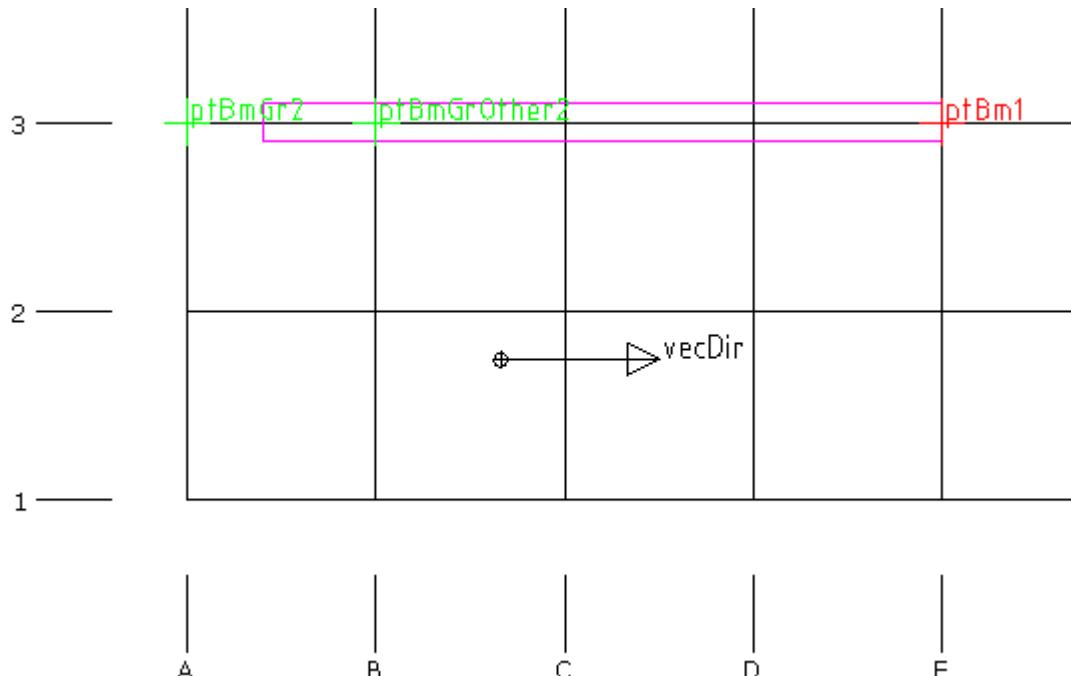
Display dp(-1);
dp.textHeight(1000);
dp.draw(seg);

for (int p=0; p<pnts.length(); p++) {
    Point3d pt = pnts[p];
    String str = gr.coordinate(pt, vecDir);
    dp.draw(str, pt, vecRead, vecUp, 0, 0);
}

```

—end example]

[Example O-type, with insert done inside script, that illustrates the hasClosestPointTo and bIsOn methods.



**Unit**(1, "mm");

```

double dEps = U(0.01);

if (_bOnInsert) {
    _Beam.append(getBeam());
    return;
}

// check array contents
if (_Beam.length()==0) { eraseInstance(); return; }

Beam bm = _Beam[0];
setDependencyOnBeamLength(bm);

Grid gr = bm.grid();
CoordSys csGr = gr.coordSys();

// determine linesegment from grid
Point3d ptBm1 = bm.ptCen() + 0.5*bm.dL()*bm.vecX();
Point3d ptBm2 = bm.ptCen() - 0.5*bm.dL()*bm.vecX();

LineSeg seg(ptBm1,ptBm2);

Vector3d vecDir = bm.vecX();
vecDir.vis(_Pt0);
Point3d pnts[] = gr.intersectPoints(seg,vecDir);

Point3d ptBmGr1, ptBmGrOther1;
if (gr.bIsOn(ptBm1,vecDir,dEps)) {
    ptBm1.vis(1);
}
else if (gr.hasClosestPointTo(ptBm1,vecDir,ptBmGr1,ptBmGrOther1)) {
    ptBmGr1.vis(5);
    ptBmGrOther1.vis(5);
}

Point3d ptBmGr2, ptBmGrOther2;
if (gr.bIsOn(ptBm2,vecDir,dEps)) {
    ptBm2.vis(1);
}
else if (gr.hasClosestPointTo(ptBm2,vecDir,ptBmGr2,ptBmGrOther2)) {
    ptBmGr2.vis(3);
    ptBmGrOther2.vis(3);
}

```

*—end example]*

## 7.27 Opening

The term Opening refers to an instance of an ADT opening: a door, window, opening, or an ADT opening assembly.

An entity can be casted into an Opening. However when the casting is not allowed, the resulting Opening will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because Opening is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity: eg.:

```
Entity ent = _Opening[0];
```

When you link the TSL instance with an opening, it will be automatically added to the predefined array `_Opening`.

```
Opening _Opening[];
```

During `_bOnInsert`, the `getOpening()` function can be used to query the user to select an opening instance.

```
Opening getOpening();  
Opening getOpening(String strPrompt);
```

When the `openingType` is queried, the returned value is one of the following:

```
_kNoOpening, _kOpening, _kWindow, _kDoor, _kAssembly
```

For Windows and Doors, that use an ACA style in which a frame width is set different from 0, there is a difference for "measurement inside of frame" compared to "outside of frame". The outside of frame state means that the width and height match the `widthRough` and `heightRough` values. But for the "inside of frame", the `widthRough` and `heightRough` are different from the width and height. The frame width is taken into account in that case. The values `sillHeight` and `headHeight` always refer to the rough dimensions. Since hsbCAD2010, build 16.0.105, the `headHeight` is now calculated correctly in this case.

---

```
class Opening : Entity { // see Entity for base functions

    CoordSys coordSys() const;

    double width() const;
    void setWidth(double dValue);
    void setWidth(double dValue, Point3d ptFixedXY);

    double height() const;
    void setHeight(double dValue);

    double rise() const;
    void setRise(double dValue);
```

---

```

double sillHeight() const;
void setSillHeight(double dValue);

double headHeight() const;
void setHeadHeight(double dValue);

Element element() const;

Entity parentEntity() const; // return the parent entity, a Wall or an Opening of type
    _kAssembly.
void releaseParentEntity(); // (added hsbCAD18.1.8) releases the anchor to the wall or
    assembly.
int setParentEntity(Point3d ptLoc, Vector3d vecX, Vector3d vecZ); // (added hsbCAD18.1.8)
    changes the parent entity by finding the closest Wall. Return success of the
    operation.

PLine plShape(Point3d ptInXY) const;
PLine plShape() const;

String description() const;
void setDescription(String strVal);

double heightRough() const;
double widthRough() const;

int openingType() const;
String standardSizeNameInUse() const;

void dbCreate(int openingType, double dWidth, double dHeight, CoordSys ecs, String
    styleName, int bStoringRoughDimensions, Entity parentEntity); // create a new
    database resident instance (added hsbCAD18.0.31)

int bStoringRoughDimensions() const; // (added hsbCAD18.1.8)
void setBStoringRoughDimensions(int bSet); // (added hsbCAD18.1.8) change value of
    measure to inside of frame / outside of frame.

String type() const; // (moved from OpeningSF and OpeningLog to Opening in hsbCAD2015
    build 20.0.49)
void setType(String strVal); // (moved from OpeningSF and OpeningLog to Opening in
    hsbCAD2015 build 20.0.49)

String openingDescr() const; // (renamed from descrSF in hsbCAD2009+ build 14.2.10,
    hsbCAD2010 build 15.1.5, moved from OpeningSF and OpeningLog to Opening in
    hsbCAD2015 build 20.0.49)
void setOpeningDescr(String strVal); // (renamed from setDescrSF in hsbCAD2009+ build
    14.2.10, hsbCAD2010 build 15.1.5, moved from OpeningSF and OpeningLog to
    Opening in hsbCAD2015 build 20.0.49)
};

```

**CoordSys** coordSys() const;

Build the coordinate system of the opening. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

```
void setSillHeight(double dValue);
void setHeadHeight(double dValue);
```

Both routines will move the opening such that the corresponding value is set correct. The width and height are not changed. So setting the sill height can change the head height too.

```
PLine plShape(Point3d ptInXY) const;
PLine plShape() const;
```

The PLine shape in the XY plane (in coordinate system of the opening) through the point ptInXY. If no point is given, the PLine is taken in the plane through the ptOrg of the coordinate system of the opening.

```
void setWidth(double dValue, Point3d ptFixedXY);
```

When the width of an opening is changed, one can specify a point in the window's XY plane that will be a fixed point.

```
int openingType() const;
```

When the openingType is queried, the returned value is one of the following:

**\_kNoOpening, \_kOpening, \_kWindow, \_kDoor, \_kAssembly**

The **\_kNoOpening** is returned when the Opening reference is not valid.

```
int bStoringRoughDimensions() const; // (added hsbCAD18.1.8)
void setBStoringRoughDimensions(int bSet); // (added hsbCAD18.1.8)
```

These methods only have effect on doors and windows that have a frame width set in their style. For these openings, one can specify the width and height to be relative to the inside or the outside of the frame. The bStoreRoughDimensions corresponds to this value.

[*Sample*

```
Opening op = _Opening[0];

String strToggleRough = T("|Toggle bStoringRoughDimensions|");
addRecalcTrigger(_kContext, strToggleRough );
if (_bOnRecalc && _kExecuteKey==strToggleRough ) {

    op.setBStoringRoughDimensions (!op.bStoringRoughDimensions ());
}
—end example]
```

```
Entity parentEntity() const; // return the parent entity, a Wall or an Opening of type  
_kAssembly.  
void releaseParentEntity(); // (added hsbCAD18.1.8) releases the anchor to the wall or  
assembly.  
int setParentEntity(Point3d ptLoc, Vector3d vecX, Vector3d vecZ); // (added hsbCAD18.1.8)  
changes the parent entity by finding the closest Wall. Return success of the  
operation.
```

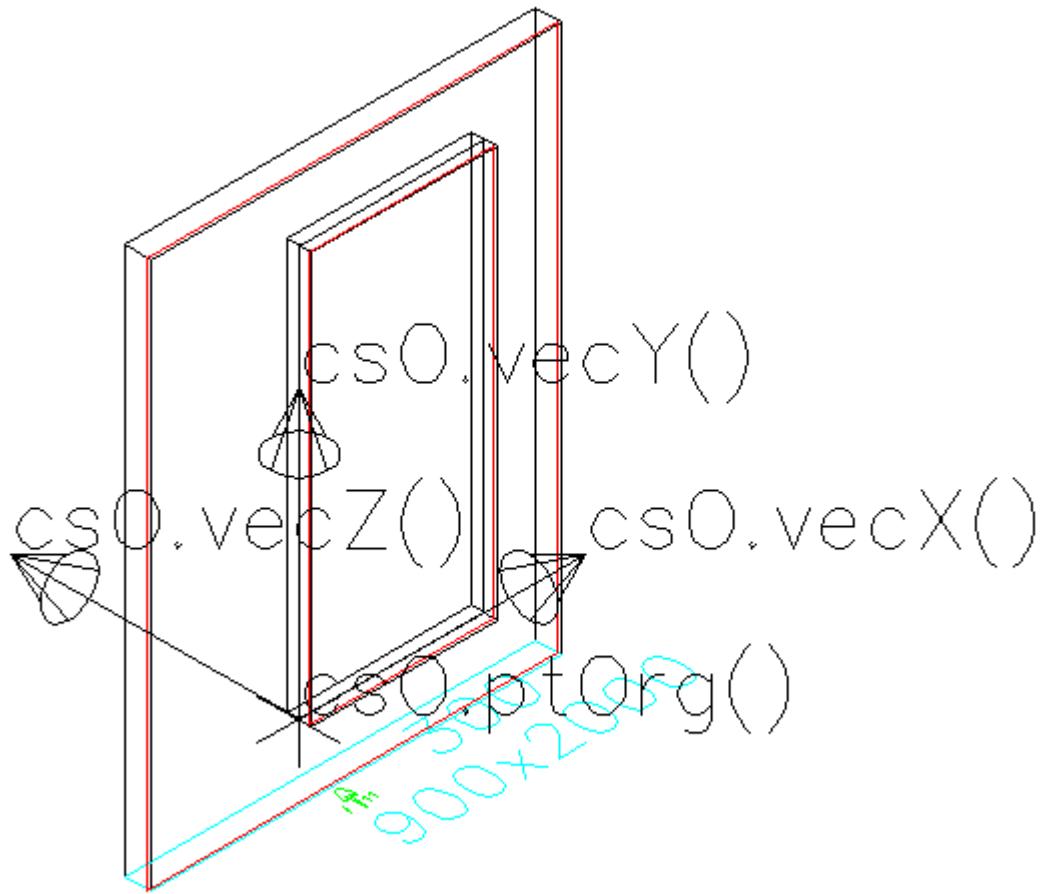
The parent entity is the wall or opening assembly to which the opening is anchored. If it is a wall, it is the wall that is cut by the opening. Releasing the parent entity does not move the opening, but it does remove the cutout from the wall.

When moving an anchored opening, ACA controls to which wall the opening will be anchored. This means we cannot explicitly assign the parent. But we can request a new location, and let ACA take over to assigning the correct wall.

Since the anchor also controls the flipped X and Z state, we can change the flipped state by calling the setParentEntity with its current (or adjusted) location. (example below)

---

*[Example O-type, with insert done inside script:*



```

U(1,"mm");
if (_bOnInsert) {

    Pt0 = getPoint(); // select point
    Opening op = getOpening();
    Opening.append(op);

    return;
}

Display dp(1); // color red

for (int i=0; i<Opening.length(); i++) {
    CoordSys csO = Opening[i].coordSys();
    csO.vis();
    reportNotice("\n\n_Opening.width = " + Opening[i].width() );
    reportNotice("\n_Opening.height = " + Opening[i].height () );
    reportNotice("\n_Opening.rise = " + Opening[i].rise() );
    reportNotice("\n_Opening.sillHeight = " + Opening[i].sillHeight() );
    reportNotice("\n_Opening.headHeight = " + Opening[i].headHeight() );
    reportNotice("\n_Opening.description = " + Opening[i].description() );

    Element el = Opening[i].element();
}

```

```

PLine plel = el.plEnvelope();
dp.draw(plel);

PLine plop = _Opening[i].plShape(el.ptOrg());
dp.draw(plop);
}

/*
// reports the following for the image shown:
_Opening.width = 900
_Opening.height = 2000
_Opening.rise = 0.3
_Opening.sillHeight = 600
_Opening.headHeight = 2600
*/

```

—end example]

---

[Example O-type, illustrating the use of setWidth with a fixed point:

```

U(1,"mm");
PropDouble dWidth(0,U(300),"Opening width");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Opening op = getOpening();
    _Opening.append(op);

    return;
}

for (int i=0; i<_Opening.length(); i++) {

    _Opening[i].setWidth(dWidth, _Pt0);

    CoordSys csO = _Opening[i].coordSys();
    csO.vis();
}
—end example]

```

---

[Example O-type, illustrating the use of dbCreate:

```

Unit(1, "inch"); // script uses inches
if (_bOnInsert) {

```

---

```

Wall wll = getWall("Select wall to attach opening to");
Point3d ptOrg = getPoint("Select location");

CoordSys csWll = wll.coordSys();
CoordSys csOp(ptOrg, csWll.vecX(), csWll.vecY(), csWll.vecZ());

int openingType = _kOpening;
double dWidth = U(24);
double dHeight = U(48);
String strStyleName = "";
int bStoreRoughDimensions = FALSE; // only meaningful in case of
// window or door with non zero frame width

Opening opNew;
opNew.dbCreate(openingType, dWidth, dHeight, csOp, strStyleName,
bStoreRoughDimensions, wll);

eraseInstance(); // this instance will not stay in the DB
return;
} // if (_bOnInsert)

—end example]

```

---

[Example O-type, illustrating some methods on Opening:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Opening op = getOpening();
    _Opening.append(op);

    return;
}

if (_Opening.length() == 0) {
    eraseInstance();
    return;
}
Opening op = _Opening[0];
setDependencyOnEntity(op);
CoordSys csO = op.coordSys();

String strToggleRough = T("|Toggle bStoringRoughDimensions|");
addRecalcTrigger(_kContext, strToggleRough );
if (_bOnRecalc && _kExecuteKey==strToggleRough ) {
    op.setBStoringRoughDimensions(!op.bStoringRoughDimensions());
}

```

---

```

}

String strReleaseParent = T("|Release parent|");
addRecalcTrigger(_kContext, strReleaseParent );
if (_bOnRecalc && _kExecuteKey==strReleaseParent ) {
    op.releaseParentEntity();
}

String strNewParent = T("|Select new parent|");
addRecalcTrigger(_kContext, strNewParent );
if (_bOnRecalc && _kExecuteKey==strNewParent ) {
    // automatically find new parent from location
    Point3d ptNew = getPoint();
    op.setParentEntity(ptNew, csO.vecX(), csO.vecZ());
}

String strFlipX = T("|Flip X|");
addRecalcTrigger(_kContext, strFlipX );
if (_bOnRecalc && _kExecuteKey==strFlipX ) {
    // will also automatically assign new parent
    op.setParentEntity(csO.ptOrg(), -csO.vecX(), csO.vecZ());
}

String strFlipZ = T("|Flip Z|");
addRecalcTrigger(_kContext, strFlipZ );
if (_bOnRecalc && _kExecuteKey==strFlipZ ) {
    // will also automatically assign new parent
    op.setParentEntity(csO.ptOrg(), csO.vecX(), -csO.vecZ());
}

String strCreateSF = T("|CreateSF|");
addRecalcTrigger(_kContext, strCreateSF);
if (_bOnRecalc && _kExecuteKey==strCreateSF ) {

    OpeningSF opsf;
    opsf.dbCreate(op);
    if (!opsf.bIsValid())
    {
        op.removeHsbData();
        opsf.dbCreate(op);
    }
    if (opsf.bIsValid())
        reportMessage("\nOpeningSF created fine!");
    else
        reportMessage("\nOpeningSF not created!");
}

String strCreateLog = T("|CreateLog|");
addRecalcTrigger(_kContext, strCreateLog );
if (_bOnRecalc && _kExecuteKey==strCreateLog ) {
}

```

```

OpeningLog oplg;
oplg.dbCreate(op);
if (!oplg.bIsValid())
{
    op.removeHsbData();
    oplg.dbCreate(op);
}
if (oplg.bIsValid())
    reportMessage("\nOpeningLog created fine!");
else
    reportMessage("\nOpeningLog not created!");
}

// maybe the coordSys was changed
cs0 = op.coordSys();
cs0.vis();

reportMessage("\nExecuting " + scriptName() + " with key " +
kExecuteKey);
reportMessage("\n_Opening.width = " + op.width() );
reportMessage("\n_Opening.height = " + op.height() );
reportMessage("\n_Opening.rise = " + op.rise() );
reportMessage("\n_Opening.sillHeight = " + op.sillHeight() );
reportMessage("\n_Opening.headHeight = " + op.headHeight() );
reportMessage("\n_Opening.bStoringRoughDimensions = " +
op.bStoringRoughDimensions() );
reportMessage("\n_Opening.parentEntity = "
+ op.parentEntity().handle() );
reportMessage("\n");

Display dp(-1); // color of tsl
PLine plop = op.plShape(cs0.ptOrg());
PlaneProfile pp(plop);
pp.shrink(U(200));
dp.draw(pp);

```

*—end example]*

## 7.28 OpeningLog

The term **OpeningLog** refers to an instance of an ADT opening to which HSB log opening data is attached.

An entity can be casted into an **OpeningLog**. However when the casting is not allowed, the resulting **OpeningLog** will become invalid. This can be checked with the **bIsValid()** function of Entity.

Because **OpeningLog** is derived from [Opening](#), it inherits all the member functions of **Opening** as well. It can also be assigned to an **Opening**.

When you link the TSL instance with an opening, it will be automatically added to the predefined array **\_Opening**. To use the i-th element of this array as a valid **OpeningLog**, the following can be used.

```
OpeningLog openLog = (OpeningLog) _Opening[i];
if (openLog.bIsValid()) {
    // here use openLog as a valid OpeningLog type
}
```

During `_bOnInsert`, the `getOpening()` function can be used to query the user to select an opening instance. With the above code sample, the casting to an `OpeningLog` can then be done.

```
Opening getOpening();
Opening getOpening(String strPrompt);
```

---

```
class OpeningLog : Opening { // see Opening for base functions

    double dGapLeft() const;
    void setDGapLeft(double dValue);
    double dGapRight() const;
    void setDGapRight(double dValue);
    double dGapTop() const;
    void setDGapTop(double dValue);
    double dGapBottom() const;
    void setDGapBottom(double dValue);

    double dThicknessLeft() const;
    void setDThicknessLeft(double dValue);
    double dThicknessRight() const;
    void setDThicknessRight(double dValue);

    double dWidthLeft() const;
    void setDWidthLeft(double dValue);
    double dWidthRight() const;
    void setDWidthRight(double dValue);

    double dDepthLeft() const;
    void setDDepthLeft(double dValue);
    double dDepthRight() const;
    void setDDepthRight(double dValue);

    double dMillHeightTop() const;
    void setDMillHeightTop(double dValue);
    double dMillWidthTop() const;
    void setDMillWidthTop(double dValue);
    double dMillExtraLengthTop() const;
    void setDMillExtraLengthTop(double dValue);

    double dMillHeightBottom() const;
    void setDMillHeightBottom(double dValue);
    double dMillWidthBottom() const;
    void setDMillWidthBottom(double dValue);
```

---

```

double dMillExtraLengthBottom() const;
void setDMillExtraLengthBottom(double dValue);

double dLateralCutOutAngle() const; // in deg
void setDLateralCutOutAngle(double dValue);
double dLateralCutOutWidth() const;
void setDLateralCutOutWidth(double dValue);
double dOffsetLateralMillings() const;
void setDOffsetLateralMillings(double dValue);
double dOffsetHorizontalMillings() const;
void setDOffsetHorizontalMillings(double dValue);

int bApplyNonSplittingTooling() const;
void setBApplyNonSplittingTooling(int bVal); // TRUE or FALSE
int bSymmetrical() const;
void setBSymmetrical(int bVal); // TRUE or FALSE
int bMeasureInLogs() const;
void setBMeasureInLogs(int bVal); // TRUE or FALSE

void dbCreate(Opening opening); // (added hsbCAD19.1.101 and hsbCAD20.0.68)
};

```

---

*[Example O-type, with insert done inside script:*

```

U(1,"mm");
if (_bOnInsert) {
    _Pt0 = getPoint(); // select point
    _Opening.append(getOpening()); // select an opening

    return;
}

// make sure 0 is a valid index in the _Opening array
if (_Opening.length()==0) { // if no openings attached
    eraseInstance(); // just erase from DB
    return;
}

// make sure we have the right type of opening
OpeningLog opLog = (OpeningLog)_Opening[0]; // explicit cast
if (!opLog.bIsValid()) { // want to keep only the OpeningLog types
    eraseInstance(); // just erase from DB
    return;
}

// all Opening as well as OpeningLog properties are allowed

```

---

```

opLog.setWidth(U(456));
CoordSys csO = opLog.coordSys();
csO.vis();
reportNotice("\n\nOpeningLog.width = " +opLog.width() );
reportNotice("\nOpeningLog.height = " +opLog.height () );
reportNotice("\nOpeningLog.rise = " +opLog.rise() );
reportNotice("\nOpeningLog.sillHeight = " +opLog.sillHeight() );
reportNotice("\nOpeningLog.headHeight = " +opLog.headHeight() );

reportNotice("\nOpeningLog.dGapTop = " +opLog.dGapTop() );
reportNotice("\nOpeningLog.dGapBottom = " +opLog.dGapBottom() );

```

*—end example]*

## 7.29 OpeningSF

The term OpeningSF refers to an instance of an ADT opening to which HSB stick frame data is attached.

An entity can be casted into an OpeningSF. However when the casting is not allowed, the resulting OpeningSF will become invalid. This can be checked with the blsValid() function of Entity.

Because OpeningSF is derived from [Opening](#), it inherits all the member functions of Opening as well. It can also be assigned to an Opening.

When you link the TSL instance with an opening, it will be automatically added to the predefined array \_Opening. To use the i-th element of this array as a valid OpeningSF, the following can be used.

```

OpeningSF openSF = (OpeningSF) _Opening[i];
if (openSF.blsValid()) {
    // here use openSF as a valid OpeningSF type
}

```

During \_bOnInsert, the getOpening() function can be used to query the user to select an opening instance. With the above code sample, the casting to an OpeningSF can then be done.

```

Opening getOpening();
Opening getOpening(String strPrompt);

```

---

```
class OpeningSF : Opening { // see Opening for base functions
```

```

double dBeam() const;
void setDBeam(double dValue);
double dBeamRight() const;
void setDBeamRight(double dValue);

int bHeaderBelowPlate() const;
void setBHeaderBelowPlate(int nVal);

```

```
int bExtraBeamsTop() const;
void setBExtraBeamsTop(int nVal);
double dSideBeamTop() const;
void setDSideBeamTop(double dValue);

int bExtraBeamsBottom() const;
void setBExtraBeamsBottom(int nVal);
double dSideBeamBottom() const;
void setDSideBeamBottom(double dValue);

double dGapSide() const;
void setDGapSide(double dValue);
double dGapTop() const;
void setDGapTop(double dValue);
double dGapBottom() const;
void setDGapBottom(double dValue);

double dPlate() const;
void setDPlate(double dValue);

double dExtraPlateTop() const;
void setDExtraPlateTop(double dValue);
double dExtraPlateBottom() const;
void setDExtraPlateBottom(double dValue);

String constrDetail() const;
String constrDetailTop() const;
String constrDetailBottom() const;
String constrDetailLeft() const;
String constrDetailRight() const;
void setConstrDetail(String strVal);
void setConstrDetailTop(String strVal);
void setConstrDetailBottom(String strVal);
void setConstrDetailLeft(String strVal);
void seConstrDetailRight(String strVal);

// added hsbCAD2011, build 16.0.40
String constrDetailShimTop() const;
String constrDetailShimBottom() const;
String constrDetailShimLeft() const;
String constrDetailShimRight() const;
void setConstrDetailShimTop(String strVal);
void setConstrDetailShimBottom(String strVal);
void setConstrDetailShimLeft(String strVal);
void seConstrDetailShimRight(String strVal);

String constrSheetingTop() const;
String constrSheetingBottom() const;
String constrSheetingLeft() const;
String constrSheetingRight() const;
void setConstrSheetingTop(String strVal);
void setConstrSheetingBottom(String strVal);
```

```

void setConstrSheetingLeft(String strVal);
void seConstrSheetingRight(String strVal);

String continueSheeting() const;
void setContinueSheeting(String strVal);

String descrPlate() const;
void setDescrPlate(String strVal);

double dPackingMaxHeight() const;
void setDPackingMaxHeight(double dValue);
String descrPacking() const;
void setDescrPacking(String strVal);

String cutBottomBeam() const;
void setCutBottomBeam(String strVal);

int numBeamsSupport() const;
void setNumBeamsSupport(int nVal);
int numBeamsNoSupport() const;
void setNumBeamsNoSupport(int nVal);
int numBeamsSupportRight() const;
void setNumBeamsSupportRight(int nVal);
int numBeamsNoSupportRight() const;
void setNumBeamsNoSupportRight(int nVal);

double dSillPlateTop() const;
void setDSillPlateTop(double dValue);
double dSillPlateTopUp() const;
void setDSillPlateTopUp(double dValue);
double dSillPlateTopDown() const;
void setDSillPlateTopDown(double dValue);

// properties added hsbCAD2009+, build 14.0.63
double dNonSuppBeamLeft() const;
void setDNonSuppBeamLeft(double dValue);
double dNonSuppBeamRight() const;
void setDNonSuppBeamRight(double dValue);
double dShiftHeaderInTopPlate() const;
void setDShiftHeaderInTopPlate(double dValue);
double dShiftHeaderInTopPlate();

// added hsbCAD2011, build 16.0.23
String styleNameSF() const;
void setStyleNameSF(String strVal);
int nFloorsToCarry() const;
void setNFloorsToCarry(int nVal);

void dbCreate(Opening opening); // (added hsbCAD19.1.101 and hsbCAD20.0.68)
};

```

```
String styleNameSF() const;
void setStyleNameSF(String strVal);
```

When the styleNameSF is set to an empty string, the style is not in use. Whenever the styleNameSF is changed by the setStyleNameSF, the instance values are also set by reading in the catalog and setting the values. If the styleNameSF is set to empty, the previous styleNameSF is used to update the instance values. This way we are trying to assure that the instance values are always corresponding to the style values whenever the user decides to go to instance values.

---

*[Example O-type, with insert done inside script:*

```
U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    _Opening.append(getOpening()); // select an opening

    return;
}

// make sure 0 is a valid index in the _Opening array
if (_Opening.length()==0) { // if no openings attached
    eraseInstance(); // just erase from DB
    return;
}

// make sure we have the right type of opening
OpeningSF opSf = (OpeningSF)_Opening[0]; // explicit cast
if (!opSf.bIsValid()) { // want to keep only the OpeningSF types
    eraseInstance(); // just erase from DB
    return;
}

// all Opening as well as OpeningSF properties are allowed
opSf.setWidth(U(456));
CoordSys csO = opSf.coordSys();
csO.vis();
reportNotice("\n\nOpeningSF.width = " +opSf.width());
reportNotice("\nOpeningSF.height = " +opSf.height ());
reportNotice("\nOpeningSF.rise = " +opSf.rise());
reportNotice("\nOpeningSF.sillHeight = " +opSf.sillHeight());
reportNotice("\nOpeningSF.headHeight = " +opSf.headHeight());
reportNotice("\nOpeningSF.dGapTop = " +opSf.dGapTop());
reportNotice("\nOpeningSF.dGapSide = " +opSf.dGapSide());
reportNotice("\nOpeningSF.descrSF = " +opSf.descrSF());
```

*--end example]*

## 7.30 OpeningRoof

The term OpeningRoof refers to an instance of the hsb roof opening entity

An entity can be casted into an OpeningRoof. However when the casting is not allowed, the resulting OpeningRoof will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because OpeningRoof is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity.

When you link the TSL instance with an opening, you must add it to the entity array `_Entity`.

During `_bOnInsert`, the `getOpeningRoof()` function can be used to query the user to select an opening instance.

```
OpeningRoof getOpeningRoof();
OpeningRoof getOpeningRoof(String strPrompt);
```

---

```
class OpeningRoof : Entity { // see Entity for base functions

    CoordSys coordSys() const;

    void dbCreate(Group floorGroup, PLine pline);

    double dOpeningHeight() const;
    void setDOpeningHeight(double dValue);
    double dTopHeight() const;
    void setDTopHeight(double dValue);

    int bAdjustPositionToTileLath() const;
    void setBAdjustPositionToTileLath(int bSet); // set to TRUE or FALSE
    double dDistanceToFirstTileLath() const;
    void setDDistanceToFirstTileLath(double dValue);

    int nExtraRaftersRight() const;
    void setNExtraRaftersRight(int nAmount);
    int nExtraRaftersLeft() const;
    void setNExtraRaftersLeft(int nAmount);

    PLine plShape(Point3d ptInXY) const;
    PLine plShape() const;
    void setPIShape(PLine pl);

    String description() const;
    void setDescription(String strVal);

    String constrDetailLeft() const;
```

---

```

void setConstrDetailLeft(String strVal);
String constrDetailRight() const;
void setConstrDetailRight(String strVal);
String constrDetailTop() const;
void setConstrDetailTop(String strVal);
String constrDetailBottom() const;
void setConstrDetailBottom(String strVal);
String constrDetailTopAngled() const;
void setConstrDetailTopAngled(String strVal);
String constrDetailBottomAngled() const;
void setConstrDetailBottomAngled(String strVal);

// the following methods were added hsbCAD2011 (build 16.0.59) and hsbCAD2010 (build
15.3.27)
String[] getListOfCatalogNames(int bFloorNotRoof) const; // return the list of catalog entry
names for the roof or floor catalog
int setValuesFromCatalog(String strCatalogEntryName, int bFloorNotRoof); // return TRUE if
successful

int blsInFloor() const; // return TRUE if this is a floor (added hsbCAD2011 build 16.0.59 and
hsbCAD2010 build 15.3.27)

};

```

---

**CoordSys** coordSys() const;

Build the coordinate system of the opening. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

```

PLine plShape(Point3d ptInXY) const;
PLine plShape() const;
void setPlShape(PLine pl);

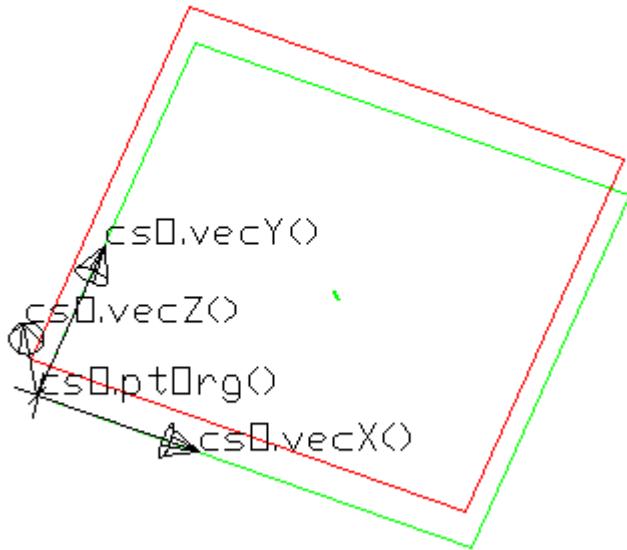
```

The PLine shape in the XY plane (in coordinate system of the opening) through the point ptInXY. If no point is given, the PLine is taken in the plane through the ptOrg of the coordinate system of the opening. With the setPlShape, the shape of the OpeningRoof can be altered.

**void** dbCreate(**Group** floorGroup, **PLine** pline);

When a new OpeningRoof is created, it is added automatically to the floor group given as first argument. If the group is a 3rd level group, only the floor part is taken. If the group is a 1st level group, then the opening is not added to the group at all.

[Example O-type, with insert done inside script:



```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    OpeningRoof op = getOpeningRoof();
    _Entity.append(op);

    return;
}

Display dp(1); // color red

for (int i=0; i<_Entity.length(); i++) {

    OpeningRoof opr = (OpeningRoof)_Entity[i];
    if (!opr.bIsValid()) continue; // go to next i

    CoordSys cs0 = opr.coordSys();
    cs0.vis();
    reportNotice("\n\n_OpeningRoof.description = "
+opr.description());
    reportNotice("\n_OpeningRoof.dOpeningHeight = "
+opr.dOpeningHeight());
    reportNotice("\n_OpeningRoof.dTopHeight = " +opr.dTopHeight());

    PLine plop = opr.plShape(_Pt0);
    dp.draw(plop);
}

```

—end example]

[Example O-type, with insert done inside script to illustrate the dbCreate

```

Unit(1, "mm");
if (_bOnInsert) {

    Group gr = _kCurrentGroup;
    if ( (gr.namePart(0)== "") || (gr.namePart(1)== "") ) {
        reportMessage("\nMake a floor group current before inserting
this TSL.");
        eraseInstance(); return;
    }

    EntPLine plent = getEntPLine();
    PLine pl = plent.getPLine();

    OpeningRoof op;
    op.dbCreate(gr,pl); // define geometry, and add to floor group

    op.setNExtraRaftersRight(1);
    op.setNExtraRaftersLeft(2);
    op.setConstrDetailRight("rechts;aa");
    op.setConstrDetailLeft("left;aa");
    op.setDescription("desctr");
    op.setBAdjustPositionToTileLath(TRUE);
    op.setDDistanceToFirstTileLath(U(200));

    _Entity.append(op); // add to persistent array for later use

    return;
}

if (_Entity.length()==0) return;
OpeningRoof opRoof = (OpeningRoof) _Entity[0];
if (!opRoof .bIsValid()) return;

CoordSys csOp = opRoof.coordSys();
_Pt0 = csOp.ptOrg();

PLine pl = opRoof.plShape(_Pt0);
pl.vis(1);

reportMessage("\nright "+opRoof.nExtraRaftersRight());
reportMessage("\nleft "+opRoof.nExtraRaftersLeft());
reportMessage("\ndetail left "+opRoof.constrDetailLeft());

reportMessage("\nadjust "+opRoof.bAdjustPositionToTileLath());
reportMessage("\ndist "+opRoof.dDistanceToFirstTileLath());

```

—end example]

[Example O-type, illustrating the use of the catalog reading:

```

Unit(1, "mm");
if (_bOnInsert) {
    _Entity.append(getOpeningRoof());
    _Pt0 = getPoint();
    return;
}

if (_Entity.length() == 0) return;
OpeningRoof opRoof = (OpeningRoof) _Entity[0];
if (!opRoof.bIsValid()) return;

// get the polyline of this element
PLine plEl = opRoof.plShape();

CoordSys cs = opRoof.coordSys();
Line ln(cs.ptOrg(), cs.vecX());
cs.vis();
plEl.vis();

reportMessage("\n ");
reportMessage("\n bIsInFloor: "+ opRoof.bIsInFloor());

// collect all the catalog values, and report them
int bFloorNotRoof= opRoof.bIsInFloor();
String arEntries[] = opRoof.getListOfCatalogNames(bFloorNotRoof);
for (int e=0; e<arEntries.length();e++) {
    reportMessage("\n entry["+e+"]= "+ arEntries[e]);
}

String strLoadCatalog = T("Load catalog entry");
addRecalcTrigger(_kContext, strLoadCatalog);
if (_bOnRecalc && _kExecuteKey==strLoadCatalog)
{
    int nIndex = getInt("enter index of catalog to read");
    if (nIndex>=0 && nIndex<arEntries.length())
    {
        String strEntry = arEntries[nIndex];
        reportMessage("\nReading catalog called: "+strEntry);
        opRoof.setValuesFromCatalog(strEntry,bFloorNotRoof);
    }
}

reportMessage("\n right "+opRoof.nExtraRaftersRight());
reportMessage("\n left "+opRoof.nExtraRaftersLeft());
reportMessage("\n detail left "+opRoof.constrDetailLeft());
reportMessage("\n adjust "+opRoof.bAdjustPositionToTileLath());
reportMessage("\n dist "+opRoof.dDistanceToFirstTileLath());

```

—end example]

## 7.31 MassElement

The term MassElement refers to an instance of a mass element entity.

An entity can be casted into a MassElement. However when the casting is not allowed, the resulting MassElement will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because MassElement is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getMassElement()` function can be used to query the user to select a MassElement instance.

```
MassElement getMassElement();
MassElement getMassElement(String strPrompt);
```

Each MassElement has a shapeType. The shapeType is one of the following int values:

```
_kMSTBox, _kMSTArch, _kMSTBarrelVault, _kMSTDoric, _kMSTCone,
_kMSTCylinder, _kMSTDome, _kMSTGable, _kMSTIsocTriangle,
_kMSTRightTriangle, _kMSTPyramid, _kMSTSphere, _kMSTExtrusion,
_kMSTRevolution, _kMSTBrep
```

The 2 methods `shapeType2String` and `string2ShapeType` allow to convert the int value to a readable string.

---

```
class MassElement : Entity { // see Entity for base functions (added hsbCAD2012 17.0.23)

    CoordSys coordSys() const;

    int hasWidth() const; // return TRUE if the entity has a width, else return FALSE.
    double width() const; // return the width if applicable, else return 0
    int setWidth(double dVal); // change the width to dVal. Return TRUE if successful. If not
                                // applicable, return FALSE.

    int hasDepth() const; // return TRUE if the entity has a depth, else return FALSE.
    double depth() const; // return the depth if applicable, else return 0
    int setDepth(double dVal); // change the depth to dVal. Return TRUE if successful. If not
                                // applicable, return FALSE.

    int hasHeight() const; // return TRUE if the entity has a height, else return FALSE.
    double height() const; // return the height if applicable, else return 0
    int setHeight(double dVal); // change the height to dVal. Return TRUE if successful. If not
                                // applicable, return FALSE.

    int hasRadius() const; // return TRUE if the entity has a radius, else return FALSE.
    double radius() const; // return the radius if applicable, else return 0
```

---

```

int setRadius(double dVal); // change the radius to dVal. Return TRUE if successful. If not
// applicable, return FALSE.

int hasRise() const; // return TRUE if the entity has a rise, else return FALSE.
double rise() const; // return the rise if applicable, else return 0
int setRise(double dVal); // change the rise to dVal. Return TRUE if successful. If not
// applicable, return FALSE.

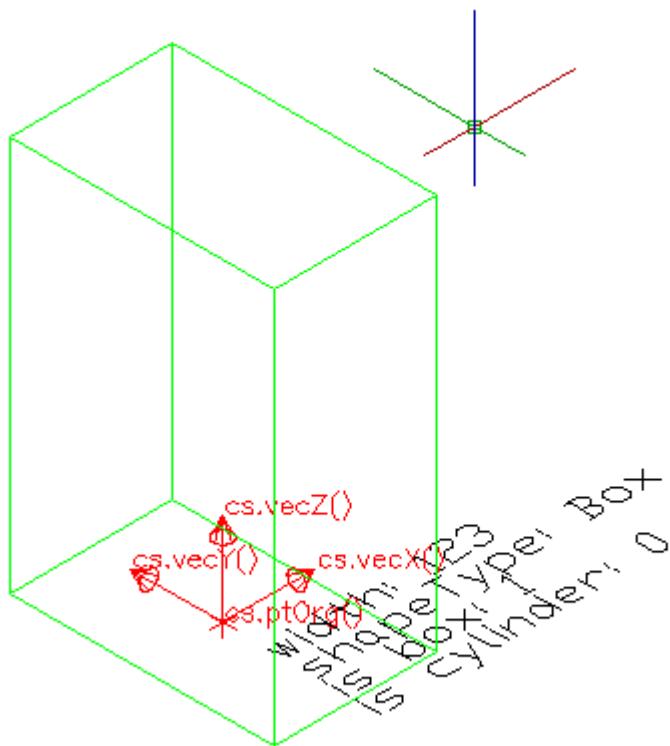
int shapeType() const; // get the shape type

static String shapeType2String(int nShapeType); // get the readable shapetype
static int string2ShapeType(String strShapeType); // get int value from the readable
// shapetype

};

```

[Example O-type, with insert done inside script:



```

U(1, "mm");
if (_bOnInsert) {

    MassElement sl = getMassElement();
}

```

```

_Entity.append(sl);

return;
}

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

if (_Entity.length()==0) return;
MassElement sp = (MassElement)_Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strChangeEntity = T("Change entity");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {

    if (sp.hasWidth()) sp.setWidth(U(123));
    if (sp.hasHeight()) sp.setHeight(U(223));
    if (sp.hasDepth()) sp.setDepth(U(323));
    if (sp.hasRadius()) sp.setRadius(U(50));
    if (sp.hasRise()) sp.setRise(U(23));
}

CoordSys cs =sp.coordSys();
cs.vis(1);

setDependencyOnEntity(sp);
_pt0 = cs.ptOrg();

if (_bOnDebug) {
    Body bd = sp.realBody();
    bd.vis(3);
}
}

String strLines[0];
strLines.append(""); strLines.append("");
if (sp.hasWidth()) strLines.append("width: "+sp.width());
if (sp.hasDepth()) strLines.append("depth: "+sp.depth());
if (sp.hasHeight()) strLines.append("height: "+sp.height());
if (sp.hasRadius()) strLines.append("radius: "+sp.radius());
if (sp.hasRise()) strLines.append("rise: "+sp.rise());
strLines.append("shapeType: "+sp.shapeType2String(sp.shapeType()));
strLines.append("is box: "+(sp.shapeType() == _kMSTBox));
strLines.append("is cylinder: "+(sp.shapeType() == _kMSTCylinder));

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);

```

```

dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example]*

## 7.32 MassGroup

The term MassGroup refers to an instance of a mass element entity.

An entity can be casted into a MassGroup. However when the casting is not allowed, the resulting MassGroup will become invalid. This can be checked with the `isValid()` function of Entity.

Because MassGroup is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getMassGroup()` function can be used to query the user to select a MassGroup instance.

```

MassGroup getMassGroup();
MassGroup getMassGroup(String strPrompt);

```

Each entity inside the MassGroup has an operation type. The operation type is one of the following int values:

`_kAOPAdditive, _kAOPSubtractive, _kAOPIIntersect`

```

class MassGroup : Entity { // see Entity for base functions (added hsbCAD2012 17.0.23)

    CoordSys coordSys() const;

    Entity[] entity() const; // return the list of entities inside the mass group
    int operation(Entity ent) const; // get the operation type: _kAOPAdditive,
                                    _kAOPSubtractive, _kAOPIIntersect

    void setCoordSysOnly(CoordSys csNewCoordSys); // It sets the coordsys of the MassGroup,
        but in contrast to the transformBy on MassGroup, it leaves the entities of the
        MassGroup untouched. (added hsbCAD2012 build 17.1.22).

    int posnum() const; // (added build 17.1.29 and 18.0.24) posnum value, could be -1 if no
                       posnum is assigned
    static int assignPosnums(int nPosnumToStartAssigning, <Entity>[]
                           arMassGroupsToConsider); // (added build 17.1.28 and 18.0.24) assign a
                                     posnum value to all the massgroups in the arMassGroupsToConsider, including
                                     its children.

```

```

static int assignPosnums(int nPosnumToStartAssigning, <Entity>[] arMassGroupsToConsider,
int bKeepExisting, int bSpecifiedMassGroups, int
bChilderenOfSpecifiedMassGroups); // (added build 17.2.6 and 18.1.9)

static int releasePosnums(<Entity>[] arMassGroupsToConsider, int bSpecifiedMassGroups, int
bChilderenOfSpecifiedMassGroups); // (added build 17.2.6 and 18.1.9)
};

```

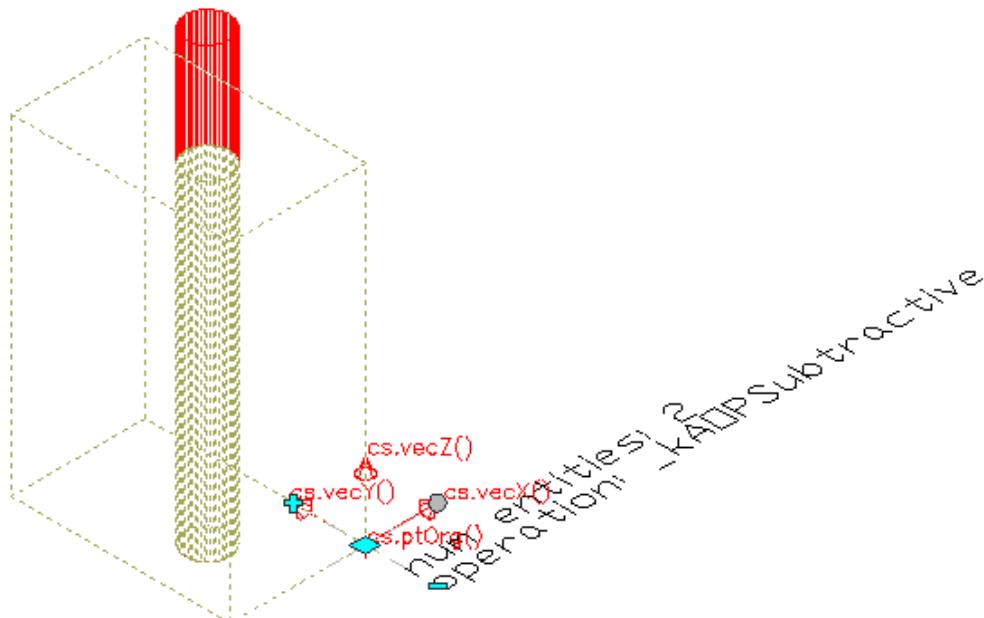
---

**int** operation(**Entity** ent) const;

The integer that is returned is one of the following values: -1, **\_kAOPAdditive**, **\_kAOPSubtractive**, **\_kAOPIIntersect**. The value -1 is returned if the entity is not part of the MassGroup.

---

[Example O-type, with insert done inside script:



```

U(1, "mm");
if (_bOnInsert) {

    MassGroup sl = getMassGroup();
    Entity.append(sl);
}

```

---

```

        return;
    }

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");
PropInt pIndex(0,-1,"index of member to vis");

if (_Entity.length()==0) return;
MassGroup sp = (MassGroup)_Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

CoordSys cs =sp.coordSys();
cs.vis(1);

setDependencyOnEntity(sp);
_Pt0 = cs.ptOrg();

if (_bOnDebug) {
    Body bd = sp.realBody();
    bd.vis(3);
}

String strLines[0];
strLines.append(""); strLines.append("");

Entity ents[] = sp.entity();
strLines.append("num entities: "+ents.length());

for (int b=0; b<ents.length(); b++)
{
    if (pIndex>=0 && pIndex!=b)
        continue;
    Entity ent = ents[b];

    Body bde = ent.realBody();
    bde.vis(1);

    int nOperation = sp.operation(ent);
    if (nOperation==_kAOPAdditive)
        strLines.append("operation: "+"_kAOPAdditive");
    else if (nOperation==_kAOPSubtractive)
        strLines.append("operation: "+"_kAOPSubtractive");
    else if (nOperation==_kAOPIIntersect)
        strLines.append("operation: "+"_kAOPIIntersect");
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);

```

```

dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example]*

## 7.33 MasterPanel

The term MasterPanel refers to an instance of a master panel entity used in the cnc output of [Sip](#) panels using child panels.

An entity can be casted into a MasterPanel. However when the casting is not allowed, the resulting MasterPanel will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because MasterPanel is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getMasterPanel()` function can be used to query the user to select a MasterPanel instance.

```

MasterPanel getMasterPanel();
MasterPanel getMasterPanel(String strPrompt);

```

---

```

class MasterPanel : Entity { // see Entity for base functions (added hsbCAD14.0.68)

    void dbCreate(String strStyle, CoordSys csEcs, PLine pline);
    void dbCreate(String strStyle, CoordSys csEcs, double dLength, double dWidth);

    CoordSys coordSys() const;

    String style() const;
    void setStyle(String strVal); // see MasterPanelStyle for more information

    String information() const;
    void setInformation(String strVal);

    String name() const;
    void setName(String strVal);

    int number() const;
    void setNumber(int nVal);

    Point3d ptRef() const; // (added v21.0.49 and v20.3.18)
    void setPtRef(Point3d pt); // (added v21.0.49 and v20.3.18)

    PLine plShape() const;
    void setPlShape(PLine pl);

    int openingCount() const; // (added v22.1.72 and v23.0.45)
    PLine plOpeningAt(int nIndex) const; // (added v22.1.72 and v23.0.45)
}

```

```

int removeOpening(int nIndex); // returns TRUE if opening was found (added v22.1.72 and
                                // v23.0.45)
void appendOpening(PLine pl); // (added v22.1.72 and v23.0.45)

double dThickness() const; // in vecZ direction. Equals the MasterPanelStyle dThickness
double dLength() const; // extents of plShape in vecY direction
double dWidth() const; // extents of plShape in vecX direction

double dYield() const; // fraction of the body from plShape that is not cut away by any
                        // ChildPanel.
void updateYield(); // recalculate the yield. This is required if child panels are modified.

ChildPanel[] nestedChildPanels() const; // Return the list of ChildPanel entities.

CncCurveEnt[] myCncCurveEnts() const; // (added 17.0.44) Return the list of CncCurveEnt
                                // entities.

String surfaceQualityOverrideTop() const; // see example below (added hsbCAD17.1.2)
void setSurfaceQualityOverrideTop(String strVal); // see SurfaceQualityStyle for more
                                // information
String surfaceQualityOverrideBottom() const; // see example below (added hsbCAD17.1.2)
void setSurfaceQualityOverrideBottom(String strVal); // see SurfaceQualityStyle for more
                                // information

Vector3d getMainGrainDirectionFromChildPanels(); // vector could be zero length if not
                                // defined (added hsbcad20.1.6 and hsbcad21.0.9)

void addTool(ElementTool tool); // added v21.1.30, allows to attach an ElemSaw,
                                // ElemMill, ElemDrill...to the MasterPanel.
};

```

[CoordSys](#) coordSys() const;

Build the coordinate system of the MasterPanel. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

```

String surfaceQualityOverrideTop() const; // see example below (added hsbCAD17.1.2)
void setSurfaceQualityOverrideTop(String strVal); // see SurfaceQualityStyle for more
                                // information
String surfaceQualityOverrideBottom() const; // see example below (added hsbCAD17.1.2)
void setSurfaceQualityOverrideBottom(String strVal); // see SurfaceQualityStyle for more
                                // information

```

*[Sample to illustrate the SurfaceQuality*

```

U(1, "mm");
if (\_bOnInsert) {

```

```

    _Pt0 = getPoint(); // select point
    _Entity.append(getMasterPanel());

    return;
}

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

PropString
pTop(2, SurfaceQualityStyle().getAllEntryNames(), "Surface Qual
Top");
PropString
pBot(3, SurfaceQualityStyle().getAllEntryNames(), "Surface Qual
Bot");

if (_Entity.length()==0) return;
MasterPanel mp = (MasterPanel)_Entity[0];
if (!mp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strSetOverrides= T("Set top and bottom overrides on Sip");
addRecalcTrigger(_kContext, strSetOverrides);
if (_bOnRecalc && _kExecuteKey==strSetOverrides) {
    mp.setSurfaceQualityOverrideTop(pTop);
    mp.setSurfaceQualityOverrideBottom(pBot);
}

CoordSys cs =mp.coordSys();
cs.vis();

MasterPanelStyle mpStyle(mp.style());

String strLines[0];
strLines.append("MasterPanel");
strLines.append("style: "+mp.style());
strLines.append("surfaceQualityOverrideTop:
"+mp.surfaceQualityOverrideTop());
strLines.append("surfaceQualityOverrideBottom:
"+mp.surfaceQualityOverrideBottom());
strLines.append("style.surfaceQualityTop:
"+mpStyle.surfaceQualityTop());
strLines.append("style.surfaceQualityBottom:
"+mpStyle.surfaceQualityBottom());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {

```

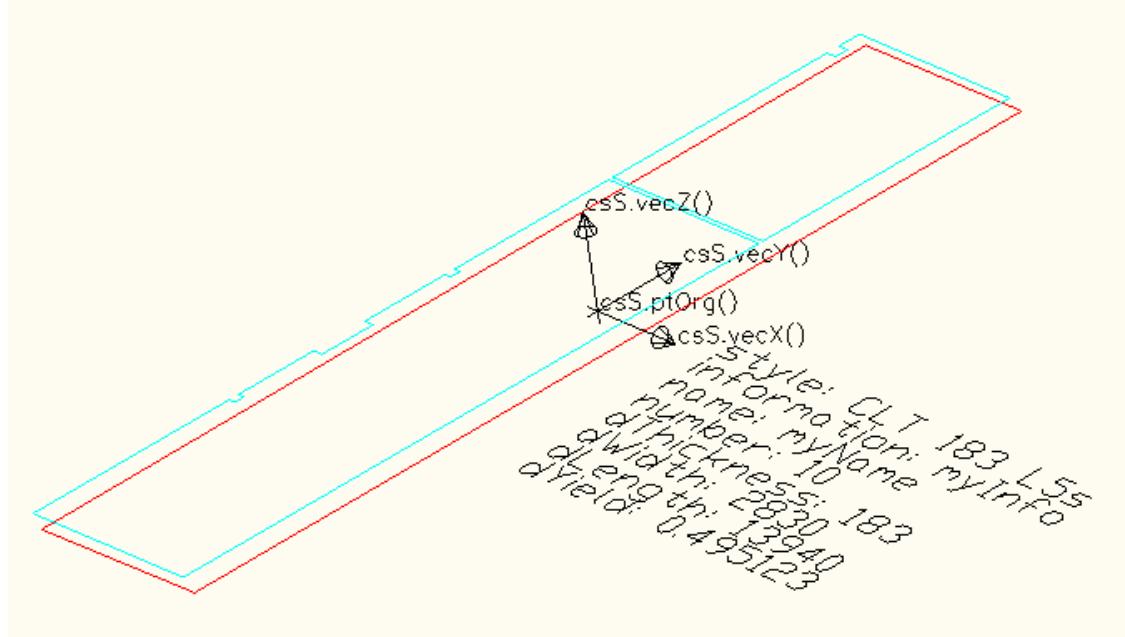
```

Vector3d vecO = -1*1.2*pTextHeight*_YU;
dp.draw(strLines[1],_Pt0+vecO,_XU,_YU, 1,1);
}

—end example]

```

[Example O-type, with insert done inside script:



```

U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    MasterPanel sl = getMasterPanel();
    Entity.append(sl);

    return;
}

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

if (_Entity.length()==0) return;
MasterPanel sp = (MasterPanel)_Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strChangeEntity = T("Change entity");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {

```

```

        sp.setNumber(10);
        sp.setInformation("myInfo");
        sp.setName("myName");
    }

CoordSys css =sp.coordSys();
css.vis();
Vector3d vecZ = css.vecZ();
Vector3d vecX = css.vecX();
PLine plShape = sp.plShape();
Point3d pnts[] = plShape.vertexPoints(TRUE);
plShape.transformBy(U(333)*vecZ);
plShape.vis(1);

String strChangeShape = T("Change plShape");
addRecalcTrigger(_kContext, strChangeShape );
if (_bOnRecalc && _kExecuteKey==strChangeShape ) {
    PLine plShapeNew(vecZ);
    Vector3d vecM = vecX;
    for (int p=0; p<pnts.length(); p++) {
        vecM = vecM.rotateBy(30, vecZ); // add an arbitrary vecMove
        plShapeNew.addVertex(pnts[p] + U(20)* vecM);
    }
    sp.setPlShape(plShapeNew);
    sp.updateYield();
}

String strUpdateYield = T("Update yield");
addRecalcTrigger(_kContext, strUpdateYield );
if (_bOnRecalc && _kExecuteKey==strUpdateYield ) {
    sp.updateYield();
}

ChildPanel arChld[] = sp.nestedChildPanels();
for (int c=0; c<arChld.length(); c++) {
    ChildPanel chp = arChld[c];
    PLine plCnc = chp.plEnvelopeCnc();
    plCnc.transformBy(U(555)*vecZ);
    plCnc.vis(4);
}

String strLines[0];
strLines.append("style: "+sp.style());
strLines.append("information: "+sp.information());
strLines.append("name: "+sp.name());
strLines.append("number: "+sp.number());
strLines.append("dThickness: "+sp.dThickness());
strLines.append("dWidth: "+sp.dWidth());
strLines.append("dLength: "+sp.dLength());
strLines.append("dYield: "+sp.dYield());

```

```

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

—end example]

[Example O-type, with insert done inside script to illustrate the dbCreate and openings

```

Unit(1, "mm");

String strAllStyles[] = MasterPanelStyle() .getAllEntryNames(); // list of all available MasterPanelStyles
PropString pStyle(0, strAllStyles, T("|MasterPanel style|"));

if (_bOnInsert)
{
    showDialog(); // allow the user to select the style

    EntPLine plent = getEntPLine();
    PLine pl = plent.getPLine();
    Point3d ptLoc = getPoint();

    CoordSys csEcs(ptLoc, _XW, _YW, _ZW);

    MasterPanel mp;
    mp.dbCreate(pStyle, csEcs, pl);
    mp.setPtRef(ptLoc);

    _Entity.append(mp);
    _Pt0 = mp.ptRef() + U(300) * _XU;

    return;
}

PropInt pIndex(0, 0, T("|Index|"));
pStyle.setReadOnly(TRUE);

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}

MasterPanel mp = (MasterPanel)_Entity[0];
if (!mp.bIsValid())
```

```

{
    eraseInstance();
    return;
}

setDependencyOnEntity(mp);

String strChangeEntity = T("|set ptRef entity|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity)
{
    Point3d ptnew = getPoint();
    mp.setPtRef(ptnew);
}

String strAddOpening = T("|add opening|");
addRecalcTrigger(_kContext, strAddOpening );
if (_bOnRecalc && _kExecuteKey==strAddOpening)
{
    EntPLine plent = getEntPLine();
    PLine pl = plent.getPLine();
    mp.appendOpening(pl);
}

String strRemoveOpening = T("|remove opening at|");
addRecalcTrigger(_kContext, strRemoveOpening );
if (_bOnRecalc && _kExecuteKey==strRemoveOpening)
{
    mp.removeOpeningAt(pIndex);
}

Vector3d vecMove = _Pt0 - mp.ptRef();

Display dp(-1);
PLine pl = mp.plShape();
pl.transformBy(vecMove);
dp.draw(pl);

for(int o=0; o<mp.openingCount(); o++)
{
    PLine pl = mp.plOpeningAt(o);
    pl.transformBy(vecMove);
    dp.color(o==pIndex ? 1 : 3);
    dp.draw(pl);
}

```

*—end example]*

## 7.34 MetalPartCollectionEnt

The term MetalPartCollectionEnt refers to an instance of a entity that refers to a [MetalPartCollectionDef](#).

An entity can be casted into a MetalPartCollectionEnt. However when the casting is not allowed, the resulting MetalPartCollectionEnt will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because MetalPartCollectionEnt is derived from [CollectionEntity](#), it inherits all the member functions of CollectionEntity as well. It can also be casted to an CollectionEntity. During `_bOnInsert`, the `getMetalPartCollectionEnt()` function can be used to query the user to select a MetalPartCollectionEnt instance.

```
MetalPartCollectionEnt getMetalPartCollectionEnt();
MetalPartCollectionEnt getMetalPartCollectionEnt(String strPrompt);
```

---

```
class MetalPartCollectionEnt : CollectionEntity { // see CollectionEntity base functions
    (added hsbCAD2010 build 15.0.3)

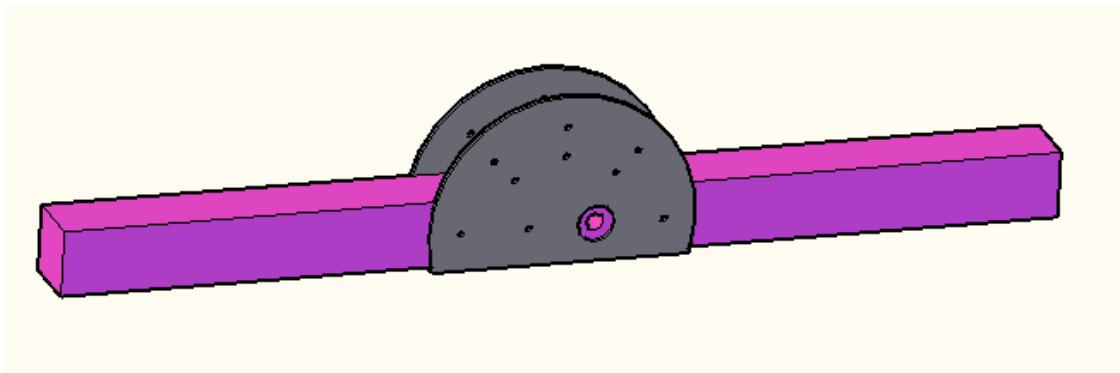
    String definition() const;
    void setDefinition(String strVal); // see MetalPartCollectionDef for more information

    MetalPartCollectionDef definitionObject() const; // (added hsbCAD18.1.24)
    void setDefinitionObject(MetalPartCollectionDef val); // (added hsbCAD18.1.24)

    void dbCreate(CoordSys csEcs, String strDefinition); // (added hsbCAD20.0.68 and
        hsbCAD19.1.99)
};
```

---

*[Example O-type, with insert done inside script:*



```

(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    MetalPartCollectionEnt ce = getMetalPartCollectionEnt();
    _Entity.append(ce);

    return;
}

PropString pDimStyle(1,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

if (_Entity.length()==0) return;
MetalPartCollectionEnt ce = (MetalPartCollectionEnt)_Entity[0];
if (!ce.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

CoordSys cs =ce.coordSys();
cs.vis();

MetalPartCollectionDef cd = ce.definition();
Beam arBm[] = cd.beam(); // find beams of collection definition
for (int e=0; e<arBm.length(); e++) {
    Beam bm = arBm[e];
    Body bd = bm.realBody();
    bd.transformBy(cs); // transform to entity coordsys
    bd.transformBy(U(500)*cs.vecX()); // additional move out
    bd.vis(1);
}

String strLines[0];
strLines.append("MetalPartCollectionEnt");
strLines.append("definition: "+ce.definition());

// display the lines
Display dp(-1);

```

```

dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

```

*—end example]*

## 7.35 MultiPage

The term MultiPage refers to an instance of a showdrawing entity used in model space.

An entity can be casted into a MultiPage. However when the casting is not allowed, the resulting MultiPage will become invalid. This can be checked with the `blsValid()` function of Entity.

Because MultiPage is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getMultiPage()` function can be used to query the user to select a MultiPage instance.

```

MultiPage getMultiPage();
MultiPage getMultiPage(String strPrompt);

```

```

class MultiPage : Entity { // see Entity for base functions (added 23.6.3 and 24.0)

    CoordSys coordSys() const;

    String style() const;
    void setStyle(String strVal); // see MultiPageStyle for more information

    Entity[] defineSet() const; // Set of entities that defines the showset eg the element for an
                               // element set, the subassembly tsl instance for a subassembly,..
    void setDefineSet(<Entity>[] arEntities); // Sets the define set
    void setDefineSet(<Entity>[] arEntities, int nObjectCollectionType); // Sets the define set, as
                           // well as the objectCollectionType (see MultiPageStyle)

    Entity[] showSet() const; // Set of entities that represent the showset eg the beams and
                           // sheet for an element, the entities of a subassembly tsl,..

    String regenerate(); // triggers regeneration of the MultiPage

    void dbCreate(CoordSys csLocation, String style, <Entity>[] arEntities); // added V23.8.55
                           // and V24.1.41
    void dbCreate(CoordSys csLocation, String style, <Entity>[] arEntities, int
                  nObjectCollectionType); // added V23.8.55 and V24.1.41

    MultiPageView[] views() const; // see MultiPageView
    String[] blocks() const; // see Block, added V24.1.87 and V25.1.53

};

```

[Example O-type, with insert done inside script:

```

U(1, "mm");
if (_bOnInsert)
{
    _Pt0 = getPoint(); //select point
    MultiPage sl = getMultiPage();
    _Entity.append(sl);

    return;
}

if (_Entity.length() == 0) return;

String strAllDefinitions[] = MultiPageStyle().getAllEntryNames(); // list of all available
PropString pDefinition(0, strAllDefinitions, T("|MultiPage styles|")); // make property

PropString pDimStyle(1, _DimStyles, T("|Dim style|"));
PropDouble pTextHeight(0, U(20), T("|Text height|"));

MultiPage mp = (MultiPage)_Entity[0];
if (!mp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strSetStyle = T("|Change style|");
addRecalcTrigger(_kContext, strSetStyle);
if (_bOnRecalc && _kExecuteKey==strSetStyle)
{
    pDefinition.setReadOnly(FALSE);
    showDialog();
    mp.setStyle(pDefinition);
}

String strRegenerate = T("|Regenerate|");
addRecalcTrigger(_kContext, strRegenerate);
if (_bOnRecalc && _kExecuteKey==strRegenerate)
{
    mp.regenerate();
}

String strSetDefineSet = T("|Set define set|");
addRecalcTrigger(_kContext, strSetDefineSet);
if (_bOnRecalc && _kExecuteKey==strSetDefineSet)
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
}
}

```

```

        mp.setDefineSet(ssE.set());
        mp.regenerate();
    }

}

pDefinition.set(mp.style());
pDefinition.setReadOnly(TRUE);

CoordSys css =mp.coordSys();
css.vis();
Vector3d vecZ = css.vecZ();
Vector3d vecX = css.vecX();

Entity showSet[] = mp.showSet();
Entity defineSet[] = mp.defineSet();

String strLines[0];
strLines.append("style: "+mp.style());
strLines.append("number of entities in showset: "+showSet.length());
strLines.append("number of entities in defineSet:
"+defineSet.length());
for (int e = 0; e < defineSet.length(); e++)
{
    Entity ent = defineSet[e];
    strLines.append("    entity: "+ent.typeName());
}
String strBlocks[] = mp.blocks();
strLines.append("number of blocks: "+strBlocks.length());
for (int b = 0; b < strBlocks.length(); b++)
{
    String block = strBlocks[b];
    strLines.append("    block: "+block);
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

String arDispReps[] = mp.dispRepNames();
String strDepsRep = "Model";
dp.color(3);
for (int e = 0; e < defineSet.length(); e++)
{
    Entity ent = defineSet[e];
    dp.draw(ent, CoordSys(), strDepsRep);
}
dp.color(1);
for (int e = 0; e < showSet.length(); e++)

```

```

{
    Entity ent = showSet[e];
    dp.draw(ent, CoordSys(), strDepsRep);
}

—end example]

```

### 7.35.1 MultiPageView

The term MultiPage refers to an instance of a showdrawing entity used in model space. The MultiPageView refers to one area in the [MultiPage](#) that displays a showSet in one direction. It is the counter part of the [ShopDrawView](#), which is part of the shopdraw block definition.

The MultiPageView's are retrieved from the [MultiPage](#) with the method views.

---

```

class MultiPageView // see also MultiPage (added 23.6.10 and 24.0)
{
    PLine plShape() const; // Shape of the viewport, located in space. Contains view in page,
                           // page in multipage, and multipage in world transformations.
    PLine plShape(int bIncludeAdditions) const; // If the bIncludeAdditions is TRUE (default
                                                 // FALSE) the additional width and height are added.
    CoordSys modelToView() const; // Combined transformation from model to view. Contains
                                  // view orientation, scaling, view in page, page in multipage, and multipage in
                                  // world transformations.
    double viewScale() const; // the viewScale of the view
    Plane snapPlaneInModel() const; // the plane that is mapped to worldXY to measure
                                    // dimensions

    Entity[] showSet() const; // the entity set shown in the view

    MultiPage multiPage() const; // the multipage to which this view belongs
    int pageIndex() const; // the index of the page in the multipage to which the view
                          // belongs

    String displaySetName() const; // the display set in use to extract solids from showSet

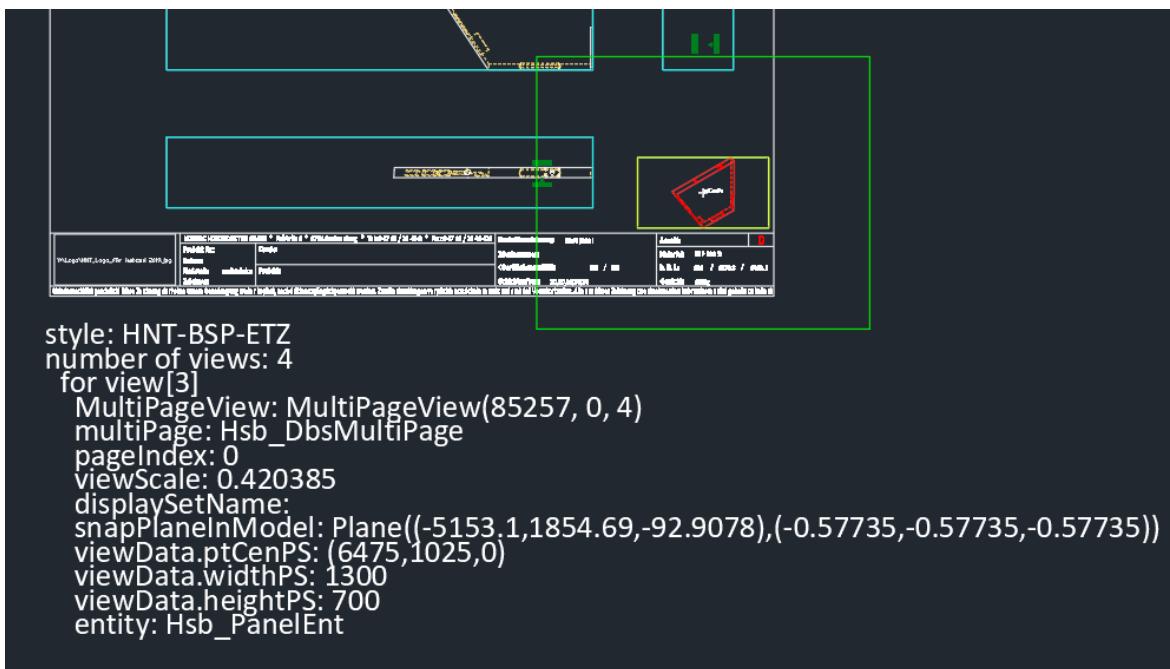
    ViewData viewData() const; // ViewData similar to ShopDrawView

    PlaneProfile[] dimCollisionAreas(int bTextOnly) const; // from the ruleset generated
                                                          // dimensions, return the extents in the form of PlaneProfiles.
}

```

---

*[Example O-type, with insert done inside script:*



```

U(1, "mm");
if (_bOnInsert)
{
    _Pt0 = getPoint(); //select point
    MultiPage sl = getMultiPage();
    _Entity.append(sl);

    return;
}

if (_Entity.length() == 0) return;

PropString pDimStyle(1, _DimStyles, T("|Dim style|"));
PropDouble pTextHeight(0, U(20), T("|Text height|"));
PropInt pViewIndex(0, 0, T("|View index to investigate|"));
PropInt pOnlyText(1, 0, T("|Show collision area of text only|"));

MultiPage mp = (MultiPage)_Entity[0];
if (!mp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);

MultiPageView arView[] = mp.views();

int bDisplayShapes = TRUE;
if (bDisplayShapes)

```

```

{
    dp.color(4); // blue
    for (int e = 0; e < arView.length(); e++)
    {
        MultiPageView mpv = arView[e];
        PLine plShape = mpv.plShape();
        dp.draw(plShape);
    }
}

String strLines[0];
strLines.append("style: "+mp.style());

int nViewIndex = pViewIndex;
strLines.append("number of views: "+arView.length());
strLines.append(" for view["+nViewIndex+"]");
if (nViewIndex>=0 && nViewIndex<arView.length() > 0)
{
    MultiPageView mpv = arView[nViewIndex];
    strLines.append("    MultiPageView: "+mpv);
    strLines.append("    multiPage: "+mpv.multiPage().typeName());
    strLines.append("    pageIndex: "+mpv.pageIndex());
    strLines.append("    viewScale: "+mpv.viewScale());
    strLines.append("    displaySetName: "+mpv.displaySetName());
    strLines.append("    snapPlaneInModel: "+mpv.snapPlaneInModel());
}

 ViewData viewDat = mpv.viewData();
strLines.append("    viewData.ptCenPS: "+viewDat.ptCenPS());
strLines.append("    viewData.widthPS: "+viewDat.widthPS());
strLines.append("    viewData.heightPS: "+viewDat.heightPS());

Point3d ptCenPs = viewDat.ptCenPS();
ptCenPs.transformBy(mp.coordSys());
ptCenPs.vis();

Entity showSetView[] = mpv.showSet();
for (int e = 0; e < showSetView.length(); e++)
{
    Entity ent = showSetView[e];
    strLines.append("    entity: "+ent.typeName());

    Body bdEnt = ent.realBody();
    bdEnt.transformBy(mpv.modelToView());

    dp.color(1); // red
    dp.draw(bdEnt);
}

PLine plShape0 = mpv.plShape();
dp.color(2); // yellow
dp.draw(plShape0);

PLine plShape1 = mpv.plShape(TRUE);

```

```

dp.color(3); // green
dp.draw(plShape1);

PlaneProfile collArea[] = mpv.dimCollisionAreas(pOnlyText);
for (int e = 0; e < collArea.length(); e++)
{
    dp.color(1+e);
    PlaneProfile prf = collArea[e];
    dp.draw(prf);
}
strLines.append("    collArea.length: "+collArea.length());

}

// display the lines
dp.color(-1);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

—end example]

```

## 7.36 MvBlockRef

The term MvBlockRef refers to an instance of a multi view block reference.

An entity can be casted into a MvBlockRef. However when the casting is not allowed, the resulting MvBlockRef will become invalid. This can be checked with the blsValid() function of Entity.

Because MvBlockRef is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During [\\_bOnInsert](#), the [getMvBlockRef\(\)](#) function can be used to query the user to select an instance.

```

MvBlockRef getMvBlockRef();
MvBlockRef getMvBlockRef(String strPrompt);

```

---

```

class MvBlockRef : Entity { // see Entity for base functions (added hsbCAD2011 build 16.0.21)

    void dbCreate(CoordSys csEcs, double dScaleX, double dScaleY, double dScaleZ, String
        strDefinition); // (added hsbCAD2011 build 16.0.42)

    CoordSys coordSys() const;
    void setCoordSys(CoordSys csNewCoordSys); // set the reference point and X, Y and Z
        vectors of MvBlockRef. (added hsbCAD2011 build 16.0.42).
}

```

---

```

double dScaleX() const;
double dScaleY() const;
double dScaleZ() const;
void setScale(double dScaleX, double dScaleY, double dScaleZ); // (added hsbCAD2011
build 16.0.42)

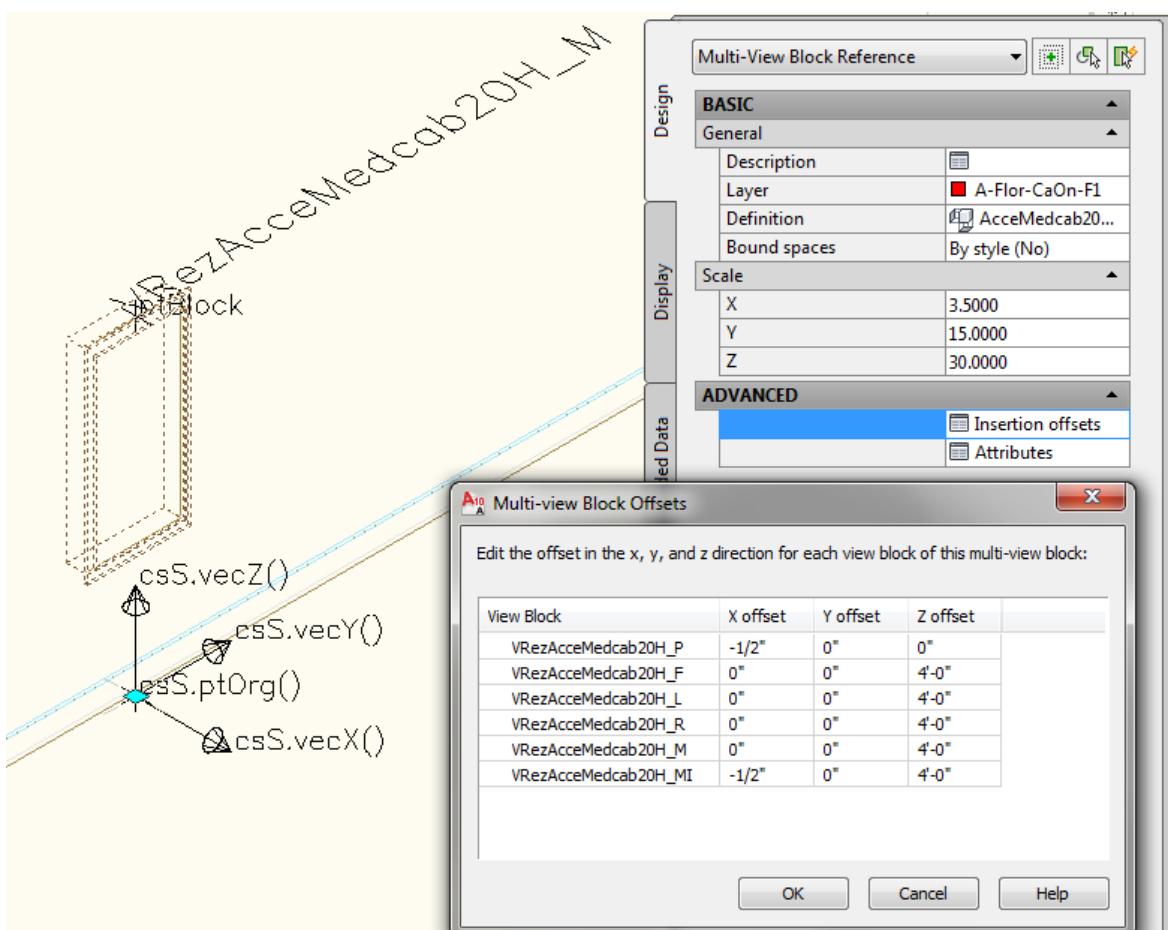
String description() const;
void setDescription(String strVal);

String definition() const; // return name of MvBlockDef
void setDefinition(String strVal); // see MvBlockDef for more information

Vector3d vecViewBlockOffset(String strBlock) const; // vector to be added to ptOrg of
coordSys to find world point

};

```



**CoordSys** coordSys() const;

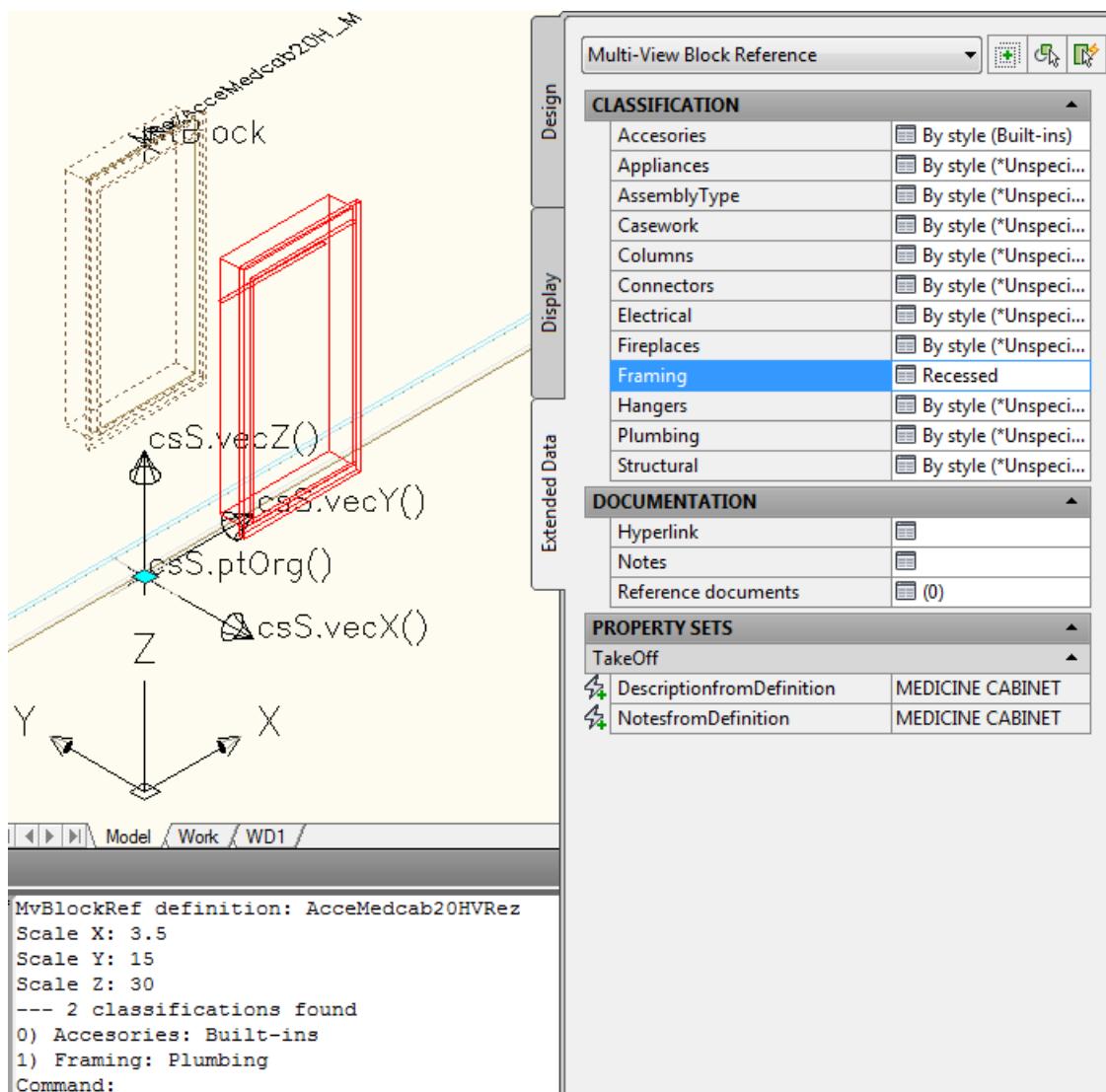
Build the coordinate system of the MvBlockRef. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

**Vector3d** vecViewBlockOffset(**String** strBlock) const;

For each block in the multi view block there can be an offset set. The list of available blocks can be retrieved from the [MvBlockDef::getAllBlocksReferenced](#). The returned vector can be added to the ptOrg of the coordSys to find the 3d point. To calculate the offset components relative to the coordinate system, one needs to project (dotProduct) the vector on the corresponding vec from the coordSys: dOffsetX = coordSys().vecX().dotProduct(vector).

---

*[Example O-type, with insert done inside script:*



```
U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    MvBlockRef mvBlockRef = getMvBlockRef();
    _Entity.append(mvBlockRef);

    return;
}

// check entity link
MvBlockRef mvBlockRef;
if (_Entity.length()>0) mvBlockRef = (MvBlockRef)_Entity[0];
if (!mvBlockRef.bIsValid()) {
    eraseInstance();
    return;
}
```

```

CoordSys css = mvBlockRef.coordSys();
css.vis();
Point3d ptO = css.ptOrg();
Vector3d vecX = css.vecX();
Vector3d vecY = css.vecY();

reportMessage("\nMvBlockRef definition: " + mvBlockRef.definition() );
reportMessage("\nScale X: " + mvBlockRef.dScaleX() );
reportMessage("\nScale Y: " + mvBlockRef.dScaleY() );
reportMessage("\nScale Z: " + mvBlockRef.dScaleZ() );

MvBlockDef mvd(mvBlockRef.definition());
String arBlocks[] = mvd.getAllBlocksReferenced();

// find the Vrez model block, which ends with _M by convention
String strModelBlock;
for (int v=0; v<arBlocks.length(); v++) {
    String strBlock = arBlocks[v];
    if (strBlock.right(2)=="_M") {
        strModelBlock = strBlock;
        break;
    }
}

// find the offset for the model block
if (strModelBlock.length()>0) { // model block was found
    Display dp(-1);
    dp.textHeight(U(40));
    Vector3d vecO = mvBlockRef.vecViewBlockOffset(strModelBlock);
    Point3d ptBlock = ptO + vecO;
    ptBlock.vis();
    dp.draw(strModelBlock,ptBlock,vecY,-vecX,1,1);
}

// If there is a solid inside the block that is associated with the
// model display, it will be collected with the realBody method
Body bdBlock = mvBlockRef.realBody();
bdBlock.transformBy(U(500)*vecX);
bdBlock.vis(1);

// Since the MvBlockRef is an ACA entity, there could be a
// classification associated with it.
int bAddClassificationFromStyle = TRUE;
Map mapEnt =
mvBlockRef.getClassificationMap(bAddClassificationFromStyle);

reportMessage("\n--- "+mapEnt.length()+" classifications found");
for (int i=0; i<mapEnt.length(); i++) {
    String strKey = mapEnt.keySet(i);
    if (mapEnt.containsKey(i))
}

```

```

        reportMessage ("\n"+i+" "+strKey+": "+mapEnt.getString(i));
    }

—end example]

```

## 7.37 NailLine

The type NailLine refers to an instance of the hsbCad Nailing line entity type. The NailLine is derived from Entity, so it is an entity in the autocad database/drawing. This is in contrast with the ElemNail, which is a tool that can be applied on an element. By the way, the NailLine entity internally also contains an ElemNail tool.

An entity can be casted into an NailLine. However when the casting is not allowed, the resulting NailLine will become invalid. This can be checked with the blsValid() function of Entity.

Because NailLine is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity.

During \_bOnInsert, the getNailLine() function can be used to query the user to select a NailLine instance.

```

NailLine getNailLine();
NailLine getNailLine(String strPrompt);

```

The nailing lines that belong to an element can be collected by calling the following routine:

```

NailLine[] Element::nailLine();
NailLine[] Element::nailLine(int nZoneIndex);

```

---

```

class NailLine : ToolEnt { // see ToolEnt for base functions

    int toolIndex() const;
    void setToolIndex(int nIndexNew);

    int zoneIndex() const;
    void setZoneIndex(int nIndexNew);

    double spacing() const;
    void setSpacing(double dSpacingNew);

    PLine plPath() const; // added hsbCAD build 14.0. line and setLine are deprecated
    void setPlPath(PLine plLineNew);

    void dbCreate(Element elToNail, ElemNail elemNailToCopy); // see ElemNail
    void dbCreate(Element elToNail, ElemNail elemNailToCopy, int bForceModelSpace); // added
        V23.7.12, with default bForceModelSpace TRUE

    static NailLine[] filterNailLinesCloseToSheetingEdge(NailLine[] arNail, Sheet[] arSheet,
        double dDist);

```

```

static LineSeg[] calculateAllowedNailLineSegments(GenBeam[] arBeam, Plane planeBeam,
double dTolDistPlaneBeam, double dShrinkDistBeam, GenBeam[] arSheet,
Plane planeSheet, double dTolDistPlaneSheet, double dShrinkDistSheet, int
bAllowSheetsToMerge, double dShrinkDistNailLine); //
bUseBeamCenterAsFirstGuess is FALSE
static LineSeg[] calculateAllowedNailLineSegments(GenBeam[] arBeam, Plane planeBeam,
double dTolDistPlaneBeam, double dShrinkDistBeam, GenBeam[] arSheet,
Plane planeSheet, double dTolDistPlaneSheet, double dShrinkDistSheet, int
bAllowSheetsToMerge, double dShrinkDistNailLine, int
bUseBeamCenterAsFirstGuess);

GenBeam[] filterGenBeamsBeingNailed(GenBeam[] arGenBeam, Plane plane, double
dToleranceDistance) const;
Beam[] filterGenBeamsBeingNailed(Beam[] arGenBeam, Plane plane, double
dToleranceDistance) const;
Sheet[] filterGenBeamsBeingNailed(Sheet[] arGenBeam, Plane plane, double
dToleranceDistance) const;
Sip[] filterGenBeamsBeingNailed(Sip[] arGenBeam, Plane plane, double dToleranceDistance)
const;

static Beam[] removeGenBeamsWithNoNailingBeamCode(Beam[] arBeam);
static Sheet[] removeGenBeamsWithNoNailingBeamCode(Sheet[] arBeam);
static Sip[] removeGenBeamsWithNoNailingBeamCode(Sip[] arBeam);
static GenBeam[] removeGenBeamsWithNoNailingBeamCode(GenBeam[] arBeam);
};


```

```

int zoneIndex() const;
void setZoneIndex(int nIndexNew);

```

The zoneIndex is the zone that the nailing is applied to. If you compare the value with the myZoneIndex of the Entity, you will see that they are different. The reason is that the entity itself is added to the autocad layer of zone 0, while the nailing is applied to zone zoneIndex. So always use the zoneIndex. The value is in the range -5,..., 5. The element can be queried from the base class Entity.

```

int toolIndex() const;
void setToolIndex(int nIndexNew);
double spacing() const;
void setSpacing(double dSpacingNew);
PLine plPath() const;
void setPPath(PLine plPathNew);

```

Set and get the properties of the NailLine entity.

```
void dbCreate(Element elToNail, EleaNail elemNailToCopy);
```

Create a new database resident instance, insert a new NailLine into the Autocad drawing. To construct the NailLine, an EleaNail tool is used, and an element that the tool will be active on.

```
static NailLine[] filterNailLinesCloseToSheetingEdge(NailLine[] arNail, Sheet[] arSheet, double dDist);
```

This routine will return a subset of the NailLines passed in as the first argument. It will return these NailLines of which the midpoint is closer than the distance dDist to a sheeting edge. The midpoint of the NailLine is taken as the ptMid() of the [PLine](#) returned by line(). To calculate the distance to a sheeting edge, the closest point to the profShape of the [Sheet](#) is taken. From this point the distance is calculated to the midpoint, but projected in the plane of the sheeting. The routine is declared static. This means that you do not need a valid instance in order to call the routine. In the example below, the routine is called on a NailLine() instance.

```
static LineSeg[] calculateAllowedNailLineSegments(GenBeam[] arBeam, Plane planeBeam, double dTolDistPlaneBeam, double dShrinkDistBeam, GenBeam[] arSheet, Plane planeSheet, double dTolDistPlaneSheet, double dShrinkDistSheet, int bAllowSheetsToMerge, double dShrinkDistNailLine); // bUseBeamCenterAsFirstGuess is FALSE
static LineSeg[] calculateAllowedNailLineSegments(GenBeam[] arBeam, Plane planeBeam, double dTolDistPlaneBeam, double dShrinkDistBeam, GenBeam[] arSheet, Plane planeSheet, double dTolDistPlaneSheet, double dShrinkDistSheet, int bAllowSheetsToMerge, double dShrinkDistNailLine, int bUseBeamCenterAsFirstGuess);
```

This routine returns an array of LineSeg. With each line segment a NailLine can be made. The set of line segments is composed by performing the following steps.

- First the PlaneProfile is extracted from each GenBeam from the array arBeam. For doing this, the planeBeam and dTolDistPlaneBeam are used in the same way as the arguments of the routine [Body::extractContactFacelnPlane](#). The surface of the body is chosen as the surface closest to the given plane. Also the distance of the surface to the plane must be smaller than dTolDistPlaneBeam.
- Each PlaneProfile of the beam is shrunk by dShrinkDistBeam.
- The same is done with the GenBeams of the arSheet array. First extract the contact PlaneProfile using the planeSheet, and the dTolDistPlaneSheet, then shrink the PlaneProfile with dShrinkDistSheet.
- If the bAllowSheetsToMerge, the PlaneProfiles coming from the arSheet are merged together.
- In the main axis direction of each of the GenBeams of arBeam, and somewhere inside the PlaneProfile of each arBeam a set of LineSeg is drawn that are also inside the PlaneProfile of the arSheet. Somewhere inside the PlaneProfile means first try the center point of the PlaneProfile in case the bUseBeamCenterAsFirstGuess is set to FALSE. If the bUseBeamCenterAsFirstGuess is set to TRUE, the centerline on the beam is tried as first guess. If not successful because the line segment or parts of it, lie outside the PlaneProfile, then try at the beam edge but offsetted with the dShringDistBeam.
- The LineSeg that are returned are finally shortened on each side by the dShrinkDistNailLine. The routine is declared static. This means that you do not need a valid instance in order to call the routine. In the example below, the routine is called on a NailLine() instance.

```
GenBeam[] filterGenBeamsBeingNailed(GenBeam[] arGenBeam, Plane plane, double dToleranceDistance) const;
Beam[] filterGenBeamsBeingNailed(Beam[] arGenBeam, Plane plane, double dToleranceDistance) const;
```

```
Sheet[] filterGenBeamsBeingNailed(Sheet[] arGenBeam, Plane plane, double dToleranceDistance) const;
Sip[] filterGenBeamsBeingNailed(Sip[] arGenBeam, Plane plane, double dToleranceDistance) const;
```

This routine returns a subset of the array of GenBeams, Beams, Sheets or Sips, depending on the first argument. The GenBeams that are being nailed by this nailing line will be returned. To determine if a GenBeam is nailed by this nailing line, the position of the start, mid and end point of the nailing lines is checked against the PlaneProfile coming from the GenBeam. If at least one of the three points lies inside the PlaneProfile, the nailing line is nailing the GenBeam. To calculate the PlaneProfile of the GenBeam, the routine [Body::extractContactFacelnPlane](#) of the [GenBeam::realBody](#) is called. To call this extractContactFacelnPlane, the plane and dToleranceDistance is used. To determine if one of the points lies inside the PlaneProfile, the [PlaneProfile::pointInProfile](#) is checked for \_kPointInProfile.

```
static Beam[] removeGenBeamsWithNoNailingBeamCode(Beam[] arBeam);
```

This routine returns a subset of the array of Beams. The Beams that do not contain "NO" as fifth/ninth argument in their beamcode are kept. The routine is declared static. This means that you do not need a valid instance in order to call the routine.

*[Example O-type TSL that creates a new NailLine and changes some properties of it:*

```
Unit(1,"mm");
if (_bOnInsert) {
    Element el = getElement();
    _Pt0 = getPoint("Get first point");
    _PtG.append(getPoint("Get second point"));

    ElemNail nail(1,_Pt0,_PtG[0],U(50),0);
    NailLine nLine;
    nLine.dbCreate(el,nail);

    nLine.setSpacing( 2*nLine.spacing() ); // double spacing distance
    nLine.setToolIndex( nLine.toolIndex() + 1 ); // increase tooling index by 1
    PLine plLine = nLine.plPath(); // get the line of the NailLine
    plLine.vis(1);
    plLine.transformBy( Vector3d(U(100),0,0) ); // move the line 100 mm
    plLine.vis(2);
    nLine.setPlPath(plLine); // set a modified line

    eraseInstance();
    return;
}
```

*--end example]*

[Example O-type TSL that finds the NailLines of zone 1 and changes the spacing of them if their distance to a sheeting edge is closer than...:]

```
Unit(1,"mm");
if (_bOnInsert) {

    Element el = getElement();
    Sheet arSh[] = el.sheet(1); // sheet of zone 1
    NailLine arNl[] = el.nailLine(1); // all nailing lines nailing on zone 1

    // filter all nailing lines closer then 100mm to a sheeting edge
    NailLine arNIClose[] = NailLine().filterNailLinesCloseToSheetingEdge(arNl,arSh,U(100));

    // change the spacing of these filtered NailLines
    for (int i=0; i<arNIClose.length(); i++) {
        arNIClose[i].setSpacing(U(200));
    }

    eraseInstance(); // this TSL instance will not stay in the drawing
    return;
}
```

—end example]

[Example O-type TSL that will automatic nail sheeting of zone 1]

```
Unit(1,"mm");
int nColorIndex = 4; // cyan

if (_bOnInsert) {

    int nArZone[]={-5,-4,-3,-2,-1,1,2,3,4,5};
    PropInt nZone(0,nArZone,T("Sheeting zone to nail"),5); // default is zone 1
    PropDouble dDist(0,U(100,4),T("Distance between nails")); // default to 100 mm or 4 inch

    // request the user for input properties
    showDialog();

    Element el = getElement();
    Beam beams[] = NailLine().removeGenBeamsWithNoNailingBeamCode(el.beam());
    Sheet sheets[] = el.sheet(nZone);

    // remove all nailing lines of nZone with color nColorIndex
    NailLine nlOld[] = el.nailLine(nZone);
    for (int n=0; n<nlOld.length(); n++) {
        NailLine nl = nlOld[n];
        if (nl.color()==nColorIndex) nl.dbErase();
    }

    int nSide = 1;
    if (nZone<0) nSide = -1;
    // get coordSys of the back of zone 1 or -1, the surface of the beams
    CoordSys csBeam = el.zone(nSide).coordSys();
```

```

// get the coordSys of the back of the zone to nail
CoordSys csSheet = el.zone(nZone).coordSys();

Plane planeBeam(csBeam.ptOrg(),csBeam.vecZ());
double dTolDistPlaneBeam = U(3);
double dShrinkDistBeam = U(10);

Plane planeSheet(csSheet.ptOrg(),csSheet.vecZ());
double dTolDistPlaneSheet = U(3);
double dShrinkDistSheet = U(10);

int bAllowSheetsToMerge = FALSE;
double dShrinkDistNailLine = U(5);

// calculate the nailing lines
LineSeg arSeg[] = NailLine().calculateAllowedNailLineSegments(
    beams, planeBeam, dTolDistPlaneBeam, dShrinkDistBeam,
    sheets, planeSheet, dTolDistPlaneSheet, dShrinkDistSheet,
    bAllowSheetsToMerge, dShrinkDistNailLine );

// now add nailing lines
for (int n=0; n<arSeg.length(); n++) {
    Point3d ptStart = arSeg[n].ptStart();
    Point3d ptEnd = arSeg[n].ptEnd();

    // make ElemNail tool to be used in the construction of a nailing line
    int nToolIndex = 0;
    ElemNail enl(nZone, ptStart, ptEnd, dDist, nToolIndex);

    // add the nailing line to the database
    NailLine nl; nl.dbCreate(el, enl);
    nl.setColor(nColorIndex); // set color of Nailing line
}

erasedInstance(); // this TSL will not be added to the Acad database
return;
}

```

*—end example]*

## 7.38 PlotViewport

The term PlotViewport refers to an instance of a plot viewport entity used in to map a [Layout](#) to model space.

An entity can be casted into a PlotViewport. However when the casting is not allowed, the resulting PlotViewport will become invalid. This can be checked with the `blsValid()` function of Entity.

Because PlotViewport is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getPlotViewport()` function can be used to query the user to select a PlotViewport instance.

```
PlotViewport getPlotViewport();
```

```
PlotViewport getPlotViewport(String strPrompt);
```

---

```
class PlotViewport : Entity { // see Entity for base functions (added 23.1.3)

    void dbCreate(String strLayout, Point3d ptLowerLeft); // see Layout

    CoordSys coordSys() const;

    String layout() const;
    void setLayout(String strVal); // see Layout for more information

    String name() const;
    void setName(String strVal);

    int posnum() const;
    void setPosnum(int nVal);

    double dY() const; // extents in vecY direction
    double dX() const; // extents in vecX direction
};
```

---

[Example O-type, with insert done inside script:

```
U(1, "mm");
if (_bOnInsert)
{
    _Pt0 = getPoint(); //select point
    PlotViewport sl = getPlotViewport();
    _Entity.append(sl);

    return;
}

if (_Entity.length()==0) return;

PropString pDimStyle(1, _DimStyles, T("|Dim style|"));
PropDouble pTextHeight(0, U(20), T("|Text height|"));

String strAllDefinitions[] = Layout().getAllEntryNames(); // list of
all available Layout
PropString pDefinition(0, strAllDefinitions, T("|Layouts|")); // make
property

String strCreateEntity = T("|Create entity|");
addRecalcTrigger(_kContext, strCreateEntity );
if (_bOnRecalc && _kExecuteKey==strCreateEntity)
{
    showDialog();
    Point3d ptLL = getPoint();
```

```

PlotViewport pvNew;
pvNew.dbCreate(pDefinition, ptLL);
_Entity[0] = pvNew;
}

PlotViewport sp = (PlotViewport)_Entity[0];
if (!sp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strChangeEntity = T("|Change entity|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    sp.setPosnum(10);
    sp.setName("myName");
}

pDefinition.set(sp.layout());
pDefinition.setReadOnly(TRUE);

CoordSys css =sp.coordSys();
css.vis();
Vector3d vecZ = css.vecZ();
Vector3d vecX = css.vecX();

String strLines[0];
strLines.append("style: "+sp.layout());
strLines.append("name: "+sp.name());
strLines.append("number: "+sp.posnum());
strLines.append("dx: "+sp.dx());
strLines.append("dy: "+sp.dy());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end example]*

## 7.39 PressEnt

The term PressEnt refers to an instance of a press which is attached to a [CurvedDescription](#) entity.

An entity can be casted into a PressEnt. However when the casting is not allowed, the resulting PressEnt will become invalid. This can be checked with the **bIsValid()** function of Entity.

Because PressEnt is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getPressEnt()` function can be used to query the user to select a PressEnt instance.

```
PressEnt getPressEnt();
PressEnt getPressEnt(String strPrompt);
```

---

```
class PressEnt : Entity { // see Entity for base functions (added hsbCAD 18.2.0)

    void dbCreate(CurvedDescription cdc, Point3d ptInsert); // see CurvedDescription

    Point3d ptInsert() const;
    Point3d ptBottom() const;
    Point3d ptTop() const;

    String label() const;
    void setLabel(String strVal);

    double curveParam() const;
    void setCurveParam(double dVal);

    double size() const;
    void setSize(double dVal);

    double offset() const;
    void setOffset(double dVal);

    CurvedDescription curvedDescription() const; // see CurvedDescription
};


```

---

[Example O-type, with insert done inside script:

```
U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    PressEnt cp = getPressEnt();
    Entity.append(cp);

    return;
}

PropString pDimStyle(1, DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");
```

---

```

if (_Entity.length()==0) return;
PressEnt cp = (PressEnt )_Entity[0];
if (!cp.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strChangeEntity = T("Change entity");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    cp.setLabel(cp.label()+"A");
    cp.setCurveParam(cp.curveParam()+U(10));
    cp.setOffset(cp.offset()+U(10));
    cp.setSize(cp.size()*2);
}

Point3d ptI = cp.ptInsert();
ptI.vis();
Point3d ptB = cp.ptBottom();
ptB.vis();
Point3d ptT = cp.ptTop();
ptT.vis();

String strLines[0];
strLines.append("PressEnt ");
strLines.append("label: "+cp.label());
strLines.append("curveParam: "+cp.curveParam());
strLines.append("offset: "+cp.offset());
strLines.append("size: "+cp.size());
strLines.append("curvedDescription:
"+cp.curvedDescription().style());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example]*

*[Example O-type, with insert done inside script to illustrate the dbCreate*

```

Unit(1,"mm");
if (_bOnInsert) {

CurvedDescription cd = getCurvedDescription();
PressEnt cp;

```

```

Point3d ptLoc = getPoint();
cp.dbCreate(cd, ptLoc);

return;
}

—end example]

```

## 7.40 SawLine

Similar to [NailLine](#), there is the type SawLine which refers to an instance of the hsbCad Sawing line entity type. The SawLine is derived from Entity, so it is an entity in the autocad database/drawing. This is in contrast with the [ElemSaw](#), which is a tool that can be applied on an element. By the way, the SawLine entity internally also contains an ElemSaw tool.

An entity can be casted into an SawLine. However when the casting is not allowed, the resulting SawLine will become invalid. This can be checked with the **blsValid()** function of Entity.

Because SawLine is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity.

During **\_bOnInsert**, the **getSawLine()** function can be used to query the user to select a SawLine instance.

```

SawLine getSawLine();
SawLine getSawLine(String strPrompt);

```

The sawing lines that belong to an element can be collected by calling the following routine:

```

SawLine[] Element::sawLine();
SawLine[] Element::sawLine(int nZoneIndex);

```

---

```

class SawLine : ToolEnt // (added since 23.7.12 and 24.0) see ToolEnt for base functions
{
    ElemSaw elemTool() const; // returns a copy of the internal ElemSaw
    void setElemTool(ElemSaw tool); // sets the properties of the internal tool

    int toolIndex() const;
    void setToolIndex(int nIndexNew);

    int zoneIndex() const;
    void setZoneIndex(int nIndexNew);

    PLine plPath() const;
    void setPILPath(PLine plLineNew);

    void dbCreate(Element elToSaw, ElemSaw elemSawToCopy); // see ElemSaw
    void dbCreate(Element elToSaw, ElemSaw elemSawToCopy, int bForceModelSpace); // default bForceModelSpace TRUE
};

```

---

---

```
ElemSaw elemTool() const; // see ElemSaw
void setElemTool(ElemSaw tool);
```

Get a copy of the internal tool, or set the values to the internal tool.

```
int zoneIndex() const;
void setZoneIndex(int nIndexNew);
```

The zoneIndex is the zone that the sawing is applied to. If you compare the value with the myZoneIndex of the Entity, you will see that they are different. The reason is that the entity itself is added to the autocad layer of zone 0, while the sawing is applied to zone zoneIndex. So always use the zoneIndex. The value is in the range -5,..., 5. The element can be queried from the base class Entity.

```
int toolIndex() const;
void setToolIndex(int nIndexNew);
PLine plPath() const;
void setPILPath(PLine plLineNew);
```

Set and get the properties of the SawLine entity.

```
void dbCreate(Element elToSaw, ElemSaw elemSawToCopy);
```

Create a new database resident instance, insert a new SawLine into the Autocad drawing. To construct the SawLine, an ElemSaw tool is used, and an element that the tool will be active on.

---

*[Example O-type TSL that creates a new SawLine and changes some properties of it:*

```
Unit(1, "mm");

PropInt nZoneIndex(0, 1, T("|Zone index|"));
PropDouble dDepth(0, U(10), T("|Depth|"));
PropInt nToolingIndex(1, 0, T("|Tooling index|"));

String arSSide[] = {T("|Left|"), T("|Right|")};
int arNSide[] = {_kLeft, _kRight};
PropString strSide(0, arSSide, T("|Side|"));

String arSTurn[] = {T("|Against course|"), T("|With course|")};
int arNTurn[] = {_kTurnAgainstCourse, _kTurnWithCourse};
PropString strTurn(1, arSTurn, T("|Turning direction|"));

String arSOShoot[] = {T("|No|"), T("|Yes|")};
int arNOShoot[] = {_kNo, _kYes};
```

```

PropString strOShoot(2,arSOShoot,T("|Overshoot|"));

PropDouble dAngle(1,30,T("|Angle|"));
dAngle.setFormat(_kAngle);

if (_bOnInsert) {

    showDialogOnce();
    int nSide = arNSide[arSSide.find(strSide,0)];
    int nTurn = arNTurn[arSTurn.find(strTurn,0)];
    int nOShoot = arNOShoot[arSOShoot.find(strOShoot,0)];

    Element el = getElement();
    _Pt0 = getPoint(T("|Get first point|"));
    _PtG.append(getPoint(T("|Get second point|")));
    _PtG.append(getPoint(T("|Get third point|")));

    double dRadius = U(1000,"mm");
    Vector3d vecN = (nZoneIndex > 0) ? el.vecZ() : -el.vecZ();
    PLine pline(vecN); // define PLine normal
    pline.addVertex(_Pt0); // add first point
    pline.addVertex(_PtG[0],dRadius,_kCWise);
    pline.addVertex(_PtG[1],dRadius,_kCCWise);

    ELEM_SAW
    tool(nZoneIndex,pline,dDepth,nToolingIndex,nSide,nTurn,nOShoot);
    tool.setAngle(dAngle);

    SawLine ent;
    ent.dbCreate(el, tool);

    _Entity.append(ent);

    return;
}

String strChangeEntity = T("|Change entity|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey == strChangeEntity)
{
    SawLine tool = getSawLine();
    if (_Entity.length()!=0)
        _Entity.removeAt(0);
    _Entity.append(tool);
}

if (_Entity.length()==0) return;
SawLine ent = (SawLine)_Entity[0];
if (!ent.bIsValid()) return;

PLine plLine = ent.plPath();
plLine.vis(1);

```

```

Point3d ptMid = plLine.ptMid(); ptMid.vis();
_Pt0 = ptMid;

Point3d ptStart = plLine.ptStart(); ptStart.vis();
Point3d ptEnd = plLine.ptEnd(); ptEnd.vis();

setDependencyOnEntity(ent);

// update my properties from the ElemSaw
ElemSaw tool = ent.elemTool();
nZoneIndex.set(tool.zoneIndex());
dDepth.set(tool.depth());
nToolingIndex.set(tool.toolIndex());
strSide.set(tool.sideIsLeft() ? arSSide[0] : arSSide[1]);
strTurn.set(tool.turningDirectionWith() == _kTurnAgainstCourse ?
arSTurn[0] : arSTurn[1]);
strOShoot.set(tool.overShoot() == _kNo ? arSOShoot[0] :
arSOShoot[1]);
dAngle.set(tool.angle());

String strChangeProps = T("|Change properties|");
addRecalcTrigger(_kContext, strChangeProps);
if (_bOnRecalc && _kExecuteKey==strChangeProps)
{
    showDialog();

    int nSide = arNSide[arSSide.find(strSide,0)];
    int nTurn = arNTurn[arSTurn.find(strTurn,0)];
    int nOShoot = arNOShoot[arSOShoot.find(strOShoot,0)];
    tool.setToolIndex(nToolingIndex);
    tool.setZoneIndex(nZoneIndex);
    tool.setDepth(dDepth);
    if (nSide == _kLeft)
        tool.setSideToLeft();
    else
        tool.setSideToRight();
    tool.setTurningDirectionWith(nTurn);
    tool.setOverShoot(nOShoot);
    tool.setAngle(dAngle);

    ent.setElemTool(tool);
}

```

*—end example]*

## 7.41 ShopDrawView

The term ShopDrawView refers to an instance of a View entity used in the definition of a shopdraw page of a multipage.

An entity can be casted into a ShopDrawView. However when the casting is not allowed, the resulting ShopDrawView will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because ShopDrawView is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getShopDrawView()` function can be used to query the user to select a ShopDrawView instance.

```
ShopDrawView getShopDrawView();
ShopDrawView getShopDrawView(String strPrompt);
```

See also [Shopdraw ViewData](#).

---

```
class ShopDrawView : Entity { // see Entity for base functions (added hsbCAD13.2.86)
```

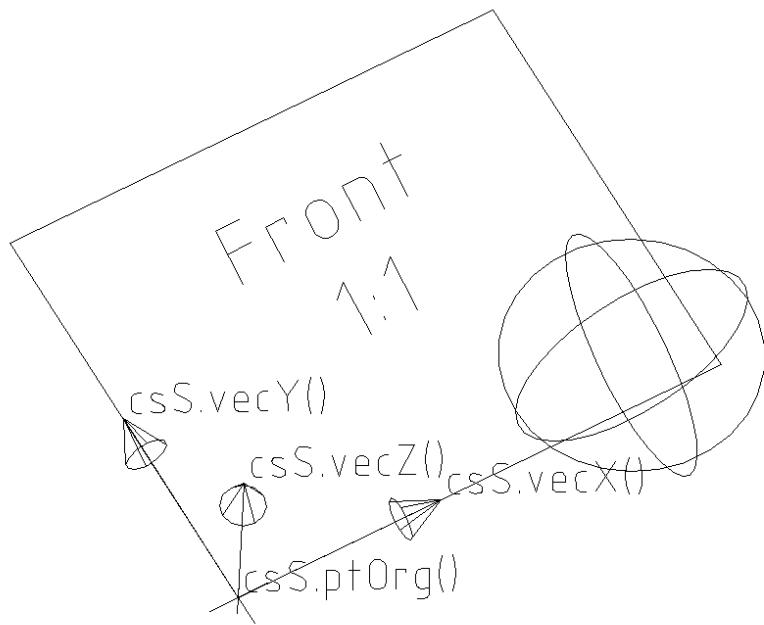
```
    CoordSys coordSys() const;
}
```

---

```
CoordSys coordSys() const;
```

Build the coordinate system of the ShopDrawView. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.

---



[Example O-type, with insert done inside script:

```

U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    ShopDrawView sl = getShopDrawView();
    _Entity.append(sl);

    return;
}

for (int i=0; i<_Entity.length(); i++) {
    ShopDrawView sl = (ShopDrawView)_Entity[i];
    CoordSys csS = sl.coordSys();
    csS.vis();

}

```

—end example]

## 7.42 Section2d

The term Section2d refers to an instance of a section or elevation, aka 2d Section/Elevation in the drawing.

An entity can be casted into a Section2d. However when the casting is not allowed, the resulting Section2d will become invalid. This can be checked with the **blsValid()** function of Entity.

Because Section2d is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During [\\_bOnInsert](#), the [getSection2d\(\)](#) function can be used to query the user to select a section clip volume instance.

```
Section2d getSection2d();
Section2d getSection2d(String strPrompt);
```

---

```
class Section2d : Entity { // see Entity for base functions (added hsbcad V21.4.36)

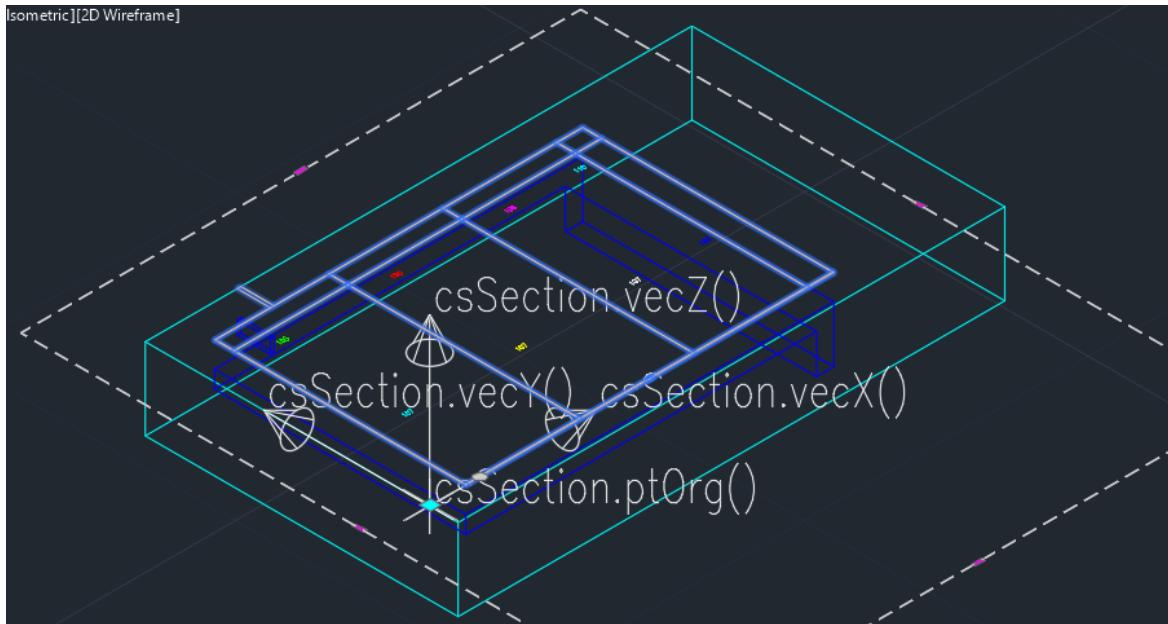
    CoordSys coordSys() const; // location of section in space
    CoordSys modelToSection() const; // transformation from ClipVolume to section

    ClipVolume clipVolume() const; // see ClipVolume

    Point3d posnumLocation(GenBeam genbeam, Vector3d vecViewFromDir) const; // genbeam
        posnum moved 1mm above the surface of the genbeam for a given viewFromDir
};
```

---

[Example O-type, with insert done inside script:



```
U(1, "mm");
if (\_bOnInsert) {

    \_Pt0 = getPoint(); // select point
    Section2d cv = getSection2d();
    Entity.append(cv);

    return;
```

```

}

PropString pDimStyle(0,_DimStyles ,T("|Dim style|"));
PropDouble pTextHeight(0,U(20),T("|Text height|"));

if (_Entity.length() == 0 || !((Section2d)_Entity[0]).bIsValid())
{
    eraseInstance();
    return;
}
Section2d section = (Section2d)_Entity[0];
ClipVolume clipvol = section.clipVolume();

setDependencyOnEntity(section);
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);

Vector3d vecView = clipvol.viewFromDir();
vecView.vis(clipvol.coordSys().ptOrg(), 1);
CoordSys csSection = section.coordSys();
csSection.vis();

CoordSys csToSection = section.modelToSection();
Body clipBody = clipvol.clippingBody();
clipBody.transformBy(csToSection);
dp.color(4);
dp.draw(clipBody);

Body combinedBody = clipvol.combinedClippedBodyOfEntities();
combinedBody.transformBy(csToSection);
dp.color(5);
dp.draw(combinedBody);

// draw the posnums of genbeams, similar to enhance section
Entity ents[] = clipvol.includedEntities();
for (int e = 0; e < ents.length(); e++)
{
    dp.color(1 + e%7);
    //Body bd = ents[e].realBody();
    //bd.transformBy(csToSection);
    //bd.intersectWith(clipBody);
    //dp.draw(bd);

    GenBeam genbeam = (GenBeam)ents[e];
    if (genbeam.bIsValid())
    {
        Point3d ptLoc = section.posnumLocation(genbeam, vecView);
        ptLoc.transformBy(csToSection);

        // if ptLoc is hidden by other geometry, do not show
        Point3d ptOnRay;
    }
}

```

```

int bIsHidden = combindedBody.rayIntersection(ptLoc, vecView,
ptOnRay);

if (!bIsHidden)
{
    String str = genbeam.posnum();
    dp.draw(str, ptLoc, csSection.vecX(), csSection.vecY(),
0, 0);
}
}
}

```

*—end example]*

## 7.43 Slab

The term Slab refers to an instance of an ADT slab.

An entity can be casted into a Slab. However when the casting is not allowed, the resulting Slab will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because Slab is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getSlab()` function can be used to query the user to select a slab instance.

```

Slab getSlab();
Slab getSlab(String strPrompt);

```

---

```
class Slab : Entity { // see Entity for base functions (added hsbCAD13.2.76)
```

```

CoordSys coordSys() const;

String description() const;
void setDescription(String strVal);

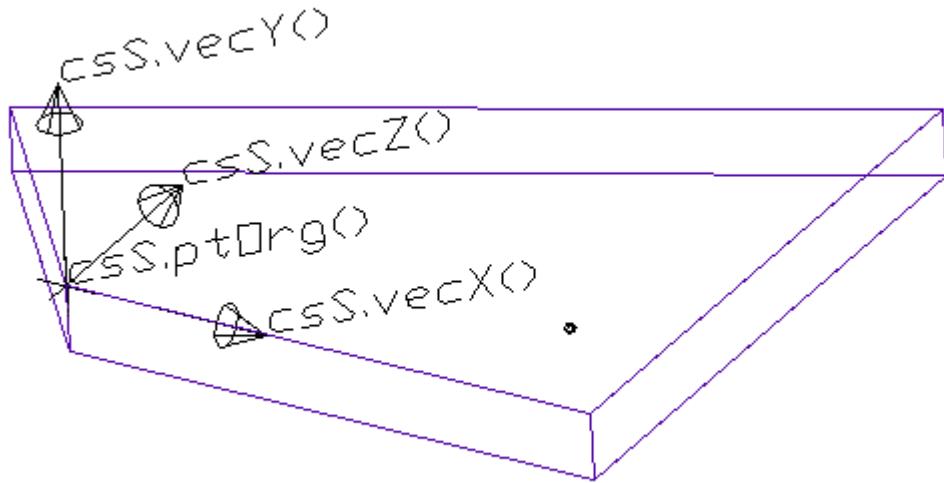
};

```

---

```
CoordSys coordSys() const;
```

Build the coordinate system of the slab. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.



[Example O-type, with insert done inside script:

```
U(1,"mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Slab sl = getSlab();
    _Entity.append(sl);

    return;
}

for (int i=0; i<_Entity.length(); i++) {
    Slab sl = (Slab)_Entity[i];
    CoordSys csS = sl.coordSys();
    csS.vis();

    reportNotice("\nSlab description = " +sl.description());
}
```

—end example]

## 7.44 TrussEntity

The term TrussEntity refers to an instance of a entity that refers to a [TrussDefinition](#).

An entity can be casted into a TrussEntity. However when the casting is not allowed, the resulting TrussEntity will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because TrussEntity is derived from [CollectionEntity](#). It inherits all the member functions of CollectionEntity as well. It can also be casted to an CollectionEntity. During `_bOnInsert`, the `getTrussEntity()` function can be used to query the user to select a TrussEntity instance.

```
TrussEntity getTrussEntity();
TrussEntity getTrussEntity(String strPrompt);
```

---

```
class TrussEntity : CollectionEntity { // see CollectionEntity base functions (added
hsbCAD14.0.73)

    String definition() const;
    void setDefinition(String strVal); // see TrussDefinition for more information

    TrussDefinition definitionObject() const; // (added hsbCAD18.1.24)
    void setDefinitionObject(TrussDefinition val); // (added hsbCAD18.1.24)

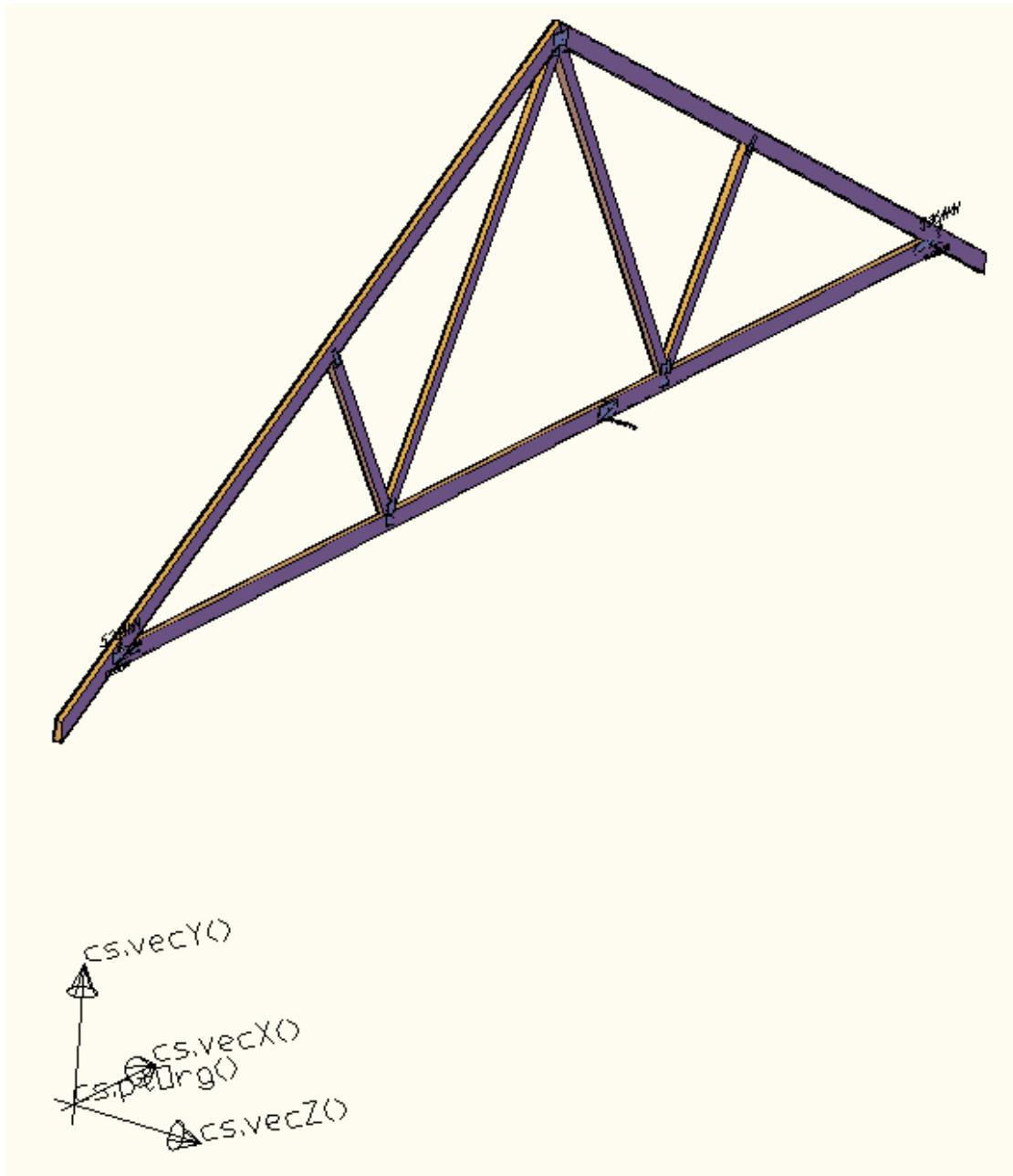
    void dbCreate(CoordSys csEcs, String strDefinition); // (added hsbCAD2014 build 19.1.1)

    double width() const; // (added V26)
    String alignment() const; // (added V26)

    TrussEnvelope trussEnvelope() const; // (added V25.1.15) see TrussEnvelope
};
```

---

*[Example O-type, with insert done inside script:*



```
U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    TrussEntity ce = getTrussEntity();
    Entity.append(ce);

    return;
}

PropString pDimStyle(1, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");
```

```

if (_Entity.length()==0) return;
TrussEntity ce = (TrussEntity)_Entity[0];
if (!ce.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

String strCreateNewTrussEntity = T("|Create new TrussEntity|");
addRecalcTrigger(_kContext, strCreateNewTrussEntity );
if (_bOnRecalc && _kExecuteKey==strCreateNewTrussEntity ) {
    CoordSys csEnt =ce.coordSys();
    csEnt .transformBy(U(1000)*csEnt.vecZ()); // additional move out
    String strDef = ce.definition();
    TrussEntity ceNew;
    ceNew.dbCreate(csEnt, strDef);
    _Entity[0] = ceNew;
    ce = ceNew;
}

CoordSys cs =ce.coordSys();
cs.vis();

TrussDefinition cd = ce.definition();
Beam arBm[] = cd.beam(); // find beams of truss definition
for (int e=0; e<arBm.length(); e++) {
    Beam bm = arBm[e];
    Body bd = bm.realBody();
    bd.transformBy(cs); // transform to entity coordsys
    bd.transformBy(U(500)*cs.vecX()); // additional move out
    // bd.vis(1);
}

String strLines[0];
strLines.append("TrussEntity");
strLines.append("definition: "+ce.definition());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example]*

### 7.44.1 TrussEnvelope

Can be instantiated from [TrussEntity::trussEnvelope\(\)](#) method.

---

```
class TrussEnvelope (added V26.0.2)
{
    int status() const; // returns one of: _kTESUndefined _kTESInvalid _kTESValid
                        _kTESGenerated _kTESLast
    int type() const; // returns one of: _kTETUndefined _kTETStandard _kTETGirder _kTETHip
                      _kTETValley _kTETJack _kTETRafter _kTETEndCreeper _kTETSideCreeper _kTETLast

    Point3d ptStart(); const;
    Point3d ptEnd(); const;
    CoordSys coordSys(); const;

    int numNodes() const;
    TrussEnvelopeNode nodeAt(int index) const;

    int numLines() const;
    TrussEnvelopeLine lineAt(int index) const;

};

class TrussEnvelopeNode (added V26.0.2)
{
    int type() const; // returns one of: _kTENTUndefined _kTENTGeneric _kTENTBearing
                      _kTENTLast
    Point3d position(); const;
}

class TrussEnvelopeLine (added V26.0.2)
{
    int type() const; // returns one of: _kTELTDDefined _kTELTRoof _kTELTRoofOverhang
                      _kTELTCeiling _kTELTLast
    int startNodeIndex(); const;
    int endNodeIndex(); const;
}
```

---

*[Example O-type, with insert done inside script:*

```
U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    TrussEntity trussEnt = getTrussEntity();
    _Entity.append(trussEnt);

    return;
}
```

```

PropString pDimStyle(1,_DimStyles , T("|Dim style|"));
PropDouble pTextHeight(0,U(20), T("|Text height|"));

if (_Entity.length()==0) return;
TrussEntity trussEnt = (TrussEntity)_Entity[0];
if (!trussEnt.bIsValid()) {
    eraseInstance(); // just erase from DB
    return;
}

setDependencyOnEntity(trussEnt);

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);

String strLines[0];
strLines.append("TrussEntity");
strLines.append("definition: "+trussEnt.definition());

TrussEnvelope trussEnv = trussEnt.trussEnvelope();
CoordSys cs = trussEnv.coordSys();
cs.vis();

strLines.append("number of nodes: "+trussEnv.numNodes());
for (int n = 0; n < trussEnv.numNodes(); n++)
{
    TrussEnvelopeNode node = trussEnv.nodeAt(n);
    Point3d pt = node.position();
    pt.transformBy(cs);
    pt.vis(1);

    int nType = node.type();
    String nodeType = (nType == 0) ? "_kTENUndefined" : (nType ==
1) ? "_kTENGeneric" : "_kTENBearing";
    dp.draw(nodeType, pt, cs.vecX(), cs.vecY(), 0, 1);
}

strLines.append("number of lines: "+trussEnv.numLines());
for (int l = 0; l < trussEnv.numLines(); l++)
{
    TrussEnvelopeLine line = trussEnv.lineAt(l);
    Point3d pts =
trussEnv.nodeAt(line.startNodeIndex()).position();
    Point3d ptE =
trussEnv.nodeAt(line.endNodeIndex()).position();
    pts.transformBy(cs);
    ptE.transformBy(cs);
    LineSeg seg(pts, ptE);
    seg.vis(2);

    Point3d pt = seg.ptMid();
}

```

```

        Vector3d dir = (seg.ptEnd() - seg.ptStart()).normal();

        int nType = line.type();
        String lnType = (nType == 0) ? "_kTELUndefined" : (nType ==
1) ? "_kTELRoof" : (nType == 2) ? "_kTELRoofOverhang" :
"_kTELCeiling";
        dp.draw(lnType, pt, dir, cs.vecZ().crossProduct(dir), 0, 1);
    }

    // display the lines
    for (int l=0; l<strLines.length(); l++) {
        Vector3d vecO = -l*1.2*pTextHeight*_YU;
        dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
    }
}

—end example]

```

## 7.44.2 TrussConnection

Works with [TrussEntity](#).

---

```

class TrussConnection : Tool (added V26.0.2)
{
    String connectionTypeName() const;
    void setConnectionTypeName(String strVal);

    void connectTrusses(TrussEntity trussHost, TrussEntity trussOther); // adds this tool to the
    trussHost

    // static calculation methods
    static Point3d refPositionHost(TrussEntity trussHost, TrussEntity trussOther);
    static Point3d refPositionOther(TrussEntity trussHost, TrussEntity trussOther);
}

```

---

[Example O-type, with insert done inside script:

```

    U(1, "mm");

    // Own properties
    PropDouble pSymbolSize(0, U(120), T("|Symbol size|"));

    if (_bOnInsert)      // Inserted directly from UI
    {
        // Get the two trusses connected
        _Entity.append(getTrussEntity("Select carried truss"));
        _Entity.append(getTrussEntity("Select carrier truss"));
    }
}

```

---

```
// Set up truss entity variables and validate
if (_Entity.length() != 2)
{
    eraseInstance(); // Erase from DB
    return;
}
TrussEntity trussHost = (TrussEntity)_Entity[0]; // Host entity
(physically carried)
TrussEntity trussOther = (TrussEntity)_Entity[1]; // Other entity
(physical carrier)
if ( !trussHost.bIsValid() || !trussOther.bIsValid() ) {
    eraseInstance(); // Erase from DB
    return;
}

setDependencyOnEntity(trussHost);
setDependencyOnEntity(trussOther);

// Update position
TrussConnection conn();
conn.setConnectionTypeName("SymbolicConnection");
conn.connectTrusses(trussHost, trussOther); // Recreating the
connection tool on update
auto ptHost = conn.refPositionHost(trussHost, trussOther);

// Draw parameters
Display dp(-1);
double outRadius = pSymbolSize / 2;
double inRadius = outRadius - U(5);

// Draw symbol: in the plane of the first
dp.trueColor(rgb(0, 0, 255));
PLine circle;
auto nHost = trussHost.trussEnvelope().coordSys().vecZ();
circle.createCircle(ptHost, nHost, outRadius);
dp.draw(circle);
circle.createCircle(ptHost, nHost, inRadius);
dp.draw(circle);

// Draw symbol: on the plane of the other
dp.trueColor(rgb(0, 255, 0));
auto ptOther = conn.refPositionOther(trussHost, trussOther);
auto nOther = trussOther.trussEnvelope().coordSys().vecZ();
circle.createCircle(ptOther, nOther, outRadius);
dp.draw(circle);
circle.createCircle(ptOther, nOther, inRadius);
dp.draw(circle);

// Draw symbol: line between host and other (prominent only when have
gap)
dp.trueColor(rgb(255, 0, 0));
PLine connLine(ptHost, ptOther);
dp.draw(connLine);
```

---

—end example]

### 7.44.3 TrussSupport

Works with [TrussEntity](#).

---

```
class TrussSupport : Tool (added V26.0.2)
{
    int isActive() const;
    void setisActive(int iVal);

    double verticalTolerance() const;
    void setVerticalTolerance(double dVal);

    String supportTypeName() const;
    void setSupportTypeName(String strVal);

    void create(TrussEntity trussHost, Entity entOther); // makes this a persistent tool for
                                                       trussHost.

    // static calculation methods
    static int canHaveValidGeom(TrussEntity trussHost, Entity entOther); // returns TRUE or
                                                               FALSE
    static LineSeg geomSupport(TrussEntity trussHost, Entity entOther); // returns the LineSeg
                                                               of the support
}
```

---

[Example O-type, with insert done inside script:

—end example]

## 7.45 Wall

The term Wall refers to an instance of an ADT wall.

An entity can be casted into a Wall. However when the casting is not allowed, the resulting Wall will become invalid. This can be checked with the `bIsValid()` function of Entity.

Because Wall is derived from [Entity](#), it inherits all the member functions of Entity as well. It can also be casted to an Entity. During `_bOnInsert`, the `getWall()` function can be used to query the user to select a wall instance.

```
Wall getWall();
Wall getWall(String strPrompt);
```

---

```
class Wall : Entity { // see Entity for base functions

    CoordSys coordSys() const; // internal coordsys
    CoordSys coordSysHsb() const; // mirrorable coordsys

    double instanceWidth() const;
    void setInstanceWidth(double dValue);
    double baseHeight() const;
    void setBaseHeight(double dValue);

    String description() const;
    void setDescription(String strVal);

    double totalWidth() const; // added v19.1.114 , v20.0.96 and v21.0.3
    double zFaceOffset() const; // added v19.1.114 , v20.0.96 and v21.0.3

    Point3d ptStart() const;
    Point3d ptEnd() const;
    void setStartEnd(Point3d ptStart, Point3d ptEnd);
    void setStartEnd(Point3d ptStart, Point3d ptEnd, int bKeepOpeningsFixedAtTheirLocation); // 
        added hsbCAD2012 build 17.0.46. If parameter is TRUE, the openings are
        fixed at their geometrical location. The version without the parameter does
        not keep the openings at their location, but honors the anchors.

    Wall dbSplit(Point3d ptSplit); // split this wall, and return the new created wall.

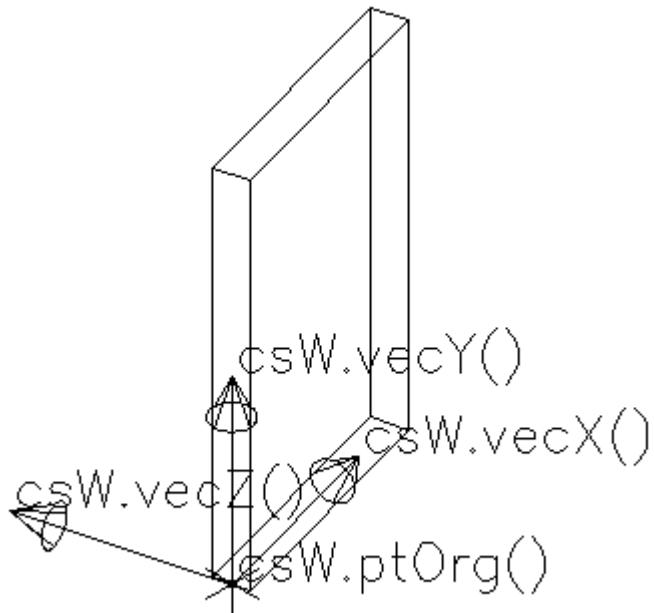
    Body shrinkWrapBody(int bCacheGraphics, int bDoMergers, int bCutOpenings, int
        bDoInterference, int bApplyBodyModifiers); // (added hsbCAD17.0.54 and
        16.3.21)

};
```

---

**CoordSys coordSys()** const;

Build the coordinate system of the wall. It corresponds with the ECS (entity coordinate system) of the entity, located at the insertion point of the entity.



[Example O-type, with insert done inside script:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

if (_bOnInsert) {
    _Pt0 = getPoint(); // select point
    Wall wl = getWall();
    _Entity.append(wl);

    return;
}

```

```

String strLines[0];

for (int i=0; i<Entity.length(); i++)
{
    Wall wl = (Wall)Entity[i];
    if (!wl.bIsValid()) continue;

    CoordSys csW = wl.coordSysHsb();
    csW.vis();

    Point3d ptStart = wl.ptStart(); ptStart.vis(1);
    Point3d ptEnd = wl.ptEnd(); ptEnd.vis(3);

    Point3d ptStartZP = ptStart + wl.zFaceOffset() * csW.vecZ();
    ptStartZP.vis(4);
    Point3d ptStartZN = ptStartZP - wl.totalWidth() * csW.vecZ();
    ptStartZN.vis(4);

    strLines.append("instanceWidth:"+wl.instanceWidth());
    strLines.append("totalWidth :" +wl.totalWidth ());
    strLines.append("zFaceOffset:" +wl.zFaceOffset());
    strLines.append("baseHeight :" +wl.baseHeight ());
    strLines.append("description:" +wl.description());
}

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YW;
    dp.draw(strLines[l], _Pt0+vecO , _XW, _YW, 1,1);
}

—end example]

```

[Example O-type, with insert done inside script:

```

U(1, "mm");
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Wall wl = getWall();
    Entity.append(wl);

    return;
}

for (int i=0; i<Entity.length(); i++) {

```

```

    Wall wl = (Wall)_Entity[i];
    int bCacheGraphics = FALSE;
    int bDoMergers = FALSE;
    int bCutOpenings = TRUE;
    int bDoInterference = TRUE;
    int bApplyBodyModifiers = TRUE;
    Body bd = wl.shrinkWrapBody(bCacheGraphics, bDoMergers,
    bCutOpenings, bDoInterference, bApplyBodyModifiers);
    bd.vis(1);

}

```

*—end example]*

*[Example O-type, that tracks the wall being mirrored, and reacts to it:*

```

String toolSides[] = {T("|Left|"), T("|Right|")};
PropString propToolSide(0, toolSides, T("|Tool side|"), 1);

if (_bOnInsert)
{
    _Pt0 = getPoint(); //select point
    _Entity.append(getWall());
    return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}

Wall wll = (Wall)_Entity.first();
if (!wll.bIsValid())
{
    eraseInstance();
    return;
}

int tracker = _Map.getInt("mirrorTracker"); // default 0, not set
int coordSysSign = (wll.coordSysHsb().det() < 0) ? -1 : 1; // value
-1 or 1
if (tracker != coordSysSign)
{
    if (tracker != 0)
    {
        String strToolSideOrig = propToolSide;
        int index = toolSides.find(propToolSide, 0); //current
        index = (index + 1) % 2; // remainder of division by 2
        String strToolSideNew = toolSides[index];
        propToolSide.set(strToolSideNew); //correct state
        reportMessage("\n" + "mirror state updated from " +
strToolSideOrig + " to " + strToolSideNew);
    }
}

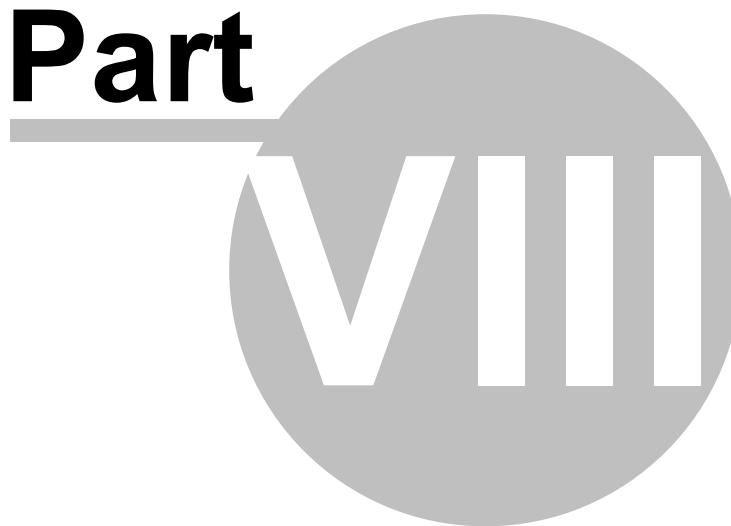
```

```
        }
        _Map.setInt("mirrorTracker", coordSysSign); // store state
    }

reportMessage("\n" + scriptName() + "[" + _ThisInst.handle() + "] "
+ ", for wall" + "[" + w1.handle() + "]"
+ ", side: " + propToolSide
+ ", tracker: " + tracker
+ ", coordSysSign: " + coordSysSign);
```

—end example]

**Part**



## 8 Tools on GenBeam

### 8.1 General Tool

All tools that are applicable on beam have some member functions in common. These routines are setDoSolid, allowMachineForCNC, excludeMachineForCNC and setJapaneseMarking.

For advanced use only, the setDoSolid function can prevent the tool to act on the solid representation of the beam. Its default value is set to TRUE. If you set the value to FALSE, the tool will not be doing its operation on the solid, but it WILL generate the machine instructions for it. Please use with care, because the posnum generation is done based on the solid.

```
setDoSolid(int bSet);
```

Two other routines for advanced usage only are

```
allowMachineForCNC(int nMachineType);
excludeMachineForCNC(int nMachineType);
```

These routines allow to specify for which machine the CNC generation should be done. Possible values of nMachineType are: **\_kAnyMachine**, **\_kHundegger**, **\_kRandek**, **\_kTorwegge**, **\_kMetalIT**, **\_kWeinmannSaege**. The default value is **\_kAnyMachine**. If nothing is specified, all machines are allowed, none are excluded.

To set the japanese marking flag to TRUE or FALSE, one must use the setJapaneseMarking routine. Default it is set to FALSE. If the japanese marking flag is set to TRUE, grid location of the tool on the beam, will be marked.

```
setJapaneseMarking(int bSet);
```

The tool can also be added at once to all GenBeams collected in an array. This routine is nothing else then a for loop over all the GenBeams passed in as the argument. For each GenBeam, the addTool routine is called. The number of GenBeams that the tool is added to is returned.

```
int addMeToGenBeams(GenBeam[] arGenBeamsToAddTo) const;
int addMeToGenBeams(Beam[] arBeamsToAddTo) const;
int addMeToGenBeams(Sheet[] arSheetsToAddTo) const;
int addMeToGenBeams(Sip[] arSipsToAddTo) const;
```

Or even better, only add the tool to the GenBeams in the array that have an intersection with the cuttingBody of the tool. Instead of just adding the tool to each GenBeam of the array, first a check is done if the cuttingBody of this tool has an intersection with the GenBeam element. If it has an intersection, the tool is added. If there is no intersection, the tool is not added. If the cuttingBody routine is not implemented for this tool, then the addMeToGenBeamsIntersect routine is also returning a syntax error. The number of GenBeams that the tool is added to is returned.

```
int addMeToGenBeamsIntersect(GenBeam[] arGenBeamsToAddTo) const;
int addMeToGenBeamsIntersect(Beam[] arBeamsToAddTo) const;
int addMeToGenBeamsIntersect(Sheet[] arSheetsToAddTo) const;
```

```
int addMeToGenBeamsIntersect(Sip[] arSipsToAddTo) const;
```

To set this tool to be dimensioned automatically in the shopdrawings, one needs to set the auto dimensioning to TRUE.

**setAutoDimInfo(int bSet);**

For some tools the solid body that represents the tool can be retrieved from

**Body** cuttingBody() const

```

class Tool // base class for all the tools
{
    setDoSolid(int bSet);

    allowMachineForCNC(int nMachineType);
    excludeMachineForCNC(int nMachineType);

    int addMeToGenBeams(GenBeam[] arGenBeamsToAddTo) const;
    int addMeToGenBeams(Beam[] arBeamsToAddTo) const;
    int addMeToGenBeams(Sheet[] arSheetsToAddTo) const;
    int addMeToGenBeams(Sip[] arSipsToAddTo) const;

    Body cuttingBody() const;

    int addMeToGenBeamsIntersect(GenBeam[] arGenBeamsToAddTo) const;
    int addMeToGenBeamsIntersect(Beam[] arBeamsToAddTo) const;
    int addMeToGenBeamsIntersect(Sheet[] arSheetsToAddTo) const;
    int addMeToGenBeamsIntersect(Sip[] arSipsToAddTo) const;

    setAutoDimInfo(int bSet);
    setJapaneseMarking(int bSet);

    // The subMapX set of methods added v23.5.22 and v24.0
    Map subMapX(String strKey) const; // see Map
    void setSubMapX(String strKey, Map mapNew);
    String[] subMapXKeys() const; // return list of available sub map keys
    void removeSubMapX(String strKey);

    void setPosnumCompareVertices(Point3d[] pnts); // added V25.1.58, example see
        FreeProfile tool.

    // Some tools also support to set values from its corresponding AnalysedTool its internal
        map.
    int setFromMap(Map mplInternalFromAnalysedTool); // returns TRUE if successful, else
        FALSE. (added V25.1.76)
}

```

[Example O-type:

```
Unit(1, "mm");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
}

if (_Element.length()==0) return; // 0 is a valid index
Element el = _Element[0];

CoordSys cs = el.coordSys(_Pt0); // locate the coordinate system in _Pt0
_Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point on the element

// add the script entity to the element group itself, so it can be toggled with the element
assignToElementGroup(el, TRUE, 0, 'C');

//Drill tool(cs.ptOrg()-U(1000)*cs.vecZ(), cs.ptOrg()+U(1000)*cs.vecZ(), U(100));
BeamCut tool(cs.ptOrg(), U(1000)*cs.vecZ(), U(100)*cs.vecY(), U(200)*cs.vecX());
tool.cuttingBody().vis();

Beam beams[] = el.beam();
int nBeams = tool.addMeToGenBeamsIntersect(beams);
Sheet sheets[] = el.sheet();
int nSheets = tool.addMeToGenBeamsIntersect(sheets);
reportNotice("\nThe tool was added to "+nBeams+" beams and "+nSheets+" sheets.");

—end example]
```

## 8.2 Arc

The Arc tool cuts out an arc from the face of a timber. The constructor has the following formats:

```
Arc name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Arc name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
           dXWidth, double dYHeight, double dZDepth);
Arc name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
           dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
           double dZFlag);
```

ptOrg:	origin point of the box defining the housing operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

The vectors `vecZ` determines the direction from the centerpoint of the circle of the arc, into the timber.

To specify the dimensions of the arc, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the housing is the same as `vecX.length()*dXWidth`. If `vecX` is a unit-length vector, then the `dXWidth` corresponds with the width. If `dXWidth` equals 1, then the length of the vector expresses the height of the arc.

The flags `dXFlag`, `dYFlag` and `dZFlag`, specify the position of the `ptOrg` inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point `ptOrg` is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the `-vecX`, `-vecY` and `-vecZ` direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the `vecX`, `vecY` and `vecZ` coordinate system with origin `ptOrg`.

The solid body that represents this tool can be used by calling

**Body** `cuttingBody()` const;

See [General](#) for member functions applicable to all tools.

---

[Example E-type with 3 required beams:

```

Unit(1, "mm");
double dDepth = U(40);

Vector3d vecN = _Z0; // normal direction on arc surface
Vector3d vecT = _Y0; // tangential direction on arc surface
double dN = _H0; // beam dimension in normal direction
double dT = _W0;

// find direction in which the arc lies
Vector3d vecDir = _Pt0 - _L1.closestPointTo(_L2);
if (abs(vecDir.dotProduct(vecN))<abs(vecDir.dotProduct(vecT))) {
    // the vectT seems to be the normal direction => swap
    vecN = _Y0;
    vecT = _Z0;
    dN = _W0;
    dT = _H0;
}

if (vecDir.dotProduct(vecN)<0) {
    // need the oposite direction
    vecN = -vecN;
}

Point3d ptArcSurf = _Pt0 + 0.5*dN*vecN;
ptArcSurf.vis();

// find intersection points with beam1 and beam2
LineBeamIntersect lb1(ptArcSurf, _X0, Beam1, 0);

```

---

```

LineBeamIntersect lb2(ptArcSurf, _X0, _Beam2, 0);

if (lb1.bHasContact()&&lb2.bHasContact()) {

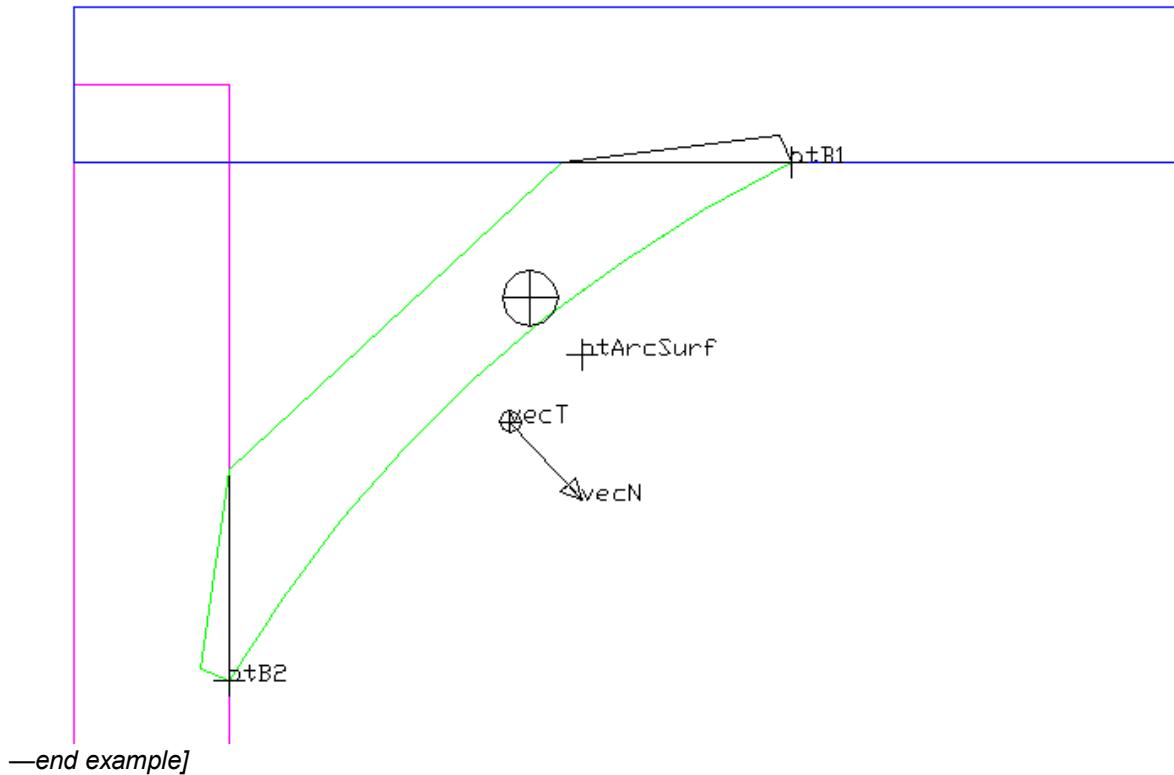
    Point3d ptB1 = lb1.pt1(); ptB1.vis(); // contact point of line beam intersection
    Point3d ptB2 = lb2.pt1(); ptB2.vis();
    Point3d ptMid = (ptB1+ptB2)/2; // midpoint of arc

    vecN.vis(ptMid);
    vecT.vis(ptMid);

    Arc tArc(ptMid - 0.5*dDepth*vecN, ptB2-ptB1, dT*vecT, -dDepth*vecN);
    _Beam0.addTool(tArc);

}

```



## 8.3 Ari

The ari tool (added since hsbCAD2012 build 17.0.15) its constructor has the following format:

```

Ari name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
        int nMillHeadIndex, double dZDepth,
        double dTopXWidth, double dTopRadius, double dAriAngle, double dYHeight,

```

**double** dBottomRadius, **int** nlsFemaleType);

ptOrg:	point opposite from top of dove, on the female surface
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
nMillHeadIndex:	index of the mill head to be used
dZDepth:	height in vecZ direction
dTopXWidth:	width in X at the top
dTopRadius:	radius
dAriAngle:	angle in degrees
dYHeight:	height in vecY direction
dBottomRadius:	bottom radius

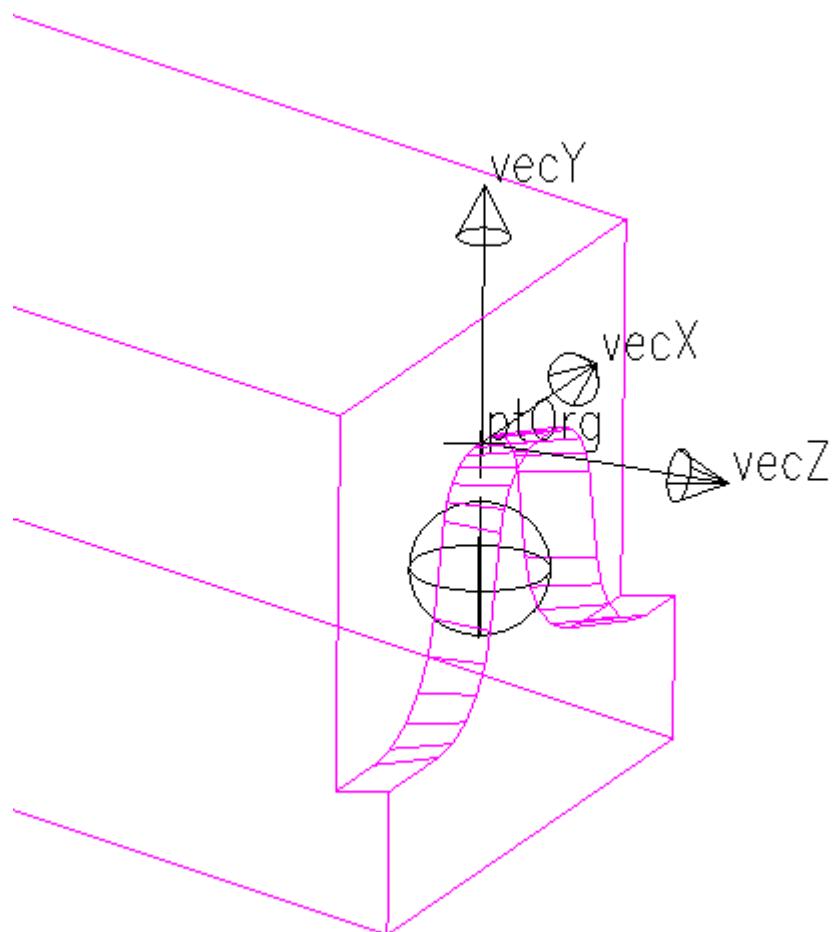
The ari tool with the same geometrical parameters can be used for both the male and the female side. The only parameter that differs is the nlsFemaleType parameter. To change this value on an existing dove, the setIsFemale function can be used.

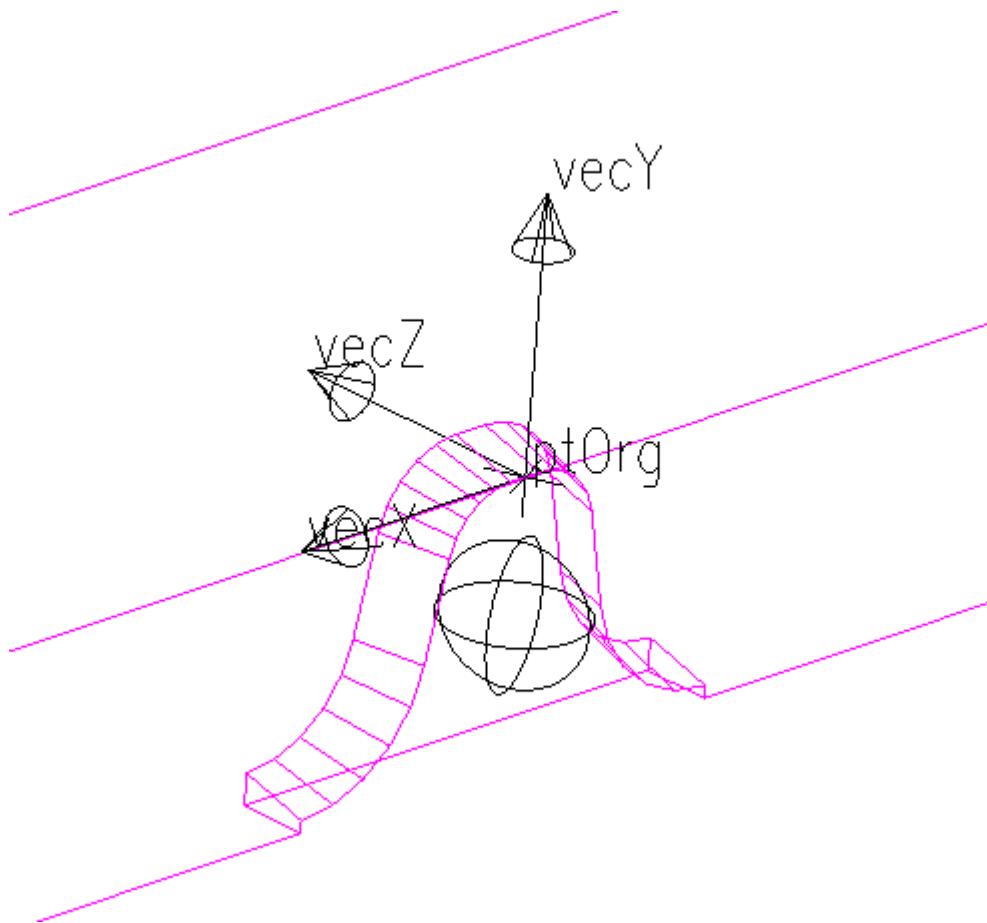
**Ari::setIsFemale(**int** blsFemaleType);**

The Ari tool is not an end tool. It cannot be stretched away on its own. To have endTool behaviour, the etool needs to have at least one end tool, eg a Cut.

If nMillHeadIndex is set to 0, the shape is cut with the dove head mill. When the index is 1, the cut is with a normal finger mill.

See [General](#) for member functions applicable to all tools.





[Example T-type:

```

Unit(1, "mm"); // use mm as unit in U() function from now on

int nMillHeadIndex = 0;
double dYHeight = 0.5*_H0;
double dZDepth= U(20);
double dTopXWidth= U(10);
double dTopRadius= U(20);
double dBottomRadius= U(40);
double dAriAngle = 12;

// rotate the coordinate system with the _Z0 axis
Vector3d vecZ = _Z1; // _Z1 is always normal to contact surface =>
// depth direction
Vector3d vecY = _Z0; // _Z0 initially most aligned with _ZW =>
// height direction
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and
vecZ
Point3d ptOrg = _Pt0 +0.25*_H0*_Z0; // move _Pt0 half beam height up

```

```

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Ari ari(ptOrg, vecX, vecY, vecZ, nMillHeadIndex, dZDepth,
          dTopXWidth, dTopRadius, dAriAngle, dYHeight,
          dBottomRadius, FALSE);
Beam0.addTool(ari); // male beam

ari.setIsFemale(TRUE);
Beam1.addTool(ari); // female beam

// The Ari tool is not an end tool. If you need endTool behaviour, a
cut needs to be added.
Cut ct(ptOrg+vecZ*dZDepth, vecZ);
Beam0.addTool(ct, kStretchOnInsert); // male beam

```

*—end example*

## 8.4 BeamCut

**BeamCut** name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>);  
**BeamCut** name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>,<X-length>,<Y-width>,<Z-width>);  
**BeamCut** name(<origin>,<Xaxis>,<Yaxis>,<Zaxis>,<X-length>,<Y-width>,<Z-width>,
 <X-flag>,<Y-flag>,<Z-flag> );

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;  
**void** setModifySectionForCnC(**int** bSet); // default is TRUE. If you call this with argument FALSE, the beamcut dimensions will not be extended during the machine generation phase. (the setModifySectionForCnc was active for BVN since 2007, and is active for AnalysedTools (Btl, Bvx...) since v20.0.116.  
**void** setFreeDirectionsFromRealSolid(**int** bSet); // default is modifySectionForCnC(). If you call this with argument TRUE, the beamcut free directions for cnc will be evaluated using the real solid of the GenBeam. If the argument is FALSE, then the free directions are determined using the envelope quader of the GenBeam. In this case the modifySectionForCnC might be important (added v27.3.2 and v26.9.22).

Other member methods

**Quader** quader() const; // see [Quader](#) (added hsbCAD21.0.103)  
**int** modifySectionForCnC() const; // (added hsbCAD21.0.103)  
**int** freeDirectionsFromRealSolid() const; // (added v27.3.2 and v26.9.22)

A copy of the BeamCut tools of a [GenBeam](#) can be retrieved with the function:

**BeamCut[]** [GenBeam](#)::getToolsStaticOfTypeBeamCut() const; // (since v21.0.103)

---

[Example E-type illustrating the removal of specific static beamcuts:

```

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

GenBeam genbeam = _GenBeam[0];

String strRemoveAllBeamCuts = T("|Remove all static beamcut|");
addRecalcTrigger(_kContext, strRemoveAllBeamCuts );
if (_bOnRecalc && _kExecuteKey==strRemoveAllBeamCuts )
{
    int nCountRemoved = genbeam.removeToolsStaticOfType(BeamCut());
    //int nCountRemoved = genbeam.removeToolsStatic(Drill()); //
obsolete
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

String strRemoveFirstBeamCut = T("|Remove first beamcut|");
addRecalcTrigger(_kContext, strRemoveFirstBeamCut );
if (_bOnRecalc && _kExecuteKey==strRemoveFirstBeamCut )
{
    BeamCut drllsAll[] = genbeam.getToolsStaticOfTypeBeamCut();
    int nCountRemoved = 0;
    if (drllsAll.length() > 0)
        nCountRemoved = genbeam.removeToolStatic(drllsAll[0]);
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

BeamCut bmcts[] = genbeam.getToolsStaticOfTypeBeamCut();

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
String strNum = "Number of beamcuts: " + bmcts.length();
dp.draw(strNum , _Pt0, _XU, _YU, 1, 1);

for (int i=0; i<bmcts.length(); i++)
{
    int nColor = i+1;
    dp.color(nColor);

    BeamCut bmct = bmcts[i];
    Quader qdr = bmct.quader();

    Body bd(qdr);
    dp.draw(bd);

    String str = "ModSec: " + bmct.modifySectionForCnC();
    dp.draw(str, qdr.ptOrg(), qdr.vecX(), qdr.vecY(), 1, 1);
}

```

*—end example]*

## 8.5 Chamfer

The Chamfer constructor has one of the following formats:

```
Chamfer name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
dLength, double dWidthBeam, double dHeightBeam, int nSides, double dDepth);
Chamfer name(Beam beam, double dLength, int nSides, double dDepth);
```

ptOrg:	point on axis of beam, center point in vecX direction of operation
vecX:	axis along beam axis
vecY:	axis of beam section
vecZ:	other axis of beam section
dLength:	length dimension in vecX direction
dWidthBeam:	beam width, in vecY direction
dHeightBeam:	beam height, in vecZ direction
nSides:	a value between 0 and 16 determining the sides
dDepth:	depth measured along a beam side
beam:	In case the beam is specified, the ptOrg is taken as the center point of the beam, the vecX, vecY, vecZ, dWidthBeam and dHeightBeam are taken from the beam.

The value of nSides is determined as a sum of values from 1,2,4 and 8. Each of the values 1,2,4 and 8 correspond to a particular edge. No side means 0, all sides mean 15. Making a sum of some values, corresponds to a combination of some edges. This sum can be composed by bitwise combination of the following predefines:

<b>_KNYPZ:</b>	edge in the negative Y and positive Z quadrant of the beam section.
<b>_KPYPZ:</b>	edge in the positive Y and positive Z
<b>_KPYNZ:</b>	edge in the positive Y and negative Z
<b>_KNYNZ:</b>	edge in the negative Y and negative Z

An example of such a bitwise combination is **\_KNYPZ | \_KPYNZ** another combination is **\_KNYPZ | \_KPYNZ | \_KNYNZ**.

The Chamfer has a feed type. It can be set separately by the function setFeed.

```
Chamfer::setFeed(int nFeedType);
```

Values of nFeedType:

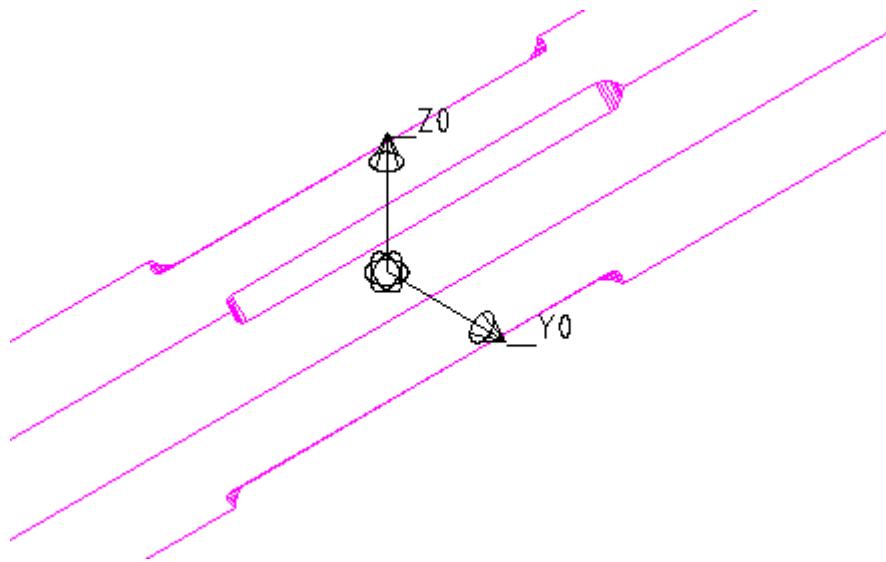
<b>_KChamferRound:</b>	rounded
<b>_KChamfer45:</b>	45 angle
<b>_KChamfer90:</b>	90 degrees angle, (default if not set)

See [General](#) for member functions applicable to all tools.

---

---

[Example:



```
U(1, "mm");
_y0.vis(_Pt0);
_z0.vis(_Pt0);
Chamfer ch(_Pt0, _x0, _y0, _z0, U(500), _w0, _h0, _kPYNZ | _kPYPZ |
_kNYPZ, U(20));
ch.setFeed(_kChamferRound);
_Beam0.addTool(ch);
```

—end example]

## 8.6 ChamferedLap

The ChamferedLap tool will add a chamfered lap tool into the mortise beam. The constructor has the following formats:

```
ChamferedLap name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
Vector3d vecSlopedEdge);
ChamferedLap name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
double dXWidth, double dYHeight, double dZDepth, Vector3d vecSlopedEdge);
ChamferedLap name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
double dXWidth, double dYHeight, double dZDepth, double dXFlag, double
dYFlag, double dZFlag, Vector3d vecSlopedEdge);
```

ptOrg:	origin point of the box defining the housing operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

`vecSlopedEdge`: vector along the sloped edge

The vectors `vecX` and `vecY` determine the direction of the parallel lines that define the cutout on the surface.

To specify the dimensions of the tooling, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the tooling is the same as `vecX.length()*dXWidth`. If `vecX` is a unit-length vector, then the `dXWidth` corresponds with the width. If `dXWidth` equals 1, then the length of the vector expresses the height of the tooling.

The flags `dXFlag`, `dYFlag` and `dZFlag`, specify the position of the `ptOrg` inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point `ptOrg` is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the `-vecX`, `-vecY` and `-vecZ` direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the `vecX`, `vecY` and `vecZ` coordinate system with origin `ptOrg`.

The solid body that represents this tool can be used by calling

**Body** `cuttingBody()` const;

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```
U nit(1,"mm"); // use mm as unit in U() function from now on

P ropD ouble dDepth(0,U(30), "Depth");

// check the condition for the tool to be correct
// this example can only handle single sloped combinations
if (!(_Y1.isParallelTo(_Y0) || _Y1.isParallelTo(_Z0))) return;

// Make sure the _X1 is always in the _X0 direction
if (_X1.dotProduct(_X0)<0) _X1 = -_X1;

// Make sure the _Pt1 is on the _X1 side, and _Pt3 is on the +_X1 side
if (_X1.dotProduct(_Pt3-_Pt1)<0) {
    P oint3d ptt=_Pt3;
    _Pt3=_Pt1;
    _Pt1=ptt;
}

// Make sure the _Pt2 is on the _X1 side, and _Pt4 is on the +_X1 side
if (_X1.dotProduct(_Pt4-_Pt2)<0) {
    P oint3d ptt=_Pt4;
    _Pt4=_Pt2;
    _Pt2=ptt;
}

//_Pt1.v is();
//_Pt2.v is();
```

```

//_P t3 .v is ();
//_P t4 .v is ();

// rotate the coordinate system with the _Z 0 axis
Vector3d vecZ = _Z1; // _Z1 is always normal to contact surface => depth direction
Vector3d vecY = _Y1;
Vector3d vecX = _X1;
Vector3d vecSlopedEdge = _X0;
Po in t3 d pTo_rg = 0.5 * (_P t1 + _P t2);
Po in t3 d pToTheer = 0.5 * (_P t3 + _P t4);

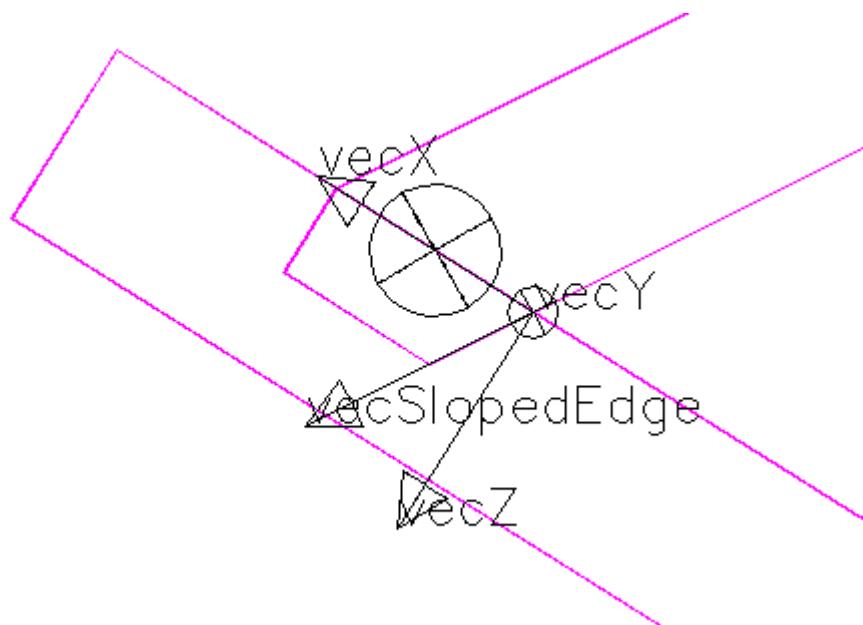
// visualize orientation in debug mode
Po in t3 d pTvIs = pTo_rg;
pTo_rg .v is ();
vecX .v is (pTvIs);
vecY .v is (pTvIs);
vecZ .v is (pTvIs);
vecslopedEdge .v is (pTvIs);

double dXW = _X1 .dotProduct(_P t3 - _P t1);
double dYH = abs(_Y1 .dotProduct(_P t2 - _P t1));

ChamferedLap cl(pTo_rg, vecX, vecY, vecZ, dXW, dYH, dDepth, 1, 0, 1, vecSlopedEdge);
_Beam1.addTool(cl); // female beam

Cutct(_P t0 + dDepth * _Z1, _Z1);
_Beam0.addTool(ct1);
Cutctp(pToTheer, _X1);
_Beam0.addTool(ctp0);

```



*—end example]*

## 8.7 CncExport

The CncExport tool enables to define arbitrary machine tools for the Hsb machine generation process. The tool is meant for advanced use only.

The constructor has the following format:

```
CncExport name(String strToolName, Map map);
```

strToolName: The name of the tool in the Hsb machine generation process.  
map: The map that describes the Cnc tool to be generated.

This tool will not result in any solid operations. It is therefor needed that the tool is accompanied with the proper tools that do the corresponding solid operations. The solid operations are important because the assignment of the posnums is based on the solid representation of the beam.

See [General](#) for member functions applicable to all tools.

The CncExport has some member functions, which return a copy of the constructor values:

```
Map map() const; // see Map  
String toolName() const;
```

A copy of the CncExport tools of a [GenBeam](#) can be retrieved with the function:

```
CncExport[] GenBeam::getToolsOfTypeCncExport() const;
```

---

[Example E-type:

```
Unit(1,"mm");  
  
Map map;  
map.setInt("int",3);  
map.setDouble("len",U(5));  
map.setPoint3d("pt0",_Pt0);  
map.setVector3d("vecXY",_XU+_YU);  
CncExport tl("CNC SPECIAL TOOL",map);  
tl.excludeMachineForCNC(_kHundegger);  
_Beam0.addTool(tl);
```

—end example]

## 8.8 CncMessage

The CncMessage tool enables to report a message in the hsbCAD Report window during CNC file generation. The constructor has the following format:

```
CncMessage name(String strToReport);
CncMessage name(String strToReport, int nMachine);
```

- strToReport: The string that will be displayed in the modeless dialog that pops up during CNC generation.
- nMachine: The type of machine that this message should show for. Possible values are: **\_KAnyMachine**, **\_kHundegger**, **\_kRandek**, **\_kTorwegge**, **\_kMetaIT**, **\_kWeinmannSaege**. The default value is **\_kAnyMachine**.

For an example, see the example of the [SolidSubtract](#) tool.

See [General](#) for member functions applicable to all tools.

---

## 8.9 ComplexProfile

The ComplexProfile tool constructor has the following format:

```
ComplexProfile name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
                     double dAngle, double dStartDepth, double dMaxDepth, double dMinDepth,
                     double dEndDepth, double dLength, int nPreMill);
```

- ptOrg: point where the endcut is placed on the beam
- vecX: axis in beam length direction, pointing towards the end point
- vecY: normal on face, oriented towards the beam
- vecZ: vecY.crossProduct(vecX)
- dAngle: angle in degrees
- dStartDepth, dMaxDepth, dMinDepth, dEndDepth: parameters determining the profile
- dLength: length of the profile
- nPreMill: premill type

Most of the parameters can be set separately by the functions:

```
ComplexProfile::setAngle(double dAngle);
ComplexProfile::setStartDepth(double dDepth);
ComplexProfile::setMaxDepth(double dDepth);
ComplexProfile::setMinDepth(double dDepth);
ComplexProfile::setEndDepth(double dDepth);
ComplexProfile::setLength(double dLength);
ComplexProfile::setPreMillType(int nPreMill);
```

Values of nPreMill are: **\_kCompleted**, **\_kOnlyBrdMth**, **\_kMillWithout**, **\_kWithUF**.

---

Important constraints for the ComplexProfile tool are:

- the ptOrg is located on the surface of the beam that is in the -vecY direction.
- the vecY is the normal on the surface pointing towards the beam
- the vecX direction must coincide with the X axis of the timber

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```
Unit(1, "mm"); // use mm as unit in U() function from now on

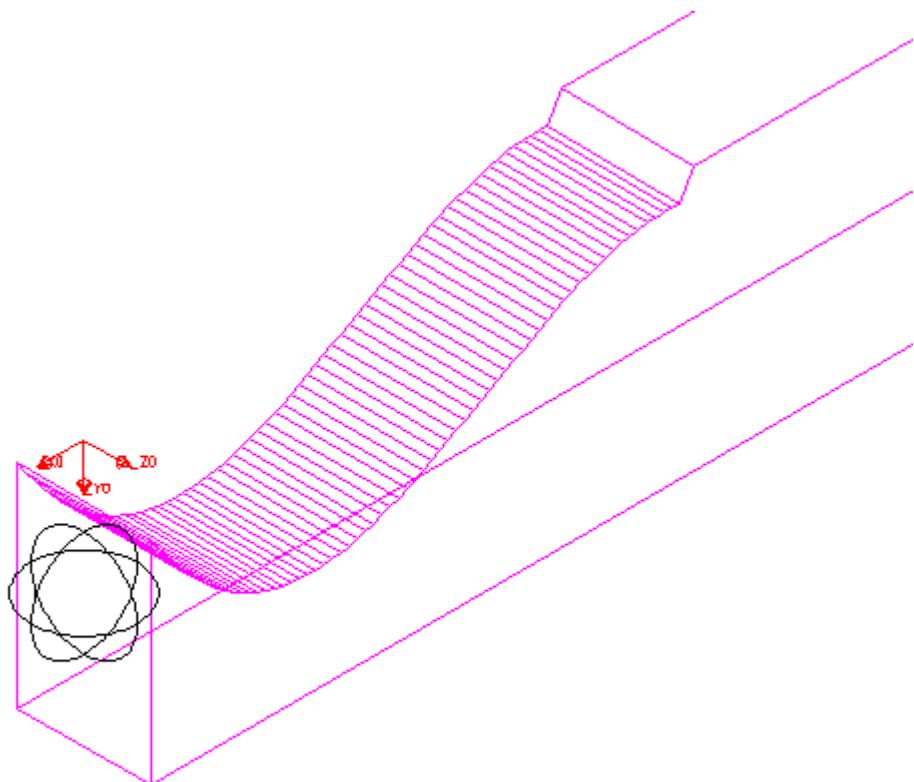
String arStrPreMillTypes []={T("|Completed|"), T("|Only
birdsmouth|"), T("|Without|"), T("|Completely with UF|")};
int arNPreMillTypes []={_kCompleted, _kOnlyBrdMth, _kMillWithout,
_kWithUF};
PropString pType(0,arStrPreMillTypes, T("|PreMill type|"));
int nPreMillType = arNPreMillTypes[arStrPreMillTypes.find(pType,0)];

Point3d ptIns = _Pt0-0.5*_w0*_y0; // calculate point on surface of
beam

double dAngle = 90;
double dStartDepth = U(20);
double dMaxDepth = U(60);
double dMinDepth = U(40);
double dEndDepth = U(10);
double dLen = U(200);

ComplexProfile cpr(ptIns, _x0, _y0, _z0, dAngle, dStartDepth,
dMaxDepth, dMinDepth, dEndDepth, dLen, nPreMillType);
_Beam0.addTool(cpr, _kStretchOnInsert);

_x0.vis(ptIns ,1);
_y0.vis(ptIns ,1);
_z0.vis(ptIns ,1);
```



*—end example]*

## 8.10 ConeDrill

The cone drill tool (added V25) constructor has one of the following format:

```
ConeDrill name(Point3d ptCenter, Vector3d vecDir, double dRadius, double
dAngleInDegrees)
```

ptCenter:	point where the radius is specified
vecDir:	vector pointing to the apex of the cone
dRadius:	radius of the cone at ptCenter
dAngleInDegrees:	angle of the cone

The radius and angle specified inside the constructor can be changed.

```
ConeDrill::setRadius(double dRadius);
ConeDrill::setOpeningAngle(double dAngleInDegrees);
```

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;

Other member methods

```
Point3d ptCenter() const;
Vector3d vecDir() const;
double dDiameter() const;
double dRadius() const;
double openingAngle() const;
```

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```
Unit(1, "mm");

PropDouble pDiam(0, U(100), T("|Diameter|"));
PropDouble pAngle(1, U(30), T("|Angle|"));
pAngle.setFormat(_kAngle);

Point3d ptSurf = _Pt0+0.5*_H0*_Z0;
ConeDrill cone(ptSurf, -_Z0, 0.5 * pDiam, pAngle);

Point3d ptC = cone.ptCenter();
Vector3d vecDr = cone.vecDir();
ptC.vis();
vecDr.vis(ptC);

_Beam0.addTool(cone);
```

—end example]

## 8.11 ConvexConcaveProfile

The ConvexConcaveProfile constructor has the following format (added in hsbCAD2011 build 16.0.52):

```
ConvexConcaveProfile name(Point3d ptOrg, Vector3d vecX, Vector3d vecZ, int
nProfileType, double dBackCut, double dRadius);
```

ptOrg:	point on start point of profile, at far end of beam
vecX:	axis along beam axis, pointing from beam center to tool
vecZ:	axis along beam height, positive vecZ is cut away
nProfileType:	type indicating convex or concave shape

dBackCut: dimension of tool in vecZ direction (value B in image below)  
dRadius: radius of circular segment

See [General](#) for member functions applicable to all tools.

Since (hsbCAD2011 build 16.2.8) the ConvexConcaveProfile tool can be set to not act as an endtool. In this case the end cut is not done. If not set, the tool is an end tool.

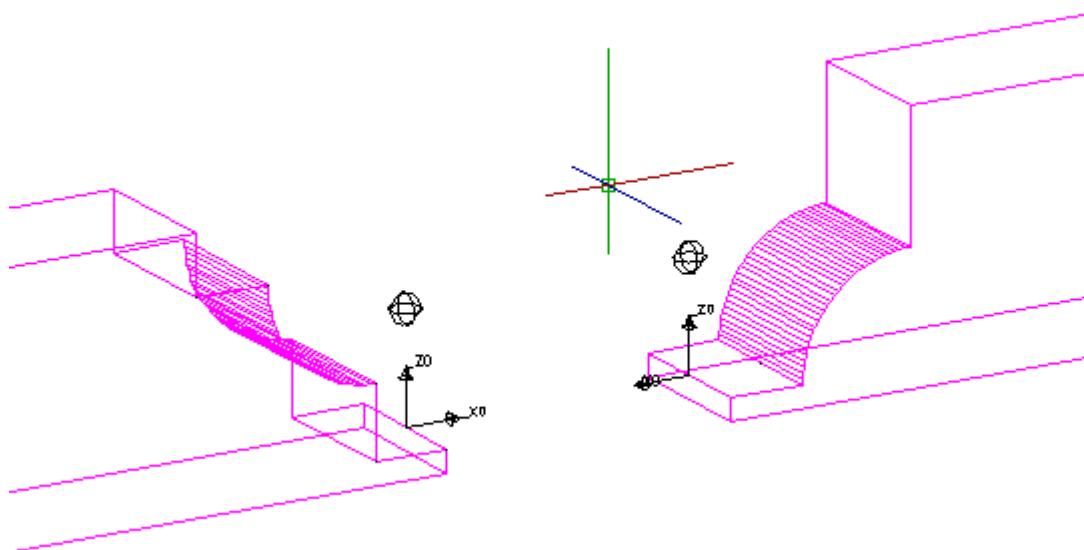
**ConvexConcaveProfile::setIsEndTool(int blsEndTool);**

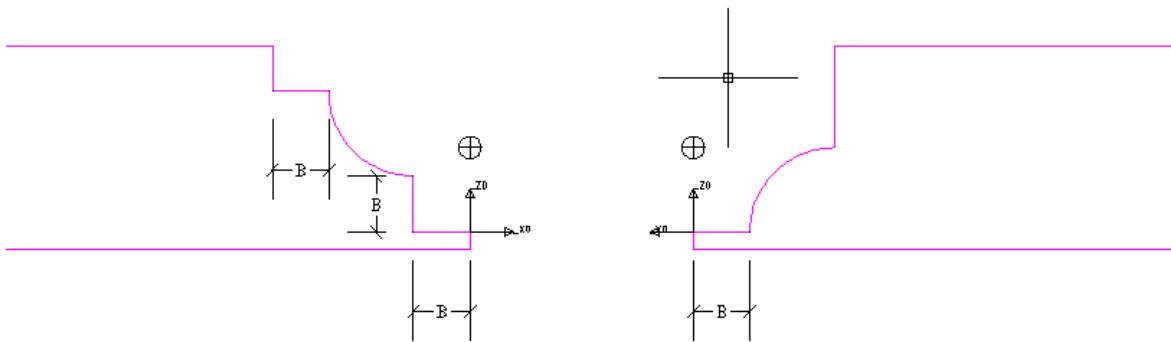
The nProfileType should be one of the following predefined variables of type int (see [AnalysedConvexConcaveProfile](#)):

**\_kACCPConvex, \_kACCPConcave**

---

[Example of E-type:





```

U(1, "mm");

PropDouble dZOffset(0,0,"offset in Z direction");
PropDouble dBackcut(2,U(210),"backcut");
PropDouble dRadius(3,U(150),"radius");

String arTypes[] = {T("Concave"),T("Convex")};
PropString pType(0,arTypes,"type");
int nType = _kACCPConcave;
if (pType==arTypes[1])
    nType = _kACCPConvex;

Point3d ptStart = _Pt0 + dZOffset*_z0;
_x0.vis(ptStart);
_z0.vis(ptStart);
ConvexConcaveProfile cp(ptStart, _x0, _z0, nType, dBackcut, dRadius);
Beam0.addTool(cp, _kStretchOnInsert);

--end example]

```

[Example of E-type illustrating the use of setIsEndTool(FALSE):



```

U(1, "mm");

PropDouble dZOffset(0,0,"offset in Z direction");

```

```

PropDouble dBackcut(2, U(210), "backcut");
PropDouble dRadius(3, U(150), "radius");

String arTypes[] = {T("|Concave|"), T("|Convex|")};
PropString pType(0, arTypes, "type");
int nType = _kACCPConcave;
if (pType==arTypes[1])
    nType = _kACCPConvex;

Point3d ptStart = _Pt0 + dZOffset*_Z0;
_X0.vis(ptStart);
_Z0.vis(ptStart);
ConvexConcaveProfile cp(ptStart, _X0, _Z0, nType, dBackcut, dRadius);
cp.setIsEndTool(FALSE);
_Beam0.addTool(cp);

—end example]

```

## 8.12 Cut

The cut tool constructor has one of the following formats:

```

Cut name(Point3d ptPlane, Vector3d vecNormal);
Cut name(Point3d ptPlane1, Point3d ptPlane2, Point3d ptPlane3, Point3d ptInOrOut, int
nThrowPointAway);
Cut name(Plane plane, Point3d ptInOrOut, int nThrowPointAway);

```

<b>ptPlane:</b>	a point on the cutting plane
<b>vecNormal:</b>	normal to the cutting plane, pointing in the direction of the part that is cutted away
<b>ptPlane1, 2, 3:</b>	three points defining the plane
<b>ptInOrOut:</b>	point indicating the part that remains or that is cut away
<b>nThrowPointAway&gt;0:</b>	<b>ptInOrOut</b> is indicating the part that is cut away
<b>nThrowPointAway&lt;=0:</b>	<b>ptInOrOut</b> is in the part that remains

To assure that the cut tool is not optimized away from the list of tools on a beam, the routine **setMustCut(**TRUE**)** must be called on the cut tool. The default value is set to **FALSE**.

```
Cut::setMustCut(TRUE);
```

See [General](#) for member functions applicable to all tools.

Other member methods (added hsbCAD20.0.114 and hsbCAD21.0.5)

```

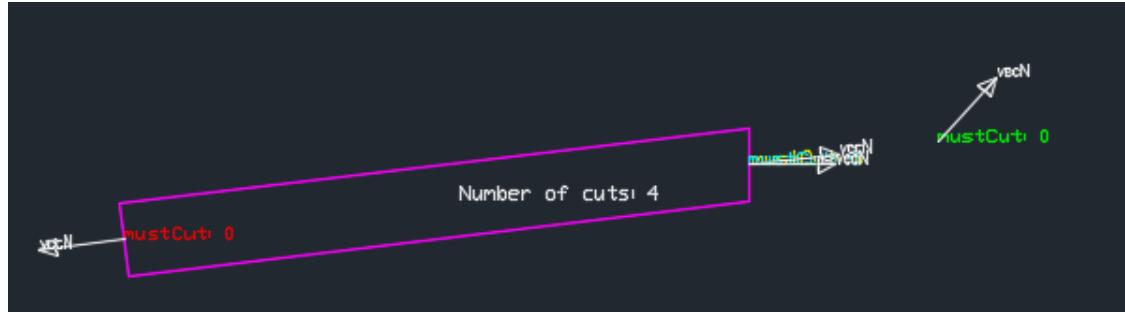
Point3d ptOrg() const; // point of cut plane
Vector3d normal() const; // outer normal to cut plane
int mustCut() const;

```

A copy of the Drill tools of a [GenBeam](#) can be retrieved with the function:

```
Drill[] GenBeam::getToolsStaticOfTypeCut() const;
```

[Example E-type illustrating the removal of specific static cuts:



```

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

GenBeam genbeam = _GenBeam[0];

String strRemoveAllCuts = T("|Remove all static cuts|");
addRecalcTrigger(_kContext, strRemoveAllCuts );
if (_bOnRecalc && _kExecuteKey==strRemoveAllCuts )
{
    int nCountRemoved = genbeam.removeToolsStaticOfType(Cut());
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

String strRemoveCut = T("|Remove third cut|");
addRecalcTrigger(_kContext, strRemoveCut );
if (_bOnRecalc && _kExecuteKey==strRemoveCut )
{
    Cut cutsAll[] = genbeam.getToolsStaticOfTypeCut();
    int nCountRemoved = 0;
    if (cutsAll.length() >= 3)
        nCountRemoved = genbeam.removeToolStatic(cutsAll[2]);
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

Cut cuts[] = genbeam.getToolsStaticOfTypeCut();

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
String strNum = "Number of cuts: " + cuts.length();
dp.draw(strNum , _Pt0, _XU, _YU, 1, 1);

for (int i=0; i<cuts.length(); i++)
{
    int nColor = i+1;
    dp.color(nColor);
}

```

```

Cut cut = cuts[i];
Point3d ptCut = cut.ptOrg();
Vector3d vecN = cut.normal();

vecN.vis(ptCut);

String str = "mustCut: " + cut.mustCut();
dp.draw(str, ptCut, _XU, _YU, 1, 1);
}

```

*—end example]*

## 8.13 Daddo

The daddo (nutz) tool constructor has one of the following formats:

```

Daddo name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Daddo name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth);
Daddo name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
    double dZFlag);

```

ptOrg:	origin point of the box defining the operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

To specify the dimensions of the tool, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the tool is the same as **vecY.length()\*dYHeight**. If **vecY** is a unit-length vector, then the **dYHeight** corresponds with the Height. If **dYHeight** equals 1, then the length of the vector expresses the height.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -**vecX**, -**vecY** and -**vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

The daddo has an **toolIndex** value, default 0.

```
Daddo::setToolIndex(int nToolIndex); // added since hsbCAD2017 build 21.0.73
```

The daddo has an overshoot type, which can be set separately by the function **setOverShoot**.

```
Daddo::setOverShoot(int nOverShoot);
```

Values of **nOverShoot** are:

**\_kNoOverShoot**: no overshoot

**\_kOverShoot:** overshoot

Important constraints for the daddo tool are:

- the direction that the tool enters the timber is the vecZ direction
- the vecX direction must coincide with the X axis of the timber

The solid body that represents this tool can be used by calling

**Body cuttingBody()** const;

See [General](#) for member functions applicable to all tools.

---

[Example T-type:]

```

Unit(1,"mm"); // use mm as unit in U() function from now on

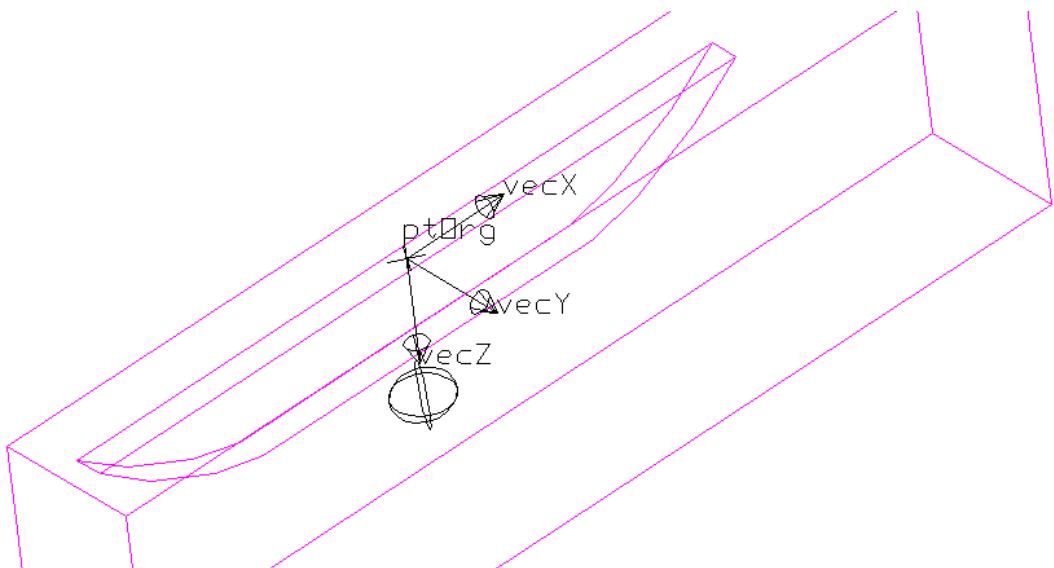
double dXW= U(500);
double dYH= U(20);
double dZD= U(60);

// rotate the coordinate system with the _Z0 axis
Vector3d vecY = _Y0;
Vector3d vecZ = _Z0; // _Z0 initially most aligned with _ZW => height direction
Vector3d vecX= vecY.crossProduct(vecZ); // perpendicular to vecY and vecZ
Point3d ptOrg = _Pt0 - 0.5*_H0*vecZ; // move point to surface

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Daddo dd(ptOrg,vecX,vecY,vecZ,dXW,dYH,dZD,0,0,1); // the ptOrg is not the center point of
the box, but shifted in -vecZ direction
dd.setOverShoot(_kNoOverShoot);
_Beam0.addTool(dd);

```



—end example]

## 8.14 DiagonalNotch

The diagonal log notch constructor has the following format:

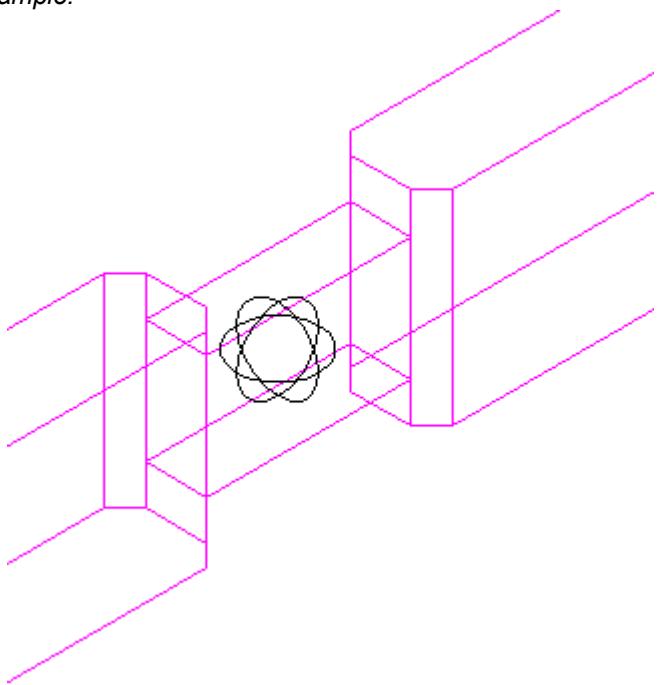
```
DiagonalNotch name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
    double dXWidth, double dYDepth, double dZHeightBeam, double dYWidthBeam,
    double dZDepthTop, double dZDepthBottom);
DiagonalNotch name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,
    double dXWidth, double dYPosDepth, double dYNegDepth, double dZHeightBeam,
    double dYWidthBeam, double dZDepthTop, double dZDepthBottom);
```

ptOrg:	point on axis of beam
vecX:	axis along this beam
vecY:	axis along other beam
vecZ:	upward direction
dXWidth:	dimension in vecX direction, typically equal to (dWidthBeamOther - 2*dDepthOther)
dYDepth:	depth of housing (in +vecY and -vecY direction)
dYPosDepth:	depth of housing (in +vecY direction)
dYNegDepth:	depth of housing (in -vecY direction)
dZHeightBeam:	beam height, in vecZ direction
dYWidthBeam:	beam width
dZDepthTop:	depth on top side, on vecZ side
dZDepthBottom:	depth on bottom side, on -vecZ side

See [General](#) for member functions applicable to all tools.

When presented as [AnalysedTool](#), it becomes an [AnalysedDiagonalNotch](#).

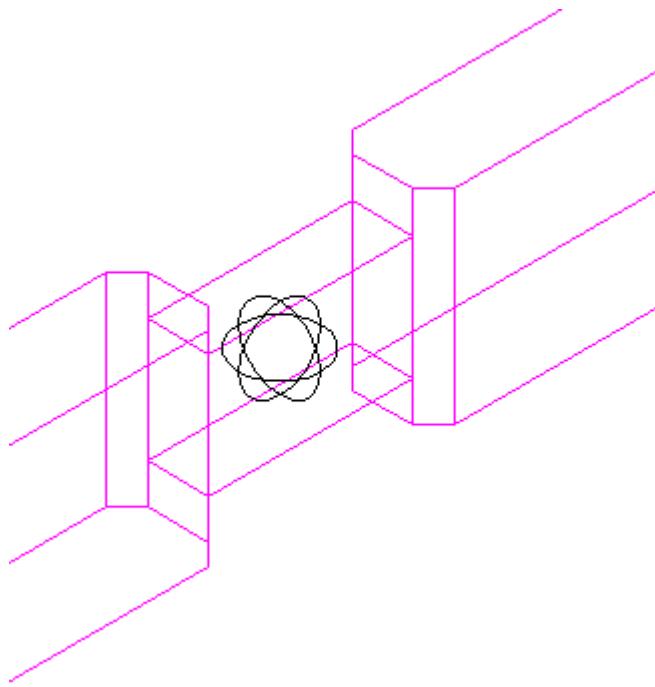
[Example:



```
U (l,"m m ");
DiagonalNotch dn(_Pt0,_X0,_Y0,_Z0,U (00),U (0),_H0,_W 0,U (0),U (0));
_Beam 0.addTool(dn);
```

—end example]

[Example E-type which (un)mystifies the LogNotch:



```

(1, "mm");

PropDouble dZHeightBeam(0, _H0, "Blok H");
PropDouble dYWidthBeam(1, _W0, "Blok W");
PropDouble dXWidthInterior(4, _W0, "XwidthInterior");
PropDouble dZDepthTop(5, (30), "ZDepthTop");
PropDouble dZDepthBottom(6, (40), "ZDepthBottom");
PropDouble dYPosDepth(8, (20), "YPosDepthHouse");
PropDouble dYNegDepth(3, (20), "YNegDepthHouse");

PropInt pAddDrills(3,1, "Add drills");
PropDouble pRadiusDrill(9, (2), "Radius drill");

if (_PtG.length() == 0)
    _PtG.append(_Pt0);

Point3d ptCBlok = _PtG[0];

Vector3d vecBlokX = _X0;
Vector3d vecBlokY = _Y0;
Vector3d vecBlokZ = _Z0;
vecBlokX.vis(ptCBlok);
vecBlokY.vis(ptCBlok);
vecBlokZ.vis(ptCBlok);

DiagonalNotch dn(ptCBlok, vecBlokX, vecBlokY, vecBlokZ,
    dXWidthInterior, dYPosDepth, dYNegDepth, dZHeightBeam,
    dYWidthBeam,
    dZDepthTop, dZDepthBottom);
_Beam0.addTool(dn);

```

```
double dX = dXWidthInterior;
double dY = dYWidthBeam - dYPosDepth - dYNegDepth;
double dZ = dZHeightBeam - dZDepthTop - dZDepthBottom;
Point3d ptOrg = ptCBlok + 0.5*(dYNegDepth-dYPosDepth)*vecBlokY +
0.5*(dZDepthBottom-dZDepthTop)*vecBlokZ;

Point3d ptYNeg, ptYNegF, ptYNegB;
Point3d ptYPos, ptYPosF, ptYPosB;
Point3d ptZNeg, ptZNegF, ptZNegB;
Point3d ptZPos, ptZPosF, ptZPosB;

ptYNeg = ptOrg - 0.5*dY*vecBlokY;
ptYNegF = ptYNeg + 0.5*dX*vecBlokX;
ptYNegB = ptYNeg - 0.5*dX*vecBlokX;

ptYPos = ptOrg + 0.5*dY*vecBlokY;
ptYPosF = ptYPos + 0.5*dX*vecBlokX;
ptYPosB = ptYPos - 0.5*dX*vecBlokX;

ptZNeg = ptOrg - 0.5*dZ*vecBlokZ;
ptZNegF = ptZNeg + 0.5*dX*vecBlokX;
ptZNegB = ptZNeg - 0.5*dX*vecBlokX;

ptZPos = ptOrg + 0.5*dZ*vecBlokZ;
ptZPosF = ptZPos + 0.5*dX*vecBlokX;
ptZPosB = ptZPos - 0.5*dX*vecBlokX;

ptYNeg.vis(1);
ptYNegF.vis(5);
ptYNegB.vis(5);

ptYPos.vis(1);
ptYPosF.vis(5);
ptYPosB.vis(5);

ptZNeg.vis(3);
ptZNegF.vis(4);
ptZNegB.vis(4);

ptZPos.vis(3);
ptZPosF.vis(4);
ptZPosB.vis(4);
```

*—end example]*

---

## 8.15 DimTool

The DimTool tool enables to attach information to the beam that will be used during generation of the shopdrawing of the beam. Such a package of information typically consists of one or more reference points, that will be used to compose a dimension line, when looking to the beam in a particular view direction. Additionally, some text might be specified also.

```
DimTool name(Point3d ptRef, Vector3d vecView);
```

ptRef:	The reference point that will be added to the dimension line.
vecView:	The view direction that determines the view in which this point will be added.

There are a number of member functions that can be called on the DimTool.

```
DimTool::addPoint(Point3d ptRef);
DimTool::addViewDirection(Vector3d vecView);
DimTool::addText(String strText);
```

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```
Unit(1, "mm"); // set units to use

DimTool dt(_Pt0, _Z0); // construct the tool with at least one point and one view direction

// add additional points, viewdirections and text.
dt.addPoint(_Pt0 + U(100)*_X0);
dt.addViewDirection(_Y0); // can also use _YW and _ZW for reference in the world CS
dt.addText(scriptName() + " here");

_Beam0.addTool(dt); // add the tool as a normal tool to the beam

—end example]
```

## 8.16 Double cut

The double cut consists of 2 normal cuts of which the result is the unite of the bodies of the single cuts. The constructor has the following formats:

```
DoubleCut name(Point3d pt1, Vector3d vec1, Point3d pt2, Vector3d vec2);
DoubleCut name(Cut ct1, Cut ct2);
```

pt1: point on cut plane 1

---

```

vec1:    normal of cut plane 1
pt2:    point on cut plane 2
vec2:    normal of cut plane 2
ct1:    cut plane 1
ct2:    cut plane 2

```

**Important:** For the DoubleCut to generate proper Hundegger machine code, there are a number of limitations:

- Both cuts need to be at the same side of the beam.
- In case the cuts are parallel to the axes of the beam, the angles to the reference face need to be in the range -89,89

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```

Unit(1,"mm"); // use mm as unit in U() function from now on

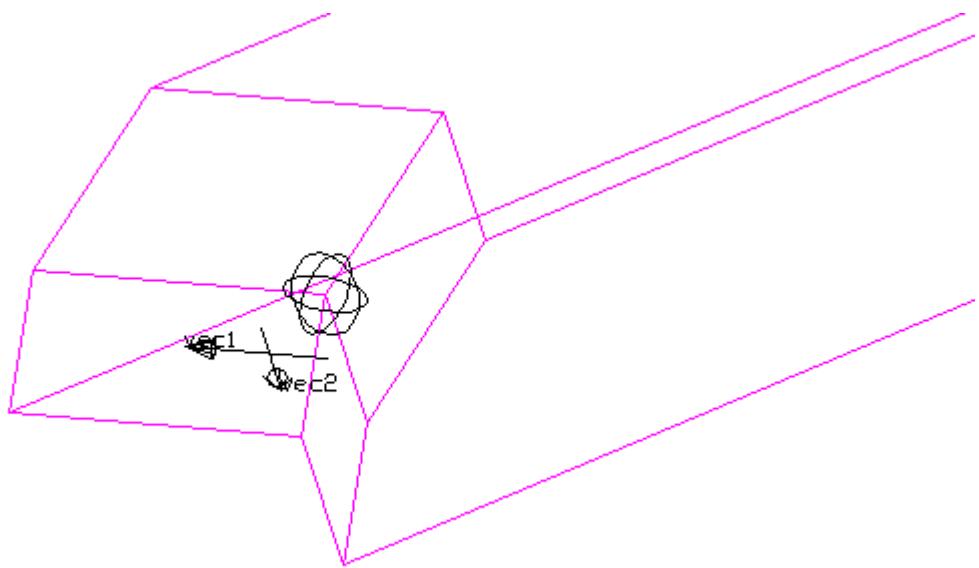
Vector3d vec1 = _X0+_Z0;
Vector3d vec2 = _X0-_Z0;

vec1.vis(_Pt0+U(20)*_Y0.crossProduct(vec1)-_Y0));
vec2.vis(_Pt0+U(20)*(-_Y0.crossProduct(vec2)-_Y0));

DoubleCut dc( _Pt0, vec1 , _Pt0, vec2 ); // constructor with 2 points and 2 vectors
_Beam0.addTool( dc, 1);

Vector3d vec1b = vec1 + _Y0;
Vector3d vec2b = vec2 + _Y0;
Cut c1( _Pt0, vec1b );
Cut c2( _Pt0, vec2b );
DoubleCut dc2( c1, c2); // constructor with 2 cuts
_Beam0.addTool( dc2, 0); // add with parameter 0, do not remove other end tools

```



—end example]

[Example E-type:

```

Unit(1,"mm"); // use mm as unit in U() function from now on
{
    Vector3d vec1 = _Y0+_Z0;
    Vector3d vec2 = _Y0-_Z0;
    Point3d pt = _Pt0+U(20)*_Y0;

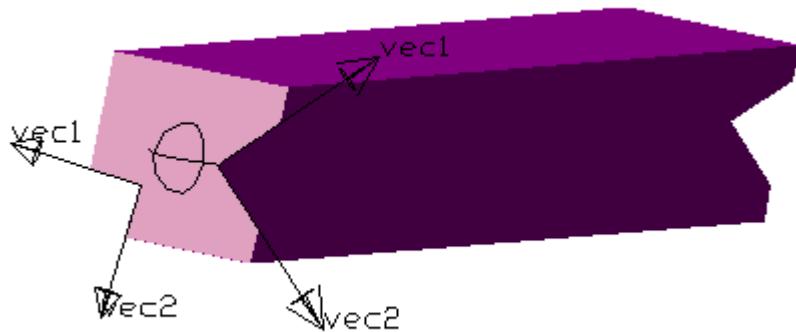
    vec1.vis(pt);
    vec2.vis(pt);

    DoubleCut dc( pt, vec1 , pt, vec2 ); // constructor with 2 points and 2 vectors
    _Beam0.addTool( dc, 0);
}
{
    Vector3d vec1 = -_Y0+0.1*_Z0;
    Vector3d vec2 = -_Z0-0.1*_Y0;
    Point3d pt = _Pt0-U(20)*_Y0-U(20)*_Z0;

    vec1.vis(pt);
    vec2.vis(pt);

    DoubleCut dc( pt, vec1 , pt, vec2 ); // constructor with 2 points and 2 vectors
    _Beam0.addTool( dc, 0);
}

```



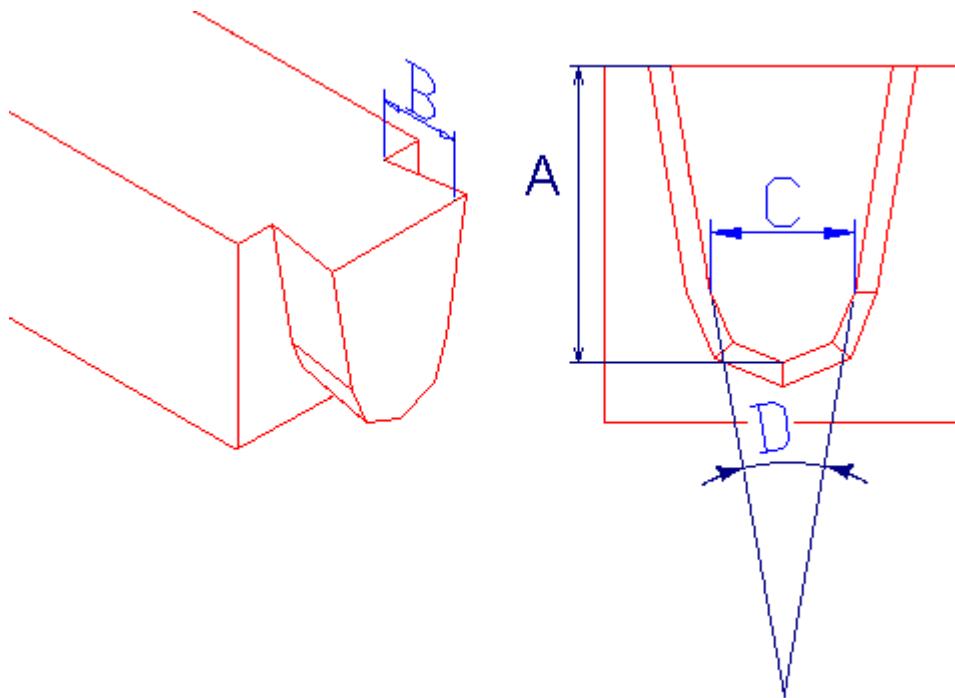
*—end example]*

## 8.17 Dove tail connection

The dove tail constructor has the following format:

```
Dove name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
dXWidth, double dYHeight, double dZDepth, double dAlfa, int nEndType);
```

ptOrg:	point opposite from top of dove, on the female surface
vecX:	axis in width direction
dXWidth (C):	width on female surface, of the rounded top
vecY:	height direction
dYHeight (A):	height on female surface, from width location to origin
vecZ:	depth direction
dZDepth (B):	depth of the dove tail
dAlfa (D):	top angle in degrees



The dove tail with the same geometrical parameters can be used for both the male and the female side. The only parameter that differs is the last one in the constructor. To change this value on an existing dove, the `setEndType` function can be used.

```
setEndType(int nEndType);
```

Values of `nEndType`:

- `_kFemaleEnd`: done at end side of beam
- `_kFemaleSide`: not done at end of beam
- `_kMaleEnd`: male one, done at the end side of beam

The following 3 functions are also members of Dove.

```
setDepthCorrection(int bSet); // default TRUE, set to FALSE to prevent the depth correction
calculation during Hgg generation
setContinuousMortise(int bSet);
setAddKeyhole(int bSet);
setFlatWidth(double dWidth);
```

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```
Unit(1,"mm"); // use mm as unit in U() function from now on
```

```

double dXWidth= U(40);
double dYHeight= U(80);
double dZDepth= U(20);
double dAlfa= 20; // in degrees

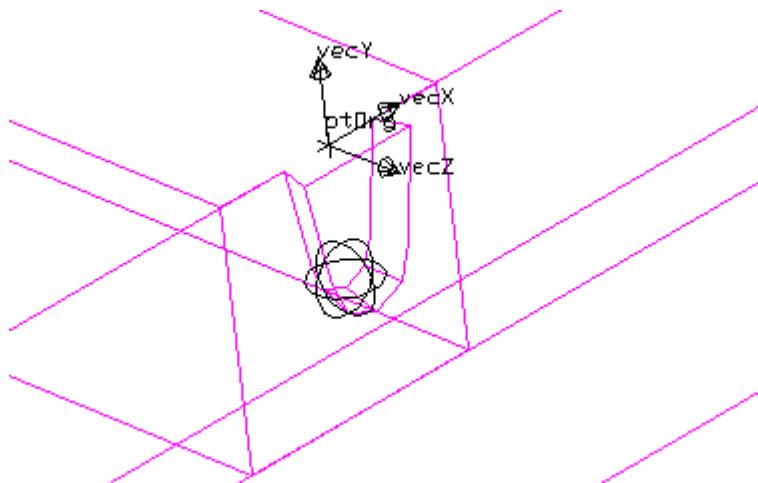
// rotate the coordinate system with the _Z0 axis
Vector3d vecZ = _Z1; // _Z1 is always normal to contact surface => depth direction
Vector3d vecY = _Z0; // _Z0 initially most aligned with _ZW => height direction
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and vecZ
Point3d ptOrg = _Pt0 +0.5*_H0*_Z0; // move _Pt0 half beam height up

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Dove dv(ptOrg,vecX,vecY,vecZ,dXWidth,dYHeight,dZDepth, dAlfa, _kFemaleSide);
_Beam1.addTool(dv); // female beam

dv.setEndType(_kMaleEnd);
_Beam0.addTool(dv,1); // is added with parameter 1: endtool active: will remove all other
endtools during insert

```



*[end example]*

## 8.18 Drill

The drill tool constructor has one of the following formats:

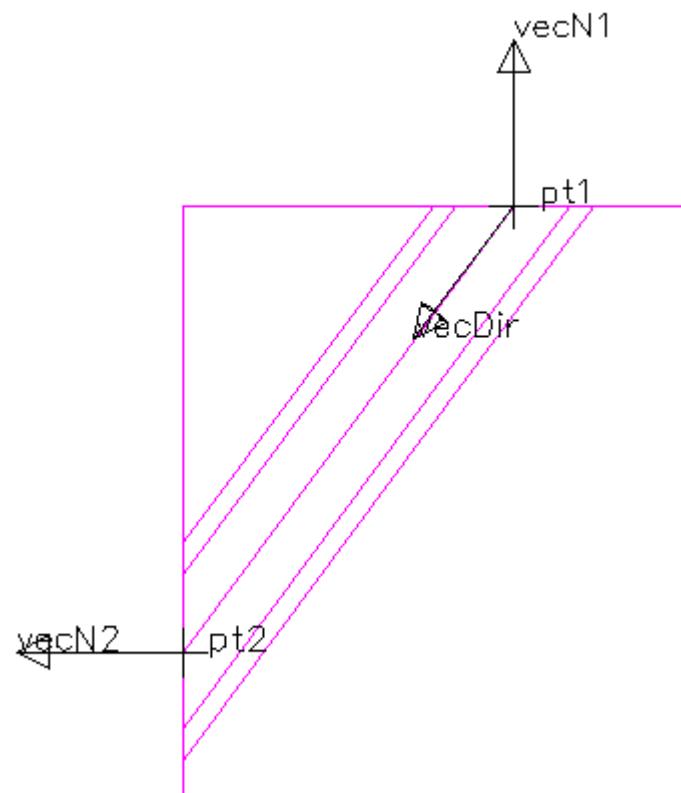
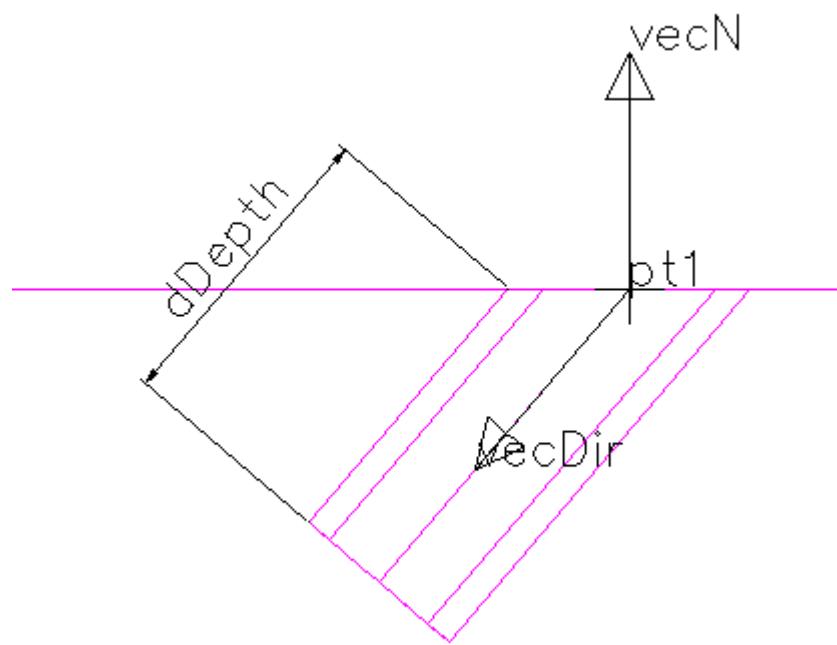
```

Drill name(Point3d pt1, Point3d pt2, double dRadius);
Drill name(Point3d pt1, Point3d pt2, double dRadius, Vector3d vecN1, Vector3d vecN2);
Drill name(Point3d pt1, Vector3d vecDir, double dDepth, double dRadius);

```

**Drill** name(**Point3d** pt1, **Vector3d** vecDir, **double** dDepth, **double** dRadius, **Vector3d** vecN);

pt1: point on the axis line of the drilling, to start drilling from  
pt2: other point on the axis line of the drilling  
dRadius: radius of the drill  
vecDir: vector along the axis of the drill  
dDepth: dDepth is multiplied with vecDir to determine the displacement  
vector  
vecN: normal vector of the surface of pt1. If vecN is supplied, the depth is  
measured from the surface, ensuring that the drill has at least this  
depth (see image below)  
vecN1: normal vector of the surface of pt1 (see image below)  
vecN2: normal vector of the surface of pt2



After the radius has been specified inside the constructor, the radius can be altered by calling `setRadius`.

---

```
Drill::setRadius(double dRadius);
```

To fix the direction that the cnc-machine will use, one must call the setUseDirection with argument TRUE. By default the value is set to FALSE which means that hsbCad determines the drilling direction. If the value of useDirection is set to TRUE then the drill will move from pt1 to pt2.

```
Drill::setUseDirection(int bSet);
```

The solid body that represents this tool can be used by calling

```
Body cuttingBody() const;
```

Other member methods (added hsbCAD20.0.76 and hsbCAD19.1.103)

```
Point3d ptStart() const;  

Point3d ptEnd() const;  

double dDiameter() const;  

double dRadius() const;  

int bUseThisDirection() const;
```

Since hsbcad v20.0.88 the Drill can have an inner diameter set, in which case it represents a hollow cylindrical cutout. It can be used for a shear plate connector.

```
double innerCylinderDiameter() const; // the default value is 0, which indicates it is not set.  

void setInnerCylinderDiameter(double dDiam); // if the dDiam is bigger than the dDiameter of the  

drill, it is considered a normal drill without an inner part.
```

See [General](#) for member functions applicable to all tools.

A copy of the Drill tools of a [GenBeam](#) can be retrieved with the function:

```
Drill[] GenBeam::getToolsStaticOfTypeDrill() const;
```

[Example E-type:

```
Unit(1,"mm");  

Drill dr1(_Pt0+0.5*_W0*_Y0, _Pt0-0.5*_W0*_Y0, U(20));  

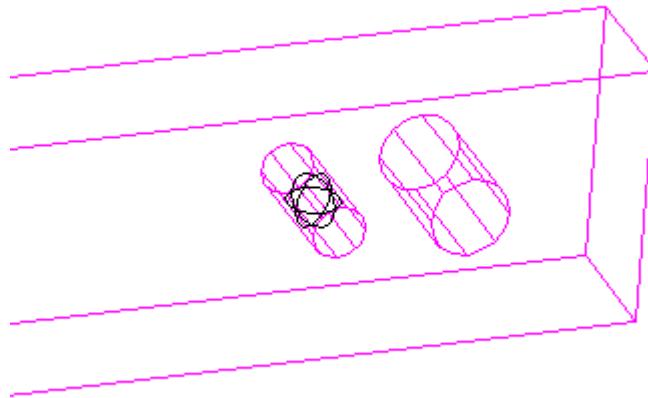
_Beam0.addTool(dr1);  
  

dr1.setRadius(U(30)); // reuse the same drill, but change the radius  

CoordSys move; move.setToTranslation(U(100)*_X0);  

dr1.transformBy(move); // move the drill  

_Beam0.addTool(dr1);
```



—end example]

[Example E-type illustrating the removal of specific static drills:

```

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

GenBeam genbeam = _GenBeam[0];

String strRemoveAllDrills = T("|Remove all static drills|");
addRecalcTrigger(_kContext, strRemoveAllDrills );
if (_bOnRecalc && _kExecuteKey==strRemoveAllDrills )
{
    int nCountRemoved = genbeam.removeToolsStaticOfType(Drill());
    //int nCountRemoved = genbeam.removeToolsStatic(Drill()); // obsolete
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

String strRemoveFirstDrill = T("|Remove first drill|");
addRecalcTrigger(_kContext, strRemoveFirstDrill );
if (_bOnRecalc && _kExecuteKey==strRemoveFirstDrill )
{
    Drill drllsAll[] = genbeam.getToolsStaticOfTypeDrill();
    int nCountRemoved = 0;
    if (drllsAll.length() > 0)
        nCountRemoved = genbeam.removeToolStatic(drllsAll[0]);
    reportMessage("\nNumber of tools removed: " + nCountRemoved);
}

Drill drlls[] = genbeam.getToolsStaticOfTypeDrill();

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
String strNum = "Number of drills: " + drlls.length();
dp.draw(strNum , _Pt0, _XU, _YU, 1, 1);

```

```

for (int i=0; i<drlls.length(); i++)
{
    int nColor = i+1;
    dp.color(nColor);

    Drill drll = drlls[i];
    Point3d ptStart = drll.ptStart();
    Point3d ptEnd = drll.ptEnd();

    Body bd(ptStart, ptEnd, drll.dRadius());
    dp.draw(bd);

    String str = "Diam: " + drll.dDiameter() + " UseDir: " +
drll.bUseThisDirection();
    dp.draw(str, ptStart, _xu, _yu, 1, 1);
}

—end example]

```

## 8.19 ExtrProfileCut

The ExtrProfileCut tool allows to add a beamcut with the shape of an extrusion profile. It does not necessarily cut away a rectangular part. It cuts away the shape of the extrusion profile, and with the tools, defined inside the extrusion profile definition. The extrusion profile is positioned in the vecY, vecZ plane. This means that the coordinate system would correspond with the coordinate system of an imaginary beam with that profile.

The ExtrProfileCut tool constructor has the following format:

```

ExtrProfileCut name(String strExtrProfileName, Point3d ptOrg, Vector3d vecX, Vector3d
vecY, Vector3d vecZ);
ExtrProfileCut name(String strExtrProfileName, Point3d ptOrg, Vector3d vecX, Vector3d
vecY, Vector3d vecZ, double dXDepth, double dYSize, double dZSize);
ExtrProfileCut name(String strExtrProfileName, Point3d ptOrg, Vector3d vecX, Vector3d
vecY, Vector3d vecZ, double dXDepth, double dYSize, double dZSize, double
dXFlag, double dYFlag, double dZFlag);

```

strExtrProfileName:	name of the <a href="#">ExtrProfile</a>
ptOrg:	origin point of the box defining the operation
vecX:	axis in depth direction
vecY:	profile X direction
vecZ:	profile Y direction
dXDepth:	depth scale factor
dYSize:	profile X scale factor
dZSize:	profile Y scale factor

To specify the dimensions of the tool, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the tool is the same as `vecY.length()*dysize`. If `vecY` is a unit-length vector, then the `dysize` corresponds with the height. If `dysize` equals 1, then the length of the vector expresses the height.

The flags `dXFlag`, `dYFlag` and `dZFlag`, specify the position of the `ptOrg` inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point `ptOrg` is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the `-vecX`, `-vecY` and `-vecZ` direction. So the flags are actually the relative coordinates of the center point of the box, in the `vecX`, `vecY` and `vecZ` coordinate system with origin `ptOrg`.

Important constraints for the `ExtrProfileCut` tool are:

- if the extrusion profile is not scalable, then the `dysize` and `dzsize` will have no effect on the scaling, only on the positioning of the centerpoint.
- the cross product of `vecY` and `vecZ` is taken as the extrusion direction, not `vecX`
- the cuttingBody has in fact the double `dXDepth`.

The solid body that represents this tool can be used by calling

**Body** `cuttingBody()` const;

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```
Unit(1, "mm"); // use mm as unit in U() function from now on

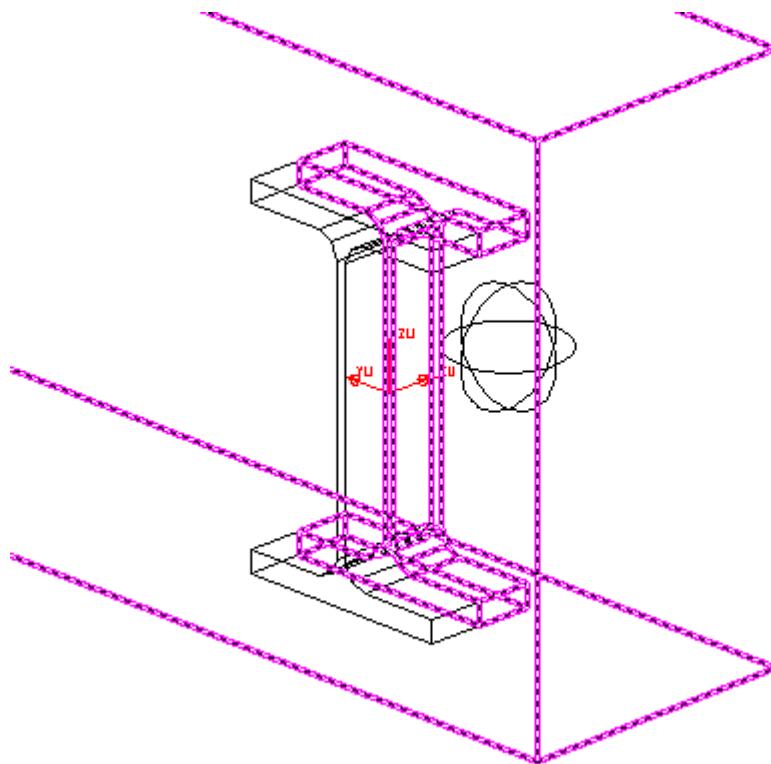
// declare a property with the names of the extrusion profiles
PropString pExtrProfile(0, ExtrProfile().getAllEntryNames(),
"Extrusion profile name");

Point3d ptIns = Pt0-0.5*w0*y0; // calculate point on surface of
beam

double dDepth = U(20);
double dSizeX = U(100);
double dSizeY = U(100);

// define a cut tool with the internal coordinate system of the tsl,
initialized with the UCS at time of insertion.
ExtrProfileCut epc(pExtrProfile, ptIns, xu, yu, zu, dDepth,
dSizeX, dSizeY, 1, 0, 0);
Beam0.addTool(epc);

xu.vis(ptIns, 1);
yu.vis(ptIns, 1);
zu.vis(ptIns, 1);
epc.cuttingBody().vis();
```



*[end example]*

## 8.20 FreeProfile

The FreeProfile tool enables to cut out any polyline shape from a beam. The constructor has the following format:

```
FreeProfile name(PLine plCutAway, Point3d ptCutAway);
FreeProfile name(PLine plCutAway, Point3d[] arPntToClosePoly);
FreeProfile name(PLine plCutAway, int nSide); // since V22.0.100
```

**plCutAway:** polyline defining the area that will be cut away from the beam. The normal of the polyline is used to extrude the body that will be extracted.  
**ptCutAway:** point used to close the polyline, or to identify the part that is cut away.  
**arPntToClosePoly:** optional list of points used to close the polyline, or to identify the part that is cut away.  
**nSide:** side to cut away. The normal of the plCutAway is used as up direction reference, walking along the pline from start to end.

For the first 2 constructors, the part that is cut away from the solid is a closed pline extrusion. If the plCutAway is not closed, then it will be closed with the arPntToClosePoly, to construct a closed polyline by which the body of the beam or sheeting will be cut. If the plCutAway is closed, the first

point of arPntToClosePoly indicates the part that will be cut away. If the point lies inside or on the polyline, the internal of the polyline will be thrown away. If the point lies outside the polyline, the inner part is kept.

For the constructors with nSide, the solid operation that is performed can either be the pline with a mill diameter, or the closed pline extrusion, depending on the state of the bSolidPathOnly. The nSide is `_kLeft` (-1), `_kCenter` (0) or `_kRight`(1).

The plCutAway PLine will be used to generate the path that the machine will trace. It is important to keep this to the part that is really cutting the GenBeam. If not, the machine generation might not be as efficient as it could be.

The type of mill that is used during the machining can be specified by the setCncMode routine. The default value is `_kFingerMill`, also 0. The other values are `_kUniversalMill` (1), and `_kVerticalFingerMill` (2). For the milling with a special head, the `_kExtrusionProfile1` (-2) and `_kExtrusionProfile2` (-3) can be used. Basically any integer value can be set. This value is passed on as export.

**FreeProfile::setCncMode(int nMode);**

The depth, with default value 0 means it is cut all the way through. But one can limit the depth to a certain value. In that case, the PLine needs to be defined on the surface of the beam.

**FreeProfile::setDepth(double dValue);**

The solid body that represents this tool can be used by calling

**Body cuttingBody() const;**

If the solidPathOnly is set to TRUE, only the mill thickness is cut away on the solid. Available since V22.0.100.

**FreeProfile::setSolidPathOnly(int nSet);** // default FALSE means complete area is removed.

Doing the operation on the machine assumes that cutting the path only, will cut off the entire part loose, and as such the complete part does not need to be machined. But if the machinePathOnly is set to FALSE, the complete part is intended to be machined.

**FreeProfile::setMachinePathOnly(int nSet);** // default TRUE means only path is machined. Available since V22.0.100.

The mill diameter, with default value 0 means the mill diameter is sourced from the cncMode, in particular sourced from the Hundegger values in the HSB\_SETTINGS. But if the value is different from 0, it is used for the solid operation. The actual diameter on the machine depends on the cncMode mapping.

**FreeProfile::setSolidMillDiameter(double dValue);** // Available since V22.0.100.

The following methods are available since V22.0.100. To query existing FreeProfiles, the method `GenBeam::getToolsOfTypeFreeProfile` can be used. It returns a copy of the FreeProfile tools attached to the GenBeam.

From FreeProfile tools the following information can be retrieved. Often a GenBeam is required to calculate the info.

```
PLine plDefining(GenBeam genBeam); // plInternal moved down over depth
PLine plBvnMillHead(GenBeam genBeam); // offset plInternal in getMillSidePolyNormal
direction over 0.5 millDiameter
PLine[] plCuts(GenBeam genBeam); // pl
Vector3d vecZOutwards(GenBeam genBeam); // vector pointing outwards of beam
```

```

int millSide(GenBeam genBeam); // millSide with vecZOutwards as up direction walking the
plDefining
double depth(GenBeam genBeam); // return the set depth, but in case of 0, return the
calculated required depth.

PLine plInternal(); // internal stored pline (internal use only)
int getMillSidePolyNormal(); // millSide with plInternal.normal() as up direction walking the
plInternal (internal use only)
double millDiameter(); // diameter in use, retrieved from setSolidMillDiameter or in case that
one is 0, from nCncMode.
int setMillSidePolyNormal(int nSet); // (added 22.1.132, 23.5.5) set millSide with
plInternal.normal() as up direction walking the plInternal (internal use only)

```

If the cutDefiningAsOne is set to TRUE, the defining polyline will not be split into parts by the bounding box of the beam to get to the plCuts. Instead the complete plDefining will be kept to cut. In this case the plCuts will just contain a copy of plDefining. Available since V23.4.18.

**FreeProfile**::setCutDefiningAsOne(**int** nSet); // default FALSE means plDefining is split into
parts in plCuts.

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```

String arSCncMode[] = {"Finger Mill", "Universal Mill", "Vertical
Finger Mill"};
int arNCncMode[] = {_kFingerMill, _kUniversalMill,
_kVerticalFingerMill };
PropString pCncMode(0, arSCncMode, "Cnc mode");
int nCncMode = arNCncMode[arSCncMode.find(pCncMode, 0)];

U(1, "mm");

PLine pline(_z0); // set normal of polyline
Point3d pnt1 = _Pt0+5*_w0*_y0 - U(500)*_x0; // point on the left
side, well above the surface
pline.addVertex(pnt1);
Point3d pnt2 = _Pt0 - U(500)*_x0; // point on the left side, on the
axis of the beam
pline.addVertex(pnt2);
Point3d pnt3 = pnt2 + U(1000)*_x0; // point on the right side, also
on the axis of the beam
pline.addVertex(pnt3, -0.2); // added with bulge factor
Point3d pnt4 = pnt1 + U(1000)*_x0; // point above the surface, but on
the right side
pline.addVertex(pnt4);

Point3d pntClosing = _Pt0 + 50*_w0*_y0; // point far away, defining
the part that is cutted away

FreeProfile fp(pline, pntClosing );
fp.setCncMode(nCncMode);

```

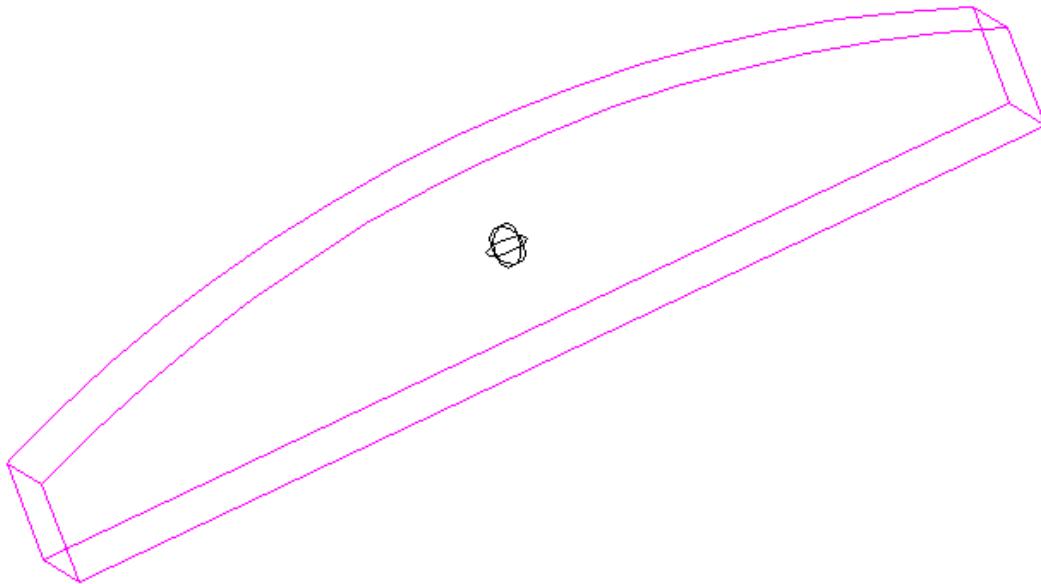
---

```

_Beam0.addTool(fp);

// also add 2 cuts, left and right
Cut ct1(pnt1,-_x0);
_Beam0.addTool(ct1,1);

Cut ct2(pnt4,_x0);
_Beam0.addTool(ct2,1);
```



*—end example]*

*[Example E-type:*

```

U(1,"mm");

String arSCncMode[] = {"Finger Mill","Universal Mill","Vertical
finger mill"};
int arNCncMode[] = {_kFingerMill, _kUniversalMill,
_kVerticalFingerMill };
PropString pCncMode(0,arSCncMode,"Cnc mode");
int nCncMode = arNCncMode[arSCncMode.find(pCncMode,0)];

PropDouble dDepth(0,U(50,2),T("Depth"));

if (_bOnInsert) {
    showDialogOnce();
    Beam.append(getBeam());
    PLine plineI = getEntPLine("Select closed polyline.").getPLine();
    Map.setPLine("pl", plineI, _kRelative); // relative will move it
with the tsl instance
    Point3d pntsPl[] = plineI.vertexPoints(FALSE);
    if (pntsPl.length()==0) eraseInstance(); // do not add this one
    else _Pt0 = pntsPl[0]; // 0 is a valid index
```

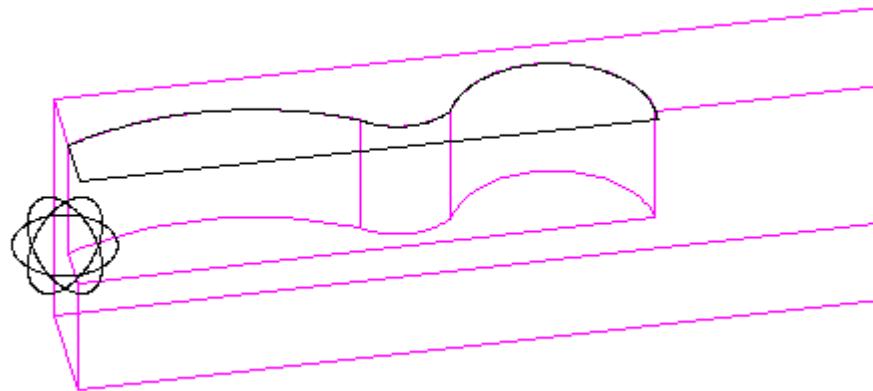
```

    return;
}

PLine pline = _Map.getPLine("pl");
pline.vis();

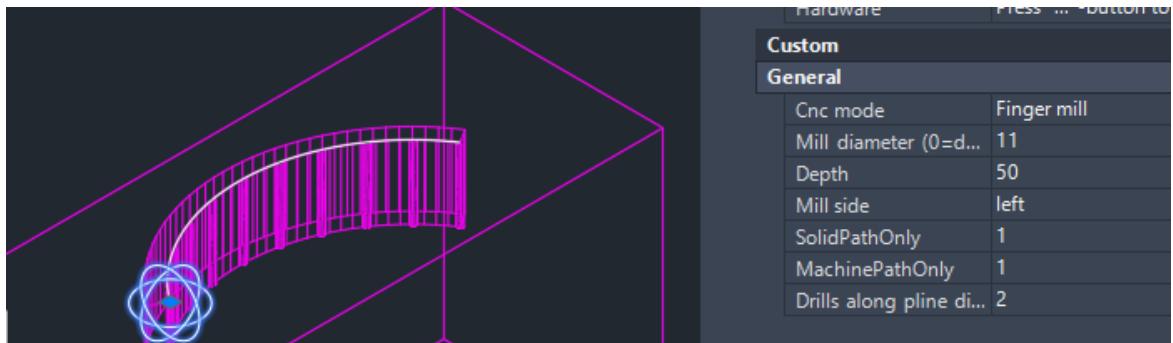
Point3d pnts[0];
FreeProfile fp(pline, pnts );
fp.setCncMode(nCncMode);
fp.setDepth(dDepth);
_Beam0.addTool(fp);

```



—end example]

[Example E-type:



```

(1,"mm");

String arSCncMode[] = {T("|No mill|"), T("|Finger mill|"), T("|Universal mill|"),
Universal mill|"), T("|Vertical finger mill|"), T("ToolIndex 3"),
T("ToolIndex 4"), T("ToolIndex 5"), T("ToolIndex 6"), T("ToolIndex 7")};
int arNCncMode[] = {-1,_kFingerMill, _kUniversalMill,
_kVerticalFingerMill,3,4,5,6,7 };
PropString pCncMode(0,arSCncMode, T("|Cnc mode|"));
int nCncMode = arNCncMode[arSCncMode.find(pCncMode,0)];

PropDouble dDiameter(1,u(0),T("|Mill diameter (0=default)|"));

```

```
PropDouble dDepth(0,U(50,2),T("|Depth|"));

String arSMillSide[] = {T("|do not use side|"), T("|left|"), T("|center|"), T("|right|")};
int arNMillSide[] = {-2,-1, 0, 1};
PropString pMillSide(1,arSMillSide, T("|Mill side|"));
int millSide = arNMillSide[arSMillSide.find(pMillSide,0)];

int nPathOnlys[]={0,1};
PropInt nPathOnly(1, nPathOnlys, T("|SolidPathOnly|")); // default is FALSE
PropInt nMachinePathOnly(2, nPathOnlys, T("|MachinePathOnly|"),
1); // default is TRUE

PropDouble dDrillDiam(2,U(0),T("|Drills along pline diameter (0=no
drills)|"));

if (_bOnInsert)
{
    showDialogOnce();
    _Beam.append(getBeam());

    EntPLine ent = getEntPLine(T("|Select polyline|"));
    _Entity.append(ent);

    PLine plineEnt = ent.getPLine();

    millSide = arNMillSide[arSMillSide.find(pMillSide,0)];
    if (millSide == -2)
        _PtG.append(getPoint(T("|Select point to close free profile
or to define part to cut|")));
}

return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}
EntPLine entPl = (EntPLine)_Entity[0];
if (!entPl.bIsValid())
{
    eraseInstance();
    return;
}

setDependencyOnEntity(entPl);

PLine pline = entPl.getPLine();
Point3d pntsPl[] = pline.vertexPoints(FALSE);
if (pntsPl.length()==0)
```

```

{
    eraseInstance();
    return;
}
else _Pt0 = pntsPl[0]; // 0 is a valid index

pline.vis();
Vector3d vecNPl = pline.coordSys().vecZ();
vecNPl.vis(_Pt0, 1);

FreeProfile fp;
if (millSide == -2)
    fp = FreeProfile(pline, _PtG);
else
    fp = FreeProfile(pline, millSide);

if (nPathOnly == 1)
    fp.setSolidPathOnly(nPathOnly);

if (nMachinePathOnly == 0)
    fp.setMachinePathOnly(nMachinePathOnly);

if (abs(dDiameter) > 0.0001)
    fp.setSolidMillDiameter(dDiameter);

fp.setCncMode(nCncMode);
fp.setDepth(dDepth);
_Beam0.addTool(fp);

if (dDrillDiam > 0.0001)
{
    double dDepthInUse = fp.depth(_Beam0);
    Vector3d vecToolEntry = -fp.vecZOutwards(_Beam0);
    PLine plPnts = pline;
    plPnts.convertToLineApprox(U(1));
    Point3d pts[] = plPnts.vertexPoints(TRUE);
    for (int p = 0; p < pts.length(); p++)
    {
        Point3d p1 = pts[p];
        Point3d p2 = p1 + dDepthInUse * vecToolEntry;
        p1.vis();
        p2.vis();

        Drill dr(p1, p2, dDrillDiam);
        _Beam0.addTool(dr);
    }
}

```

*—end example]*

[Example O-type with insert

```
U(1, "mm");
```

```
if (_bOnInsert)
{
    _Entity.append(getEntPLine());
    _GenBeam.append(getBeam());
    return;
}

int bInputOk = TRUE;
if (_Entity.length() == 0 || _GenBeam.length() == 0)
{
    eraseInstance();
    return;
}

GenBeam gb = _GenBeam[0];
EntPLine entPLine = (EntPLine)_Entity[0];
if (!entPLine.bIsValid())
{
    eraseInstance();
    return;
}

PLine pline = entPLine.getPLine();
Point3d ptLoc = pline.ptStart();

_pt0 = ptLoc;
setMarbleDiameter(0.1 * pline.length());
setDependencyOnEntity(entPLine);

PropDouble pDepth(0, U(10), T("|Depth|"));
PropInt doSolid(0, 1, T("|Do solid|"));
String arSMillSide[] = { T("|left|"), T("|center|"), T("|right|") };
int arNMillSide[] = { _kLeft, _kCenter, _kRight };
PropString pMillSide(1,arSMillSide, T("|Mill side|"));
int millSide = arNMillSide[arSMillSide.find(pMillSide,0)];

// get point on surface of beam
Vector3d vecOut = ptLoc - gb.ptCen();
vecOut = gb.vecD(vecOut);

Vector3d vecDir = pline.coordSys().vecZ();
if (vecDir.dotProduct(vecOut) > 0)
    vecDir *= -1;

pline.close();
FreeProfile tool(pline, millSide);
tool.setDoSolid(doSolid);
tool.setDepth(pDepth);

if (!doSolid)
{
    Body body = gb.envelopeBody();
```

```

    body.intersectWith(tool.cuttingBody());
    Point3d pnts[] = body.allVertices();
    tool.setPosnumCompareVertices(pnts);
    reportMessage("\n" + scriptName() + " added " + pnts.length() + " posnum compare vertices");
}

gb.addTool(tool);

—end example]

```

## 8.21 FreeText

The FreeText defines a text freely positioned on a GenBeam surface. The FreeText constructor has the following format:

```
FreeText name(String strText, CoordSys csOrient, double dTextHeigth, int xPos, int yPos);
```

csOrient: location and orientation of the text on the surface. Text is in vecX or -vecX direction, with vecY pointing in text height. VecZ is pointing genbeam surface inwards. vecY and vecZ are always honoured, and presented readable on the surface. So if vecX is the read direction, xPos corresponds to its meaning. Mirrored text will also be readable on the surface, and presented in the same rectangle.

strText: the text

textHeight: text height.

xPos, yPos: the position of the origin relative to the text

Values of xPos:

<b>_kLeft:</b>	-1
<b>_kCenter:</b>	0 (default value)
<b>_kRight:</b>	1

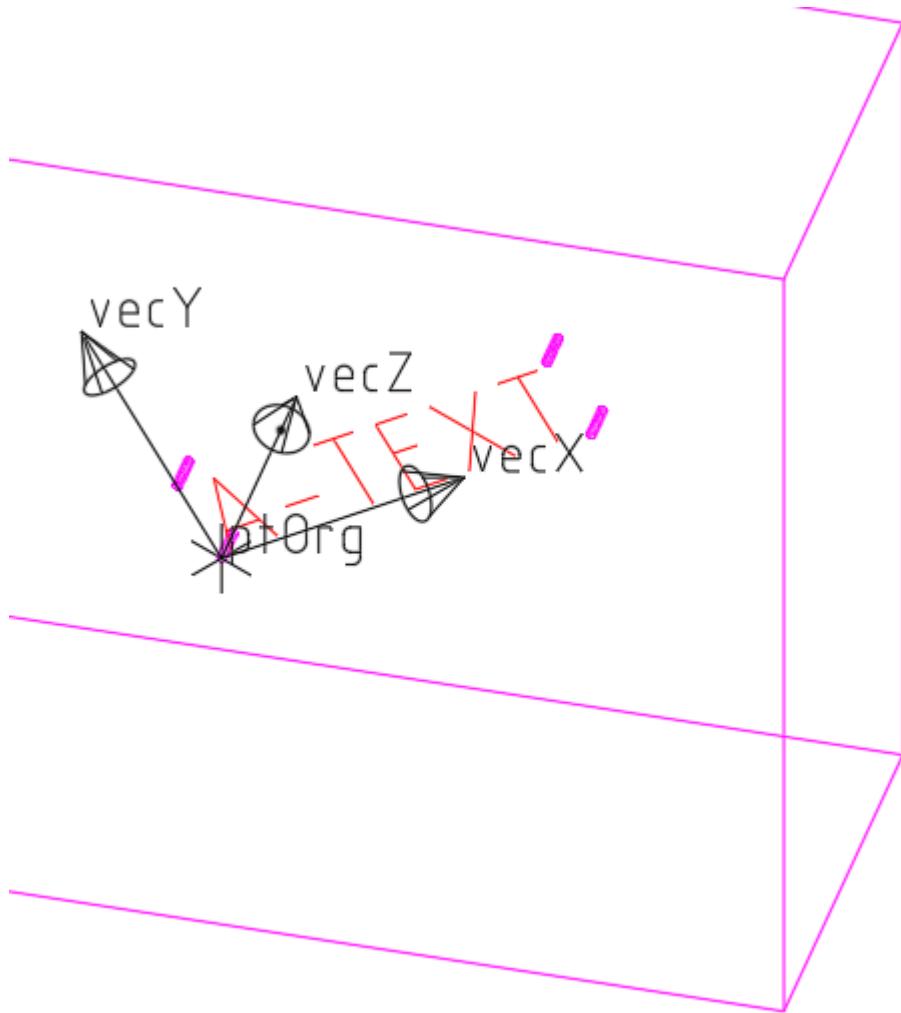
Values of yPos:

<b>_kBottom:</b>	-1
<b>_kCenter:</b>	0 (default value)
<b>_kTop:</b>	1

See [General](#) for member functions applicable to all tools.

---

[Example of E-type:



```

Unit(1, "mm"); // use mm as unit in U() function from now on

PropDouble dYH(0, U(13), T("|YHeight|"));
PropString txt(0, "a text", T("|text|"));

String arXpos[] = { T("|Left|"), T("|Center|"), T("|Right|") };
int arIXpos[] = { _kLeft, _kCenter, _kRight };
PropString xpos(1, arXpos, T("|xpos|"));
int ixpos = arIXpos[arXpos.find(xpos)];

String arYpos[] = { T("|Bottom|"), T("|Center|"), T("|Top|") };
int arIYpos[] = { _kBottom, _kCenter, _kTop };
PropString ypos(2, arYpos, T("|ypos|"));
int iypos = arIYpos[arYpos.find(ypos)];

PropInt pYN(0, 0, T("|flip x|"));

PropDouble dAngle(1, 0, T("|angle|"));
dAngle.setFormat(_kAngle);

setMarbleDiameter(U(1));

```

```

// rotate the coordinate system with the _Z0 axis
Vector3d vecX = _x0;
if (pYN)
    vecX *= -1;
Vector3d vecZ = -_z0;
Vector3d vecY = _y0;
vecX = vecX.rotateBy(dAngle, vecX.crossProduct(vecY));
vecY = vecY.rotateBy(dAngle, vecX.crossProduct(vecY));

//bring point to surface
Point3d ptOrg = _Pt0 + 0.5*_H0*_z0;

vecX.normalize();
vecY.normalize();
vecZ.normalize();
ptOrg.vis();
vecX.vis(ptOrg);
vecY.vis(ptOrg);
vecZ.vis(ptOrg);
reportMessage("\n" + "xpos:" + ixpos + "ypos:" + i ypos);

FreeText ft(txt, CoordSys(ptOrg, vecX, vecY, vecZ), dYH, ixpos,
i ypos);
_Beam0.addTool(ft);

double dXH = 0.8 * dYH * txt.length();
double dx = 0.5 * dXH;
double dy = 0.5 * dYH;

for (int i = 0; i < 4; i++)
{
    Vector3d vecOffset;
    if (i == 0) vecOffset = -dx * vecX - dy * vecY;
    else if (i == 1) vecOffset = dx * vecX - dy * vecY;
    else if (i == 2) vecOffset = dx * vecX + dy * vecY;
    else if (i == 3) vecOffset = -dx * vecX + dy * vecY;
    Point3d ptCenter = ptOrg - dx * ixpos * vecX - dy * i ypos * vecY +
vecOffset;
    Drill dr(ptCenter, ptCenter + U(10) * vecZ, U(1));
    _Beam0.addTool(dr);
}

```

*—end example]*

## 8.22 HalfCut

The HalfCut defines a saw cut in a beam. The saw cut can be seen as a infinite half plane, but with a certain thickness. The HalfCut constructor has the following format:

```
HalfCut name(Point3d ptOrg, Vector3d vecN, Vector3d vecY, int bDoSolid);
HalfCut name(Point3d ptOrg, Vector3d vecN, Vector3d vecY, int bDoSolid, Vector3d
    vecN2);
```

<b>ptOrg:</b>	point on border line of half plane saw line
<b>vecN:</b>	normal to halfplane
<b>vecY:</b>	vector inside the halfplane, perpendicular to border line. When looking from the point ptOrg, in the direction of the saw.
<b>bDoSolid:</b>	flag to indicate if the solid operation needs to be done by this tool
<b>vecN2:</b>	normal of "the other" halfcut. This normal is used to reposition the ptOrg, such that for sharp angles, the cut side is not damaged (see second example).

See [General](#) for member functions applicable to all tools.

Because it is likely that the solid operation is not done by this tool, the bDoSolid is an argument in the constructor of this tool. It has the same effect as calling the general setDoSolid routine.

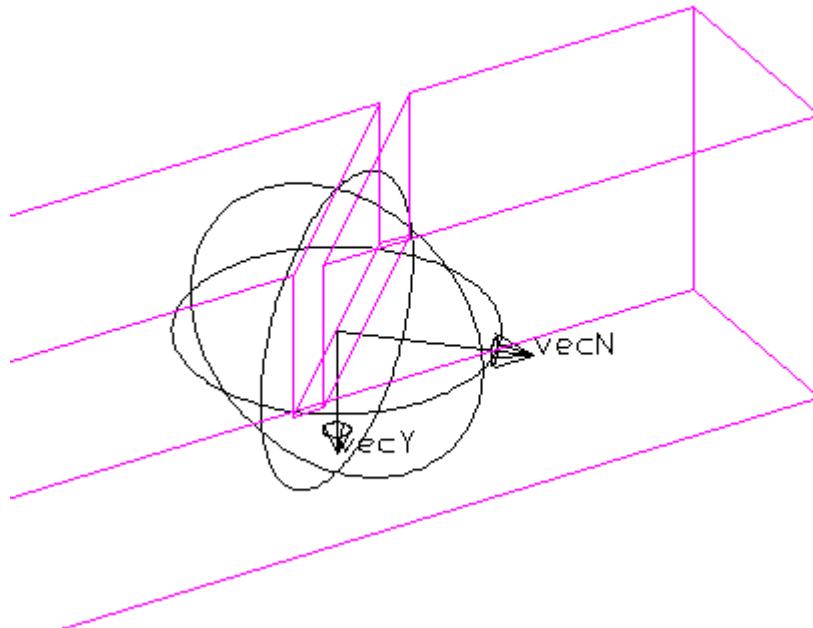
The length, with default value 0 (which means infinity) means the length in the direction perpendicular to vecN and perpendicular to vecY direction. The length is measured centered on ptOrg.

```
HalfCut::setLength(double dValue); // (added hsbCad2011 build 16.0.47)
```

---

[Example of E-type:

```
U(1,"mm");
Vector3d vecN = _X0+_Y0;
vecN.vis(_Pt0);
Vector3d vecY = _Z0;
vecY.vis(_Pt0);
HalfCut hc(_Pt0,vecN,vecY,TRUE);
_Beam0.addTool(hc);
```



—end example]

[Example of E-type with 2 extra grip points:

```

U(1, "mm");

// need 2 grippoints
if (_PtG.length()<2) return;

// The grippoints are projected in the _Pt0, _y0(normal) plane
Plane plProj(_Pt0, _y0);
_PtG = plProj.projectPoints(_PtG);

// The first grippoint determines the vecY and vecN
Vector3d vecY = _Pt0 - _PtG[0];
vecY.normalize();
vecY.vis(_PtG[0]);

Vector3d vecN = _y0.crossProduct(vecY);
vecN.normalize();
vecN.vis(_Pt0);

// The second grippoint determines the vecY2 and vecN2
Vector3d vecY2 = _Pt0 - _PtG[1];
vecY2.normalize();
vecY2.vis(_PtG[1]);

```

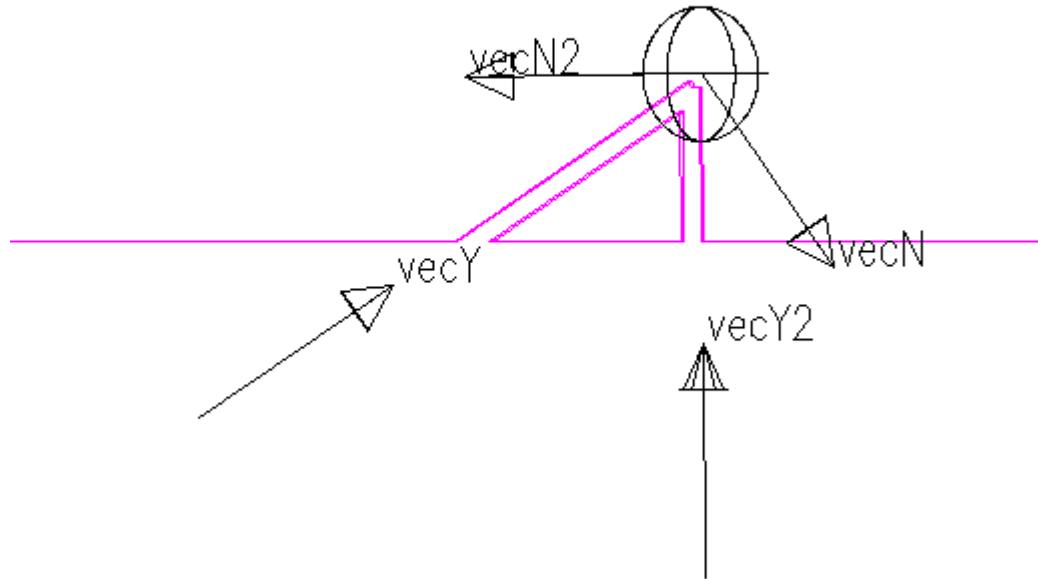
```

Vector3d vecN2 = _y0.crossProduct(-vecY2);
vecN2.normalize();
vecN2.vis(_Pt0);

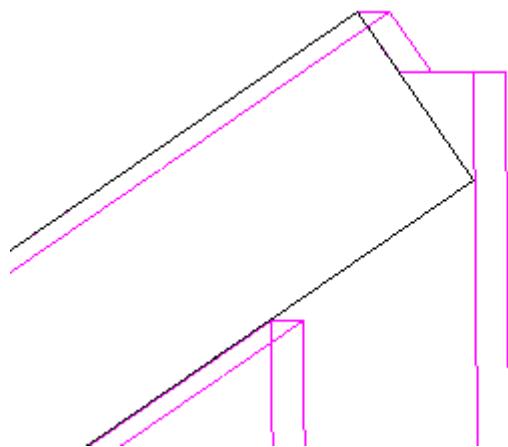
HalfCut hc(_Pt0,vecN,vecY,TRUE,vecN2);
_Beam0.addTool(hc);
HalfCut hc2(_Pt0,vecN2,vecY2,TRUE,vecN);
_Beam0.addTool(hc2);

```

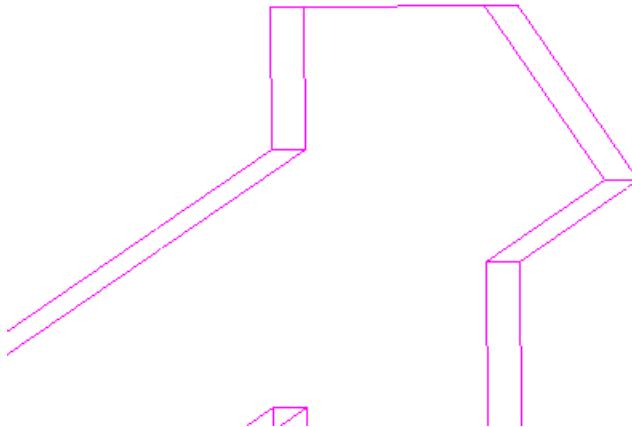
---



With the use of the second normal.



Without the use of the second normal:



*—end example]*

## 8.23 House

The housing tool constructor has one of the following formats:

```
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    nEndType, int nRoundType);
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth);
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, int nEndType, int nRoundType);
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
    double dZFlag);
House name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
    double dZFlag, int nEndType, int nRoundType);
```

ptOrg:	origin point of the box defining the operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

To specify the dimensions of the tool, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the tool is the same as `vecY.length()*dYheight`. If `vecY` is a unit-length vector, then the `dYheight` corresponds with the height. If `dYHeight` equals 1, then the length of the vector expresses the height.

The flags dXFlag, dYFlag and dZFlag, specify the position of the ptOrg inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point ptOrg is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -vecX, -vecY and -vecZ direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the vecX, vecY and vecZ coordinate system with origin ptOrg.

The housing with the same geometrical parameters can be used for both the male and the female side. The only parameter that differs is the **nEndType** parameter. To change this value on an existing housing tool, the **setEndType** function can be used.

**House::setEndType(int nEndType);**

Values of nEndType:

- \_kFemaleEnd:** done at end side of beam
- \_kFemaleSide:** not done at end of beam
- \_kMaleEnd:** male one, done at the end side of beam

The housing has a round type, which is set by the last parameter in the constructor. It can also be changed separately by the function [setRoundType](#).

**House::setRoundType(int nRoundType);**

Values of nRoundType:

- \_kNotRound:** not rounded
- \_kRound:** rounded
- \_kRelief:** relief
- \_kRoundSmall:** rounded with small diameter
- \_kReliefSmall:** relief with small diameter
- \_kRounded:** rounded

The solid body that represents this tool can be used by calling

**Body cuttingBody() const;**

For the housing to work properly, the X and Y need to be orthogonal. But for the parallel housing, the X and Y vectors do not need to be parallel. In this case you can set the XYOrthogonal to FALSE. Default value is TRUE. If XYOrthogonal is set to FALSE, and X and Y are not orthogonal, the housing is exported as [ParHouse](#).

**House::setXYOrthogonal(int bXYOrthogonal); // added v19.1.91 and v20.0.62.**

If the tool is combined with other tools on the same plane, eg like the situation of double housed tenon, the space around the tenon should not be cut away completely. The cnc machine should only go around one pass with the finger mill. To achieve this, the tool is flagged to mill around TRUE. The flag is only important for the male tool. The part that needs to be cut away further needs to be added through additional beamcuts.

**House::setMillAround(int bMillAround); // added v21.4.41 and v22.0.75.**

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```

Unit(1,"mm"); // use mm as unit in U() function from now on

double dXW= U(80);
double dYH= U(120);
double dZD= U(20);

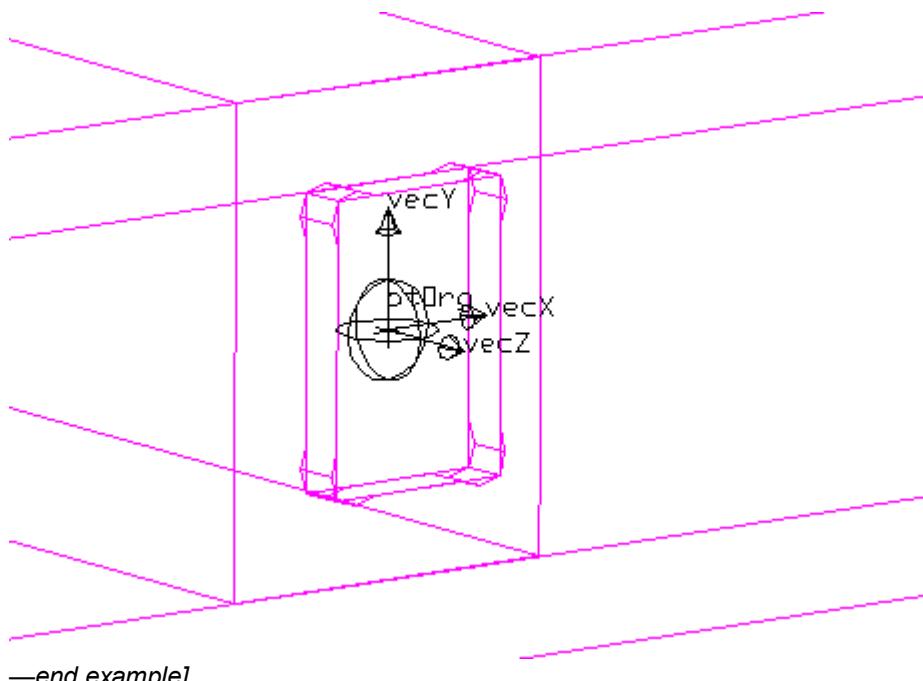
// rotate the coordinate system with the _Z0 axis
Vector3d vecZ= _Z1; // _Z1 is always normal to contact surface => depth direction
Vector3d vecY= _Z0; // _Z0 initially most aligned with _ZW => height direction
Vector3d vecX= vecY.crossProduct(vecZ); // perpendicular to vecY and vecZ
Point3d ptOrg = Pt0 ;

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

House hs(ptOrg,vecX,vecY,vecZ,dXW,dYH,dZD,0,0,1,_kFemaleSide,_kRelief);
_Beam1.addTool(hs); // female beam

hs.setEndType(_kMaleEnd);
_Beam0.addTool(hs,1); // is added with parameter 1: endtool active: will remove all other
endtools during insert

```

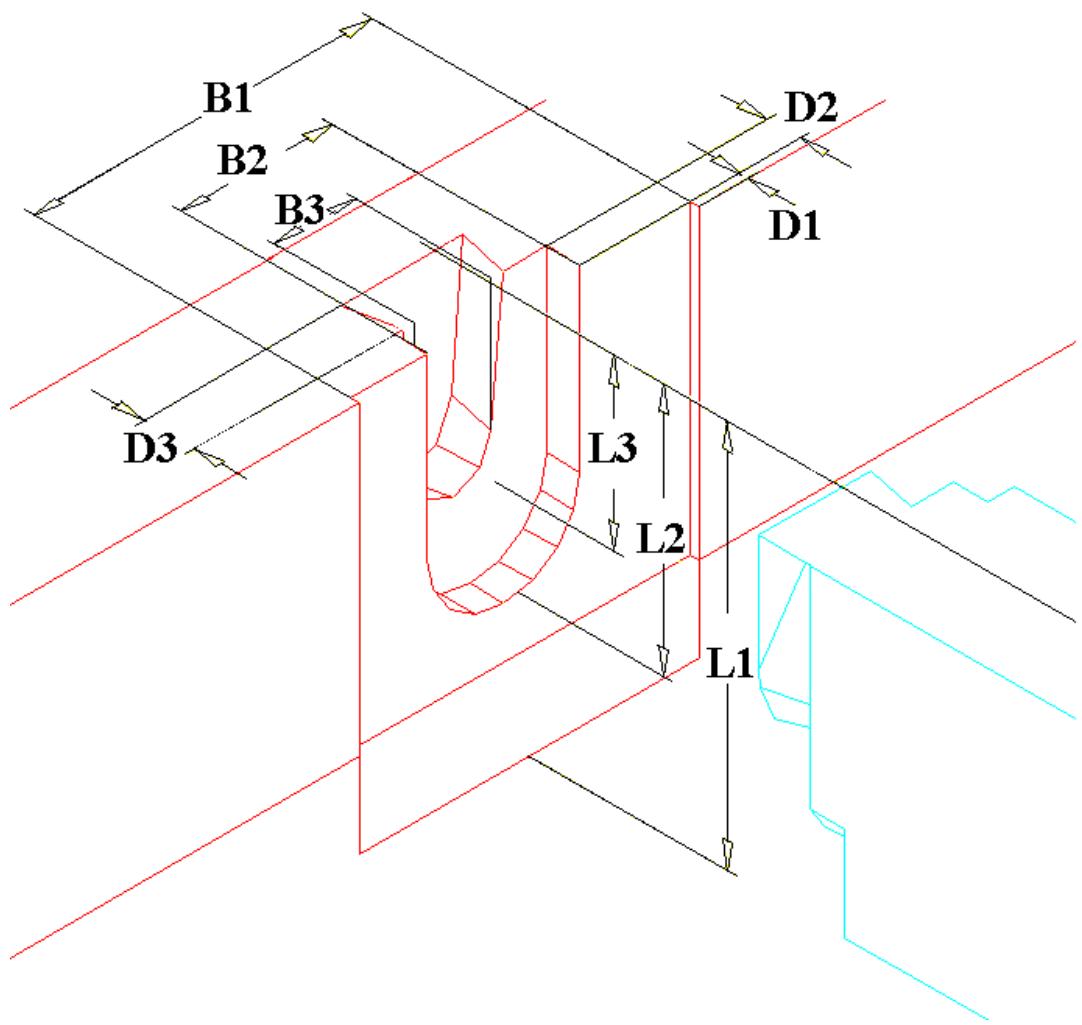


## 8.24 Housed dove tail connection

The housed dove tail constructor has the following format:

```
HousedDove name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ,  
        double dXW1, double dYH1, double dZD1,  
        double dXW2, double dYH2, double dZD2, double dO2,  
        double dXW3, double dYH3, double dZD3, double dO3, double dAlfa,  
        int nEndType);
```

ptOrg:	point opposite from top of dove, on the female surface
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXW1 (B1):	width on female surface, of main housing
dYH1 (L1):	height on female surface, of main housing
dZD1 (D1):	depth of main housing
dXW2 (B2):	width on female surface, of second housing
dYH2 (L2):	height on female surface, of second housing
dZD2 (D2):	depth of second housing
dO2:	offset in vecX direction of the second housing with respect to the main housing
dXW3 (B3):	width on female surface, of dove tail
dYH3 (L3):	height on female surface, of dove tail
dZD3 (D3):	depth of dove tail
dO3:	offset in vecX direction of the dove tail with respect to the main housing
dAlfa:	top angle in degrees (see <a href="#">dove connection</a> )



The dove tail with the same geometrical parameters can be used for both the male and the female side. The only parameter that differs is the nEndType parameter. To change this value on an existing dove, the setEndType function can be used.

```
HousedDove::setEndType(int nEndType);
```

Values of nEndType allowed for the housed dove tail

- \_kFemaleSide:** not done at end of beam
- \_kMaleEnd:** male one, done at the end side of beam

The housed dove tail has an extra member function to set the width of the flat bottom side of the rounded tenon and rounded dove tail. Default values are 0 for both.

```
HousedDove::setFlatWidth(double dFlatWidthTenon, double dFlatWidthDoveTail);
```

See [General](#) for member functions applicable to all tools.

[Example T-type:

```

Unit(1, "mm"); // use mm as unit in U() function from now on

double dXW1= _W0;
double dYH1= _H0;
double dZD1= U(3);

double dXW2= U(80);
double dYH2= U(120);
double dZD2= U(20);
double dO2 = 0;

double dXW3= U(40);
double dYH3= U(80);
double dZD3= U(20);
double dO3 = 0;
double dAlfa= 20; // in degrees

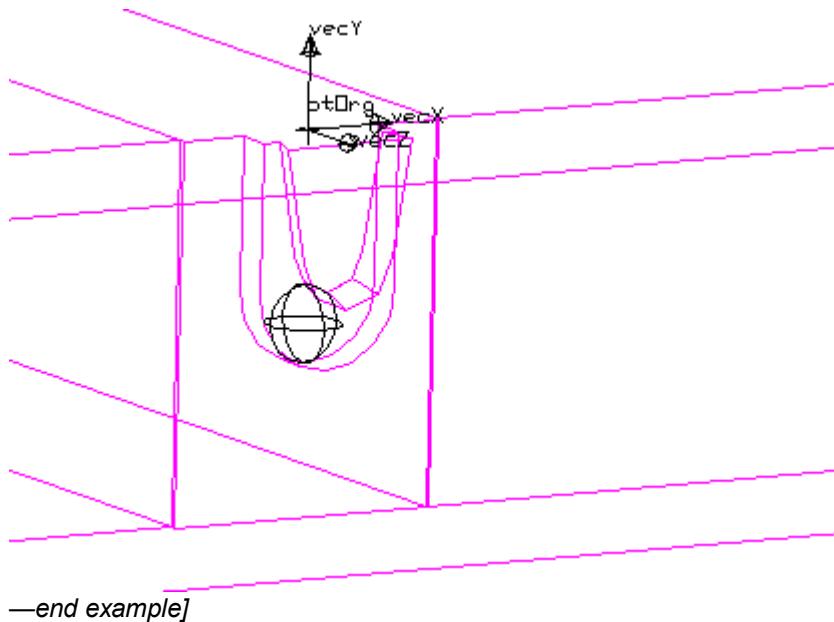
// rotate the coordinate system with the _Z0 axis
Vector3d vecZ = _Z1; // _Z1 is always normal to contact surface => depth direction
Vector3d vecY = _Z0; // _Z0 initially most aligned with _ZW => height direction
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and vecZ
Point3d ptOrg = _Pt0 + 0.5*_H0*_Z0; // move _Pt0 half beam height up

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

HousedDove dv(ptOrg, vecX, vecY, vecZ, dXW1, dYH1, dZD1, dXW2, dYH2, dZD2, dO2,
dXW3, dYH3, dZD3, dO3, dAlfa, _KFemaleSide);
_Beam1.addTool(dv); // female beam

dv.setEndType(_kMaleEnd);
_Beam0.addTool(dv, 1); // is added with parameter 1: endtool active: will remove all other
endtools during insert

```



## 8.25 Kamatsugi

The Kamatsugi tool constructor has the following format (added V25.1.52):

```
Kamatsugi name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
nEndType,
double dTenonZDepth, double dSeatHeight, double dTopToBackstep, double
dSeatLength,
double dTenonXWidth, double dTenonStep, double dTenonYHeight);
```

ptOrg, vecX, vecY, vecZ: location and orientation of the tool

nEndType: indicating male or female

dTenonZDepth: total depth in vecZ direction

dSeatHeight, dTopToBackstep, dSeatLength: parameters determining the seat cutout

dTenonXWidth, dTenonStep, dTenonYHeight: tenon parameters

Values of nEndType:

**\_kFemaleEnd**: done at end side of beam

**\_kMaleEnd**: male one, done at the end side of beam

Some additional parameters can be set separately by the functions:

```
setEndType(int endtype);
setFlatWidth(double dWidth);
setReducedTipWidth(double dWidth);
setExtraTopNeckWidth(double dWidth);
```

Important constraints for the Kamatsugi tool are:

- the ptOrg is located at the base surface of the tenon, and at the bottom center of the tenon.
- the vecZ is pointing along the male tool
- the vecY direction points towards the surface that is open

See [General](#) for member functions applicable to all tools.

---

*[Example E-type but with 2 beams selected:*

```

Unit(1, "mm"); // use mm as unit in U() function from now on

String arEndTypeNames[] = { T("|Female end|"), T("|Male end|") };
int arEndTypeInts[] = { _kFemaleEnd, _kMaleEnd };
PropString pEndType(1, arEndTypeNames, T("|End type|"), 0);
int nEndType = arEndTypeInts[arEndTypeNames.find(pEndType, 0)];

// the tool follows the location and orientation of _Beam0
Point3d ptOrg = _Pt0;
Vector3d vecZ = _X0;
Vector3d vecY = _Z0;
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and
vecZ

double dTenonZDepth = U(200);
double dSeatHeight = U(30);
double dTopToBackstep = U(150);

double dSeatLength = U(10);
double dTenonXWidth = U(80);
double dTenonStep = U(15);

double dTenonYHeight = U(140);
double dFlatWidth = U(40);
double dReducedTipWidth = U(5);
double dExtraTopNeckWidth = U(11);

if (nEndType==_kMaleEnd)
{
    vecZ = -vecZ;
    vecX = -vecX;
}

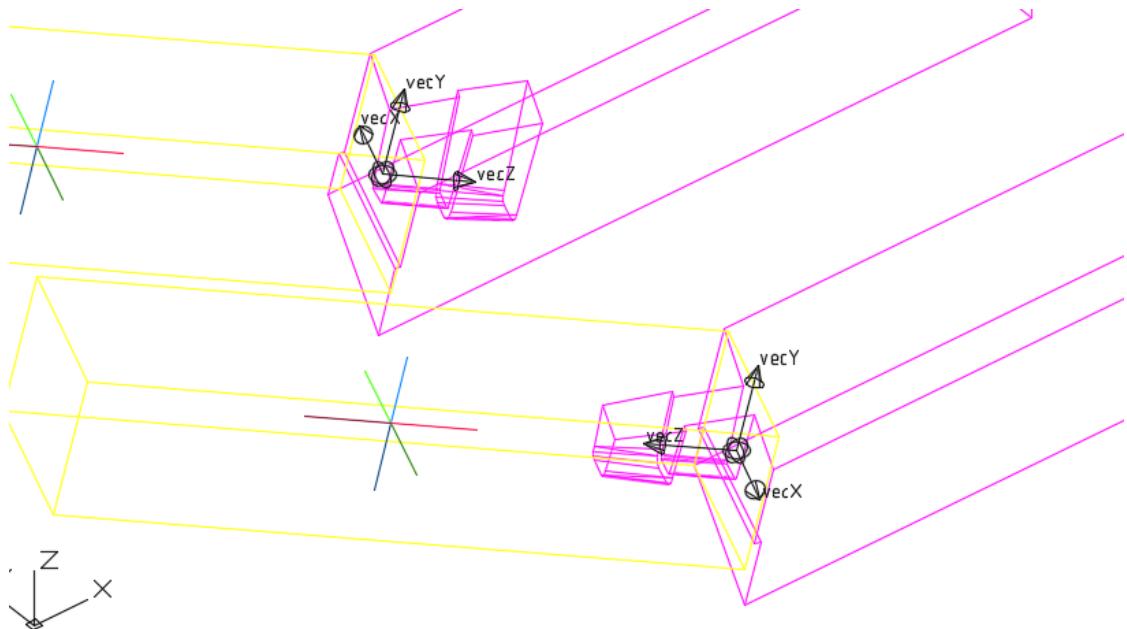
// visualize orientation in debug mode
Point3d ptVis = ptOrg;
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Kamatsugi ks(ptOrg, vecX, vecY, vecZ, nEndType,
    dTenonZDepth, dSeatHeight, dTopToBackstep,
    dSeatLength, dTenonXWidth, dTenonStep,
    dTenonYHeight);
ks.setFlatWidth(dFlatWidth);
ks.setReducedTipWidth(dReducedTipWidth);

```

```
ks.setExtraTopNeckWidth (dExtraTopNeckWidth);

_Beam1.addTool (ks, _kStretchOnInsert);
```



*—end example]*

## 8.26 Klingschrot

The Klingschrot tool (added since hsbCAD2012 build 17.0.24) its constructor has the following format:

```
Klingschrot name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    nMillHeadIndex, double dRadius, double dYArcWidth, double dYLeadWidth);
Klingschrot name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int
    nMillHeadIndex, double dRadius, double dYArcWidth, double dYLeadWidth,
    double dSpeed); // (added hsbCAD2012 build 17.0.52)
```

ptOrg:	point opposite from top of dove, on the female surface
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
nMillHeadIndex:	index of the mill head to be used
dRadius:	radius
dYArcWidth:	dimension in vecY direction of arc part
dYLeadWidth:	dimension in vecY direction
dSpeed:	speed in mm/sec, units independent of units of dwg

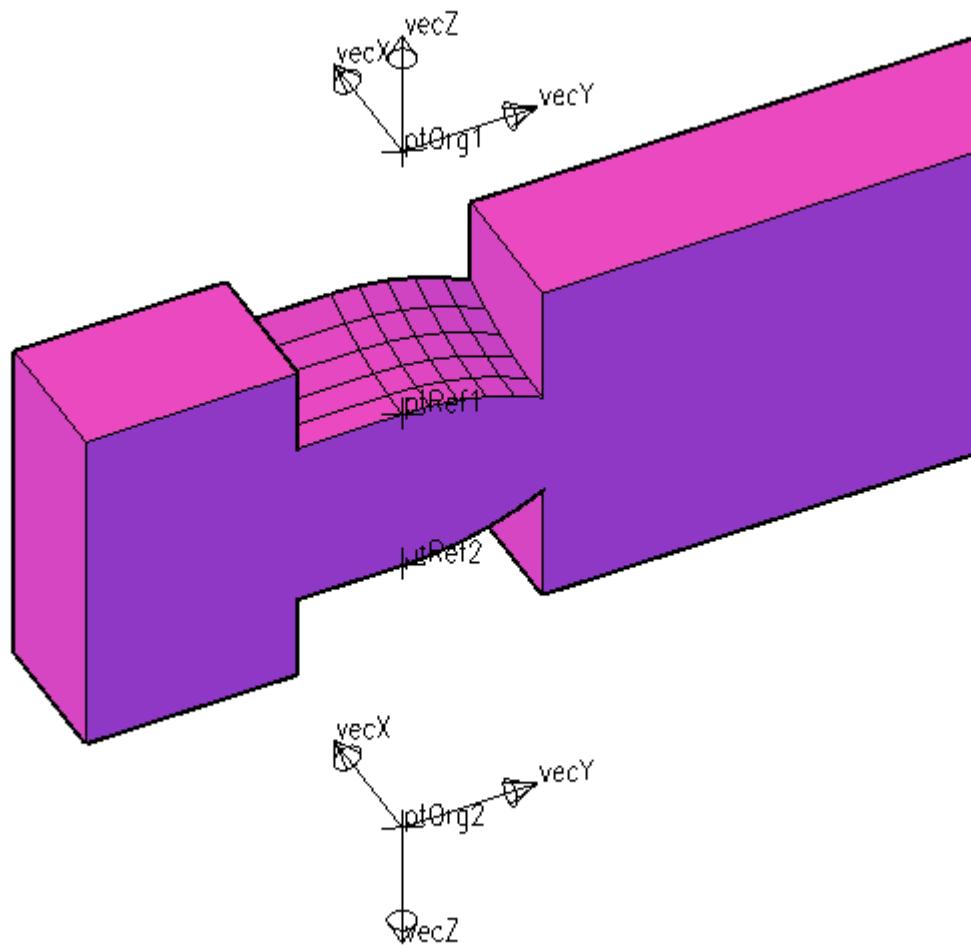
The Klingschrot tool is not an end tool. It cannot be stretched away on its own. To have endTool behaviour, the etool needs to have at least one end tool, eg a Cut.

If nMillHeadIndex identifies the mill head that is used on the machine.

See [General](#) for member functions applicable to all tools.

---

[Example E-type:



```
Unit(1, "mm"); // use mm as unit in U() function from now on
```

---

```

String arYN[] = {T("|No|"), T("|Yes|")};
PropString pAddCut(0,arYN,T("|Add cut and become end tool|"));
int bIsEndTool = (pAddCut==arYN[1]);
String arDir[] = {T("|Negative|"), T("|Positive|")};
PropString pDirection(1,arDir,T("|Direction|"));
int iDir = (pDirection==arDir[1])- (pDirection==arDir[0]);

int nMillHeadIndex = 0;
double dRadius = U(175);
double dYArcWidth= U(80);
double dYLeadWidth= U(60);

// rotate the coordinate system with the _Z0 axis
Vector3d vecZ = _Z0;
Vector3d vecX = _Y0;
Vector3d vecY = _X0*iDir; //vecZ.crossProduct(vecX);
Point3d ptRef1 = _Pt0 + 0.25*H0*Z0 - 0.5*W0*Y0;
Point3d ptOrg1 = ptRef1 + dRadius*vecZ;
Point3d ptRef2 = _Pt0 - 0.25*H0*Z0 - 0.5*W0*Y0;
Point3d ptOrg2 = ptRef2 - dRadius*vecZ;

// visualize orientation in debug mode
ptRef1.vis();
ptOrg1.vis();
vecX.vis(ptOrg1);
vecZ.vis(ptOrg1);
vecY.vis(ptOrg1);
ptOrg2.vis();
ptRef2.vis();
vecX.vis(ptOrg2);
(-vecZ).vis(ptOrg2);
vecY.vis(ptOrg2);

Klingschrot klingschrot1(ptOrg1,vecX,vecY,vecZ, nMillHeadIndex,
dRadius, dYArcWidth, dYLeadWidth);
Beam0.addTool(klingschrot1);
Klingschrot klingschrot2(ptOrg2,vecX,vecY,-vecZ, nMillHeadIndex,
dRadius, dYArcWidth, dYLeadWidth);
Beam0.addTool(klingschrot2);

// The Klingschrot tool is not an end tool. If you need endTool
behaviour, a cut needs to be added.
if (bIsEndTool) {
    Cut ct(ptOrg1-vecY*dYLeadWidth, -vecY);
    Beam0.addTool(ct, kStretchOnInsert); // male beam
}

```

*—end example]*

## 8.27 LogNotch

This tool, called standard log notch, is the most often used for log wall connections. The standard log notch constructor has the following format:

```
LogNotch name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dYWidthBeam, double dZHeightBeam, double dXWidthInterior, double dZDepthTop,
    double dZDepthBottom, double dXWidthSide, double dYDepthHouse, int
    nTypeShoulder, double dXAxisOffsetOtherBeam);
```

ptOrg: point on axis of beam

vecX: axis along beam

vecY: beam Y direction

vecZ: beam Z direction, must be perpendicular to the axis of the 2 beams

dYWidthBeam: beam width, in vecY direction

dZHeightBeam: beam height, in vecZ direction

dXWidthInterior: interior width, parameter p4

dZDepthTop: depth top, parameter p5

dZDepthBottom: depth bottom, parameter p6

dXWidthSide: side width, parameter p7

dYDepthHouse: house depth, parameter p8

nTypeShoulder: shoulder type, must have a value (**\_kBackward**, **\_kCentered**,  
**\_kForward**)

dXAxisOffsetOtherBeam: offset distance in vecX direction of the others beam axis, relative to  
its centerline. The value should only be different from 0 if the other beam  
its extrusion profile has a center point different from the middle of the  
extends of the profile.

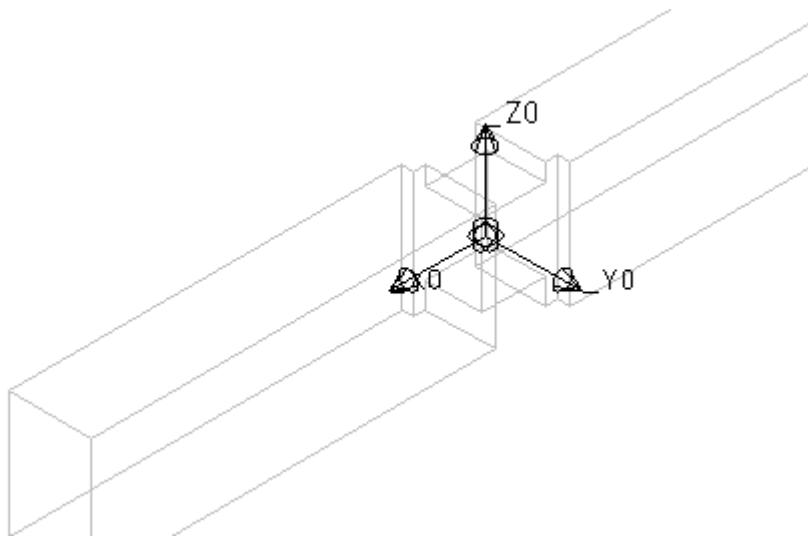
For a symmetrical connection there is a relation between the different parameters, see example  
below.

See [General](#) for member functions applicable to all tools.

When presented as [AnalysedTool](#), it becomes an [AnalysedLogNotch](#).

---

[Example E-type:



```

(1, "mm");

String arStrShoulderTypes []={T("|Forward|"), T("|Centered|"), T("|Backward|")};
int arNShoulderTypes []={_kForward, _kCentered, _kBackward};
PropString pType(0,arStrShoulderTypes, T("|Shoulder type|"));
int nShoulderType =
arNShoulderTypes[arStrShoulderTypes.find(pType,0)];

_x0.vis(_Pt0);
_y0.vis(_Pt0);
_z0.vis(_Pt0);
//Vector3d vecOffset = U(5)*_x0;
double dOffset = 0;

double dDeptTop = U(30);
double dDeptBottom = U(40);
double dDepthHouse = U(20);

// calculate the dWidthSide and dWidthInterior for matching crossbeam
double dWidthSide, dWidthInterior;
if (nShoulderType==_kCentered) {
    dWidthSide = _w0;
    dWidthInterior = _w0;
}
else {
    dWidthSide = _w0- dDepthHouse;
    dWidthInterior = _w0- 2*dDepthHouse;
}

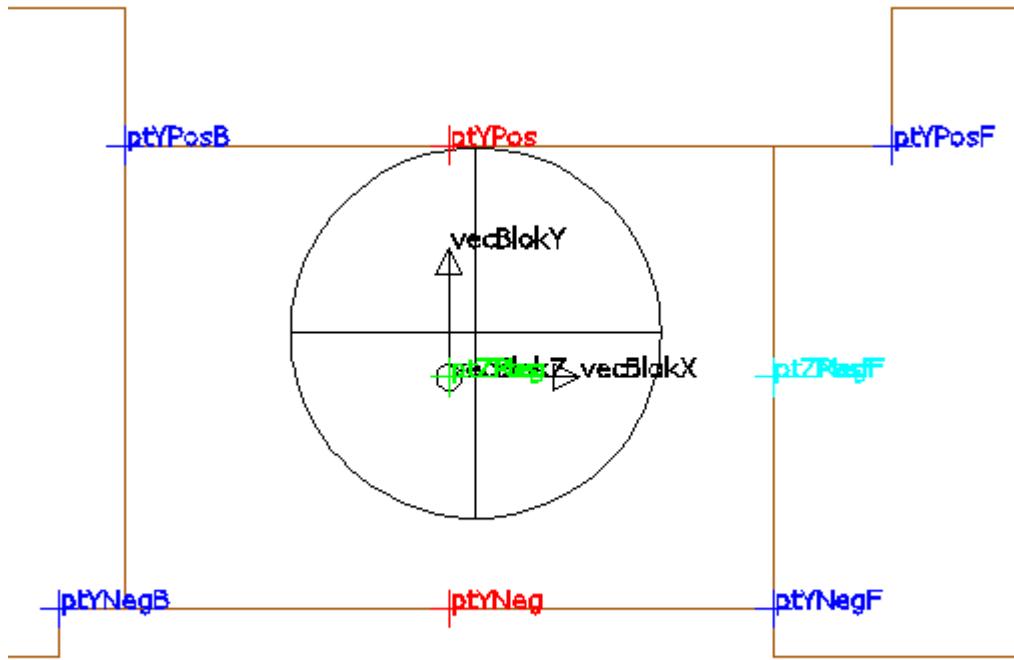
LogNotch ln(_Pt0,_x0,_y0,_z0,_w0,_h0, dWidthInterior, dDeptTop,
dDeptBottom, dWidthSide, dDepthHouse,
nShoulderType , dOffset);
_Beam0.addTool(ln);

```

—end example]

---

[Example E-type which (un)mystifies the LogNotch:



```

U(1, "mm");

PropDouble pBlokH(0, _H0, "Blok H");
PropDouble pBlokW(1, _W0, "Blok W");
PropDouble pWidthInterior(4, _W0, "WidthInterior = P4");
PropDouble pDeptTop(5, U(30), "DeptTop = P5");
PropDouble pDeptBottom(6, U(40), "DeptBottom = P6");
PropDouble pWidthSide(7, _W0+U(10), "WidthSide= P7");
PropDouble pDepthHouse(8, U(20), "DepthHouse= P8");

String arStrShoulderTypes []={T("|Forward|"), T("|Centered|"), T("|Backward|")};
int arNShoulderTypes []={_kForward, _kCentered, _kBackward};
PropString pType(0, arStrShoulderTypes, T("|Shoulder type|"));
int nShoulderType =
arNShoulderTypes[arStrShoulderTypes.find(pType, 0)];

PropInt pUseOffset(0,1, "Use offset from grip");

```

---

```

if (_PtG.length()==0)
    _PtG.append(_Pt0);

Vector3d vecBlokX = x0;
Vector3d vecBlokY = y0;
Vector3d vecBlokZ = z0;

Point3d ptCBlok = PtG[0];

double dOffsetXx = 0;
if (pUseOffset) {
    dOffsetXx = vecBlokX.dotProduct(_PtG[0] - Pt0);
}

vecBlokX.vis(ptCBlok);
vecBlokY.vis(ptCBlok);
vecBlokZ.vis(ptCBlok);

LogNotch ln(ptCBlok,vecBlokX,vecBlokY,vecBlokZ, pBlokW, pBlokH,
    pWidthInterior, pDeptTop, pDeptBottom, pWidthSide, pDepthHouse,
    nShoulderType , dOffsetXx);
Beam0.addTool(ln);

// 
Point3d ptCBeam = Pt0;
double dWBeam = Beam0.dD(vecBlokY); // beam dim in LogNotch Y dir
double dHBeam = Beam0.dD(vecBlokZ); // beam dim in LogNotch Z dir

Point3d ptYNeg, ptYNegF, ptYNegB;
Point3d ptYPos, ptYPosF, ptYPosB;
Point3d ptZNeg, ptZNegF;
Point3d ptZPos, ptZPosF;

double dDepthInterior = pBlokW-2*pDepthHouse;

if (nShoulderType==kCentered) {

    ptCBlok -= dOffsetXx*vecBlokX;

    ptYNeg = ptCBlok-0.5*dDepthInterior*vecBlokY;
    ptYNegF = ptYNeg + 0.5*pWidthSide*vecBlokX;
    ptYNegB = ptYNeg - 0.5*pWidthSide*vecBlokX;

    ptYPos = ptCBlok+0.5*dDepthInterior*vecBlokY;
    ptYPosF = ptYPos + 0.5*pWidthSide*vecBlokX;
    ptYPosB = ptYPos - 0.5*pWidthSide*vecBlokX;
}

```

```
ptZNeg = ptCBlok - 0.5*pBlokH*vecBlokZ + pDeptBottom*vecBlokZ;
ptZNegF = ptZNeg + 0.5*pWidthInterior*vecBlokX;

ptZPos = ptCBlok + 0.5*pBlokH*vecBlokZ - pDeptTop*vecBlokZ;
ptZPosF = ptZPos + 0.5*pWidthInterior*vecBlokX;

}

else { // only valid if _kForward or _kBackward

    int nForward = (nShoulderType==_kForward) ? 1 : 0;
    int nBackward = (nShoulderType==_kForward) ? 0 : 1;

    ptYPos = ptCBlok+0.5*dDepthInterior*vecBlokY;
    ptYPosF = ptYPos - 0.5*pWidthInterior*vecBlokX
        + (pWidthSide - 0.5*dOffsetXx*(1+nShoulderType))
*nForward*vecBlokX
        + pWidthInterior*nBackward*vecBlokX;
    ptYPosB = ptYPos + 0.5*pWidthInterior*vecBlokX
        - (pWidthSide + 0.5*dOffsetXx*(1-nShoulderType))
*nBackward*vecBlokX
        - pWidthInterior*nForward*vecBlokX;

    ptYNeg = ptCBlok-0.5*dDepthInterior*vecBlokY;
    ptYNegF = ptYNeg - 0.5*pWidthInterior*vecBlokX
        + (pWidthSide - 0.5*dOffsetXx*(1-nShoulderType))
*nBackward*vecBlokX
        + pWidthInterior*nForward*vecBlokX;
    ptYNegB = ptYNeg + 0.5*pWidthInterior*vecBlokX
        - (pWidthSide + 0.5*dOffsetXx*(1+nShoulderType))
*nForward*vecBlokX
        - pWidthInterior*nBackward*vecBlokX;

    ptZNeg = ptCBlok - 0.5*pBlokH*vecBlokZ + pDeptBottom*vecBlokZ;
    ptZNegF = ptZNeg + 0.5*pWidthInterior*vecBlokX;

    ptZPos = ptCBlok + 0.5*pBlokH*vecBlokZ - pDeptTop*vecBlokZ;
    ptZPosF = ptZPos + 0.5*pWidthInterior*vecBlokX;

}

ptYNeg.vis(1);
ptYNegF.vis(5);
ptYNegB.vis(5);

ptYPos.vis(1);
ptYPosF.vis(5);
ptYPosB.vis(5);

ptZNeg.vis(3);
ptZNegF.vis(4);
ptZPos.vis(3);
ptZPosF.vis(4);
```

—end example}

## 8.28 Mark

Adding this tool to a beam will draw 2 lines, through points pntLocation1 and pntLocation2, on the beam face with outer normal vecNormalFace, and place a text, strText in the middle of these two lines. This is done with the following constructor :

**Mark** mrk( pntLocation1, pntLocation2, vecNormalFace, strText);

Instead of drawing a text, draw the posnum of the beam with number nIndexPosnumBeam

**Mark** mrk( pntLocation1, pntLocation2, vecNormalFace, nIndexPosnumBeam);

Only draw one line, and place some text

**Mark** mrk( pntLocation1, vecNormalFace, strText);

Only one line and a posnum

**Mark** mrk( pntLocation1, vecNormalFace, nIndexPosnumBeam);

Only draw 2 lines, no text

**Mark** mrk( pntLocation1, pntLocation2, vecNormalFace);

Only draw one line

**Mark** mrk( pntLocation, vecNormalFace);

When text is drawn, one can set the text position and alignment options with the following routine:

**Mark**::setTextPosition(**int** iYPosText, **int** iXPosText, **int** nTextOrientation);

Values of iYPosText:

<b>_kBottom:</b>	-1
<b>_kCenter:</b>	0 (default value)
<b>_kTop:</b>	1

Values of iXPosText:

<b>_kLeft:</b>	-1
<b>_kCenter:</b>	0 (default value)
<b>_kRight:</b>	1

Values of nTextOrientation. Default value is 0.

<b>0:</b>	along positive X (default value)
<b>1:</b>	along positive Y
<b>2:</b>	along negative X
<b>3:</b>	along negative Y

For Hundegger BVN there is also 4 and 5, which express text of which the characters are oriented perpendicular to the text orientation.

If the text contains the substring <MYPOS>, that substring will be substituted with the genbeam his own posnum. You could argue that the <MYPOS> could be replaced by the [GenBeam](#)::posnum() value, but that is not completely true. If the Tsl retrieves the posnum value, and adds the Mark tool to

the beam, the beam is triggered to reevaluate its posnum. Reevaluation of its posnum is requested when any tool is added to the beam. But of course when the Mark tool itself is dependent on the posnum, we have an unresolvable dependency problem. Furthermore, when using the <MYPOS> substring, the Tsl will not need to be retriggered for the Mark tool to resolve its correct appearance.

To suppress the machine generation of the line, one can use the suppressLine, which has a default value of FALSE. If suppressLine is called, only the text is shown.

**Mark::suppressLine();**

To set the text height, one can use the setTextHeight. The default value of 0 will use the machine default one.

**Mark::setTextHeight(double dTextHeight); // (added hsbCAD13.2.101)**

As an alternative of specifying the index of the beam to be used for the posnum to show, one can also specify the Beam directly.

**Mark::setPosnumBeam(Beam bmPosnum); // (added hsbCAD14.1.1)**

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```
Cut endcut (_Pt0- U(20,"mm")*_Z1, _Z1);
_Beam0.addTool(endcut,1);

Point3d pt1 = Line(_Pt1,_X0).intersect(_Plf,0);
Point3d pt2 = Line(_Pt3,_X0).intersect(_Plf,0);

pt1.visualize(3);
pt2.visualize(3);
_Z1.visualize(pt1,3);
_Z1.visualize(pt2,3);

PropInt nType(0,0,"Marking type");

Mark mrk; // default constructor

if (nType==0) {
    PropString strText(0,"Marking text","Text");
    mrk = Mark(pt1,pt2,-_Z1,strText);
}
else if (nType==1)
    mrk = Mark(pt1,pt2,-_Z1,0); // 0 is index of _Beam0
else if (nType==2) {
    PropString strText(0,"Marking text","Text");
```

```

        mrk = Mark(pt1,-_z1,"Mark-text");
    }
    else if (nType==3)
        mrk = Mark(pt1,-_z1,0); // 0 is index of _Beam0
    else if (nType==4)
        mrk = Mark(pt1,pt2,-_z1);
    else if (nType==5)
        mrk = Mark(pt1,-_z1);

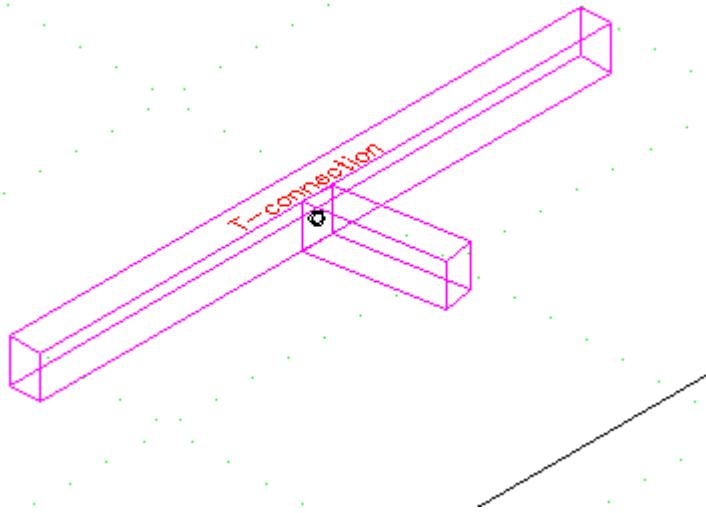
    mrk.setTextPosition(_kBottom,_kLeft,0);

    _Beam1.addTool(mrk);
}

```

*—end example]*

*[Example T-type that illustrates the setTextPosition*



```

Cut c0(_Pt0, _z1); // normal to contact face
_Beam0.addTool(c0);

// the setTextPosition is dependent on the vecX of the beam
Vector3d vecBmX = _Beam1.vecX();
vecBmX.vis(_Pt0);

Point3d ptLocation1 = _Pt0;
Vector3d vecNormalFace = _Beam1.vecD(_ZW); // show text on face
pointing up

String strPosi="T-connection";

// bottom or top is depending on vecBmX
Vector3d vecBottom2Top = vecNormalFace.crossProduct(vecBmX);
vecBottom2Top.normalize();

```

```

vecBottom2Top.vis(ptLocation1);

// for a T-tool, the _Z1 is normal to the face where _Beam0 hits
// Beam1, and pointing to away from _Beam0.
// So _Z1 is the desired text read up direction.
Vector3d vecReadUp = _Z1;
int iYPosText = _kBottom; // determines location
int nTextDir = 0;
if (vecReadUp.dotProduct(vecBottom2Top)<0) {
    iYPosText = _kTop;
    nTextDir = 1; // means upside down
}

Mark mrk1(ptLocation1, vecNormalFace, strPosi);
mrk1.setTextPosition(iYPosText, _kCenter, nTextDir);
mrk1.suppressLine();
_Beam1.addTool(mrk1);

—end example]

```

## 8.29 MarkerLine

The MarkerLine defines a drawn line on a beam. The face on which it is drawn, is determined by the vecN. The outer normal of the face of the beam must have the same direction of vecN. The MarkerLine constructor has the following format:

**MarkerLine** name(**Point3d** ptStart, **Point3d** ptEnd, **Vector3d** vecN);

ptStart: start point of line to be drawn  
 ptEnd: end point of line to be drawn  
 vecN: normal to face on beam

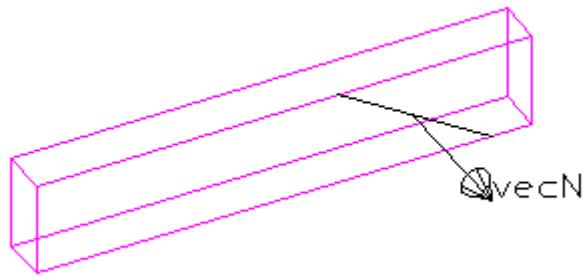
To export a MarkerLine as a [Mark](#) tool call exportAsMark with TRUE as argument.

**MarkerLine::exportAsMark(int bSet);** // (added hsbCAD21.0.71)

See [General](#) for member functions applicable to all tools.

---

*[Example of E-type with insert done in script:]*



```

if (_bOnInsert) {

    // get input
    Beam bm = getBeam();
    _Beam.append(bm);
    _PtG.append(getPoint("Select start point"));
    _PtG.append(getPoint("Select end point"));

    // find point on axis of beam, and set _Pt0 to it
    Point3d ptM = 0.5*(_PtG[0]+_PtG[1]);
    Line ln(bm.ptCen(), bm.vecX());
    _Pt0 = ln.closestPointTo(ptM);

    return;
}

if (_PtG.length()<2) return;

PropInt pAsMark(0, 0, T("Export as mark"));

// project the 2 selected points to the surface of the beam
Point3d ptM = 0.5*(_PtG[0]+_PtG[1]);
Vector3d vecN = _Beam0.vecD(ptM-_Pt0);
double dD = _Beam0.dD(vecN);
Plane pl(_Pt0+0.5*dD*vecN, vecN);
_PtG = pl.projectPoints(_PtG);

vecN.vis(ptM);
MarkerLine ml(_PtG[0], _PtG[1], vecN);
ml.exportAsMark(pAsMark);
_Beam0.addTool(ml);

Display dp(-1);
dp.draw(LineSeg(_PtG[0], _PtG[1]));

```

*—end example]*

## 8.30 MetalTKey

The MetalTKey tool (added since hsbCAD2017 build 21.0.68) its constructor has the following format:

```
MetalTKey name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int nMtTool,
    double dLength1, double dLength2, double dLength3, double dLength4, double
    dWidth1, double dWidth2, double dWidth4, double dDepth1, double dDepth2, double
    dDepth3);
```

ptOrg:	origin point of the box defining the operation
vecX:	axis in length direction
vecY:	width direction
vecZ:	depth direction
nMtTool:	Tool index for sub Mortise 4 (Normally 4 for Hundegger machine)
dLength1:	Length of sub Mortise 1
dLength2:	Length of sub Mortise 2
dLength3:	Length of sub Mortise 3
dLength4:	Length of sub Mortise 4
dWidth1:	Width of sub Mortise 1
dWidth2:	Width of sub Mortise 2
dWidth4:	Width of sub Mortise 4
dDepth1:	Depth of sub Mortise 1
dDepth2:	Depth of sub Mortise 2
dDepth3:	Depth of sub Mortise 3

Make certain that the dimension of dLength2 is long enough to allow tool for sub Mortise 4 to complete full length.

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```
int MtTool = 4;

double dLength1 = U(60);
double dLength2 = U(90);
double dLength3 = U(140);
double dLength4 = U(110);

double dWidth1= U(40);
double dWidth2 = U(20);
double dWidth4 = U(41);

double dDepth1 = U(35);
double dDepth2 = U(30);
double dDepth3 = U(5);

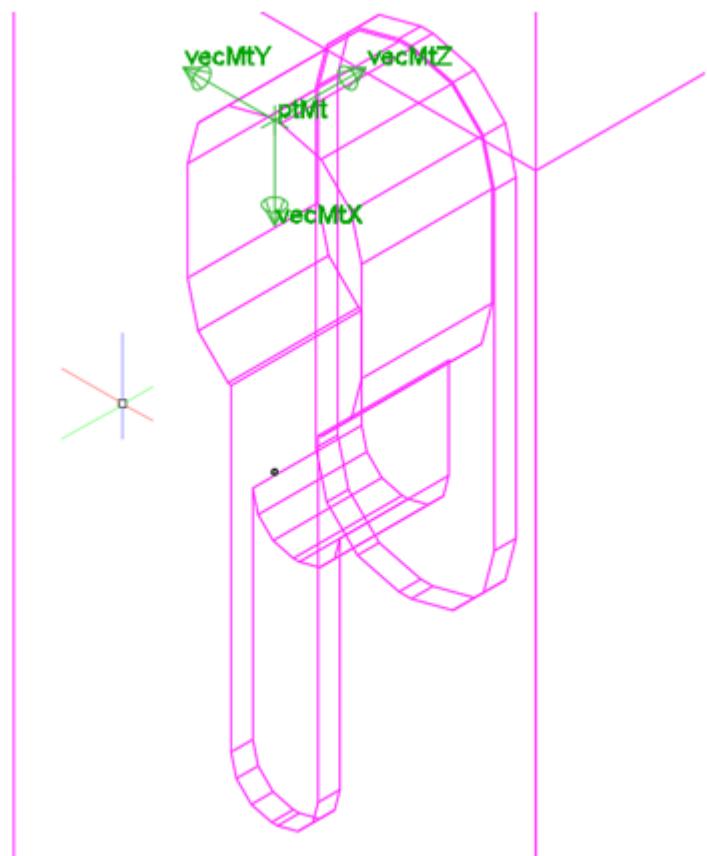
Vector3d vecMtX = _x1;
```

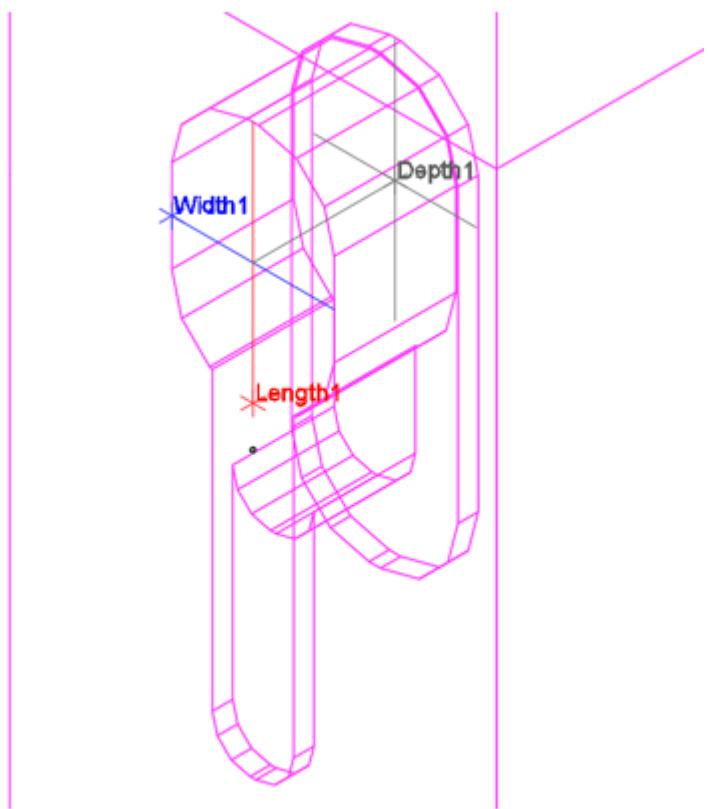
```
Vector3d vecMtY = _y1;
Vector3d vecMtZ = _z1;

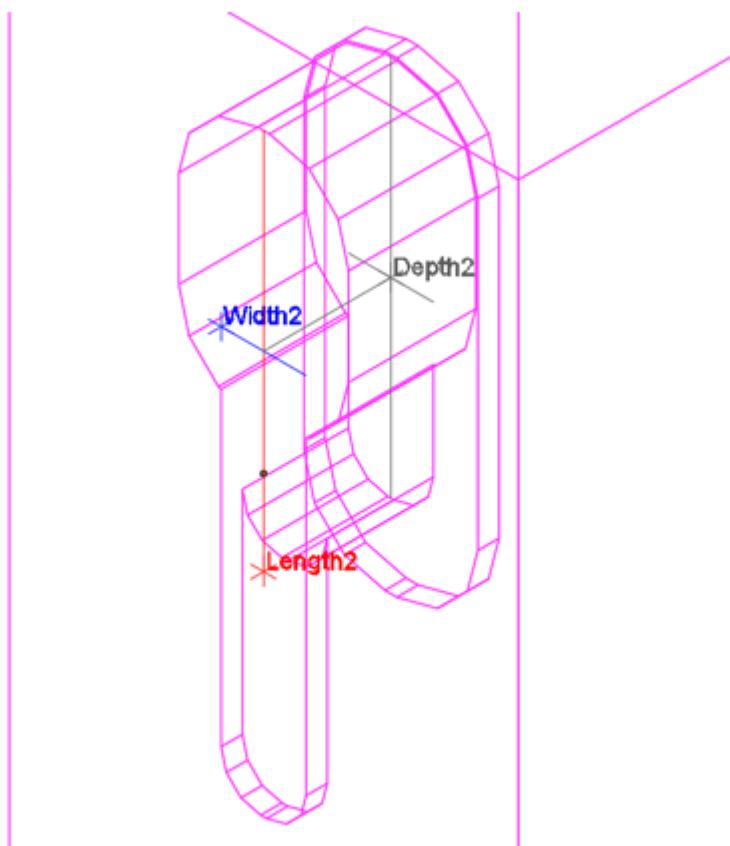
Point3d ptMt = _Pt0 - 0.5*dLength3*vecMtX;

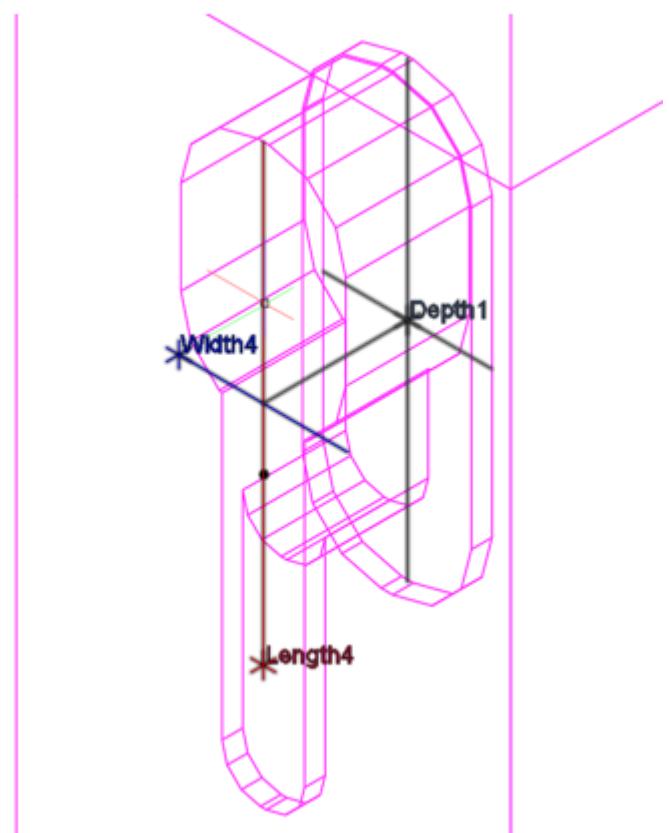
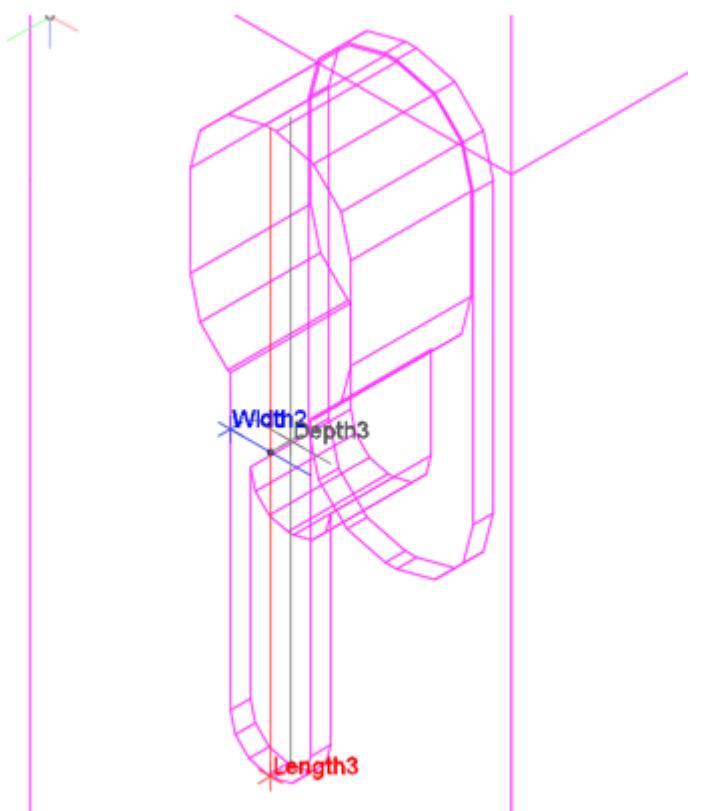
MetalTKey MfK1 (ptMt, vecMtX, vecMtY, vecMtZ, MtTool,
dLength1, dLength2, dLength3, dLength4,
dWidth1, dWidth2, dWidth4,
dDepth1, dDepth2, dDepth3);

_Beam1.addTool(MfK1);
```









—end example]

## 8.31 Mortise

```
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, int nEndType, int
nRoundType);
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth);
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth, int nEndType, int nRoundType);
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth, double dXFlag, double dYFlag, double dZFlag);
Mortise name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth, double dXFlag, double dYFlag, double dZFlag, int nEndType,
int nRoundType);
```

ptOrg:	origin point of the box defining the operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor
Default round type == <code>_kRound</code>	
Default end type == <code>_kFemaleSide</code>	

vecZ needs to be aligned with axis of mill, in direction of entering the beam. A tenon and a mortise that need to join together, will have the same coordinate system.

Functions allowed:

```
setEndType(int endtype);
setRoundType(int roundtype);
setExplicitRadius(double dRadius); // added v21.3.1 and v22.0.32
```

The solid body that represents this tool can be used by calling

```
Body cuttingBody() const;
```

If the tool is combined with other tools on the same plane, eg like the situation of double housed tenon, the space around the tenon should not be cut away completely. The cnc machine should only go around one pass with the finger mill. To achieve this, the tool is flagged to mill around TRUE. The flag is only important for the male tool. The part that needs to be cut away further needs to be added through additional beamcuts.

```
Mortise::setMillAround(int bMillAround); // added v21.4.41 and v22.0.75.
```

Values of nRoundType:

<b>_kNotRound</b> :	not rounded
<b>_kRound</b> :	round
<b>_kRounded</b> :	rounded
<b>_kExplicitRadius</b> :	explicit radius // added v21.3.1 and v22.0.32

Values of nEndType:

- \_kFemaleEnd:** done at end side of beam
- \_kFemaleSide:** not done at end of beam
- \_kMaleEnd:** male one, done at the end side of beam

See [General](#) for member functions applicable to all tools.

---

[Example E-type:

```

Unit(1,"mm"); // use mm as unit in U() function from now on

// order the round types such that their index corresponds to the
value of the round type
String strRoundTypes[] = { T("|Squared|"), T("|Round|"), T("|
Rounded|"), T("|Explicit Radius|") };
int arRoundTypeInts[] = { _kNotRound, _kRound, _kRounded,
_kExplicitRadius };
PropString pStrRound(0,strRoundTypes, T("|Round type|"));
int nRndType = arRoundTypeInts[strRoundTypes.find(pStrRound,0)];

String arEndTypeNames[] = { T("|Female end|"), T("|Male end|"), T("|
Female side|") };
int arEndTypeInts[] = { _kFemaleEnd, _kMaleEnd, _kFemaleSide };
PropString pEndType(1,arEndTypeNames, T("|End type|"),0);
int nEndType = arEndTypeInts[arEndTypeNames.find(pEndType,0)];

PropDouble pDepthMortise(0,U(33), T("|Depth|"));
PropDouble pExplRadius(1,U(0), T("|Explicit Radius|"));

if (_bOnInsert)
{
    if (insertCycleCount()>1) { eraseInstance(); return; }

    // show the dialog if no catalog in use
    if (_kExecuteKey == "")
        showDialog();
    // set properties from catalog
    else
        setPropValuesFromCatalog(_kExecuteKey);

    _Beam.append(getBeam(T("|Select dummy beam|")));
    _Beam.append(getBeam(T("|Select beam to add Mortise to|")));
    _Pt0 = getPoint();
    return;
}

Vector3d vecZ = _x0; // pointing towards beam1
Vector3d vecY = _z0;

```

---

```

Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and
                                             vecZ
Point3d ptOrgM = _Pt0;
double dXM= _Beam0.dW();
double dYM= _Beam0.dH();
double dZM= pDepthMortise;

double dXH= _Beam0.dW();
double dYH= _Beam0.dH();

if (nEndType==_kMaleEnd) {
    vecZ = -vecZ;
    vecX = -vecX;
}

// visualize orientation in debug mode
Point3d ptVis = ptOrgM;
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Mortise mt(ptOrgM, vecX, vecY, vecZ, dXM, dYM, dZM, 0, 0, 1,
nEndType, nRndType);
mt.setExplicitRadius(pExplRadius);
mt.setEndType(nEndType);
mt.setRoundType(nRndType);
_Beam1.addTool(mt, kStretchOnInsert);

```

*—end example]*

## 8.32 PanelStop

The PanelStop defines a foam cutout in a Sip panel. The PanelStop constructor has the following format:

```

PanelStop name(Beam bmLumber, String strEdgeDetailCode);
PanelStop name(Beam bmLumber, String strEdgeDetailCode, Point3d ptSipEdge,
Vector3d vecSipEdgeNormal);

```

bmLumber: strEdgeDetailCode: ptSipEdge: vecSipEdgeNormal:	piece of lumber to be cut away specifies the edge detail code to be used point on the edge of the panel normal to the panel edge, must point outside the panel
--	---

The PanelStop has a edge recess type. It can be set seperately by the function **setEdgeRecessType**. The recess type will return in the [SipEdge](#).

```
PanelStop::setEdgeRecessType(int nRecessType);
```

Values of nRecessType:

```

_kNoEdgeRecess
_kSplineDouble
_kSplineIJoist
_kSpline2xLumber
_kSplineSingleLumber
_kSplineSingleTopSkin
_kSplineBlock
_kSplineCustom
_kLetIn:           default
_kHeaderRecess

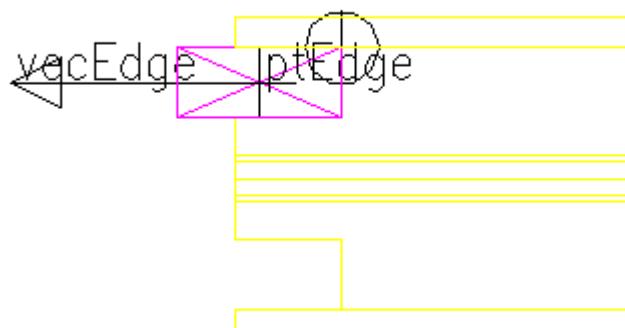
```

The PanelStop has the following member function.

```
PanelStop::setEdgeDetailCode(String strCode);
```

See [General](#) for member functions applicable to all tools.

[Example of O-type with insert done in script:



```

Unit(1, "mm");

if (_bOnInsert) {
    _Beam.append(getBeam("\nSelect beam for panel stop"));
    _Sip.append(getSip("\nSelect panel for panel stop"));
    _Pt0 = getPoint("\nSelect Tsl location");
    return;
}

// check running conditions
if (_Beam.length()==0) return;
if (_Sip.length()==0) return;

```

```
Beam bm = Beam[0];
Sip pnl = Sip[0];

Point3d ptEdge = bm.ptCen(); // point near the edge of the panel
Vector3d vecEdge = bm.vecD(ptEdge - pnl.ptCen()); // point outwards
of panel

PanelStop ps(bm, "", ptEdge, vecEdge );
ps.setEdgeRecessType(kSplineIJoist);
ps.setEdgeDetailCode("aa");
pnl.addTool(ps);
setDependencyOnBeamLength(bm); // when the beam length changes, this
TSL needs to be recalculated.
```

*—end example]*

## 8.33 PanelWirechase

The PanelWirechase tool (added since hsbCAD2012 build 17.0.41) its constructor has the following format:

```
PanelWirechase name(Point3d ptStart, Point3d ptEnd, Vector3d vecY, double
dYWidth, double dZHeight);
```

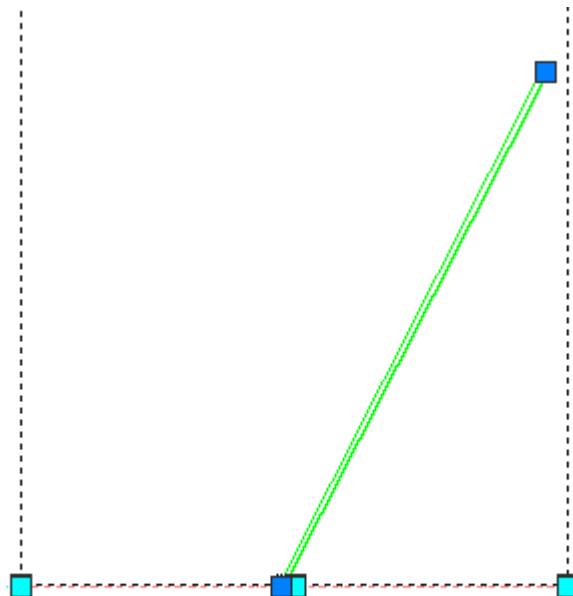
ptStart:	start point of wire chase
ptEnd:	end point of wire chase
vecY:	direction perpendicular to ptStart-ptEnd direction in which the dYWidth dimension needs to be cut out
dYWidth:	dimension in vecY direction
dZHeight:	dimension in vecZ direction

The PanelWirechase tool is typically added to a Sip entity.

See [General](#) for member functions applicable to all tools.

---

*[Example E-type:*



```

Unit(1, "mm");

PropDouble dYWidth(0, U(10), T("Wirechase width"));
PropDouble dZHeight(1, U(20), T("Wirechase height"));

if (_bOnInsert) {
    _Sip.append(getSip(T("Select panel for panel stop")));
    _Pt0 = getPoint(T("Select start of wire chase"));
    _PtG.append(getPoint(T("Select end of wire chase")));
    return;
}

// check running conditions
if (_Sip.length()==0 || _PtG.length()==0) return;

Sip pnl = _Sip[0];

Point3d pts = _Pt0;
Point3d ptE = _PtG[0];
Vector3d vecX = ptE-pts;
Vector3d vecZ = pnl.vecZ();
Vector3d vecY = vecX.crossProduct(vecZ);
vecY.normalize();

PanelWirechase ps(pts, ptE, vecY, dYWidth, dZHeight);
pnl.addTool(ps);

Display dp(-1);
PLine pl(pts,ptE);
dp.draw(pl);

```

—end example]

## 8.34 ParHouse

The Parhouse tool will add a parallel housing, also called plumb housing, into the mortise beam. The constructor has the following formats:

```
ParHouse name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
ParHouse name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth);
ParHouse name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
    double dZFlag);
```

ptOrg:	origin point of the box defining the housing operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

The vectors vecX and vecY determine the direction of the parallel lines that define the cutout of the ParHouse.

To specify the dimensions of the housing, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the housing is the same as **vecX.length()\*dXWidth**. If **vecX** is a unit-length vector, then the **dXWidth** corresponds with the width. If **dXWidth** equals 1, then the length of the vector expresses the height of the housing.

The flags dXFlag, dYFlag and dZFlag, specify the position of the ptOrg inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point ptOrg is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -vecX, -vecY and -vecZ direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the vecX, vecY and vecZ coordinate system with origin ptOrg.

The housing has a round type. It can be set separately by the function [setRoundType](#).

```
ParHouse::setRoundType(int nRoundType);
```

Values of nRoundType:

<b>_kNotRound:</b>	not rounded
<b>_kRound:</b>	rounded
<b>_kRelief:</b>	relief
<b>_kRoundSmall:</b>	rounded with small diameter
<b>_kReliefSmall:</b>	relief with small diameter

The solid body that represents this tool can be used by calling

```
Body cuttingBody() const;
```

See [General](#) for member functions applicable to all tools.

---

[Example T-type:]

```

Unit(1, "mm"); // use mm as unit in U() function from now on
double dDepth = U(40);

// Calculate the vectors of the ParHouse tool
Vector3d vX = _Pt2-_Pt1;
double dX = vX.length(); vX.normalize();
Vector3d vY = _Pt3-_Pt2;
double dY = vY.length(); vY.normalize();
Vector3d vZ = _Z1;
vX.vis(_Pt0);
vY.vis(_Pt0);
vZ.vis(_Pt0);

ParHouse ph(_Pt0, vX, vY, vZ, dX, dY, dDepth, 0, 0, 1);
_Beam1.addTool(ph);

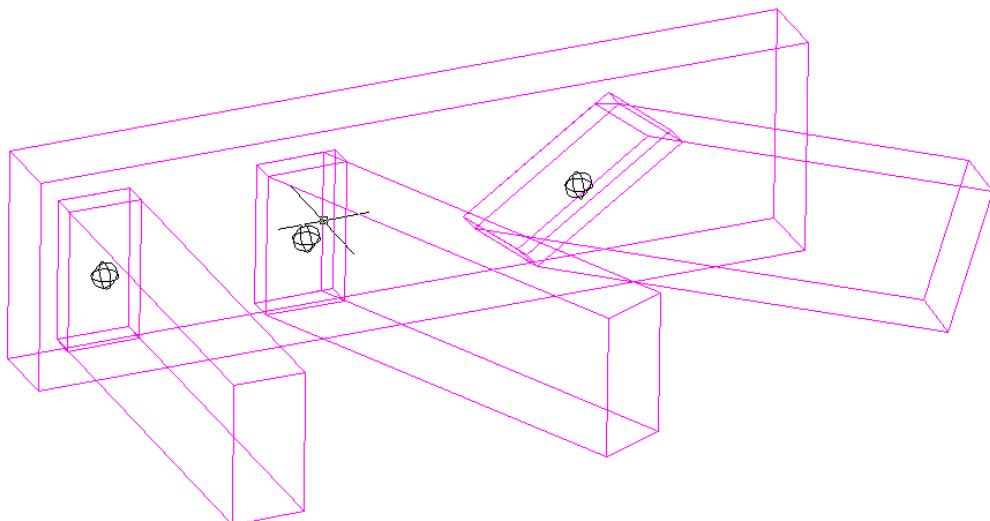
// Calculate the cut tools for _Beam0
// endcut
Cut ct(_Pt0+dDepth*_Z1, _Z1);
_Beam0.addTool(ct, 1);

// cut, parallel with vX (and _Z1), at a distance dY/2 of the _Pt0
Vector3d vCut2 = _Z1.crossProduct(vX);
if (!!(vCut2.isPerpendicularTo(_X0))) { // only cut if it makes sense
    if (vCut2.dotProduct(_X0)<0) vCut2 = -vCut2; // make sure the vCut2 is pointing in the _X0 direction
    if (vY.dotProduct(_X0)<0) vY = -vY; // make sure the vY is pointing in the _X0 direction
    vCut2.normalize();
    Cut ct2(_Pt0+0.5*dY*vY, vCut2);
    _Beam0.addTool(ct2, 0);
}

// cut, parallel with vY (and _Z1), at a distance dX/2 of the _Pt0
Vector3d vCut3 = _Z1.crossProduct(vY);
if (!!(vCut3.isPerpendicularTo(_X0))) { // only cut if it makes sense
    if (vCut3.dotProduct(_X0)<0) vCut3= -vCut3; // make sure the vCut3 is pointing in the _X0 direction
    if (vX.dotProduct(_X0)<0) vX = -vX; // make sure the vX is pointing in the _X0 direction
    vCut3.normalize();
    Cut ct3(_Pt0+0.5*dX*vX, vCut3);
    _Beam0.addTool(ct3, 0);
}

```

---



*—end example]*

## 8.35 PropellerSurfaceTool

The PropellerSurfaceTool tool (added since hsbCAD2015 build 20.0.6) its constructor has the following format:

```
PropellerSurfaceTool name(PLine plDefining, PLine plBevel, double dMillDiam, double dMaxDeviation);
```

plDefining, plBevel: the 2 PLines that are used to describe the surface  
 dMillDiam: offset dimension that is used to create a body from the surface that is subtracted from the beam solid  
 dMaxDeviation: maximum deviation allowed between tool definition and solid approximation, the machining is not dependent on this value.

The PropellerSurfaceTool tool is not an end tool. It cannot be stretched away on its own. To have endTool behaviour, the etool needs to have at least one end tool, eg a Cut.

See [General](#) for member functions applicable to all tools.

The solid body that represents this tool can be used by calling

```
Body cuttingBody() const;
```

The type of mill that is used during the machining can be specified by the setCncMode routine. The default value is `_kFingerMill`, also 0. The other values are `_kUniversalMill`, and `_kVerticalFingerMill`. Any int can be specified.

```
PropellerSurfaceTool::setCncMode(int nMode);
```

Walking along the plDefining (from start to end) with plBevel below the surface, the mill side can be either left, center or right from the plDefining. The normal of the pline is not important, it is the position of the plBevel that defines the reference for left, and right.

<b>_kLeft:</b>	-1
<b>_kCenter:</b>	0
<b>_kRight:</b>	1 (default value)

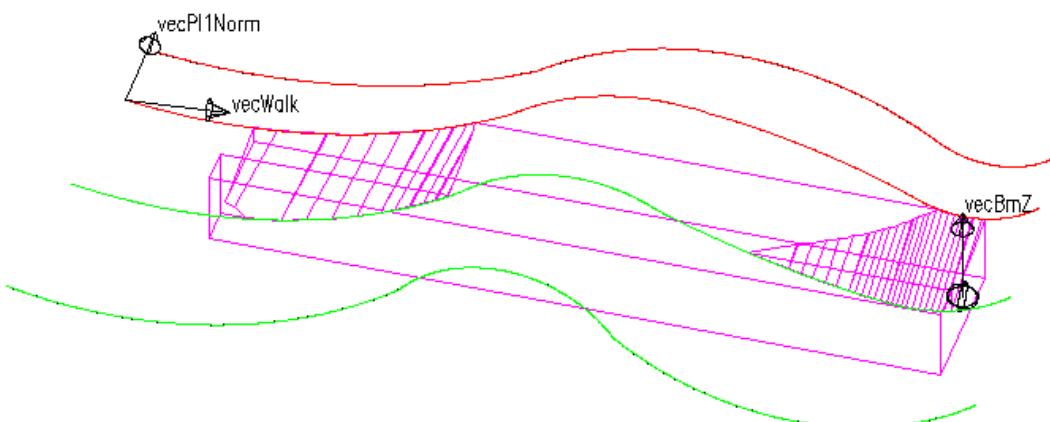
```
PropellerSurfaceTool::setMillSide(int nMillSide);
```

Other methods:

```
PropellerSurfaceTool::setMaximumDeviation(double dMaxDeviation);
PropellerSurfaceTool::setMillDiameter(double dMillDiameter);
```

**IMPORTANT:** Both parameters dMaxDeviation and dMillDiameter are only used for visualisation, and construction of the solid. Machining is not dependent on that. For machining the actual PLines are translated to the machine, and the milling diameter is derived by the cnc tool, specified through the CncMode. This also means that the PLines need to be sufficiently smooth, depending on the machine. To get PLines that are composed with arcs and have continuously changing tangents, one should use [PLine::createSmoothArcsApproximation](#) or use bReduce = TRUE in the [PropellerSurface::intersectWithPlane](#) method.

[Example E-type:



```
Unit(1, "mm"); // use mm as unit in U() function from now on

String arStrLCR[] = {T("|Left|"), T("|Center|"), T("|Right|")};
int arIntLCR[] = {_kLeft, _kCenter, _kRight};
PropString pLCR(0, arStrLCR, T("|Offset direction|"), 2);
int iLCR = arIntLCR[arStrLCR.find(pLCR, 2)];

String arSCncMode[] = {T("|Finger Mill|"), T("|Universal Mill|"), T("|Vertical Finger Mill|"),
"3", "4", "5", "6", "7", "8", "9", "10"};
```

```

int arNCncMode[] = {_kFingerMill, _kUniversalMill,
_kVerticalFingerMill, 3, 4, 5, 6, 7, 8, 9, 10 };
PropString pCncMode(1,arSCncMode,T("|Cnc mode|"));
int nCncMode = arNCncMode[arSCncMode.find(pCncMode,0)];

String arYN[] = {T("|No|"), T("|Yes|")};
PropString pYNZPos(3,arYN,T("|First pline is on z pos|"), 1);
int nUseZPos = (pYNZPos == arYN[1]);

PropDouble pMillDiam(0,U(30),T("|Mill diameter|"));
PropDouble pDevMax(1,U(2),T("|Maximum allowed deviation|"));
PropDouble pDepth(2,U(40),T("|Depth of operation|"));

if (_bOnInsert)
{
    _Map.setEntity("plEnt1", getEntPLine(T("|Select first pline|")));
    _Map.setEntity("plEnt2", getEntPLine(T("|Select second pline|")));
    Beam bm = getBeam();
    _Beam.append(bm);
    _Pt0 = getPoint();
    return;
}

Entity ent1 = _Map.getEntity("plEnt1");
Entity ent2 = _Map.getEntity("plEnt2");
EntPLine plEnt1 = (EntPLine)ent1;
EntPLine plEnt2 = (EntPLine)ent2;
Beam bm0 = _Beam0;
Vector3d vecBmZ = _z0; // vector controlled by E property, most
aligned with _zu at time of insert

if (!plEnt1.bIsValid() || !plEnt2.bIsValid() || !bm0.bIsValid())
{
    eraseInstance();
    return;
}

PLine pl1 = plEnt1.getPLine();
pl1.vis(1);
PLine pl2 = plEnt2.getPLine();
pl2.vis(2);
Point3d ptCenBm = bm0.ptCen();

// decide which face of the beam will be used for intersection with
PropellerSurface
Vector3d vecBmFace = bm0.vecD(vecBmZ.crossProduct(bm0.vecX()));
double dBmFace = bm0.dD(vecBmFace);

double dDepth = pDepth;
if (dDepth<U(0.001))
    dDepth = dBmFace;

```

```

// define the beam faces
Plane planeBm1;
Plane planeBm2;
if (nUseZPos)
{
    Point3d ptDefiningPlane = ptCenBm + 0.5*dBmFace*vecBmFace;
    planeBm1 = Plane(ptDefiningPlane , vecBmFace);
    planeBm2 = Plane(ptDefiningPlane - dDepth*vecBmFace, -vecBmFace);
}
else
{
    Point3d ptDefiningPlane = ptCenBm - 0.5*dBmFace*vecBmFace;
    planeBm1 = Plane(ptDefiningPlane , -vecBmFace);
    planeBm2 = Plane(ptDefiningPlane + dDepth*vecBmFace, vecBmFace);
}

// propeller surface
PropellerSurface ps(p11, p12, pDevMax);

// find intersecting plines with beam faces
PLine plBm1, plBm2;
{
    PLine pls[] = ps.intersectWithPlane(planeBm1, TRUE);
    if (pls.length()!=1) {
        reportMessage(T("|Intersection with beam face cannot be
used. Intersection curves found:| "+pls.length()));
        return;
    }
    plBm1 = pls[0];
    plBm1.vis(1);
}
{
    PLine pls[] = ps.intersectWithPlane(planeBm2, TRUE);
    if (pls.length()!=1) {
        reportMessage(T("|Intersection with beam face cannot be
used. Intersection curves found:| "+pls.length()));
        return;
    }
    plBm2 = pls[0];
    plBm2.vis(2);
}

// Walking along plBm1 defines left or right.
vecBmZ.vis(_Pt0);
Point3d ptP11Start = plBm1.ptStart();
Vector3d vecWalk = plBm1.ptEnd()-plBm1.ptStart();
vecWalk.normalize();
Vector3d vecP11Norm = plBm1.coordSys().vecZ();
vecWalk.vis(ptP11Start);
vecP11Norm.vis(ptP11Start);

```

```

PropellerSurfaceTool tt(plBm1, plBm2, pMillDiam, pDevMax);
tt.setMillSide(iLCR);
tt.setCncMode(nCncMode);
bm0.addTool(tt);

if (_bOnDebug && 0)
{
    Body bd = tt.cuttingBody();
    bd.vis();
}

```

*—end example]*

## 8.36 Rabbet

The rabbet (falz) tool constructor has one of the following formats:

```

Rabbet name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Rabbet name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth);
Rabbet name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double
    dXWidth, double dYHeight, double dZDepth, double dXFlag, double dYFlag,
    double dZFlag);

```

ptOrg:	origin point of the box defining the operation
vecX:	axis in width direction
vecY:	height direction
vecZ:	depth direction
dXWidth:	width scale factor
dYHeight:	height scale factor
dZDepth:	depth scale factor

To specify the dimensions of the tool, the length of the vector is multiplied with the scale factor in that direction. Eg.: the height of the tool is the same as **vecY.length()\*dYHeight**. If **vecY** is a unit-length vector, then the **dYHeight** corresponds with the Height. If **dYHeight** equals 1, then the length of the vector expresses the height.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -**vecX**, -**vecY** and -**vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

The rabbet has an overshoot type, which can be set separately by the function **setOverShoot**.

```
Rabbet::setOverShoot(int nOverShoot);
```

Values of **nOverShoot** are:

<b>_kNoOverShoot:</b>	no overshoot
<b>_kOverShoot:</b>	overshoot

Important constraints for the rabbet tool are:

- the direction that the tool enters the timber is the vecZ direction
- the vecY direction is along the rotation axis of the main mill.
- the vecX direction must coincide with the X axis of the timber

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;

See [General](#) for member functions applicable to all tools.

---

[Example T-type:

```
Unit(1, "mm"); // use mm as unit in U() function from now on

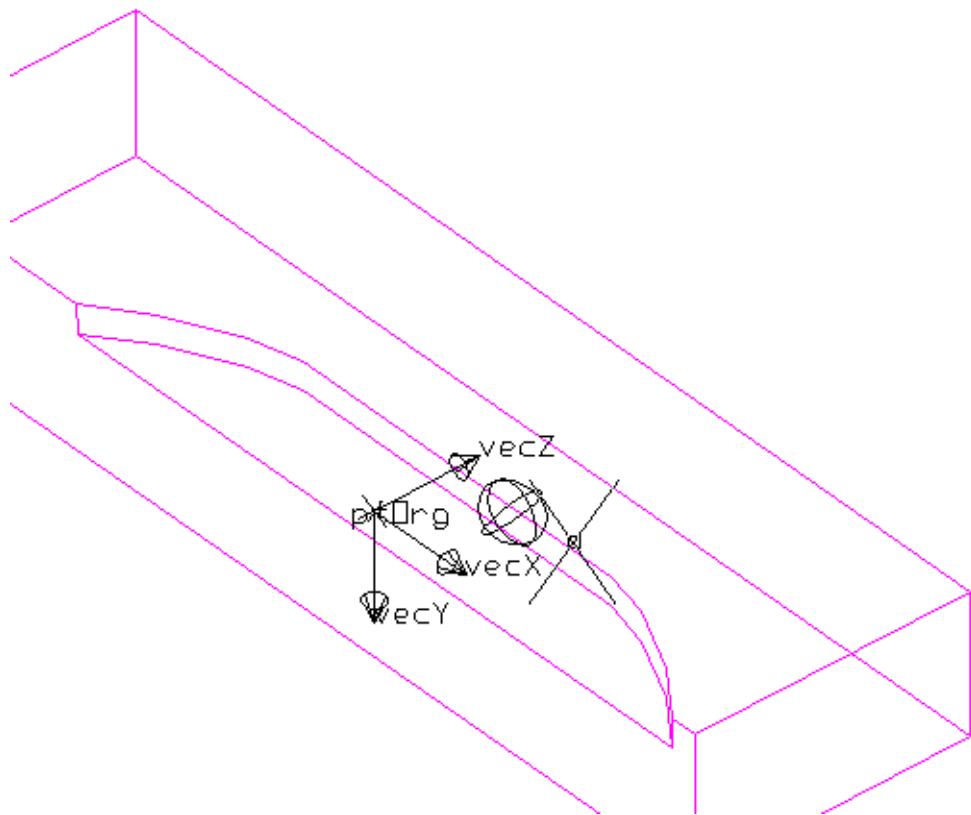
String arAlignYAxis[] = {"_Y0", "-_Y0"};
PropString pAlignYAxis(0, arAlignYAxis, "Align Y-axis");
int nSign = -1;
if (pAlignYAxis==arAlignYAxis[0]) nSign = 1;

double dXW= U(500);
double dYH= U(20);
double dZD= U(60);

// rotate the coordinate system with the _Z0 axis
Vector3d vecY = nSign*_Y0;
Vector3d vecZ = -_Z0; // _Z0 initially most aligned with _ZW => height direction
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and vecZ
Point3d ptOrg = _Pt0 - 0.5*_H0*vecZ - 0.5*_W0*vecY; // move point to edge

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

Rabbit rb(ptOrg, vecX, vecY, vecZ, dXW, dYH, dZD, 0, 1, 1); // the ptOrg is not the center point of
the box, but shifted in -vecZ and -vecY direction
rb.setOverShoot(_kNoOverShoot);
_Beam0.addTool(rb);
```



*—end example]*

## 8.37 RevolutionMill

The RevolutionMill tool (added since hsbCAD2009+, build 14.0.36) will cut the timber with a custom shaped mill head. The constructor has the following formats:

**RevolutionMill** name(**Point3d** ptOrg, **Vector3d** vecX, **Vector3d** vecY, **Vector3d** vecZ);

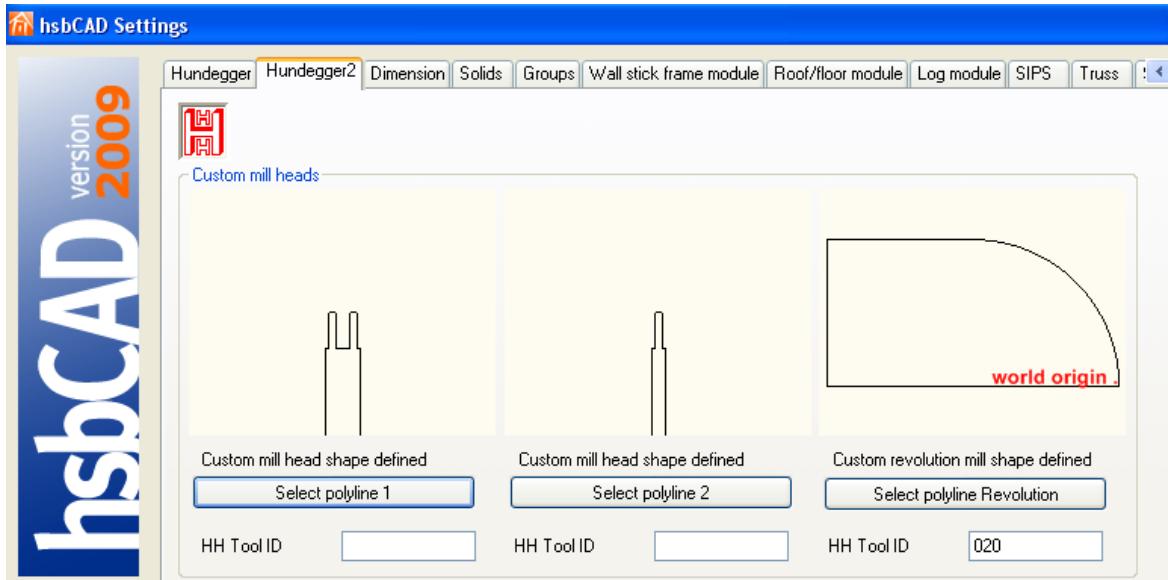
- ptOrg: origin point, corresponding with the world origin of the polyline definition if the dDepth parameter is zero
- vecX: direction normal to the polyline shape, so equal to the direction of the mill path
- vecY: corresponds with the y axis of the profile.
- vecZ: corresponds with the y axis of the profile as well (after rotation around the x axis).

The index of the mill used is 0 by default, but another index can be set by the **setToolIndex** method. The index determines what polyline is applied, defined in the Hsb\_Settings dialog.

**RevolutionMill::setToolIndex(int nIndex);**

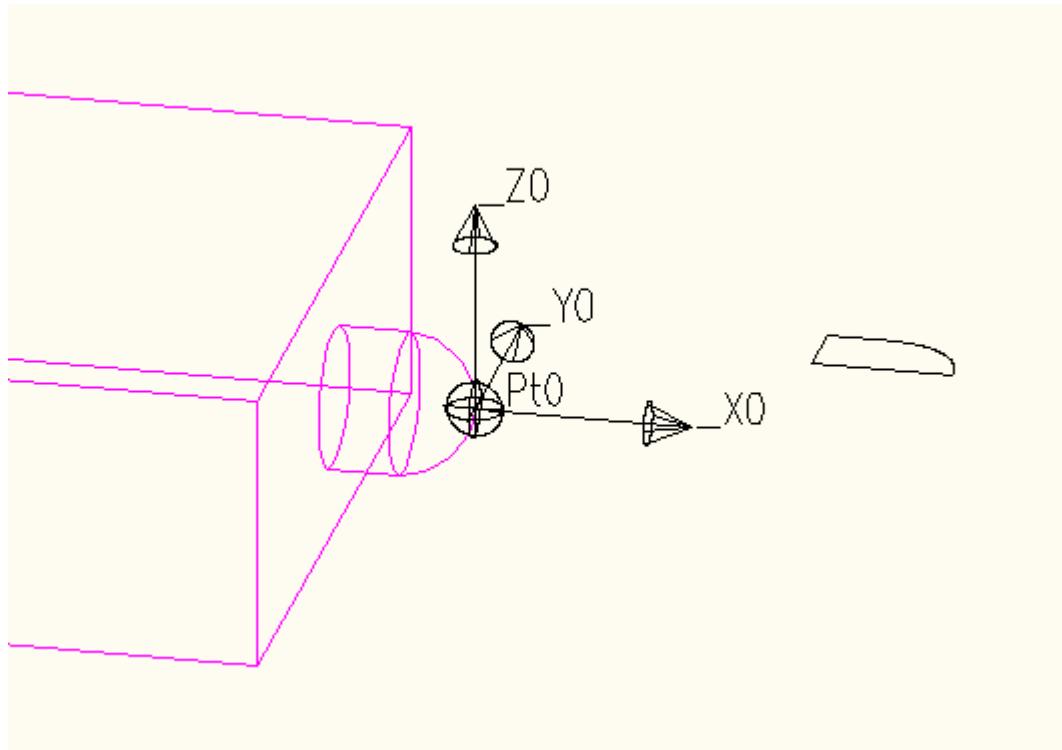
The vecX and vecY correspond with the xy plane of the profile. The profile that will be used, needs to be defined inside the Hundegger2 tab of the Hsb\_settings dialog (see figure). There are a number of constraints on the definition of the profile:

- needs to be defined in the x y plane, with x the tool direction
- the profile needs to be closed
- the profile needs to be defined with the world origin as in the image below.



See [General](#) for member functions applicable to all tools.

*[Example E-type:*



```

U(1, "mm"); // use mm

X0.vis(_Pt0);
Y0.vis(_Pt0);
Z0.vis(_Pt0);
Pt0.vis();

int nArIndex[] = {0}; // currently only index 0 is supported
PropInt nIndex(0, nArIndex, T("Mill index"));

// define special mill tool
RevolutionMill sm(_Pt0, _Y0, _Z0, _X0);
sm.setToolIndex(nIndex);
Beam0.addTool(sm);

// define additional cut
double dDistCut = 0;
Cut ct(_Pt0+_X0*dDistCut, _X0);

Beam0.addTool(ct, 1);

—end example]

```

## 8.38 Round types

The Mortise and House and the ParHouse toolings have a round type parameter in their constructor. These types also have a member function to set the round type:

```
setRoundType(int nRoundType);
```

Values of nRoundType valid for House:

<code>_kNotRound:</code>	not rounded
<code>_kRound:</code>	round
<code>_kRelief:</code>	relief
<code>_kRoundSmall:</code>	rounded with small diameter
<code>_kReliefSmall;</code>	relief with small diameter
<code>_kRounded;</code>	rounded

Values of nRoundType valid for ParHouse:

<code>_kNotRound:</code>	not rounded
<code>_kRound:</code>	round
<code>_kRelief:</code>	relief
<code>_kRoundSmall:</code>	rounded with small diameter
<code>_kReliefSmall;</code>	relief with small diameter

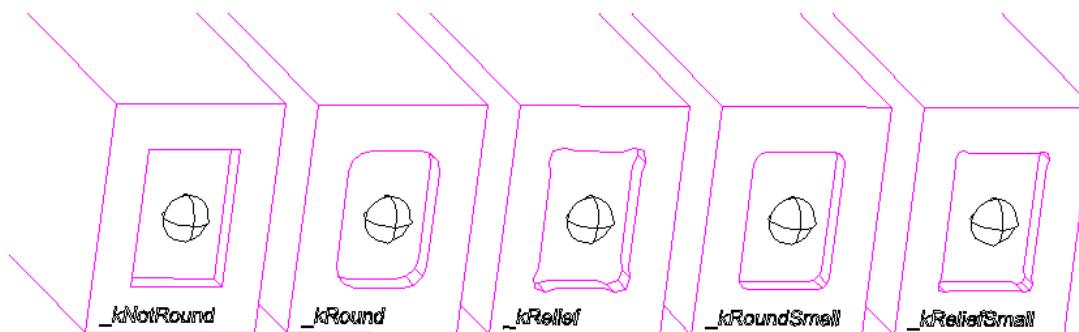
Values of nRoundType valid for Mortise:

<code>_kNotRound:</code>	not rounded
<code>_kRound:</code>	round
<code>_kRounded;</code>	rounded
<code>_kExplicitRadius:</code>	explicit radius // added v21.3.1 and v22.0.32

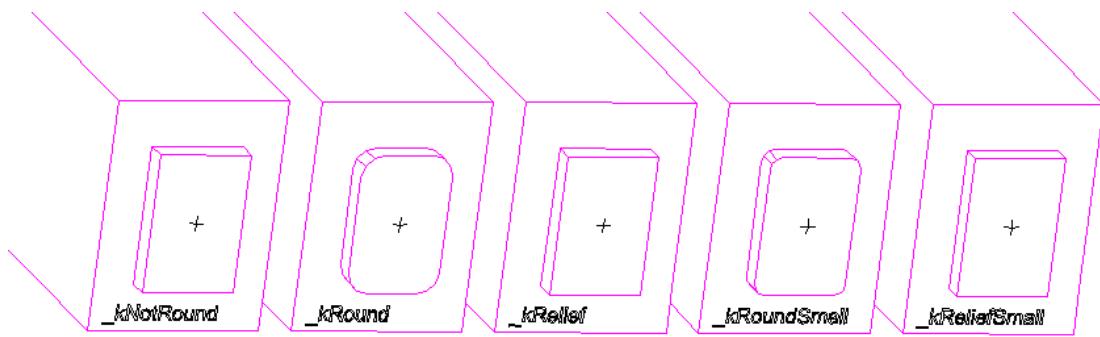
The code snap below make a property with the possible values

```
// order the round types such that their index corresponds to the value of the round
// type
String strRoundTypes[] = {"Not round", "Round", "Relief", "Rounded small","Relief
small","Rounded"};
PropString pStrRound(0,strRoundTypes,"Round type");
int nRoundType = strRoundTypes.find(pStrRound,0); // find the index of the property in
the array
```

The pictures below explain the influence of this parameter on the housing on the female side.



Below is the influence shown on the male side.



The images above were made with the following example.

[Example E-type:

```

Unit(1,"mm"); // use mm as unit in U() function from now on

// order the round types such that their index corresponds to the
// value of the round type
String arStrRoundTypes[] = {"NotRound", "Round", "Relief",
"RoundSmall", "ReliefSmall", "Rounded"};
int arNRoundTypes[] = { _kNotRound, _kRound, _kRelief, _kRoundSmall,
_kReliefSmall, _kRounded };
PropString pStrRound(0,arStrRoundTypes, "Round type");
int nRoundType = arNRoundTypes[arStrRoundTypes.find(pStrRound,0)];

double dXW= U(80);
double dYH= U(120);
double dZD= U(20);

Vector3d vecZ = -_x0; // depth along beam axis
Vector3d vecY = _z0; // _z0 initially most aligned with _zw =>
height direction
Vector3d vecX = vecY.crossProduct(vecZ); // perpendicular to vecY and
vecZ
Point3d ptOrg = _pt0; //

// visualize orientation in debug mode
Point3d ptVis = ptOrg;
ptOrg.vis();
vecX.vis(ptVis);
vecY.vis(ptVis);
vecZ.vis(ptVis);

House hs(ptOrg,vecX,vecY,vecZ,dXW,dYH,dZD,0,0,1,_kFemaleEnd,
nRoundType);
_Beam0.addTool(hs,1); // is added with parameter 1: endtool active:
will remove all other endtools during insert

```

—end example]

## 8.39 RoundWoodMill

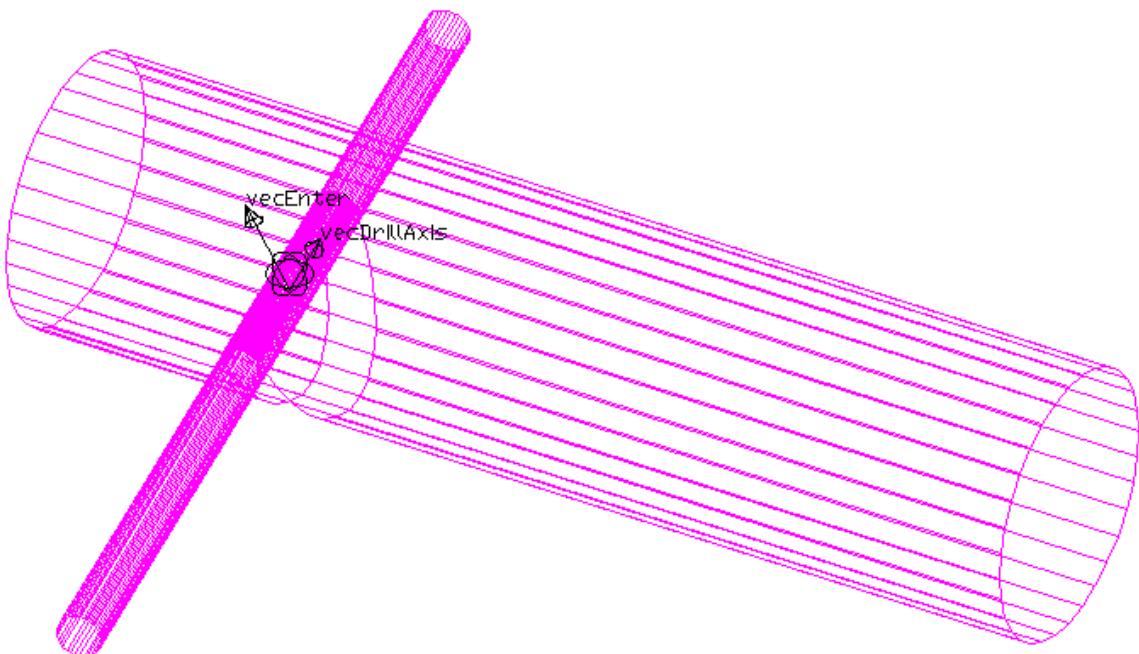
This tool, called RoundWoodMill is a log connection tool. The constructor has the following format:

```
RoundWoodMill name(Point3d ptAxis, Vector3d vecDrillAxis, Vector3d vecEnter,
double dRadius, Vector3d vecXSealingGroove, double dSealingGrooveWidth);
```

pt:	point on axis of drill
vecDrillAxis:	vector along axis drill
vecEnter:	direction from where the wood mill is coming
dRadius:	radius of the drill
vecXSealingGroove:	direction along the sealing groove, typically along the axis of the other beam
dSealingGrooveWidth:	width of the sealing groove

See [General](#) for member functions applicable to all tools.

[Example X-type:



```
U(1, "mm");  
  

Point3d ptAxis = PtMin;  

Vector3d vecDrillAxis = x0;
```

```

Vector3d vecEnter = _z0+0.5*_y0;
Vector3d vecXGroove = _x1;
double dSealingGrooveWidth = u(0);

vecDrillAxis.vis(ptAxis);
vecEnter.vis(ptAxis);
vecXGroove.vis(ptAxis);

double dRad = _w0/2;

RoundWoodMill
rwm(ptAxis,vecDrillAxis,vecEnter,dRad,vecXGroove,dSealingGrooveWidth)
;
_Beam1.addTool(rwm);

—end example]

```

## 8.40 ScarfJoint

The ScarfJoint constructor has the following format:

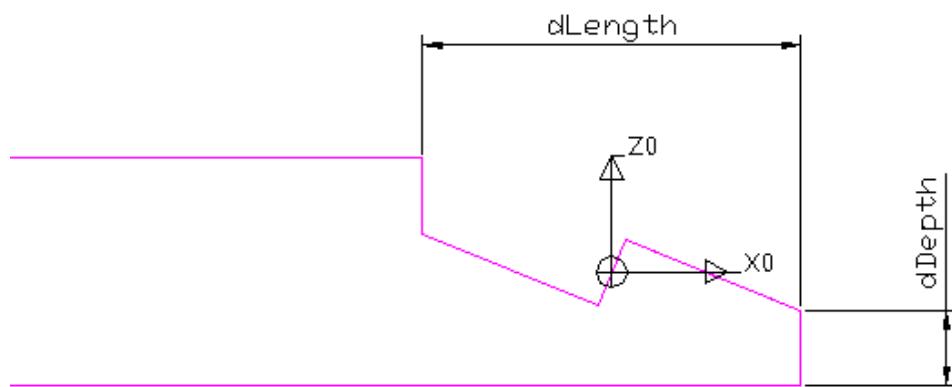
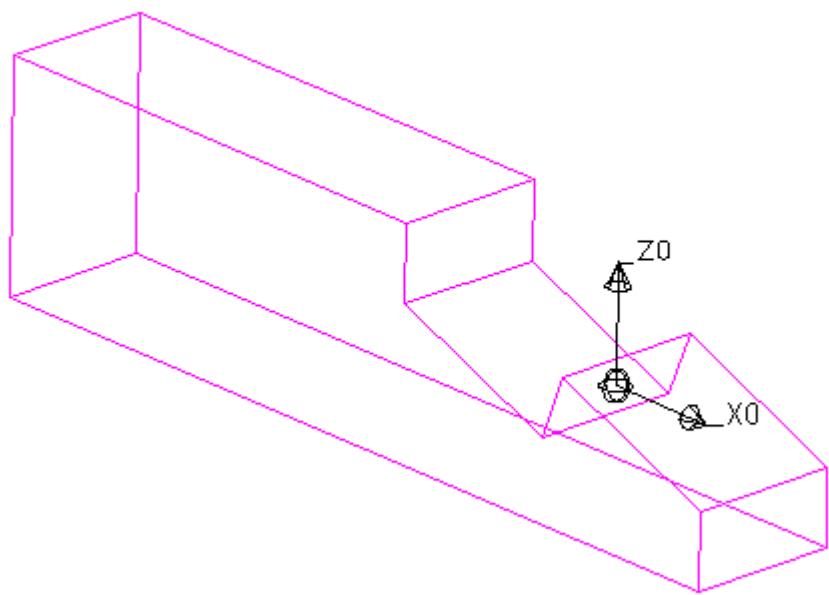
```
ScarfJoint name(Point3d ptOrg, Vector3d vecX, Vector3d vecZ, double dLength, double dDepth, double dHeightBeam);
```

ptOrg:	point on axis of beam, center point in vecX direction of operation
vecX:	axis along beam axis, pointing from beam center to tool
vecZ:	axis along beam height, positive vecZ is cut away
dLength:	length dimension of tool in vecX direction
dDepth:	depth dimension of tool in vecZ direction
dHeightBeam:	beam height, in vecZ direction

See [General](#) for member functions applicable to all tools.

---

[Example of E-type:



```
U (l,"m m ");
_X0.vis(_Pt0);
_Z0.vis(_Pt0);
ScanJointsj(_Pt0,_X0,_Z0,U (500),U (100),_H0);
_Beam 0.addTool(sj,_kStretchOnInsert);
```

*—end example]*

## 8.41 SimpleScarf

The SimpleScarf was added in hsbCAD2011 (build 16.1.15). Its constructor has the following format. The arguments of the constructor match the meaning of the properties of the [AnalysedSimpleScarf](#).

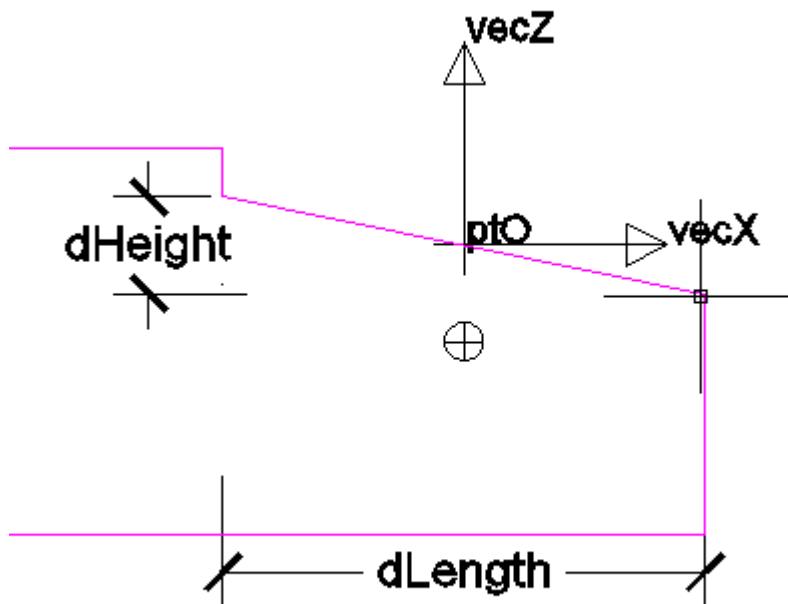
```
SimpleScarf name(Point3d ptOrg, Vector3d vecX, Vector3d vecZ, double dLength,
double dHeight);
```

ptOrg: point on center of tool, at the midpoint of the sloped rectangle  
 vecX: axis along beam axis, pointing from beam center to tool  
 vecZ: axis along beam height, positive vecZ is cut away  
 dLength: length dimension of tool in vecX direction  
 dHeight: height dimension of tool in vecZ direction

See [General](#) for member functions applicable to all tools.

---

[Example of E-type:



```
U(1, "mm");

double dOffset = U(100);
Point3d ptO = _Pt0+dOffset*_z0;
Vector3d vecX = _x0;
Vector3d vecZ = _z0;
```

---

```

ptO.vis();
vecX.vis(ptO);
vecZ.vis(ptO);

SimpleScarf ss(ptO, vecX, vecZ, U(500), U(100));
_Beam0.addTool(ss, _kStretchOnInsert);

—end example]

```

## 8.42 Slot

```

Slot name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ);
Slot name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth);
Slot name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dXWidth,
double dYHeight, double dZDepth, double dXFlag, double dYFlag, double dZFlag);

```

<b>ptOrg:</b>	origin point of the box defining the operation
<b>vecX:</b>	axis in width direction
<b>vecY:</b>	height direction
<b>vecZ:</b>	depth direction
<b>dXWidth:</b>	width scale factor
<b>dYHeight:</b>	height scale factor
<b>dZDepth:</b>	depth scale factor

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;

## 8.43 SolidSubtract

The SolidSubtract tool enables to cut away the shape from a straight cut beam. The constructor has the following format:

```

SolidSubtract name(GenBeam beamToCutAway);
SolidSubtract name(GenBeam beamToCutAway, int nMode);
SolidSubtract name(Body solidToCutAway);
SolidSubtract name(Body solidToCutAway, int nMode);

```

**beamToCutAway:** The beam of which the body is taken that will be cut away from the solid on which this tool is applied.

**solidToCutAway:** The solid that will be cut away from the solid on which this tool is applied.

**nMode:** parameter to indicate to add or subtract the body, possible values: **\_kAdd**,  
**\_kSubtract**. The default value is subtract.

The solid body that represents this tool can be used by calling

**Body** cuttingBody() const;

**Important:** This tool will not result in proper Hundegger operations on the beam. Only use if the operations on the beam can only be done manually, and no other machining operations are available. It is advised to use the CncMessage tool, to report during the CNC generation, that this tool cannot be processed on the machine.

**Important 2:** The use of SolidSubtract is expensive in terms of dwg size, since the solid itself is filed. Watch the dwg size closely.

See [General](#) for member functions applicable to all tools.

---

[Example any-type:

```
Unit(1,"mm");

double dSize = U(50);

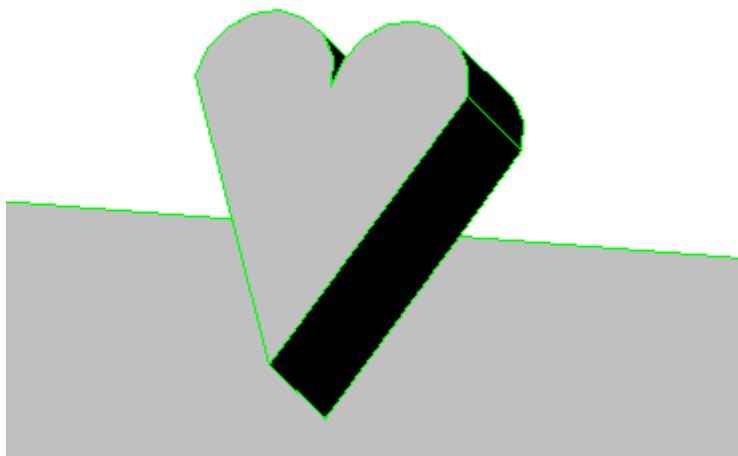
// make a polyline
PLine pline(_Z0);
Point3d pt1 = _Pt0 + 0.5*_H0*_Z0 + 0.5*dSize*_Y0 ;
pt1.vis();
pline.addVertex(pt1);
Point3d pt2 = pt1 + 0.25*dSize*_X0;
pline.addVertex(pt2,-1); // bulge 1 means half a circle
Point3d pt3 = pt1 - 0.5*dSize*_Y0;
pline.addVertex(pt3);
Point3d pt4 = pt1 - 0.25*dSize*_X0;
pline.addVertex(pt4);
pline.addVertex(pt1,-1); // close it with bulge
pline.vis();

int nMode = _kAdd;
//int nMode = _kSubtract;
int nSide = (nMode==_kAdd) ? -1 : 1;

// make a body from the polyline
Body bd(pline, -0.25*dSize*_Z0, nSide);
bd.vis();

// subtract or add the solid
SolidSubtract sosu(bd, nMode);
_Beam0.addTool(sosu);

CncMessage msg("This tool cannot be processed on a CNC machine.");
_Beam0.addTool(msg);
```



*[end example]*

## 8.44 SpecialMill

The SpecialMill tool will cut the timber with a custom shaped mill head. The constructor has the following formats:

```
SpecialMill name(Point3d ptOrg, Vector3d vecX, Vector3d vecY, Vector3d vecZ, double dDepth);
```

ptOrg:	origin point, corresponding with the world origin of the polyline definition if the dDepth parameter is zero
vecX:	direction normal to the polyline shape, so equal to the direction of the mill path
vecY:	corresponds with the x axis of the profile
vecZ:	corresponds with the y axis of the profile
dDepth:	depth in the vecZ direction

The index of the mill used is 0 by default, but another index can be set by the **setToolIndex** method. The index determines what polyline is applied, defined in the Hsb\_Settings dialog. Index 0 is the "Polyline 1", index 1 is the "Polyline 2".

```
SpecialMill::setToolIndex(int nIndex);
```

The SpecialMill with the same geometrical parameters can be used put on the head as well as on the side. The only parameter that differs is the **nEndType** parameter. To change this value on an existing SpecialMill tool, the **setEndType** function can be used.

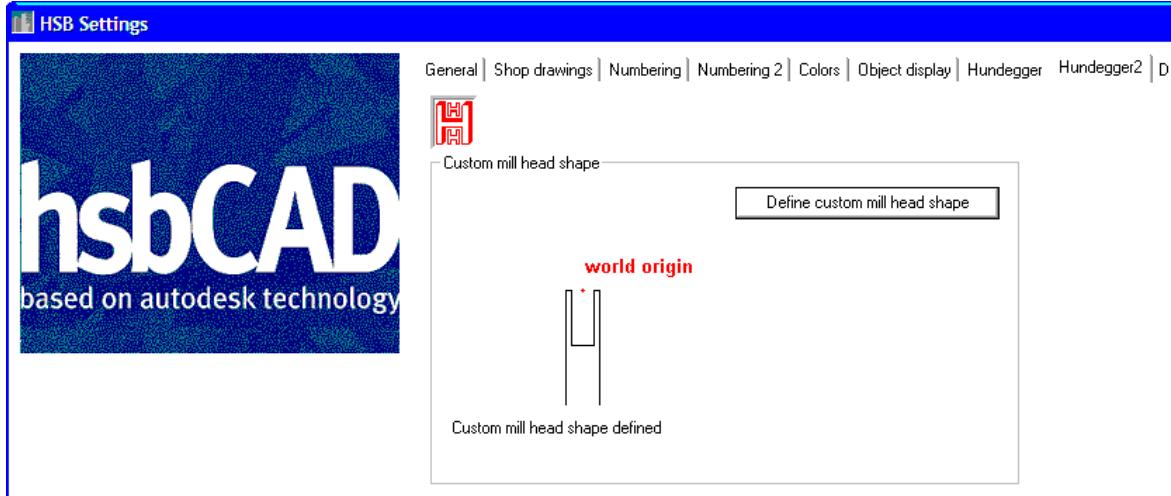
```
SpecialMill::setEndType(int nEndType);
```

Values of nEndType:

<b>_kFemaleEnd:</b>	done at end side of beam (default)
<b>_kFemaleSide:</b>	not done at end of beam

The vecY and vecZ correspond with the xy plane of the profile. The profile that will be used, needs to be defined inside the Hundegger2 tab of the Hsb\_settings dialog (see figure). There are a number of constraints on the definition of the profile:

- needs to be defined in the x y plane, with y the tool direction
- the profile needs to be closed
- the profile needs to be defined with the world origin as in the image below.



See [General](#) for member functions applicable to all tools.

*[Example T-type:*

```

U( 1, " mm" ); // use mm

_X0.vis(_Pt0);
_Y0.vis(_Pt0);
_Z0.vis(_Pt0);
_Pt0.vis();

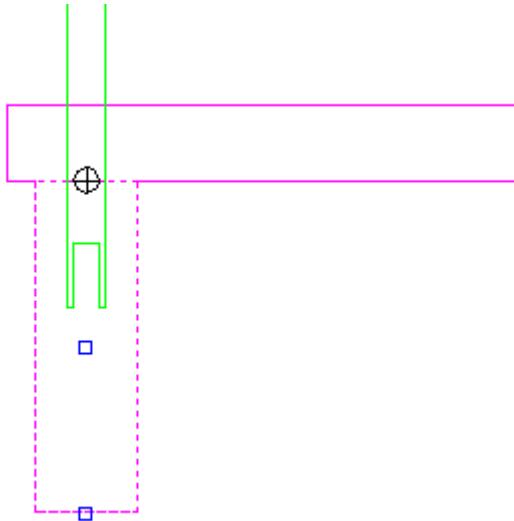
PropDouble dDepth( 0, U( 200 ), T( "| Depth| " ) );

int nArlIndex[ ] = { 0, 1 };
PropInt nIndex( 0, nArlIndex, T( "| Mill index| " ) );

// define special mill tool
SpecialMill sm(_Pt0, _Z0, _Y0, -_X0, dDepth);
sm.setToolIndex( nIndex );
_Beam0.addTool( sm );

// define additional cut
Cut ct(_Pt0, _Z1);
_Beam0.addTool( ct, 1 );

```



*—end example]*

## 8.45 TirolerSchloss

This tool, called TirolerSchloss is a log connection tool. The constructor has the following format:

```
TirolerSchloss name(Point3d pt, Vector3d vecXThis, Vector3d vecXOther, double
    dWidthBeamThis, double dWidthBeamOther, double dOffsetThis, double
    dOffsetOther, double dDovetailHeight, double dDovetailAngle, double dExtraLength,
    int nType);
```

pt: point on axis of beam

vecXThis: axis along this beam

vecXOther: axis along other beam

dWidthBeamThis: dimension of this beam, in vecXOther direction

dWidthBeamOther: dimension of other beam, in vecXThis direction

dOffsetThis: seat cut depth in vecXOther direction

dOffsetOther: seat cut depth in vecXThis direction

dDovetailHeight:

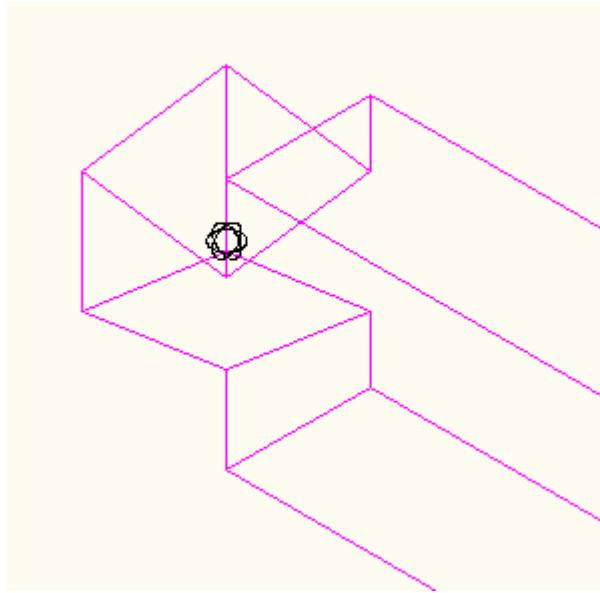
dDovetailAngle: angle in degrees

dExtraLength:

nType: type, must have a value (**\_kCConvex**, **\_kCConcave**, **\_kCDiagonal**,
**\_kTICConvex**, **\_kTICConcave**, **\_kTIDiagonal**, **\_kTEConvex**,
**\_kTEConcave**, **\_kTEDiagonal**)

See [General](#) for member functions applicable to all tools.

[Example E-type, thanks to RL:



```

U(1,"mm");
String arStrTypes[]={T("|corner convex|"),T("|corner concave|"),T("|corner diagonal|"),T("|T-inner convex|"),T("|T-inner concave|"),T("|T-inner diagonal|"),T("|T-end convex|"),T("|T-end concave|"),T("|T-end diagonal|")};
int arNTypes[]={_kCConvex, _kCConcave, _kCDiagonal, _kTIConvex,
_kTIConcave, _kTIDiagonal, _kTEConvex, _kTEConcave, _kTEDiagonal};
PropString pType(0,arStrTypes, T("|Type|"));
int nType = arNTypes[arStrTypes.find(pType,0)];

String arLocations[]={T("|Top and Bottom|"),T("|Top Only|"),T("|Bottom Only|")};
PropString pLocation(1,arLocations, T("|Apply Tool|"));

int bIsOnBottom=pLocation==arLocations[2]?1:0;
int bIsOnTop=pLocation==arLocations[1]?1:0;

_x0.vis(_Pt0);
_y0.vis(_Pt0);
_z0.vis(_Pt0);
double dWidthOther = _w0;
double dOffsetThis = U(0);
double dOffsetOther = U(0);
double dDovetailHeight = U(170);
double dDovetailAngle = 8;

```

```
double dExtraLength = U(0);

//change for bottom or top only
double dMove=U(0);
double dBmSize=_H0;
if(bIsOnBottom) {
    dMove=dBmSize/2;
    dDovetailHeight+=dBmSize;
}
if(bIsOnTop) {
    dMove=-1*dBmSize/2;
    dDovetailHeight+=dBmSize;
}

TirolerSchloss ts(_Pt0+dMove*_z0,_x0,_y0,_w0, dWidthOther,
dOffsetThis, dOffsetOther, dDovetailHeight, dDovetailAngle,
dExtraLength, nType);
_Beam0.addTool(ts);

—end example]
```

**Part**

**IX**

## 9 Analysed tools

### 9.1 AnalysedTool

The AnalysedTool type represents an evaluated tool received from a [GenBeam](#). When a tool is added to a GenBeam, it is stored inside an internal list of tools. Dimensions of the tool might be too big. The tool might be outside the actual geometry of the GenBeam. An analysed tool is constructed from the added tool, but adjusted to the actual geometry of the GenBeam. From a GenBeam the analysed tools can be received. Each AnalysedTool represents a relevant tool eg a Cut, BeamCut, ... but analysed and modified such that it represents the tool that is relevant for the beam. For instance, the BeamCut dimensions are reduced to the minimal dimensions of the required BeamCut, that still would have the same solid result.

The AnalysedTool also has a set of pertinent questions implemented. The answers to those questions should allow a straightforward evaluation of the tools output.

The analysed tools from a GenBeam are found by calling the

```
AnalysedTool[] GenBeam::analysedTools() const; // return the list of AnalysedTool that
belong to this GenBeam
AnalysedTool[] GenBeam::analysedTools(int nVerboseMode) const;
```

The convertToMap returns a collection of tools. This collection of tools has the mapkey [\\_kAnalysedTools](#) (which is of type String, with value "AnalysedTools").

---

```
class AnalysedTool { //
    AnalysedTool(); // default constructor
    AnalysedTool(Map map); // constructor with a Map

    int blsValid() const;

    String toolType() const;
    String toolSubType() const;
    String toolEntTypeName() const;

    ToolEnt toolEnt() const; // see ToolEnt
    GenBeam genBeam() const; // see GenBeam

    Map mapInternal() const; // returns the internal map.

    Quader genBeamQuader() const; // return the Quader from the GenBeam which was used
        in composing the analysed tool. For this quader the ptCenSolid and
        solidLength of the GenBeam are used.

    Map genericQuestion(String strQuestion) const; // see Map
    Map genericQuestion(String strQuestion, Vector3d vecSide) const; // see Map
    Map genericQuestion(String strQuestion, Vector3d[] arSides) const; // see Map
```

```

static AnalysedTool[] convertFromSubMap(Map map, String strSubMapKey); // convert the
submap to an array of AnalysedTool
static AnalysedTool[] convertFromSubMap(Map map, String strSubMapKey, int
nVerboseMode);

static Map convertToMap(AnalysedTool[] toolList); // convert to a map which can be added
as a submap that could be read with convertFromSubMap
static Map convertToMap(AnalysedTool tool);

static AnalysedTool[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolType);
static AnalysedTool[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolType, String strToolSubType);
static AnalysedTool[] filterToolsOfEToolType(AnalysedTool[] toolListToFilter, String
strEToolType);

};


```

---

```

static AnalysedTool[] convertFromSubMap(Map map, String strSubMapKey);
static AnalysedTool[] convertFromSubMap(Map map, String strSubMapKey, int nVerboseMode);


```

The sub map with name strSubMapKey is converted into a list of AnalysedTool. If the strSubMapKey name is empty, the map itself is converted. Each AnalysedTool can indeed be represented by a Map. If nVerboseMode is not 0 (which is the default value), error reporting is done. If nVerboseMode is 1, the reportMessage is used, while the reportNotice is used for nVerboseMode equal to 2.

The method is a static method, which means that the contents of the instance itself is not used.

```
static Map convertToMap(AnalysedTool[] toolList);
```

The map representation of each tool from the toolList is collected into a new map which is returned.

```
static Map convertToMap(AnalysedTool tool);
Map mapInternal() const;
```

Both methods return the map representation of one tool. The mapInternal returns the map directly. The convertToMap has the map returned as a submap of another map, similar to the convertToMap of an array of tools.

```

static AnalysedTool[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolType)
static AnalysedTool[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolType,
String strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument. Each of the returned AnalysedTool has its toolType set equal to strToolType. In case the strToolSubType is specified, and not an

empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

```
static AnalysedTool[] filterToolsOfEToolType(AnalysedTool[] toolListToFilter, String strEToolType)
```

The list returned is a sub list of the toolListToFilter argument. Each of the returned AnalysedTool has its eToolType set equal to strEToolType. The match is not case sensitive.

*[Example E-type that retrieves the analysed tools from \_Beam0, and will give output similar to:*

```
/*
analysedTools reports:

analysedTools recognized tools:
Tool: 0 AnalysedBeamCut[2130641936] of subtype _kABCSeatCut at (3234.19,-3949.02,100.013)
toolType: AnalysedBeamCut
toolSubType: _kABCSeatCut
toolEntTypeName: Hsb_Ebeamcut
toolEnt: ToolEnt(2130641936)
genBeam: GenBeam(2130641480)
mpTool: Map(AnalysedBeamCut)
{MAV,MIV,AnalysedTool,ptOrg,vecX,vecY,vecZ,dX,dY,dZ,bIsFreeXP,bIsFreeXN,bIsFreeYP,bIsFreeYN,bIsFreeZP,bIsFreeZN,bForCncReplaceWithSawcut,bModifySectionForCnC}

Tool: 1 AnalysedDrill[2130641488] of subtype _kADPerpendicular at (1512.7,-4004.68,200.028)
toolType: AnalysedDrill
toolSubType: _kADPerpendicular
toolEntTypeName: Hsb_Esurface_drill
toolEnt: ToolEnt(2130641488)
genBeam: GenBeam(2130641480)
mpTool: Map(AnalysedDrill)
{MAV,MIV,AnalysedTool,ptStart,ptEnd,dDiameter,bUseThisDirection,bDrillBit,bZNFree,bZPFree}

Tool: 2 AnalysedCut[0] of subtype _kACPPerpendicular at (3877.33,-3999.03,100.013)
toolType: AnalysedCut
toolSubType: _kACPPerpendicular
toolEntTypeName:
toolEnt: ToolEnt(0)
genBeam: GenBeam(2130641480)
mpTool: Map(AnalysedCut)
{MAV,MIV,AnalysedTool,ptOrg,normal,mustCut,touchesBody,bodyPointsInPlane}

Tool: 3 AnalysedCut[0] of subtype _kACPPerpendicular at (1165.15,-3999.03,100.013)
toolType: AnalysedCut
toolSubType: _kACPPerpendicular
toolEntTypeName:
```

```

toolEnt: ToolEnt(0)
genBeam: GenBeam(2130641480)
mpTool: Map(AnalysedCut)
{MAV,MIV,AnalysedTool,ptOrg,normal,mustCut,touchesBody,bodyPointsInPlane}

_kAnalysedTools
converToMap: Map(AnalysedTools){AnalysedBeamCut,AnalysedDrill,AnalysedCut,AnalysedCut}
*/
reportMessage("\n\nanalysedTools reports: ");

int nReportMessage = 1;
AnalysedTool tls[] = _Beam0.analysedTools(nReportMessage);

reportMessage("\n\nanalysedTools recognized tools: ");
for (int i=0; i<tls.length(); i++) {
    AnalysedTool at = tls[i];
    reportMessage("\nTool: "+i+" "+at);
    reportMessage("\n"+ "toolType: "+at.toolType());
    reportMessage("\n"+ "toolSubType: "+at.toolSubType());
    reportMessage("\n"+ "toolEntTypeName: "+at.toolEntTypeName());
    reportMessage("\n"+ "toolEnt: "+at.toolEnt());
    reportMessage("\n"+ "genBeam: "+at.genBeam());
    Map mpTool = at.mapInternal();
    reportMessage("\n"+ "mpTool: "+mpTool);
    reportMessage("\n");
}

Map mpContents = AnalysedTool().convertToMap(tls);
String strMapKey = mpContents.getMapKey();
if (strMapKey==_kAnalysedTools) {
    reportMessage("\n"+"_kAnalysedTools");
}
reportMessage("\n"+ "converToMap: "+mpContents);
reportMessage("\n");

```

*—end example]*

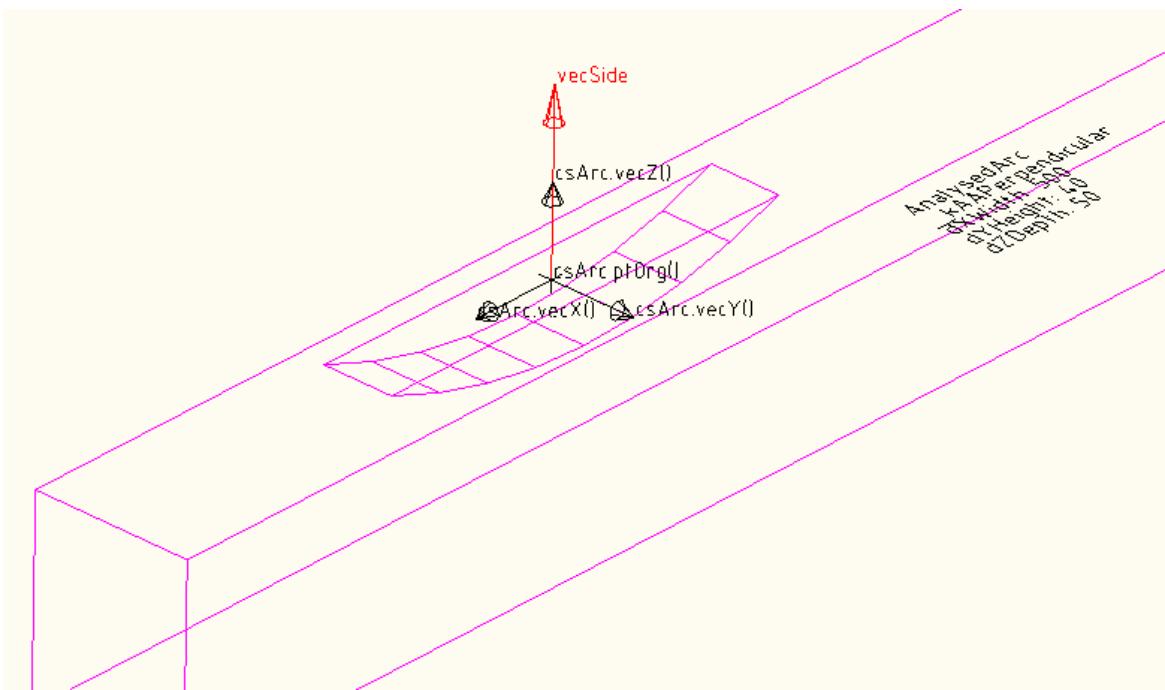
## 9.2 AnalysedArc

The AnalysedArc type represents an evaluated tool of type [Arc](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedArc. However when the casting is not allowed, the resulting AnalysedArc will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedArc is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type [String](#):

- `_kAAPerpendicular, _kAA5Axis`



```

class AnalysedArc : AnalysedTool { // derived from AnalysedTool (added since 14.0.73)

    AnalysedArc(); // default constructor
    AnalysedArc(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
                           // tool
    CoordSys coordSys() const; // return the CoordSys

    double dXWidth() const;
    double dYHeight() const;
    double dZDepth() const;

    Vector3d vecSide() const; // relevant face for Arc

    static AnalysedArc\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedArc\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String
                                              strToolSubType);

    static int findClosest(AnalysedArc\[\] toolListToSearch, Point3d ptRef); // returns -1 or index in
                           // array.
};

```

```
static int findClosest(AnalysedArc[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedArc which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedArc[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedArc[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedArc, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedArc anArcs[] = AnalysedArc().filterToolsOfToolType(tools);

int nIndClosest = AnalysedArc().findClosest(anArcs, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedArc anArc = anArcs[nIndClosest];

// retrieve some data from the Arc itself
CoordSys csArc = anArc.coordSys();
csArc.vis();

Point3d ptOrg = anArc.ptOrg();
Vector3d vecSide = 2*anArc.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(anArc.toolType());
```

```

strLines.append(anArc.toolSubType());
strLines.append("dXWidth: "+anArc.dXWidth());
strLines.append("dYHeight: "+anArc.dYHeight());
strLines.append("dZDepth: "+anArc.dZDepth());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

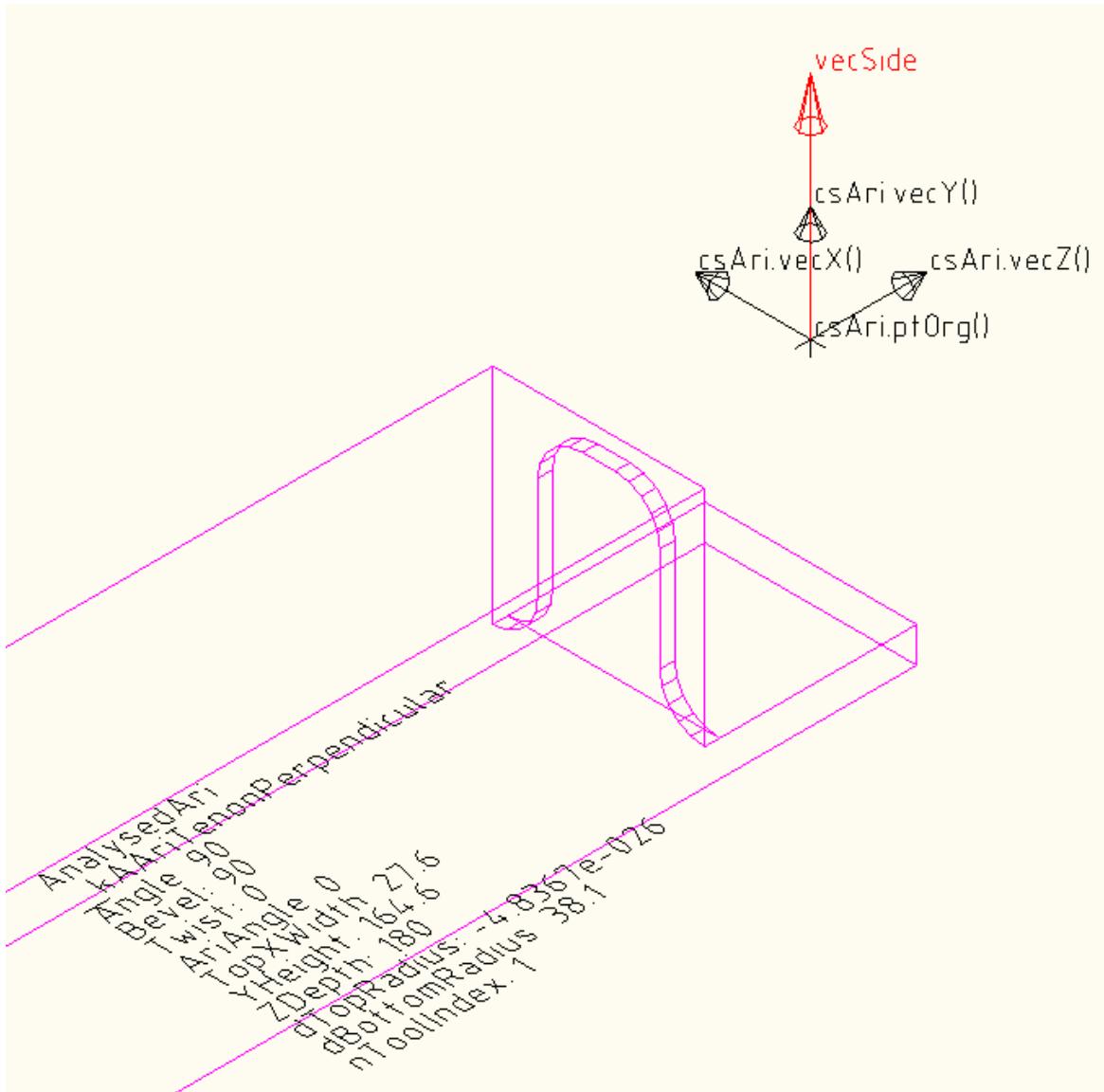
### 9.3 AnalysedAri

The AnalysedAri type represents an evaluated tool of type Ari. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedAri. However when the casting is not allowed, the resulting AnalysedAri will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedAri is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:

- `_kAAriPerpendicular, _kAAriRotated, _kAAriTilted, _kAAri5Axis,`
- `_kAAriHeadPerpendicular, _kAAriHeadSimpleAngled,`
- `_kAAriHeadSimpleAngledTwisted, _kAAriHeadSimpleBeveled, _kAAriHeadCompound,`
- `_kAAriTenonPerpendicular, _kAAriTenonSimpleAngled,`
- `_kAAriTenonSimpleAngledTwisted, _kAAriTenonSimpleBeveled,`
- `_kAAriTenonCompound`




---

```
class AnalysedAri : AnalysedTool { // derived from AnalysedTool (added since 14.0.63)
```

```
AnalysedAri(); // default constructor
AnalysedAri(Map map); // constructor with a Map, check validity is recommended

Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the tool
CoordSys coordSys() const; // return the CoordSys

double dAngle() const; // value in degrees
double dBevel() const; // value in degrees
double dTwist() const; // value in degrees
double dAriAngle() const; // value in degrees
double dTopXWidth() const;
```

```

double dYHeight() const;
double dZDepth() const;
double dTopRadius() const;
double dBottomRadius() const;

int nToolIndex() const;

Vector3d vecSide() const; // relevant face for Ari

static AnalysedAri[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedAri[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType);

static int findClosest(AnalysedAri[] toolListToSearch, Point3d ptRef); // retuns -1 or index in array.
};

```

---

```
static int findClosest(AnalysedAri[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedAri which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedAri[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedAri[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedAri, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedAri Aris[] = AnalysedAri().filterToolsOfToolType(tools);

int nIndClosest = AnalysedAri().findClosest(Aris, _Pt0);
if (nIndClosest<0) {
    reportMessage("\\n"+scriptName() +": No tool found. Instance erased.");
    eraseInstance(); // calling eraseInstance will notify that the tool is not consumed.
    return;
}

```

---

```

AnalysedAri ari = Aris[nIndClosest];

// retrieve some data from the ari itself
CoordSys csAri = ari.coordSys();
csAri.vis();

Point3d ptOrg = ari.ptOrg();
Vector3d vecSide = 2*ari.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(ari.toolType());
strLines.append(ari.toolSubType());
strLines.append("Angle: "+ari.dAngle());
strLines.append("Bevel: "+ari.dBevel());
strLines.append("Twist: "+ari.dTwist());
strLines.append("AriAngle: "+ari.dAriAngle());
strLines.append("TopXWidth: "+ari.dTopXWidth());
strLines.append("YHeight: "+ari.dYHeight());
strLines.append("ZDepth: "+ari.dZDepth());
strLines.append("dTTopRadius: "+ari.dTTopRadius());
strLines.append("dBBottomRadius: "+ari.dBBottomRadius());
strLines.append("nToolIndex: "+ari.nToolIndex());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end example*

## 9.4 AnalysedBeamCut

The AnalysedBeamCut type represents an evaluated tool of type BeamCut. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedBeamCut. However when the casting is not allowed, the resulting AnalysedBeamCut will become invalid. This can be checked with the blsValid() function of AnalysedTool. Because AnalysedBeamCut is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

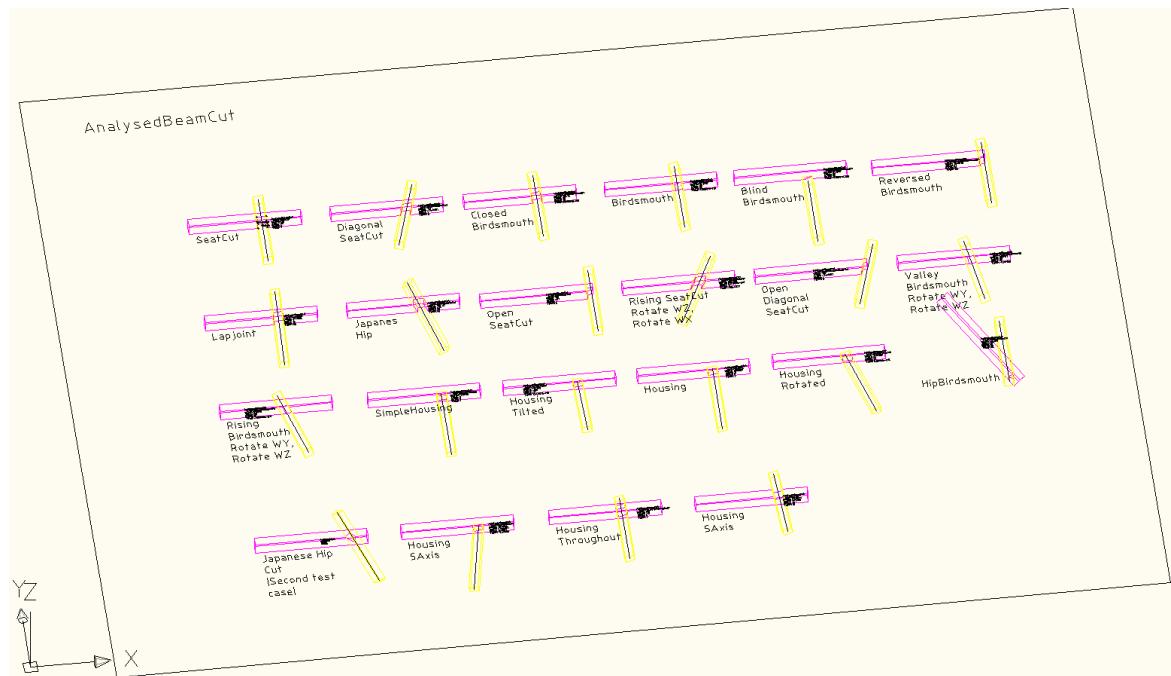
The [AnalysedTool::toolSubType\(\)](#) returns one of the following predefined variables of type **String**:

- \_kABCSeatCut**, **\_kABCRisingSeatCut**, **\_kABCOpenSeatCut**, **\_kABCLapJoint**,
- \_kABCBirdsmouth**, **\_kABCReversedBirdsmouth**, **\_kABCClosedBirdsmouth**,
- \_kABCDiagonalSeatCut**, **\_kABCOpenDiagonalSeatCut**, **\_kABCBlindBirdsmouth**,
- \_kABC Housing**, **\_kABC Housing Throughout**, **\_kABC House Rotated**, **\_kABC House Tilted**,
- \_kABC Japanese Hip Cut**, **\_kABC Hip Birdsmouth**, **\_kABC Valley Birdsmouth**,

`_kABC5AxisBirdsmouth, _kABC5AxisBlindBirdsmouth`

The subtypes `_kABC5AxisBirdsmouth` and `_kABC5AxisBlindBirdsmouth` are added in hsbCAD 14.1.39.

The subtypes `_kABC5AxisBirdsMouth` and `_kABC5AxisBlindBirdsMouth` are added in hsbCAD 16.0.52.



```
class AnalysedBeamCut : AnalysedTool { // derived from AnalysedTool
```

```
    AnalysedBeamCut(); // default constructor
```

```
    AnalysedBeamCut(Map map); // constructor with a Map, check validity is recommended
```

```
    Point3d ptOrg() const; // return center point of quader
```

```
    CoordSys coordSys() const; // return the defining quader of BeamCut as a CoordSys
```

```
    Quader quader() const; // return the defining Quader of the BeamCut.
```

```
    Quader quader(double dExtendFreeDirectionsDimension) const; // (added hsbCAD22.1.43  
and hsbCAD23.0.23) return the defining Quader of the BeamCut, but  
extended the free directions by the given value.
```

```
    Point3d[] genBeamQuaderIntersectPoints() const;
```

```
    int blsFreeD(Vector3d vecDir) const; // TRUE if the direction most aligned with vecDir is a  
free direction when the beam solid is considered.
```

```
    double dAngle() const; // value in degrees
```

```
    double dBevel() const; // value in degrees
```

```
    double dTwist() const; // value in degrees
```

```
Vector3d vecSide() const; // relevant face for BeamCut
double dDepth() const; // distance that it cuts in beam

Body cuttingBody() const; // returns the quader as Body

static AnalysedBeamCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedBeamCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType);

static int findClosest(AnalysedBeamCut[] toolListToSearch, Point3d ptRef); // retuns -1 or
index in array

};
```

---

```
static int findClosest(AnalysedBeamCut[] toolListToSearch, Point3d ptRef);
```

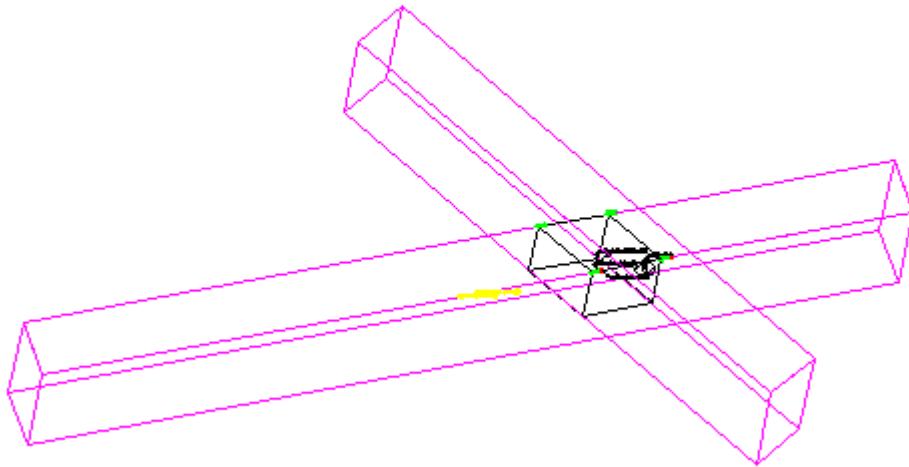
The index is returned of the AnalysedBeamCut which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedBeamCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedBeamCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedBeamCut, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedBeamCut beamcuts[]=
AnalysedBeamCut().filterToolsOfToolType(tools);

int nIndClosest = AnalysedBeamCut().findClosest(beamcuts, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedBeamCut bmCut = beamcuts[nIndClosest];
Point3d ptArVertices[] = bmCut.genBeamQuaderIntersectPoints();
for (int p=0; p<ptArVertices.length(); p++) {
    ptArVertices[p].vis();
}

// retrieve some data from the beamcut itself
Quader qdrBmCut = bmCut.quader();
CoordSys csBmCut = bmCut.coordSys();
csBmCut.vis();

String strLines[0];
strLines.append(bmCut.toolType());
strLines.append(bmCut.toolSubType());

```

```

strLines.append("Depth: "+bmCut.dDepth());
strLines.append("Angle: "+bmCut.dAngle());
strLines.append("Bevel: "+bmCut.dBevel());
strLines.append("Twist: "+bmCut.dTwist());

Body bdBmCut = bmCut.cuttingBody();

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

—end example

[Example E-type that builds a set of DimRequests for an AnalysedBeamCut of type `_kABSSeatCut` of a Beam. If type of Tsl is E-type, the tsl can be added also in the drawing.

```

Unit(1, "mm");
double dEps = U(0.1);

String strActiveToolSubType = _kABCSeatCut;

if (_bOnInsert) {

    _Beam.append(getBeam());
    _Pt0 = getPoint("Select point near tool");
    return;
}

// Compose a _Map from the beam, in case the Tsl is appended to the
// database.
// The contents of the Map should be equal to the _Map generated by
// the shopdraw machine.

if (_ThisInst.handle()!="") { // if handle is not empty, then
instance is database resident
    reportMessage("\nRecompose map from beam entity");
    Beam bm = _Beam0; // should always be valid for an E-type

    // get a relevant tool, and store it inside the _Map->ToolList
    AnalysedTool tools[] = bm.analysedTools(2); // 2 means verbose
reportNotice
    AnalysedBeamCut cuts[]=
AnalysedBeamCut().filterToolsOfToolType(tools, strActiveToolSubType);
    int nIndClosest = AnalysedBeamCut().findClosest(cuts, _Pt0);

    if (nIndClosest>=0) {

```

```

        // found a close tool. Compose _Map as if it was coming from
        the shopdraw machine
        Map mpToolList =
AnalysedTool().convertToMap(cuts[nIndClosest]);
        _Map.setMap(_kAnalysedTools,mpToolList);
    }
    else {
        reportMessage("\nNo tool found. Instance erased.");
        eraseInstance(); // no appropriate tool found
        return;
    }

}

// get the tools from _Map
AnalysedTool tools[] =
AnalysedTool().convertFromSubMap(_Map,_kAnalysedTools,2); // 2 means
verbose reportNotice
AnalysedBeamCut cuts[]=
AnalysedBeamCut().filterToolsOfToolType(tools, strActiveToolSubType);
int nIndClosest = AnalysedBeamCut().findClosest(cuts,_Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedBeamCut bmCut = cuts[nIndClosest];

// the beamcut defines a Quader, and the beam defines a Quader used
in the analysis
Quader qdrBeam = bmCut.genBeamQuader();
Quader qdrBmCut = bmCut.quader();

// for this SeatCut the vecZ, and vecX are important
Vector3d vecZTop = qdrBmCut.vecZ(); // free direction, not running
through
vecZTop.vis(_Pt0);
Vector3d vecXSide = qdrBmCut.vecX(); // free direction, running
through
vecXSide.vis(_Pt0);

String strParentKey = String(bmCut.ptOrg()); // the ptOrg is a unique
key for the tool
reportMessage("\n"+scriptName() +": Tool is accepted:
"+bmCut.toolSubType() + " with key: "+strParentKey);

int nDebugColorCounter = 1;

// add the dimensioning in the beam vecX direction

```

```

{
    Point3d pnts[0];
    Point3d ptTop1 = qdrBmCut.pointAt(0,1,1); // take in vecY, vecZ
dir
    Point3d ptTop2 = qdrBmCut.pointAt(0,-1,1); // take in -vecY, vecZ
dir
    pnts.append(ptTop1);
    pnts.append(ptTop2);

    Vector3d vecView = vecXSide;
    Vector3d vecLineDir = qdrBeam.vecX();
    Vector3d vecPerp= vecZTop;

    for (int p=0; p<pnts.length(); p++) {
        Point3d ptDim = pnts[p];

        // compose a dimrequest
        DimRequestPoint dr(strParentKey, ptDim, vecLineDir, vecPerp);
        dr.setStereotype("BeamCut");
        dr.addAllowedView(vecView, TRUE);
        dr.vis(nDebugColorCounter++); // visualize for debugging
        addDimRequest(dr); // append to shop draw engine collector
    }
}

// add the dimensioning for the depth of the SeatCut in the vecY dir
{
    Point3d ptSide1 = qdrBmCut.pointAt(0,1,1); // take in vecY, vecZ
dir
    Point3d ptSide2 = qdrBmCut.pointAt(0,1,-1); // take in vecY, -vecZ
dir

    Vector3d vecView = vecXSide;
    Vector3d vecDimLinePerpDir = qdrBmCut.vecY();

    // compose a dimrequest
    DimRequestLinear dr(strParentKey, ptSide1, ptSide2,
vecDimLinePerpDir);
    dr.addAllowedView(vecView, TRUE);
    dr.vis(nDebugColorCounter++); // visualize for debugging
    addDimRequest(dr); // append to shop draw engine collector
}

// add the dimensioning for the depth of the SeatCut in the -vecY dir
{
    Point3d ptSide1 = qdrBmCut.pointAt(0,-1,1); // take in -vecY, vecZ
dir
    Point3d ptSide2 = qdrBmCut.pointAt(0,-1,-1); // take in -vecY, -
vecZ dir

    Vector3d vecView = vecXSide;
}

```

```

Vector3d vecDimLinePerpDir = -qdrBmCut.vecY() ;

// compose a dimrequest
DimRequestLinear dr(strParentKey, ptSide1, ptSide2,
vecDimLinePerpDir);
dr.addAllowedView(vecView, TRUE);
dr.vis(nDebugColorCounter++); // visualize for debugging
addDimRequest(dr); // append to shop draw engine collector
}

```

*—end example]*

## 9.5 AnalysedChamfer

The AnalysedChamfer type represents an evaluated tool of type [Chamfer](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedChamfer. However when the casting is not allowed, the resulting AnalysedChamfer will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedChamfer is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAChExposed, _kAChOthersExposed`

The `AnalysedTool::nFeed()` returns one of the following predefined variables (see also [Chamfer](#)) of type **int**:  
`_kChamferRound, _kChamfer45, _kChamfer90`

```

class AnalysedChamfer : AnalysedTool { // derived from AnalysedTool (added since 16.0.18)

    AnalysedChamfer(); // default constructor
    AnalysedChamfer(Map map); // constructor with a Map, check validity is recommended

    Point3d pt1() const;
    Point3d pt2() const;

    double dDepth() const; // measured along the beam face. Since the chamfer cut plane
                          // always has an angle of 45 degrees, the depth can be measured on both
                          // faces of the beam.

    int nFeed() const; // returns _kChamferRound, _kChamfer45 or _kChamfer90

    Vector3d vecSide() const; // a relevant face

```

```

Vector3d[] vecChamferEdges() const; // list of outer normal's to chamfered faces. Each
// face results from a chamfer operation on an edge.

static AnalysedChamfer[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedChamfer[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType);

static int findClosest(AnalysedChamfer[] toolListToSearch, Point3d ptRef); // returns -1 or
index in array.

};

```

---

```
static int findClosest(AnalysedChamfer[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedChamfer which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

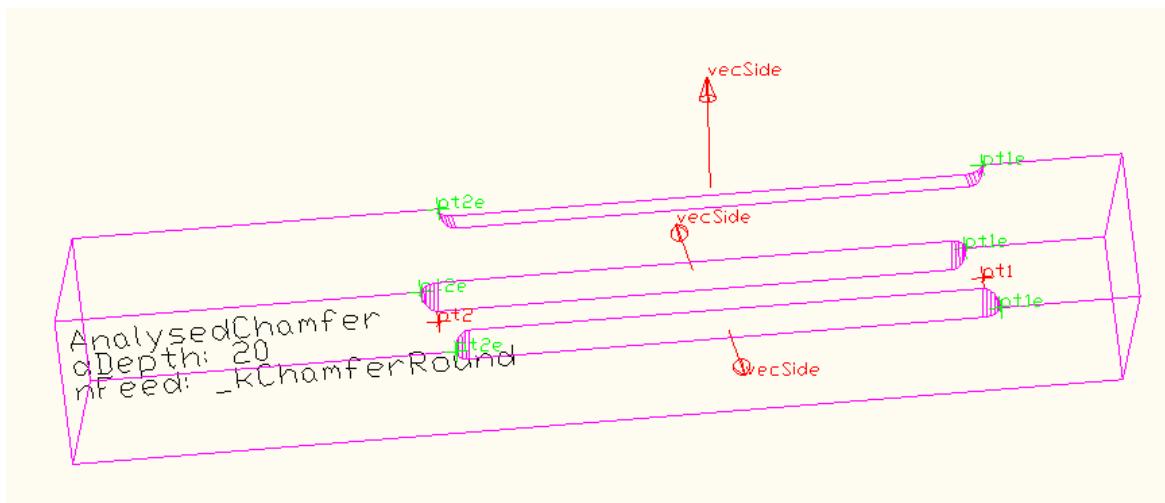
```

static AnalysedChamfer[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedChamfer[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedChamfer, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

```

```

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedChamfer arChamfer[]=
AnalysedChamfer().filterToolsOfToolType(tools);

int nIndClosest = AnalysedChamfer().findClosest(arChamfer,_Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedChamfer anChamfer = arChamfer[nIndClosest];

// retrieve some data from the Chamfer itself
Point3d pt1 = anChamfer.pt1();
Point3d pt2 = anChamfer.pt2();
pt1.vis(1);
pt2.vis(1);

// vecChamferEdges
Vector3d vecSides[] = anChamfer.vecChamferEdges();
for (int v=0; v<vecSides.length(); v++) {
    Vector3d vecSide = vecSides[v];

    // move the points to the beam edge
    Vector3d vecOffset(0,0,0);
    Vector3d vecB = 0.5*bm.vecZ()*bm.dH();
    if (vecB.dotProduct(vecSide)>0)
        vecOffset +=vecB;
    else
        vecOffset -=vecB;

    vecB = 0.5*bm.vecY()*bm.dW();
    if (vecB.dotProduct(vecSide)>0)
        vecOffset +=vecB;
    else
        vecOffset -=vecB;

    Point3d pt1e = pt1+vecOffset;
    Point3d pt2e = pt2+vecOffset;
    pt1e.vis(3);
    pt2e.vis(3);

    vecSide.vis(0.5*(pt1e+pt2e),1);
}

```

```

}

String strLines[0];
strLines.append(anChamfer.toolType());
strLines.append(anChamfer.toolSubType());
strLines.append("dDepth: "+anChamfer.dDepth());

String strFeed = "unknown";
if (anChamfer.nFeed() == _kChamferRound) strFeed = "_kChamferRound";
else if (anChamfer.nFeed() == _kChamfer45) strFeed = "_kChamfer45";
else if (anChamfer.nFeed() == _kChamfer90) strFeed = "_kChamfer90";
strLines.append("nFeed: "+strFeed);

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end example*

## 9.6 AnalysedChamferedLap

The AnalysedChamferedLap type represents an evaluated tool of type [ChamferedLap](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedChamferedLap. However when the casting is not allowed, the resulting AnalysedChamferedLap will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedChamferedLap is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:  
`_kACLPerpendicular, _kACL5Axis`

---

```

class AnalysedChamferedLap : AnalysedTool { // derived from AnalysedTool (added since
14.0.30)

AnalysedChamferedLap(); // default constructor
AnalysedChamferedLap(Map map); // constructor with a Map, check validity is
recommended

Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
tool

```

---

```

CoordSys coordSys() const; // return the CoordSys

double dChamferAngle() const; // value in degrees
double dAngle() const; // value in degrees

double dX() const;
double dY() const;
double dZ() const;

Vector3d vecSide() const; // relevant face for anChamferedLap
Vector3d vecSlope() const; // vector pointing in slope direction, is always in plane that
                           contains vecX and vecZ

static AnalysedChamferedLap[] filterToolsOfType(AnalysedTool[] toolListToFilter);
static AnalysedChamferedLap[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType);

static int findClosest(AnalysedChamferedLap[] toolListToSearch, Point3d ptRef); // returns -1
                           or index in array.
};

```

---

static **int** findClosest(**AnalysedChamferedLap**[] toolListToSearch, **Point3d** ptRef);

The index is returned of the AnalysedChamferedLap which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedChamferedLap[] filterToolsOfType(AnalysedTool[] toolListToFilter)
static AnalysedChamferedLap[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType)

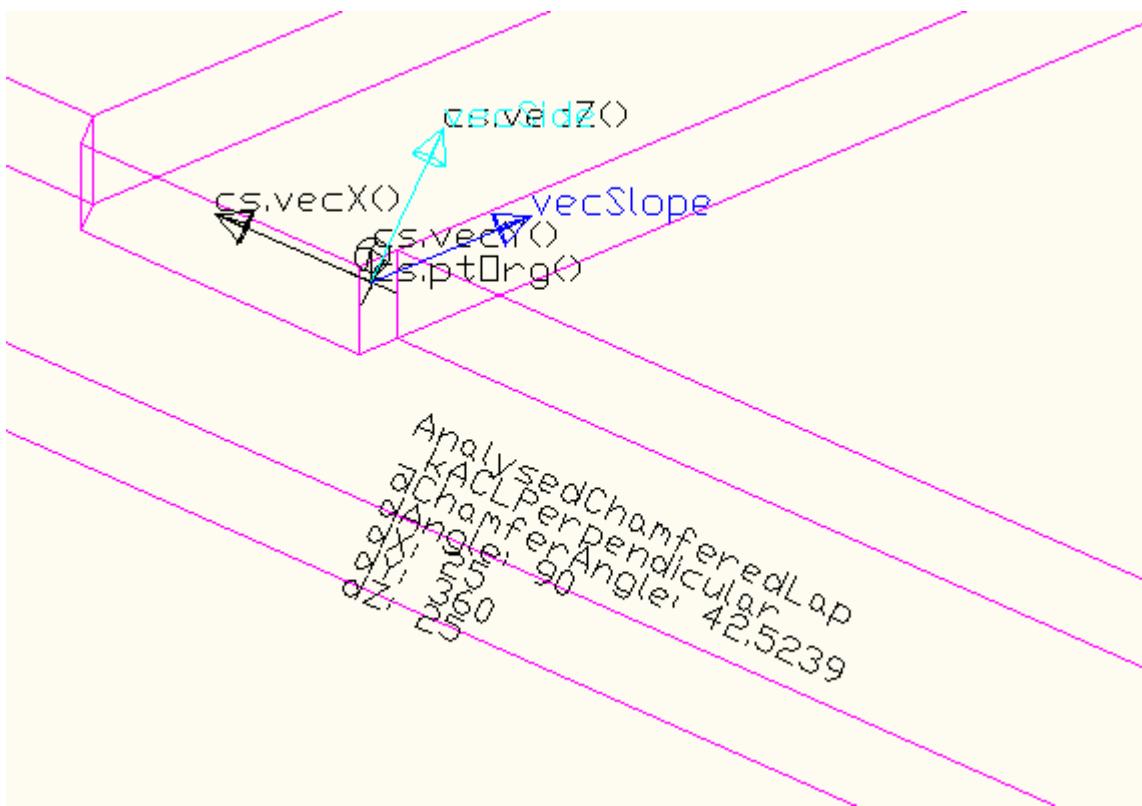
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedChamferedLap, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*

*Image for the tsl attached to the female beam:*



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedChamferedLap anChamferedLaps[] =
AnalysedChamferedLap().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedChamferedLap().findClosest(anChamferedLaps, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedChamferedLap anChamferedLap = anChamferedLaps[nIndClosest];

// retrieve some data from the anChamferedLap itself
CoordSys cs = anChamferedLap.coordSys();
cs.vis();

```

```

Point3d ptOrg = anChamferedLap.ptOrg();
Vector3d vecSide = anChamferedLap.vecSide();
vecSide.vis(ptOrg, 4);
Vector3d vecSlope = anChamferedLap.vecSlope();
vecSlope.vis(ptOrg, 5);

String strLines[0];
strLines.append(anChamferedLap.toolType());
strLines.append(anChamferedLap.toolSubType());
strLines.append("dChamferAngle: "+anChamferedLap.dChamferAngle());
strLines.append("dAngle: "+anChamferedLap.dAngle());
strLines.append("dX: "+anChamferedLap.dX());
strLines.append("dY: "+anChamferedLap.dY());
strLines.append("dZ: "+anChamferedLap.dZ());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1, 1);
}

—end example

```

## 9.7 AnalysedComplexProfile

The AnalysedComplexProfile type represents an evaluated tool of type [ComplexProfile](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedComplexProfile. However when the casting is not allowed, the resulting AnalysedComplexProfile will become invalid. This can be checked with the `isValid()` function of AnalysedTool. Because AnalysedComplexProfile is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kACPPerpendicular, _kACP5Axis,`

The `AnalysedTool::nPreMillType()` returns one of the following predefined variables of type **int**: (see also [ComplexProfile](#))  
`_kCompleted, _kOnlyBrdMth, _kMillWithout, _kWithUF`

---

```

class AnalysedComplexProfile : AnalysedTool { // derived from AnalysedTool (added since
14.0.11)

```

---

```

AnalysedComplexProfile(); // default constructor
AnalysedComplexProfile(Map map); // constructor with a Map, check validity is
    recommended

Point3d ptOrg() const;
CoordSys coordSys() const; // return the CoordSys

double dAngle() const; // the cut is not done if the dAngle is 0
double dStartDepth() const;
double dMaxDepth() const;
double dMinDepth() const;
double dEndDepth() const;
double dLength() const;

int nPreMillType() const; // returns \_kCompleted, \_kOnlyBrdMth, \_kMillWithout, \_kWithUF

PLine plCurve() const; // PLine that starts at ptCut of the tool

Point3d ptCut() const; // point on the cut plane if dAngle is not 0
Vector3d vecCut() const; // normal vector of cut plane
int bHasCut() const; // the cut is not done if the dAngle is 0, in which case FALSE is
    returned by bHasCut

Vector3d vecSide() const; // relevant face for ComplexProfile

static AnalysedComplexProfile[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedComplexProfile[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
    strToolSubType);

static int findClosest(AnalysedComplexProfile[] toolListToSearch, Point3d ptRef); // retuns -1
    or index in array.
};

```

---

**static int findClosest(AnalysedComplexProfile[] toolListToSearch, Point3d ptRef);**

The index is returned of the AnalysedComplexProfile which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

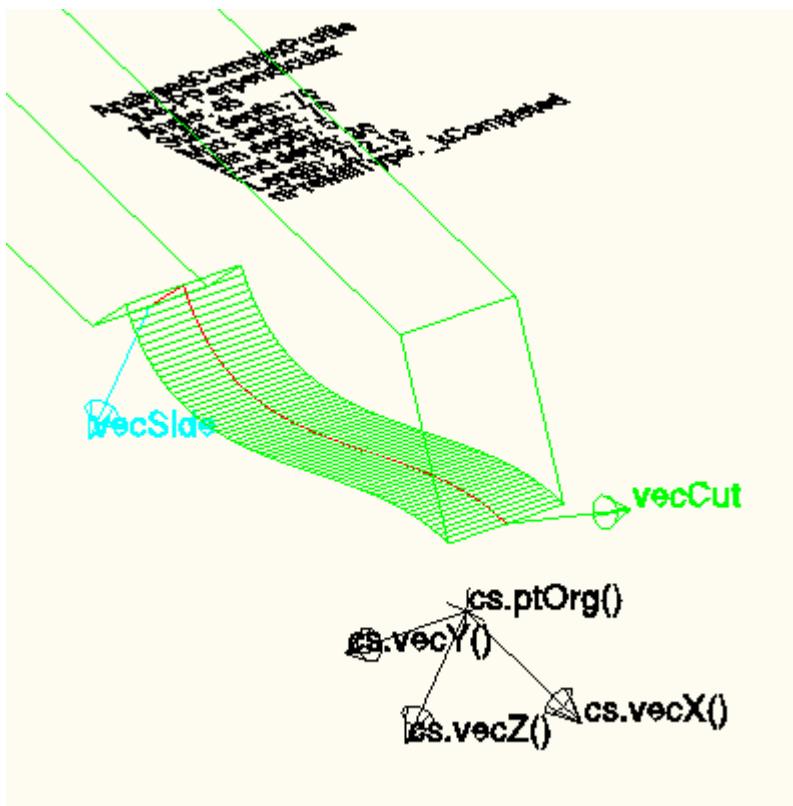
```

static AnalysedComplexProfile[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedComplexProfile[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
    strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedComplexProfile, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedComplexProfile arComplexProfile[]=
AnalysedComplexProfile().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedComplexProfile().findClosest(arComplexProfile, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

```

```

AnalysedComplexProfile anComplexProfile =
arComplexProfile[nIndClosest];

// retrieve some data from the ComplexProfile itself
CoordSys cs = anComplexProfile.coordSys();
cs.vis();

PLine plCurve = anComplexProfile.plCurve();
plCurve.vis(1);

Point3d ptSide = plCurve.ptEnd();
Vector3d vecSide = anComplexProfile.vecSide();
vecSide.vis(ptSide, 4);

Point3d ptCut = anComplexProfile.ptCut();
Vector3d vecCut= anComplexProfile.vecCut();
vecCut.vis(ptCut, 3);

String strLines[0];
strLines.append(anComplexProfile.toolType());
strLines.append(anComplexProfile.toolSubType());
strLines.append("Angle: "+anComplexProfile.dAngle());
strLines.append("Start depth: "+anComplexProfile.dStartDepth());
strLines.append("Max depth: "+anComplexProfile.dMaxDepth());
strLines.append("Min depth: "+anComplexProfile.dMinDepth());
strLines.append("End depth: "+anComplexProfile.dEndDepth());
strLines.append("Length: "+anComplexProfile.dLength());

String strPreMillType = "unknown";
if (anComplexProfile.nPreMillType()==_kCompleted) strPreMillType =
"_kCompleted";
else if (anComplexProfile.nPreMillType()==_kOnlyBrdMth)
strPreMillType = "_kOnlyBrdMth";
else if (anComplexProfile.nPreMillType()==_kMillWithout)
strPreMillType = "_kMillWithout";
else if (anComplexProfile.nPreMillType()==_kWithUF) strPreMillType =
"_kWithUF";
strLines.append("nPreMillType: "+strPreMillType );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

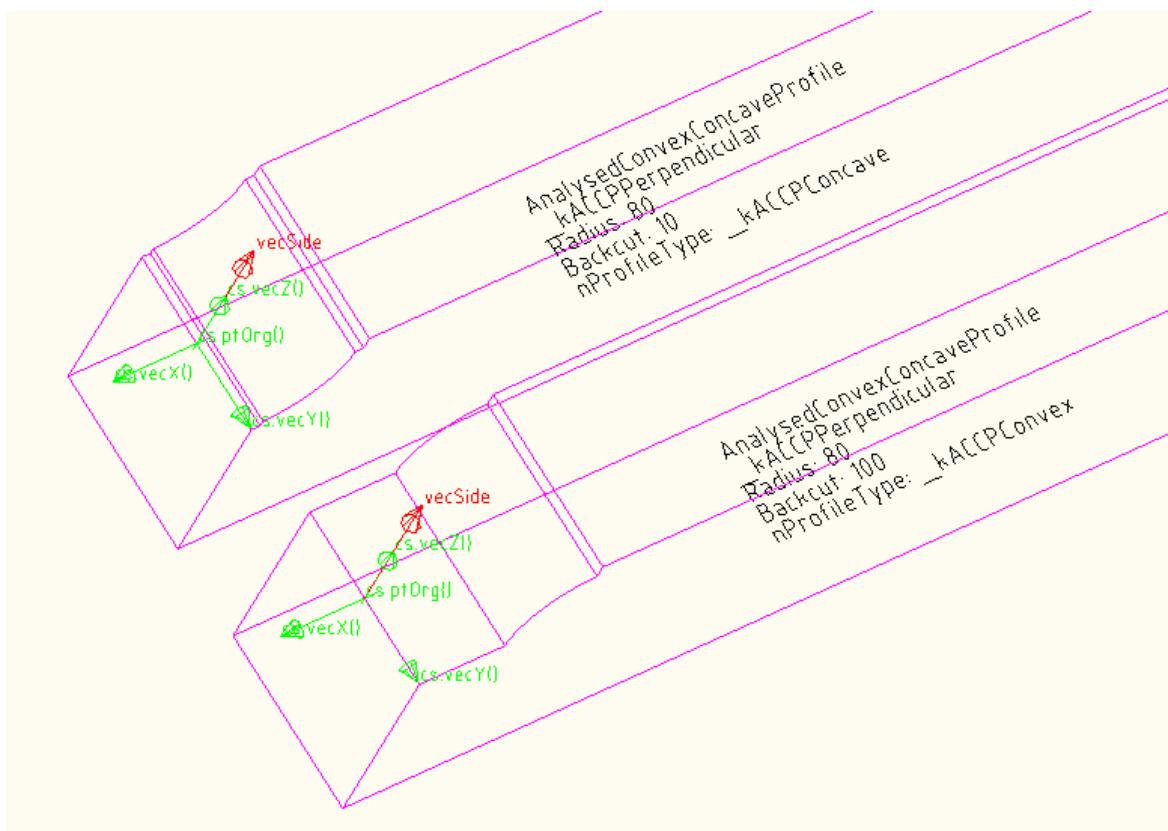
## 9.8 AnalysedConvexConcaveProfile

The AnalysedConvexConcaveProfile type represents an evaluated tool of type [ConvexConcaveProfile](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedConvexConcaveProfile. However when the casting is not allowed, the resulting AnalysedConvexConcaveProfile will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedConvexConcaveProfile is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kACCPPerpendicular, _kACCP5Axis,`

The `AnalysedTool::nProfileType()` returns one of the following predefined variables of type **int**:  
`_kACCPConvex, _kACCPConcave`




---

```
class AnalysedConvexConcaveProfile : AnalysedTool { // derived from AnalysedTool (added since 13.2.118)
```

```
AnalysedConvexConcaveProfile(); // default constructor
```

---

```

AnalysedConvexConcaveProfile(Map map); // constructor with a Map, check validity is
      recommended

Point3d ptOrg() const;
CoordSys coordSys() const; // return the CoordSys

double dRadius() const;
double dBackcut() const;

int nProfileType() const; // returns _kACCPConvex, _kACCPConcave

Vector3d vecSide() const; // relevant face for ConvexConcaveProfile

static AnalysedConvexConcaveProfile[] filterToolsOfToolType(AnalysedTool[]
    toolListToFilter);
static AnalysedConvexConcaveProfile[] filterToolsOfToolType(AnalysedTool[]
    toolListToFilter, String strToolSubType);

static int findClosest(AnalysedConvexConcaveProfile[] toolListToSearch, Point3d ptRef); //
    retuns -1 or index in array.
};

```

---

**static int** findClosest(**AnalysedConvexConcaveProfile**[] toolListToSearch, **Point3d** ptRef);

The index is returned of the AnalysedConvexConcaveProfile which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

**static AnalysedConvexConcaveProfile**[] filterToolsOfToolType(**AnalysedTool**[] toolListToFilter)
**static AnalysedConvexConcaveProfile**[] filterToolsOfToolType(**AnalysedTool**[] toolListToFilter,
 **String** strToolSubType)

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedConvexConcaveProfile, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;

```

---

```

AnalysedTool tools[] = bm.analysedTools(1);
AnalysedConvexConcaveProfile arConvexConcaveProfile[]=
AnalysedConvexConcaveProfile().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedConvexConcaveProfile().findClosest(arConvexConcaveProfile, _Pt
0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedConvexConcaveProfile anConvexConcaveProfile =
arConvexConcaveProfile[nIndClosest];

// retrieve some data from the ConvexConcaveProfile itself
CoordSys csConvexConcaveProfile = anConvexConcaveProfile.coordSys();
csConvexConcaveProfile.vis();

Point3d ptOrg = anConvexConcaveProfile.ptOrg();
Vector3d vecSide = anConvexConcaveProfile.vecSide();
vecSide.vis(ptOrg);

String strLines[0];
strLines.append(anConvexConcaveProfile.toolType());
strLines.append(anConvexConcaveProfile.toolSubType());
strLines.append("Radius: "+anConvexConcaveProfile.dRadius());
strLines.append("Backcut: "+anConvexConcaveProfile.dBackcut());

String strProfileType = "unknown";
if (anConvexConcaveProfile.nProfileType()==_kACCPConvex)
strProfileType = "_kACCPConvex";
else if (anConvexConcaveProfile.nProfileType()==_kACCPConcave)
strProfileType = "_kACCPConcave";
strLines.append("nProfileType: "+strProfileType );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

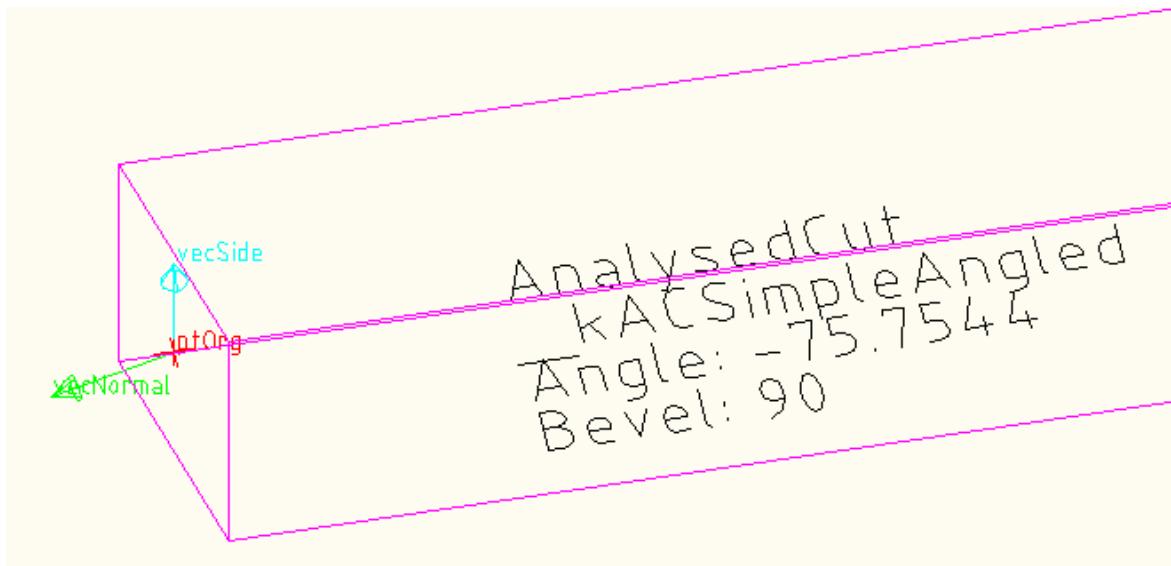
*—end example*

## 9.9 AnalysedCut

The AnalysedCut type represents an evaluated tool of type [Cut](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedCut. However when the casting is not allowed, the resulting AnalysedCut will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedCut is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kACHip`, `_kACPPerpendicular`, `_kACSimpleAngled`, `_kACSimpleBeveled`,  
`_kACCCompound`




---

```
class AnalysedCut : AnalysedTool { // derived from AnalysedTool

    AnalysedCut(); // default constructor
    AnalysedCut(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // point of cut plane
    Vector3d normal() const; // outer normal to cut plane

    double dAngle() const; // value in degrees, refers to the vecSide direction
    double dBevel() const; // value in degrees, refers to the vecSide direction

    Vector3d vecSide() const; // relevant face on beam quader

    Point3d[] bodyPointsInPlane() const; // list of the real body vertices

    int questionIsCompoundCut(Vector3d vecDir, Point3d& ptPosEdge, Point3d&
        ptPosOtherEdge, double& dAngle, double& dBevel) const;
```

---

```

static AnalysedCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                         strToolSubType);

static int findClosest(AnalysedCut[] toolListToSearch, Point3d ptRef); // retuns -1 or index in
                                         array

};


```

---

static int findClosest(AnalysedCut[] toolListToSearch, Point3d ptRef);

The index is returned of the AnalysedCut which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the closest point in the cut plane is calculated. The distance to that closest point is used to determine the closest AnalysedCut.

```

static AnalysedCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)


```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedCut, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

```

int questionIsCompoundCut(Vector3d vecDir, Point3d& ptPosEdge, Point3d& ptPosOtherEdge,
                           double& dAngle, double& dBevel) const;


```

For the given vector vecDir, the cut is analysed, and checked if it is a compound cut. If it is, TRUE is returned, otherwise FALSE is returned.

In the case TRUE is returned, then the other (call by reference) arguments are set appropriately. The ptPosEdge, and the ptPosOtherEdge are the points on the beam edge which is most far out. The line segment defined by these points, defines the dAngle with the beam axis. The dBevel is the angle in the plane perpendicular to the line segment.

The variables dAngle and dBevel are set in degrees.

---

[Example E-type that illustrates map convert to and from:

```

Vector3d vN = _x0+_y0;
vN.normalize();

// compose a map from which an analysed tool can be constructed
Map mpCut;
mpCut.setMapKey("AnalysedCut");
Map mpBase;


```

```

mpBase.setEntity("genBeam", _Beam0);
mpCut.setMap("AnalysedTool", mpBase);
mpCut.setPoint3d("ptOrg", _Pt0);
mpCut.setVector3d("normal", VN);
mpCut.setInt("mustCut", 0);
mpCut.setInt("touchesBody", 1);

AnalysedCut at(mpCut); // construct from map contents
if (!at.bIsValid()) {
    reportMessage("\nAnalysedCut construction from map was not
possible");
}
else { // tool is valid
    reportMessage("\n"+toolType: "+at.toolType());
    reportMessage("\n"+toolSubType: "+at.toolSubType());
    reportMessage("\n"+genBeam: "+at.genBeam());

    // retrieve map from tool
    Map mpContents = AnalysedTool().convertToMap(at);
    reportMessage("\n"+converToMap:
"+mpContents.getDxContent(FALSE));
}

```

*—end example*

[Example E-type that illustrates casting, and locates the closest Analysed cut on the beam:

```

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(2); // 2 means verbose
reportNotice

for (int t=0; t<tools.length(); t++) {
    AnalysedTool tool = tools[t];
    AnalysedCut ct = (AnalysedCut)tool; // explicit cast
    AnalysedTool tool2 = ct; // allowed without explicit casting,
because a AnalysedCut is derived from AnalysedTool
    // ct = tool; // not allowed without explicit casting

    if (ct.bIsValid()) {
        reportMessage("\nTool "+t+" is a cut: "+ct.toolType()+" of
subType: "+ct.toolSubType());
        Point3d ptOrg = ct.ptOrg();
        ptOrg.vis();
    }
    else {
        reportMessage("\nTool "+t+" is not cut: "+tool.toolType());
    }
}

AnalysedCut cuts[] = AnalysedCut().filterToolsOfToolType(tools);
int nIndCutClosest = AnalysedCut().findClosest(cuts, _Pt0);

```

```
if (nIndCutClosest>=0) {  
    Point3d ptOrgClosest = cuts[nIndCutClosest].ptOrg();  
    ptOrgClosest.vis(1);  
}
```

*—end example*

## 9.10 AnalysedDiagonalNotch

The AnalysedDiagonalNotch type represents an evaluated tool of type [DiagonalNotch](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedDiagonalNotch. However when the casting is not allowed, the resulting AnalysedDiagonalNotch will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedDiagonalNotch is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`: `kADNPerpendicular`, `kADN5Axis`

```
class AnalysedDiagonalNotch : AnalysedTool { // derived from AnalysedTool (added since  
13.2.119)  
  
    AnalysedDiagonalNotch(); // default constructor  
    AnalysedDiagonalNotch(Map map); // constructor with a Map, check validity is  
    recommended  
  
    Point3d ptOrg() const; // return center point of tool  
    CoordSys coordSys() const; // return the CoordSys  
  
    double dX() const;  
    double dY() const;  
    double dZ() const;  
  
    Vector3d vecSide() const; // relevant face for DiagonalNotch  
  
    static AnalysedDiagonalNotch[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);  
    static AnalysedDiagonalNotch[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String  
    strToolSubType);  
  
    static int findClosest(AnalysedDiagonalNotch[] toolListToSearch, Point3d ptRef); // returns -  
    or index in array.  
}
```

```
static int findClosest(AnalysedDiagonalNotch[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedDiagonalNotch which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedDiagonalNotch[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedDiagonalNotch[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedDiagonalNotch, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedDiagonalNotch arDiagonalNotch[]=
AnalysedDiagonalNotch().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedDiagonalNotch().findClosest(arDiagonalNotch, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedDiagonalNotch anDiagonalNotch = arDiagonalNotch[nIndClosest];

// retrieve some data from the DiagonalNotch itself
CoordSys csDiagonalNotch = anDiagonalNotch.coordSys();
csDiagonalNotch.vis();

Point3d ptOrg = anDiagonalNotch.ptOrg();
Vector3d vecSide = anDiagonalNotch.vecSide();
vecSide.vis(ptOrg);
```

```

String strLines[0];
strLines.append(anDiagonalNotch.toolType());
strLines.append(anDiagonalNotch.toolSubType());
strLines.append("dX: "+anDiagonalNotch.dX());
strLines.append("dY: "+anDiagonalNotch.dY());
strLines.append("dZ: "+anDiagonalNotch.dZ());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

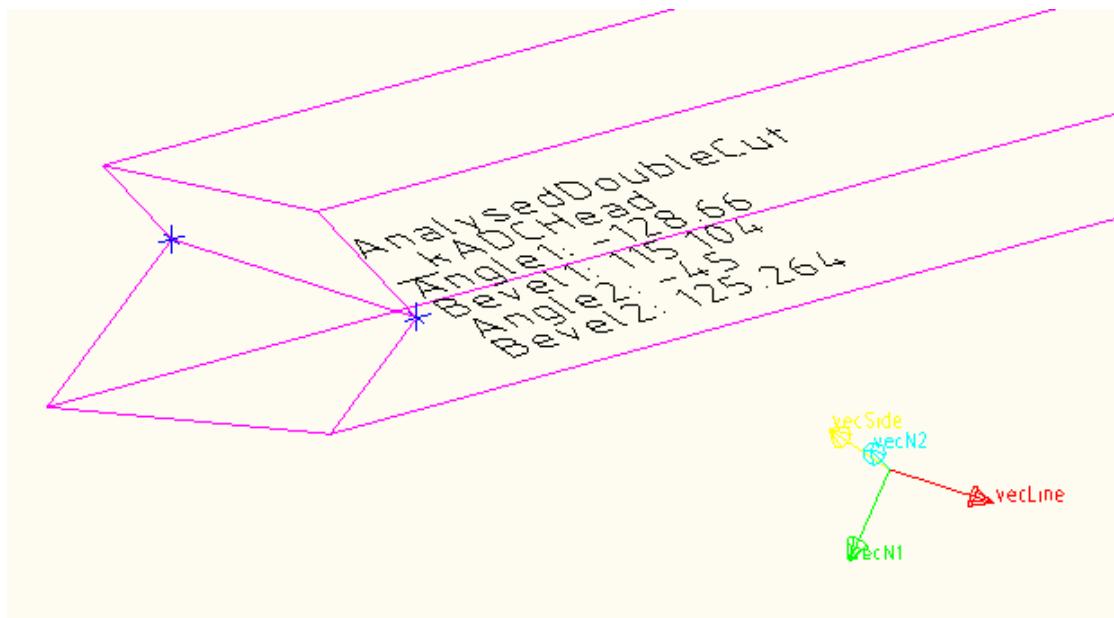
—end example

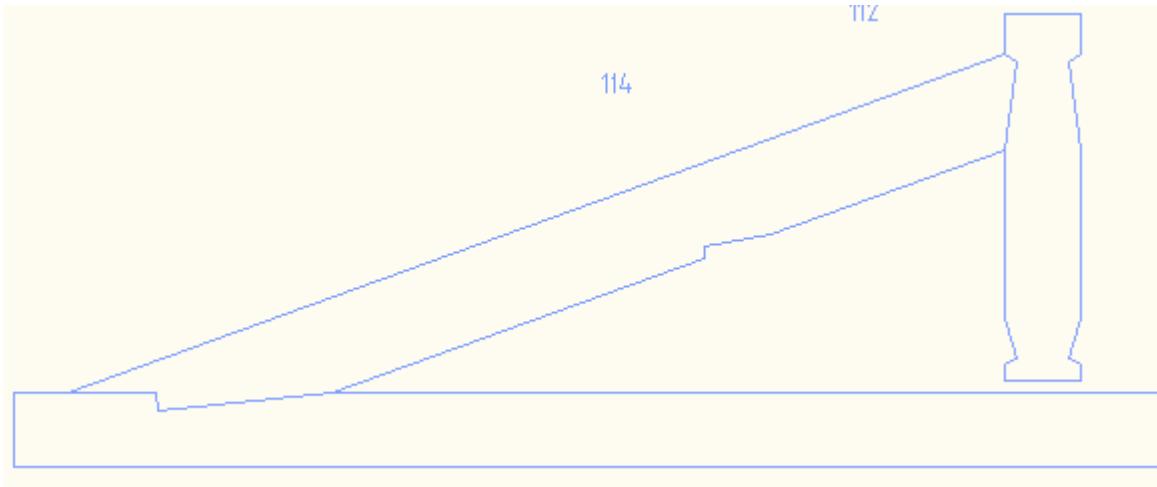
## 9.11 AnalysedDoubleCut

The AnalysedDoubleCut type represents an evaluated tool of type [DoubleCut](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedDoubleCut. However when the casting is not allowed, the resulting AnalysedDoubleCut will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedDoubleCut is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kADCValley`, `_kADCHead`, `_kADCSide`, `_kADCSimpleCut`





```

class AnalysedDoubleCut : AnalysedTool { // derived from AnalysedTool

    AnalysedDoubleCut(); // default constructor
    AnalysedDoubleCut(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return point on valley line closest to center point of beam
    Vector3d vecLine() const; // vector along the valley line

    Vector3d vecSide() const; // relevant face of beam quader for DoubleCut
    Point3d\[\] genBeamQuaderIntersectPoints() const;

    Point3d\[\] bodyPointsAlongLine() const;

    // one of the cuts of the DoubleCut
    Vector3d vecN1() const; // outer normal of the cut
    double dAngle1() const; // value in degrees
    double dBevel1() const; // value in degrees

    // the other cut of the DoubleCut
    Vector3d vecN2() const; // outer normal of the cut
    double dAngle2() const; // value in degrees
    double dBevel2() const; // value in degrees

    static AnalysedDoubleCut\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedDoubleCut\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String strToolSubType);

    static int findClosest(AnalysedDoubleCut\[\] toolListToSearch, Point3d ptRef); // returns -1 or
        // index in array

};

```

---

```
Point3d[] bodyPointsAlongLine() const;
```

From the set of points on the valley line of the double cut that lie on the real body of the beam, the extreme points are collected, and returned by this function. So if the real body of the beam has some points in common with the valley line, the function bodyPointsAlongLine returns some points. At most 2 points are returned. In case there are no solid points on the valley line (eg, when the beam has a drill along the valley line), an empty array will be returned.

```
static int findClosest(AnalysedDoubleCut[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedDoubleCut which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedDoubleCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedDoubleCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedDoubleCut, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedDoubleCut cuts[]=
AnalysedDoubleCut().filterToolsOfToolType(tools);

int nIndClosest = AnalysedDoubleCut().findClosest(cuts, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}
```

---

```

AnalysedDoubleCut dbCut = cuts[nIndClosest];
Point3d ptArVertices[] = dbCut.genBeamQuaderIntersectPoints();
for (int p=0; p<ptArVertices.length(); p++) {
    ptArVertices[p].vis(1); // points in red
}

Point3d ptArLine[] = dbCut.bodyPointsAlongLine();
for (int p=0; p<ptArLine.length(); p++) {
    ptArLine[p].vis(3); // points in green
}

Point3d ptOrg = dbCut.ptOrg();
Vector3d vecN1 = dbCut.vecN1();
Vector3d vecN2 = dbCut.vecN2();
Vector3d vecLine = dbCut.vecLine();
vecN1.vis(ptOrg,4);
vecN2.vis(ptOrg,4);
vecLine.vis(ptOrg,4);

String strLines[0];
strLines.append(dbCut.toolType());
strLines.append(dbCut.toolSubType());
strLines.append("Angle1: "+dbCut.dAngle1());
strLines.append("Bevel1: "+dbCut.dBevel1());
strLines.append("Angle2: "+dbCut.dAngle2());
strLines.append("Bevel2: "+dbCut.dBevel2());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

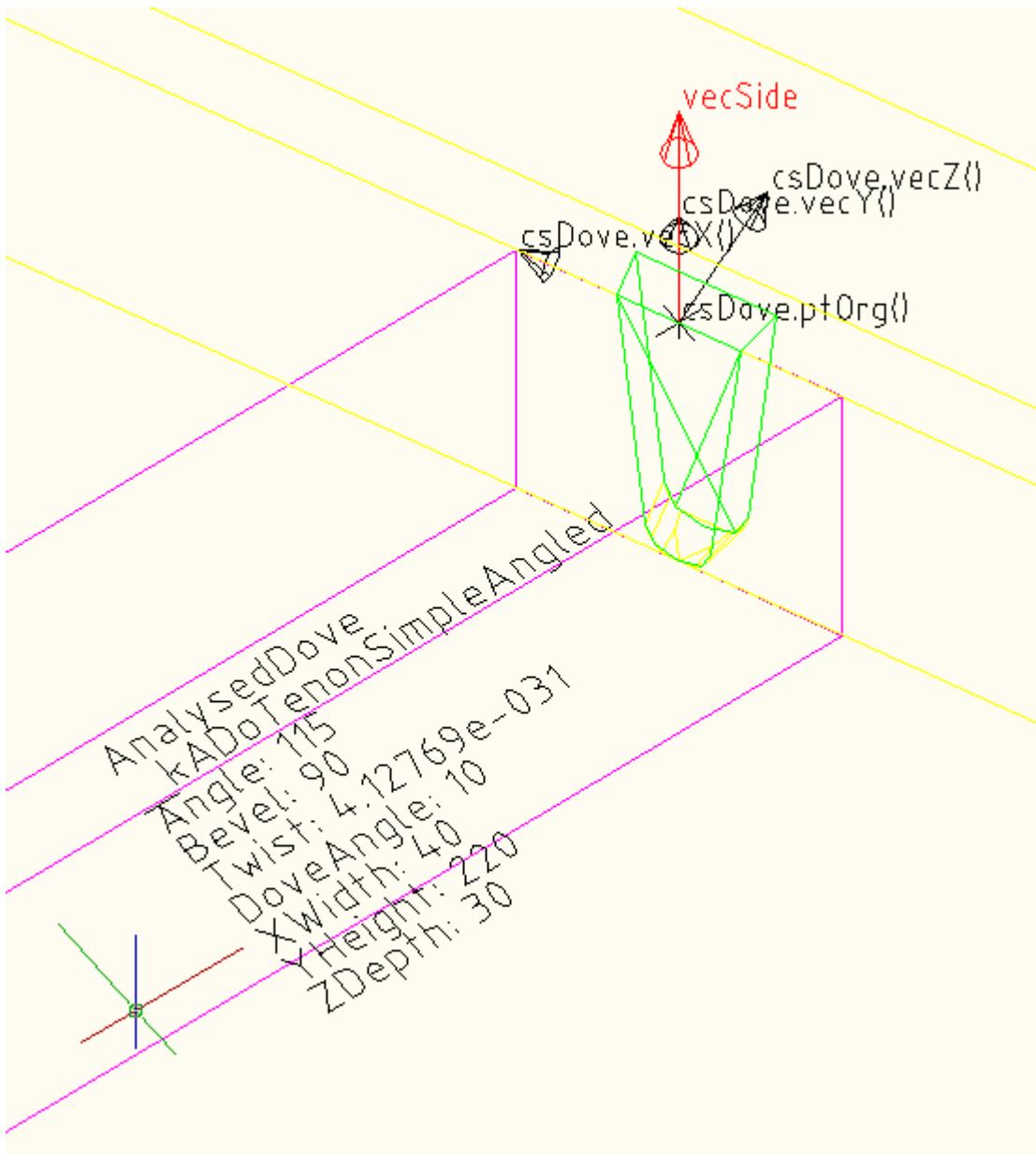
## 9.12 AnalysedDove

The AnalysedDove type represents an evaluated tool of type [Dove](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedDove. However when the casting is not allowed, the resulting AnalysedDove will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedDove is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:

```
_kADoPerpendicular, _kADoRotated, _kADoTilted, _kADo5Axis,
_kADoHeadPerpendicular, _kADoHeadSimpleAngled, _kADoHeadSimpleAngledTwisted,
_kADoHeadSimpleBeveled, _kADoHeadCompound, _kADoTenonPerpendicular,
_kADoTenonSimpleAngled, _kADoTenonSimpleAngledTwisted,
_kADoTenonSimpleBeveled, _kADoTenonCompound
```




---

```
class AnalysedDove : AnalysedTool { // derived from AnalysedTool (added since 13.2.94)
```

```
AnalysedDove\(\); // default constructor
```

```
AnalysedDove\(Map map\); // constructor with a Map, check validity is recommended
```

---

```

Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
tool
CoordSys coordSys() const; // return the CoordSys

double dAngle() const; // value in degrees
double dBevel() const; // value in degrees
double dTwist() const; // value in degrees
double dDoveAngle() const; // value in degrees
double dXWidth() const;
double dYHeight() const;
double dZDepth() const;

Vector3d vecSide() const; // relevant face for Dove

static AnalysedDove[] filterToolsOfType(AnalysedTool[] toolListToFilter);
static AnalysedDove[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
strToolSubType);

static int findClosest(AnalysedDove[] toolListToSearch, Point3d ptRef); // returns -1 or index
in array.
};

```

---

```
static int findClosest(AnalysedDove[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedDove which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedDove[] filterToolsOfType(AnalysedTool[] toolListToFilter)
static AnalysedDove[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedDove, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0..1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);

```

---

```

AnalysedDove doves[ ] = AnalysedDove().filterToolsOfType(tools);

int nIndClosest = AnalysedDove().findClosest(doves, _Pt0);
if (nIndClosest < 0) {
    reportMessage("\n" + scriptName() + ": No tool found. Instance erased.");
    eraseInstance(); // calling eraseInstance will notify that the tool is not consumed.
    return;
}

AnalysedDove dove = doves[nIndClosest];

// retrieve some data from the dove itself
CoordSys csDove = dove.coordSys();
csDove.vis();

Point3d ptOrg = dove.ptOrg();
Vector3d vecSide = dove.vecSide();
vecSide.vis(ptOrg);

String strLines[0];
strLines.append(dove.toolType());
strLines.append(dove.toolSubType());
strLines.append("Angle: " + dove.dAngle());
strLines.append("Bevel: " + dove.dBevel());
strLines.append("Twist: " + dove.dTwist());
strLines.append("DoveAngle: " + dove.dDoveAngle());
strLines.append("XWidth: " + dove.dXWidth());
strLines.append("YHeight: " + dove.dYHeight());
strLines.append("ZDepth: " + dove.dZDepth());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int i=0; i<strLines.length(); i++) {
    Vector3d vecO = -i * 1.2 * pTextHeight * _YU;
    dp.draw(strLines[i], _Pt0 + vecO, _XU, _YU, 1, 1);
}

```

—end example

## 9.13 AnalysedDovesugi

The AnalysedDovesugi type represents an evaluated tool of type Dovesugi. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedDovesugi. However when the casting is not allowed, the resulting AnalysedDovesugi will become invalid. This can be checked with the `isValid()` function of AnalysedTool. Because AnalysedDovesugi is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:  
`_kADSFemalePerpendicular`, `_kADSFemale5Axis`, `_kADSMalePerpendicular`,  
`_kADSMale5Axis`

```

class AnalysedDovesugi : AnalysedTool { // derived from AnalysedTool (added since 14.0.62)

    AnalysedDovesugi(); // default constructor
    AnalysedDovesugi(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const;
    CoordSys coordSys() const; // return the CoordSys

    double dSeatLength() const;
    double dSeatHeight() const;
    double dDoveZDepth() const;
    double dDoveXWidth() const;
    double dDoveYHeight() const;
    double dDoveAngle() const; // in degrees

    Vector3d vecSide() const; // relevant face for Dovesugi

    static AnalysedDovesugi\[\] filterToolsOfType(AnalysedTool\[\] toolListToFilter);
    static AnalysedDovesugi\[\] filterToolsOfType(AnalysedTool\[\] toolListToFilter, String strToolSubType);

    static int findClosest(AnalysedDovesugi\[\] toolListToSearch, Point3d ptRef); // returns -1 or
        index in array.
};

```

---

static int findClosest([AnalysedDovesugi\[\]](#) toolListToSearch, [Point3d](#) ptRef);

The index is returned of the AnalysedDovesugi which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

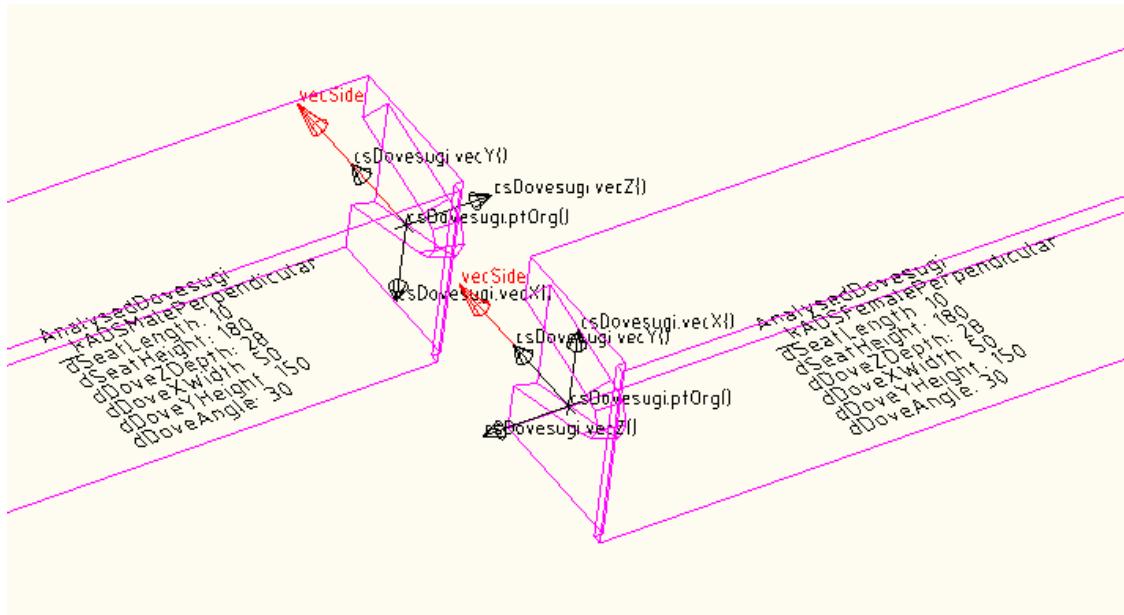
static AnalysedDovesugi\[\] filterToolsOfType(AnalysedTool\[\] toolListToFilter)
static AnalysedDovesugi\[\] filterToolsOfType(AnalysedTool\[\] toolListToFilter, String strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedDovesugi, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*



```

Unit(1,"mm");
double dEps = U(0.1);

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20),"Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedDovesugi arDovesugi[]=
AnalysedDovesugi().filterToolsOfToolType(tools);

int nIndClosest = AnalysedDovesugi().findClosest(arDovesugi,_Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedDovesugi anDovesugi = arDovesugi[nIndClosest];

// retrieve some data from the Dovesugi itself
CoordSys csDovesugi = anDovesugi.coordSys();
csDovesugi.vis();

Point3d ptOrg = anDovesugi.ptOrg();
Vector3d vecSide = 2*anDovesugi.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(anDovesugi.toolType());
strLines.append(anDovesugi.toolSubType());

```

```

strLines.append("dSeatLength: "+anDovesugi.dSeatLength());
strLines.append("dSeatHeight: "+anDovesugi.dSeatHeight());
strLines.append("dDoveZDepth: "+anDovesugi.dDoveZDepth());
strLines.append("dDoveXWidth: "+anDovesugi.dDoveXWidth());
strLines.append("dDoveYHeight: "+anDovesugi.dDoveYHeight());
strLines.append("dDoveAngle: "+anDovesugi.dDoveAngle());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

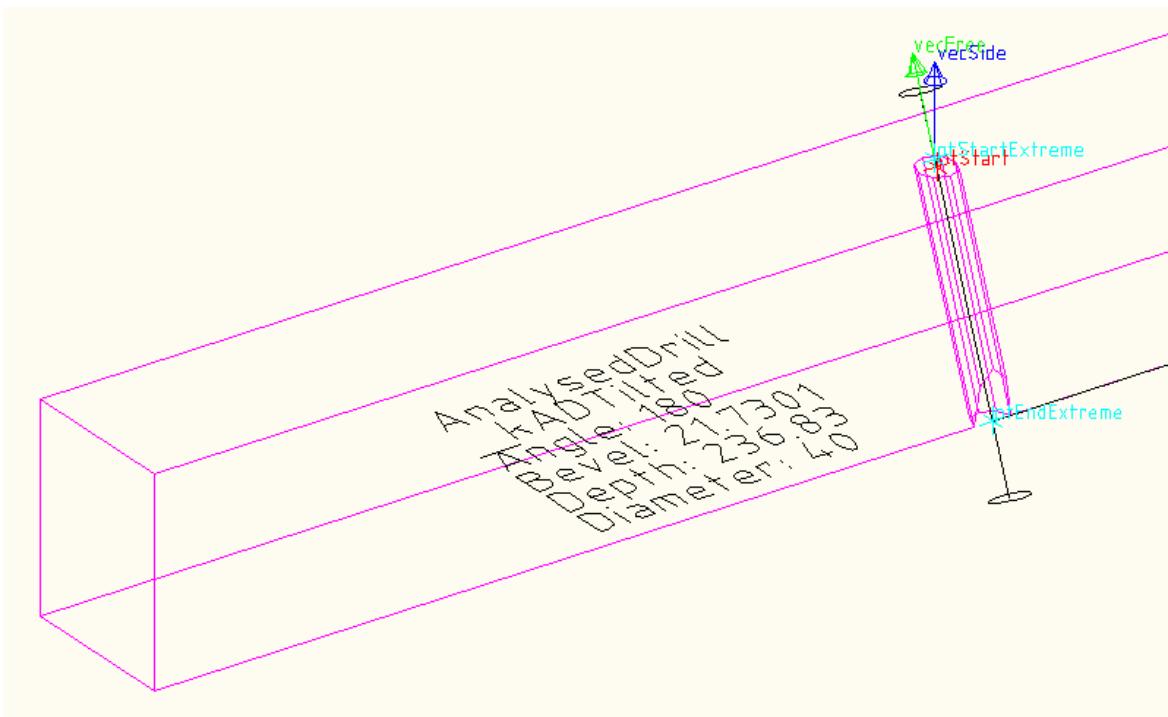
*—end example*

## 9.14 AnalysedDrill

The AnalysedDrill type represents an evaluated tool of type [Drill](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedDrill. However when the casting is not allowed, the resulting AnalysedDrill will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedDrill is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:  
`_kADPerpendicular, _kADRotated, _kADTilted, _kADHead, _kAD5Axis`



```

class AnalysedDrill : AnalysedTool { // derived from AnalysedTool

AnalysedDrill(); // default constructor
AnalysedDrill(Map map); // constructor with a Map, check validity is recommended

Point3d ptStart() const; // return point on beam surface
Point3d ptStartExtreme() const; // return extreme point of cutting body, but along drill
axis
Point3d ptEndExtreme() const; // return extreme point of cutting body, but along drill axis

Vector3d vecFree() const; // vector along the drill axis line pointing to outside of beam
Vector3d vecZ() const; // synonym for vecFree. The other analysed tools have also their
vecZ vector pointing away from the beam surface.
Vector3d vecSide() const; // relevant face of drill

double dDiameter() const;
double dRadius() const;

double dAngle() const; // value in degrees
double dBevel() const; // value in degrees
double dDepth() const; //

int bThrough() const; // returns TRUE if the drill runs completely through the beam
int bUseThisDirection() const; // returns TRUE if it is required to use the vecFree direction

double dInnerCylinderDiameter() const; // from v20.0.88 on

```

```

static AnalysedDrill[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedDrill[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                         strToolSubType);

static int findClosest(AnalysedDrill[] toolListToSearch, Point3d ptRef); // returns -1 or index in
                           array

};

```

---

**static int findClosest(**AnalysedDrill[]** toolListToSearch, **Point3d** ptRef);**

The index is returned of the AnalysedDrill which is considered closest to the given ptRef point.  
To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedDrill[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedDrill[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                         strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedDrill, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedDrill drills[] = AnalysedDrill().filterToolsOfToolType(tools);

int nIndClosest = AnalysedDrill().findClosest(drills, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

```

---

```

AnalysedDrill dbDrill = drills[nIndClosest];
Point3d ptStartExtreme = dbDrill .ptStartExtreme();
Point3d ptStart = dbDrill .ptStart();
Point3d ptEndExtreme = dbDrill .ptEndExtreme();
Vector3d vecFree = dbDrill .vecFree ();
vecFree .vis(ptStartExtreme ,3);
Vector3d vecSide = dbDrill .vecSide();
vecSide .vis(ptStartExtreme ,2);
ptStart .vis(1);
ptEndExtreme .vis(4);
ptStartExtreme .vis(4);

String strLines[0];
strLines.append(dbDrill .toolType());
strLines.append(dbDrill .toolSubType());
strLines.append("Angle: "+dbDrill .dAngle());
strLines.append("Bevel: "+dbDrill .dBevel());
strLines.append("Depth: "+dbDrill .dDepth());
strLines.append("Diameter: "+dbDrill .dDiameter());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO ,_XU, _YU, 1,1);
}

```

*—end example*

## 9.15 AnalysedFreeProfile

The AnalysedFreeProfile type represents an evaluated tool of type [FreeProfile](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedFreeProfile. However when the casting is not allowed, the resulting AnalysedFreeProfile will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedFreeProfile is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAfpPerpendicular, _kAfp5Axis,`

The `AnalysedTool::nCncMode()` returns one of the following predefined variables of type **int**: (see also [FreeProfile](#))  
`_kFingerMill, _kUniversalMill, _kVerticalFingerMill`

The `AnalysedTool::nMillSide()` returns one of the following predefined variables of type **int**:

---

[\\_kAFPLeft](#), [\\_kAPCenter](#), [\\_kAFPRight](#)

---

```
class AnalysedFreeProfile : AnalysedTool { // derived from AnalysedTool (added since
14.0.11)

    AnalysedFreeProfile(); // default constructor
    AnalysedFreeProfile(Map map); // constructor with a Map, check validity is
        recommended

    Point3d ptOrg() const;
    Vector3d vecZ() const;

    double dDepth() const;
    double millDiameter() const; // since V22.0.100

    int nCncMode() const; // returns \_kFingerMill, \_kUniversalMill, \_kVerticalFingerMill
    int nMillSide() const; // returns \_kAFPLeft, \_kAPCenter, \_kAFPRight
    int machinePathOnly() const; // since V22.0.100
    int solidPathOnly() const; // since V22.0.100

    PLine plDefining() const; // PLine that comes from the tool
    PLine\[\] plCuts() const; // array of PLine that express the actual cut paths that cut the beam

    Vector3d vecSide() const; // relevant face for tool

    static AnalysedFreeProfile[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedFreeProfile[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String
        strToolSubType);

    static int findClosest(AnalysedFreeProfile\[\] toolListToSearch, Point3d ptRef); // retuns -1 or
        index in array.
};
```

---

static int findClosest([AnalysedFreeProfile\[\]](#) toolListToSearch, [Point3d](#) ptRef);

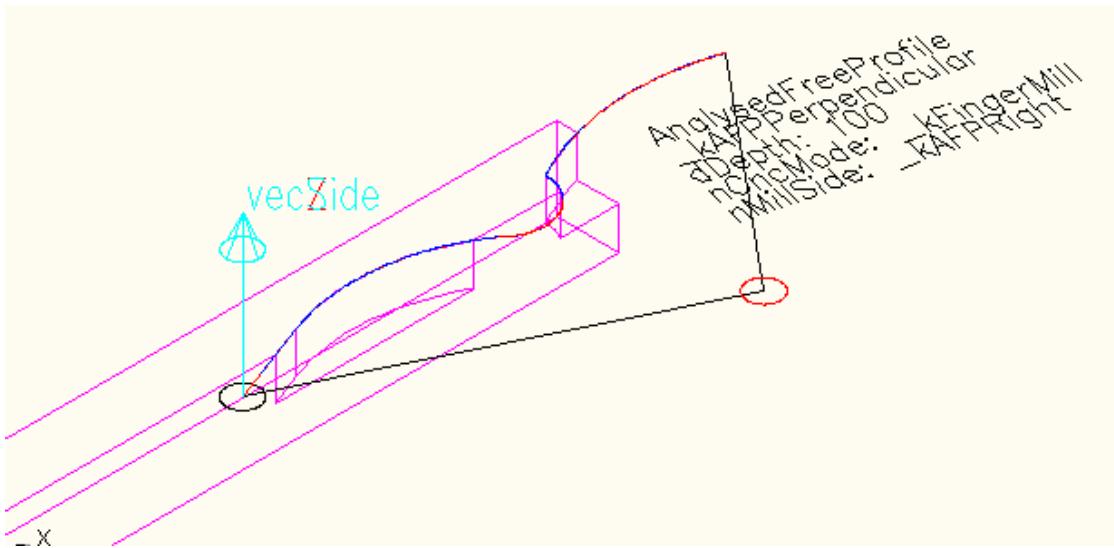
The index is returned of the AnalysedFreeProfile which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedFreeProfile[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter)
static AnalysedFreeProfile[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String
    strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedFreeProfile, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:



```

Unit(1,"mm");
double dEps = U(0.1);

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedFreeProfile arFreeProfile[]=
AnalysedFreeProfile().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedFreeProfile().findClosest(arFreeProfile, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedFreeProfile anFreeProfile = arFreeProfile[nIndClosest];

// retrieve some data from the FreeProfile itself
Point3d ptOrg = anFreeProfile.ptOrg();
Vector3d vecZ = anFreeProfile.vecZ();
vecZ.vis(ptOrg, 1);

PLine plDefining = anFreeProfile.plDefining();
plDefining.vis(1);

PLine plCuts[] = anFreeProfile.plCuts();
for (int c=0; c<plCuts.length(); c++) {

```

```

    plCuts[c].vis(5);
}

vector3d vecSide = anFreeProfile.vecSide();
vecSide.vis(ptOrg, 4);

string strLines[0];
strLines.append(anFreeProfile.toolType());
strLines.append(anFreeProfile.toolSubType());
strLines.append("dDepth: "+anFreeProfile.dDepth());

String strCncMode = "xxx";
if (anFreeProfile.nCncMode() == _kFingerMill) strCncMode =
"_kFingerMill";
else if (anFreeProfile.nCncMode() == _kUniversalMill) strCncMode =
"_kUniversalMill";
else if (anFreeProfile.nCncMode() == _kVerticalFingerMill) strCncMode =
"_kVerticalFingerMill";
strLines.append("nCncMode: "+strCncMode );

String strMillSide = "xxx";
if (anFreeProfile.nMillSide() == _kAFPLeft) strMillSide = "_kAFPLeft";
else if (anFreeProfile.nMillSide() == _kAFPCenter) strMillSide =
"_kAFPCenter";
else if (anFreeProfile.nMillSide() == _kAFPRight) strMillSide =
"_kAFPRight";
strLines.append("nMillSide: "+strMillSide );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

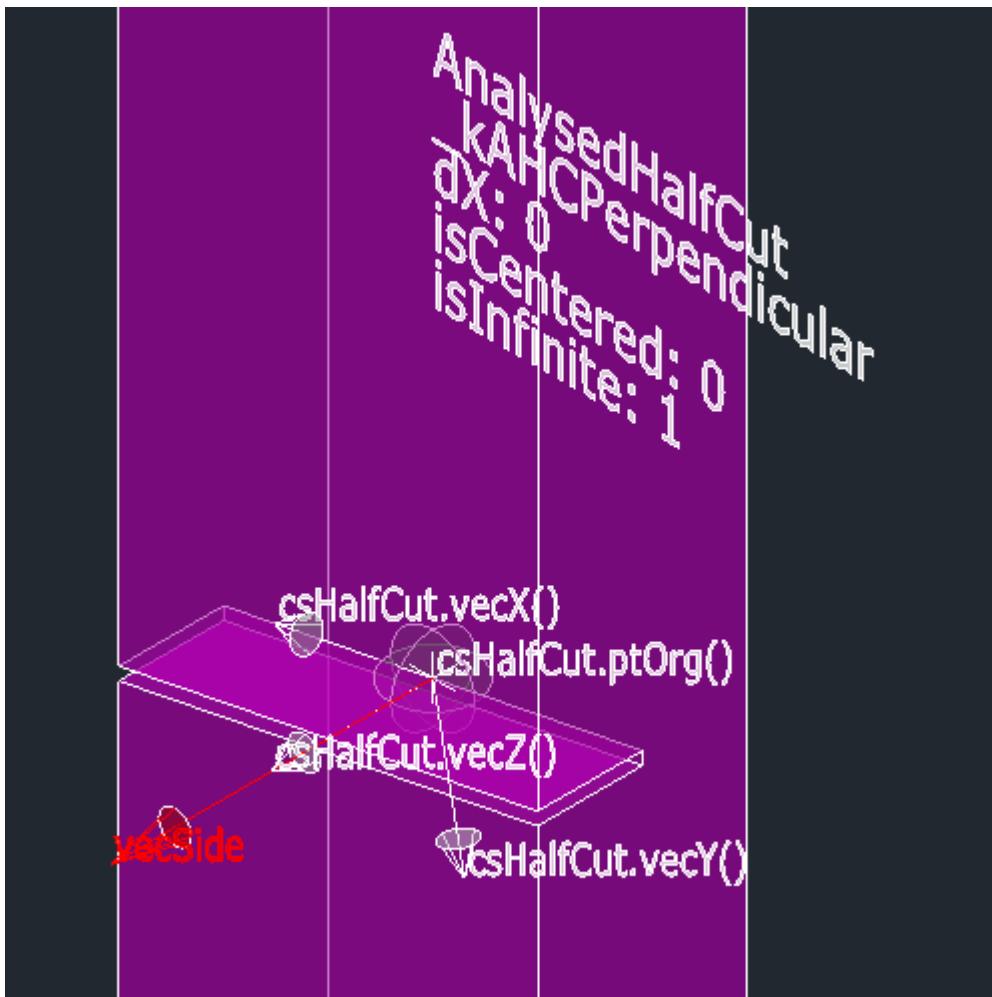
*—end example*

## 9.16 AnalysedHalfCut

The AnalysedHalfCut type represents an evaluated tool of type [HalfCut](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedHalfCut. However when the casting is not allowed, the resulting AnalysedHalfCut will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedHalfCut is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
**\_kAHCPerpendicular**, **\_kAHCRotated**, **\_kAHCTilted**, **\_kAHC5Axis**



```
class AnalysedHalfCut : AnalysedTool { // derived from AnalysedTool (added since 18.2.13)

    AnalysedHalfCut(); // default constructor
    AnalysedHalfCut(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return center point of quader
    CoordSys coordSys() const; // return the CoordSys, vecZ is pointing outwards, vecY is
        // pointing in cut normal direction.

    Vector3d vecSide() const; // relevant face for HalfCut

    int isCentered() const;
    int isInfinite() const;
    double dX() const; // if not infinite, then return length in vecX direction

    static AnalysedHalfCut\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
```

```

static AnalysedHalfCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType);

static int findClosest(AnalysedHalfCut[] toolListToSearch, Point3d ptRef); // retuns -1 or
                           index in array

};

```

---

```
static int findClosest(AnalysedHalfCut[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedHalfCut which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedHalfCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedHalfCut[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedHalfCut, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedHalfCut halfCuts[];
AnalysedHalfCut().filterToolsOfToolType(tools);

int nIndClosest = AnalysedHalfCut().findClosest(halfCuts, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedHalfCut halfCut = halfCuts[nIndClosest];

```

```

// retrieve some data from the halfCut itself
CoordSys csHalfCut = halfCut.coordSys();
csHalfCut.vis();

Vector3d vecSide = 2*halfCut.vecSide();
vecSide.vis(csHalfCut.ptOrg(),1);

String strLines[0];
strLines.append(halfCut.toolType());
strLines.append(halfCut.toolSubType());
strLines.append("dX: "+halfCut.dX());
strLines.append("isCentered: "+halfCut.isCentered());
strLines.append("isInfinite: "+halfCut.isInfinite());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

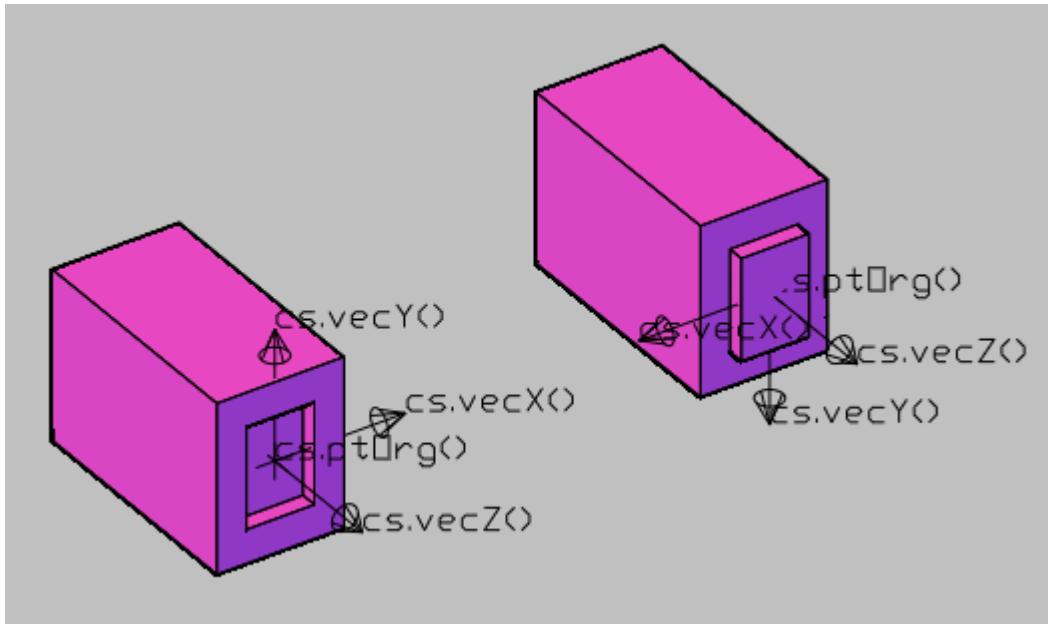
## 9.17 AnalysedHouse

The AnalysedHouse type represents an evaluated tool of type [House](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedHouse. However when the casting is not allowed, the resulting AnalysedHouse will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedHouse is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:

`_kAHSimple, _kAHPerpendicular, _kAHRotated, _kAHTilted, _kAH5Axis,`  
`_kAHHeadPerpendicular, _kAHHeadSimpleAngled, _kAHHeadSimpleAngledTwisted,`  
`_kAHHeadSimpleBeveled, _kAHHeadCompound, _kAHTenonPerpendicular`  
`_kAHTenonSimpleAngled, _kAHTenonSimpleAngledTwisted,`  
`_kAHTenonSimpleBeveled, _kAHTenonCompound`




---

```
class AnalysedHouse : AnalysedTool { // derived from AnalysedTool (added since 13.2.86)
```

```
    AnalysedHouse(); // default constructor
    AnalysedHouse(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return center point of quader
    CoordSys coordSys() const; // return the CoordSys
    Quader quader() const; // return the defining Quader of the House.

    Point3d[] genBeamQuaderIntersectPoints() const;

    double dAngle() const; // value in degrees
    double dBevel() const; // value in degrees
    double dTwist() const; // value in degrees

    Vector3d vecSide() const; // relevant face for House
    double dDepth() const; // distance that it cuts in beam, measured along vecZ
    Point3d ptOnSurface() const; // return point on beam surface, mortise convention

    int nRoundType() const; // simplified round type (added since 14.0.11)
    double dCornerRadius() const; // (added since 24.1.92 and 25.1.76)

    Body cuttingBody() const; // returns the quader as Body

    static AnalysedHouse[] filterToolsOfType(AnalysedTool[] toolListToFilter);
    static AnalysedHouse[] filterToolsOfType(AnalysedTool[] toolListToFilter, String strToolSubType);
```

---

```
static int findClosest(AnalysedHouse[] toolListToSearch, Point3d ptRef); // returns -1 or index  
in array  
};
```

---

```
static int findClosest(AnalysedHouse[] toolListToSearch, Point3d ptRef);
```

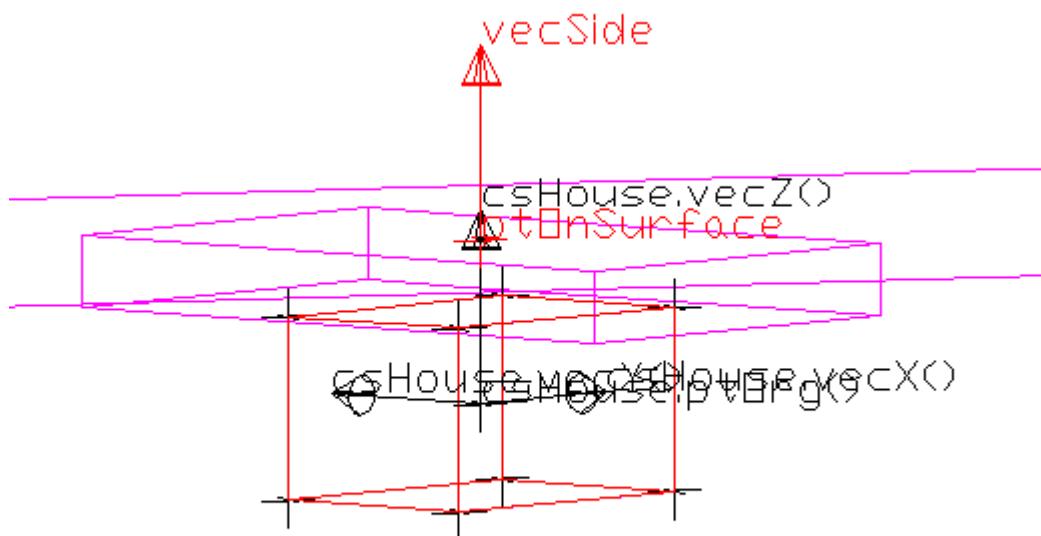
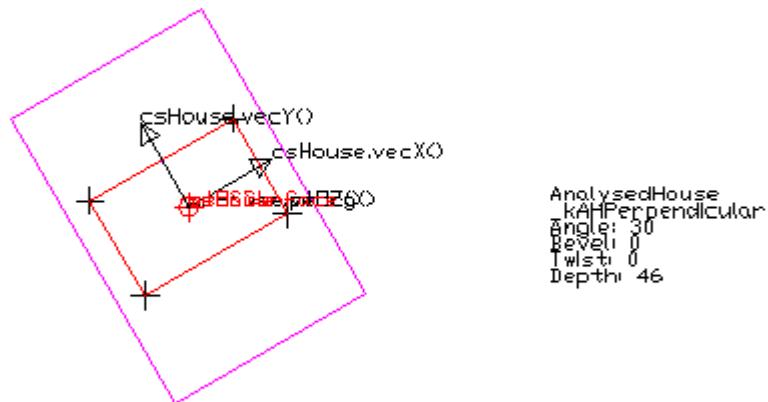
The index is returned of the AnalysedHouse which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)  
static AnalysedHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String  
strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedHouse, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type, with images from a housed House tool:*



```

Unit( 1, " mm" );
double dEps = U( 0..1 );

PropString pDimStyle( 0, _DimStyles, "Dim style" );
PropDouble pTextHeight( 0, U( 20 ), "Text height" );

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools( 1 );
AnalysedHouse houses[] =
AnalysedHouse().filterToolsOfType( tools );

```

```

    int nIndClosest = AnalysedHouse().findClosest(houses, _Pt0);
    if (nIndClosest<0) {
        reportMessage("\n"+scriptName()+" : No tool found. Instance erased.");
        eraseInstance(); // calling eraseInstance will notify that the tool is not consumed.
        return;
    }

    AnalysedHouse house = houses[nIndClosest];
    Point3d ptArVertices[] = house.genBeamQuaderIntersectPoints();
    for (int p=0; p<ptArVertices.length(); p++) {
        ptArVertices[p].vis();
    }

    // retrieve some data from the house itself
    Quader qdrHouse = house.quader();
    qdrHouse.vis(1);
    Coordsys csHouse = house.coordsys();
    // csHouse.vis();

    Point3d ptOnSurface = house.ptOnSurface();
    Vector3d vecSide = house.vecSide();
    vecSide.vis(ptOnSurface);

    String strLines[0];
    strLines.append(house.toolType());
    strLines.append(house.toolSubType());
    strLines.append("Angle: "+house.dAngle());
    strLines.append("Bevel: "+house.dBevel());
    strLines.append("Twist: "+house.dTwist());
    strLines.append("Depth: "+house.dDepth());

    Body bdHouse = house.cuttingBody();

    // display the lines
    Display dp(-1);
    dp.dimStyle(pDimStyle);
    dp.textHeight(pTextHeight);
    for (int l=0; l<strLines.length(); l++) {
        Vector3d vecO = -l*1.2*pTextHeight*_YU;
        dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1, 1);
    }
}

```

*—end example*

## 9.18 AnalysedKamatsugi

The AnalysedKamatsugi type represents an evaluated tool of type Kamatsugi. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedKamatsugi. However when the casting is not allowed, the resulting AnalysedKamatsugi will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedKamatsugi is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:  
`_kAKSFemalePerpendicular`, `_kAKSFemale5Axis`, `_kAKSMalePerpendicular`,  
`_kAKSMale5Axis`

---

```
class AnalysedKamatsugi : AnalysedTool { // derived from AnalysedTool (added since  

14.0.60)

    AnalysedKamatsugi(); // default constructor
    AnalysedKamatsugi(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const;
    CoordSys coordSys() const; // return the CoordSys

    double dSeatLength() const;
    double dSeatHeight() const;
    double dTenonZDepth() const;
    double dTenonStep() const;
    double dTenonXWidth() const;
    double dTenonYHeight() const;

    // following 4 methods added since 24.1.87 and 25.1.52
    double dTopToBackstep() const;
    double dExtraTopNeckWidth() const;
    double dReducedTipWidth() const;
    double dFlatWidth() const;

    Vector3d vecSide() const; // relevant face for Kamatsugi

    static AnalysedKamatsugi[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
    static AnalysedKamatsugi[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String  

        strToolSubType);

    static int findClosest(AnalysedKamatsugi[] toolListToSearch, Point3d ptRef); // retuns -1 or  

        index in array.
};


```

---

```
static int findClosest(AnalysedKamatsugi[] toolListToSearch, Point3d ptRef);
```

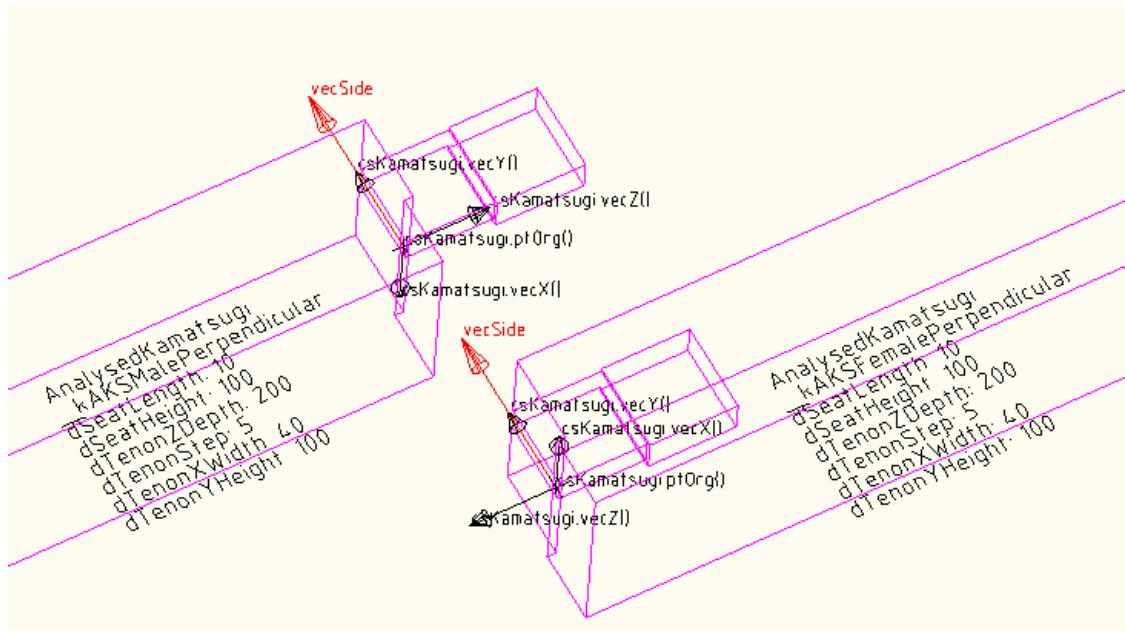
The index is returned of the `AnalysedKamatsugi` which is considered closest to the given `ptRef` point. To calculate the closest to a `ptRef`, the distance to the `ptOrg` is used.

```
static AnalysedKamatsugi[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
```

```
static AnalysedKamatsugi[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedKamatsugi, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:



```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedKamatsugi arKamatsugi[];
AnalysedKamatsugi().filterToolsOfToolType(tools);

int nIndClosest = AnalysedKamatsugi().findClosest(arKamatsugi, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance erased.");
    eraseInstance(); // calling eraseInstance will notify that the tool is not consumed.
    return;
}
```

```

AnalysedKamatsugi anKamatsugi = arKamatsugi[nIndClosest];

// retrieve some data from the Kamatsugi itself
CoordSys csKamatsugi = anKamatsugi.coordSys();
csKamatsugi.vis();

Point3d ptOrg = anKamatsugi.ptOrg();
Vector3d vecSide = 2*anKamatsugi.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(anKamatsugi.toolType());
strLines.append(anKamatsugi.toolSubType());
strLines.append("dSeatLength: "+anKamatsugi.dSeatLength());
strLines.append("dSeatHeight: "+anKamatsugi.dSeatHeight());
strLines.append("dTenonZDepth: "+anKamatsugi.dTenonZDepth());
strLines.append("dTenonStep: "+anKamatsugi.dTenonStep());
strLines.append("dTenonXWidth: "+anKamatsugi.dTenonXWidth());
strLines.append("dTenonYHeight: "+anKamatsugi.dTenonYHeight());
strLines.append("dTopToBackstep: "+anKamatsugi.dTopToBackstep());
strLines.append("dExtraTopNeckWidth:
"+anKamatsugi.dExtraTopNeckWidth());
strLines.append("dReducedTipWidth: "+anKamatsugi.dReducedTipWidth());
strLines.append("dFlatWidth: "+anKamatsugi.dFlatWidth());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU,1,1);
}

```

*—end example*

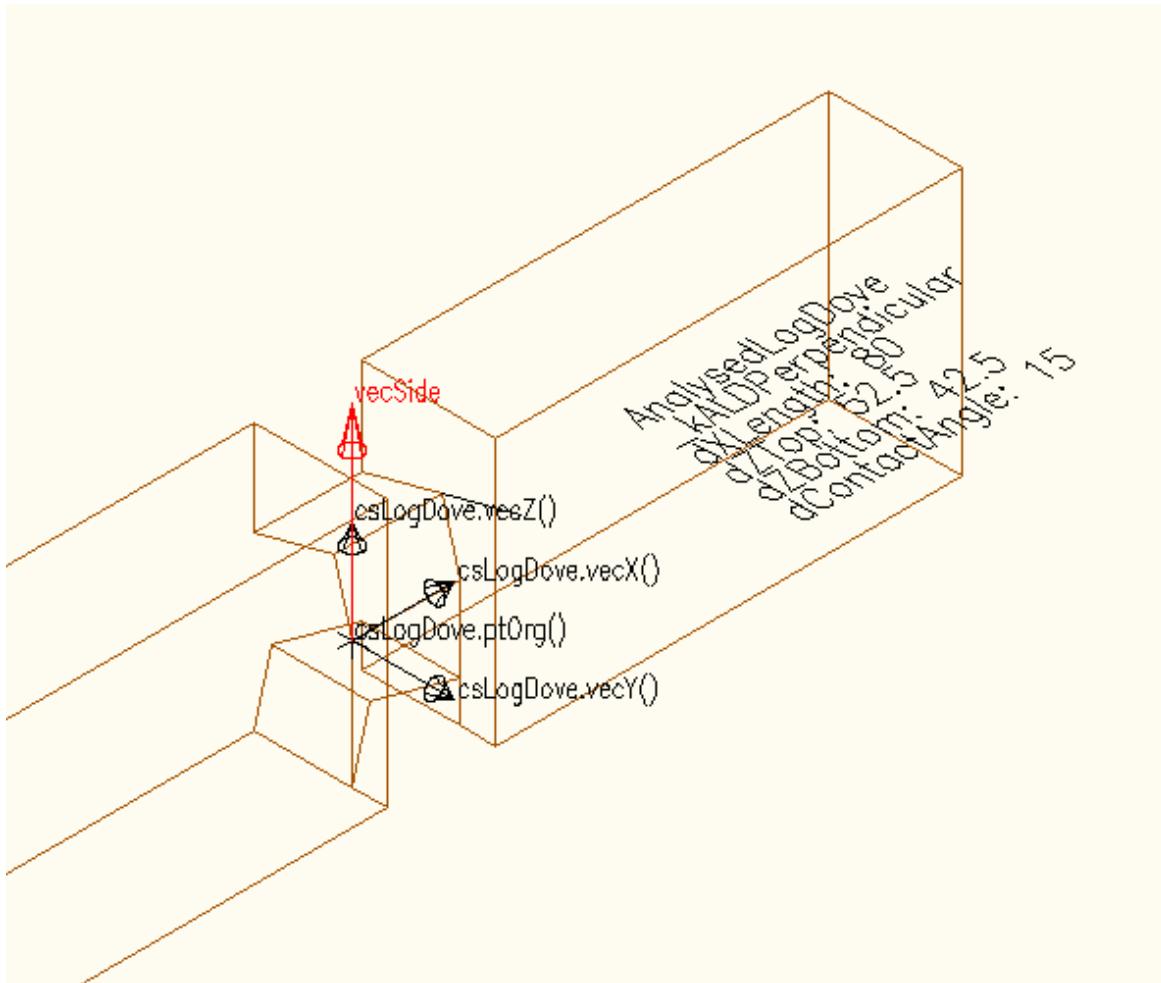
## 9.19 AnalysedLogDove

The AnalysedLogDove type represents an evaluated tool of type LogDove. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedLogDove. However when the casting is not allowed, the resulting AnalysedLogDove will become invalid. This can be checked with the blsValid() function of AnalysedTool. Because AnalysedLogDove is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The **AnalysedTool::toolSubType()** returns one of the following predefined variables of type **String**:  
**\_kALDPerpendicular, \_kALD5Axis**

The convention is that the vecX is pointing to the end of the beam and VecZ is in vecSide direction.



```

class AnalysedLogDove : AnalysedTool { // derived from AnalysedTool (added since 16.0.18)

    AnalysedLogDove(); // default constructor
    AnalysedLogDove(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
                           // tool
    CoordSys coordSys() const; // return the CoordSys

    double dLength() const;
    double dZTop() const;
    double dZBottom() const;
    double dContactAngle() const;
    int nTopBottomProcessing() const;

    Vector3d vecSide() const; // relevant face for LogDove

    static AnalysedLogDove[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
}

```

```
static AnalysedLogDove[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType);

static int findClosest(AnalysedLogDove[] toolListToSearch, Point3d ptRef); // retuns -1 or
                           index in array.
};
```

---

```
static int findClosest(AnalysedLogDove[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedLogDove which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedLogDove[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedLogDove[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                             strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedLogDove, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedLogDove anLogDoves[] =
AnalysedLogDove().filterToolsOfToolType(tools);

int nIndClosest = AnalysedLogDove().findClosest(anLogDoves, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedLogDove anLogDove = anLogDoves[nIndClosest];
```

```

// retrieve some data from the LogDove itself
CoordSys csLogDove = anLogDove.coordSys();
csLogDove.vis();

Point3d ptOrg = anLogDove.ptOrg();
Vector3d vecSide = 2*anLogDove.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(anLogDove.toolType());
strLines.append(anLogDove.toolSubType());
strLines.append("dXLength: "+anLogDove.dXLength());
strLines.append("dZTop: "+anLogDove.dZTop());
strLines.append("dZBottom: "+anLogDove.dZBottom());
strLines.append("dContactAngle: "+anLogDove.dContactAngle());
strLines.append("nTopBottomProcessing:
"+anLogDove.nTopBottomProcessing());

```

```

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

## 9.20 AnalysedLogNotch

The AnalysedLogNotch type represents an evaluated tool of type [LogNotch](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedLogNotch. However when the casting is not allowed, the resulting AnalysedLogNotch will become invalid. This can be checked with the blsValid() function of AnalysedTool. Because AnalysedLogNotch is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The [AnalysedTool::toolSubType\(\)](#) returns one of the following predefined variables of type **String**:  
`_kALNPerpendicular, _kALN5Axis`

The [AnalysedTool::nShoulderType\(\)](#) returns one of the following predefined variables of type **int**:  
`_kBackward, _kCentered, _kForward`

```

class AnalysedLogNotch : AnalysedTool { // derived from AnalysedTool (added since
13.2.119)

    AnalysedLogNotch(); // default constructor
    AnalysedLogNotch(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return center point of tool
    CoordSys coordSys() const; // return the CoordSys

    double dZHeightBeam() const;
    double dYWidthBeam() const;

    double dYDepthHouse() const;
    double dZDepthTop() const;
    double dZDepthBottom() const;
    double dXWidthSide() const;
    double dXWidthInterior() const;
    double dXAxisOffsetOtherBeam() const;

    int nShoulderType() const; // returns \_kBackward, \_kCentered, \_kForward

    Vector3d vecSide() const; // relevant face for LogNotch

    static AnalysedLogNotch[] filterToolsOfType(AnalysedTool[] toolListToFilter);
    static AnalysedLogNotch[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
strToolSubType);

    static int findClosest(AnalysedLogNotch[] toolListToSearch, Point3d ptRef); // retuns -1 or
index in array.
};

```

---

**static int findClosest(AnalysedLogNotch[] toolListToSearch, Point3d ptRef);**

The index is returned of the AnalysedLogNotch which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

    static AnalysedLogNotch[] filterToolsOfType(AnalysedTool[] toolListToFilter)
    static AnalysedLogNotch[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedLogNotch, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedLogNotch arLogNotch[] =
AnalysedLogNotch().filterToolsOfToolType(tools);

int nIndClosest = AnalysedLogNotch().findClosest(arLogNotch, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedLogNotch anLogNotch = arLogNotch[nIndClosest];

// retrieve some data from the LogNotch itself
CoordSys csLogNotch = anLogNotch.coordSys();
csLogNotch.vis();

Point3d ptOrg = anLogNotch.ptOrg();
Vector3d vecSide = anLogNotch.vecSide();
vecSide.vis(ptOrg);

String strLines[0];
strLines.append(anLogNotch.toolType());
strLines.append(anLogNotch.toolSubType());
strLines.append("dYDepthHouse: "+anLogNotch.dYDepthHouse());
strLines.append("dZDepthTop: "+anLogNotch.dZDepthTop());
strLines.append("dZDepthBottom: "+anLogNotch.dZDepthBottom());
strLines.append("dZHeightBeam: "+anLogNotch.dZHeightBeam());
strLines.append("dYWidthBeam: "+anLogNotch.dYWidthBeam());
strLines.append("dXWidthSide: "+anLogNotch.dXWidthSide());
strLines.append("dXWidthInterior: "+anLogNotch.dXWidthInterior());
strLines.append("dXAxisOffsetOtherBeam:
"+anLogNotch.dXAxisOffsetOtherBeam());

String strShoulderType = "unknown";
if (anLogNotch.nShoulderType()==_kBackward) strShoulderType =
"_kBackward";
else if (anLogNotch.nShoulderType()==_kCentered) strShoulderType =
"_kCentered";
else if (anLogNotch.nShoulderType()==_kForward) strShoulderType =
"_kForward";
strLines.append("nShoulderType: "+strShoulderType );

```

```
// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

—end example
```

## 9.21 AnalysedMarker

The AnalysedMarker type represents an evaluated tool of type [Mark](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedMarker. However when the casting is not allowed, the resulting AnalysedMarker will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedMarker is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAMVisible, _kAMShadow`

Values of iYPosText:

<code>_kBottom:</code>	-1
<code>_kCenter:</code>	0 (default value)
<code>_kTop:</code>	1

Values of iXPosText:

<code>_kLeft:</code>	-1
<code>_kCenter:</code>	0 (default value)
<code>_kRight:</code>	1

Values of textOrientation:

<code>_KTOXP, _KTOYP, _KTOXN, _KTOYN</code>
---

---

```
class AnalysedMarker : AnalysedTool { // derived from AnalysedTool (added since 14.0.13)

    AnalysedMarker(); // default constructor
    AnalysedMarker(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return center point of tool
    CoordSys coordSys() const; // return the CoordSys

    double dX() const;
    double dY() const;
    double dTextHeight() const;
```

---

```

String strText() const;

int iXPosText() const;
int iYPosText() const;
int textOrientation() const;
int iLinesToShow() const; // 0 means no lines, 1 means 1 vecY line, 2 mean 2 vecY lines
int bTextLiesDown() const;

Vector3d vecSide() const; // relevant face for tool

static AnalysedMarker[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedMarker[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                         strToolSubType);

static int findClosest(AnalysedMarker[] toolListToSearch, Point3d ptRef); // retuns -1 or
                           index in array.
};

```

---

static **int** findClosest(**AnalysedMarker**[] toolListToSearch, **Point3d** ptRef);

The index is returned of the AnalysedMarker which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

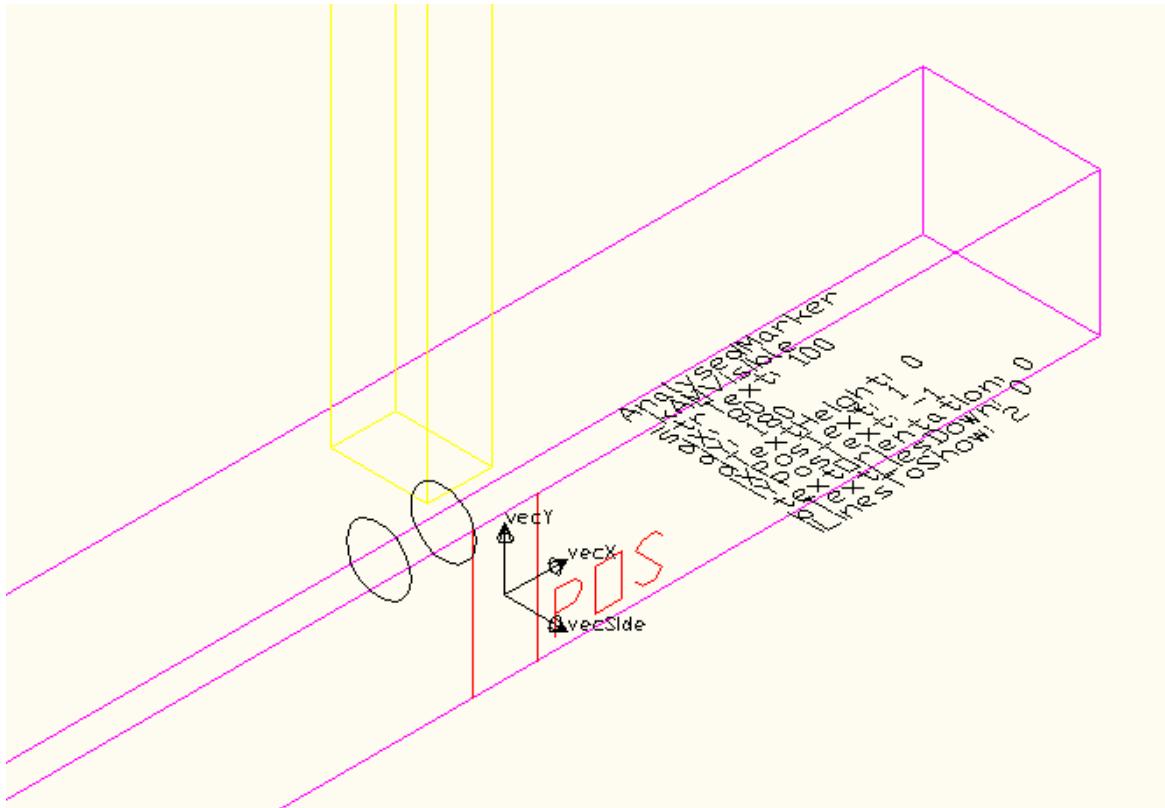
static AnalysedMarker[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedMarker[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                         strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedMarker, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

*[Example E-type:*



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedMarker arMarker[] =
AnalysedMarker().filterToolsOfToolType(tools);

int nIndClosest = AnalysedMarker().findClosest(arMarker, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    //eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedMarker anMarker = arMarker[nIndClosest];

// retrieve some data from the Marker itself
CoordSys csMarker = anMarker.coordSys();
//csMarker.vis();

```

```

Point3d ptOrg = anMarker.ptOrg();

Vector3d vecX = csMarker.vecX();
vecX.vis(ptOrg);
Vector3d vecY = csMarker.vecY();
vecY.vis(ptOrg);

Vector3d vecSide = anMarker.vecSide();
vecSide.vis(ptOrg);

String strLines[0];
strLines.append(anMarker.toolType());
strLines.append(anMarker.toolSubType());
strLines.append("strText: "+anMarker.strText());
strLines.append("dX: "+anMarker.dX());
strLines.append("dY: "+anMarker.dY());
strLines.append("dTTextHeight: "+anMarker.dTextHeight());
strLines.append("iXPosText: "+anMarker.iXPosText());
strLines.append("iYPosText: "+anMarker.iYPosText());
strLines.append("textOrientation: "+anMarker.textOrientation());
strLines.append("bTextLiesDown: "+anMarker.bTextLiesDown());
strLines.append("iLinesToShow: "+anMarker.iLinesToShow());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

```

—end example

## 9.22 AnalysedMarkerLine

The AnalysedMarkerLine type represents an evaluated tool of type [MarkerLine](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedMarkerLine. However when the casting is not allowed, the resulting AnalysedMarkerLine will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedMarkerLine is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAMLVisible`, `_kAMLSHadow`, `_kAMLIInvisible`

```

class AnalysedMarkerLine : AnalysedTool { // derived from AnalysedTool (added since
17.2.4, and 18.1.8)

    AnalysedMarkerLine(); // default constructor
    AnalysedMarkerLine(Map map); // constructor with a Map, check validity is
        recommended

    Point3d pt1() const;
    Point3d pt2() const;

    Vector3d vecSide() const; // a relevant face
    Vector3d vecMarkerSide() const; // the defining normal.

    static AnalysedMarkerLine\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedMarkerLine\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String
        strToolSubType);

    static int findClosest(AnalysedMarkerLine\[\] toolListToSearch, Point3d ptRef); // returns -1 or
        index in array.
}

```

---

[static int findClosest\(\[AnalysedMarkerLine\\[\\]\]\(#\) toolListToSearch, \[Point3d\]\(#\) ptRef\);](#)

The index is returned of the AnalysedMarkerLine which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

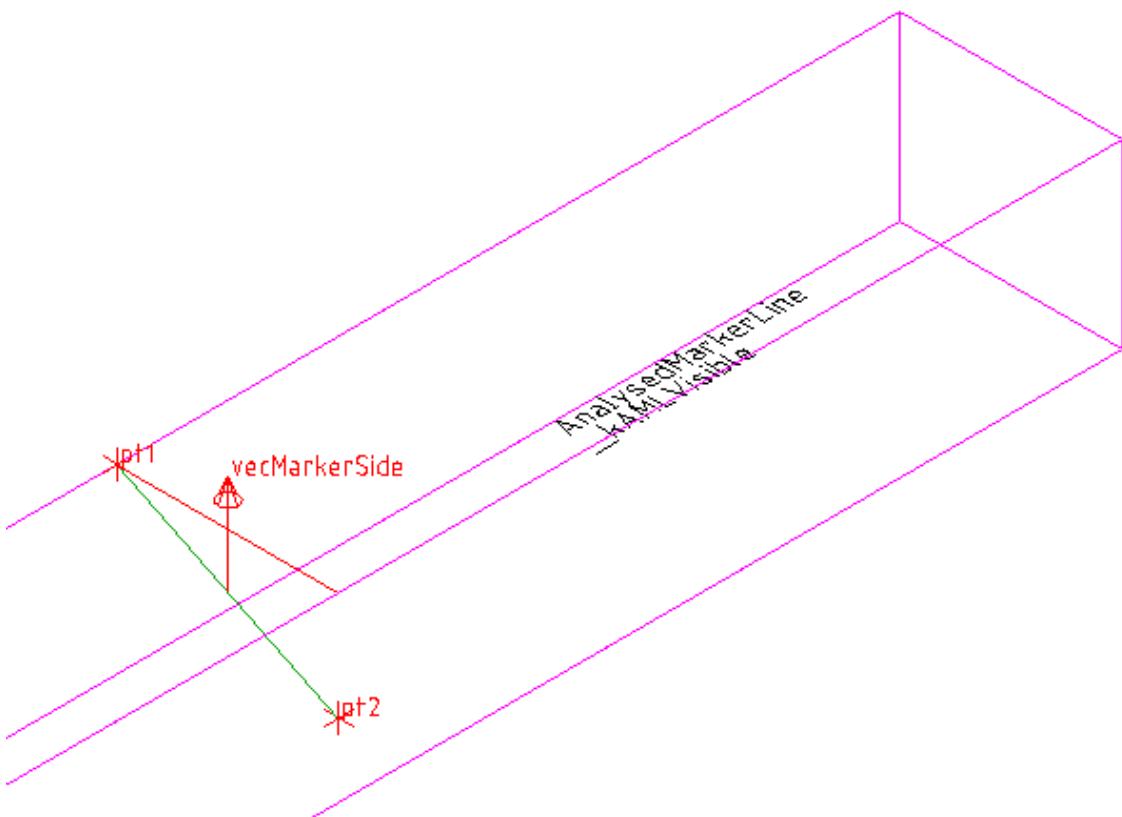
[static \[AnalysedMarkerLine\\[\\]\]\(#\) filterToolsOfToolType\(\[AnalysedTool\\[\\]\]\(#\) toolListToFilter\)](#)

[static \[AnalysedMarkerLine\\[\\]\]\(#\) filterToolsOfToolType\(\[AnalysedTool\\[\\]\]\(#\) toolListToFilter, \[String\]\(#\)
 strToolSubType\)](#)

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedMarkerLine, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedMarkerLine arMarkerLine[]=
AnalysedMarkerLine() .filterToolsOfToolType(tools);

int nIndClosest =
AnalysedMarkerLine() .findClosest(arMarkerLine, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedMarkerLine anMarkerLine = arMarkerLine[nIndClosest];

// retrieve some data from the MarkerLine itself
Point3d pt1 = anMarkerLine.pt1();
Point3d pt2 = anMarkerLine.pt2();
pt1.vis(1);

```

```

pt2.vis(1);

// vecMarkerSide
Vector3d vecMarkerSide = anMarkerLine.vecMarkerSide();
vecMarkerSide.vis(0.5*(pt1+pt2),1);

String strLines[0];
strLines.append(anMarkerLine.toolType());
strLines.append(anMarkerLine.toolSubType());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU,1,1);
}

```

*—end example*

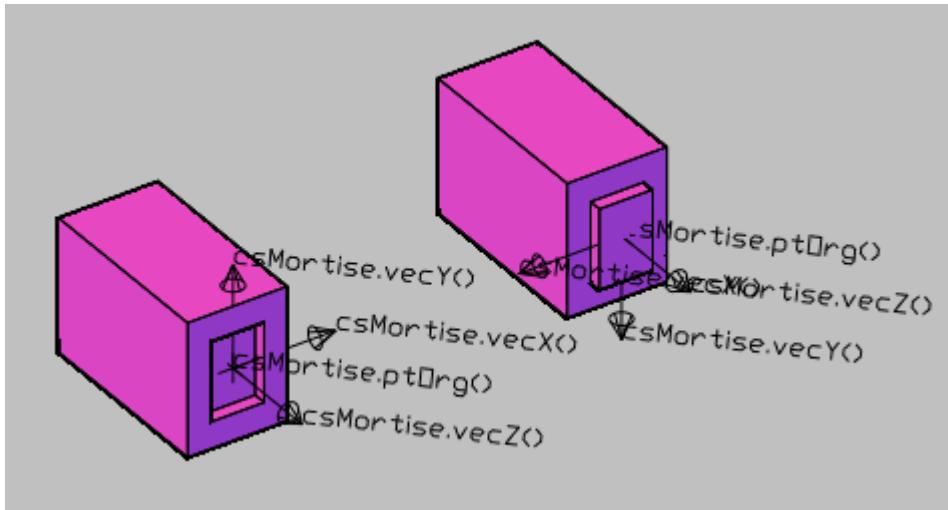
## 9.23 AnalysedMortise

The AnalysedMortise type represents an evaluated tool of type Mortise. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedMortise. However when the casting is not allowed, the resulting AnalysedMortise will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedMortise is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:

`_kAMPerpendicular, _kAMRotated, _kAMTilted, _kAM5Axis, _kAMHeadPerpendicular,`  
`_kAMHeadSimpleAngled, _kAMHeadSimpleAngledTwisted, _kAMHeadSimpleBeveled,`  
`_kAMHeadCompound, _kAMTenonPerpendicular, _kAMTenonSimpleAngled,`  
`_kAMTenonSimpleAngledTwisted, _kAMTenonSimpleBeveled, _kAMTenonCompound`



```

class AnalysedMortise : AnalysedTool { // derived from AnalysedTool (added since 13.2.86)

    AnalysedMortise(); // default constructor
    AnalysedMortise(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const; // return center point of quader
    CoordSys coordSys() const; // return the CoordSys
    Quader quader() const; // return the defining Quader of the Mortise.

    Point3d\[\] genBeamQuaderIntersectPoints() const;

    double dAngle() const; // value in degrees
    double dBevel() const; // value in degrees
    double dTwist() const; // value in degrees

    Vector3d vecSide() const; // relevant face for tool
    double dDepth() const; // distance that it cuts in beam, measured along vecZ
    Point3d ptOnSurface() const; // return point on beam surface, mortise convention

    int nRoundType() const; // see round types (added since 14.0.11)
    double dCornerRadius() const; // (added since 24.1.92 and 25.1.76)

    Body cuttingBody() const; // returns the quader as Body

    static AnalysedMortise\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedMortise\[\] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String strToolSubType);

    static int findClosest(AnalysedMortise\[\] toolListToSearch, Point3d ptRef); // retuns -1 or
        index in array

};

```

```
static int findClosest(AnalysedMortise[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedMortise which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

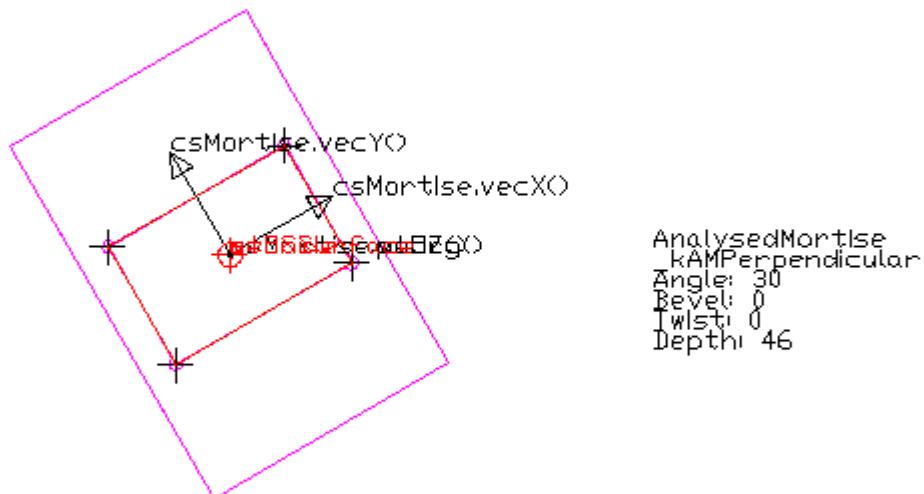
```
static AnalysedMortise[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)  
static AnalysedMortise[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String  
strToolSubType)
```

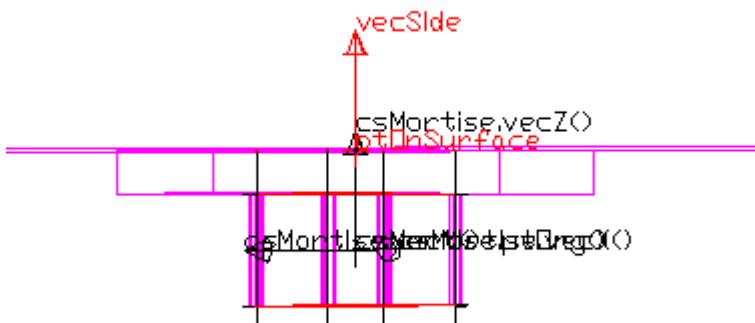
The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedMortise, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type, with images from a housed mortise:

---





```

Unit(1, "mm");
double dEps = U(0, 1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedMortise mortises[] =
AnalysedMortise().filterToolsOfToolType(tools);

int nIndClosest = AnalysedMortise().findClosest(mortises, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName()+" : No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that
the tool is not consumed.
    return;
}

AnalysedMortise mortise = mortises[nIndClosest];
Point3d ptArVertices[] = mortise.genBeamQuaderIntersectPoints();
for (int p=0; p<ptArVertices.length(); p++) {
    ptArVertices[p].vis();
}

// retrieve some data from the mortise itself
Quader qdrMortise = mortise.quader();
qdrMortise.vis(1);
CoordSys csMortise = mortise.coordSys();
// csMortise.vis();

Point3d ptOnSurface = mortise.ptOnSurface();
Vector3d vecSide = mortise.vecSide();
vecSide.vis(ptOnSurface);

String strLines[0];
strLines.append(mortise.toolType());
strLines.append(mortise.toolSubType());
strLines.append("Angle: "+mortise.dAngle());
strLines.append("Bevel: "+mortise.dBevel());
strLines.append("Twist: "+mortise.dTwist());

```

```

strLines.append("Depth: " + mortise.dDepth());
body.bdMortise = mortise.cuttingBody();

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int i=0; i<strLines.length(); i++) {
    Vector3d vecO = -i*1.2*pTextHeight*_YU;
    dp.draw(strLines[i],_Pt0+vecO,_XU,_YU,1,1);
}

—end example

```

## 9.24 AnalysedParHouse

The AnalysedParHouse type represents an evaluated tool of type [ParHouse](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedParHouse. However when the casting is not allowed, the resulting AnalysedParHouse will become invalid. This can be checked with the `isValid()` function of AnalysedTool. Because AnalysedParHouse is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The [AnalysedTool::toolSubType\(\)](#) returns one of the following predefined variables of type [String](#):  
[\\_kAPHPerpendicular](#), [\\_kAPH5Axis](#)

---

**class AnalysedParHouse : AnalysedTool { // derived from [AnalysedTool](#) (added since 14.0.11)**

```

AnalysedParHouse(); // default constructor
AnalysedParHouse(Map map); // constructor with a Map, check validity is recommended

Point3d ptOrg() const; // return center point of tool
CoordSys coordSys() const; // return the CoordSys

double dX() const;
double dY() const;
double dZ() const;

int nRoundType() const; // see round types.

Vector3d vecSide() const; // relevant face for ParHouse

static AnalysedParHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);

```

---

```

static AnalysedParHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
    strToolSubType);

static int findClosest(AnalysedParHouse[] toolListToSearch, Point3d ptRef); // retuns -1 or
    index in array.
};

```

---

```
static int findClosest(AnalysedParHouse[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedParHouse which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

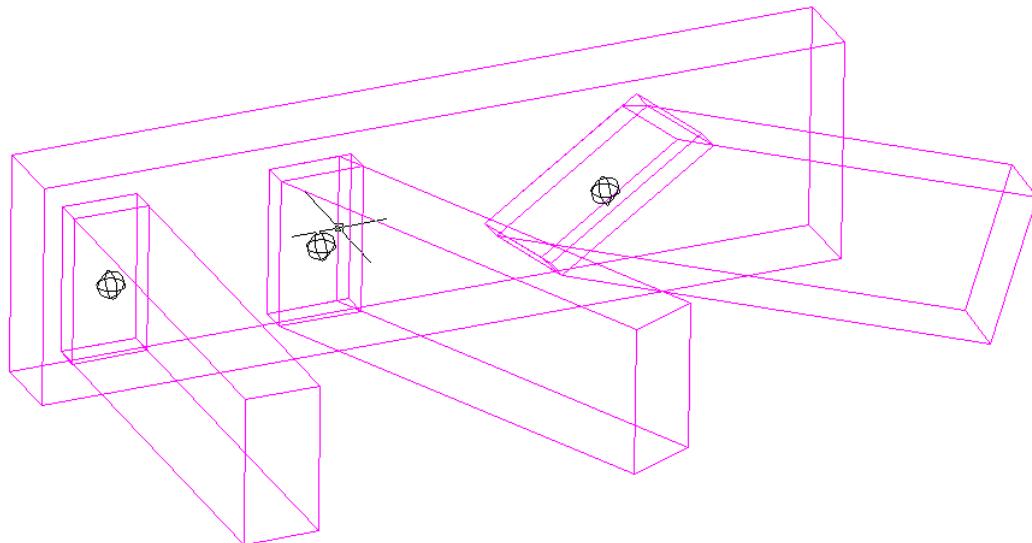
static AnalysedParHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedParHouse[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
    strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedParHouse, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

```

---

```

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedParHouse arParHouse[] =
AnalysedParHouse().filterToolsOfToolType(tools);

int nIndClosest = AnalysedParHouse().findClosest(arParHouse,_Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedParHouse anParHouse = arParHouse[nIndClosest];

// retrieve some data from the ParHouse itself
CoordSys csParHouse = anParHouse.coordSys();
csParHouse.vis();

Point3d ptOrg = anParHouse.ptOrg();
Vector3d vecSide = anParHouse.vecSide();
vecSide.vis(ptOrg);

String strLines[0];
strLines.append(anParHouse.toolType());
strLines.append(anParHouse.toolSubType());
strLines.append("dX: "+anParHouse.dX());
strLines.append("dY: "+anParHouse.dY());
strLines.append("dZ: "+anParHouse.dZ());
strLines.append("nRoundType: "+anParHouse.nRoundType());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

## 9.25 AnalysedPlaning

The AnalysedPlaning type represents an evaluated tool of type Hobel. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedPlaning. However when the casting is not allowed, the resulting AnalysedPlaning will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedPlaning is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAPIExposed, _kAPIOthersExposed`

---

```
class AnalysedPlaning : AnalysedTool { // derived from AnalysedTool (added since 16.0.21)
```

```
    AnalysedPlaning(); // default constructor
    AnalysedPlaning(Map map); // constructor with a Map, check validity is recommended

    Point3d pt1() const;
    Point3d pt2() const;

    Vector3d vecSide() const; // a relevant face
    Vector3d[] vecPlaningSurfaces() const; // list of outer normal's to planing faces.

    static AnalysedPlaning[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
    static AnalysedPlaning[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType);

    static int findClosest(AnalysedPlaning[] toolListToSearch, Point3d ptRef); // returns -1 or
                                // index in array.
};
```

---

```
static int findClosest(AnalysedPlaning[] toolListToSearch, Point3d ptRef);
```

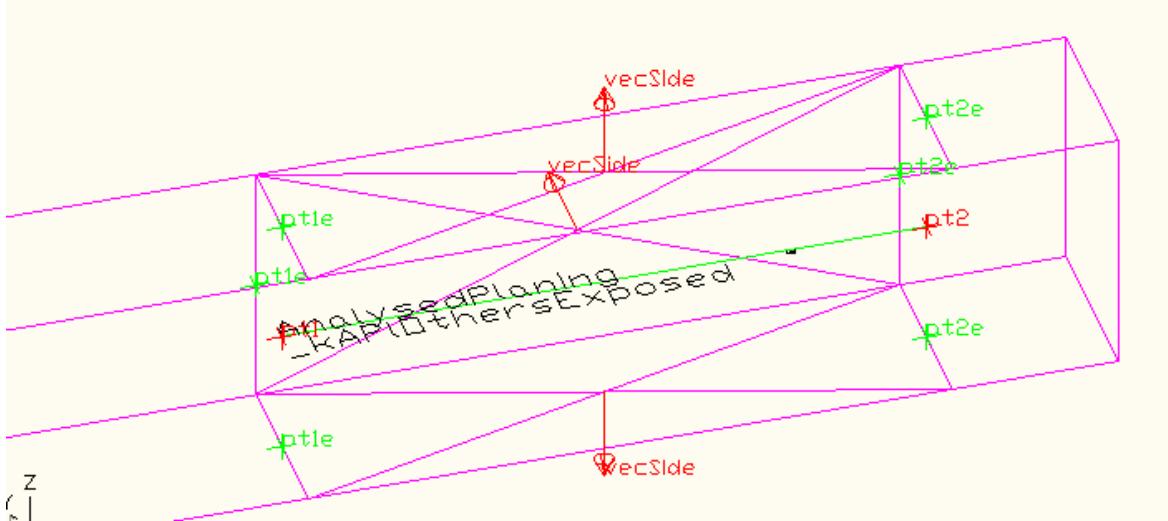
The index is returned of the AnalysedPlaning which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedPlaning[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedPlaning[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)
```

---

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedPlaning, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles , "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedPlaning arPlaning[]=
AnalysedPlaning().filterToolsOfToolType(tools);

int nIndClosest = AnalysedPlaning().findClosest(arPlaning, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedPlaning anPlaning = arPlaning[nIndClosest];

// retrieve some data from the Planing itself
Point3d pt1 = anPlaning.pt1();
Point3d pt2 = anPlaning.pt2();
pt1.vis(1);

```

```

pt2.vis(1);

// vecPlaningSurfaces
Vector3d vecSides[] = anPlaning.vecPlaningSurfaces();
for (int v=0; v<vecSides.length(); v++) {
    Vector3d vecSide = vecSides[v];

    // move the points to the beam surface
    Vector3d vecOffset = 0.5*bm.vecD(vecSide)*bm.dD(vecSide);
    Point3d pt1s = pt1+vecOffset;
    Point3d pt2s = pt2+vecOffset;
    pt1s.vis(3);
    pt2s.vis(3);

    vecSide.vis(0.5*(pt1s+pt2s),1);
}

String strLines[0];
strLines.append(anPlaning.toolType());
strLines.append(anPlaning.toolSubType());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

*—end example*

## 9.26 AnalysedPropellerSurface

The AnalysedPropellerSurface type represents an evaluated tool of type [PropellerSurface](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedPropellerSurface. However when the casting is not allowed, the resulting AnalysedPropellerSurface will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedPropellerSurface is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kAFTPPerpendicular, _kAFTP5Axis,`

The `AnalysedTool::nCncMode()` returns one of the following predefined variables of type **int**: (see also [PropellerSurface](#))  
`_kFingerMill, _kUniversalMill, _kVerticalFingerMill`

The `AnalysedTool::nMillSide()` returns one of the following predefined variables of type **int**:  
`_kAFPLeft, _kAPCenter, _kAFPRight`

```

class AnalysedPropellerSurface : AnalysedTool { // derived from AnalysedTool (added since
14.0.11)

    AnalysedPropellerSurface(); // default constructor
    AnalysedPropellerSurface(Map map); // constructor with a Map, check validity is
        recommended

    Point3d ptOrg() const;
    Vector3d vecZ() const;

    int nCncMode() const; // returns _kFingerMill, _kUniversalMill, _kVerticalFingerMill
    int nMillSide() const; // returns _kAFPLeft, _kAFPCenter, _kAFPRight

    PLine plDefining() const; // PLine that comes from the tool
    PLine plBevel() const; // PLine that is used to connect the plDefining to, to create the surface.

    Vector3d vecSide() const; // relevant face for tool

    static AnalysedPropellerSurface[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
    static AnalysedPropellerSurface[] filterToolsOfToolType(AnalysedTool[] toolListToFilter,
        String strToolSubType);

    static int findClosest(AnalysedPropellerSurface[] toolListToSearch, Point3d ptRef); // returns
        -1 or index in array.
}

```

---

static [int](#) findClosest([AnalysedPropellerSurface](#)[] toolListToSearch, [Point3d](#) ptRef);

The index is returned of the AnalysedPropellerSurface which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

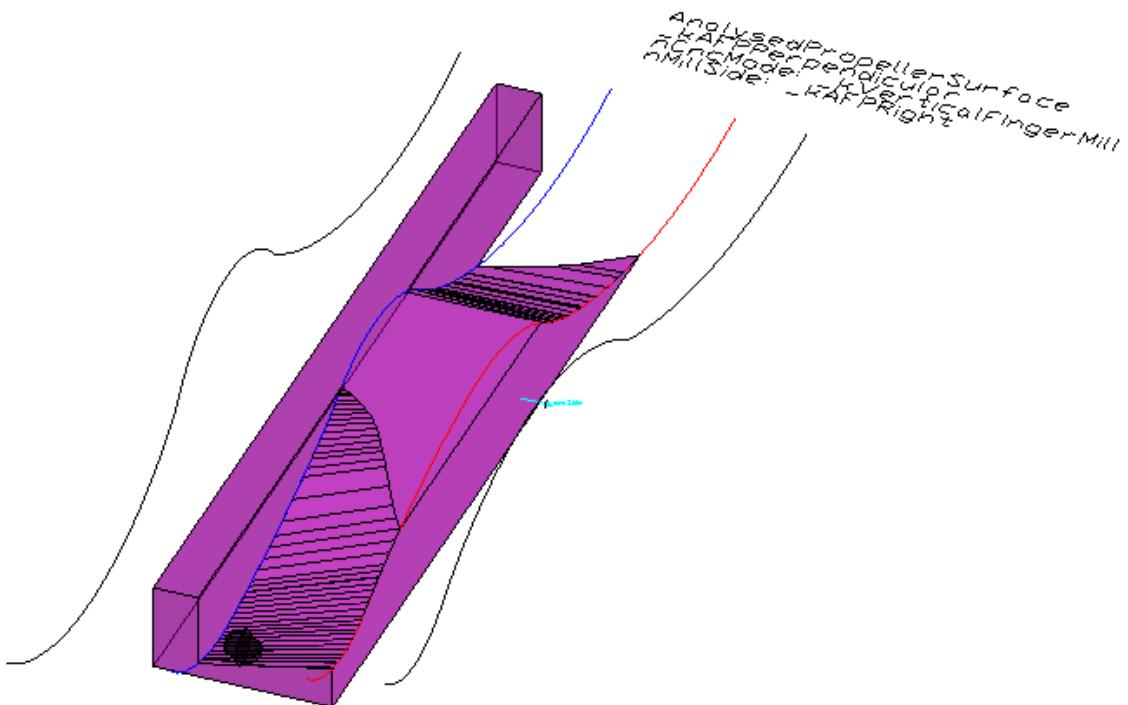
static AnalysedPropellerSurface[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedPropellerSurface[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
    strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedPropellerSurface, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0,_DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedPropellerSurface arPropellerSurface[] =
AnalysedPropellerSurface().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedPropellerSurface().findClosest(arPropellerSurface, _Pt0);
if (nIndClosest<0) {
    reportMessage ("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedPropellerSurface anPropellerSurface =
arPropellerSurface[nIndClosest];

// retrieve some data from the PropellerSurface itself
Point3d ptOrg = anPropellerSurface.ptOrg();
Vector3d vecZ = anPropellerSurface.vecZ();
vecZ.vis(ptOrg, 1);

```

```

PLine plDefining = anPropellerSurface.plDefining();
plDefining.vis(1);
PLine plBevel = anPropellerSurface.plBevel();
plBevel.vis(5);

Vector3d vecSide = anPropellerSurface.vecSide();
vecSide.vis(ptOrg, 4);

String strLines[0];
strLines.append(anPropellerSurface.toolType());
strLines.append(anPropellerSurface.toolSubType());

String strCncMode = "xxx";
if (anPropellerSurface.nCncMode() == _kFingerMill) strCncMode =
"_kFingerMill";
else if (anPropellerSurface.nCncMode() == _kUniversalMill) strCncMode =
"_kUniversalMill";
else if (anPropellerSurface.nCncMode() == _kVerticalFingerMill)
strCncMode = "_kVerticalFingerMill";
strLines.append("nCncMode: "+strCncMode );

String strMillSide = "xxx";
if (anPropellerSurface.nMillSide() == _kAFPLeft) strMillSide =
"_kAFPLeft";
else if (anPropellerSurface.nMillSide() == _kAFPCenter) strMillSide =
"_kAFPCenter";
else if (anPropellerSurface.nMillSide() == _kAFPRight) strMillSide =
"_kAFPRight";
strLines.append("nMillSide: "+strMillSide );

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

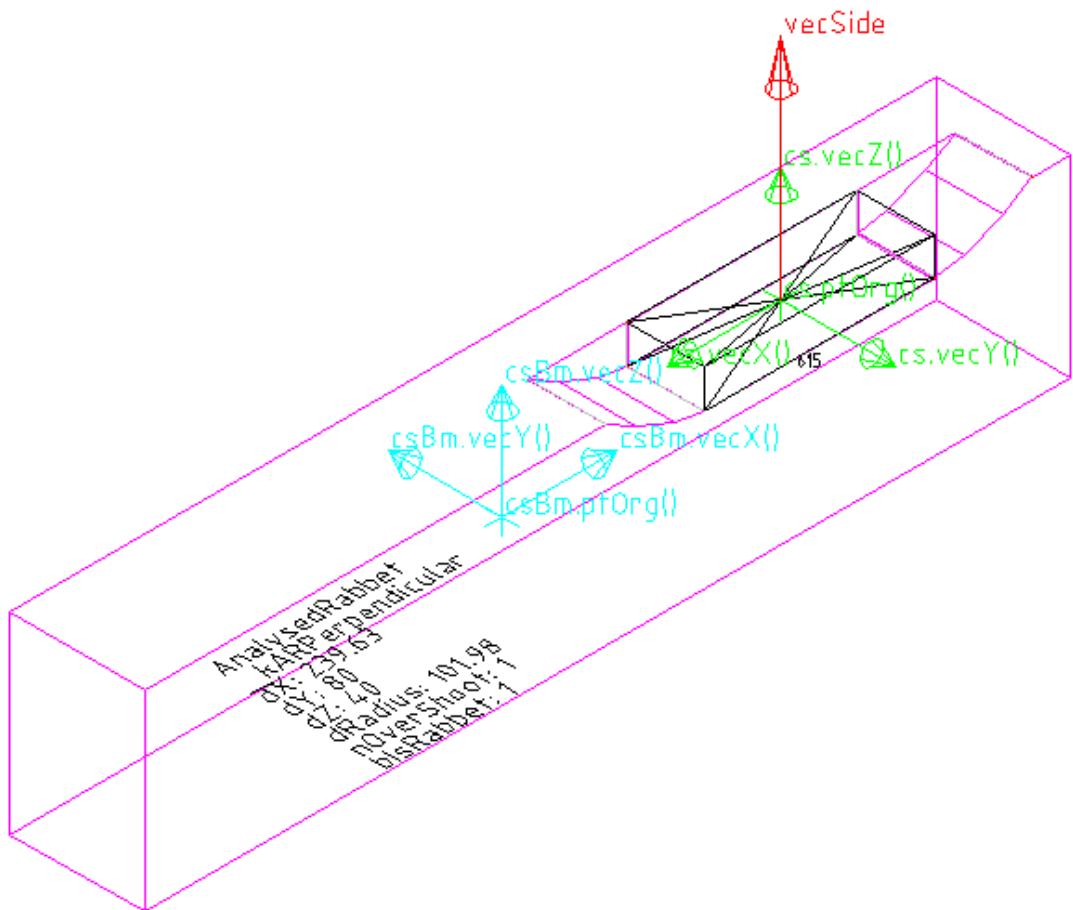
*—end example*

## 9.27 AnalysedRabbit

The AnalysedRabbit type represents an evaluated tool of type [Rabbit](#) or [Daddo](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedRabbit. However when the casting is not allowed, the resulting AnalysedRabbit will become invalid. This can be checked with the blsValid() function of AnalysedTool. Because AnalysedRabbit is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The [AnalysedTool::toolSubType\(\)](#) returns one of the following predefined variables of type **String**:  
[\\_kARPerpendicular](#), [\\_kAR5Axis](#),



```

class AnalysedRabbit : AnalysedTool { // derived from AnalysedTool (added since 15.0.15)

    AnalysedRabbit(); // default constructor
    AnalysedRabbit(Map map); // constructor with a Map, check validity is recommended

    Point3d ptOrg() const;
    CoordSys coordSys() const; // return the CoordSys. The convention is that vecY and vecZ
        point outwards for the rabbet type.

    double dX() const;
    double dY() const;
    double dZ() const;

    int nOverShoot() const; // returns 0 for no overshoot and 1 for overshoot
    int blsRabbit() const; // returns 1 for rabbet and 0 for dado

```

```

Vector3d vecSide() const; // relevant face for Rabbet

static AnalysedRabbit[] filterToolsOfToolType(AnalysedTool[] toolListToFilter);
static AnalysedRabbit[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType);

static int findClosest(AnalysedRabbit[] toolListToSearch, Point3d ptRef); // retuns -1 or
index in array.
};

```

---

static **int** findClosest(**AnalysedRabbit[]** toolListToSearch, **Point3d** ptRef);

The index is returned of the AnalysedRabbit which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedRabbit[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedRabbit[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedRabbit, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedRabbit arRabbit[]=
AnalysedRabbit().filterToolsOfToolType(tools);

int nIndClosest = AnalysedRabbit().findClosest(arRabbit, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

```

---

```

AnalysedRabbit anRabbit = arRabbit[nIndClosest];

// retrieve some data from the Rabbet itself
CoordSys cs = anRabbit.coordSys();
cs.vis(3);

CoordSys csBm = bm.coordSys();
csBm.vis(4);

Point3d ptOrg = anRabbit.ptOrg();
Vector3d vecSide = 2*anRabbit.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
strLines.append(anRabbit.toolType());
strLines.append(anRabbit.toolSubType());
strLines.append("dX: "+anRabbit.dX());
strLines.append("dY: "+anRabbit.dY());
strLines.append("dZ: "+anRabbit.dZ());
strLines.append("dRadius: "+anRabbit.dRadius());
strLines.append("nOverShoot: "+anRabbit.nOverShoot());
strLines.append("bIsRabbit: "+anRabbit.bIsRabbit());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU, 1,1);
}

```

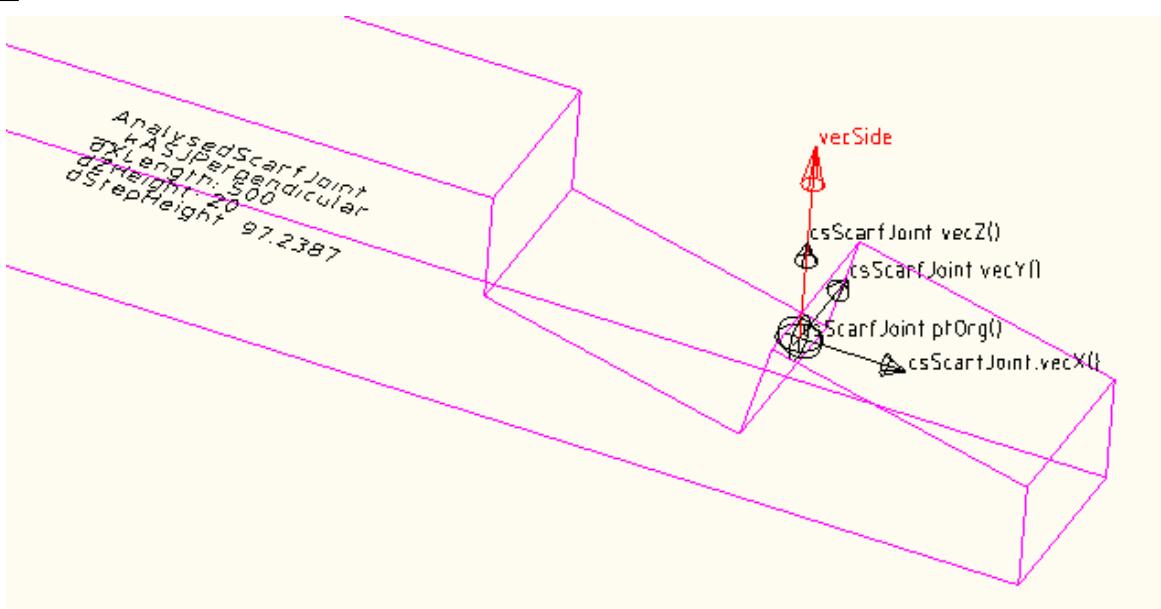
*—end example*

## 9.28 AnalysedScarfJoint

The AnalysedScarfJoint type represents an evaluated tool of type [ScarfJoint](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedScarfJoint. However when the casting is not allowed, the resulting AnalysedScarfJoint will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedScarfJoint is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
`_kASJPerpendicular, _kASJ5Axis`



The convention is that the vecX is pointing to the end of the tool and VecZ is in vecSide direction.

```
class AnalysedScarfJoint : AnalysedTool { // derived from AnalysedTool (added since  
14.0.73)  
  
    AnalysedScarfJoint(); // default constructor  
    AnalysedScarfJoint(Map map); // constructor with a Map, check validity is recommended  
  
    Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the  
    tool  
    CoordSys coordSys() const; // return the CoordSys  
  
    double dXLength() const;  
    double dZHeight() const;  
    double dStepHeight() const;  
  
    Vector3d vecSide() const; // relevant face for ScarfJoint  
  
    static AnalysedScarfJoint[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);  
    static AnalysedScarfJoint[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String  
        strToolSubType);  
  
    static int findClosest(AnalysedScarfJoint\[\] toolListToSearch, Point3d ptRef); // returns -1 or  
    index in array.  
}
```

```
static int findClosest(AnalysedScarfJoint[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedScarfJoint which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedScarfJoint[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedScarfJoint[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
                                                strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedScarfJoint, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles , "Dim style");
PropDouble pTextHeight(0,U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedScarfJoint ScarfJoints[]=
AnalysedScarfJoint().filterToolsOfToolType(tools);

int nIndClosest = AnalysedScarfJoint().findClosest(ScarfJoints, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedScarfJoint ScarfJoint = ScarfJoints[nIndClosest];

// retrieve some data from the ScarfJoint itself
CoordSys csScarfJoint = ScarfJoint.coordSys();
csScarfJoint.vis();

Point3d ptOrg = ScarfJoint.ptOrg();
Vector3d vecSide = 2*ScarfJoint.vecSide();
vecSide.vis(ptOrg,1);

String strLines[0];
```

```

strLines.append(ScarfJoint.toolType());
strLines.append(ScarfJoint.toolSubType());
strLines.append("dXLength: "+ScarfJoint.dXLength());
strLines.append("dZHeight: "+ScarfJoint.dZHeight());
strLines.append("dStepHeight: "+ScarfJoint.dStepHeight());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU, _YU, 1,1);
}

—end example

```

## 9.29 AnalysedShoulderTenon

The AnalysedShoulderTenon type represents an evaluated tool of type ShoulderTenon. The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedShoulderTenon. However when the casting is not allowed, the resulting AnalysedShoulderTenon will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedShoulderTenon is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type `String`:  
`_kASTPerpendicular, _kASTRotated, _kASTTilted, _kAST5Axis,`  
`_kASTTenonPerpendicular, _kASTTenonSimpleAngled, _kASTTenonSimpleBeveled,`  
`_kASTTenonCompound`

---

```

class AnalysedShoulderTenon : AnalysedTool { // derived from AnalysedTool (added since
14.0.11)

AnalysedShoulderTenon(); // default constructor
AnalysedShoulderTenon(Map map); // constructor with a Map, check validity is
recommended

Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
tool
CoordSys coordSys() const; // return the CoordSys

double dBraceAngle() const; // value in degrees
double dBraceHeight() const;
double dGap() const;
double dOuterDepth() const;

```

---

```

double dInnerDepth() const;

Vector3d vecSide() const; // relevant face for anShoulderTenon

static AnalysedShoulderTenon[] filterToolsOfType(AnalysedTool[] toolListToFilter);
static AnalysedShoulderTenon[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
    strToolSubType);

static int findClosest(AnalysedShoulderTenon[] toolListToSearch, Point3d ptRef); // returns -1
    or index in array.
};

```

---

**static int** findClosest(**AnalysedShoulderTenon[]** toolListToSearch, **Point3d** ptRef);

The index is returned of the AnalysedShoulderTenon which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedShoulderTenon[] filterToolsOfType(AnalysedTool[] toolListToFilter)
static AnalysedShoulderTenon[] filterToolsOfType(AnalysedTool[] toolListToFilter, String
    strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedShoulderTenon, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

*Image for the tsI attached to the male part:*

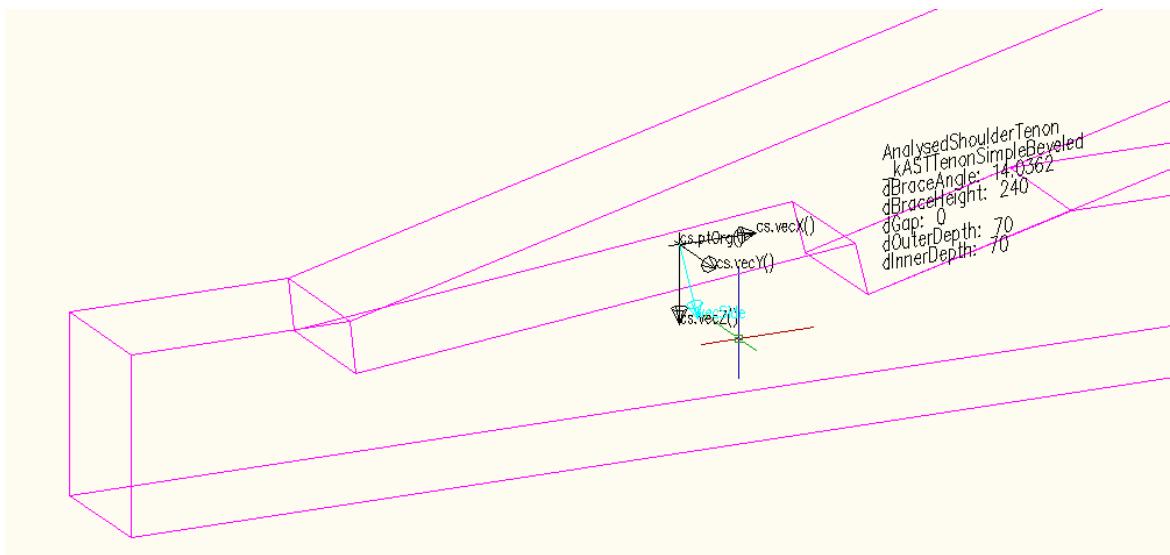
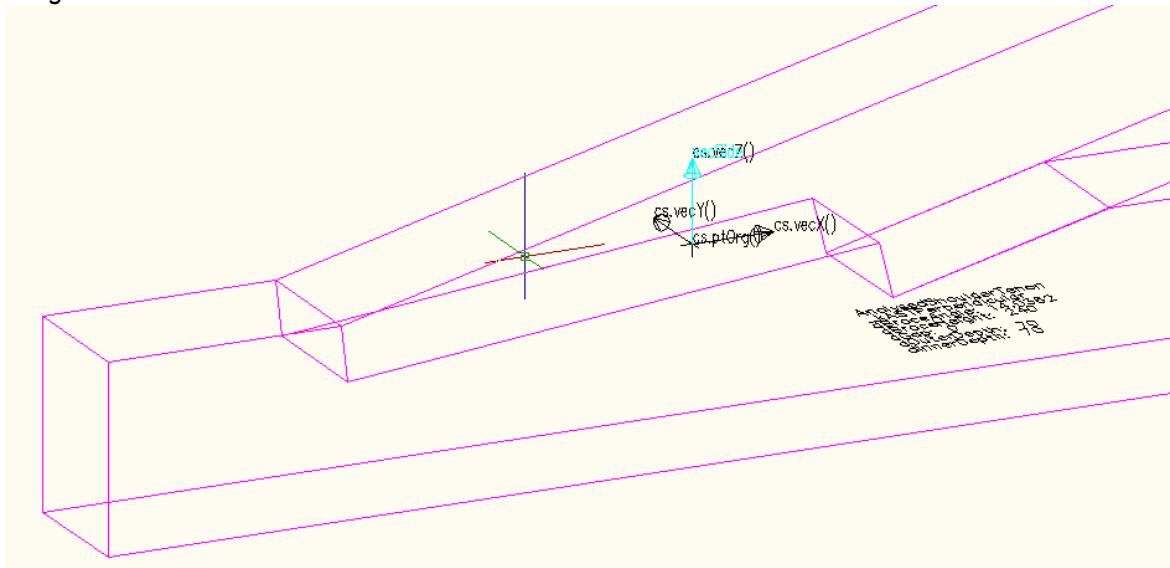


Image for the tsl attached to the female beam:



```

Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedShoulderTenon anShoulderTenons[]=
AnalysedShoulderTenon().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedShoulderTenon().findClosest(anShoulderTenons, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
}

```

```

        eraseInstance(); // calling eraseInstance will notify that the
        tool is not consumed.
    }

AnalysedShoulderTenon anShoulderTenon =
anShoulderTenons[nIndClosest];

// retrieve some data from the anShoulderTenon itself
CoordSys cs = anShoulderTenon.coordSys();
cs.vis();

Point3d ptOrg = anShoulderTenon.ptOrg();
Vector3d vecSide = anShoulderTenon.vecSide();
vecSide.vis(ptOrg,4);

String strLines[0];
strLines.append(anShoulderTenon.toolType());
strLines.append(anShoulderTenon.toolSubType());
strLines.append("dBraceAngle: "+anShoulderTenon.dBraceAngle());
strLines.append("dBraceHeight: "+anShoulderTenon.dBraceHeight());
strLines.append("dGap: "+anShoulderTenon.dGap());
strLines.append("dOuterDepth: "+anShoulderTenon.dOuterDepth());
strLines.append("dInnerDepth: "+anShoulderTenon.dInnerDepth());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l],_Pt0+vecO,_XU,_YU,1,1);
}

```

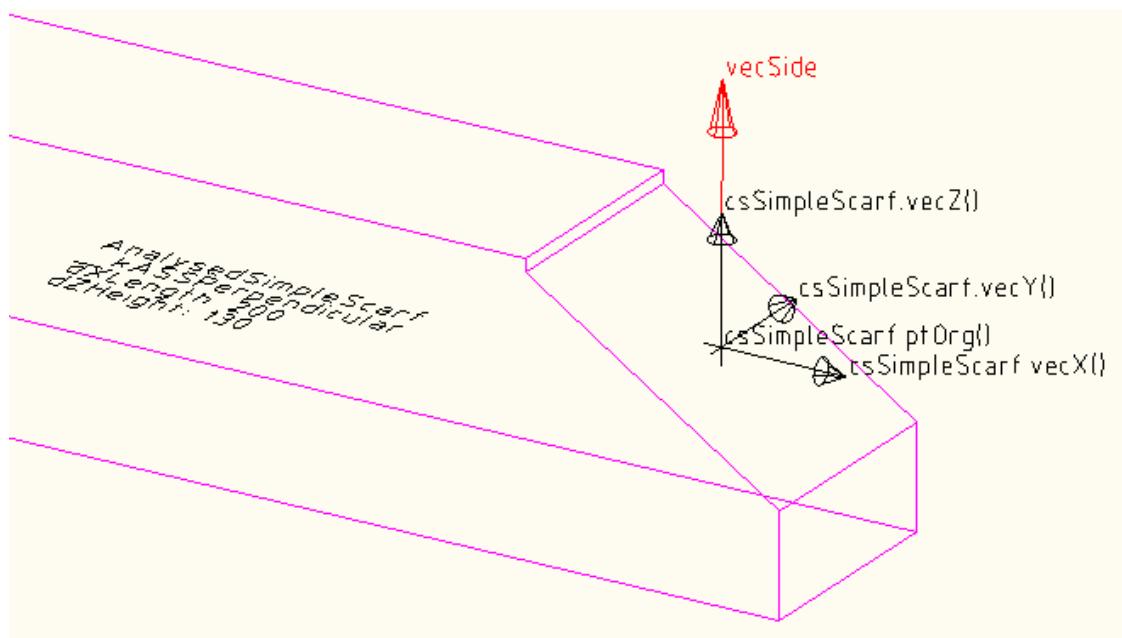
*—end example*

## 9.30 AnalysedSimpleScarf

The AnalysedSimpleScarf type represents an evaluated tool of type [SimpleScarf](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedSimpleScarf. However when the casting is not allowed, the resulting AnalysedSimpleScarf will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedSimpleScarf is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:  
**\_kASSPerpendicular**, **\_kASS5Axis**



The convention is that the vecX is pointing to the end of the tool and VecZ is in vecSide direction.

---

```

class AnalysedSimpleScarf : AnalysedTool { // derived from AnalysedTool (added since
14.0.69)

    AnalysedSimpleScarf(); // default constructor
    AnalysedSimpleScarf(Map map); // constructor with a Map, check validity is
        recommended

    Point3d ptOrg() const; // return point on surface of female beam, at the widest part of the
        tool
    CoordSys coordSys() const; // return the CoordSys

    double dXLength() const;
    double dZHeight() const;

    Vector3d vecSide() const; // relevant face for SimpleScarf

    static AnalysedSimpleScarf[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
    static AnalysedSimpleScarf[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String
        strToolSubType);

    static int findClosest(AnalysedSimpleScarf\[\] toolListToSearch, Point3d ptRef); // returns -1 or
        index in array.
}

```

---

```
static int findClosest(AnalysedSimpleScarf[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedSimpleScarf which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedSimpleScarf[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedSimpleScarf[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String
strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedSimpleScarf, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

[Example E-type:

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedSimpleScarf SimpleScarf[] =
AnalysedSimpleScarf().filterToolsOfToolType(tools);

int nIndClosest =
AnalysedSimpleScarf().findClosest(SimpleScarf[], _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedSimpleScarf simpleScarf = SimpleScarf[nIndClosest];

// retrieve some data from the simpleScarf itself
CoordSys csSimpleScarf = simpleScarf.coordSys();
csSimpleScarf.vis();

Point3d ptOrg = simpleScarf.ptOrg();
Vector3d vecSide = 2*simpleScarf.vecSide();
vecSide.vis(ptOrg,1);
```

```

String strLines[0];
strLines.append(simpleScarf.toolType());
strLines.append(simpleScarf.toolSubType());
strLines.append("dXLength: "+simpleScarf.dXLength());
strLines.append("dZHeight: "+simpleScarf.dZHeight());

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

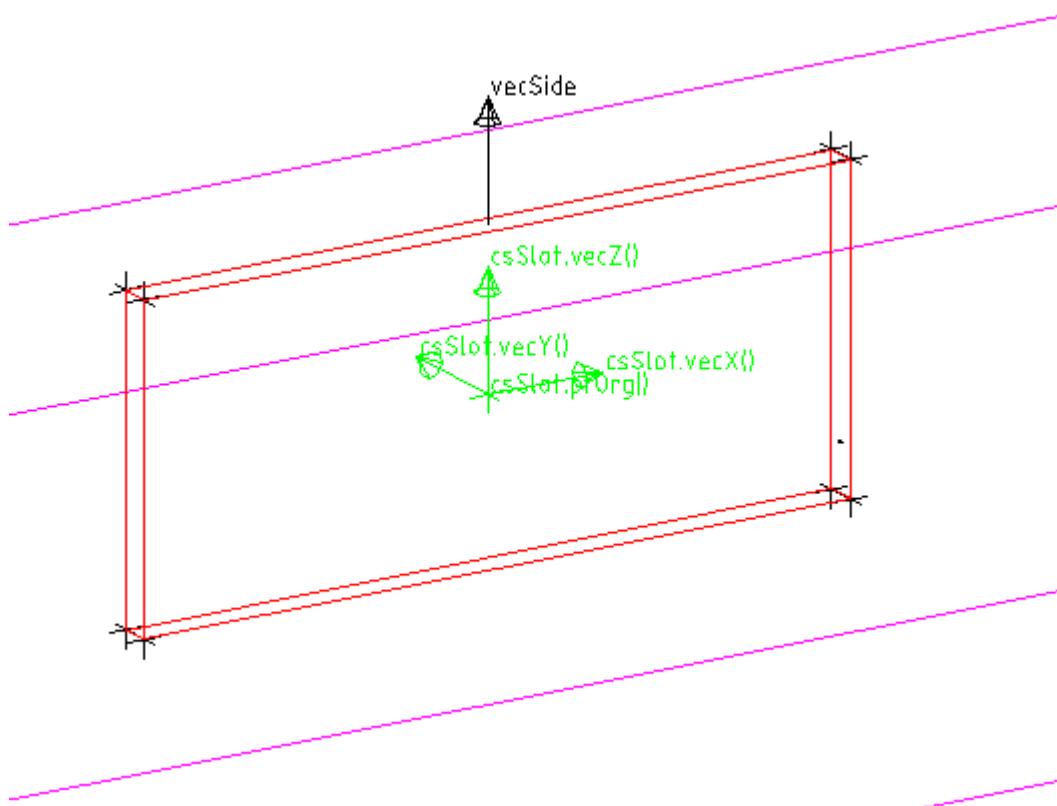
*—end example*

## 9.31 AnalysedSlot

The AnalysedSlot type represents an evaluated tool of type [Slot](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedSlot. However when the casting is not allowed, the resulting AnalysedSlot will become invalid. This can be checked with the `blsValid()` function of AnalysedTool. Because AnalysedSlot is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The [AnalysedTool::toolSubType\(\)](#) returns one of the following predefined variables of type **String**:  
`_kASIPerpendicular, _kASIRotated, _kASITilted, _kASI5Axis`



```
static int findClosest(AnalysedSlot[] toolListToSearch, Point3d ptRef); // retuns -1 or index in
array
};
```

---

```
static int findClosest(AnalysedSlot[] toolListToSearch, Point3d ptRef);
```

The index is returned of the AnalysedSlot which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```
static AnalysedSlot[] filterToolsOfToolType(AnalysedTool[] toolListToFilter)
static AnalysedSlot[] filterToolsOfToolType(AnalysedTool[] toolListToFilter, String strToolSubType)
```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedSlot, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```
Unit(1, "mm");
double dEps = U(0.1);

PropString pDimStyle(0, _DimStyles, "Dim style");
PropDouble pTextHeight(0, U(20), "Text height");

Beam bm = _Beam0;
AnalysedTool tools[] = bm.analysedTools(1);
AnalysedSlot slots[] = AnalysedSlot().filterToolsOfToolType(tools);

int nIndClosest = AnalysedSlot().findClosest(slots, _Pt0);
if (nIndClosest<0) {
    reportMessage("\n"+scriptName() +": No tool found. Instance
erased.");
    eraseInstance(); // calling eraseInstance will notify that the
tool is not consumed.
    return;
}

AnalysedSlot slot = slots[nIndClosest];
Point3d ptArVertices[] = slot.genBeamQuaderIntersectPoints();
for (int p=0; p<ptArVertices.length(); p++) {
    ptArVertices[p].vis();
}
```

```

// retrieve some data from the slot itself
Quader qdrSlot = slot.quader();
qdrSlot.vis(1);
CoordSys csSlot = slot.coordSys();
//csSlot.vis();

Point3d ptOnSurface = slot.ptOnSurface();
Vector3d vecSide = slot.vecSide();
vecSide.vis(ptOnSurface);

String strLines[0];
strLines.append(slot.toolType());
strLines.append(slot.toolSubType());
strLines.append("Angle: "+slot.dAngle());
strLines.append("Bevel: "+slot.dBevel());
strLines.append("Twist: "+slot.dTwist());
strLines.append("Depth: "+slot.dDepth());

Body bdSlot = slot.cuttingBody();

// display the lines
Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end example*

## 9.32 AnalysedSpecialMill

The AnalysedSpecialMill type represents an evaluated tool of type [SpecialMill](#). The type is derived from [AnalysedTool](#).

An AnalysedTool can be casted into an AnalysedSpecialMill. However when the casting is not allowed, the resulting AnalysedSpecialMill will become invalid. This can be checked with the `bIsValid()` function of AnalysedTool. Because AnalysedSpecialMill is derived from AnalysedTool, it inherits all the member functions of AnalysedTool as well. It can also be casted into an AnalysedTool.

The `AnalysedTool::toolSubType()` returns one of the following predefined variables of type **String**:

- \_kASMPерпендикуляр**, **\_kASMRотированный**, **\_kASMTильт**, **\_kASM5Axis**,
- \_kASMHeadPerpendicular**, **\_kASMHeadSimpleAngled**, **\_kASMHeadSimpleBeveled**,
- \_kASMHeadCompound**

```

class AnalysedSpecialMill : AnalysedTool { // derived from AnalysedTool (added since  

13.2.94)

AnalysedSpecialMill(); // default constructor
AnalysedSpecialMill(Map map); // constructor with a Map, check validity is  

recommended

Point3d ptOrg() const;
CoordSys coordSys() const; // return the CoordSys

double dAngle() const; // value in degrees
double dBevel() const; // value in degrees
double dTwist() const; // value in degrees

double dYWidth() const;
double dZDepth() const;

int nToolIndex() const;
String toolName() const;

Vector3d vecSide() const; // relevant face for SpecialMill

static AnalysedSpecialMill[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter);
static AnalysedSpecialMill[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String  

strToolSubType);

static int findClosest(AnalysedSpecialMill\[\] toolListToSearch, Point3d ptRef); // retuns -1 or  

index in array.
};

```

---

static **int** **findClosest([AnalysedSpecialMill\[\]](#) toolListToSearch, [Point3d](#) ptRef);**

The index is returned of the AnalysedSpecialMill which is considered closest to the given ptRef point. To calculate the closest to a ptRef, the distance to the ptOrg is used.

```

static AnalysedSpecialMill[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter)
static AnalysedSpecialMill[] filterToolsOfToolType(AnalysedTool\[\] toolListToFilter, String  

strToolSubType)

```

The list returned is a sub list of the toolListToFilter argument, but casted into AnalysedSpecialMill, if the cast is valid. In case the strToolSubType is specified, and not an empty string, the returned list also matches the strToolSubType. The match is not case sensitive.

---

[Example E-type:

```

U n i t ( 1 , " mm" );
d o u b l e dEps = U( 0 . 1 );

P r o p S t r i n g pDimStyle( 0 , _D I m S t y l e s , " Dim style" );
P r o p D o u b l e pTextHeight( 0 , U( 20 ) , " Text height" );

B e a m bm = _B e a m 0 ;
A n a l y s e d T o o l tools[] = bm. a n a l y s e d T o o l s ( 1 );
A n a l y s e d S p e c i a l M i l l arSpecialMill[] =
A n a l y s e d S p e c i a l M i l l ( ) . f i l t e r T o o l s O f T o o l T y p e ( t o o l s );

I n t n l n d C l o s e s t =
A n a l y s e d S p e c i a l M i l l ( ) . f i n d C l o s e s t ( arSpecialMill , _P t 0 );
i f ( n l n d C l o s e s t < 0 ) {
    r e p o r t M e s s a g e ( " \n " + s c r i p t N a m e ( ) + ": No tool found. Instance erased." );
    e r a s e l n s t a n c e ( ) ; // calling eraseInstance will notify that the tool is not consumed.
    r e t u r n ;
}

A n a l y s e d S p e c i a l M i l l anSpecialMill = arSpecialMill[ n l n d C l o s e s t ];

// retrieve some data from the specialMill itself
C o o r d S y s csSpecialMill = anSpecialMill. c o o r d S y s ( );
csSpecialMill. v i s ( ) ;

P o i n t 3 d ptOrg = anSpecialMill. p t O r g ( );
V e c t o r 3 d vecSide = anSpecialMill. v e c S i d e ( );
vecSide. v i s ( ptOrg );

S t r i n g strLines[ 0 ];
strLines. append( anSpecialMill. t o o l T y p e ( ) );
strLines. append( anSpecialMill. t o o l S u b T y p e ( ) );
strLines. append( " Angle: " + anSpecialMill. d A n g l e ( ) );
strLines. append( " Bevel: " + anSpecialMill. d B e v e l ( ) );
strLines. append( " Twist: " + anSpecialMill. d T w i s t ( ) );
strLines. append( " Y W i d t h: " + anSpecialMill. d Y W i d t h ( ) );
strLines. append( " Z D e p t h: " + anSpecialMill. d Z D e p t h ( ) );
strLines. append( " n T o o l I n d e x: " + anSpecialMill. n T o o l I n d e x ( ) );
strLines. append( " t o o l N a m e: " + anSpecialMill. t o o l N a m e ( ) );

// display the lines
D i s p l a y dp( - 1 );
dp. d i m S t y l e ( p D i m S t y l e );
dp. t e x t H e i g h t ( p T e x t H e i g h t );
f o r ( I n t l = 0 ; l < strLines. l e n g t h ( ) ; l ++ ) {
    V e c t o r 3 d vecO = - l * 1 . 2 * p T e x t H e i g h t * _Y U ;
    dp. d r a w ( strLines[ l ] , _P t 0 + vecO , _X U , _Y U , 1 , 1 );
}

```

—end example

**Part**



X

## 10 Shopdraw engine

### 10.1 EntCollector

During the running of the shopdraw engine, ts1's are triggered either from the ruleset specified in the shopdraw style or from the block that is part of the layout in the shopdraw style. When these ts1's are triggered information is given to them in the \_Map, typically in a subMap with a name referring to the context of the event that fires them.

For Ts1's as part of the ruleset the subMap is called "Generation". It contains information about the multipage entity, and the generation process taking place. The multipage entity is found as "entCollector" (added hsbCAD13.2.124).

For Ts1's as part of the block that defines the layout, the subMap is called "OnGenerateShopDrawing" also available as `_kOnGenerateShopDrawing`. It contains the multipage entity as "entCollector" (added hsbCAD16.1.26).

*[Example of O-type that illustrates the Generator map and entCollector entry. This Ts1 can be added to the ruleset in the shopdraw style or to the block of the layout of a shopdraw style.*

```
{
    // for debugging export map
    String strTsl = scriptName()+"-"+_kExecuteKey;
    reportMessage("\nTsl execution: "+strTsl);
    _Map.writeToDxxFile(_kPathDwg+"\\"+strTsl+".dxx", FALSE);

    // get Multipage entity being created or regenerated
    Entity entCollector;
    entCollector = _Map.getEntity("Generation\\entCollector");
    if (!entCollector.bIsValid())
        entCollector = _Map.getEntity(_kOnGenerateShopDrawing +"\\
entCollector");
    if (!entCollector.bIsValid()) {
        reportMessage("\n"+strTsl+": entCollector not found");
    }
    else {
        Map mp;
        if (entCollector.subMapKeys().find("subMap3")>=0) {
            Map mpL = entCollector.subMap("subMap3");
            String strState= mpL.getString("state");
            reportMessage("\n"+strTsl+": Map already contains subMap3:
"+strState);
        }
        else {
            reportMessage("\n"+strTsl+": Map does not contain
subMap3");
        }
        mp.setString("state",strTsl);
        entCollector.setSubMap("subMap3",mp);
    }
}
```

```
}
```

—end sample]

## 10.2 ActiveRule subMap

During the running of the shopdraw enging, tsI's are triggered from the ruleset specified in the shopdraw style. When these tsI's are triggered the \_Map contains another special subMap. This subMap is called "ActiveRule". It contains information about the rule of the ruleset which is causing the tsI to be executed. One of the members of the ActiveRule is the [shopdraw category](#).

*[Example of O-type that illustrates the ActiveRule map entry. This TsI can be added to the ruleset in the shopdraw style.*

```
Map mpActiveRule = _Map.getMap("ActiveRule");

// String strActiveRule = mpActiveRule.getDxContent(FALSE);
// reportMessage("\n"+strActiveRule);

reportMessage("\n"+scriptName()+" AcitveRule:");
String str;
str = mpActiveRule.getString("Category");
reportMessage("\nCategory: "+str);
str = mpActiveRule.getString("ToolType");
reportMessage("\nToolType: "+str);
str = mpActiveRule.getString("ToolSubType");
reportMessage("\nToolSubType: "+str);
int nVal = mpActiveRule.getInt("ConsumeIt");
reportMessage("\nConsumeIt: "+nVal);
```

*The following output could be generated:*

```
Category: _kRCTool
ToolType: AnalysedCut
ToolSubType: _kACPPerpendicular
ConsumeIt: 0
```

—end sample]

## 10.3 Selection filter

From hsbCAD2009+ (build 14.0.30) on, one can specify a selection filter tsI in the shopdraw style. This tsI is active only during the insertion of multipages in the drawing. The selection set for which a shopdraw multipage is generated is investigated by the tsI specified.

From hsbCAD2009+ (build 14.4.6) and hsbCAD2010 (build 15.1.26) the point in the process of the insertion of the multipage when the tsI is called is changed. From these builds on, the tsI is called after the selection of the entities. This means before the Beams are grouped on posnum.

The set of entities that are selected are passed to the tsI in the \_Entity array. This array can be changed at will by the tsI author. Entities can be added and deleted resulting in more or less entities presented to the shopdraw multipage engine.

Versions prior to hsbCAD2009+ (build 14.4.6) and hsbCAD2010 (build 15.1.26) have the tsl called for each entity right before the shopdraw multipage is about to be inserted, after the sorting for posnums.

If the Tsl returns eraselInstance the \_Entity array is not accepted and no shopdrawings are inserted.

From hsbCAD2013 (build 18.1.51) on you should be able to implement the entity selection by tsl. Before it was only possible to filter out, or add entities to the selection set. Now the tsl can actually do the prompting. The toggle that is used to identify this, is the “implement insert” toggle in the tsl script. If that one is on, the filter tsl is the one in charge of the prompting.

---

*[Example of O-type that illustrates the use as shopdraw multipage insertion filter. The Tsl works with shop drawings for beams. The beams that do not belong to an element are rejected by this tsl. As a result when this tsl is specified in the "Selection filter TSL", no shopdrawing is generated for these rejected beams.*

```

int bInDebug = TRUE;
if (bInDebug) reportNotice("\n"+scriptName()+" : ");

if (_Entity.length()==0) {
    if (bInDebug) reportNotice("No entities in selection set.");
    return; // do not do anything
}

// take copy of _Entity array
Entity arAll[_Entity.length()];
for (int i=0; i<_Entity.length(); i++)
    arAll[i] = _Entity[i];

// reset content of _Entity
_Entity.setLength(0);

// filter the beams that belong to an element
for (int i=0; i<arAll.length(); i++) {

    Beam bm = (Beam)arAll[i];
    if (!bm.bIsValid()) {
        if (bInDebug) reportNotice("Not a beam in selection set.");
        continue;
    }
    Element el = bm.element();
    if (!el.bIsValid()) {
        if (bInDebug) reportNotice("Beam does not belong to an
element.");
        continue;
    }
}

```

---

```

        }

        // add the entities back to _Entity which need to be kept in the
        selection
        if (bInDebug) reportNotice("\nKeep beam: "+bm);
        _Entity.append(bm);
    }

    if (bInDebug) reportNotice("_Entity length: "+_Entity.length());
}

—end sample]

```

*[Example of O-type that illustrates the use as shopdraw multipage insertion filter but also implements the entity selection itself. The Tsl works with shop drawings for beams. The beams that do not belong to an element are rejected by this tsl. As a result when this tsl is specified in the "Selection filter TSL", no shopdrawing is generated for these rejected beams.*

```

if (_bOnInsert) {
    // _Entity.append(getBeam(T("|Select one beam|")));

    PrEntity ssE(T("|Select a set of element or red shopdraw beams|"),
    Beam());
    if (ssE.go()) {
        _Entity = ssE.set();
        reportMessage (T("\n|Number of entities selected:| ") +
        _Entity.length());
    }
}

int bInDebug = FALSE;
if (bInDebug) reportNotice("\n"+scriptName()+":");

if (_Entity.length() == 0) {
    if (bInDebug) reportNotice("\nNo entities in selection set.");
    return; // do not do anything
}

// take copy of _Entity array
Entity arAll[_Entity.length()];
for (int i=0; i<_Entity.length(); i++)
    arAll[i] = _Entity[i];

// reset content of _Entity
_Entity.setLength(0);

// filter the beams that belong to an element or have color red
for (int i=0; i<arAll.length(); i++) {
    Beam bm = (Beam)arAll[i];
}

```

```

    if (!bm.bIsValid()) {
        if (bInDebug) reportNotice("\nNot a beam in selection set.");
        continue;
    }

    int bBeamBelongsToElement = bm.element().bIsValid();
    int bBeamIsRed = (bm.color() == 1);

    if (!bBeamBelongsToElement && !bBeamIsRed) {
        if (bInDebug) reportNotice("\nBeam does not belong to an element
and is not red.");
        continue;
    }

    // add the entities back to _Entity which need to be kept in the
selection
    if (bInDebug) reportNotice("\nKeep beam: " + bm);
    _Entity.append(bm);
}

if (bInDebug) reportNotice("\n_Entity length: " + _Entity.length());

```

*—end sample]*

## 10.4 OnMapIO

From hsbCAD 2009+, build 14.0.78 on. When the Map button "..." is pressed in the "Define rule for the shopdrawing tsl to be applied" dialog (see button marked with red circle in image below), the tsl specified in the "Tsl name" field is executed in a special mode. The purpose of this mode is to allow the user of the Tsl to set the Map content through some UI. The variable `_bOnMapIO` is TRUE during this execution.

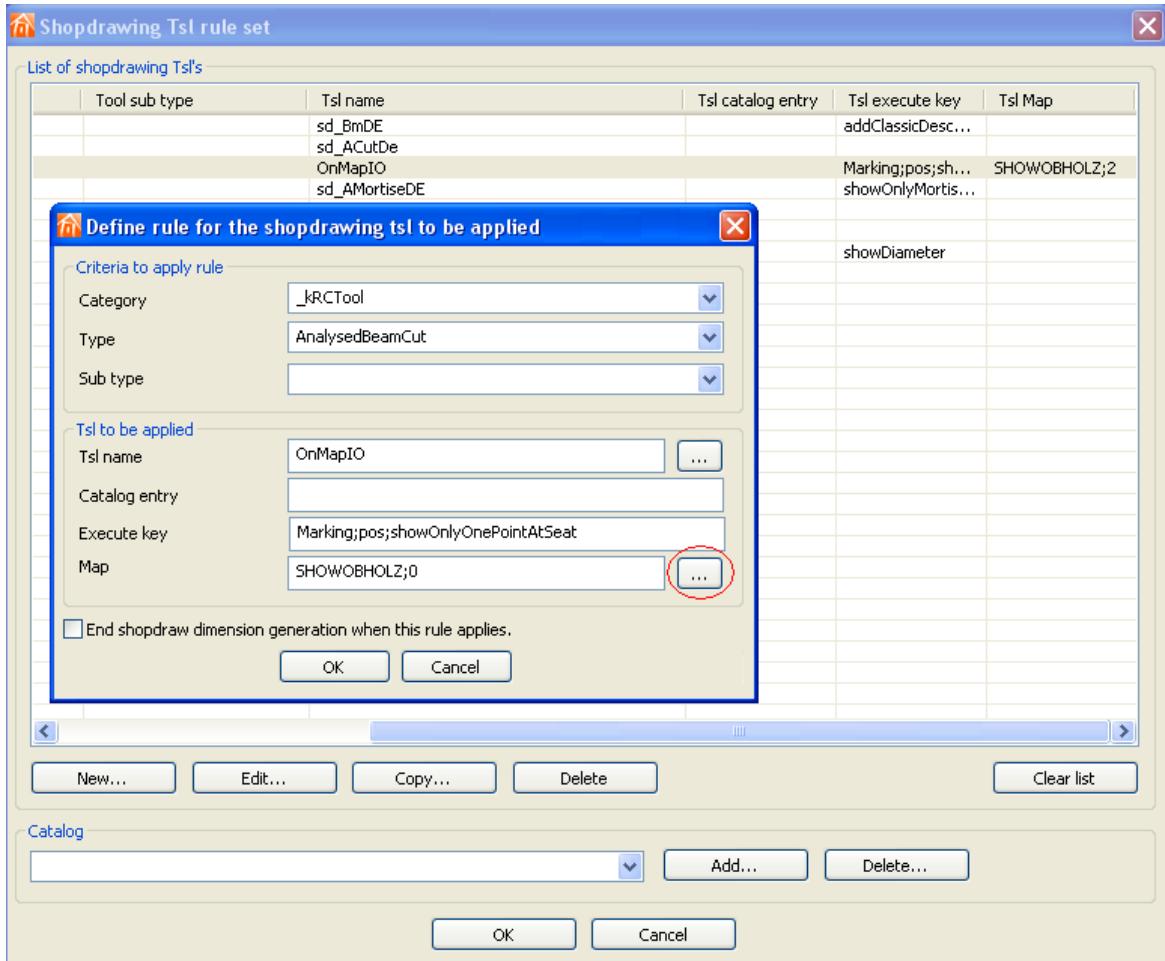
The Tsl is fired with the execution key, catalog entry and Map set. It is up to the Tsl to interpret the `_Map`, present a dialog to the user, and change the `_Map` from the dialog. If the tsl does not fire the `eraseInstance()` method, the semicolon separated `_Map` content will be set to the dialogs "Map" field.

The examples below illustrates a typical use, both with and without the global insert methods:

```

Map mapWithPropValues(); // (added hsbCAD14.0.78) // see also global insert functions and
TslInst
int setPropValuesFromMap(Map mapWithPropValues); // return TRUE if successful (added
hsbCAD14.0.78)

```



[Example1 of O-type that illustrates a ruleset Tsl that has a property dialog for the Mapkey entry.

```

U(1,"mm");

if (_bOnMapIO) {

    // define property
    String arObholz[] = {T("|Do not dimension Obholz|"),T("|display
plump Obholz|"),T("|display perpendicular Obholz|")};
    PropString pObholz(0,arObholz,T("|Show Obholz dimension|"));
    pObholz.setDescription(T("|Sets how the Obholz needs to be
dimensioned|"));

    // find value in _Map, if found, change the property values
    String strKeyObholz = "showObholz"; // key in _Map
    if (_Map.hasString(strKeyObholz)) {
        int nInd = _Map.getString(strKeyObholz).atoi();
        String strOb = arObholz[nInd];
        pObholz.set(strOb); // set the property value from the _Map
contents
}

```

```

        reportNotice("\n_map contains "+strKeyObholz+": "+ nInd);
        reportNotice("\nproperty value changed to: "+ strOb);
    }
    else {
        reportNotice("\n_map does not contain key: "+strKeyObholz);
    }

    // show the dialog to the user
    showDialog("----"); // use "----" such that the set values are used,
and not the last dialog values

    // interpret the properties, and fill _Map
    reportNotice("\nproperty value after showDialog: "+ pObholz);
    int nInd = arObholz.find(pObholz,0);
    _Map.setString(strKeyObholz,nInd);
    reportNotice("\n_map value changed to: "+ nInd);

    // do not call eraseInstance(), otherwise no _Map changes are
accepted
    return;
}

```

*—end sample]*

*[Example2 of O-type that illustrates a ruleset Tsl that has a property dialog for the Mapkey entry.*

```

U(1,"mm");

if (_bOnMapIO) {

    // define property
    String arObholz[] = {T("|Do not dimension Obholz|"),T("|display
plump Obholz|"),T("|display perpendicular Obholz|")};
    PropString pObholz(0,arObholz,T("|Show Obholz dimension|"));
    pObholz.setDescription(T("|Sets how the Obholz needs to be
dimensioned|"));

    // load the property values from my Map
    setPropValuesFromMap(_Map);

    // show the dialog to the user
    showDialog("----"); // use "----" such that the set values are used,
and not the last dialog values

    // interpret the properties, and fill _Map
    _Map = mapWithPropValues();

    // for debug view the map file
    String strFileMap = "c:\\\\Temp\\\\" + scriptName() + ".dxx";
    _Map.writeToDxxFile(strFileMap);
}

```

```

spawn_detach("", _kPathHsbInstall+"\\Utilities\\DxxExplorer\
\\DXExplorer.exe", strFileMap, "");

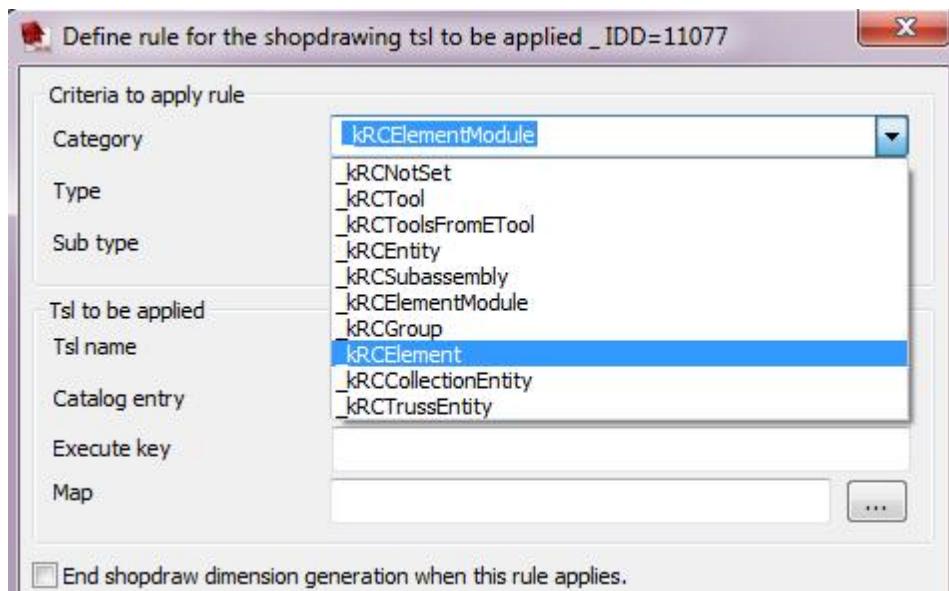
// do not call eraseInstance(), otherwise no _Map changes are
accepted
return;
}

—end sample]

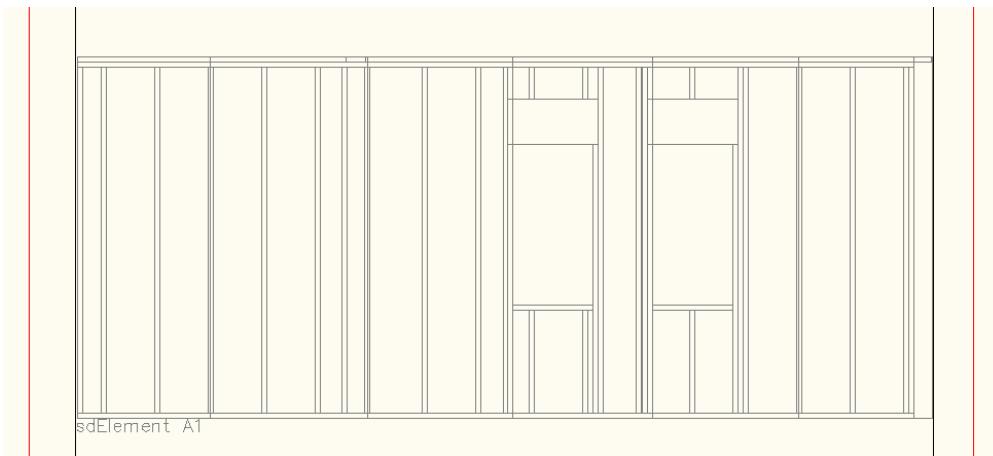
```

## 10.5 Shopdraw categories

Each rule in the Tsl ruleset has a category. The rule information, including the category is available during execution of the tsl through the [ActiveRule](#) submap in [\\_Map](#). The category value is a string and can have the following values: [\\_kRCTool](#), [\\_kRCToolsFromETool](#), [\\_kRCEntity](#), [\\_kRCSubassembly](#), [\\_kRCElementModule](#), [\\_kRCGroup](#), [\\_kRCElement](#), [\\_kRCCollectionEntity](#), [\\_kRCTrussEntity](#),...



*[Example of O-type that illustrates a ruleset tsl that shows the code and number of the element. This Tsl can be added to the ruleset in the shopdraw style.*



```

if (_bOnInsert) {
    // when this ts1 is inserted as entity in the drawing (so not in
    shopdraw ruleset)
    Element elem = getElement();
    _Element.append(elem);
    _Pt0 = elem.ptOrg();
    return;
}

if (_bOnDebug || _bOnGenerateShopDrawing) {
    int nEnts = _Entity.length();
    String strRep = scriptName()+" reports "+nEnts+" entities.";
    for (int e=0; e<_Entity.length(); e++) {
        Entity entE = _Entity[e];
        String strEnt = entE.typeName();
        strRep += "\n" + strEnt;
    }
    reportMessage("\n"+strRep );
}

if (_Entity.length()==0)
    return;
Element elem = (Element) _Entity[0];
if (!elem.bIsValid())
    return;

CoordSys csEl = elem.coordSys();

// compose a DimRequestText
int nDebugColorCounter = 1;
String strParentKey = String("Dim "+scriptName());
Point3d ptText = csEl .ptOrg();
Vector3d vecText = csEl .vecX();
Vector3d vecOff = -csEl .vecY();

```

```

String strText = scriptName() + " " + elem.code() + elem.number();
Vector3d vecView = csEl .vecZ();

DimRequestText dr(strParentKey, strText, ptText, vecText, vecOff);
dr.addAllowedView(vecView, TRUE);
dr.vis(nDebugColorCounter++); // visualize for debugging
addDimRequest(dr); // append to shop draw engine collector

—end sample]

```

## 10.6 Shopdraw ViewData

The class [ViewData](#) is described in relation with [Viewport](#) for working in paperspace layouts. But it can also hold the data corresponding to a [ShopDrawView](#) entity (hsbCad View Placeholder) which is part of a shopdraw layout. The example below explains how your Tsl can be put to work in a shopdraw context.

During the shopdraw context the Tsl would be inserted selecting a ShopDrawView.

```

Entity ent = getShopDrawView(T("|Select the view entity|")); // 
select ShopDrawView
_Entity.append(ent);

```

During shopdraw generation this ShopDrawView entity can be used to retrieve the ViewData for that ShopDrawView. The ViewData is given to the Tsl in its `_Map`. In fact all ViewData of all the ShopDrawViews might be present in the `_Map`, But the ViewData corresponding to the ShopDrawView that was selected during insert can be found with the following method (error handling stripped).

```

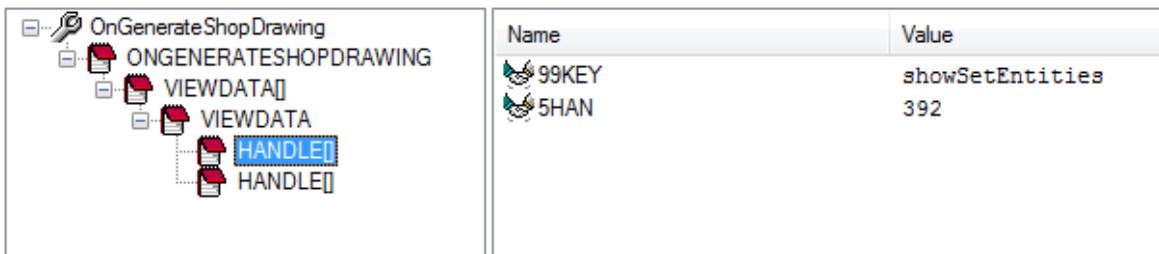
ShopDrawView sv = (ShopDrawView)_Entity[0];
ViewData arViewData[] = ViewData().convertFromSubMap( _Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets, 0);
int nIndFound = ViewData().findDataForViewport(arViewData, sv); // 
find the viewData for my view
ViewData vwData = arViewData[nIndFound];

```

As you see, the `_kOnGenerateShopDrawing` and `_kViewDataSets` predefined variables are used to access the map. The `convertFromSubMap` returns a collection of ViewData instances.

When drawing text by default the text style and height are used specified in the shopdraw style.





[Example O-type TSL called "PageBeamHatch" to be attached to a ShopDrawView inside a shopdraw block, illustrating ViewData by hatching a surface of a beam in the shopdrawing.



```

Unit (1, "mm");
double dEps = U(0.1);

PropString sHatchPattern(0, _HatchPatterns, T("|Hatch pattern|"));

//Insert
if( _bOnInsert ){
    _Entity.append(getShopDrawView(T("Select the view entity")));
    _Pt0 = getPoint();
    return;
}

if( _Entity.length() == 0 ){ eraseInstance(); return; }
_Entity entView = _Entity[0];

// interprete the list of ViewData in my _Map
ViewData arViewData[] = ViewData().convertFromSubMap( _Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets, 2); // 2 means
verbose
int nIndFound = ViewData().findDataForViewport(arViewData,
entView); // find the ViewData for my view

PlaneProfile prof;
CoordSys ms2ps;

if (nIndFound<0) { // not during _bOnGenerateShopDrawing

```

```

PLine plCir;
plCir.createCircle(_Pt0,_ZU,U(500));
prof = PlaneProfile(plCir);
}

if (nIndFound>=0) {
ViewData vwData = arViewData[nIndFound];
Entity arEnt[] = vwData.showSetDefineEntities();
ms2ps = vwData.coordSys();

CoordSys ps2ms = ms2ps;
ps2ms.invert();

Vector3d vecView = _ZW;
vecView .transformBy(ps2ms);

Beam bm;
if (arEnt.length()>0) bm = (Beam)arEnt[0];
if (bm.bIsValid()) {
    // hatch the face of the beam which we are looking at
    Body bdBeam = bm.realBody();
    Vector3d vecBeamView = bm.vecD(vecView);
    Point3d ptBeamSurface = bm.ptCen()+0.5*bm.dD(vecBeamView)
*vecBeamView;
    prof = bdBeam.extractContactFaceInPlane(Plane(ptBeamSurface,
vecBeamView),U(1));
    prof.transformBy(ms2ps); // the prof is in ms, so transform it
to ps
}
}

Display dp(3); // green

// if (nIndFound== -1) // we are not during _bOnGenerateShopDrawing
{
    // use the style and size specified in the shopdraw style
    dp.draw(scriptName(), _Pt0, _XU, _YU, 1, 7 );
    dp.draw(sHatchPattern, _Pt0, _XU, _YU, 1, 4 );
}

Hatch hatch(sHatchPattern,U(10));
dp.draw(prof,hatch);

—end example]

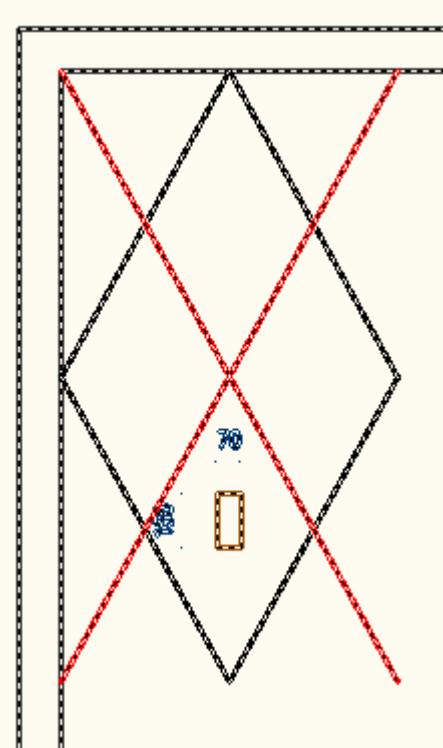
```

## 10.7 Display for shopdraw multipage

If a Tsl is added to the layout block that is used in the page of a multipage, then the Tsl will be executed whenever the regeneration of the shopdrawing is triggered. In this case the status variable `_bOnGenerateShopDrawing` is set to TRUE. This status variable is also set to TRUE during special `shopdraw recalc triggers`, otherwise it is FALSE.

Information from the page regarding view data, is put in the `_Map` of the tsl with mapkey `_kOnGenerateShopDrawing`. See also [ShopDraw ViewData](#).

*[Example O-type TSL to be attached to a ShopDrawView inside a shopdraw block, illustrating ViewData, ShopDrawView.*



```
Unit(1, "mm");

if (_bOnInsert)
{
    _Pt0 = getPoint();
    _Entity.append(getShopDrawView(T("|Select the view entity|")));
    return;
}

if(_Entity.length() == 0){ eraseInstance(); return;}
Entity entView = _Entity[0];
```

```

// interprete the list of ViewData in my _Map
ViewData ar ViewData[] = ViewData() .convertFromSubMap( _Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets, 2); // 2 means
verbose
int nIndFound = ViewData() .findDataForViewport(ar ViewData,
entView); // find the viewData for my view
if (nIndFound== -1) { // we are not during _bOnGenerateShopDrawing
    Display dp(-1);
    dp.textHeight(U(100));
    dp.draw(scriptName(), _Pt0, _XU, _YU, 1.2, 1.2, _kDevice);
    reportMessage("\nNo ViewData that matches my entity "+entView);
    return;
}

ViewData vwData = ar ViewData[nIndFound];

Point3d ptCenPS = vwData.ptCenPS();
CoordSys vwCsPS = vwData.coordSysPS();
Vector3d vecXPS = vwCsPS.vecX();
Vector3d vecYPS = vwCsPS.vecY();
double dWidth = vwData.widthPS();
double dHeight = vwData.heightPS();

Point3d pt1M = ptCenPS + 0.5*dWidth*vecXPS;
Point3d pt2M = ptCenPS + 0.5*dHeight*vecYPS;
Point3d pt3M = ptCenPS - 0.5*dWidth*vecXPS;
Point3d pt4M = ptCenPS - 0.5*dHeight*vecYPS;
Point3d pt1C = ptCenPS + 0.5*dWidth*vecXPS + 0.5*dHeight*vecYPS;
Point3d pt2C = ptCenPS - 0.5*dWidth*vecXPS + 0.5*dHeight*vecYPS;
Point3d pt3C = ptCenPS - 0.5*dWidth*vecXPS - 0.5*dHeight*vecYPS;
Point3d pt4C = ptCenPS + 0.5*dWidth*vecXPS - 0.5*dHeight*vecYPS;

// draw hatch pattern
Display dp(-1);
dp.draw(LineSeg(pt1M, pt2M));
dp.draw(LineSeg(pt2M, pt3M));
dp.draw(LineSeg(pt3M, pt4M));
dp.draw(LineSeg(pt4M, pt1M));
dp.color(1);
dp.draw(LineSeg(pt1C, pt3C));
dp.draw(LineSeg(pt2C, pt4C));

—end example]

```

## 10.8 Shopdraw recalc triggers

All Tsl's that are part of the block that defines the shopdraw layout are fired to collect graphics. In this case the status variable **\_bOnGenerateShopDrawing** is set to TRUE (see also [Display for shopdraw multipage](#)).

But the Tsl can also register itself to be fired at other events during the shopdraw creation. The event trigger must be set as:

### **\_kShopDrawEvent**

Similar to registering a context action the Tsl instance can register itself by calling the addRecalcTrigger:

```
addRecalcTrigger(_kShopDrawEvent, strKey );
```

The strKey needs to be one of the following keys:

### **\_kShopDrawChildPageArea, \_kShopDrawChildPageCreate**

The purpose of such an event execution is to pass information to the shopdraw engine. This is done through the \_Map of the Tsl instance. A special submap with the name

**\_kShopDrawChildPageArea, \_kShopDrawChildPageCreate** is used to pass information from the shopdraw engine to the Tsl, and back.

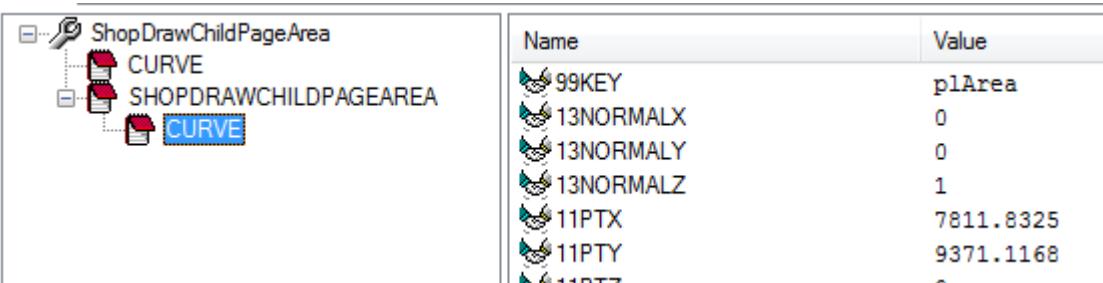
Please see the topics [ShopDrawChildPageArea](#) and [ShopDrawChildPageCreate](#) for examples.

#### **10.8.1 ShopDrawChildPageArea**

If the tsl which is part of the block of the shopdraw layout has registered itself for the event **\_kShopDrawChildPageArea** then it gets executed with the state:

```
if (_bOnGenerateShopDrawing && _kExecuteKey==_kShopDrawChildPageArea)
```

The purpose of the execution is to pass a PLine to the shopdraw engine. This must be done through \_Map. Here you see the \_Map that was generated by the Tsl example below:



The screenshot shows a software interface with a tree view on the left and a table on the right. The tree view shows a node named 'ShopDrawChildPageArea' which contains a 'CURVE' node and a 'SHOPDRAWCHILDPAGEAREA' node. The 'SHOPDRAWCHILDPAGEAREA' node contains another 'CURVE' node. The table on the right lists the following data:

Name	Value
99KEY	plArea
13NORMALX	0
13NORMALY	0
13NORMALZ	1
11PTX	7811.8325
11PTY	9371.1168
11DT7	~

[Example O-type TSL to be added inside a shopdraw block, illustrating special recalc trigger.

```
String strPrompt = T("|Select pline area where child pages can be placed|");

if (_bOnInsert)
{
    _Pt0 = getPoint();
```

```

EntPLine ent = getEntPLine(strPrompt);

// store the PLine from the selected entity in my map
.setPLine("myPL",ent.getPLine());

// erase selected pline
ent.dbErase();
return;
}

// for debugging
reportMessage("\nTsl execution: " + scriptName() + ":" + 
_kExecuteKey + " " + _bOnGenerateShopDrawing );

String strChangeEntity = T("|Select different pline|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    EntPLine ent = getEntPLine(strPrompt );
    // store the PLine from the selected entity in my map
    .setPLine("myPL",ent.getPLine());
    // erase selected pline
    ent.dbErase();
}

// retrieve my own pline
PLine pline = .getPLine("myPL");

addRecalcTrigger(_kShopDrawEvent, _kShopDrawChildPageArea );
if (_bOnGenerateShopDrawing && _kExecuteKey==_kShopDrawChildPageArea)

{
    // create a map with content plArea
    .setPLine(_kShopDrawChildPageArea+ "\\"+plArea, pline);

    // for debugging export map
    .writeToDxxFile(_kPathDwg+"\\"+
"+_kShopDrawChildPageArea+".dxx");
}

// do not draw pline during shopdraw generation
if (_bOnGenerateShopDrawing==TRUE) {
    return;
}

// display in normal block mode
Display dp(-1);
dp.draw(pline);

—end example]

```

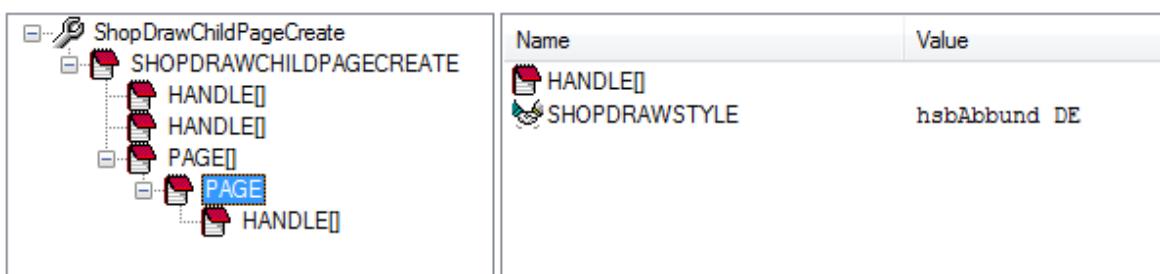
## 10.8.2 ShopDrawChildPageCreate

If the tsl which is part of the block of the shopdraw layout has registered itself for the event `_kShopDrawChildPageCreate` then it gets executed with the state:

```
if (_bOnGenerateShopDrawing &&
    _kExecuteKey==_kShopDrawChildPageCreate)
```

The purpose of the execution is to pass the list of pages to the shopdraw engine for which child shopdraw pages needs to be generated. For each page, a defineSetEntities list of entities must be added, as well as a shopdrawstyle. This must be done through `_Map`. Here you see the `_Map` that was generated by the Tsl example below.

Besides adding the list of defineSetEntities, the showSetEntities can be specified as well. (since build 16.1.3).



*[Example O-type TSL to be added inside a shopdraw block, illustrating special recalc trigger.*

```
Unit(1, "mm");

PropString sShopDrawStyle(0, "", T("Shopdraw style"));

if (_bOnInsert)
{
    _Pt0 = getPoint();
    showDialogOnce();
    return;
}

reportMessage("\nTsl execution: " + scriptName() + ":" + 
sShopDrawStyle + " - " + _kExecuteKey + " " +
_bOnGenerateShopDrawing);

addRecalcTrigger(_kShopDrawEvent, _kShopDrawChildPageCreate );
if (_bOnGenerateShopDrawing &&
    _kExecuteKey==_kShopDrawChildPageCreate )
{
    String strMapLocation = _kShopDrawChildPageCreate + "\\" +
"Handle[]";
```

```

Entity arEnts[] = _Map.getEntityArray(strMapLocation,
"defineSetEntities", "han");

if (arEnts.length()==0) { // no entities to shopdraw?
    reportMessage("\nNo entities to shopdraw.");
}

// find the horizontal beams
Beam arBeams[0];
for (int e=0; e<arEnts.length(); e++) {
    Beam bm = (Beam)arEnts[e];
    if (!bm.bIsValid())
        continue;
    if (bm.vecX().isPerpendicularTo(_ZW)) {
        arBeams.append(bm);
    }
}

String strPage = "Page";
// For each beam create a map that provides the info for one page
// to be inserted in the layout of the current shopdrawing
for (int b=0; b<arBeams.length(); b++) {
    Beam bm = arBeams[b];
    Map mpPage;

    // add the shopDrawStyle
    mpPage.setString("shopDrawStyle", sShopDrawStyle);

    // add the define set for the page to be inserted
    Entity arEntsNewDefineSet[0];
    arEntsNewDefineSet.append(bm);
    mpPage.setEntityArray(arEntsNewDefineSet, TRUE, "Handle[]",
"defineSetEntities", "han");

    // now add Page[]\\Page to _Map
    _Map.appendMap(_kShopDrawChildPageCreate + "\\" + strPage +
"[]" + "\\" + strPage, mpPage); // add to _Map with special key
}

// for debugging export map
_Map.writeToDxxFile(_kPathDwg+"\\"+_kShopDrawChildPageCreate
+".dxx");
}

// do not draw anything during shopdraw generation
if (_bOnGenerateShopDrawing==TRUE) {
    return;
}

// display in normal block mode

```

```

Display dp(-1);
dp.textHeight(U(100));
dp.draw(scriptName()+" "+sShopDrawStyle, _Pt0, _XU, _YU,
1.2,1.2, _kDevice );

```

*—end example]*

### 10.8.3 ShopDrawViewDataSet

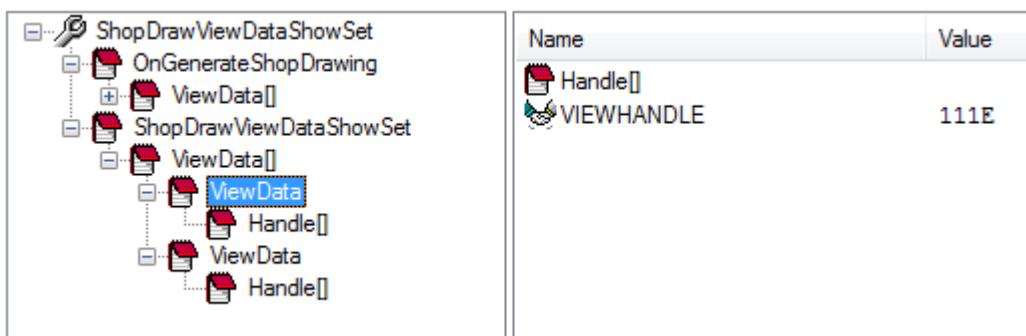
Since hsbCAD2011, build 16.1.3. If the ts1 which is part of the block of the shopdraw layout has registered itself for the event **\_kShopDrawViewDataSet** then it gets executed with the state:

```

if (_bOnGenerateShopDrawing &&
_kExecuteKey==_kShopDrawViewDataSet)

```

The purpose of the execution is to change the showset of a ShopDrawView and possibly its orientation. For each View, a showSetEntities list of entities must be added, as well as a View id. This must be done through **\_Map**. Here you see the **\_Map** that was generated by the Tsl example below:



When this Tsl is added to the block that defines the layout, the Tsl would be inserted selecting a [ShopDrawView](#).

```

Entity ent = getShopDrawView(T("Select the view entity")); //
select ShopDrawView
_Entity.append(ent);

```

During shopdraw generation this ShopDrawView entity can be used to retrieve the [ViewData](#) for that ShopDrawView. The ViewData is given to the Tsl in its **\_Map**. In fact all ViewData of all the ShopDrawViews might be present in the **\_Map**, But the ViewData corresponding to the ShopDrawView that was selected during insert can be found with the following method (error handling stripped).

```

ShopDrawView sv = (ShopDrawView)_Entity[0];
ViewData arViewData[] = ViewData().convertFromSubMap(_Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets,0);
int nIndFound = ViewData().findDataForViewport(arViewData, sv);//
find the ViewData for my view
ViewData vwData = arViewData[nIndFound];

```

As you see, the `_kOnGenerateShopDrawing` and `_kViewDataSets` predefined variables are used to access the map. The `convertFromSubMap` returns a collection of `ViewData` instances.

---

*[Example O-type TSL to be added inside a shopdraw block, illustrating special recalc trigger.*

```

Unit(1, "mm");

//Insert
if ( bOnInsert ){
    PrEntity ssE(T("|Select a set of shopdraw views|"),
ShopDrawView());
    if (ssE.go()) {
        Entity = ssE.set();
    }
    Pt0 = getPoint();
    return;
}

if ( Entity.length() == 0 ) { eraseInstance(); return; }

reportMessage("\\nTsl execution: " + scriptName() + ":" + kExecuteKey
+ " - " + bOnGenerateShopDrawing );

addRecalcTrigger(kShopDrawEvent, kShopDrawViewDataShowSet);
if ( bOnGenerateShopDrawing &&
kExecuteKey==kShopDrawViewDataShowSet )
{
    // interprete the list of ViewData in my Map
    ViewData arViewData[] = ViewData().convertFromSubMap( Map,
kOnGenerateShopDrawing + "\\\" + kViewDataSets,2); // 2 means
verbose

    // use the first entity / ShopDrawView for defining the show set
    // from the define set.
    Entity entView = Entity[0];

    int nIndFound = ViewData().findDataForViewport(arViewData,
entView); // find the ViewData for my view

    Entity entDefine;
    Entity entsShowSet[0];
    if (nIndFound>=0) { // my entView its viewdata is found at index
nIndFound
        ViewData vwData = arViewData[nIndFound];
        Entity ents[] = vwData.showSetDefineEntities();
        entsShowSet = vwData.showSetEntities();
}

```

---

```

        if (ents.length()==1) // in case there is only one entity
            entDefine = ents[0];
    }

    if (!entDefine.bIsValid()) {
        reportMessage("\nError in "+scriptName()+": no define set
entity found.");
    }
    GenBeam gbm = (GenBeam)entDefine;
    if (!gbm.bIsValid()) {
        reportMessage("\nError in "+scriptName()+": define set entity
is not a genbeam.");
    }
    else { // there was one entity, and this one entity is a genbeam

        // append to the existing showset all the tsl's attached to the
beam
        Entity entTools[] = gbm.eToolsConnected();

        for (int e=0; e<entTools.length(); e++) {
            Entity ent = entTools[e];
            if (entsShowSet.find(ent)>=0)
                continue; // entity already in list
            TslInst tsl = (TslInst)ent;
            if (!tsl.bIsValid())
                continue; // entity is not a Tsl

            // now append the tsl entity to the showset
            entsShowSet.append(tsl);
        }

        String strViewData = "ViewData";

        // now loop over all the ShopDrawViews and append a map to _Map
for each
        for (int e=0; e<_Entity.length(); e++)
        {
            ShopDrawView sdv = (ShopDrawView)_Entity[e];
            if (!sdv.bIsValid())
                continue;

            String strViewHandle = sdv.handle();

            // adjust the show set for the view
            Map mpViewData;
            // add new show set
            mpViewData.setEntityArray(entsShowSet, TRUE, "Handle[]",
"showSetEntities", "han");
            // add view identification
            mpViewData.setString("ViewHandle",strViewHandle);
        }
    }
}

```

```

        // now add ViewData[]\\ViewData to _Map
        _Map.appendMap(_kShopDrawViewDataShowSet + "\\\" +
strViewData + "[]" + "\\\" + strViewData,
                    mpViewData); // add to _Map with special key
    }
}

// for debugging export map
_map.writeToDxxFile(_kPathDwg+"\\\"+_kShopDrawViewDataShowSet
+".dxx",FALSE);
}

// do not draw anything during shopdraw generation
if (_bOnGenerateShopDrawing==TRUE) {
    return;
}

// display in normal block mode
Display dp(-1);
dp.textHeight(100);
dp.draw(scriptName(), _Pt0, _XU, _YU, 1.2,1.2,_kDevice );

```

*—end example]*

#### 10.8.4 ShopDrawViewDataShowSet orientation

Since hsbCAD2011, build 16.1.27. If the tsl which is part of the block of the shopdraw layout has registered itself for the event **\_KShopDrawViewDataShowSet** then it gets executed with the state:

```

if (_bOnGenerateShopDrawing &&
_kExecuteKey==_kShopDrawViewDataShowSet)

```

Besides the optional specification of the showset, the tsl can also determine the orientation of the view. This is done through specifying a coordsys consisting of an origin point and vectors that are defined in model space. The ptOrg of the coordsys is the center point of the view. The vecZ is the vector pointing to the viewer, the vecX is the horizontal view vector, the vecY is the vertical up vector for the view.

The tsl can be easily combined for changing the showset if needed. The example below just sets the view orientation. For debugging purposes, a triangle is added to model space, expressing the location horizontal and vertical direction of the view. The shortest edge is pointing upwards.

Data		3D Model	
ShopDrawViewDataCoordSys-ShopDrawVi			
OnGenerateShopDrawing			
ShopDrawViewDataSet			
ViewData			
ViewData			
ViewData			
Name	Value		
VIEWHANDLE	12D9		
13VECWX	0		
13VECXWY	0		
13VECXWZ	1		
13VECYWX	-1		
13VECYWY	0		
13VECYWZ	0		
13VECZWX	0		
13VECZWY	-1		
13VECZWZ	0		
11PTORGWX	387.26129		
11PTORGWY	-140.43144		
11PTORGWZ	1964.3		

[Example O-type TSL to be added inside a shopdraw block, illustrating the manipulation of the view orientation.

```

Unit(1,"mm");

//Insert
if( _bOnInsert ){
    PrEntity ssE(T("|Select a set of shopdraw views|"),
ShopDrawView());
    if( ssE.go() ) {
        _Entity = ssE.set();
    }
    _Pt0 = getPoint();
    return;
}

if( _Entity.length() == 0 ){ eraseInstance(); return; }

reportMessage("\nTsl execution: " +scriptName() + ":" + _kExecuteKey
+ " - " + _bOnGenerateShopDrawing );

addRecalcTrigger(_kShopDrawEvent, _kShopDrawViewDataSet );
if( _bOnGenerateShopDrawing &&
_kExecuteKey==_kShopDrawViewDataSet )
{
    // interprete the list of ViewData in my _Map
    ViewData arViewData[] = ViewData().convertFromSubMap( _Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets,2); // 2 means
verbose

    // use the first entity / ShopDrawView for defining the show set
from the define set.
    Entity entView = _Entity[0];
}

```

```

    int nIndFound = ViewData().findDataForViewport(arViewData,
entView); // find the viewData for my view

    Entity entDefine;
    if (nIndFound>=0) { // my entView its viewdata is found at index
nIndFound
        ViewData vwData = arViewData[nIndFound];
        Entity ents[] = vwData.showSetDefineEntities();
        if (ents.length()==1) // in case there is only one entity
            entDefine = ents[0];
    }

    if (!entDefine.bIsValid()) {
        reportMessage("\nError in "+scriptName()+" : no define set
entity found.");
    }
    GenBeam gbm = (GenBeam)entDefine;
    if (!gbm.bIsValid()) {
        reportMessage("\nError in "+scriptName()+" : define set entity
is not a genbeam.");
    }
    else { // there was one entity, and this one entity is a genbeam

        String strViewData = "ViewData";

        // now loop over all the ShopDrawViews and append a map to _Map
for each
        for (int e=0; e<_Entity.length(); e++)
        {
            ShopDrawView sdv = (ShopDrawView)_Entity[e];
            if (!sdv.bIsValid())
                continue;

            String strViewHandle = sdv.handle();

            // Compose map for view to adjust the coordSys set for the
view
            Map mpViewData;
            // add view identification
            mpViewData.setString("ViewHandle",strViewHandle);

            // add a modified orientation of the view window
            Vector3d vecXW = gbm.vecX();
            Vector3d vecYW = gbm.vecY();
            Vector3d vecZW = gbm.vecZ();
            mpViewData.setVector3d("vecXW",vecXW);
            mpViewData.setVector3d("vecYW",vecYW);
            mpViewData.setVector3d("vecZW",vecZW);
            mpViewData.setDouble("scale",0.1); // change scale: added in
17.1.10
        }
    }
}

```

```

// add a modified location of the view window
Point3d ptW = gbm.ptCen() + 0.5 * gbm.dL() * vecXW;
mpViewData.setPoint3d("ptOrgW", ptW);

// add a pline to the database to visualize where the view
should be looking at
PLine pl(ptW + U(200) * vecYW, ptW, ptW + U(400) * vecXW);
EntPLine entPLine;
entPLine.dbCreate(pl);
entPLine.setColor(4);

// now add ViewData[]\\ViewData to _Map
_Map.appendMap(_kShopDrawViewDataShowSet + "\\\" +
strViewData + "[]" + "\\" + strViewData,
mpViewData); // add to _Map with special key
}

}

// for debugging export map
_Map.writeToDxxFile(_kPathDwg + "\\\" + scriptName() + "-" +
_kShopDrawViewDataShowSet + ".dxx", FALSE);
}

// do not draw anything during shopdraw generation
if (_bOnGenerateShopDrawing == TRUE) {
    return;
}

// display in normal block mode
Display dp(-1);
dp.textHeight(U(100));
dp.draw(scriptName(), _Pt0, _XU, _YU, 1.2, 1.2, _kDevice );

```

**—end example]**

[Example O-type TSL to be added inside a shopdraw block and to be attached to shop draw viewport entities, illustrating the manipulation of the view orientation. The orientation follows is set by the "view direction" property of the viewport, but corrected by the flipAxisToOptimalGridOrientation.

```

/// <summary Lang=en>
/// When this tsl is part of the block that defines a layout for a
shopdrawing, and it is attached to one or more
/// View placeholders, it will modify the orientation of these view
placeholders to be aligned with the grid in the drawing.
/// </summary>

/// <insert Lang=en>
/// This tsl needs to be attached to one or more View placeholders in
a block for a shop draw layout.

```

```

/// </insert>

/// <remark Lang=en>
/// * If there is no grid in the drawing, this is a do nothing tsl.
/// </remark>

/// <version value="1.0" date="1nov10" author="kr">initial
version</version>
/// <version value="1.1" date="12jan11" author="kr">bugfix for beam
vecX aligned with view Y or view Z</version>

Unit(1,"mm");
int bInDebug = FALSE;

//Insert
if( _bOnInsert ){
    PrEntity ssE(T("|Select a set of shopdraw views|"),
ShopDrawView());
    if (ssE.go()) {
        _Entity = ssE.set();
    }
    _Pt0 = getPoint();
    return;
}

if( _Entity.length() == 0 ){ eraseInstance(); return; }

String strChangeEntity = T("|Add additional shopdraw view|");
addRecalcTrigger(_kContext, strChangeEntity );
if (_bOnRecalc && _kExecuteKey==strChangeEntity) {
    _Entity.append(getEntity());
}

if (bInDebug)
    reportMessage("\nTsl execution: " +scriptName() + ":" +
_kExecuteKey + " - " + _bOnGenerateShopDrawing );

addRecalcTrigger(_kShopDrawEvent, _kShopDrawViewDataShowSet );
if (_bOnGenerateShopDrawing &&
_kExecuteKey==_kShopDrawViewDataShowSet )
{
    // for debugging export map
    if (bInDebug)
        _Map.writeToDxxFile(_kPathDwg+"\\"+scriptName() + "-" +
_kShopDrawViewDataShowSet +"_Pre.dxx",FALSE);

    // interprete the list of ViewData in my _Map
    ViewData arViewData[] = ViewData() .convertFromSubMap( _Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets,2); // 2 means
verbose

```

```

// loop over all the entities. Each of them should be a
ShopDrawView.
for (int indexEnt = 0; indexEnt<_Entity.length(); indexEnt++) {

    ShopDrawView sdv = (ShopDrawView)_Entity[indexEnt];
    if (!sdv.bIsValid())
        continue;

    int nIndFound = ViewData().findDataForViewport(arViewData,
    sdv); // find the ViewData for my view
    if (nIndFound<0) {
        reportMessage("\nError in "+scriptName()+" : entity attached
is not a ShopDrawView.");
        continue;
    }

    Entity entDefine;
    ViewData vwData = arViewData[nIndFound]; // my entView its
viewdata is found at index nIndFound
    Entity ents[] = vwData.showSetDefineEntities();
    if (ents.length()==1) // in case there is only one entity
        entDefine = ents[0];
    CoordSys csMs2Ps = vwData.coordSys();
    CoordSys csPs2Ms = csMs2Ps;
    csPs2Ms.invert();

    if (!entDefine.bIsValid()) {
        reportMessage("\nError in "+scriptName()+" : no define set
entity found.");
        continue;
    }
    GenBeam gbm = (GenBeam)entDefine;
    if (!gbm.bIsValid()) {
        reportMessage("\nError in "+scriptName()+" : define set
entity is not a genbeam.");
        continue;
    }

    Vector3d vecXW = _XW;
    Vector3d vecYW = _YW;
    Vector3d vecZW = _ZW;
    Point3d ptW = vwData.ptCenPS();

    // transform from PS to model
    // these are the vectors that are currently in action, if
nothing is changed
    vecXW.transformBy(csPs2Ms);
    vecYW.transformBy(csPs2Ms);
    vecZW.transformBy(csPs2Ms);
    ptW.transformBy(csPs2Ms);
}

```

```

CoordSys csBm = gbm.coordSys();
Grid grd = gbm.grid();
if (!grd.bIsValid())
    continue;
CoordSys csBO = grd.flipAxisToOptimalGridOrientation(csBm);

// check if the beam xas is along view x axis
Vector3d vecXBmView = csBm.vecX();
if (vecXW.isParallelTo(csBm.vecX())) {
    if (vecXW.isCodirectionalTo(csBm.vecX()))
        vecXW = csBO.vecX(); // instead of taking csBm.vecX()
take csBO.vecX
else
    vecXW = -csBO.vecX();
}
else if (vecXW.isParallelTo(csBm.vecY())) {
    if (vecXW.isCodirectionalTo(csBm.vecY()))
        vecXW = csBO.vecY();
else
    vecXW = -csBO.vecY();
}
else if (vecXW.isParallelTo(csBm.vecZ())) {
    if (vecXW.isCodirectionalTo(csBm.vecZ()))
        vecXW = csBO.vecZ();
else
    vecXW = -csBO.vecZ();
}
if (vecZW.isParallelTo(csBm.vecX())) {
    if (vecZW.isCodirectionalTo(csBm.vecX()))
        vecZW = csBO.vecX(); // instead of taking csBm.vecX()
take csBO.vecX
else
    vecZW = -csBO.vecX();
}
else if (vecZW.isParallelTo(csBm.vecY())) {
    if (vecZW.isCodirectionalTo(csBm.vecY()))
        vecZW = csBO.vecY(); // instead of taking csBm.vecY()
take csBO.vecY
else
    vecZW = -csBO.vecY();
}
else if (vecZW.isParallelTo(csBm.vecZ())) {
    if (vecZW.isCodirectionalTo(csBm.vecZ()))
        vecZW = csBO.vecZ(); // instead of taking csBm.vecZ()
take csBO.vecZ
else
    vecZW = -csBO.vecZ();
}

```

```

    vecYW = vecZW.crossProduct(vecXW);
    vecXW.normalize();
    vecYW.normalize();
    vecZW.normalize();

    // Compose map for view to adjust the coordSys set for the view
Map mpViewData;

    // add view identification
String strViewHandle = sdv.handle();
mpViewData.setString("ViewHandle",strViewHandle);

    // add a modified orientation of the view window
mpViewData.setVector3d("vecXW",vecXW);
mpViewData.setVector3d("vecYW",vecYW);
mpViewData.setVector3d("vecZW",vecZW);
    // add a modified location of the view window
//Point3d ptW = gbm.ptCenSolid();
mpViewData.setPoint3d("ptOrgW",ptW);

    // add a pline to the database to visualize where the view
should be looking at
//PLine pl(ptW+U(200)*vecYW, ptW, ptW+U(400)*vecXW);
//EntPLine entPLine;
//entPLine.dbCreate(pl);
//entPLine.setColor(4);

    // now add ViewData[]\\ViewData to _Map
    // append a map to _Map for each ShopDrawView
String strViewData = "ViewData";
    _Map.appendMap(_kShopDrawViewDataShowSet + "\\" + strViewData +
"[]" + "\\" + strViewData,
    mpViewData); // add to _Map with special key

}

// for debugging export map
if (bInDebug)
    _Map.writeToDxxFile(_kPathDwg+"\\"+scriptName() + "-" +
_kShopDrawViewDataShowSet +"_Post.dxx",FALSE);
}

// do not draw anything during shopdraw generation
if (_bOnGenerateShopDrawing==TRUE) {
    return;
}

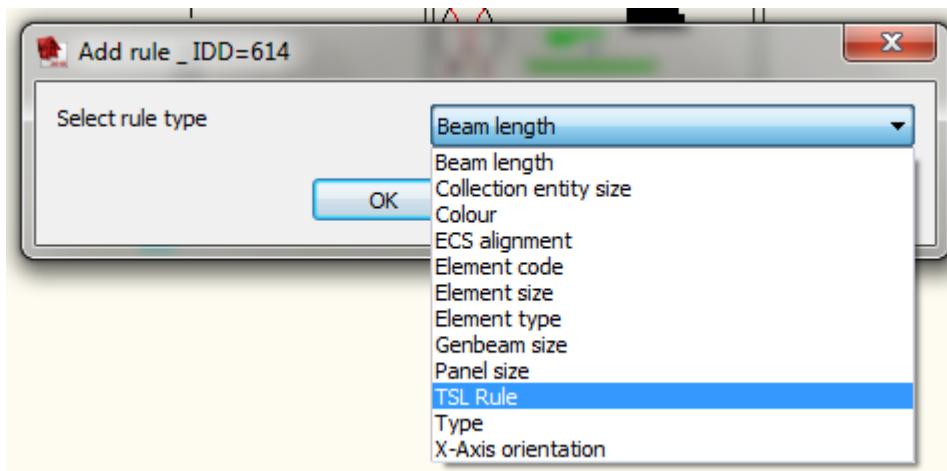
// display in normal block mode
Display dp(-1);
dp.textHeight(U(100));
dp.draw(scriptName(), _Pt0, _XU, _YU, 1.2,1.2,_kDevice );

```

—end example]

## 10.9 ShopDrawTemplateRule

In a shopdraw style there is a default layout, and a set of layout overwrites. Each overwrite has a set of rules attached. If the rule allows it, the overwrite is taken. For different shopdraw entities, there are different rules. But there is also a rule type that can be implemented by Tsl the shopdraw designer. The rule is called "TSL Rule".



When a tsl is executed for such a purpose, the aim is to either accept or reject the current entity for this layout. The acceptance is added as an int to the \_Map. Here you see the \_Map that was generated by the Tsl example below.

Name	Value
70BACCEPT	1

The submap inside \_Map can be set by using the following predefined:

[\\_kShopDrawTemplateRule](#)

[Example O-type TSL to be used as Tsl Rule accepting entities with a certain color.

```

String strKey = "nRejectColor"; // key in _Map

// when the ellipses button is pressed in the "Define Tsl to be
// inserted" dialog, the _bOnMapIO is TRUE.
if (_bOnMapIO) {

    // define property
}

```

```

PropInt pColor(0,1,T("Entity color to reject"));
pColor.setDescription(T("Entities that have this color will be
rejected by this template rule."));

if (_Map.hasInt(strKey)) {
    pColor.setInt(_Map.getInt(strKey)); // set the property value from
the _Map contents
}

// show the dialog to the user
showDialog("---"); // use "___" such that the set values are used,
and not the last dialog values

    _Map.setInt(strKey,pColor); // interpret the properties, and fill
_map

// do not call eraseInstance(), otherwise no _Map changes are
accepted
    return;
}

int nRejectColor = 1; // default red
if (_Map.hasInt(strKey)) {
    nRejectColor = _Map.getInt(strKey); // set the property value from
the _Map contents
}

int bInDebug = TRUE;
if (bInDebug) reportMessage("\n"+scriptName()+" : ");

if (_Entity.length()==0) {
    if (bInDebug) reportMessage("No entities in selection set.");
    return; // do not do anything
}

// assume one relevant entity in define set
Entity ent = _Entity[0];

int bAccept = FALSE;

// accept the beam if the color is not red.
int nColor = ent.color();
if (nColor!=nRejectColor)
    bAccept = TRUE;

// add the accept value to the _Map
_map.setInt(_kShopDrawTemplateRule + "\\bAccept", bAccept);

// for debugging export map
_map.writeToDxxFile(_kPathDwg+"\\"+_kShopDrawTemplateRule + ".dxx");

```

```
if (bInDebug) {
    if (bAccept)
        reportMessage("_Entity[0] accepted for rejectColor:
"+nRejectColor);
    else
        reportMessage("_Entity[0] rejected for rejectColor:
"+nRejectColor);
}
```

—*end example*]

---

**Part**

**XI**

## 11 DimRequests

### 11.1 DimRequest

The DimRequest type assists in the description of shop drawing ts1's. Shop drawing ts1's are called during the execution of the shop drawing command. The DimRequest proposes a dimensioning primitive to the shop drawing engine. The request is analysed and if possible converted into an actual dimension primitive. A dimension primitive is for instance an angular dimension at a certain location, with a certain dimension style,...

DimRequest instances or DimRequest derived instances need to be added to the shop drawing framework by calling the global function:

```
void addDimRequest(DimRequest dimRequest);
```

---

```
class DimRequest {

    DimRequest(Map map); // constructor with a Map, for internal use only

    int blsValid() const; // check if the casting was valid

    void addAllowedView(Vector3d vecDir); // add a view direction
    void addAllowedView(Vector3d vecDir, Vector3d vecUp); // A specific up direction of the
    // view is specified.
    void addAllowedView(Vector3d vecDir, int bAlsoReverseDirection); // if
    // bAlsoReverseDirection is TRUE, then both vecDir and -vecDir are added. If
    // bAlsoReverseDirection is FALSE, then only the vecDir is added.
    void addAllowedView(Vector3d vecDir, Vector3d vecUp, int bAlsoReverseDirection); // if
    // bAlsoReverseDirection is TRUE, then both vecDir and -vecDir are added, both with
    // the same vecUp direction.

    void visualize(int indColor = -1) const;
    void vis(int indColor = -1) const;

    void setStereotype(String strStereotype); // the stereotype determines the grouping of the
    // points in different dim lines.

    static void addInternalSet(Entity ent, String strSet, String strStereotype, Map mpParams); //
    // added in (hsbCAD13.3.35 and hsbCAD14.0.11)

};
```

---

```
static void addInternalSet(Entity ent, String strSet, String strStereotype, Map mpParams); //
// added in (hsbCAD13.3.35 and hsbCAD14.0.11)
```

---

If the entity is of type Sip, one could add the internal generated dimrequests. To add such automatic dimrequests, one has to provide a number of parameters. This is done through composing a map mpParams.

Here is the list of supported Sets as well as the Key name for mpParams as well as description of type needed.

```
// FoamRecessDepth
// Cutbacks
    PREFIX - [String] - prefix to use for dimensions
// PanelShape
    OnlyVertAndHorizontal - [int (TRUE or FALSE)] - Only dimension vertical or
    horizontal segs
// EdgeDetailCodes
    OverrideWithRecessDepth - [int (TRUE or FALSE)] - Use Recess Depth instead
    of edge detail code
// DimLumberInstall
    UseNameAsPrefix - [int (TRUE or FALSE)] - Use lumber name as dimension
    prefix
// Wirechases
    ExcludeVerticalDims - [int (TRUE or FALSE)] - Do not draw vertical wirechases
// EmbeddedLumber
// DimensionWirechases
    ExcludeVerticalDims - [int (TRUE or FALSE)] - Do not dimension vertical
    wirechases
    Prefix - [STRING] - dimension Prefix
// HatchComponents
    LumberVisible - [int (TRUE or FALSE)] - tells the component hatcher whether or
    not lumber is being shown in the multipage
```

---

[Example E-type that builds a set of DimRequests for a Beam. If type of Tsl is E-type, the tsl can be added also in the drawing.

```
Unit(1, "mm");

Beam entBeam= _Beam0; // would be valid if added to a beam in the
drawing
if (!entBeam.bIsValid() && _Entity.length()>0)
    entBeam= (Beam)_Entity[0]; // when the shopdraw engine calls this
Tsl, the _Entity array contains the Beam

if (!entBeam.bIsValid()) {
    reportMessage("\n"+scriptName() +": No beam found. Instance
erased.");
    eraseInstance();
    return;
}
```

```

// The parent key which is used in definition of a DimRequest is used
// to group all DimRequests.
String strParentKey = String("Dim "+scriptName());
reportMessage("\n"+scriptName() +": DimRequests with key:
"+strParentKey);

Vector3d vecBmX = entBeam.vecX();
Vector3d vecBmY = entBeam.vecY();
Vector3d vecBmZ = entBeam.vecZ();
Point3d ptBmCenSolid = entBeam.ptCenSolid();
double dLenSolid = entBeam.solidLength();

int nDebugColorCounter = 1;

// Add the length dimensions.
// For a perpendicular cut, there can be in total 8 possible dimlines
in which
// this cut contributes one point. Given a certain view direction, it
is either the top point or the bottom point
// of the cut that contributes to a parallel top or bottom dimline.
The two points are also valid when
// looking from the -vecView.
{
    Vector3d arVecView[] = {vecBmY, vecBmZ};
    for (int v=0; v<arVecView.length(); v++) {

        Vector3d vecView = arVecView[v];
        Vector3d vecLineDir = vecBmX;

        for (int p=0; p<2; p++) { // 0 for top point, 1 for bottom
point
            for (int lr=0; lr<2; lr++) { // 0 for left beam point, 1 for
right point

                Vector3d vecPerp= vecView.crossProduct(vecLineDir);
                if (p==1) vecPerp= -vecPerp;
                vecPerp.normalize();

                // calculate the dimension point
                Point3d ptLR = ptBmCenSolid + 0.5*dLenSolid*vecBmX;
                if (lr==1) ptLR = ptBmCenSolid - 0.5*dLenSolid*vecBmX;

                Point3d ptDim = ptLR + 0.5*entBeam.dD(vecPerp)*vecPerp;

                // compose a DimRequest
                DimRequestPoint dr(strParentKey, ptDim, vecLineDir,
vecPerp);
                dr.setStereotype("*"); // add to any chain dimension in the
same direction
                dr.addAllowedView(vecView, TRUE);
                dr.vis(nDebugColorCounter++); // visualize for debugging
                addDimRequest(dr); // append to shop draw engine collector
            }
        }
    }
}

```

```

        } }

    }

// Add the width and height dimensions.
{
    Vector3d arVecOffset[] = {vecBmY, vecBmZ, -vecBmY, -vecBmZ};
    for (int v=0; v<arVecOffset.length(); v++) {
        Vector3d vecOffsetDir = arVecOffset[v];
        double dOffsetDir = entBeam.dD(vecOffsetDir);
        Vector3d vec12 = vecBmX.crossProduct(vecOffsetDir);
        vec12.normalize();
        double d12 = entBeam.dD(vec12);

        Point3d pt1 = ptBmCenSolid + 0.5*dOffsetDir*vecOffsetDir +
        0.5*d12*vec12;
        Point3d pt2 = ptBmCenSolid + 0.5*dOffsetDir*vecOffsetDir -
        0.5*d12*vec12;

        // compose a DimRequest
        DimRequestLinear dr(strParentKey, pt1, pt2, vecOffsetDir);
        dr.addAllowedView(vecBmX, TRUE);
        dr.vis(nDebugColorCounter++); // visualize for debugging
        addDimRequest(dr); // append to shop draw engine collector
    }
}

// add a pitch dimension
if ((!vecBmX.isPerpendicularTo(_ZW)) && (!vecBmX.isParallelTo(_ZW)))
{
    Vector3d vecUp = entBeam.vecD(_ZW); // vecUp is the vecY or vecZ
    most aligned with _ZW
    if (vecBmX.dotProduct(_ZW)<0) vecBmX = -vecBmX; // make sure the
    vecBmX is in the upper dir

    double dDotZ = vecBmX.dotProduct(_ZW);
    Vector3d vecDirection = vecBmX - dDotZ*_ZW;
    vecDirection.normalize();
    double dDotX = vecBmX.dotProduct(vecDirection);

    double dScale = U(250,10);
    double dDeltaX = dScale*dDotX;
    double dDeltaY = dScale*dDotZ;
    if (abs(dDotX)>U(0.1)) { // normal case
        dDeltaX = dScale;
        dDeltaY = dScale*dDotZ/dDotX;
    }
}

```

```

Point3d ptLocation = ptBmCenSolid - 0.5*dLenSolid*vecDirection +
0.5*dLenSolid*dDotZ*_ZW;

DimRequestPitch dr(strParentKey, ptLocation, vecDirection,
dDeltaX, dDeltaY);
dr.addAllowedView(vecBmX.crossProduct(vecUp), _ZW, TRUE);
dr.vis(nDebugColorCounter++); // visualize for debugging
addDimRequest(dr); // append to shop draw engine collector
}

```

*—end example*

## 11.2 DimRequestPoint

The DimRequestPoint type represents a specific DimRequest that expresses a required dimensioning point along a dimension line. By the shop drawing engine, the point will be joined together with other points of the same stereotype into one dimension line. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestPoint. However when the casting is not allowed, the resulting DimRequestPoint will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestPoint is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```

class DimRequestPoint : DimRequest { // derived from DimRequest

    DimRequestPoint(String strParentKey, Point3d ptRef, Vector3d vecDimLineDir, Vector3d
                    vecPerpDimLineDir);

    void setIsChainDimReferencePoint(int bVal); // set to TRUE or FALSE, default is false.

    void setNodeTextDelta(String strText); // sets the text to appear in between the previous
                                         // node and this node. Use "<>" as placeholder for the actual dimension value.
    void setNodeTextCumm(String strText); // sets the text to appear on the node of the chain
                                         // dimension. Use "<>" as placeholder for the actual dimension value.

};

```

## 11.3 DimRequestLinear

The DimRequestLinear type represents a specific DimRequest that expresses a linear dimension. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestLinear. However when the casting is not allowed, the resulting DimRequestLinear will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestLinear is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestLinear : DimRequest { // derived from DimRequest

    DimRequestLinear(String strParentKey, Point3d pt1, Point3d pt2, Vector3d
        vecPerpDimLineDir);

    void setMinimumOffsetFromDimLine(double dOffset);
};
```

## 11.4 DimRequestText

The DimRequestText type represents a specific DimRequest that expresses a normal text. The text can be multiline. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestText. However when the casting is not allowed, the resulting DimRequestText will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestText is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestText : DimRequest { // derived from DimRequest

    DimRequestText(String strParentKey, String strText, Point3d ptLocation, Vector3d
        vecTextDirection, Vector3d vecTextOffsetDirection);

    void addLine(String strText);
    void setShowLeaderLine(int bVal); // default value is TRUE (added hsbCAD2009+ 14.1.2)
};
```

---

## 11.5 DimRequestAngular

The DimRequestAngular type represents a specific DimRequest that expresses an angular dimension. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestAngular. However when the casting is not allowed, the resulting DimRequestAngular will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestAngular is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestAngular : DimRequest { // derived from DimRequest

    DimRequestAngular(String strParentKey, Point3d ptCenter, Point3d ptXLine1, Point3d
        ptXLine2, Point3d ptOnArc);

}
```

## 11.6 DimRequestPitch

The DimRequestPitch type represents a specific DimRequest that expresses a pitch. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestPitch. However when the casting is not allowed, the resulting DimRequestPitch will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestPitch is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestPitch : DimRequest { // derived from DimRequest

    DimRequestPitch(String strParentKey, Point3d ptLocation, Vector3d vecDirection, double
        dDeltaX, double dDeltaY);

    // following 2 methods added since in 23.8.99, 24.1.92 and 25.1.76
    DimRequestPitch(String strParentKey, Point3d ptLocation, Vector3d vecDirection, double
        dDeltaX, double dDeltaY, String strDimStyleName);
    DimRequestPitch(String strParentKey, Point3d ptLocation, Vector3d vecDirection, double
        dDeltaX, double dDeltaY, String strDimStyleName, String strDeltaTextX, String
        strDeltaTextY);

    void setDimStyle(String strStyleName); // added 23.8.99, 24.1.92 and 25.1.76
    void setDeltaTextX(String strDeltaTextX); // empty means no overwrite (added 23.8.99, 24.1.92
        and 25.1.76)
    void setDeltaTextY(String strDeltaTextY); // empty means no overwrite (added 23.8.99, 24.1.92
        and 25.1.76)
```

---

```
};
```

## 11.7 DimRequestObholz

The DimRequestObholz type represents a specific DimRequest that allows to dimension a birds mouth type of beamcut. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestObholz. However when the casting is not allowed, the resulting DimRequestObholz will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestObholz is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestObholz : DimRequest { // derived from DimRequest

    DimRequestObholz(String strParentKey, Point3d ptBirdsMouthTop, Point3d
                    ptBirdsMouthBottom1, Point3d ptBirdsMouthBottom2, Point3d ptPerpendicular,
                    Point3d ptVertical);

    void setUseVerticalDimensions(int bVal); // default value is TRUE (added hsbCAD2010 15.0.0)
};
```

## 11.8 DimRequestHeightLevel

The DimRequestHeightLevel type represents a specific DimRequest that expresses a height level indication. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestHeightLevel. However when the casting is not allowed, the resulting DimRequestHeightLevel will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestHeightLevel is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestHeightLevel : DimRequest { // derived from DimRequest

    DimRequestHeightLevel(String strParentKey, Point3d ptRef, Vector3d
                           vecTextPlacementSide);

};
```

## 11.9 DimRequestRadial

The DimRequestRadial type represents a specific DimRequest that expresses a linear dimension. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestRadial. However when the casting is not allowed, the resulting DimRequestRadial will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestRadial is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestRadial : DimRequest { // derived from DimRequest
    DimRequestRadial(String strParentKey, Point3d ptCenter, Point3d ptChord);
}
```

## 11.10 DimRequestMultiViewLine

The DimRequestMultiViewLine type represents a specific DimRequest that expresses a reference line from one view to another view. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestMultiViewLine. However when the casting is not allowed, the resulting DimRequestMultiViewLine will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestMultiViewLine is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestMultiViewLine : DimRequest { // derived from DimRequest
    DimRequestMultiViewLine(String strParentKey, Point3d ptRef);

    void addToView(Vector3d vecDir); // add a view direction
    void addToView(Vector3d vecDir, int bAlsoReverseDirection); // if bAlsoReverseDirection is
        // TRUE, then both vecDir and -vecDir are added. If bAlsoReverseDirection is FALSE,
        // then only the vecDir is added.

    void addFromView(Vector3d vecDir); // add a view direction
    void addFromView(Vector3d vecDir, int bAlsoReverseDirection); // if bAlsoReverseDirection is
        // TRUE, then both vecDir and -vecDir are added. If bAlsoReverseDirection is FALSE,
        // then only the vecDir is added.
```

---

```
};
```

## 11.11 DimRequestPLine

The DimRequestPLine type represents a specific DimRequest that expresses Pline. The PLine is drawn with the specified color. Similar to the other DimRequests, the PLine is specified in 3d, on the entity being dimensioned. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestPLine. However when the casting is not allowed, the resulting DimRequestPLine will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestPLine is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

---

```
class DimRequestPLine : DimRequest { // derived from DimRequest
    DimRequestPLine(String strParentKey, PLine plShape, int nColor);
    void setLineType(String strLineTypeName); // there is a predefine \_LineTypes
};
```

## 11.12 DimRequestChain

The DimRequestChain type represents a specific DimRequest that expresses chain dimension. Similar to the other DimRequests, the chain dimension is specified in 3d, on the entity being dimensioned. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestChain. However when the casting is not allowed, the resulting DimRequestChain will become invalid. This can be checked with the `blsValid()` function of DimRequest. Because DimRequestChain is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

The chain dimension typically consists of multiple point. For each of the points, there is a dimension from the previous node (also called delta, middle, in between dimension), and a dimension from the reference point of the chain (also called end dimension or cumulative dimension).

The first point added is the reference point of the chain.

If the dimension text needs to be placed perpendicular to the dimension line, the `nDisplayModusMiddle` and `nDisplayModusEnd` need to be used:

- [\\_kDimNone](#) the value is not displayed
  - [\\_kDimPar](#) the value is displayed parallel to the dim line
  - [\\_kDimPerp](#) the value is displayed perpendicular to the dim line
-

```

class DimRequestChain : DimRequest { // derived from DimRequest

    DimRequestChain(String strParentKey, Vector3d vecDimLineDir, Vector3d
                      vecPerpDimLineDir, int nDisplayModusMiddle, int nDisplayModusEnd);

    void addNode(Point3d pt);
    void addNode(Point3d pt, String txtMiddle, String txtEnd); // sets the text to appear in this
                                                               dimension node. Use "<>" as placeholder for the actual dimension value.

    void setDeltaOnTop(int bOnTop); // set location of delta dimension, default is on top, TRUE
    void setDimStyle(String strDimStyle); // sets the dim style

    void setMinimumOffsetFromDimLine(double dOffset);
}

```

## 11.13 DimRequestHatch

The DimRequestHatch type represents a specific DimRequest that expresses Pline. The PLine is drawn with the specified color. Similar to the other DimRequests, the PLine is specified in 3d, on the entity being dimensioned. The type is derived from [DimRequest](#).

A DimRequest can be casted into a DimRequestHatch. However when the casting is not allowed, the resulting DimRequestHatch will become invalid. This can be checked with the `bIsValid()` function of DimRequest. Because DimRequestHatch is derived from DimRequest, it inherits all the member functions of DimRequest as well. It can also be casted into an DimRequest.

In the DimRequest there is a method `setStereotype` which is applicable for

---

```

class DimRequestHatch : DimRequest { // derived from DimRequest

    DimRequestHatch(String strParentKey, PlaneProfile plShape); // see PlaneProfile

    void setHatch(Hatch hatch, int nColor); // overwrite the Hatch specified in the setStereotype of
                                                DimRequest.
}

```

---

*[Example E-type that builds a set of DimRequests for a Beam. If type of Tsl is E-type, the tsl can be added also in the drawing.]*

```

Unit(1, "mm");

Beam entBeam= Beam0; // would be valid if added to a beam in the
                           drawing
if (!entBeam.bIsValid() && Entity.length()>0)

```

---

```

entBeam= (Beam) _Entity[0]; // when the shopdraw engine calls this
Tsl, the _Entity array contains the Beam

if (!entBeam.bIsValid()) {
    reportMessage("\n"+scriptName() +": No beam found. Instance
erased.");
    eraseInstance();
    return;
}

// The parent key which is used in definition of a DimRequest is used
to group all DimRequests.
String strParentKey = String("Dim "+scriptName());
reportMessage("\n"+scriptName() +": DimRequests with key:
"+strParentKey);

Vector3d vecBmY = entBeam.vecY();
Point3d ptBmCenSolid = entBeam.ptCenSolid();
double dWSolid = entBeam.solidWidth();

int nDebugColorCounter = 1;

// Add DimRequestHatch with stereotype on vecBmY
{
    Vector3d vecView = vecBmY;

    Body bdBeam = entBeam.realBody();
    PlaneProfile prof =
bdBeam.extractContactFaceInPlane
(Plane(ptBmCenSolid+0.5*dWSolid*vecView , vecView ),U(1));

    DimRequestHatch dr(strParentKey, prof);
    dr.setStereotype("kr");
    dr.addAllowedView(vecView, FALSE);
    dr.vis(nDebugColorCounter++); // visualize for debugging
    addDimRequest(dr); // append to shop draw engine collector
}

// Add DimRequestHatch with specified Hatch -on vecBmY
{
    Vector3d vecView = -vecBmY;

    Body bdBeam = entBeam.realBody();
    PlaneProfile prof =
bdBeam.extractContactFaceInPlane
(Plane(ptBmCenSolid+0.5*dWSolid*vecView , vecView ),U(1));

    Hatch hatchNet("NET",U(10));
    hatchNet.setAngle(30); // 30 degrees

    DimRequestHatch dr(strParentKey, prof);
    dr.setStereotype("*");
}

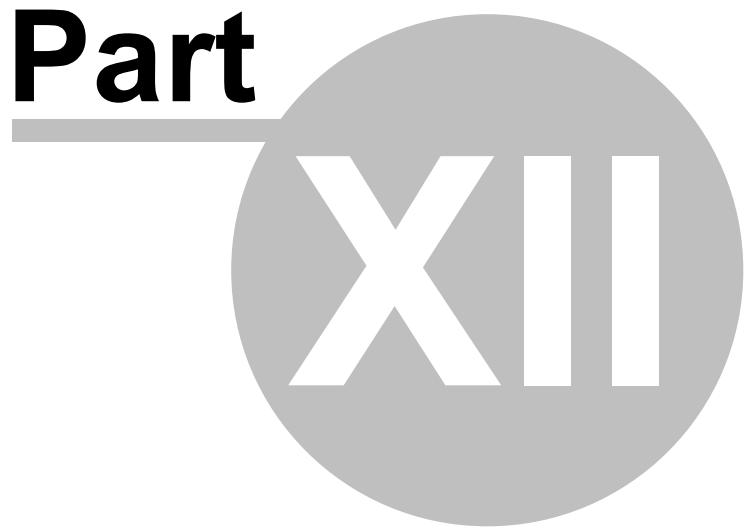
```

```
    dr.addAllowedView(vecView, FALSE);
    dr.setHatch(hatchNet,5);
    dr.vis(nDebugColorCounter++); // visualize for debugging
    addDimRequest(dr); // append to shop draw engine collector
}
```

*—end example]*

---

**Part**



## 12 Tools on elements

### 12.1 ElemNail

The ElemNail tool defines a nailing line on an element. With this ElemNail tool one can either make a [NailLine](#) entity, or apply the tool to the element from inside the TSL.

On [Element](#) also find `getToolsOfTypeNail`. // added since 23.5.9

See [General](#) for member functions applicable to all tools.

---

```
class ElemNail {
    ElemNail(int nZoneIndex, PLine plNail, double dSpacing, int nToolingIndex);
    ElemNail(int nZoneIndex, Point3d ptFrom, Point3d ptTo, double dSpacing, int nToolingIndex);

    void setToolIndex(int nVal); // added since 23.5.9
    void setZoneIndex(int nVal); // added since 23.5.9
    void setSpacing(double dVal); // added since 23.5.9
    void setPIShape(PLine plShape); // added since 23.5.9

    int toolIndex() const; // added since 23.5.9
    int zoneIndex() const; // added since 23.5.9
    double spacing() const; // added since 23.5.9
    PLine plShape() const; // added since 23.5.9
};
```

---

The constructor has the following formats:

```
ElemNail(int nZoneIndex, PLine plNailing, double dSpacing, int nToolingIndex);
ElemNail(int nZoneIndex, Point3d ptFrom, Point3d ptTo, double dSpacing, int nToolingIndex);
```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
plNailing:	polyline defining the location of the operation
dSpacing:	maximum spacing between the nails
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used
ptFrom:	starting point of line
ptTo:	ending point of line

---

*[Example O-type with insert implemented:*

```
Unit(1,"mm"); // U(x) in this script means x mm
```

---

```

PropInt nZoneIndex(0,1,T("Zone index"));
PropDouble dSpacing(0,U(10),T("Spacing"));
PropInt nToolingIndex(1,0,T("Tooling index"));

if (_bOnDebug) {
    reportMessage("\nZone index:" + nZoneIndex);
    reportMessage("\nSpacing:" + dSpacing);
    reportMessage("\nTooling index:" + nToolingIndex);
}

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    _PtG.append(getPoint());
}

// check if the instance was added to another element
if (_Element.length()>1) {
    for (int i=1; i<_Element.length(); i++) {
        if (_Element[i].blsValid()) {
            _Element[0] = _Element[i]; // take last element as THE reference
            _Element0 = _Element[0];
            _Element[i] = Element(); // set to no reference
        }
    }
}

// check if the _Element0 was added to the instance
if (_Element0.blsValid()) {

    int nZoneInc = (nZoneIndex>=0)?1:-1;
    ElemZone ez = _Element0.zone(nZoneIndex+nZoneInc); // go on top surface of zone:
    (nZoneIndex+1)
    CoordSys cs = ez.coordSys(_Pt0);
    _Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point on the zone

    // also move the grippoint to the surface of the zone
    Vector3d vec = _PtG[0]-_Pt0;
    double dxv = vec.dotProduct(cs.vecX());
    double dyv = vec.dotProduct(cs.vecY());
    _PtG[0] = _Pt0 + dxv*cs.vecX() + dyv*cs.vecY();

    PLine nailline(cs.vecZ()); // sets the normal of the polyline
    nailline.addVertex(_Pt0);
    nailline.addVertex(_PtG[0]);
    ElemNail nailtool(nZoneIndex,nailline,dSpacing,nToolingIndex);
    _Element0.addTool(nailtool);

    // add the script entity to the element group itself
    assignToElementGroup(_Element0,TRUE,0,'E');

    Vector3d vecX = cs.vecX();
}

```

```
Vector3d vecY = cs.vecY();
Vector3d vecZ = cs.vecZ();
vecX.vis(_Pt0);
vecY.vis(_Pt0);
vecZ.vis(_Pt0);
}
```

—end example]

## 12.2 ElemNoNail

The ElemNoNail tool defines a no-nailing area on an element. The constructor has the following formats:

```
ElemNoNail name(int nZoneIndex, PLine plNailing);
nZoneIndex: zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
plNailing: polyline that bounds the no-nailing zone
```

The plNailing is automatically closed, if necessary.

The tooltype is an int, use the following predefines:

```
_kTTAllTools = -1, _KTTNail = 0, _KTTSaw, _KTTMill, _KTTDrill, _KTTMarker, _KTTItem,  
_KTTNailCluster
```

On [Element](#) also find `getToolsOfTypeNoNail`. // (added in build 21.1.1)

See [General](#) for member functions applicable to all tools.

---

```
class ElemNoNail {
    ElemNoNail(int nZoneIndex, PLine plNailing);

    void setToolType(int nToolType); // (added in build 21.1.1)
    void setToolIndex(int nToolIndex); // (added in build 21.1.1)

    int toolType(); // (added in build 21.1.1)
    int toolIndex(); // (added in build 21.1.1)
    int zonelIndex(); // (added in build 21.1.1)
    PLine plShape(); // (added in build 21.1.1)

};
```

---

[Example O-type with insert implemented:

```
int nArZone [ ]={-5,-4,-3,-2,-1,1,2,3,4,5};
```

---

```

PropInt nZone(0,nArZone,T("|Active zone|"),5); // default is
zone 1

PropInt nToolingIndex(1,0,T("|Tooling index (0 means
all)|"));

int arIToolTypes[] = { _kTTAllTools, _kT TNail, _kTT Saw,
_kTTMill, _kTTDrill, _kTTMarker, _kTTItem, _kTTNailCluster };
String arSToolTypes[] = { "AllTools", "Nail", "Saw", "Mill",
"Drill", "Marker", "Item", "NailCluster" };
PropString sToolType(2, arSToolTypes, T("|Tool type|"));
int nToolTypeID = arSToolTypes.find(sToolType, 0);
if (nToolTypeID < 0) nToolTypeID = 0;
int nToolType = arIToolTypes[nToolTypeID];

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint(T("|Select first point|"));
    _PtG.append(getPoint(T("|Select second point|")));

    showDialog();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;
if (_PtG.length()==0) return;

CoordSys cs = el.coordSys();
ElemNoNail tls[] = el.getToolsOfTypeNoNail();
reportMessage("\nNumber of ElemNoNails found: " +
tls.length());

PlaneProfile prof(cs);
for(int t=0; t<tls.length(); t++)
{
    ElemNoNail enn = tls[t];
    if (enn.zoneIndex() != nZone)
        continue;
    if (nToolingIndex != 0 && enn.toolIndex() != nToolingIndex)
        continue;
    if (nToolType != _kTTAllTools && enn.toolType() != _kTTAllTools && enn.toolType() != nToolType)
        continue;
}

```

```

        prof.joinRing(enn.plShape(), 0);
    }

Display dp(-1);
dp.draw(prof);

LineSeg seg(_Pt0, _PtG[0]);

dp.color(3);
dp.draw(seg);

dp.color(1);
int bKeepInternal = TRUE;
LineSeg segsIn[] = prof.splitSegments(seg, bKeepInternal);
dp.draw(segsIn);

dp.color(4);
bKeepInternal = FALSE;
LineSeg segsOut[] = prof.splitSegments(seg, bKeepInternal);
dp.draw(segsOut);

```

*—end example]*

## 12.3 ElemSaw

The ElemSaw tool defines a sawing line on an element. With this ElemSaw tool one can either make a [SawLine](#) entity, or apply the tool to the element from inside the TSL.

A sawing line, as well as a milling line has a "nSideRight" property. But what is left and right in space! For this the tool uses the PLine coordinate system. Walking from start point to end point, with the normal of the PLine pointing upwards, the nSideRight should be 1 for the right side.

Therefore it is essential that the normal is set for the PLine that is being used. A normal for a [PLine](#) can be set in the constructor or with the method [setNormal](#).

On [Element](#) also find [getToolsOfTypeSaw](#). // added since 23.5.9

Example also see [ElemMill](#).

See [General](#) for member functions applicable to all tools.

---

```

class ElemSaw {

    ElemSaw(int nZoneIndex, PLine plSaw, double dDepth, int nToolingIndex, int nSideRight, int
    nTurningDirectionWith, int nOverShoot);
    ElemSaw(int nZoneIndex, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double
    dDepth, int nToolingIndex, int nSideRight, int nTurningDirectionWith, int nOverShoot);
}

```

---

```

// deprecated, old, not to be used anymore:
ElemSaw(int nZoneIndex, Point3d ptFrom, Point3d ptTo, double dDepth, int nToolingIndex,
int nSideRight, int nTurningDirectionWith, int nOverShoot); // constructor without vecNormal

void setVacuum(int nVacuum); // nVacuum 0: no vacuum, 1: has vacuum on; use predefines
_kNo, _kYes

void setAngle(double dAngle); // The angle in degrees that the saw blade has to turn in the
right hand turning direction running along the polyline.
void setAngle(Vector3d vecAngle); // The angle is calculated from the first segment of the
polyline, and the vecAngle vector. This vector lies in the plane of the saw blade.

void setSideToCenter(); // Sets the side not to be left nor right but center. (added 17.0.41).

void setSideToLeft(); // added since 23.5.9
void setSideToRight(); // added since 23.5.9
void setToolIndex(int nVal); // added since 23.5.9
void setZoneIndex(int nVal); // added since 23.5.9
void setDepth(double dVal); // added since 23.5.9
void setPIShape(PLine plShape); // added since 23.5.9
void setOverShoot(int nVal); // added since 23.5.9
void setTurningDirectionWith(int nVal); // added since 23.5.9

int toolIndex() const; // added since 23.5.9
int zoneIndex() const; // added since 23.5.9
double angle() const; // added since 23.5.9
double depth() const; // added since 23.5.9
PLine plShape() const; // added since 23.5.9
int overShoot() const; // added since 23.5.9
int vacuum() const; // added since 23.5.9
int turningDirectionWith() const; // added since 23.5.9
int sidelsLeft() const; // added since 23.5.9
int sidelsRight() const; // added since 23.5.9
int sidelsCenter() const; // added since 23.5.9

};


```

---

The constructor has the following formats:

```

ElemSaw(int nZoneIndex, PLine plSaw, double dDepth, int nToolingIndex, int nSideRight,
int nTurningDirectionWith, int nOverShoot);
ElemSaw(int nZoneIndex, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double
dDepth, int nToolingIndex, int nSideRight, int nTurningDirectionWith, int
nOverShoot);
deprecated, old, not to be used anymore: ElemSaw(int nZoneIndex, Point3d ptFrom,
Point3d ptTo, double dDepth, int nToolingIndex, int nSideRight, int
nTurningDirectionWith, int nOverShoot); // constructor without vecNormal

```

nZoneIndex: zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5

---

plNailing: polyline defining the location of the operation  
 dDepth: depth of the operation  
 nToolingIndex: the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used  
 ptFrom: starting point of line  
 ptTo: ending point of line  
 vecNormal: normal vector for line, needed for definition of nSideRight  
 nSideRight: !=1 left, 1 right; use predefines **\_kLeft, \_kRight**  
 nTurningDirectionWith: 0 against course , 1 with course; use predefines  
**\_kTurnAgainstCourse, \_kTurnWithCourse**  
 nOverShoot: 0: no overshoot, 1: has overshoot; use predefines **\_kNo, \_kYes**

---

[Example O-type with insert implemented:

```

if (_bOnInsert) {

    _Element.append(getElementFromEntity());
    _Pt0 = getPoint();
    _PtG.append(getPoint());
    _PtG.append(getPoint());
}

PropInt nZoneIndex(0,1,T("Zone index"));
PropDouble dDepth(0,U(10),T("Depth"));
PropInt nToolingIndex(1,0,T("Tooling index"));

String arSSide[] = {T("Left"),T("Right")};
int arNSide[] = {_kLeft, _kRight};
PropString strSide(0,arSSide,T("Side"));
int nSide = arNSide[arSSide.find(strSide,0)];

String arSTurn[] = {T("Against course"),T("With course")};
int arNTurn[] = {_kTurnAgainstCourse, _kTurnWithCourse};
PropString strTurn(1,arSTurn,T("Turning direction"));
int nTurn = arNTurn[arSTurn.find(strTurn,0)];

String arSOShoot[] = {T("No"),T("Yes")};
int arNOShoot[] = {_kNo, _kYes};
PropString strOShoot(2,arSOShoot,T("Overshoot"));
int nOShoot = arNOShoot[arSOShoot.find(strOShoot,0)];

double dRadius = U(1000,"mm");

PropDouble dAngle(1,30,T("Angle"));

PLine pline(_ZU); // define PLine normal
pline.addVertex(_Pt0); // add first point
pline.addVertex(_PtG[0],dRadius ,_kCWise);
pline.addVertex(_PtG[1],dRadius ,_kCCWise);

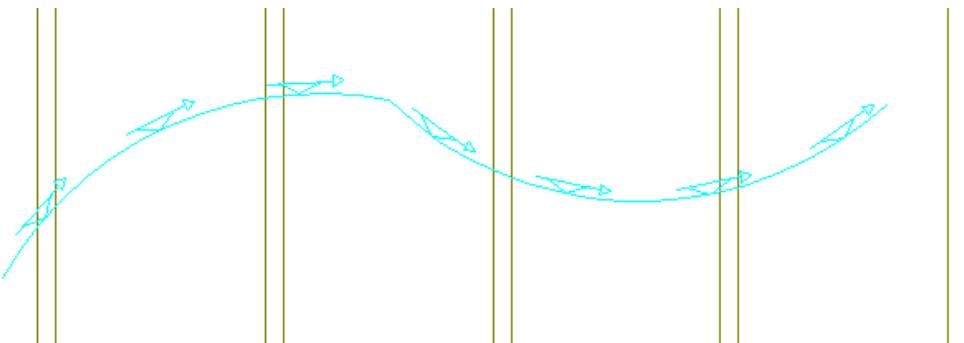
```

---

```
//Display dp(-1);
//dp.draw(pline);

ElemSaw tool(nZoneIndex,pline,dDepth,nToolingIndex,nSide,nTurn,nOShoot);
tool.setAngle(dAngle);
_Element0.addTool(tool);

// add the script entity to the element group itself
assignToElementGroup(_Element0,TRUE,0,'E');
```



*[—end example]*

## 12.4 ElemMill

The ElemMill tool defines a milling line on an element.

A sawing line, as well as a milling line has a "nSideRight" property. But what is left and right in space! For this the tool uses the PLine coordinate system. Walking from start point to end point, with the normal of the PLine pointing upwards, the nSideRight should be 1 for the right side. Therefore it is essential that the normal is set for the PLine that is being used. A normal for a [PLine](#) can be set in the constructor or with the method [setNormal](#).

On [Element](#) also find [getToolsOfTypeMill](#). // added since 23.5.9

See [General](#) for member functions applicable to all tools.

---

```
class ElemMill {

    ElemMill(int nZoneIndex, PLine plMill, double dDepth, int nToolingIndex, int nSideRight, int
    nTurningDirectionWith, int nOverShoot);
    ElemMill(int nZoneIndex, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double
    dDepth, int nToolingIndex, int nSideRight, int nTurningDirectionWith, int nOverShoot);

    // deprecated, old, not to be used anymore:
    ElemMill(int nZoneIndex, Point3d ptFrom, Point3d ptTo, double dDepth, int nToolingIndex,
    int nSideRight, int nTurningDirectionWith, int nOverShoot); // constructor without vecNormal
}
```

---

```

void setVacuum(int nVacuum); // nVacuum 0: no vacuum, 1: has vacuum on; use predefines
    _kNo, _kYes

void setSideToCenter(); // Sets the side not to be left nor right but center. (added 17.0.41).

void setSideToLeft(); // added since 23.5.9
void setSideToRight(); // added since 23.5.9
void setToolIndex(int nVal); // added since 23.5.9
void setZoneIndex(int nVal); // added since 23.5.9
void setDepth(double dVal); // added since 23.5.9
void setPIShape(PLine plShape); // added since 23.5.9
void setOverShoot(int nVal); // added since 23.5.9
void setTurningDirectionWith(int nVal); // added since 23.5.9

int toolIndex() const; // added since 23.5.9
int zoneIndex() const; // added since 23.5.9
double depth() const; // added since 23.5.9
PLine plShape() const; // added since 23.5.9
int overShoot() const; // added since 23.5.9
int vacuum() const; // added since 23.5.9
int turningDirectionWith() const; // added since 23.5.9
int sidelsLeft() const; // added since 23.5.9
int sidelsRight() const; // added since 23.5.9
int sidelsCenter() const; // added since 23.5.9

};


```

---

The constructor has the following formats:

```

ElemMill(int nZoneIndex, PLine plMill, double dDepth, int nToolingIndex, int nSideRight,
    int nTurningDirectionWith, int nOverShoot);
ElemMill(int nZoneIndex, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double
    dDepth, int nToolingIndex, int nSideRight, int nTurningDirectionWith, int
    nOverShoot);
deprecated, old, not to be used anymore: ElemMill(int nZoneIndex, Point3d ptFrom,
    Point3d ptTo, double dDepth, int nToolingIndex, int nSideRight, int
    nTurningDirectionWith, int nOverShoot); // constructor without vecNormal

```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
plNailing:	polyline defining the location of the operation
dDepth:	depth of the operation
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used
ptFrom:	starting point of line
ptTo:	ending point of line
vecNormal:	normal vector for line, needed for definition of nSideRight
nSideRight:	!=1 left, 1 right; use predefines <b>_kLeft</b> , <b>_kRight</b>
nTurningDirectionWith:	0 against course , 1 with course; use predefines <b>_kTurnAgainstCourse</b> , <b>_kTurnWithCourse</b>

---

nOverShoot: 0: no overshoot, 1: has overshoot; use predefines `_kNo`, `_kYes`

---

[Example O-type with insert implemented:

```

Unit(1,"mm"); // U(x) in this script means x mm

PropInt nZoneIndex(0,1,T("Zone index"));
PropDouble dDepth(0,U(10),T("Depth"));
PropInt nToolingIndex(1,0,T("Tooling index"));

String arSSide[] = {T("Left"),T("Right")};
int arNSide[] = {_kLeft, _kRight};
PropString strSide(0,arSSide,T("Side"));
int nSide = arNSide[arSSide.find(strSide,0)];

String arSTurn[] = {T("Against course"),T("With course")};
int arNTurn[] = {_kTurnAgainstCourse, _kTurnWithCourse};
PropString strTurn(1,arSTurn,T("Turning direction"));
int nTurn = arNTurn[arSTurn.find(strTurn,0)];

String arSOShoot[] = {T("No"),T("Yes")};
int arNOShoot[] = {_kNo, _kYes};
PropString strOShoot(2,arSOShoot,T("Overshoot"));
int nOShoot = arNOShoot[arSOShoot.find(strOShoot,0)];

String arSVacuum[] = {T("No"),T("Yes")};
int arNVacuum[] = {_kNo, _kYes};
PropString strVacuum(3,arSVacuum,T("Vacuum"));
int nVacuum = arNVacuum[arSVacuum.find(strVacuum,0)];

if (_bOnDebug) {
    reportMessage("\n Zone index: " + nZoneIndex);
    reportMessage("\n Depth: " + dDepth);
    reportMessage("\n Tooling index: " + nToolingIndex);
    reportMessage("\n Right side: " + nSide);
    reportMessage("\n Turning direction with course: " + nTurn);
    reportMessage("\n Overshoot: " + nOShoot);
    reportMessage("\n Vacuum: " + nVacuum);
}

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    _PtG.append(getPoint());
}

// check if the instance was added to another element, only keep the last element
if (_Element.length() > 1) {

```

---

```

for (int i=1; i<_Element.length(); i++) {
    if (_Element[i].blsValid()) {
        _Element[0] = _Element[i]; // take last element as THE reference
        _Element0 = _Element[0];
        _Element[i] = Element(); // set to no reference
    }
}

// check if the _Element0 was added to the instance
if (_Element0.blsValid()) {

    int nZoneInc = (nZoneIndex>=0)?1:-1;
    ELEMZone ez = _Element0.zone(nZoneIndex+nZoneInc); // go on top surface of zone:
(nZoneIndex+1)
    CoordSys cs = ez.coordSys(_Pt0);
    _Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point on the zone

    // also move the grippoint to the surface of the zone
    Vector3d vec = _PtG[0]-_Pt0;
    double dxv = vec.dotProduct(cs.vecX());
    double dyv = vec.dotProduct(cs.vecY());
    _PtG[0] = _Pt0 + dxv*cs.vecX() + dyv*cs.vecY();

    PLine line(cs.vecZ()); // sets the normal of the polyline
    line.addVertex(_Pt0);
    line.addVertex(_PtG[0]);

    // add the tool
    ELEMMill tool(nZoneIndex,line,dDepth,nToolingIndex,nSide,nTurn,nOShoot);
    tool.setVacuum(nVacuum);
    _Element0.addTool(tool);

    // add the script entity to the element group itself
    assignToElementGroup(_Element0,TRUE,0,'E');
}

```

*—end example]*

## 12.5 ELEMDrill

The ELEMDrill tool defines a drilling on an element.

On [Element](#) also find `getToolsOfTypeDrill`. // added since 23.5.9

See [General](#) for member functions applicable to all tools.

---

```

class ELEMDrill {

    ELEMDrill(int nZoneIndex, Point3d ptLocation, Vector3d vecDirection, double dDepth, int
nToolingIndex);

```

---

```

ElemDrill(int nZoneIndex, Point3d ptLocation, Vector3d vecDirection, double dDepth,
double dDiameter, int nToolingIndex);

void setToolIndex(int nVal); // added since 23.5.9
void setZoneIndex(int nVal); // added since 23.5.9
void setDepth(double dVal); // added since 23.5.9
void setDiameter(double dVal); // added since 23.5.9
void setLocation(Point3d ptLocation); // added since 23.5.9
void setDirection(Vector3d vecDirection); // added since 23.5.9
void setVacuum(int nVacuum); // nVacuum 0: no vacuum, 1: has vacuum on; use predefines
    _kNo, _kYes

int toolIndex() const; // added since 23.5.9
int zoneIndex() const; // added since 23.5.9
double depth() const; // added since 23.5.9
double diameter() const; // added since 23.5.9
Point3d location() const; // added since 23.5.9
Vector3d direction() const; // added since 23.5.9
int vacuum() const; // added since 23.5.9
};


```

---

The constructor has the following formats:

```

ElemDrill name(int nZoneIndex, Point3d ptLocation, Vector3d vecDirection, double
    dDepth, int nToolingIndex);
ElemDrill name(int nZoneIndex, Point3d ptLocation, Vector3d vecDirection, double
    dDepth, double dDiameter, int nToolingIndex);

```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
ptLocation:	location of drill
vecDirection:	drilling direction
dDepth:	drilling depth
dDiameter:	diameter of the drill
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used

```

void setVacuum(int nVacuum);
nVacuum: 0: no vacuum, 1: has vacuum on; use predefines _kNo, _kYes

```

---

[Example O-type:

```

PropInt nZoneIndex(0,1,T("Zone index"));
PropInt nToolingIndex(1,0,T("Tooling index"));

```

---

```

if (_bOnInsert)
{
    _Element.append(getElement());
    _Pt0 = getPoint();
}

// check if the _Element0 was added to the instance
if (_Element0.bIsValid()) {

    ELEMZone ez = _Element0.zone(nZoneIndex);
    CoordSys cs = ez.coordSys(_Pt0);
    _Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point
on the zone

    Point3d ptOnZoneSurface = _Pt0+ez.dH()*cs.vecZ();
    ELEMDrill drilltool(nZoneIndex, ptOnZoneSurface ,-
    cs.vecZ(),ez.dH(),nToolingIndex);
    _Element0.addTool(drilltool);

    Vector3d vz = -cs.vecZ();
    vz.visualize(ptOnZoneSurface );
}

```

*—end example]*

[Example O-type:

```

int nArZone[]={-5,-4,-3,-2,-1,1,2,3,4,5};
PropInt nZone(0,nArZone,T("|Active zone|"),5); // default is zone 1
PropInt nToolingIndex(1,0,T("|Tooling index (0 means all)|"));
PropString pDimStyle(10,_DimStyles ,T("|Dim style|"));
PropDouble pTextHeight(10,U(20),T("|Text height|"));

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint(T("|Select point|"));

    showDialog();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

CoordSys cs = el.coordSys();
ELEMDrill tls[] = el.getToolsOfTypeDrill();

String strLines[0];
strLines.append("Number of ElemDrills found: "+ tls.length());

```

```

Display dp(-1);
for(int t=0; t<tls.length(); t++)
{
    ElemDrill enn = tls[t];
    strLines.append("== tool: "+t);
    strLines.append("toolIndex: "+enn.toolIndex());
    strLines.append("zoneIndex: "+enn.zoneIndex());
    strLines.append("depth: "+enn.depth());
    strLines.append("diameter: "+enn.diameter());

    if (enn.zoneIndex() != nZone)
        continue;
    if (nToolingIndex != 0 && enn.toolIndex() != nToolingIndex)
        continue;

    PLine plCir;
    plCir.createCircle(enn.location(), enn.direction(),
0.5*enn.diameter());
    dp.draw(plCir);
}

// display the lines
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YU;
    dp.draw(strLines[l], _Pt0+vecO, _XU, _YU, 1,1);
}

```

*—end example]*

## 12.6 ElemMarker

The ElemMarker tool defines a marking line on an element.

See [General](#) for member functions applicable to all tools.

The constructor has the following formats:

```

ElemMarker name(int nZoneIndex, PLine plMark, int nToolingIndex);
ElemMarker name(int nZoneIndex, Point3d ptFrom, Point3d ptTo, int nToolingIndex);

```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
plNailing:	polyline defining the location of the operation
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used
ptFrom:	starting point of line
ptTo:	ending point of line

## 12.7 ElemlItem

The ElemlItem tool defines an item on an element.

See [General](#) for member functions applicable to all tools.

The constructor has the following formats:

```
ElemlItem name(int nZoneIndex, String strName, Point3d ptLocation);
ElemlItem name(int nZoneIndex, String strName, Point3d ptLocation, Vector3d
    vecDirection, Map map);
```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
strName:	the name of the item
ptLocation:	location
vecDirection:	optional direction (meaning of this depends on item purpose), default set to (0,0,0)
map:	an optional list of key-value couples, default empty list

```
void setShow(int nShow);
```

nShow: 0: no show, 1: do show; use predefines **\_kNo**, **\_kYes**. If the item is shown, it means that the item name is displayed with the posnum style, at the location ptLocation. The default is value is **\_kYes**.

---

[Example O-type:

```
PropInt nZoneIndex(0,1,T("Zone index"));
PropString strName(0,"MyName",T("Item name"));

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// check if the _Element0 was added to the TSL
if (_Element0.blsValid()) {

    ElemlZone ez = _Element0.zone(nZoneIndex);
    CoordSys cs = ez.coordSys(_Pt0);
    _Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point on the zone

    Point3d ptOnZoneSurface = _Pt0+ez.dH()*cs.vecZ();
    Map map;
    ElemlItem item(nZoneIndex, strName, ptOnZoneSurface ,cs.vecX(), map);
    item.setShow(_kYes); // _kYes is default anyway, so actually not needed here
    _Element0.addTool(item);
```

---

```

    }
—end example]

```

## 12.8 ElemNailCluster

The ElemNailCluster tool defines a cluster of nailing points on an element. It could also be just one nail point.

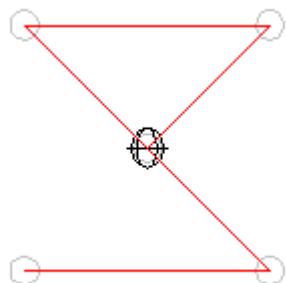
See [General](#) for member functions applicable to all tools.

The constructor has the following formats:

```
ElemNailCluster name(int nZoneIndex, Point3d pnts, int nToolingIndex);
```

nZoneIndex: zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5  
 pnts: points where a nail is required  
 nToolingIndex: the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used

[Example O-type with insert implemented:



```

Unit(1, "mm"); // U(x) in this script means x mm

PropInt nZoneIndex(0,1,T("Zone index"));
PropInt nToolingIndex(1,0,T("Tooling index"));

if (_bOnInsert) {
    _Element.append(getElement());

    _Pt0 = getPoint(TN("|Select start point|"));
    Point3d ptLast = _Pt0;

    while (1) {
        PrPoint ssP2(TN("|Select next point|"), ptLast);

```

```

        if (ssP2.go() == _kOk) { // do the actual query
            ptLast = ssP2.value(); // retrieve the selected point
            _PtG.append(ptLast); // append the selected points to the list of
grippers _PtG
        }
        else { // no proper selection
            break; // out of infinite while
        }
    }

// check if the instance was added to another element
if (_Element.length() > 1) {
    for (int i=1; i < _Element.length(); i++) {
        if (_Element[i].bIsValid()) {
            _Element[0] = _Element[i]; // take last element as THE reference
            _Element0 = _Element[0];
            _Element[i] = Element(); // set to no reference
        }
    }
}

// check if the _Element0 was added to the instance
if (_Element0.bIsValid()) {

    int nZoneInc = (nZoneIndex >= 0)?1:-1;
    ELEMZone ez = _Element0.zone(nZoneIndex+nZoneInc); // go on top surface of
zone: (nZoneIndex+1)
    CoordSys cs = ez.coordSys(_Pt0);
    _Pt0 = cs.ptOrg(); // move the _Pt0 point to the projected point on the
zone

    // start to collect pline points
    PLine spline(cs.vecZ()); // sets the normal of the polyline
    spline.addVertex(_Pt0);

    // also move the gripper to the surface of the zone
    for (int p=0; p < _PtG.length(); p++) {
        Vector3d vec = _PtG[p] - _Pt0;
        double dxv = vec.dotProduct(cs.vecX());
        double dyv = vec.dotProduct(cs.vecY());
        _PtG[p] = _Pt0 + dxv*cs.vecX() + dyv*cs.vecY();
        spline.addVertex(_PtG[p]); // also add the points to the spline
    }

    spline.vis(1);

    // add the tool
    ELEMNailCluster tool(nZoneIndex, spline.vertexPoints(TRUE), nToolingIndex);
    _Element0.addTool(tool);
}

```

```
// add the script entity to the element group itself
assignToElementGroup(_Element0, TRUE, 0, 'E');
}
```

*—end example]*

## 12.9 ElemConstructionBeam

The ElemConstructionBeam tool does not create a beam directly. It gives the element/panel generation, at time of generation of the construction, the directive to make a beam.

See [General](#) for member functions applicable to all tools.

The ElemConstructionBeam tool constructor has one of the following formats:

```
ElemConstructionBeam name(int nZoneIndex, Point3d ptOrg, Vector3d vecX, Vector3d
    vecY, Vector3d vecZ);
ElemConstructionBeam name(int nZoneIndex, Point3d ptOrg, Vector3d vecX, Vector3d
    vecY, Vector3d vecZ, double dXLen, double dYWidth, double dHeight);
ElemConstructionBeam name(int nZoneIndex, Point3d ptOrg, Vector3d vecX, Vector3d
    vecY, Vector3d vecZ, double dXLen, double dYWidth, double dHeight, double
    dXFlag, double dYFlag, double dZFlag);
```

nZoneIndex:	zone index into element: -5,-4,-3,-2,-1,0,1,2,3,4,5
ptOrg:	origin point of the box defining the beam
vecX:	axis in length direction
vecY:	width direction
vecZ:	height direction
dXLen:	length scale factor
dYWidth:	width scale factor
dHeight:	height scale factor

To specify the dimensions of the tool, the length of the vector is multiplied with the scale factor in that direction. Eg.: the length of the beam is the same as **vecX.length()\*dXLen**. If **vecX** is a unit-length vector, then the **dXLen** corresponds with the length. If **dXLen** equals 1, then the length of the vector expresses the length of the beam.

The flags **dXFlag**, **dYFlag** and **dZFlag**, specify the position of the **ptOrg** inside the box defined by the vectors and their lengths. If the flags are all equal to 0, the point **ptOrg** is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the **-vecX**, **-vecY** and **-vecZ** direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the **vecX**, **vecY** and **vecZ** coordinate system with origin **ptOrg**.

The ElemConstructionBeam has a number of other properties, which can be set separately.

```
ElemConstructionBeam::setMaterial(String strVal);
ElemConstructionBeam::setName(String strVal);
ElemConstructionBeam::setLabel(String strVal);
```

```

ElemConstructionBeam::setSubLabel(String strVal);
ElemConstructionBeam::setSubLabel2(String strVal);
ElemConstructionBeam::setInfo(String strVal);
ElemConstructionBeam::setGrade(String strVal);
ElemConstructionBeam::setStretch(Vector3d vecDir, int bStretch); // e.g.
ecb.setStretch(_ZW,TRUE);
ElemConstructionBeam::setType(int nBeamType);
ElemConstructionBeam::setExtrProfile(String strVal);

```

The beamtype is one of the enumerated beam types: **\_kBeam**, **\_kLog**, **\_kPanelTopPlate**,...

---

**setExtrProfile(**String** strVal);**

To set the extrusion profile of the beam that will be created. If the strVal is equal to "Round" or "Rectangular" the predefined extrusion shapes are used.

---

[Example O-type:

**Unit**(1,"mm");

```

PropInt nZoneIndex(0,0,T("Zone index"));

if (_bOnInsert) {
    Opening opng = getOpening();
    _Opening.append(opng);
    _Element.append(opng.element());
    _Pt0 = opng.coordSys().ptOrg();
    return;
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.blsValid()) return;
if (_Opening.length()==0) return;
Opening opn = _Opening[0];
if (!opn.blsValid()) return;

ElemZone ez = el.zone(nZoneIndex);
CoordSys cs = el.coordSys();
PLine plOpn = opn.plShape();

// add the script entity to the element group itself
assignToElementGroup(el,TRUE,0,'E');

// make a ElemConstructionBeam tool

```

```

ElemConstructionBeam ecb(nZoneIndex, _Pt0, U(1000)*cs.vecX(), U(100)*cs.vecY(), U(50)
*cs.vecZ(),1,1,1,1,1,1);
ecb.setMaterial("Pine");
ecb.setName("Slam");
ecb.setType(_kPanelTopPlate);
ecb.setLabel("MyLabel");
ecb.setSubLabel("sub1");
ecb.setSubLabel2("sub12");
ecb.setGrade("gr");
ecb.setInfo("info");
ecb.setStretch(cs.vecX(),TRUE);
ecb.setStretch(-cs.vecX(),TRUE);
el.addTool(ecb);

```

*—end example]*

## 12.10 PanelSplit

The PanelSplit defines a location where the Sip panel is to be split during panelization. This tool is to be appended to an element.

See [General](#) for member functions applicable to all tools.

---

```

class PanelSplit {

    PanelSplit(Point3d ptLocation, Vector3d vecDirection);

    void setEdgeRecessType(int nRecessType);
    int edgeRecessType() const;

    void setEdgeDetailCodeLeft(String strCode);
    String edgeDetailCodeLeft() const;
    void setEdgeDetailCodeRight(String strCode);
    String edgeDetailCodeRight() const;

    void setDSplineWidth(double dVal);
    double dSplineWidth() const;
    void setDSplineHeight(double dVal);
    double dSplineHeight() const;
    void setDGap(double dVal);
    double dGap() const;
    void setDWidthSteg(double dVal);
    double dWidthSteg() const;
    void setDWidthCenter(double dVal);
    double dWidthCenter() const;
    void setDLumberOffset(double dVal);
    double dLumberOffset() const;

    void setNQtyLeft(int nVal);
}

```

```

int nQtyLeft() const;
void setNQtyRight(int nVal);
int nQtyRight() const;
void setNQtyCenter(int nVal);
int nQtyCenter() const;
void setBCenterStudsHaveFullPanelWidth(int nVal);
int bCenterStudsHaveFullPanelWidth() const;

void setExtrProfile(String str);
String extrProfile() const;

void setOpening(Opening op, int nOpeningRelation); // see Opening

String material() const;
void setMaterial(String strValue);
String name() const;
void setName(String strValue);

};

```

---

Values of nRecessType:

- \_kNoEdgeRecess**
- \_kSplineDouble**
- \_kSplineIJoist**
- \_kSpline2xLumber**
- \_kSplineSingleLumber**
- \_kSplineSingleTopSkin**
- \_kSplineBlock**
- \_kSplineCustom**
- \_kLetIn**
- \_kHeaderRecess**

Values of nOpeningRelation:

- \_kSplitAboveOpening**
- \_kSplitBelowOpening**

See [General](#) for member functions applicable to all tools.

---

[Example of O-type with insert done in script:

```

Unit(1, "inch");

if (_bOnInsert) {

```

```

_Element.append(getElement());
_Pt0 = getPoint();
_Opening.append(getOpening());
}

// check execute conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

CoordSys cs = el.coordSys();
// move the _Pt0 point
_Pt0 -= cs.vecZ().dotProduct(_Pt0-cs.ptOrg())*cs.vecZ();

PLine plEl = el.plEnvelope();
Point3d pntsEl[] = plEl.intersectPoints(Line(cs.ptOrg(),cs.vecX()));
if (pntsEl.length()<2) return;
Point3d ptMin = pntsEl[0];
Point3d ptMax = pntsEl[pntsEl.length()-1];
double dPtY = cs.vecY().dotProduct(_Pt0-ptMin);
double dPtX = cs.vecX().dotProduct(_Pt0-ptMin);
Point3d ptBase = _Pt0 - dPtY*cs.vecY();

ExtrProfile extrProfs[] = ExtrProfile().getAllEntries();
String strExtrProfName;
if (extrProfs.length()>0) {
    ExtrProfile profL = extrProfs[extrProfs.length()-1];
    strExtrProfName = profL.entryName();
}

// add the script entity to the element group itself
assignToElementGroup(el,TRUE,0,'E');

PanelSplit ps(ptBase, cs.vecY());
ps.setEdgeRecessType(_kSplineIJoist);
ps.setEdgeDetailCodeLeft("laa");
ps.setEdgeDetailCodeRight("raa");

ps.setDSplineWidth(U(10));
ps.setDSplineHeight(U(11));
ps.setDGap(U(12));
ps.setDWidthSteg(U(13));
ps.setDWidthCenter(U(14));

ps.setNQtyLeft(2);
ps.setNQtyRight(3);
ps.setNQtyCenter(4);
ps.setBCenterStudsHaveFullPanelWidth(1);

ps.setExtrProfile(strExtrProfName);

if (_Opening.length()>0) {
    ps.setOpening(_Opening[0],_kSplitBelowOpening);
}

```

```

}

el.addTool(ps);

reportMessage("\n");
reportMessage("\n edgeRecessType: "+ps.edgeRecessType());
reportMessage("\n edgeDetailCodeLeft: "+ps.edgeDetailCodeLeft());
reportMessage("\n edgeDetailCodeRight: "+ps.edgeDetailCodeRight());
reportMessage("\n dSplineWidth: "+ps.dSplineWidth());
reportMessage("\n dSplineHeight: "+ps.dSplineHeight());
reportMessage("\n dGap: "+ps.dGap());
reportMessage("\n dWidthSteg: "+ps.dWidthSteg());
reportMessage("\n dWidthCenter: "+ps.dWidthCenter());
reportMessage("\n nQtyLeft: "+ps.nQtyLeft());
reportMessage("\n nQtyRight: "+ps.nQtyRight());
reportMessage("\n nQtyCenter: "+ps.nQtyCenter());
reportMessage("\n bCenterStudsHaveFullPanelWidth:
"+ps.bCenterStudsHaveFullPanelWidth());
reportMessage("\n extrProfile: "+ps.extrProfile());

```

*—end example]*

## 12.11 ElemConstructionMap

The ElemConstructionMap is to be appended to an element. It gives the element/panel generation, at time of generation of the construction, a certain directive. The tool is meant for advanced use only.

See [General](#) for member functions applicable to all tools.

The constructor has the following format:

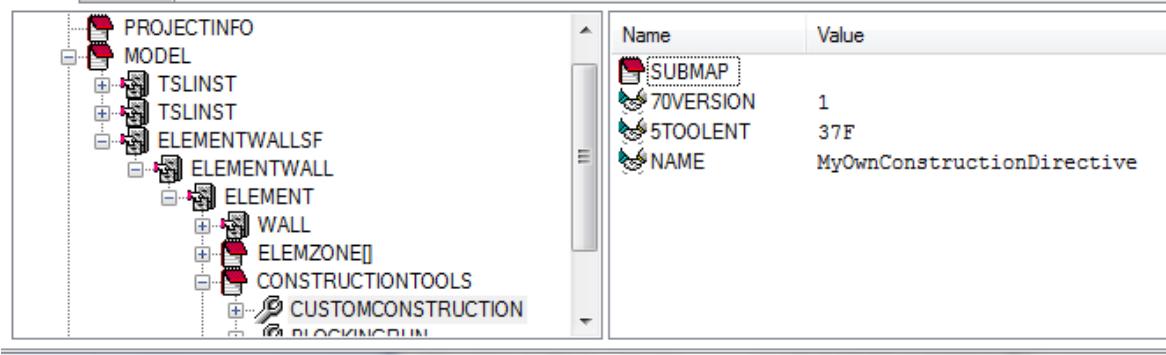
**ElemConstructionMap** name(**String** strToolName, **Map** map); // (added hsbCAD14.0.76)

strToolName: The name of the tool in the generation process.  
map: The map that describes the contents of the tool.

The ElemConstructionMap has some member functions, which return a copy of the constructor values:

**Map** map() const; // see [Map](#)  
**String** toolName() const;

Since hsbCAD2013 build 18.1.9 on, the ElemConstructionMap is exported different in ModelX than before. Since this version, it is exported as CustomConstruction, and appears in CadModel as CustomConstructionTool.



[Example O-type:

```

Unit(1, "mm");

if (_bOnInsert) {

    _Element.append(getElement());
    _Pt0 = _Element[0].coordSys().ptOrg();
    return;
}

// check execute conditions
if (_Element.length() == 0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

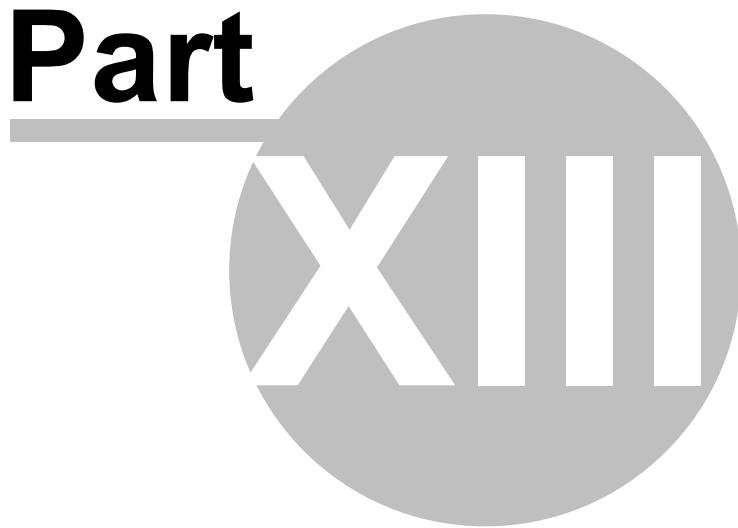
Map map;
map.setInt("int", 3);
map.setDouble("len", U(5));
map.setPoint3d("pt0", _Pt0);
map.setVector3d("vecXY", _XU + _YU);
ElemConstructionMap cd("MyOwnConstructionDirective", map);

el.addTool(cd);

```

—end example]

**Part**



## 13 Tools on MasterPanels

### 13.1 ElemSaw\_MasterPanel

The [ElemSaw](#) tool can be attached to a [MasterPanel](#) (since v21.1.43 and v22.0.30)

A sawing line, as well as a milling line has a "nSideRight" property. But what is left and right in space! For this the tool uses the PLine coordinate system. Walking from start point to end point, with the normal of the PLine pointing upwards, the nSideRight should be 1 for the right side. Therefore it is essential that the normal is set for the PLine that is being used. A normal for a [PLine](#) can be set in the constructor or with the method `setNormal`.

The constructor has the following formats:

```
ElemSaw name(int nn, PLine plSaw, double dDepth, int nToolingIndex, int nSideRight,
           int nn, int nn);
ElemSaw name(int nn, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double
           dDepth, int nToolingIndex, int nSideRight, int nn, int nn);
```

nn:	argument without any meaning for application on MasterPanel
plSaw:	polyline defining the location of the operation
dDepth:	depth of the operation
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used
ptFrom:	starting point of line
ptTo:	ending point of line
vecNormal:	normal vector for line, needed for definition of nSideRight
nSideRight:	!=1 left, 1 right; use predefines <a href="#">_kLeft</a> , <a href="#">_kRight</a>

```
void setAngle(double dAngle);
void setAngle(Vector3d vecAngle);
```

dAngle: The angle in degrees that the saw blade has to turn in the right hand turning direction running along the polyline.

vecAngle: The angle is calculated from the first segment of the polyline, and the vecAngle vector. This vector lies in the plane of the saw blade.

```
void setSideToCenter();
```

Sets the side not to be left or right but center. (added 17.0.41).

[Example O-type with insert implemented:

```
U(1, "mm");

if (_bOnInsert)
{
    _Entity.append(getMasterPanel());
    _Pt0 = getPoint(TN("Select start point"));
    Point3d ptLast = _Pt0;

    while (1) {
```

```

        PrPoint ssp2(TN("|Select next point|"), ptLast);
        if (ssp2.go() == _kOk) {
            ptLast = ssp2.value();
            _PtG.append(ptLast);
        }
        else { //no proper selection
            break; //out of infinite while
        }
    }

    return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}
MasterPanel mp = (MasterPanel)_Entity[0];
if (!mp.bIsValid())
{
    eraseInstance();
    return;
}

PropDouble dSawDepth(0,U(10),T("|Saw depth|"));
PropInt nToolingIndex(1,0,T("|Tooling index|"));

String arSSide[] = {T("|Left|"),T("|Right|")};
int arNSide[] = {_kLeft, _kRight};
PropString strSide(0,arSSide,T("|Side|"));
int nSide = arNSide[arSSide.find(strSide,0)];

String arSTurn[] = {T("|Against course|"),T("|With course|")};
int arNTurn[] = {_kTurnAgainstCourse, _kTurnWithCourse};
PropString strTurn(1,arSTurn,T("|Turning direction|"));
int nTurn = arNTurn[arSTurn.find(strTurn,0)];

String arSOShoot[] = {T("|No|"),T("|Yes|")};
int arNOShoot[] = {_kNo, _kYes};
PropString strOShoot(2,arSOShoot,T("|Overshoot|"));
int nOShoot = arNOShoot[arSOShoot.find(strOShoot,0)];

String arSTopBottom[] = {T("|Top|"),T("|Bottom|")};
int arNTopBottom[] = {1, -1};
PropString strTopBottom(3,arSTopBottom,T("|Face|"));
int nTopBottom = arNTopBottom[arSTopBottom.find(strTopBottom,0)];

PropDouble dSawAngle(1,0,T("|Saw angle in degrees|"));

String arSShow[] = {T("|No|"),T("|Yes|")};
int arNShow[] = {FALSE, TRUE};
PropString strShow(4,arSShow,T("|Show|"));

```

```

int nShow = arNShow[arSShow.find(strShow,1)];

// ----

CoordSys csMP = mp.coordSys();
csMP.vis();
Vector3d vecZ = csMP.vecZ();
if (nTopBottom < 0)
    vecZ = - vecZ;

double dOffset = 0;
if (nTopBottom < 0)
    dOffset = -mp.dThickness();

// move _Pt0
Point3d ptSurface = csMP.ptOrg(); // coordSys of MasterPanel is on
surface
_Pt0 -= (dOffset + vecZ.dotProduct(_Pt0 - ptSurface)) * vecZ;
// move all grippoints to the same plane as _Pt0
for (int i=0; i<_PtG.length(); i++)
{
    _PtG[i] -= (dOffset + vecZ.dotProduct(_PtG[i] - ptSurface)) *
vecZ;
}

// make a polyline for sawing
PLine plSaw(vecZ); // add correct normal
plSaw.addVertex(_Pt0);
for (int i=0; i<_PtG.length(); i++)
{
    plSaw.addVertex(_PtG[i]);
}

//Display dp(-1);
//dp.draw(plSaw);

ElemSaw tool(0, plSaw, dSawDepth, nToolingIndex, nSide, nTurn,
nShoot);
tool.setAngle(dSawAngle); // angle in degrees
mp.addTool(tool);

_ThisInst.setShowElementTools(nShow);

```

*—end example]*

## 13.2 ElemMill\_MasterPanel

The [ElemMill](#) tool can be attached to a [MasterPanel](#) (since v21.1.43 and v22.0.30).

A sawing line, as well as a milling line has a "nSideRight" property. But what is left and right in space! For this the tool uses the PLine coordinate system. Walking from start point to end point, with the normal of the PLine pointing upwards, the nSideRight should be 1 for the right side.

Therefore it is essential that the normal is set for the PLine that is being used. A normal for a [PLine](#) can be set in the constructor or with the method [setNormal](#).

The constructor has the following formats:

```
ElemMill name(int nn, PLine plMill, double dDepth, int nToolingIndex, int nSideRight, int nn, int nn);
ElemMill name(int nn, Point3d ptFrom, Point3d ptTo, Vector3d vecNormal, double dDepth, int nToolingIndex, int nSideRight, int nn, int nn);
```

nn: argument without any meaning for application on MasterPanel  
 plMill: polyline defining the location of the operation  
 dDepth: depth of the operation  
 nToolingIndex: the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used  
 ptFrom: starting point of line  
 ptTo: ending point of line  
 vecNormal: normal vector for line, needed for definition of nSideRight  
 nSideRight: !=1 left, 1 right; use predefines [\\_kLeft](#), [\\_kRight](#)

**void** [setSideToCenter](#)();

Sets the side not to be left or right but center. (added 17.0.41).

[Example O-type with insert implemented:

```
U(1, "mm");

if (_bOnInsert)
{
    _Entity.append(getMasterPanel());

    Pt0 = getPoint(TN("|Select start point|"));
    Point3d ptLast = Pt0;

    while (1) {
        PrPoint ssP2(TN("|Select next point|"), ptLast);
        if (ssP2.go() == kOk) {
            ptLast = ssP2.value();
            PtG.append(ptLast);
        }
        else { //no proper selection
            break; //out of infinite while
        }
    }
}
```

```

        return;
    }

    if (_Entity.length() == 0)
    {
        eraseInstance();
        return;
    }
    MasterPanel mp = (MasterPanel)_Entity[0];
    if (!mp.bIsValid())
    {
        eraseInstance();
        return;
    }

    PropDouble dMillDepth(0,U(10),T("|Mill depth|"));
    PropInt nToolingIndex(1,0,T("|Tooling index|"));

    String arSSide[] = {T("|Left|"),T("|Right|")};
    int arNSide[] = {_kLeft, _kRight};
    PropString strSide(0,arSSide,T("|Side|"));
    int nSide = arNSide[arSSide.find(strSide,0)];

    int nTurn = 0;
    int nOShoot = 0;

    String arSTopBottom[] = {T("|Top|"),T("|Bottom|")};
    int arNTopBottom[] = {1, -1};
    PropString strTopBottom(3,arSTopBottom,T("|Face|"));
    int nTopBottom = arNTopBottom[arSTopBottom.find(strTopBottom,0)];

    // -----
    CoordSys csMP = mp.coordSys();
    Vector3d vecZ = csMP.vecZ();
    if (nTopBottom < 0)
        vecZ = -vecZ;

    double dOffset = 0;
    if (nTopBottom < 0)
        dOffset = -mp.dThickness();

    // move _Pt0
    Point3d ptSurface = csMP.ptOrg(); // coordSys of MasterPanel is on
    surface
    _Pt0 -= (dOffset + vecZ.dotProduct(_Pt0 - ptSurface)) * vecZ;
    // move all grippoints to the same plane as _Pt0
    for (int i=0; i<_PtG.length(); i++)
    {
        _PtG[i] -= (dOffset + vecZ.dotProduct(_PtG[i] - ptSurface)) *
        vecZ;
    }
}

```

```

// make a polyline
PLine plMill(vecZ); // add correct normal
plMill.addVertex(_Pt0);
for (int i=0; i<_PtG.length(); i++)
{
    plMill.addVertex(_PtG[i]);
}

//Display dp(-1);
//dp.draw(plMill);

ElemMill tool(0, plMill, dMillDepth, nToolingIndex, nSide, nTurn,
nShoot);
mp.addTool(tool);

—end example]

```

### 13.3 ElemMarker\_MasterPanel

The [ElemMarker](#) tool can be attached to a [MasterPanel](#) (since v21.1.43 and v22.0.30).

```

ELEMARKER name(int nn, PLine plMark, int nToolingIndex);
ELEMARKER name(int nn, Point3d ptFrom, Point3d ptTo, int nToolingIndex);

nn:           argument without any meaning for application on MasterPanel
plMark:       polyline defining the location of the operation
nToolingIndex: the machine might have multiple aggregates/toolings, the nToolingIndex
                identifies the correct one to be used
ptFrom:       starting point of line
ptTo:         ending point of line

```

---

*[Example O-type with insert implemented:*

```

U(1, "mm");

if (_bOnInsert)
{
    _Entity.append(getMasterPanel());

    _Pt0 = getPoint(TN("Select start point"));
    Point3d ptLast = _Pt0;

    while (1) {
        PrPoint ssP2(TN("Select next point"), ptLast);
        if (ssP2.go() == _kOk) {
            ptLast = ssP2.value();
            _PtG.append(ptLast);
        }
        else { //no proper selection
            break; //out of infinite while
        }
    }
}

```

```

        return;
    }

    if (_Entity.length() == 0)
    {
        eraseInstance();
        return;
    }
    MasterPanel mp = (MasterPanel)_Entity[0];
    if (!mp.bIsValid())
    {
        eraseInstance();
        return;
    }

    PropInt nToolingIndex(1,0,T("|Tooling index|"));

    String arSTopBottom[] = {T("|Top|"),T("|Bottom|")};
    int arNTopBottom[] = { 1, -1 };
    PropString strTopBottom(3,arSTopBottom,T("|Face|"));
    int nTopBottom = arNTopBottom[arSTopBottom.find(strTopBottom,0)];

    // ----

    CoordSys csMP = mp.coordSys();
    Vector3d vecZ = csMP.vecZ();
    if (nTopBottom < 0)
        vecZ = - vecZ;

    double dOffset = 0;
    if (nTopBottom < 0)
        dOffset = -mp.dThickness();

    // move _Pt0
    Point3d ptSurface = csMP.ptOrg(); // coordSys of MasterPanel is on
    surface
    _Pt0 -= (dOffset + vecZ.dotProduct(_Pt0 - ptSurface)) * vecZ;
    // move all grippoints to the same plane as _Pt0
    for (int i=0; i<_PtG.length(); i++)
    {
        _PtG[i] -= (dOffset + vecZ.dotProduct(_PtG[i] - ptSurface)) *
    vecZ;
    }

    // make a polyline
    PLine plMarker(vecZ); // add correct normal
    plMarker.addVertex(_Pt0);
    for (int i=0; i<_PtG.length(); i++)
    {
        plMarker.addVertex(_PtG[i]);
    }
}

```

```
//Display dp(-1);
//dp.draw(plMarker);

ElemMarker tool(0, plMarker, nToolingIndex);
mp.addTool(tool);

—end example]
```

## 13.4 ElemDrill\_MasterPanel

The [ElemDrill](#) tool can be attached to a [MasterPanel](#) (since v21.1.43 and v22.0.30).

```
ElemDrill name(int nn, Point3d ptLocation, Vector3d vecDirection, double dDepth, int
nToolingIndex);
ElemDrill name(int nn, Point3d ptLocation, Vector3d vecDirection, double dDepth,
double dDiameter, int nToolingIndex);
```

nn:	argument without any meaning for application on MasterPanel
ptLocation:	location of drill
vecDirection:	drilling direction
dDepth:	drilling depth
dDiameter:	diameter of the drill
nToolingIndex:	the machine might have multiple aggregates/toolings, the nToolingIndex identifies the correct one to be used

---

[Example O-type:

```
U(1, "mm");
// PropInt nToolingIndex(1,0,T("|Tooling index|")); // not used
int nToolingIndex = 0;
PropDouble dDrillDiam(0, U(10), T("|Drill diameter|"));
PropDouble dDrillDepth(1, U(40), T("|Drill depth|"));

String arSTopBottom[] = {T("|Top|"), T("|Bottom|")};
int arNTopBottom[] = {1, -1};
PropString strTopBottom(3, arSTopBottom, T("|Face|"));
int nTopBottom = arNTopBottom[arSTopBottom.find(strTopBottom, 0)];

if (_bOnInsert)
{
    _Entity.append(getMasterPanel());
    _Pt0 = getPoint();
    return;
}

if (_Entity.length() == 0)
{
    eraseInstance();
    return;
}
MasterPanel mp = (MasterPanel)_Entity[0];
```

---

```

if (!mp.bIsValid())
{
    eraseInstance();
    return;
}

// project _Pt0 on XY plane
CoordSys csMP = mp.coordSys();
Vector3d vecZ = csMP.vecZ();
if (nTopBottom < 0)
    vecZ = - vecZ;

double dOffset = 0;
if (nTopBottom < 0)
    dOffset = -mp.dThickness();

// move _Pt0
Point3d ptSurface = csMP.ptOrg(); // coordSys of MasterPanel is on
surface
_pt0 -= (dOffset + vecZ.dotProduct(_Pt0 - ptSurface)) * vecZ;

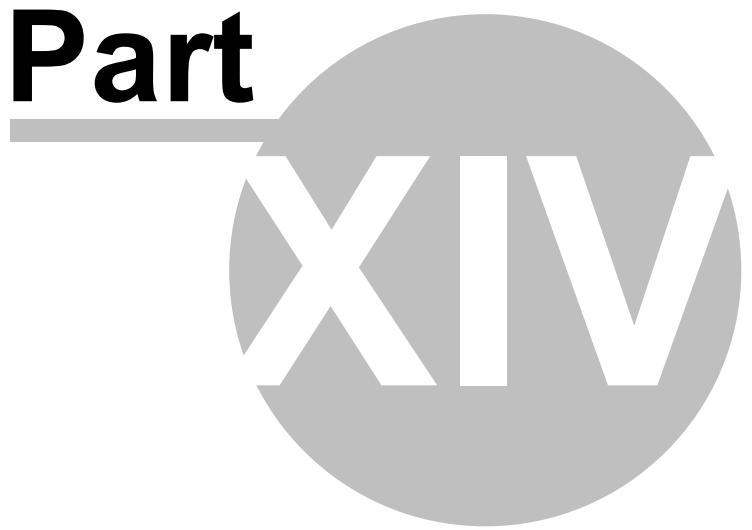
Point3d ptDrill = _Pt0;
Vector3d vecDrill = -vecZ;
ElemDrill drilltool(0, ptDrill, vecDrill, dDrillDepth, dDrillDiam,
nToolingIndex);
mp.addTool(drilltool);

vecDrill.visualize(ptDrill);

—end example

```

**Part**



## 14 Implementing insert in the script

### 14.1 Different approaches possible

When a TSL entity is inserted into the drawing, the required beam, points, element,... need to be selected. Different scripts might require different input to be made.

If the script does not contain any directives on how to do the insert, beams and points will be selected based on the type of the script. E.g. an G-type requires 2 beams, an E-type requires 1 beam and a point, an O-type requests at least 1 point. In this case, also a number of grip-points might be required, given by the value of "Number of extra grip/insert points".

If the flag "Insert done in script" is on, the script contains the actions on what to do during insertion into the drawing. No default actions are taken in this case. The value of "Number of extra grip/insert points" is not important. Also the type of the script does not influence the insert (only the set of predefines). What is done during insert, is completely determined inside the script.

During insert, the value of the predefined `_bOnInsert` will be TRUE. During normal execution, the value will be FALSE.

The insert can be implemented using Prompt types: [PrPoint](#), [PrEntity](#),... or using [global function calls: getPoint\(\), getElement\(\)](#),...

The prompt types are more difficult to use, but give more flexibility, while the global functions are more easy to use, and straight forward to understand.

The state of the TSL entity is special during the insert process. Indeed, during the actual insertion, the entity is not added to the autocad database yet. But once the entity is appended, the script will be executed again. So during the insert process, each TSL will be executed twice, once with the status `_bOnInsert` equal to TRUE (and not appended yet), and once with the `_bOnInsert` being FALSE (already appended to the autocad database).

The insert process will be repeated automatically. So the command of the user to start inserting Tsl's, is always seen as a request of inserting multiple instances. Each time a new Tsl is inserted, the insert process (the 2 execution runs) is repeated. This is done, until the user hits escape on one of the input functions, or if the Tsl fires the command `eraseInstance`. Since the entity is not yet added to the database, the `eraseInstance` is not erasing the instance from the database, but rather erasing it from the input sequence. So firing

`eraseInstance()`

during `_bOnInsert`, will stop the input sequence, and will not append that particular instance to the database.

### 14.2 PrEntity

The `PrEntity` class allows to prompt the user to select an Autocad entity. Of course only the entity types that are known in TSL can be selected. The entity types that will be excepted during the selection can be specified through the type in the constructor, or the `addAllowedClass` function.

---

```
class PrEntity {
    PrEntity(String strPrompt);
    PrEntity(String strPrompt, Entity type);
```

---

```

void addAllowedClass(Entity type);

int go();
Entity[] set() const;

Element[] elementSet() const;
Beam[] beamSet() const;
Sheet[] sheetSet() const;

// added hsbCAD 2012 build 17.1.8: pick first selection set support
static Entity[] getPickFirstSS();
static void setPickFirstSS(<Entity>[] set);

// allowNested supports for selection inside xrefs, added hsbCAD2013 build 18.0.2
void allowNested(int bAllow);
};

```

---

```

PrEntity(String strPrompt);
PrEntity(String strPrompt, Entity type);

```

Constructor, make a new instance of the PrEntity class. Specify the question that will be asked on the command line: strPrompt. The second constructor specifies the type of entity that will be excepted during prompt.

```
void addAllowedClass(Entity type);
```

Add a type of entity that will be excepted during prompt.

```
int go();
```

The actual prompt action is done. Return the success of the operation.

```
Entity[] set() const;
```

Returns the set of entities that are selected. The set is returned as an array of entities.

```

Element[] elementSet() const;
Beam[] beamSet() const;
Sheet[] sheetSet() const;

```

Returns the set of entities that are selected. Each routine is dedicated to one type. The beamSet for instance, will check each entity selected, if it can be casted safely into a Beam. If it can, it is added to the array that is returned.

*[Example O-type that will accepts beams.*

```
if (_bOnInsert){
```

```

PrEntity ssE(T("|Select a set of beams|"), Beam());
if (ssE.go()) {
    Beam ssBeams[] = ssE.beamSet();
    reportMessage (T("\n|Number of beams selected| ") + ssBeams.length());
}
eraseInstance();
return;
}
—end example]

```

```

static Entity[] getPickFirstSS();
static void setPickFirstSS(Entity[] set);

```

The pick first selection set is the set of entities that is selected before the command is run. These methods can only be called in a context that supports user input.

*[Example O-type that manipulates the pick first selection set.*

```

if (_bOnInsert)
{
    PrEntity ssE(T("|Select a set of entities|"), Entity());
    if (ssE.go()) {
        _Entity = ssE.set();
    }
    _Pt0 = getPoint();
    return;
}

String strChange = T("|Change pickfirst ss|");
addRecalcTrigger(_kContext, strChange );
if (_bOnRecalc && _kExecuteKey==strChange )
{
    PrEntity().setPickFirstSS(_Entity);
    reportMessage("\nNumber of entities set to ss: "+_Entity.length()
+"\\n");
    return;
}

// show pick first selection set
{
    Entity ents[] = PrEntity().getPickFirstSS();
    reportMessage("\nNumber of entities in ss: "+ents.length()+"\\n");
    for (int e=0; e<ents.length(); e++)
    {
        reportMessage(" "+ents[e].typeName());
    }
}
—end example]

```

[Example O-type that will accept all the elements that are selected.

```

if (_bOnInsert) {

    _Pt0 = getPoint(); // first select the insertion point

    // the constructor of PrEntity requires a type of entity. Here we use Element()
    PrEntity ssE(T("\n|Select a set of elements|"),Element());

    // let the prompt class do its job, keep requesting, until escape or nothing selected
    //while (ssE.go() && (ssE.set().length()>0)) {
    //while (ssE.go()) { // let the prompt class do its job, keep requesting, until escape
    if (ssE.go()) { // let the prompt class do its job, only one run

        Entity ents[0]; // the PrEntity will return a list of entities, and not elements
        // copy the list of selected entities to a local array: for performance and readability
        ents = ssE.set();

        // turn the selected set into an array of elements
        for (int i=0; i<ents.length(); i++) {

            // The ents (but also the ssE.set() is an array of type Entity. An Element is always an
            // Entity, but an Entity is not necessarily a Element.
            // So in order to append them to a list of elements, the entity needs to be turned
            // into an element. This is called casting. The entity,
            // if allowed, can be casted into an element. To cast, you need to use the (type)
            // operator.
            Element el = (Element)ents[i]; // cast the entity to a element
            _Element.append(el);
        }
    }
    return;
}

reportMessage(T("\n|Number of elements in _Element array:| ")+ _Element.length());

```

—end example]

---

[Example E-type:

```

if (_bOnInsert) {

    // the constructor of PrEntity requires a type of entity. This could be Beam(), Sip(),
    GenBeam(),...
    PrEntity ssE(T("\n|Select a set of beams|"),Beam());

    // let the prompt class do its job, keep requesting, until escape or nothing selected
    while (ssE.go() && (ssE.set().length()>0)) {
    // while (ssE.go()) { // let the prompt class do its job, keep requesting, until escape

```

```

// if (ssE.go()) { // let the prompt class do its job, only one run

Entity ents[0]; // the PrEntity will return a list of entities, and not beams
// copy the list of selected entities to a local array: for performance and readability
ents = ssE.set();

// turn the selected set into an array of beams
Beam ssBeams[0]; // array of zero length
for (int i=0; i<ents.length(); i++) {

    // The ents (but also the ssE.set()) is an array of type Entity. A Beam is always an
    Entity, but an Entity is not necessarily a Beam.
    // So in order to append them to a list of beams, the entity needs to be turned into
    a beam. This is called casting. The entity,
    // if allowed, can be casted into a beam. To cast, you need to use the (type)
    operator.
    Beam bm = (Beam)ents[i]; // cast the entity to a beam
    ssBeams.append(bm);
}
reportMessage(T("\n|Number of beams selected:| "+ ssBeams.length()));

// loop over all the beams, and for each beam, find the other beams that have
// a head or tail face in common. If Beams are parallel, then insert tool.
for (int i=0; i<ssBeams.length(); i++) {
    Beam lstBeamsContact[0];
    lstBeamsContact = ssBeams; // copy the list of beams as list of potential contact
    beams

    // the current beam, with index i is not a good one: replace with an invalid
    reference
    lstBeamsContact[i] = Beam();

    // filter out the beams that are parallel with the beam
    lstBeamsContact = ssBeams[i].filterBeamsParallel(lstBeamsContact);

    // filter out the beams that make contact at their heads with a cut
    lstBeamsContact = ssBeams[i].filterBeamsContactCutHead(lstBeamsContact);

    // for each beam that makes contact, insert a script.
    for (int b=0; b<lstBeamsContact.length(); b++) {
        // find the contact plane
        Plane pln = ssBeams[i].findPlaneContactCutHead(lstBeamsContact[b]);

        String strScriptName = scriptName(); // name of this script, insert myself
        Vector3d vecUcsX(1,0,0);
        Vector3d vecUcsY(0,1,0);
        Beam lstBeams[0];
        Element lstElements[0];
        Point3d lstPoints[0];
        int lstPropInt[0];
        double lstPropDouble[0];
        String lstPropString[0];
    }
}

```

```

IstPoints.append(pln.ptOrg()); // will become _Pt0 because it is an E type
IstBeams.append(ssBeams[i]); // will become _Beam0
IstBeams.append(IstBeamsContact[b]); // will become _Beam1

// add to the Autcad database another instance of this TSL. Set the position,
grippers, list of beams, properties,...
TslInst tsI;
tsI.dbCreate(strScriptName, vecUcsX, vecUcsY, IstBeams, IstElements, IstPoints,
    IstPropInt, IstPropDouble, IstPropString); // create new instance
reportMessage("\nTsl inserted:" + strScriptName + " with beams: " + ssBeams[i] + " "
+ IstBeamsContact[b] );

} // end for each IstBeamsContact

// the current beam, with index i is not a good one: replace with an invalid
reference
ssBeams[i] = Beam();
} // end for each ssBeam

} // end go
eraselInstance(); // this instance will not stay in the DB
return;
} // if (_bOnInsert)

// Normal script.
Unit (1, "mm");
Vector3d vecX, vecY, vecZ;
vecX = _Y0;
vecY = _Z0;
vecZ = _X0;
Point3d ptOrg = _Pt0 + _H0 * 0.5 * vecY;
double dXwidth, dYheight, dZdepth, dAngle;
dXwidth = U(45);
dYheight = U(50);
dZdepth = U(28);
dAngle = U(0);
Dove dvF (ptOrg, vecX, vecY, -vecZ, dXwidth, dYheight, dZdepth, dAngle, _kFemaleEnd );
Dove dvM (ptOrg, vecX, vecY, vecZ, dXwidth, dYheight, dZdepth, dAngle, _kFemaleEnd );
_Beam0.addTool (dvF, 1);
_Beam1.addTool (dvM, 1);

vecX.vis(_Pt0);
vecY.vis(_Pt0);
vecZ.vis(_Pt0);

—end example]

```

## 14.3 PrPoint

The PrPoint class allows to prompt the user to select a point.

Since V23.0.75 it also supports a Jig (just in-time graphics). On Jig, the tsJ will call itself with state `_bOnJig` set to TRUE.

Similarly since V23.0.103 one can show graphics on [dragging a grip point](#).

Since V23.7.12 you can also set the snap mode. The snap model will be active during the go or goJig call, and will automatically be reset to its original state after the action.

Possible snap values that can be combined are:

```
_kOsModeEnd
_kOsModeMid
_kOsModeCen
_kOsModeNode
_kOsModeQuad
_kOsModeIns
_kOsModePerp
_kOsModeTan
_kOsModeNear
_kOsModeCentroid
```

Typical use would be:

```
ssP2.setSnapMode(TRUE, _kOsModeEnd | _kOsModeNear); // turn on only specific End
and Near snap modes
ssP2.setSnapMode(TRUE, 0); // turn off all snaps
ssP2.setSnapMode(FALSE, _kOsModeEnd | _kOsModeMid); // turn on, on top of existing
snap modes, also the End and Mid snap modes
```

---

```
class PrPoint {

    PrPoint(String strPrompt);
    PrPoint(String strPrompt, Point3d ptBase);

    int go(); // returns _kNone(1) or _kOk(3) (bReturnCancel default is FALSE, meaning _kCancel is
              not returned but execution ends)
    int go(int bReturnCancel); // (added V27.4.3 and V28.0.21) returns _kCancel (0) (if
                             bReturnCancel is TRUE), _kNone(1) or _kOk(3)
    Point3d value() const;

    // added since V23.0.75 and 22.1.108
    int goJig(String strExecuteKey, Map mapArg); // returns _kCancel (0), _kNone(1),
                                                 _kKeyword(2) or _kOk(3)
    int keywordIndex();

    void setSnapMode(int bResetSnap, int nSnapToSetOrAdd); // added since V23.7.12. If
                bResetSnap is FALSE, the current snap settings are augmented with additional
                settings.
}
```

---

```
PrPoint(String strPrompt);
PrPoint(String strPrompt, Point3d ptBase);
```

Constructor, make a new instance of the PrPoint class. Specify the question that will be asked on the command line: strPrompt. The second constructor specifies a base point, ptBase, to draw a line from during selection of the new point.

```
int go(int bReturnCancel); // (bReturnCancel default is FALSE, meaning _kCancel is not returned but execution ends)
```

The actual prompt action is done. Return the success of the operation by means of \_kCancel (0), \_kNone(1) or \_kOk(3). In fact \_kCancel is only returned if bReturnCancel is TRUE. If not, it automatically cancels the complete Tsl insert.

**Point3d value() const;**

Returns the location of the point selected.

```
int goJig(String strExecuteKey, Map mapArg); // added since V23.0.75 and 22.1.108
```

This method allows to draw just in-time graphics (Jig) while requesting the user to select a point. The jig graphics is collected from the Tsl by calling it with **\_bOnJig** set to TRUE, as well as passing in the proper execute key. Anything the Tsl displays, will be drawn for each mouse move.

Returns \_kCancel (0), \_kNone(1), \_kKeyword(2) or \_kOk(3). If you want to cancel the complete insert on cancel, then react on \_kCancel by calling **eraseInstance()**.

If you want to display also geometry with vis or visualize, make sure you call **\_ThisInst.setDebug(TRUE)**.

Beware, do not call **reportMessage** during **\_bOnJig**, since it will break the showing of the context menu while jiggling.

If a keyword is selected, the return of **goJig** is **\_kKeyword**. The 0-based index of the keyword selected is returned by **keywordIndex** in that case.

If the mapArgs contains an int key "**\_Highlight**", set to TRUE, all transparency will be suppressed during jiggling in 2dwireframe mode, and zero transparency will be supported in 3d modes.

---

[Example O-type:

```
if (_bOnInsert) {
    _Pt0 = getPoint(T("\n|Select start point:|"));
    Point3d ptLast = _Pt0;

    while (1) {
```

```

PrPoint ssP2(T("\n|Select next point:|"),ptLast);
    if (ssP2.go()==_kOk) { // do the actual query
        ptLast = ssP2.value(); // retrieve the selected point
        _PtG.append(ptLast); // append the selected points to the list
    of gripoints _PtG
    }
    else { // no proper selection
        break; // out of infinite while
    }
}
return;
}

```

*—end example]*

*[Example O-type:*

```

if (_bOnInsert)
{
    _Pt0 = getPoint(T("\n|Select start point:|"));
    Point3d ptLast = _Pt0;

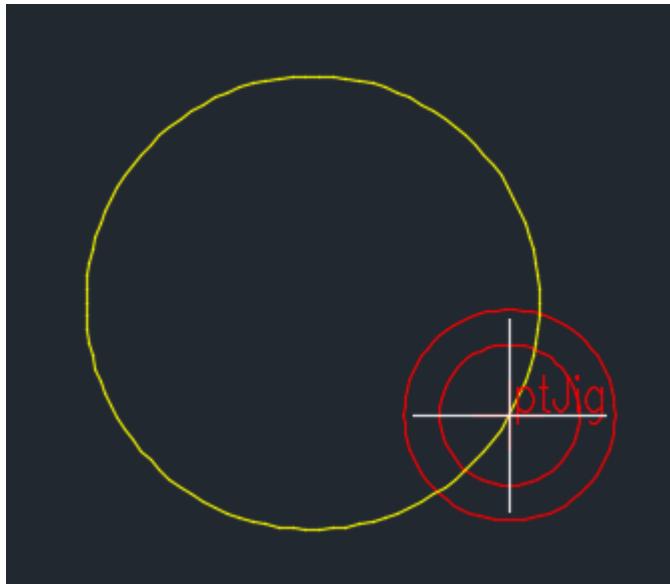
    PrPoint ssP2(T("\n|Select next point:|"), ptLast);
    int nRet = ssP2.go(TRUE); //do the actual query, argument TRUE
allows to receive a cancel.
    if (nRet == _kOk)
    {
        ptLast = ssP2.value(); //retrieve the selected point
        reportMessage(T("\n|Second point selected.|") + nRet);
    }
    else { //no proper selection
        reportMessage(T("\n|No second point selected.| ") + nRet);
    }

    return;
}

```

*—end example]*

*[Example O-type with Jigging:*



```

Unit(1, "mm");

if (!_bOnJig) // do not reportMessage uring OnJig, otherwise right
click keywords will not work
    reportMessage("\n" + scriptName() + ", execute key: " +
_kExecuteKey + ", bOnJig: " + _bOnJig + ", _bOnGripPointDrag: " +
_bOnGripPointDrag);

if (_bOnGripPointDrag && (_kExecuteKey=="_PtG0" ||
_kExecuteKey=="_Pt0"))
{
    Point3d ptBase = _Pt0;
    Point3d ptDrag = _PtG[0];

    _ThisInst.setDebug(TRUE); // sets the debug state on the dragging
TslInstance only, if you want to see the effect of vis methods
    ptDrag.vis(1);

    double dRad = Vector3d(ptDrag - ptBase).length();

    Display dpJ(1);
    PLine plCir;
    plCir.createCircle(ptDrag, _ZU, dRad/2);
    dpJ.draw(plCir);

    dpJ.color(2);
    plCir.createCircle(ptBase, _ZU, dRad);
    dpJ.draw(plCir);

    return;
}

String strJigAction1 = "strJigAction1";
if (_bOnJig && _kExecuteKey==strJigAction1)

```

```

{
    //Point3d ptBase = _Map.getPoint3d("ptBase");
    Point3d ptBase = _Map.getPoint3d("_PtBase"); // set by second
    argument in PrPoint
    Point3d ptJig = _Map.getPoint3d("_PtJig"); // running point

    _ThisInst.setDebug(TRUE); // sets the debug state on the jig
    only, if you want to see the effect of vis methods
    ptJig.vis(1);

    int nSingle = TRUE;
    if (_Map.hasInt("Single"))
        nSingle = _Map.getInt("Single");

    double dRad = Vector3d(ptJig - ptBase).length();

    Display dpJ(1);
    PLine plCir;
    plCir.createCircle(ptJig, _ZU, dRad/2);
    dpJ.draw(plCir);
    if (!nSingle)
    {
        plCir.createCircle(ptJig, _ZU, dRad/3);
        dpJ.draw(plCir);
    }

    dpJ.color(2);
    plCir.createCircle(ptBase, _ZU, dRad);
    dpJ.draw(plCir);

    return;
}

if (_bOnInsert)
{
    _Pt0 = getPoint(T("\n|Select start point|"));
    Point3d ptLast = _Pt0;

    PrPoint ssP2(T("|Select second point [Single/Double]|"),
    ptLast); // second argument will set _PtBase in map
    ssP2.setSnapMode(TRUE, _kOsModeEnd | _kOsModeNear); // bitwise or
    Map mapArgs;
    mapArgs.setInt("_Highlight", TRUE); // highlight mode will
    suppress transparency of all graphics drawn

    int nGoJig = -1;
    while (nGoJig != _kOk && nGoJig != _kNone)
    {
        nGoJig = ssP2.goJig(strJigAction1, mapArgs);
        reportMessage("\ngoJig returned: " + nGoJig);

        if (nGoJig == _kOk)
        {
}
}

```

```

        ptLast = ssP2.value(); //retrieve the selected point
        _PtG.append(ptLast); //append the selected points to the
list of grippoints _PtG
    }
    else if (nGoJig == _kKeyWord)
    {
        if (ssP2.keywordIndex() == 0)
            mapArgs.setInt("Single", TRUE);
        else
            mapArgs.setInt("Single", FALSE);
    }
    else if (nGoJig == _kCancel)
    {
        eraseInstance(); // do not insert this instance
        return;
    }
}

return;
}

// normal database resident behavior, just draw something
Display dp(-1);
addRecalcTrigger(_kGripPointDrag, "_Pt0");
for (int i = 0; i < _PtG.length(); i++)
{
    dp.draw(PLine(_Pt0, _PtG[i]));
    addRecalcTrigger(_kGripPointDrag, "_PtG"+i );
}
PLine plCir;
plCir.createCircle(_Pt0, _ZU, U(10));
dp.draw(plCir);
_Pt0.vis(1);

—end example]
```

## 14.4 Global insert functions

A number of global functions exist to perform easy user selections. For each function there is a default prompt implemented. Therefore, the TSL author can omit the prompt string.

```

Point3d getPoint();
Point3d getPoint(String strPrompt);

Element getElement();
Element getElement(String strPrompt);
Element getElement(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

Element getElementFromEntity();
Element getElementFromEntity(String strPrompt);
```

```
Beam getBeam();
Beam getBeam(String strPrompt);
Beam getBeam(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

Sheet getSheet();
Sheet getSheet(String strPrompt);
Sheet getSheet(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

Sip getSip();
Sip getSip(String strPrompt);
Sip getSip(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build 18.0.2

GenBeam getGenBeam();
GenBeam getGenBeam(String strPrompt);
GenBeam getGenBeam(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013
build 18.0.2

TslInst getTslInst();
TslInst getTslInst(String strPrompt);
TslInst getTslInst(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

Viewport getViewPort();
Viewport getViewPort(String strPrompt);

Opening getOpening();
Opening getOpening(String strPrompt);
Opening getOpening(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013
build 18.0.2

OpeningRoof getOpeningRoof(); // see OpeningRoof
OpeningRoof getOpeningRoof(String strPrompt);
OpeningRoof getOpeningRoof(String strPrompt, int bAllowSelectionInXref); // added
hsbCAD2013 build 18.0.2

ERoofPlaneOpening getERoofPlaneOpening(); // see ERoofPlaneOpening
ERoofPlaneOpening getERoofPlaneOpening(String strPrompt);
ERoofPlaneOpening getERoofPlaneOpening(String strPrompt, int bAllowSelectionInXref); // added
hsbCAD2013 build 18.0.2

String getString();
String getString(String strPrompt);

int getInt();
int getInt(String strPrompt);

double getDouble();
double getDouble(String strPrompt);

Entity getEntity(); // see Entity
Entity getEntity(String strPrompt);
```

```

Entity getEntity(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

NailLine getNailLine(); // see NailLine
NailLine getNailLine(String strPrompt);
NailLine getNailLine(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build
18.0.2

ERoofPlane getERoofPlane(); // see ERoofPlane
ERoofPlane getERoofPlane(String strPrompt);
ERoofPlane getERoofPlane(String strPrompt, int bAllowSelectionInXref); // added
hsbCAD2013 build 18.0.2

Wall getWall(); // see Wall
Wall getWall(String strPrompt);
Wall getWall(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build 18.0.2

Slab getSlab(); // see Slab
Slab getSlab(String strPrompt);
Slab getSlab(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013 build 18.0.2

// (added V28) Following methods will return _kOk, _kUpdate. The _kCancel automatically
returns control internally.
int showDialog(); // see also TsllInst, and Master slave insert.
int showDialogOnce();
int showDialog(String strCatalogEntryNameToStartFrom);
int showDialogOnce(String strCatalogEntryNameToStartFrom);
int insertCycleCount();

setCatalogFromPropValues(String strCatalogEntryName); // see also TsllInst
int setPropValuesFromCatalog(String strCatalogEntryName); // return TRUE if successful

Map mapWithPropValues(); // (added hsbCAD14.0.78) // see also TsllInst
Map mapWithPropValues(int bOnlyIndexAndValues); // (added V25.1.92) default FALSE, if
TRUE get back a compact map.
int setPropValuesFromMap(Map mapWithPropValues); // return TRUE if successful (added
hsbCAD14.0.78)

EntPLine getEntPLine(); // see EntPLine
EntPLine getEntPLine(String strPrompt);
EntPLine getEntPLine(String strPrompt, int bAllowSelectionInXref); // added hsbCAD2013
build 18.0.2

ElementRoof getElementRoof(); // see ElementRoof
ElementRoof getElementRoof(String strPrompt);
ElementRoof getElementRoof(String strPrompt, int bAllowSelectionInXref); // added
hsbCAD2013 build 18.0.2

```

---

**int** insertCycleCount();

Whenever a TSL insert command comes from the user, the TSL is inserted until escape is pressed. So the insert is repeated until the user explicitly ends it. When insert is

implemented inside the TSL script, the author can decide how this insert mechanism behaves. Each time a new instance is created, and inserted, until the user presses escape, or the script calls an `eraseInstance()`. The `insertCycleCount` keeps track of the number of inserts for that one command.

During the first cycle, the value is 1. If not during insert, the value is 0.

Writing

```
if (insertCycleCount()>1) { eraseInstance(); return; }
```

will assure a single instance being inserted.

```
int showDialog();  
int showDialogOnce();  
int showDialog(String strCatalogEntryNameToStartFrom);  
int showDialogOnce(String strCatalogEntryNameToStartFrom);
```

The `showDialog` and `showDialogOnce` allow to the user to set the properties through a dialog during insert (when `_bOnInsert` is TRUE) or special events. The `showDialog` will show each time the insert process takes place. The `showDialogOnce` on the other hand will show only once during one insert command. So the `showDialogOnce` will only show when the value returned from `insertCycleCount` is 1 (see above).

When no argument or an empty string is passed as argument, the last inserted values are presented as initial values in the dialog. These values are the values that where present the last time the ok button was pressed. There are also default values defined in the script, during the definition of the properties. That set of values is called the "`_Default`" set of values, and they are stored inside the catalog of values for that TSL script.

When the initial value of the properties needs to be retrieved from the catalog, one should use the catalog entry name as argument for the `showDialog` call. If the entry is not found in the catalog, the `_Default` set of values is chosen.

So the following are possible choices:

```
showDialog(""); // empty string means T("LastInserted")
showDialog(); // no argument means T("LastInserted")
showDialog(T("LastInserted"));
showDialog(T("|_Default|")); // the entry called _Default is used
showDialog("---"); // if entry is not found, then T("|_Default|") is used
showDialog("myCatalogEntry"); // specific named entry

showDialogOnce(""); // empty string means T("LastInserted")
showDialogOnce(); // no argument means T("LastInserted")
showDialogOnce(T("LastInserted"));
showDialogOnce(T("|_Default|")); // the entry called _Default is used
showDialogOnce("---"); // if entry is not found, then T("|_Default|") is used
showDialogOnce("myCatalogEntry"); // specific named entry
```

Since V27, the `showDialog` method returns either `kOk` or `kUpdate`. The `kCancel` is never returned, since control is directly returned internally if cancel is pressed.

```
int setPropValuesFromCatalog(String strCatalogEntryName);
```

This routine allows to set the property values to the values stored in the catalog for this TSL. This routine is only allowed when the hsbCAD application is loaded (so not in DBX mode). It works nicely together with the \_kExecuteKey value, passed during the Hsb\_ScriptInsert lisp call.

Returns TRUE, if the entry was found in the catalog, and the reading was successful. If a script does not implement insert, it can use this routine to read its catalog values during the event of \_bOnDbCreated. The execute key passed during the scriptinsert can be used then. The key could have been set through a command: (Hsb\_ScriptInsert "scriptname" "executekey")

```
if (_bOnDbCreated) setPropValuesFromCatalog(_kExecuteKey);
```

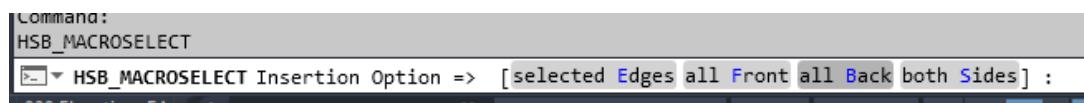
```
setCatalogFromPropValues(String strCatalogEntryName);
```

This routine allows writing the property values of this instance to the catalog. This routine is only allowed when the hsbCAD application is loaded (so not in DBX mode).

```
String getString();
String getString(String strPrompt);
```

When you declare the strPrompt like this, you get predefined options, clickable: clicking on one of the options returns the value.

```
String stInsertChoice = getString( T("|Insertion Option| => " +
" [ " + T("|selected Edges|") + "/" + T("|all Front|") + "/" + T("|all Back|") + "/" + T("|both Sides|") + "] "));
```



[Example O-type:

```
if (_bOnInsert) {

    _Pt0 = getPoint("\\nSelect start point"); // move the TSL instance to the selected point by
    setting _Pt0
    Point3d ptLast = _Pt0;

    for (int i =1; i<5; i++) {
        ptLast = getPoint("\\nSelect next point");
        _PtG.append(ptLast);
    }
}
```

```

    return;
}

```

—end example]

[Example O-type:

```

if (_bOnInsert){

    _Pt0 = getPoint ();
    _Element.append(getElement());
    return;
}

```

—end example]

[Example O-type, set the initial value of a property through user input:

```

String strInit; // declare the string outside the if (_bOnInsert) scope, for later use
if (_bOnInsert){
    strInit= getString(); // request the user to input a string
}
PropString pS(0,strInit,"String"); // use the inputed string for initialization of property
if (_bOnInsert) { // during insert, do not execute further
    return;
}

```

—end example]

[Example O-type:

```

String strInit = "Initial value";
PropString pS(0,strInit,"Prop name"); // use the inputed string for initialization of property

if (_bOnInsert){
    showDialog();
    _Pt0 = getPoint();
    return;
}

```

—end example]

[Example O-type with insert done in script, illustrating the \_kExecuteKey and setPropValuesFromCatalog

```

int arColorInd[]={1,2,3};
String arColorName[] = {T("red"),T("yellow"),T("green")};
PropString pColor(0,arColorName,"Color");

if (_bOnInsert {

    // the _kExecuteKey is set by the (Hsb_ScriptInsert "tslname" "ExecuteKey")
}

```

```

    reportMessage("\n_kExecuteKey is " + _kExecuteKey);

    // During insert, an execute key or any other string can be used to load the
    properties saved in the catalog.
    setPropValuesFromCatalog(_kExecuteKey); // might change the value of pColor

    _Entity.append(getEntity());
    _Pt0 = getPoint();
}

if (_Entity.length()==0) {
    eraseInstance();
    return;
}

Entity ent = _Entity[0];

int nEntColor = ent.color();
int nPropColor = arColorInd[arColorName.find(pColor,0)];

if (nEntColor!=nPropColor) {
    ent.setColor(nPropColor);
}

reportMessage("\ncolor of entity is " + ent.color());

—end example]

```

*[Example O-type with insert NOT done in script, illustrating the \_kExecuteKey and setPropValuesFromCatalog  
If the tsl is named "tslScript" , the the script could be inserted with (Hsb\_ScriptInsert "tslScript" "key").]*

```

int arColorInd[]={1,2,3};
String arColorName[] = {T("red"),T("yellow"),T("green")};
PropString pColor(0,arColorName,"Color");

if (_bOnDbCreated){ // Will be true during the first execution right after the appending to
the autocad database.

    // the _kExecuteKey is set by the (Hsb_ScriptInsert "tslname" "ExecuteKey")
    reportMessage("\n_kExecuteKey is " + _kExecuteKey);

    // During dbcreated, an execute key or any other string can be used to load the
    properties saved in the catalog.
    setPropValuesFromCatalog(_kExecuteKey); // might change the value of pColor
}

// Need to integrate tooling for the _Entity array not to be empty.
if (_Entity.length()==0) {
    return;
}

```

```

Entity ent = _Entity[0];

int nEntColor = ent.color();
int nPropColor = arColorInd[arColorName.find(pColor,0)];

if (nEntColor!=nPropColor) {
    ent.setColor(nPropColor);
}

reportMessage("\ncolor of entity is " + ent.color());

—end example]

```

[Example O-type with property that controls other properties:

```

U(1, "mm");

String pSelectorName = T("|Selector|");
PropInt pSelector(0, 0, pSelectorName);
pSelector.setControlsOtherProperties(TRUE);

String opmExtra = (pSelector == 0) ? "A" : "B";
setOPMKey(opmExtra);

String strLastChanged = _kNameLastChangedProp.length() > 0
    ? ", kNameLastChangedProp: '" + _kNameLastChangedProp + "'"
    : "";
String strExecuteKey = _kExecuteKey.length() > 0
    ? ", kExecuteKey: '" + _kExecuteKey + "'"
    : "";
reportMessage("\n"
    + scriptName() + " - " + _ThisInst.handle()
    + strLastChanged
    + strExecuteKey
    + ", bOnControlPropertyChanged: " + _bOnControlPropertyChanged
    + ", for value: " + pSelector);

PropString pS0(0, "aa", T("|Dependent prop|"));
PropDouble pD0(0, U(10), T("|Dependent prop|"));
PropDouble pD1(1, U(10), T("|A double prop|"));

void setReadOnlyFlagOfProperties()
{
    reportMessage(" selector: " + pSelector );
    if (pSelector == 0)
    {
        pS0.setReadOnly(FALSE);
        pD0.setReadOnly(_kHidden);
        pD1.setReadOnly(_kHidden);
        reportMessage(" and for value: " + pS0 + ".\n");
    }
    else
    {

```

```

        pS0.setReadOnly(_kHidden);
        pD0.setReadOnly(FALSE);
        pD1.setReadOnly(FALSE);
        reportMessage(" and for values: " + pD0 + ", " + pD1 + ".
\n");
    }

if (_bOnControlPropertyChanged)
{
    setReadOnlyFlagOfProperties();
    return;
}

if (_bOnInsert)
{
    String strLastInserted = T("|_LastInserted|");
    setPropValuesFromCatalog(strLastInserted); // need to set last
inserted
    reportMessage(" setPropValuesFromCatalog: '" +strLastInserted +
"'\n");
    setReadOnlyFlagOfProperties(); // need to set hidden state

    // "----" means not found, means do not change current values
    while (showDialog("----") == _kUpdate) // _kUpdate means a
controlling property changed.
    {
        setReadOnlyFlagOfProperties(); // need to set hidden state
    }

    _Pt0 = getPoint();
    return;
}

// always call the update of properties
setReadOnlyFlagOfProperties();

String strJustSD = T("|Just showDialog|");
addRecalcTrigger(_kContext, strJustSD );
if (_bOnRecalc && _kExecuteKey==strJustSD)
{
    // "----" means not found, means do not change current values
    while (showDialog("----") == _kUpdate) // _kUpdate means a
controlling property changed.
    {
        setReadOnlyFlagOfProperties(); // need to set hidden state
    }

    // on ok, also update hidden state
    setReadOnlyFlagOfProperties();
}

```

---

```

String strSDThisInst = T("|ShowDialog on ThisInst|");
addRecalcTrigger(_kContext, strSDThisInst );
if (_bOnRecalc && _kExecuteKey==strSDThisInst)
{
    // showDialog on _ThisInst does not show.
    ThisInst.showDialog();
}

String strSDOther = T("|ShowDialog on other|");
addRecalcTrigger(_kContext, strSDOther );
if (_bOnRecalc && _kExecuteKey==strSDOther)
{
    TslInst tslOther = getTslInst();
    tslOther.showDialog("----"); // "----" means not found, means do
not change current values
}

if (pSelector == -1)
{
    eraseInstance();
}
—end example]

```

## 14.5 Master slave insert

When different tsl's are to be applied depending on whatever condition, a master slave insert construction could be very handy. The idea is to write a special tsl that does the dbCreate of one, or a few other tsl's. The tsIs that implements insert is called the master. The slave ones are dbCreated by the master. Since they are dbCreated the \_bOnInsert will never be true for these slave tsl's. If the showDialog needs to be called on the slave tsl's, it is the responsibility of the master to call it.

The example below shows a master and a slave tsl. Here the master just inserts the slave that the user has selected in its own showDialog (see showDialog in [global insert functions](#)). After the slave is dbCreated, the showDialog of the slave is called (see showDialog in [TslInst](#)).

---

*[Example O-type, Insert implemented, should be called showDialogMaster:  
When used in conjunction with the showDialogSlave given below, the output could look like the following:*

*Running: showDialogSlave2  
\_bOnInsert=0, \_bOnDbCreated=1  
Calling showDialog from within showDialogSlave2  
Ended showDialog from within showDialogSlave2  
Calling showDialog from within showDialogMaster  
Running: showDialogSlave2*

```

_bOnInsert=0, _bOnDbCreated=0
Calling showDialog from within showDialogSlave2
Ended showDialog from within showDialogSlave2
Ended showDialog from within showDialogMaster
Running: showDialogSlave1
_bOnInsert=0, _bOnDbCreated=1
Calling showDialog from within showDialogSlave1
Ended showDialog from within showDialogSlave1
Calling showDialog from within showDialogMaster
Running: showDialogSlave1
_bOnInsert=0, _bOnDbCreated=0
Calling showDialog from within showDialogSlave1
Ended showDialog from within showDialogSlave1
Ended showDialog from within showDialogMaster

```

```

if (_bOnInsert) {

    String arSlaves[] = {"showDialogSlave1", "showDialogSlave2"};
    PropString strSlaveName(0, arSlaves, "Slave tsl name");
    showDialog(); // request the user to select the slave tsl

    String strScriptName = strSlaveName; // name of the script to
insert
    Point3d lstPoints[0];
    lstPoints.append(getPoint("Get point from within
"+scriptName())); // will become _Pt0

    int lstPropInt[0];
    lstPropInt.append(11); // will become PropInt with index 0

    Beam lstBeams[0];
    Element lstElements[0];
    double lstPropDouble[0];
    String lstPropString[0];

    TslInst tsl;
    tsl.dbCreate(strScriptName, Vector3d(1, 0, 0), Vector3d(0, 1, 0),
lstBeams, lstElements, lstPoints,
        lstPropInt, lstPropDouble, lstPropString); // create new
instance

    reportNotice("\nCalling showDialog from within "+scriptName());
    tsl.showDialog();
    reportNotice("\nEnded showDialog from within "+scriptName());

    return;
}

if (_bOnDbCreated) {
    eraseInstance(); // this instance will not stay in the DB, but the
eraseInstance will not abort the insert cycle here.
}

```

```
}
```

*—end example]*

*[Example O-type, Insert implemented (but not used by the master, should be called showDialogSlave1 and showDialogSlave2:*

```
reportNotice("\nRunning: "+scriptName());
reportNotice("\n\t_bOnInsert='"+_bOnInsert);
reportNotice(", _bOnDbCreated='"+_bOnDbCreated);

PropInt pI(0,0,"My int"); // just a simple property

// If this tsl is inserted through a dbCreate by the master, the
// _bOnInsert will never be TRUE.
if (_bOnInsert) { // Implement insert if you want to be able to
// insert the slave by itself.
    _Pt0 = getPoint();
    showDialog(); // This showDialog will never be reached when
// created by master
    return; // break the execution during insert here
}

// The current tsl runtime framework will ignore the showDialog call
// during normal
// execution, but since this might need to change in the future, it
// is not recommended
// that the showDialog is called on normal execution.
reportNotice("\n\tCalling showDialog from within "+scriptName());
showDialog(); // Please do not call showDialog on normal execution.
// Here it is only called to prove the point that the showDialog is not
// shown at all.
reportNotice("\n\tEnded showDialog from within "+scriptName());
```

*—end example]*

**Part**

**XV**

## 15 Predefined Variables

### 15.1 General predefined variables

To define a script, a list of predefined variables is available for the author. These predefined variables are given values, depending on the type of the script, the orientation of the UCS at time of insertion into the drawing, and depending on the properties of the beams added to the TSL instance.

For every beam that is appended to the script a number of variables are predefined (`_X0, _Y0, ...`). Even if the beam is not added, these variables can be used without runtime errors, although in that case, the variables will have no meaning.

Besides the predefined variables for a beam (`_X0, _Y0, ...`) a complete set of member functions of the Beam type is available, see [Entity](#), [GenBeam](#) and [Beam](#).

In the following, the `?` should be substituted with the index of the beam: `_X?` for `_Beam0` means `_X0`.

Each beam has some coordinate systems associated with it. The `_X?`, `_Y?` and `_Z?` vectors define a coordinate system which is set towards the connection. Therefor the T-type might have a different definition of it than the E-type. In general, the following is true.

The orientation of the `_X?` axis is from the midpoint of the beam towards the insert point of the script instance. The `_Z?` axis is the `_YF?` or `_ZF?` that is most aligned with the UCS Z axis at time of insertion. The `_Y?` is perpendicular to `_X?` and `_Z?`.

The `_X?, _Y? and _Z?` are recalculated every time the script is executed.

The values `_X?, _Y? and _Z?` can be changed during execution of the script. This can be useful if the initial values do not react as required. As an example, the following explains a few alternatives to set the `_X0` or `_X1` in a specific initial position.

1. Align the `_X0` vector to a calculated vector `vecA`:  
`if (_X0.dotProduct(vecA)<0) _X0 = -X0;`
2. Align the `_X0` with the UCS Z:  
`if (_X0.dotProduct(_ZU)<0) _X0 = -X0;`
3. Choose the `_X0` direction with a property value:  
`String strXDirections[]={"_XF0", "-_XF0"};`  
`PropString strXDir(0,strXDirections,"Align the X0 direction");`  
`if (strXDir==strXDirections[0]) _X0 = _XF0;`  
`else _X0 = -_XF0;`
4. If the direction needs to be kept as it is initially calculated, the OPM property value can be used to store the initial direction over multiple executions.  
`String strXDirections[0];`  
`if (_X0.dotProduct(_XF0)<0) {`  
 `strXDirections.append(" -_XF0");`  
 `strXDirections.append(" _XF0");`  
`}`  
`else {`  
 `strXDirections.append(" _XF0");`  
 `strXDirections.append("-_XF0");`  
`}`

```
PropString strXDir(0,strXDirections,"Align the _X0 direction");
if (strXDir== "_XF0") _X0 = _XF0;
else _X0 = -_XF0;
```

5. For a T-type the \_X1 axis can be oriented towards the smallest inner angle between the \_X1 and \_X0 directions. The following line can be included at the beginning of the script  
**if** (\_X1.dotProduct(\_X0)<0) \_X1 = -\_X1;
- 

### List of general predefined variables, available for all types

```
Point3d _Pt0; // the insertion point of the TSL instance. The value of this predefined is type dependent.

// UCS at the time of insertion. The vectors are however transformed during mirror, move, rotate.
Vector3d _XU, _YU, _ZU ;
Point3d _PtU ;

// WCS (world coordinate system)
Vector3d _XW, _YW, _ZW ;
Point3d _PtW ;

// ECS (entity coordinate system) It is predefined for every connection but can be altered
// by the user. ECS is used by the solid comparison during posnum generation.
// Predefined to _PtE = _Pt0; _XE = _X0; _YE=_Y0; _ZE=_Z0;
Vector3d _XE, _YE, _ZE ;
Point3d _PtE ;

// A number of entity arrays are managed as predefind variables
Beam _Beam[];
GenBeam _GenBeam[];
Sheet _Sheet[];
Sip _Sip[];
Element _Element[];
Viewport _Viewport[];

// The list of grippoints is now kept as an array. This array can be modified (appended,
// redefined) inside the script.
Point3d _PtG[];

// The list of new grippoints. This array can be modified (appended, redefined) inside the
// script.
Grip _Grip[]; // see Grip

// To call the member functions of the TSL instance itself, the reference to itself must be
// used:
TslInst _ThisInst; // see TslInst
```

---

**Map \_Map;** // The persistent key-value [Map](#) stored inside the `tsl`-instance.

**String \_DimStyles[];** // The string array of names of dimstyles.

[Example any type:

```
PropString pDimStyle(0, _DimStyles, "Dimension style");
reportMessage("\nMy dimstyle selected is:" + pDimStyle);
—end example]
```

**String \_LineTypes[];** // The string array of names of line types.

[Example any type:

```
PropString pLineType(0, _LineTypes, "Line type");
reportMessage("\nMy line type selected is:" + pLineType);
—end example]
```

**String \_BeamTypes[];** // The string array of names of [GenBeam](#) types.

[Example any type:

```
PropString pTypes(0, _BeamTypes, "Beam type");
int type = _BeamTypes.find(pTypes);
—end example]
```

**String \_HatchPatterns[];** // The string array of names of predefined [hatch](#) patterns.

[Example any type:

```
PropString pHatchPattern(0, _HatchPatterns, "Hatch pattern");
reportMessage("\nMy hatch pattern selected is:" + pHatchPattern);
—end example]
```

**String \_BlockNames[];** // The string array of names of predefined block names

[Example any type:

```
PropString pBlockName(0, _BlockNames, "Block name");
reportMessage("\nMy block name selected is:" + pBlockName);
—end example]
```

In the following substitute ? with the index of the beam. First index is 0, and maximum index is 9.

**Beam \_Beam?;** // so \_Beam0, \_Beam1,..., \_Beam9

**Sheet \_Sheet?;**

**Sip \_Sip?;**

**Element \_Element?;**

**Vector3d \_X?,\_Y?,\_Z?;** // axes of \_Beam?

**double \_H?, \_W?;** // dimensions of \_Beam? in \_Z? and \_Y? direction respectively (\_H? is always is the \_Z? direction)

// Fixed beam coordinate system

// This coordinate system is defined by the internally used coordinate system of the beam.

// It is unique for a given beam. It is required for use with extrusion profiles and curved timbers.

**Vector3d \_XF?, \_YF?, \_ZF?;** // internally fixed coordinate system of \_Beam?

**double \_HF?, \_WF?;** // dimensions of \_Beam? In \_ZF? and \_YF? direction.

**Line \_L?;** // axis line \_Beam?

**Point3d \_PtL?;** // reference point on line \_L? So \_L? is equivalent to Line \_L?(\_PtL?,\_X?);

```
double _LMin?, _LMax?; // minimum and maximum distance from _PtL? of beam ends in  
the _XF? direction.  
double _Len?; // length of _Beam?, in _X? direction.
```

---

**See also:**

[Predefined variables E-connection](#)  
[Predefined variables G-connection](#)  
[Predefined variables O-connection](#)  
[Predefined variables T-connection](#)  
[Predefined variables X-connection](#)

## 15.2 Predefined status variables

To indicate special execution modes of the TSL instance, special predefined variables will have a TRUE or FALSE value. For instance, the value of `_bOnInsert` will be TRUE during insert, and FALSE during normal execution.

`_bOnInsert` will be set to TRUE during the insert action into the drawing. The value will be FALSE otherwise. The value is only set to TRUE if it is selected from the TSL select dialog, or if it is inserted through the lisp command "Hsb\_ScriptInsert". So inserting a TSL through a copy action, through a dbCreate script action, a lisp command "Hsb\_ScriptInsertNoPrompt", or any other action will not set the value to TRUE.

`_bOnRecalc` will be set to TRUE when this instance is selected in the "Hsb\_Recalc" command. The value will be FALSE otherwise.

`_bOnDebug` will be set to TRUE when the debug OPM value is also TRUE. The value will be FALSE otherwise.

`_bOnElementDeleted` will be set to TRUE after an element to which this instance is attached, is deleted. The value will be FALSE otherwise.

`_bOnElementConstructed` will be set to TRUE after an element to which this instance is attached, is newly constructed. The value will be FALSE otherwise.

`_bOnElementRead` will be set to TRUE after an element to which this instance is attached, is read through HSB\_READCONSTRUCTION command. The value will be FALSE otherwise. (added hsbCAD2011 build 16.0.11)

`_bOnElementRecalc` will be set to TRUE when an element to which this instance is attached, is selected in the "Hsb\_Recalc" command. The value will be FALSE otherwise.

`_bOnViewportsSetInLayout` will be set to TRUE when the group tree console function called "Set viewports in layout" is fired. The value will be FALSE otherwise.

**\_bOnCheckSubAssembly0** and **\_bOnCheckSubAssembly1** will be set to TRUE when during a special insert done with the subassembly insert command.

**\_bOnCheckConRepair0** and **\_bOnCheckConRepair1** will be set to TRUE when during a special insert done with the construction repair command.

**\_bOnDbCreated** will be set to TRUE only during the first execution of the TslInst, after it has been appended to the Autocad database.

**\_bOnWriteEnabled** will be set to TRUE in normal execution, and will be set to FALSE, when the tsl instance is not allowed to write to the Autocad database. During dwg file open, the TSL instances are executed to calculate their graphical representation. But during this execution, the TSL is not allowed to change anything in the Autocad database: it is not allowed to do dbCreate, dbErase... actions. The predefined variable **\_bOnWriteEnabled** allows to detect if the script is running in the normal mode (**\_bOnWriteEnabled == TRUE**), or in this dwg file open mode (**\_bOnWriteEnabled == FALSE**).

**\_bOnElementListModified** will be set to TRUE when an execution of this Tsl is triggered by changing the **\_Element** list. The value will be FALSE otherwise. The execution with the value set to TRUE could be fired during the integrate tooling command, and during wall split command. See example below.

**\_bOnOpeningListModified** will be set to TRUE when an execution of this Tsl is triggered by changing the **\_Element** list. The value will be FALSE otherwise. The execution with the value set to TRUE could be fired during the integrate tooling command.

**\_bOnViewportListModified** will be set to TRUE when an execution of this Tsl is triggered by changing the **\_Element** list. The value will be FALSE otherwise. The execution with the value set to TRUE could be fired during the integrate tooling command.

**\_bOnGenerateShopDrawing** will be set to TRUE during the update or calculation of the shop drawings of one or a set of entities. The value will be FALSE otherwise.

**\_bOnOptionChanged** will be set to TRUE when an option definition has been toggled of which this tsl is part off. The value will be FALSE otherwise. (added since 14.0.30). See the [option system](#).

**\_bOnMapIO** will be set to TRUE during the configuration of a shopdraw tsl rule. The value will be FALSE otherwise. (added hsbCAD14.0.78)

### How can these status variables be used?

Wall and roof PANEL definitions have been extended with the possibility to have TSL's attached and executed automatically when the element is constructed. From the 'TSL' tab TSL's can be added by entering their name (or by browsing for the TSL file). The user can also specify an optional execute keyword for each TSL. This keyword is available in the **\_kExecuteKey** during the first TSL execution. If the TSL has not been imported into the drawing already, the software will scan the 'TSL' subfolder of the detail directory (including all subfolders) and load it (if found). When an element is panelized the software will automatically insert an instance of each linked TSL style(in case no such instance is already linked to the element). The software will automatically append the Element to the TSL **\_Element** array.

The first execution has the variable **\_bOnDbCreated** set to TRUE. This execution takes place during this call.

At the end of element generation, the tsl's are fired with the variable `_bOnElementConstructed` set to TRUE.

*[Example O-type, no action is performed, but the output is generated]*

```
reportNotice("\n----\n" + scriptName());
if (_Element.length()>0)
    reportNotice("\n_Element " + _Element[0].number() );
    reportNotice("\n_ExecuteKey: " + _kExecuteKey);
    reportNotice("\n_bOnDbCreated: " + _bOnDbCreated);
    reportNotice("\n_bOnElementConstructed: " + _bOnElementConstructed);
    reportNotice("\n_bOnElementRead: " + _bOnElementRead);
    reportNotice("\n_bOnElementDeleted: " + _bOnElementDeleted);

—end example]
```

*[Example O-type attached to ElementWallSF. When the wall is split, the Tsl reassigned to the closest element. This example illustrates the `_bOnElementListModified`]*

```
u(1,"mm");

if (_bOnInsert) {
    _Element.append(getElement());
    _Pt0 = getPoint();
    return;
}

// Find among all the _Element entries the element which is closest
to _Pt0.
// The _bOnElementListModified will be TRUE after wall-splitting-in-
length, or integrate tsl as tooling to element.
if (_bOnElementListModified && (_Element.length()>1)) // at least 2
items in _Element array
{
    reportMessage("\n"+scriptName()+" bOnElementListModified value:
"+_bOnElementListModified);

    // now find closest among these elements, for that project the
    point into the XY plane of each element
    int nIndexWinner = -1;
    double dDistWinner;
    for (int e=0; e<_Element.length(); e++) {
        Element ele = _Element[e];
        CoordSys csEle = ele.coordSys(_Pt0);
        Point3d ptE = csEle.ptOrg(); // project point into XY plane of
ele
        PlaneProfile ppEn = PlaneProfile(ele.plEnvelope());
        // but if point is not inside the envelope, find the closest
        point on the envelope
        if (ppEn.pointInProfile(ptE)!=_kPointInProfile) ptE =
ppEn.closestPointTo(ptE);
```

```

        double dDistE = Vector3d(ptE-_Pt0).length();
        if (nIndexWinner== -1 || dDistE<=dDistWinner) {
            nIndexWinner = e;
            dDistWinner = dDistE;
        }
    }
    if (nIndexWinner>0) { // the new winner is has not index 0 (or -1)
        Element elNew = _Element[nIndexWinner];
        Element elOld = _Element[0];
        reportMessage("\n"+scriptName()+" moved from "+elOld.number()+
" to "+elNew.number());
        _Element[0] = elNew; // overwrite 0 entry will replace the
existing reference to elem0
    }
}

// check running conditions
if (_Element.length()==0) return;
Element el = _Element[0];
if (!el.bIsValid()) return;

// move the _Pt0 to the element XY plane
CoordSys cs = el.coordSys(_Pt0);
_Pt0 = cs.ptOrg();

// just draw the element number this Tsl is attached to.
String str = el.number();
if (str=="") str = "--";

PLine plCir; plCir.createCircle(_Pt0,cs.vecZ(),U(300));

Display dp(-1);
dp.draw(str,_Pt0,cs.vecX(),cs.vecY(),0,0); // draw text
dp.draw(plCir); // draw circle

```

*—end example]*

### 15.3 Predefined variables T-connection

For a T type, there are always 2 beams: the | part (vertical in T) and the \_ part (horizontal in T). The | is the first beam that is selected, and corresponds with **\_Beam0**, it is the tenon beam. The \_ is the second one selected. It is the **\_Beam1**, the mortise beam. It is by definition like that.

Every beam, **\_Beam0**, **\_Beam1**, has a coordinate system associated with it: an **\_X?**, **\_Y?** and **\_Z?** - axis. **\_X0**, **\_Y0** and **\_Z0** belong to beam **\_Beam0**. The X-axis is in the length direction of the beam. For a T connection, the **\_X0** axis is always oriented towards the connection. (The direction of the internal axis is changed so that the orientation of the **\_X0** is always towards the connection).

The Z and Y-axis are perpendicular to the X-axis. The **\_Z0** direction is determined by the OPM value: "Align Z-axis". It can either be aligned with the internal Z-axis, the internal Y-axis, or their negative directions. Initially it is chosen most aligned with the UCS Z axis at time of insertion.

Having defined the **\_Z0** axis, and knowing that the **\_Y0** axis must be perpendicular to **\_Z0** and **\_X0**, we can only choose the side of the **\_Y0** axis, resulting in a positive or a negative axis system.

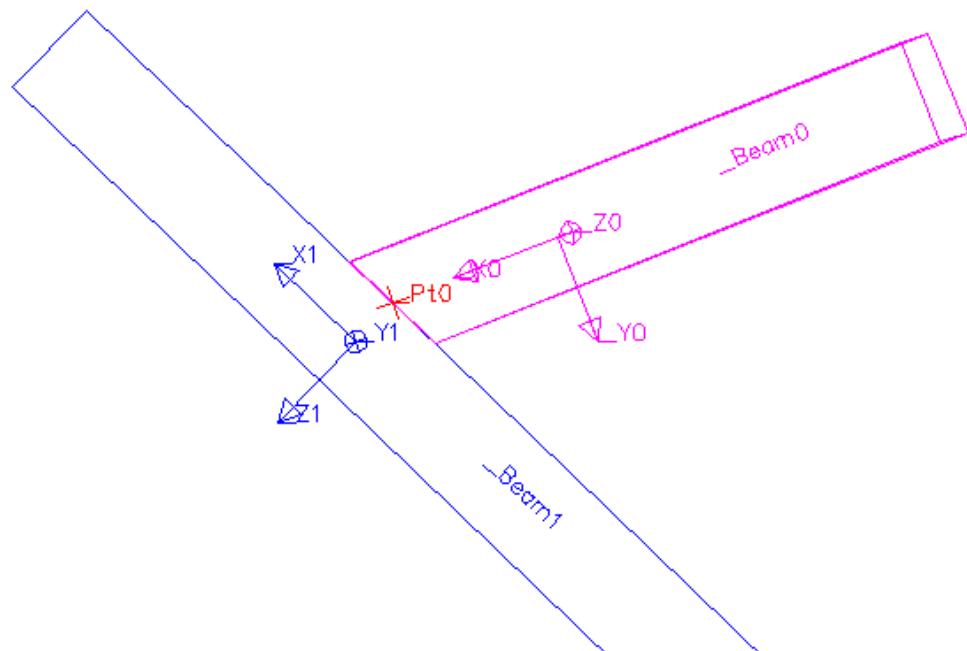
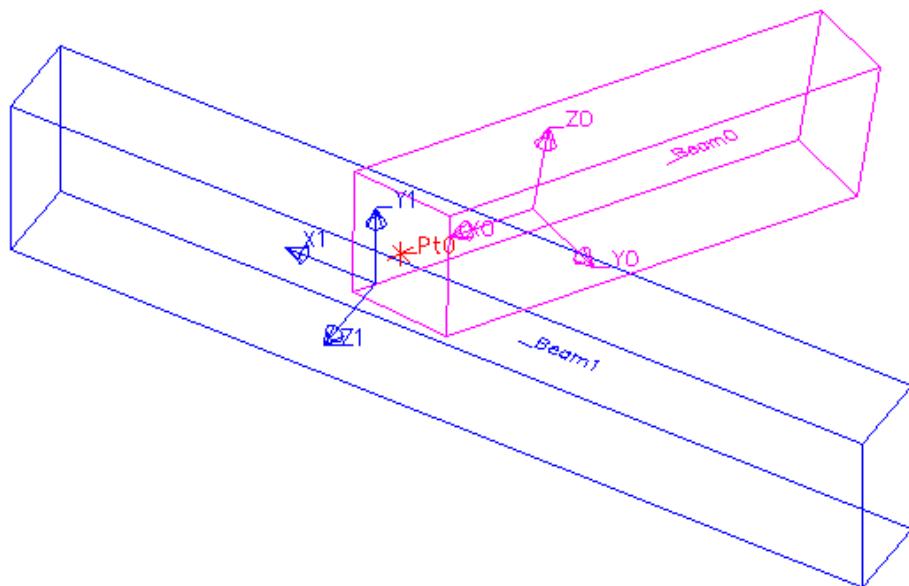
In the Z direction, the dimension of the beam is always the H (height), in the Y direction the dimension is always the W (width). So also **\_W0** and **\_H0** are changed when you select another "Align Z-axis".

The orientation of the mortise beam is better defined: the **\_X1**-axis is along the length direction of the beam, directed towards the WCS (1,0,0) vector at time of insertion. The **\_Z1** is always normal to the contact plane of the T-connection, in the direction of the positive **\_X0**. The **\_Y1** is perpendicular to **\_X1** and **\_Z1**. The direction of **\_Y1** is chosen most aligned with the positive UCS Z direction at the time of insertion.

The orientation of the axis system of **\_Beam1** is NOT influenced by the OPM values: "Align Z-axis" and "Flip Y-axis".

The image shows the initial orientation of the axis system of **\_Beam0**, and **\_Beam1**.

---



---

#### List of special predefined variables for T-type

**Beam \_Beam0;** // | beam in T  
**Beam \_Beam1;** // - beam in T

**Vector3d \_X0,\_Y0,\_Z0 ;** // axes of \_Beam0, respond to OPM

---

```

double _H0, _W0 ; // dimensions of _Beam0, respond to OPM

Vector3d _X1, _Y1, _Z1 ; // _Beam1
double _H1, _W1; // _Beam1

// Predefines on surface of connection
Vector3d _Yf, _Zf ; // Local coordinate system on surface
Plane _Plf; // connection plane

Point3d _Pt0; // intersection of _X0 with _Plf
Point3d _Pt1, _Pt2, _Pt3, _Pt4 ; // corner points of _Beam0 on the connection plane
Point3d _Ptc; // midpoint on surface of _Beam1, closest to _Pt0

```

---

[Example T-type used to make the image above:

```

U (1 ,"m m ");

_P t0 .v is (1 );

in t nC o lo r0 = 6 ;
P o in t3 d p t0 = _P t0 - U (1 0 0)*_X 0 ;
_X 0 .v is (p t0 , nC o lo r0 );
_Y 0 .v is (p t0 , nC o lo r0 );
_Z 0 .v is (p t0 , nC o lo r0 );

in t nC o lo r1 = 5 ;
P o in t3 d p t1 = _P t0 + 0 .5 *_H 1 *_Z 1 ;
_X 1 .v is (p t1 , nC o lo r1 );
_Y 1 .v is (p t1 , nC o lo r1 );
_Z 1 .v is (p t1 , nC o lo r1 );

C u t ct(_P t0 ,_Z 1 );
_B e a m 0 .a d D t o o l(ct,1 );

D i s p l a y d p 0 (nC o lo r0 );
P o in t3 d p t0 T = p t0 - U (1 0 0)*_X 0 ;
d p 0 .t e x tH e i g h t(U (9 ));
d p 0 .d r a w ("_B e a m 0 ",p t0 T ,-_X 0 ,-_Y 0 ,1 ,1 );

D i s p l a y d p 1 (nC o lo r1 );
P o in t3 d p t1 T = p t1 - U (1 0 0)*_X 1 ;
d p 1 .t e x tH e i g h t(U (9 ));
d p 1 .d r a w ("_B e a m 1 ",p t1 T ,-_X 1 ,-_Z 1 ,1 ,1 );

```

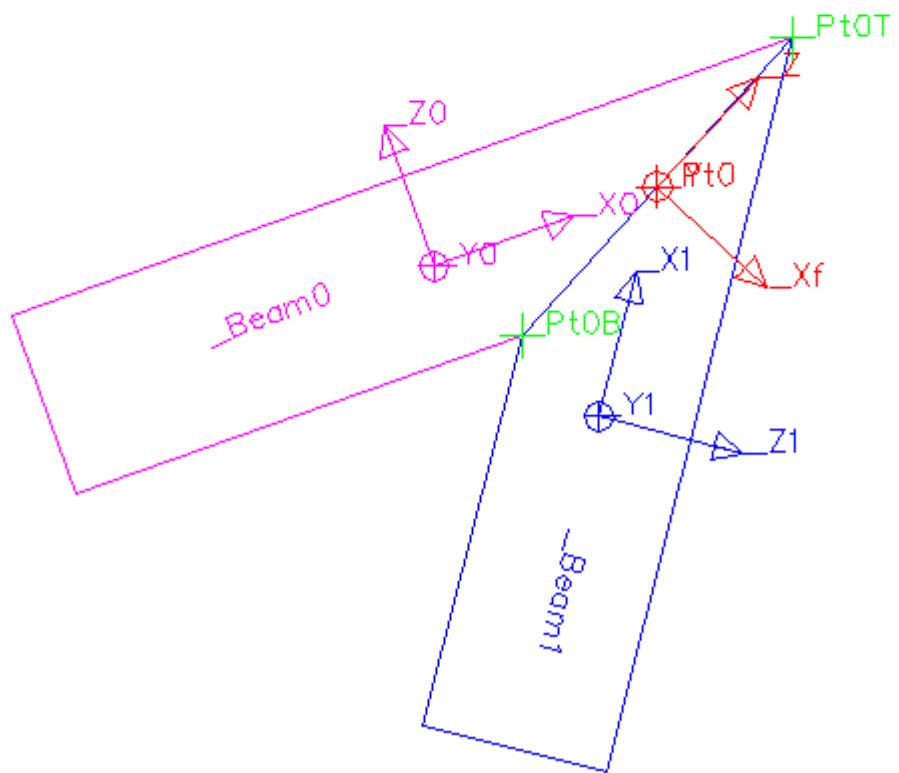
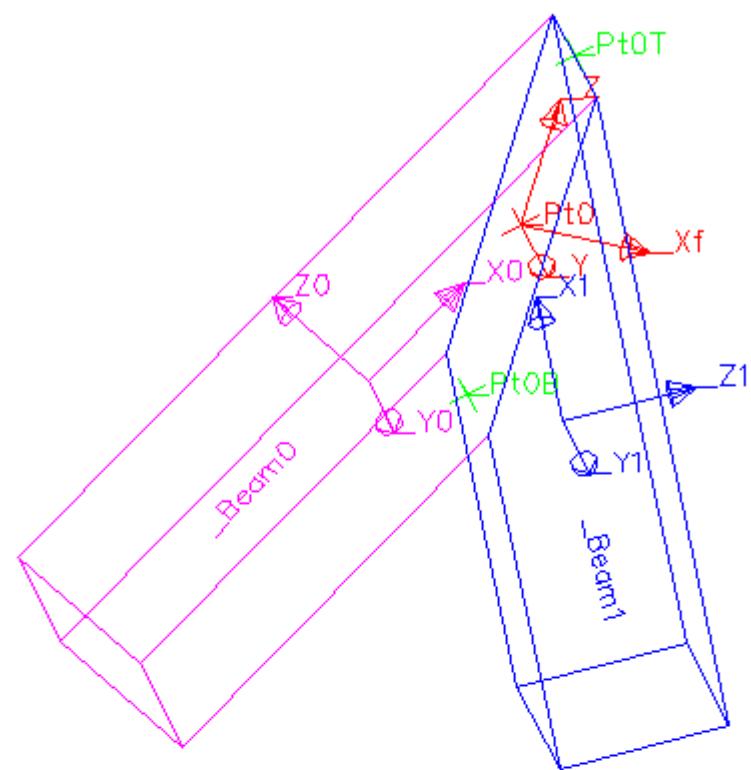
—end example]

## 15.4 Predefined variables G-connection

For a G-type, which is suitable for mitre or "gehrung" type of connection, there are always 2 beams.

The G-type has a special coordinate system \_X, \_Y and \_Z. The initial value of \_X, is most aligned with the UCS X at time of insertion. The \_Z vector is in the plane of the 2 beam axes, pointing from the bottom point to the top point. The \_Y vector is forming a positive axis system with the \_X and \_Z vectors.

The \_X0 and \_X1 vectors of the beams are pointing towards the tool. The \_Y0 and \_Y1 are most aligned with the \_Y vector. The \_Z0 and \_Z1 vectors are most aligned with the \_Z vector.



### List of special predefined variables for G-type

```
// Predefines on intersection surface
Vector3d _X,_Y,_Z; // Local coordinate system on surface, respond to OPM
Vector3d _Xf; // fixed, equal to _X or -_X, and most aligned with _X0, always
double _W, _H; // dimension of contact surface
Plane _Plf; // intersection plane, _X is normal

Point3d _Pt0; // intersection of _X0 with _Plf
Point3d _Pt0T, _Pt0B; // top and bottom points on intersection

// ECS (entity coordinate system)
// Predefined to _PtE = _Pt0; _XE = _X; _YE=_Y; _ZE=_Z;
Vector3d _XE, _YE, _ZE;
Point3d _PtE;
```

[Example G-type used to make the image above:

```
U (1,"m m ");

in t n C o lo r = 1;
_P t0 .v is (n C o lo r);
_X f.v is (_P t0 , n C o lo r);
_Y .v is (_P t0 , n C o lo r);
_Z .v is (_P t0 , n C o lo r);

in t n C o lo rM = 3;
_P t0 T .v is (n C o lo rM );
_P t0 B .v is (n C o lo rM );

in t n C o lo r0 = 6;
P o in t3 d p t0 = _P t0 - U (1 0 0)*_X 0 ;
_X 0 .v is (p t0 , n C o lo r0 );
_Y 0 .v is (p t0 , n C o lo r0 );
_Z 0 .v is (p t0 , n C o lo r0 );

in t n C o lo r1 = 5;
P o in t3 d p t1 = _P t0 - U (1 0 0)*_X 1 ;
_X 1 .v is (p t1 , n C o lo r1 );
_Y 1 .v is (p t1 , n C o lo r1 );
_Z 1 .v is (p t1 , n C o lo r1 );

C u t c t0 (_P t0 ,_X f);
_B e a m 0 .a d d T o o l(c t0 ,1 );
C u t c t1 (_P t0 ,_X f);
```

```

_B e a m 1 .a d d T o o l(c t1 ,1 );

D i s p l a y  d p 0 (n C o lo r0 );
P o i n t3 d p t0 T = p t0 - U (1 0 0 )*_ X 0 ;
d p 0 .t e x tH e i g h t(U (9 ));
d p 0 .d r a w ("_B e a m 0 ",p t0 T ,_ X 0 ,_ Z 0 ,1 ,1 );

D i s p l a y  d p 1 (n C o lo r1 );
P o i n t3 d p t1 T = p t1 - U (1 0 0 )*_ X 1 ;
d p 1 .t e x tH e i g h t(U (9 ));
d p 1 .d r a w ("_B e a m 1 ",p t1 T ,- X 1 ,_ Z 1 ,-1 ,1 );

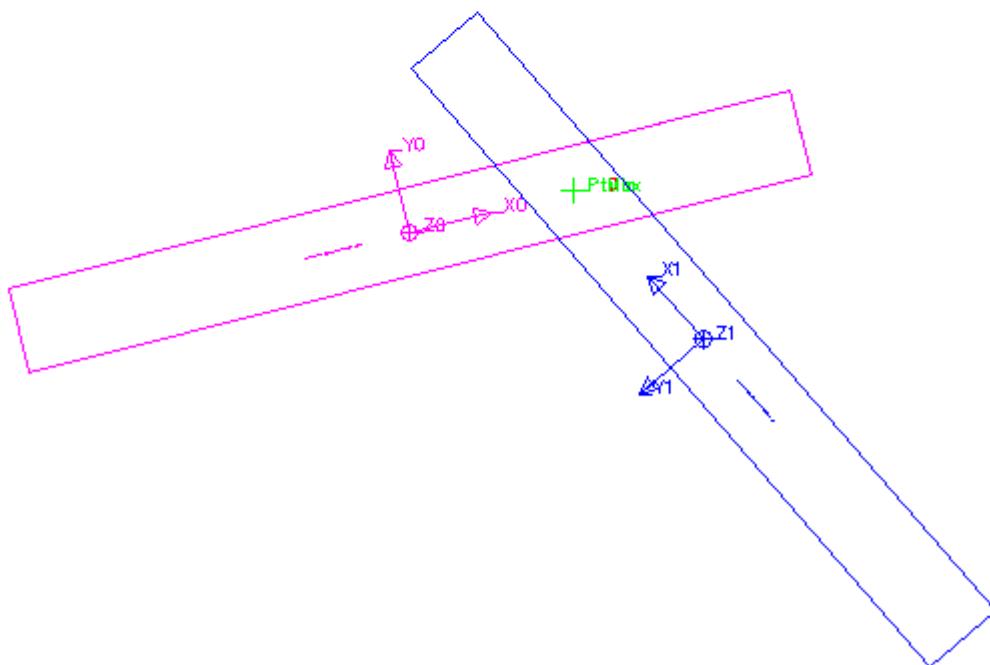
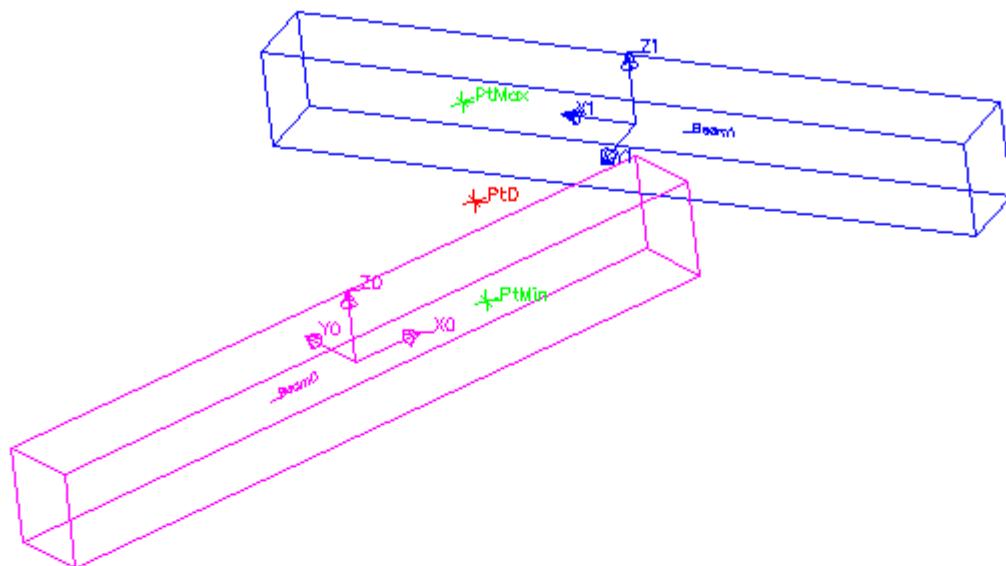
```

*—end example]*

## 15.5 Predefined variables X-connection

The X-type also requires 2 beams. The type is suitable for connections between crossing beams.

The \_X0 and \_X1 directions are pointing towards the script entity. The \_Z0 axis is the axis which is most aligned with the closest line between the two beams. So the \_Z0 axis oriented towards the connection. It is pointing towards \_Beam1 as closely as possible. The \_Y0 axis forms a positive axis system with the \_X0 and \_Z0 vectors. The \_Z1 is most aligned with the \_Z0 vector. The \_Y1 forms a positive axis system with the \_X1 and \_Z1 vectors.




---

#### List of special predefined variables for X-type

```
Beam _Beam0; // first beam
Beam _Beam1; // second beam
```

```
Vector3d _X0,_Y0,_Z0 ; // axes of _Beam0, _Z0 axis oriented towards the connection
Line _L0; // axis line _Beam0
```

---

```

Vector3d _X1, _Y1, _Z1 ; // _Beam1
Line _L1; // axis line _Beam1

// Predefines on interaction location
Point3d _PtMin; // Point on _L0 "closest to other beam along _Z0 direction"
Point3d _PtMax; // Point on _L1 "closest to other beam along _Z0 direction"
Point3d _Pt0; // Midpoint between _PtMin and _PtMax

// ECS (entity coordinate system)
// Initially set to _PtE = _Pt0; _XE = _X0; _YE= _Y0; _ZE= _Z0;
Vector3d _XE, _YE, _ZE ;
Point3d _PtE ;

```

---

[Example X-type used to make the image above:

```

 (1 , "m m " );

int nC o lo r = 1 ;
_ P t0 .v is (nC o lo r);

int nC o lo rM = 3 ;
_ P tM .in .v is (nC o lo rM );
_ P tM .ax .v is (nC o lo rM );

int nC o lo r0 = 6 ;
P o in t3 d p t0 = _Beam 0 .ptC en ();
_ X 0 .v is (p t0 , nC o lo r0 );
_ Y 0 .v is (p t0 , nC o lo r0 );
_ Z 0 .v is (p t0 , nC o lo r0 );

int nC o lo r1 = 5 ;
P o in t3 d p t1 = _Beam 1 .ptC en ();
_ X 1 .v is (p t1 , nC o lo r1 );
_ Y 1 .v is (p t1 , nC o lo r1 );
_ Z 1 .v is (p t1 , nC o lo r1 );

D is p la y d p 0 (nC o lo r0 );
P o in t3 d p t0 T = p t0 - u (1 0 0 )* _ X 0 ;
d p 0 .te x tH e ig h t(u (9 ));
d p 0 .d raw (" _ Beam 0 ", p t0 T , _ X 0 , _ Z 0 , 1 , 1 );

D is p la y d p 1 (nC o lo r1 );
P o in t3 d p t1 T = p t1 - u (1 0 0 )* _ X 1 ;
d p 1 .te x tH e ig h t(u (9 ));
d p 1 .d raw (" _ Beam 1 ", p t1 T , - _ X 1 , - _ Z 1 , - 1 , 1 );

```

---

*—end example]*

## 15.6 Predefined variables E-connection

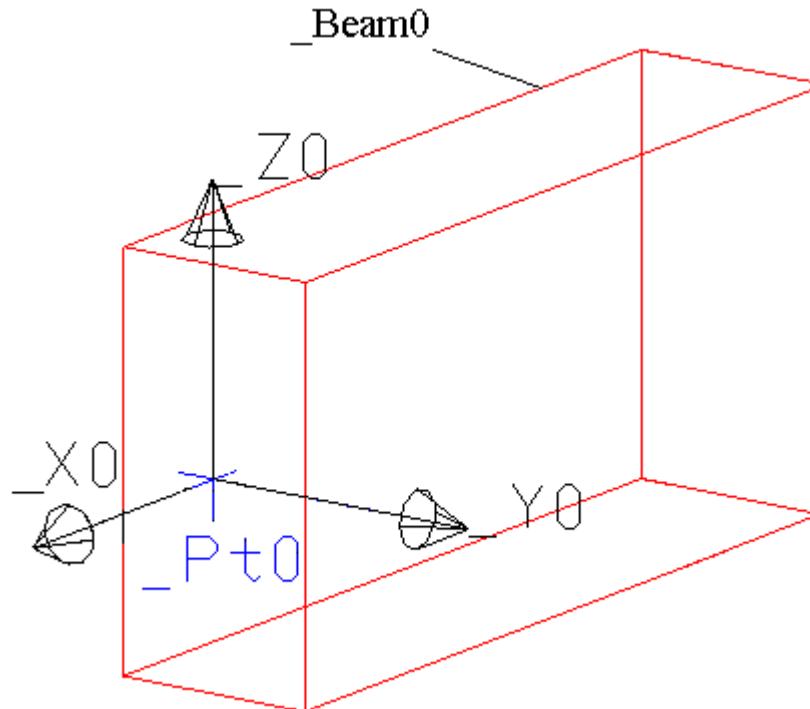
For an E macro, there is at least 1 beam: the first beam that is selected, and corresponds with **\_Beam0**.

The beam, **\_Beam0**, has a coordinate system associated with it: an X, Y and Z-axis. **\_X0**, **\_Y0** and **\_Z0** belong to beam **\_Beam0**.

The X-axis is in the length direction of the beam. The initial orientation of the **\_X0** axis is from the midpoint of the beam towards the insert point of the script instance. Once inserted, the **\_X0** orientation is kept, even if the script entity is moved to the other side of the midpoint of the beam. The **\_Z0** axis is the **\_YF0** or **\_ZF0** that is most aligned with the UCS Z axis at time of insertion. The **\_Y0** is perpendicular to **\_X0** and **\_Z0**. The **\_Z0** direction, is from then on depending on the OPM value.

The Z and Y-axis are perpendicular to the X-axis. The **\_Z0** direction is determined by the OPM value: "Align Z-axis". It can either be aligned with the internal Z-axis, the internal Y-axis, or their negative directions. Having defined the **\_Z0** axis, and knowing that the **\_Y0** axis must be perpendicular to **\_Z0** and **\_X0**, we can only choose the side of the **\_Y0** axis, resulting in a positive or a negative axis system.

In the Z direction, the dimension of the beam is always the H (height), in the Y direction the dimension is always the W (width). So also **\_W0** and **\_H0** are changed when you select another "Align Z-axis".



### List of special predefined variables for E-type

**Beam \_Beam0:**

```
Vector3d _X0,_Y0,_Z0 ; // axes of _Beam0
double _H0, _W0 ; // dimensions of _Beam0
```

```
Point3d _Pt0; // closest point to insertion point along the axis of _Beam0
```

The type of the TSL is important for the behaviour with respect to one or more beams. The E-type TSL is meant for a set of tools or metalpart to be attached to one beam. Typical use could be a set of end operations on a timber to shape it in a custom way. Another typical use is a surface tool, like a surface drill with some custom properties, or any other set of surface tools.

The specific behaviour of an E-type, is that it always stays attached to the \_Beam0, the primary beam. This behaviour is noticed when you select the primary beam, and move it with the autocad move command. Also copy, mirror, rotate, clip-copy,... of a beam with an attached E-type will trigger copy,mirror,... of these TSL's.

The following variables are influenced by these actions:

1. \_Pt0
2. \_X0, \_Y0, \_Z0
3. \_PtG[0], \_PtG[1],...
4. \_XU, \_YU, \_ZU
5. \_Map, with all relative points and vectors inside

The following actions can modify the above persistent variables:

- move the \_Pt0 grip point of the TSL itself ==> keep \_Pt0 on axis of beam, then move all (1 to 5).
- move/rotate the TSL entity itself ==> first move/rotate all (1-5), then correct the \_Pt0 only
- copy/mirror the TSL entity itself ==> first copy/mirror all (1-5), then correct the \_Pt0 only
- move/rotate the beam it is attached to (with or without the TSL in the selection set) ==> move/rotate all (1-5), no correction needed
- copy/mirror the beam it is attached to (with or without the TSL in the selection set) ==> copy/mirror all (1-5), no correction needed
- change the OPM properties "Align Z0", and OPM "Flip Y0" => only the \_X0, \_Y0, \_Z0 are adapted
- change the \_Pt0 from within the TSL itself, or from another TSL => only the \_Pt0 is modified, and corrected

As general rules, the following should be kept in mind:

- The points inside the \_Map and the \_PtG array react in the same way. The same transformation is applied to them.
- The vectors \_X0, \_Y0, \_Z0 are the only values that are controlled by the OPM align properties.
- Move/rotate/copy/mirror has 2 phases: the first one is to do the complete move/... to all the variables: \_Map, \_PtG,... The second phase is to correct only the \_Pt0 to move it to the axis line of the \_Beam0.

This last rule can be overwritten by the TSL author. You might want to keep the relative distance to the \_Pt0 of the \_PtG points fixed. To do that, you can use the \_Map to store a temporary copy of the \_Pt0, to figure out the additional correction that was applied to the \_Pt0. The correction can then be applied to the variables that the TLS is using. There are 2 examples below.

*[Example E-type to keep the relative distance to the \_Pt0 of the \_PtG points fixed.*

```
// Correct the position of the grippoints, _PtG[?], during move/rotate/_Pt0-grip-move
// such that the relative distance to _Pt0 stays the same.
if (_Map.hasPoint3d("Pt0")) { // check if this value was stored in the map
    Point3d ptOld = _Map.getPoint3d("Pt0"); // get value from map
    Vector3d vecMove = _Pt0-ptOld; // compose translation vector
    for (int p=0; p<_PtG.length(); p++) { // move all grippoints
        _PtG[p].transformBy(vecMove);
    }
}
_Map.setPoint3d("Pt0",_Pt0);
```

*—end example]*

*[Example E-type to keep the relative distance to the \_Pt0 of the \_PtG points fixed, and to react on the OPM properties.*

```
// Correct the position of the grippoints, _PtG[?], during move/rotate/_Pt0-grip-move
// such that they react and follow to the OPM align property values.
if (_Map.hasVector3d("X0")) { // check if this value was stored in the map
    Point3d ptOld = _Map.getPoint3d("Pt0"); // get values from map
    Vector3d X0 = _Map.getVector3d("X0");
    Vector3d Y0 = _Map.getVector3d("Y0");
    Vector3d Z0 = _Map.getVector3d("Z0");
    // compose coordinate transformation from stored (and auto transformed) coord system
    //
    CoordSys trans; trans.setToAlignCoordSys(ptOld,X0,Y0,Z0,_Pt0,_X0,_Y0,_Z0);
    for (int p=0; p<_PtG.length(); p++) {
        _PtG[p].transformBy(trans);
    }
}
_Map.setPoint3d("Pt0",_Pt0);
_Map.setVector3d("X0",_X0);
_Map.setVector3d("Y0",_Y0);
_Map.setVector3d("Z0",_Z0);
```

*—end example]*

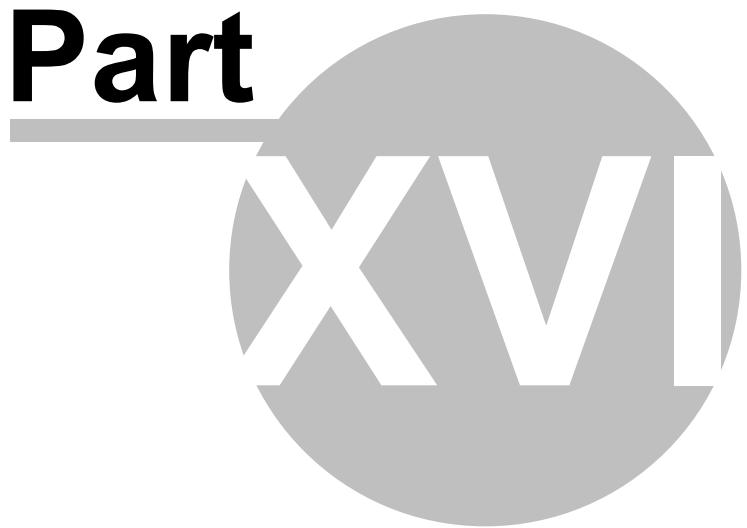
## 15.7 Predefined variables O-connection

The O-type does not require any beams. It can be located anywhere in space. Ofcourse, as all the other types, it may have references to beams.

**List of special predefined variables for O-type**

```
Point3d _Pt0; // insertion point  
  
// ECS (entity coordinate system)  
// Initially set to _PtE = _Pt0; _XE = _XU; _YE=_YU; _ZE=_ZU;  
Vector3d _XE, _YE, _ZE ;  
Point3d _PtE ;
```

**Part**



## 16 Drawing by TSL

### 16.1 Display

To build the graphical representation of a TSL the display system, in combination with MetalPart should be used. The following possibilities are available.

- Already from the beginning of the existence of TSL, there is the possibility to define a MetalPart. A MetalPart is always shown, if defined.
- **Obsolete:** Hardware can be visualized by calling the visualize method. Visualizing hardware is independent of the debug state.
- **Obsolete:** Using the visualize method on a block is also possible, and as the hardware, independent of the debug state.
- Point and Vectors can be visualized in debug, using the textstyle from the dimstyle that is specified in the Hsb\_settings->dimension->TSL. The height of the text, that also determines the scaling of the vectors, can be adjusted by the "Change text height" in the TSL Editor window. If the value is set to 0 of the "Change text height" in the TSL Editor window, the textheight from the dimstyle is used.
- The preferred way of building a graphical representation of a TSL is the display system.

Multiple displays can be specified in one script. A display has always an active color, and active textstyle and dimstyle, an active linetype and possibly an active element zone. All these active properties can be changed. If this is done, only new things drawn with the display will use the modified properties.

A display can also have one or multiple display directions set. A display direction will be active for the complete display, even if set at the end of the script. If no display direction is set, the display is shown independent of the display direction.

Also the hardware and the blocks can be displayed in this way.

When displaying text, one can specify to draw the text always parallel to the device/screen. The following values can be used:

- \_kModel ==0: means aligned in the model
- \_kDevice ==1: means parallel to the screen/device
- \_kDeviceX ==2: means parallel to the screen/device, and also horizontal

When displaying a PlaneProfile, the nDrawMode can be one of the following

- \_kDrawAsCurves ==0: (default) means that only the lines of the profile will be displayed
- \_kDrawAsShell ==1: means that during shading, or hiding, the profile is seen as a shell
- \_kDrawFilled ==2: means that not only during shade, but also in wireframe, the profile is filled.

When filing dwg or dxf from a Display (writeToDxfFile or writeToDwgFile) the nVersion can be one of the following (added 16.2.14)

- \_kVersionCurrent ==0: (default) means the current version
- \_kVersion2000 == 2000: Acad version 2000 compatible
- \_kVersion2004 == 2004: Acad version 2004 compatible
- \_kVersion2007 == 2007: Acad version 2007 compatible
- \_kVersion2010 == 2010: Acad version 2010 compatible

**See also:**

[Dim and DimLine](#)

[Display text](#)

[Display on Element](#)  
[Display in layout](#)  
[BOM in layout](#)

---

```
class Display {

    Display(int colorIndex); // colorIndex == -1 takes the entity its color

    color(int colorIndex); // set color to color index, do not change transparency, color index -1 takes
    the entity its color
    transparency(int transparencyPercent); // (added 23.4.4) set transparency, but does not change
    the color, transparency -1 takes the entity its transparency
    trueColor(int rgb); // (added 23.0.75) Use global method rgb(255,0,0) for red, but does not
    change the transparency
    trueColor(int rgb, int transparencyPercent); // (added 23.0.75)

    dimStyle(String strDimStyleName); // if dLinearScale is not specified, it is reset to the one
    specified in the dimstyle.
    dimStyle(String strDimStyleName, double dLinearScale); // if dLinearScale set to 0, the one
    specified in the dimstyle is used.

    lineType(String strLineTypeName); // there is a predefine LineTypes
    lineType(String strLineTypeName, double dLineTypeScale); // (added hsbCAD19.0.37) if
    dLineTypeScale is not specified (or equal to -1), it is set to the one of the tsl instance.

    layer(String strLayerName); // (added in hsbCad13.2.89)
    elemZone(Element el); // Element
    elemZone(Element el, int nZoneIndex, char cZoneCharacter);

    addViewDirection(Vector vecDirectionTowardsViewer);
    addHideDirection(Vector vecDirectionTowardsViewer);
    textHeight(double dTextHeight);

    showInDxa(int bSet);
    showInTslnst(int bSet);
    showDuringSnap(int bSet); // (added in V26.4.5 and V25.2.22) default TRUE, if set to FALSE,
    the display will not trigger snap points.
    showInShopDraw(Entity ent);
    showInDispRep(String strDispRepName);
    showInDispComp(String strDispCompName); // added since V27.3.7 and V28.0.9, used by
    fastener graphics.

    // Beware: the following methods may only be fired during _bOnInsert or _bOnRecalc
    // (with _kExecuteKey is not empty).
    String writeToDxfFile(String strFullDxfFileName, int bRelativeToEcs) const;
    String writeToDwgFile(String strFullDwgFileName, int bRelativeToEcs) const;
    String writeToDxfFile(String strFullDxfFileName, int bRelativeToEcs, int nVersion) const; //
    (added 16.2.14) nVersion default is _kVersionCurrent
    String writeToDwgFile(String strFullDwgFileName, int bRelativeToEcs, int nVersion) const; //
    (added 16.2.14) nVersion default is _kVersionCurrent
```

---

```

String writeToDxfFile(String strFullDwgFileName, int bRelativeToEcs, int nVersion, int
bExplodeExtraLevel) const; // 22.1.132 and 23.5.9 default bExplodeExtraLevel is FALSE
String writeToDwgFile(String strFullDwgFileName, int bRelativeToEcs, int nVersion, int
bExplodeExtraLevel) const; // 22.1.132 and 23.5.9 default bExplodeExtraLevel is FALSE

draw(PLine plPoly); // PLine
draw(Dim dimension); // Dim
draw(DimAngular dimension); // DimAngular
draw(Hardware hardware, Point3d ptLocation, Vector3d vecX); // see Hardware
draw(HardWrComp hardWrComp, Point3d ptLocation, Vector3d vecX); // see HardWrComp
(added hsbCAD2011, build 16.0.18)
draw(Body body); // Body
draw(LineSeg seg); // LineSeg. From v20.1.65 on, LineSegs with zero length are drawn
as Autocad points.
draw(LineSeg[] arSegs); // draw all segments at once LineSeg

draw(PlaneProfile plProf); // PlaneProfile
draw(PlaneProfile plProf, int nDrawMode);
draw(PlaneProfile plProf, Hatch hatchInfo); // Hatch
draw(PlaneProfile plProf, int nDrawMode, int nTransparencyPercent); // default
nTransparencyPercent = 0 (added v21.0.105)

draw(Block block, Point3d ptLocation, Vector3d vecX, Vector3d vecY, Vector3d vecZ); // Block
draw(Block block, Point3d ptLocation); // there is a predefine \_BlockNames

draw(String strText, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag,
double dYFlag);
draw(String strText, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag,
double dYFlag, int nTextOrientation);

draw(Entity ent, CoordSys transFromEntToDisplay, String strDisplayRepName); // Entity.
Example below
draw(TruckDefinition truck, CoordSys transToDisplay); // draw the display of the
TruckDefinition into the current display (added in V26)

double textLengthForStyle(String strText, String strDimStyle) const;
double textHeightForStyle(String strText, String strDimStyle) const;
double textLengthForStyle(String strText, String strDimStyle, double dTextHeight) const; // added hsbCAD2017 build 21.4.31
double textHeightForStyle(String strText, String strDimStyle, double dTextHeight) const; // added hsbCAD2017 build 21.4.31

// special method, for internal use only
fillDisplayFromEntity(Entity ent, String strSpecial, int nIndex, Point3d ptLocation, Vector3d
vecX, Vector3d vecY, Map mapSettings);

drawQR(String strQRCode, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double
dXFlag, double dYFlag, double dSize); // (added in build v21.3.101) see also
String::qrEncode and String::qrDecode.

drawHsbLogo(int nColorSet, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double
dXFlag, double dYFlag, double dSize); // (added in build v23.3.10)

```

```

drawImage(String strEncodedImage, Point3d ptLocation, Vector3d vecX, Vector3d vecY,
double dXFlag, double dYFlag, double dSizeX, double dSizeY); // (added in build v23.4.4).
Images can be encoded by the String::imgEncodeImageFromFile method, which can
then be drawn by this method.
};

```

---

### **Display(int colorIndex);**

This constructor of a Display builds a new display, with an active color given by the colorIndex. If the color is set to -1, the color of the instance will be used.

```

color(int colorIndex); // set color to color index, do not change transparency, color index -1 takes the
entity its color
transparency(int transparencyPercent); // (added 23.4.2) set transparency, do not change color,
transparency -1 takes the entity its transparency
trueColor(int rgb); // (added 23.0.75) Use global method rgb(255,0,0) for red, do not change
transparency
trueColor(int rgb, int transparencyPercent); // (added 23.0.75)

```

The color that is active in the Display can be set in 2 ways. Through the autocad color index (eg 1 means red). Or through setting the rgb value. For that you should use the [global method rgb\(int r, int g, int b\)](#) method.

The transparencyPercent is a value in the range [-1,100]. -1 means take the entity its transparency value. 0 not transparent, and 100 is fully transparent. The default value is -1.

```

draw(PlaneProfile plProf);
draw(PlaneProfile plProf, int nDrawMode);
draw(PlaneProfile plProf, int nDrawMode, int nTransparencyPercent); // default
nTransparencyPercent = 0 (added v21.0.105)

```

When displaying a PlaneProfile, the nDrawMode can be one of the following

- \_kDrawAsCurves ==0: (default) means that only the lines of the profile will be displayed
- \_kDrawAsShell ==1: means that during shading, or hiding, the profile is seen as a shell
- \_kDrawFilled ==2: means that not only during shade, but also in wireframe, the profile is filled.

In case of \_kDrawFilled an optional parameter nTransparencyPercent can be added,  
0 means not transparent (default)  
100 fully transparent

```

addViewDirection(Vector vecDirectionTowardsViewer);
addHideDirection(Vector vecDirectionTowardsViewer);

```

Each display keeps a list of view directions and a list of hide directions. When the viewing direction in the viewport coincides (is codirectional) with one of the specified view directions added with the addViewDirection instruction, the display will be shown inside the viewport. A display that has no vectors added to the list of view directions, will always be shown. But before the display is actually displaying, the added hide directions are checked agains the viewing

direction in the viewport. If the viewing direction matches one of these hide directions, the display will not show.

Remember, multiple displays can be specified inside one script. Each displaying different things.

*[Example E-type that requires 2 blocks to be present: "circle" and "cross":*

```
Unit(1,"mm");

Block blCircle("circle");
Block blCross("cross");

Display dp1(-1); // use color of entity
dp1.addHideDirection(_ZW); // show always except for directions _ZW and -_ZW
dp1.addHideDirection(-_ZW);
dp1.draw(blCross,_Pt0+U(10)*_X0);

Display dp2(-1); // use color of entity
dp2.addViewDirection(_ZW); // only show for directions _ZW and -_ZW
dp2.addViewDirection(-_ZW);
dp2.draw(blCircle,_Pt0+U(10)*_X0);
```

*--end example]*

**showInDxa(int bSet);**

When the **showInDxa** is called with TRUE as argument, the display will also output into the dxa file structure, allowing graphics, and text to be saved in the file. The default value of this state is FALSE.

**showInTslInst(int bSet);**

If you want to suppress a display to be shown in the normal autocad worldDraw, or viewportDraw, you need to call the **showInTslInst** with argument FALSE. The default value of this state is TRUE. A typical use is in combination with **showInShopDraw**, where the Display that is made for shopdrawing purpose, is not showing in normal display. For easy debugging it can be called :

**showInTslInst(\_bOnDebug);**

**showInShopDraw(Entity ent);**

To create a display that will ouput graphics in the shopdrawing of another entity, eg, a beam or a panel, one has to pass the entity as argument to this routine. A display can only have one Entity set. To suppress the display in the normal representation of the TSL, use the **showInTslInst** routine with FALSE as argument.

To make the Display only appear in one particular view of the shopdrawing, the view direction needs to be set with **addViewDirection**. The following coordinate transformations are applicable for the shopdrawing of a beam. The third vector is the view direction to be set:

Top: beamX, beamY, beamZ

Top in sloped view: view direction: \_ZW

Bottom: beamX, -beamY, -beamZ

Bottom in sloped view: view direction: -\_ZW  
Front: beamX, beamZ, -beamY  
Back: beamX, -beamZ, beamY // below bottom view  
Back: -beamX, beamZ, beamY // left of front view  
Right: beamY, beamZ, beamX  
Left: -beamY, beamZ, -beamX  
Isometric view direction: beamZ-beamX-beamY  
Isometric X: not known  
Isometric Y: not known

```
double textLengthForStyle(String strText, String strDimStyle) const;
double textHeightForStyle(String strText, String strDimStyle) const;
double textLengthForStyle(String strText, String strDimStyle, double dTextHeight) const; //
    added hsbCAD2017 build 21.4.31
double textHeightForStyle(String strText, String strDimStyle, double dTextHeight) const; //
    added hsbCAD2017 build 21.4.31
```

For a given string strText, and a given dimension style strDimStyle, these routines calculate the length and height, in drawing units, of the text when it would be displayed.  
Since hsbCAD2017 build 21.4.31, these methods support also multi-line.

```
showInDispRep(String strDispRepName);
```

By default, the tsl instance will only show in the adt-display-representation called Model. If you want the contents of this display to show only in the a particular adt-display-representation, the showInDispRep should be called. The argument must match (case insensitive) the name of the adt-display-representation. In this case, the contents of the Display will not show anymore in the adt-display-representation Model.

There can only be one strDispRepName name set per Display. Setting a second one, will overwrite the first one.

The contents of a Display will show if (strDispRepName is empty (by default) AND it is the Model display representation) OR the display representation name matches the strDispRepName.

The strDispRepName should be in the list of \_ThisInst.dispRepNames() (see [Entity](#)) to have any effect.

```
dimStyle(String strDimStyleName); // if dLinearScale is not specified, it is reset to the one specified
in the dimstyle.
dimStyle(String strDimStyleName, double dLinearScale); // if dLinearScale set to 0, the one
specified in the dimstyle is used.
```

A display has an active dimstyle. It can be set by this function. The argument should be one of the entries of the array [\\_DimStyles](#). Inside the dimstyle one can set an scaling eg to draw dimensions in paper space to dimension distances from modelspace. To overwrite the value set inside the dimstyle, the dLinearScale can be used.

[Example:

```
    CoordSys ms2ps = vp.coordSys();
    CoordSys ps2ms = ms2ps; ps2ms.invert(); // take the inverse of ms2ps
    double dScale = ps2ms.scale();
    dp.dimStyle("HSB-vp",dScale);
—end example]
```

```
lineType(String strLineTypeName);
```

A display has an active line type. It can be set by this function. The argument should be one of the entries of the array **\_LineTypes**.

```
layer(String strLayerName); // (added in hsbCad13.2.89)
```

Specify the layer on which the graphics is drawn. If the layer does not exist in the drawing, it is created automatically. When the color index 256 is used, the ColorByLayer will be active. In order for the graphics to show, the tsl entity itself must be drawn on a visible layer.

```
String writeToDxfFile(String strFullDxfFileName, int bRelativeToEcs) const
String writeToDwgFile(String strFullDwgFileName, int bRelativeToEcs) const
String writeToDxfFile(String strFullDxfFileName, int bRelativeToEcs, int nVersion) const; // (added 16.2.14) nVersion default is _kVersionCurrent
String writeToDwgFile(String strFullDwgFileName, int bRelativeToEcs, int nVersion) const; // (added 16.2.14) nVersion default is _kVersionCurrent
```

Everything that is drawn in a display can be dumped into a DXF or DWG file. The filename needs to be given as argument, and needs to have the correct extension. If the bRelativeToEcs is set to TRUE, then the contents of the DXF file will be relative to the coordinate system of this tsl instance. If the bRelativeToEcs is FALSE, then the graphics content will appear as drawn by the Display.

Beware: may only be fired during **\_bOnInsert** or **\_bOnRecalc** (with **\_kExecuteKey** is not empty). From 17.1.8 on, it is also allowed to call during **\_bOnDbCreated**.

When filing dwg or dxf from a Display (writeToDxfFile or writeToDwgFile) the nVersion can be one of the following (added 16.2.14)

```
_kVersionCurrent ==0: (default) means the current version
_kVersion2000 == 2000: Acad version 2000 compatible
_kVersion2004 == 2004: Acad version 2004 compatible
_kVersion2007 == 2007: Acad version 2007 compatible
_kVersion2010 == 2010: Acad version 2010 compatible
```

*[Example:*

```
U(1, "mm");

int arVersions[] = {_kVersionCurrent, _kVersion2000,
_kVersion2004, _kVersion2007, _kVersion2010 };
PropInt pVersion(0,arVersions,T("|Dxf or Dwg version to file|"));

// make a display that shows some graphics
Display dpShow(-1);
PLine plCir; plCir.createCircle(_Pt0, _ZU,U(100));
dpShow.draw(plCir);

// build a display that outputs to DXF or DWG
```

```

Display dpDxf(-1);
dpDxf.showInTslInst(_bOnDebug); // but only show in dwg when
during debug

PLine pl1(_zU); // define polyline
pl1.addVertex(_Pt0);
pl1.addVertex(_Pt0+U(100)*_xU);
pl1.addVertex(_Pt0+U(100)*_xU+U(200)*_yU);
pl1.addVertex(_Pt0+U(200)*_yU);
pl1.addVertex(_Pt0);

dpDxf.draw(pl1); // draw polyline in active

String str = "This is a text!";

dpDxf.dimStyle("ISO-25"); // specify new active dimstyle
dpDxf.textHeight(U(10)); // specify active textHeight

dpDxf.draw(str, _Pt0, _xU, _yU, 1, 1); // draw text

String strOutputDxf = T("output Dxf file");
addRecalcTrigger(_kContext, strOutputDxf);
if (_bOnRecalc && _kExecuteKey==strOutputDxf) {
    int bRelativeToPt0 = FALSE;
    String strFile = _kPathDwg +"\\"+ scriptName(); // extension
will be added automatically
    dpDxf.writeToDxfFile(strFile, bRelativeToPt0, pVersion);
}

String strOutputDwg = T("output Dwg file");
addRecalcTrigger(_kContext, strOutputDwg );
if (_bOnRecalc && _kExecuteKey==strOutputDwg ) {
    int bRelativeToPt0 = TRUE;
    String strFile = _kPathDwg +"\\"+ scriptName() + ".dwg";
    dpDxf.writeToDwgFile(strFile, bRelativeToPt0, pVersion);
}

—end example]
strQRCode

```

**drawQR(String strQRCode, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag, double dYFlag, double dSize); // (added in build v21.3.101)**

Draw a QR code. The QR code must be in the form returned by **String::qrEncode**. This format starts with "QR:", then followed with a series of 0's and 1's expressing the state of the pixels of the QR code. The ptLocation, vecX, vecY, dXFlag and dYFlag, as well as the dSize contribute to the location and dimension of the image.

*[Example:*



```

U(1, "mm");

PropDouble pSize(0, U(100), T("|Size of QR|"));
PropString strToQr(0, "www.hsbcad.com", T("|QR content|"));
PropInt pMargin(0, 4, T("|White margin|"));

String strQRed = strToQr.qrEncode(pMargin);
reportMessage("\\n'" + strToQr + "' encoded becomes '" + strQRed +
"');

String strFromQr = strQRed.qrDecode();
reportMessage("\\n'" + strToQr + "' encoded and decoded becomes: '" +
strFromQr + "'");

Display dp(-1);
dp.drawQR(strQRed, _Pt0, _XU, _YU, 1, 1, pSize);

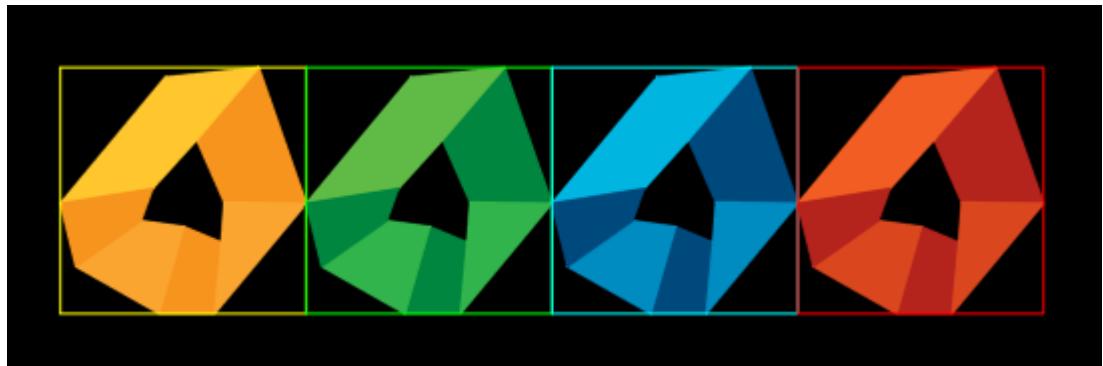
—end example]

drawHsbLogo(int nColorSet, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag,
double dYFlag, double dSize); // (added in build v23.3.10)

```

Draw an hsbcad logo. The hsbcad logo comes in different color sets. 1: red, 2: yellow, 3: green and 4: cyan, following the autocad color index meaning. Any other color set index will draw the logo in red. The ptLocation, vecX, vecY, dXFlag and dYFlag, as well as the dSize contribute to the location and dimension of the image.

*[Example:*



```

U(1, "mm");

PropDouble pSize(0, U(100), T("|Size|"));
PropInt pColor(0,0, T("|Logo color|"));

Display dp(-1);
dp.drawHsbLogo(pColor, _Pt0, _XU, _YU, 1, 1, pSize);

dp.color(pColor);
PLine plRect;
plRect.createRectangle(LineSeg(_Pt0, _Pt0 + pSize * _XU + pSize * _YU), _XU, _YU);
dp.draw(plRect);

—end example]

```

---

[Example:

```

U(1,"mm");

Display dp(1); // choose color red

PLine pl1(_ZU); // define polyline
pl1.addVertex(_Pt0);
pl1.addVertex(_Pt0+U(100)*_XU);
pl1.addVertex(_Pt0+U(100)*_XU+U(200)*_YU);
pl1.addVertex(_Pt0+U(200)*_YU);
pl1.addVertex(_Pt0);

dp.draw(pl1); // draw polyline in active color red

dp.color(3); // change active color to green

String str = "Dit is een tekstje!";

dp.dimStyle("ISO-25"); // specify new active dimstyle
dp.textHeight(U(10)); // specify active textHeight

```

---

```

dp.draw(str,_Pt0,_XU,_YU,1,1); // draw text

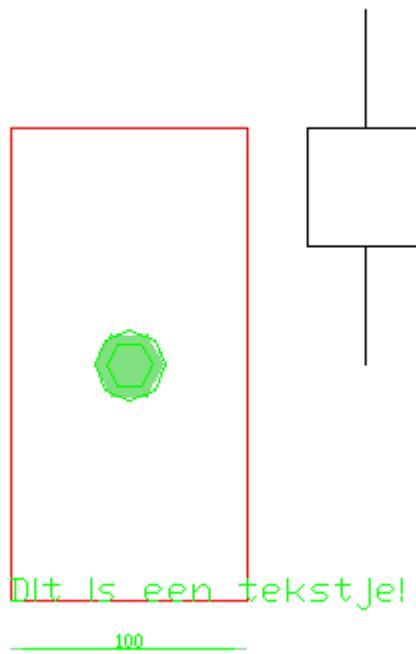
// specify and draw a dimension
DimLine dl(_Pt0-20*_YU,_XU,_YU);
Dim dim(dl,_Pt0,_Pt0+U(100)*_XU);
dp.draw(dim);

// specify hardware and draw
Hardware bolt1("name","type",U(100),U(20),4,"Iron"); // use strings
Point3d ptBolt = _Pt0+U(50)*_XU+U(100)*_YU;
dp.draw(bolt1, ptBolt, _ZU);

dp.addViewDirection(_XW); // If for a display, the viewing direction is set, the display will
only show for that direction
dp.addViewDirection(_ZW); // The viewing direction is pointing from the entity to the
viewer, so _ZW means top view

// specify and draw a block drawing
Block aa("aa");
Point3d ptBlock = _Pt0+U(150)*_XU+U(100)*_YU;
dp.draw(aa, ptBlock );

```



—end example]

[Example of O tsl with insert done in script to illustrate the draw Entity.

```

if (_bOnInsert) {
    _Entity.append(getEntity());
    _Pt0 = getPoint();
    return;
}

```

```

}

if (_Entity.length()==0) {
    eraseInstance();
    return;
}

Entity ent = _Entity[0];
PropString pDispRep(0,ent.dispRepNames(),"Disp rep");

// get the coordinate system of the entity
CoordSys csEnt = ent.coordSys();

// possibly permute some
CoordSys csEntTr(csEnt.ptOrg(),csEnt.vecX(),csEnt.vecZ(),-csEnt.vecY());

// invert
csEntTr.invert();

// map to the coordinate system associated with this Tsl
CoordSys cs(_Pt0,_XU,_YU, _ZU);
csEntTr.transformBy(cs);

Display dp(-1);
dp.draw(ent, csEntTr, pDispRep); // display the entity
dp.draw(scriptName(),_Pt0,_XU,_YU,1,1); // also show this tsl name

```

*—end example]*

*[Example of O tsl with insert done in script to illustrate the drawQr and drawHsbLogo.*



```

U(1, "mm");

PropDouble pSize(0, U(100), T("|Size of QR|"));

PropString strToQr(0, "www.hsbcad.com", T("|QR content|"));

String arEcTypes[] = {T("|low|"), T("|medium|"), T("|high|")};
PropString pErrCor(1, arEcTypes, T("|Error correction|"));
int nErrCor = arEcTypes.find(pErrCor, 0);

PropInt pMargin(1, 2, T("|White margin|"));

String arImgSizes[] = {T("|no embedded
image|"), "1%", "2%", "3%", "4%", "5%"};
PropString pRoomForImage(2, arImgSizes, T("|Size of embedded image,
in percentage of area of relevant pixels|"));
int nRoomForImage = arImgSizes.find(pRoomForImage, 0);

String arLogoColors[] = {T("|no logo|"), T("|red|"), T("|yellow|"), T("|green|"),
T("|cyan|")};
PropString pLogoColorSet(3, arLogoColors, T("|Color set for embedded
hsbcad logo|"));
int nLogoColorSet = arLogoColors.find(pLogoColorSet, 0);

String strQRed = strToQr.qrEncode(pMargin, nErrCor, nRoomForImage);
//reportMessage("\n'" + strToQr + "' encoded becomes '" + strQRed +
"''");

String strFromQr = strQRed.qrDecode();
reportMessage("'\n'" + strToQr + "' encoded and decoded becomes: '" +
strFromQr + "'");

Display dp(-1);
dp.drawQR(strQRed, _Pt0, _XU, _YU, 1, 1, pSize);

PLine plRect;
plRect.createRectangle(LineSeg(_Pt0, _Pt0 + pSize * _XU + pSize *
_YU, _XU, _YU);
dp.draw(plRect);

if (nLogoColorSet > 0 && nRoomForImage > 0)
{
    PLine plRectImg;
    Point3d ptC = _Pt0 + 0.5 * pSize * _XU + 0.5 * pSize * _YU;
    double dSizeImg = strQRed.inQrEmbeddedImageSizeFactor() * pSize;
    LineSeg lnSegImg(ptC - 0.5 * dSizeImg * _XU - 0.5 * dSizeImg *
_YU, ptC + 0.5 * dSizeImg * _XU + 0.5 * dSizeImg * _YU);
    plRectImg.createRectangle(lnSegImg, _XU, _YU);

    //dp.color(nLogoColorSet);
    //dp.draw(plRectImg);
}

```

```

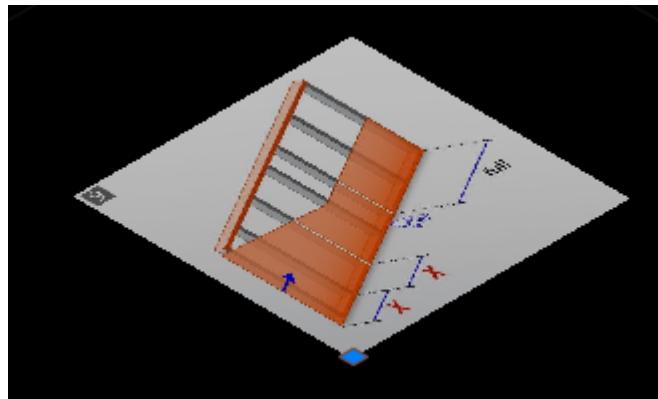
    plRectImg.vis(nLogoColorSet);

    double dSizeLogo = dSizeImg * 0.9;
    dp.drawHsbLogo(nLogoColorSet, ptc, _XU, _YU, 0, 0, dSizeLogo);
}

—end example]

```

[Example of O tsI drawImage.



```

U(1, "mm");
PropInt pFileIndex(0, 0, T("|File index|"));

String arFiles[] =
{
    _kPathHsbInstall + "\\Wand\\Distribution_9.gif",
    _kPathHsbInstall + "\\hsbRevitData\\res\\Hsb_Revit_Settings.bmp",
    _kPathHsbInstall + "\\Content\\Dutch\\hsbCompany\\ElementRoof\
\\Valley-rafter-1.jpg",
    _kPathHsbInstall + "\\Abbund\\Hsb.ico",
    _kPathHsbInstall + "\\Lang\\EULA 2020 v2_files\\image001.png",
    _kPathHsbInstall + "\\none existant file"
};

int nInd = pFileIndex;
nInd = abs(nInd % arFiles.length());
String strFile = arFiles[nInd];
String strImg = strFile.encodeImageFromFile();
if (strImg.length() == 0)
{
    reportMessage("\nFile '" + strFile + "' could not be read as
image.");
}

double dHOverW = strImg.encodedImageHeightOverWidth();
if (dHOverW < 0)
{
    reportMessage("\nImage '" + strImg + "' could not decode into
image.");
}

```

```

Display dp(-1);

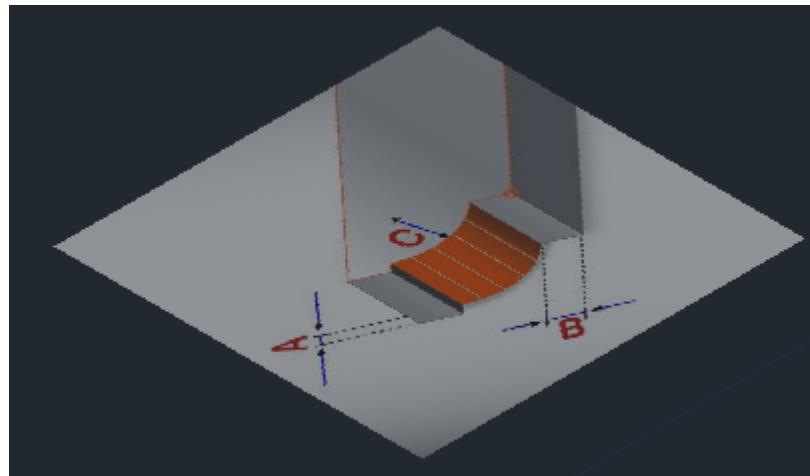
double dSizeInX = U(1000);
double dSizeInY = dHOverW * dSizeInX;

dp.transparency(50); // superimpose 50% transparency
dp.drawImage(strImg, _Pt0, _XU, _YU, 1, 1, dSizeInX, dSizeInY);

—end example]

```

[Example of O tsl drawImage from an embedded image. Make sure to add an image in the Tsl Resource manager called "myimage".



**U**(1, "mm");

```

String strImgName = "myimage";
String strImg = TslScript(scriptName()).subMapX("Resource[]\\Image[]\\"
    "+strImgName).getString("Encoding");
if (strImg.length() == 0)
{
    reportMessage("\\nAdd image '" + strImgName + "' to the resources of
the tsl.");
}

double dHOverW = strImg.encodedImageHeightOverWidth();
if (dHOverW < 0)
{
    reportMessage("\\nImage '" + strImgName + "' could not decode into
image.");
}

Display dp(-1);

double dSizeInX = U(1000);
double dSizeInY = dHOverW * dSizeInX;

dp.transparency(50); // superimpose 50% transparency

```

```

if (strImg.length() == 0)
{
    dp.draw(scriptName(), _Pt0, _XU, _YU, 1, 1);
}
else
{
    dp.drawImage(strImg, _Pt0, _XU, _YU, 1, 1, dSizeInX, dSizeInY);
}

—end example]

```

## 16.2 Dim and DimLine

Dimensioning can be done by using the DimLine and Dim types. In order to draw a dimension between 2 points along a certain line, one has to define a dimension line. Along this line the distance will be measured, and also the distance will be displayed. To draw the actual dimension, the Dim type need to be declared, and then used in a display. To change the color, linetype, text font, arrow size,... adapt the dimstyle, and make this dimstyle active for the display with the [dimStyle](#) call.

The Dim when drawn, will generate a set of dimensioning values. These values are positioned such that they do not overlap. Only in the case of [\\_kDimClassic](#) mode, the collision checking is not done.

---

```

class DimLine {

    DimLine(Point3d ptOnLine, Vector3d vecX, Vector3d vecY); // make sure
        vecX.crossProduct(vecY).dotProduct(getViewDirection()) > 0

    Point3d[] collectDimPoints(Beam[] arEnts, int nType) const;
    Point3d[] collectDimPoints(Sheet[] arEnts, int nType) const;
    Point3d[] collectDimPoints(Sip[] arEnts, int nType) const;
    Point3d[] collectDimPoints(GenBeam[] arEnts, int nType) const;

    Point3d[] collectDimPoints(Beam[] arEnts, int nType, int bAcceptAllBeams) const; // added
        since 22.1.132 and 23.5.9
    Point3d[] collectDimPoints(Sheet[] arEnts, int nType, int bAcceptAllBeams) const; // added
        since 22.1.132 and 23.5.9
    Point3d[] collectDimPoints(Sip[] arEnts, int nType, int bAcceptAllBeams) const; // added
        since 22.1.132 and 23.5.9
    Point3d[] collectDimPoints(GenBeam[] arEnts, int nType, int bAcceptAllBeams) const; // added
        since 22.1.132 and 23.5.9
}
```

---

```

Point3d[] collectDimPoints(PLine[] arEnts, int nType) const;
Point3d[] collectDimPoints(PLine pline, int nType) const;

transformBy(CoordSys csTransformationMatrix);
transformBy(Vector vecTranslate);

Point3d ptOrg() const;
Vector3d vecX() const;
Vector3d vecY() const;

void setUseDisplayTextHeight(int bSet); // (added since 23.7.9). If TRUE (default FALSE),
// will force the text height to be the one from the Display instead of the text
// height of the DimStyle
};

```

---

```

class Dim {

    Dim(DimLine InDim, Point3d pt1, Point3d pt2);
    Dim(DimLine InDim, Point3d pt1, Point3d pt2, String txtDelta);
    Dim(DimLine InDim, Point3d pt1, Point3d pt2, int nChainModus);

    Dim(DimLine InDim, Point3d[] pnts, String txtDelta, String txtCummulative, int
        nChainModus);
    Dim(DimLine InDim, String txtDelta, String txtCummulative, int nChainModus);

    Dim(DimLine InDim, Point3d[] pnts, String txtDelta, String txtCummulative, int
        nDisplayModusDelta, int nDisplayModusCummulative);

    append(Point3d pt);
    append(Point3d pt, String txtMiddle, String txtEnd);

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    setDeltaOnTop(int bOnTop); // set location of delta dimension, default is on top, TRUE. The
        // cummulative dimension is always at opposite side.
    setReadDirection(Vector3d vecRead); // set direction of your head up to read the
        // dimension text
    correctTextNormalForViewDirection(Vector3d vecView); // (added V25.1.115 and V26.1.19)
        // correct the text normal vecZ (by flipping vecX) such that vecZ has positive
        // dotProduct with vecView direction.

    CoordSys coordSys() const; // (added V25.1.115 and V26.1.19) returns vecX along the dim
        // line, vecY controlling deltaOnTop, and vecZ the view direction.
    int displayModusDelta() const; // (added V25.1.115 and V26.1.19) returns _kDimClassic,
        _kDimNone, _kDimPar or _kDimPerp
    int displayModusCummulative() const; // (added V25.1.115 and V26.1.19) returns
        _kDimClassic, _kDimNone, _kDimPar or _kDimPerp
}

```

---

```
PLine[] getTextAreas(Display dp) const; // added 22.1.84 and 23.0.46
};
```

---

**DimLine(Point3d ptOnLine, Vector3d vecX, Vector3d vecY);**

This constructor of a DimLine requires a point and a vecX, which define the line along which the dimensions will be taken. The third argument is a vecY, which defines the upside direction for the dimension text. So dimensions will be displayed in the plane given by vecX and vecY, through the point ptOnLine.

```
Point3d[] collectDimPoints(Beam[] arEnts, int nType) const;
Point3d[] collectDimPoints(Sheet[] arEnts, int nType) const;
Point3d[] collectDimPoints(Sip[] arEnts, int nType) const;
Point3d[] collectDimPoints(GenBeam[] arEnts, int nType) const;
Point3d[] collectDimPoints(PLine[] arEnts, int nType) const;
Point3d[] collectDimPoints(PLine pline, int nType) const;
Point3d[] collectDimPoints(Beam[] arEnts, int nType, int bAcceptAllBeams) const; // added
since 22.1.132 and 23.5.9
Point3d[] collectDimPoints(Sheet[] arEnts, int nType, int bAcceptAllBeams) const; // added
since 22.1.132 and 23.5.9
Point3d[] collectDimPoints(Sip[] arEnts, int nType, int bAcceptAllBeams) const; // added since
22.1.132 and 23.5.9
Point3d[] collectDimPoints(GenBeam[] arEnts, int nType, int bAcceptAllBeams) const; // added
since 22.1.132 and 23.5.9
```

The collectDimPoints routines will loop over all the elements in the array arEnts, and find appropriate dimensioning points.

Current values of nType supported:

**\_kLeft** only the point with the smallest X coordinate (along the vecX of the dimline) is appended for each entity.

**\_kRight** only the point with the biggest X coordinate (along the vecX of the dimline) is appended for each entity.

**\_kLeftAndRight** both the point with the smallest and the biggest X coordinate (along the vecX of the dimline) are appended for each entity.

**\_kCenter** the average point of \_kLeft and \_kRight is appended for each entity.

The call with the PLine as argument, will loop over the vertex points of the PLine, and find the appropriate points.

If bAcceptAllBeams is TRUE (default FALSE), all GenBeams are accepted, otherwise only perpendicular are accepted.

*[Sample code :*

```
{
    DimLine ln(pp, csEl.vecX(), csEl.vecY()); // csEl could be the CoordSys of an element

    Beam arBeams[] = elToUse.beam(); // get an array of beams of the element elToUse
    Point3d pnts[0]; // define a new array of Point3d, called pnts, with 0 points in it.
    pnts.append(csEl.ptOrg()); // append origin of element to the array pnts
```

```

pnts.append(ln.collectDimPoints(arBeams,_kCenter)); // add an other array to the

Dim dim(ln,pnts,"<>","<>",_kDimCumm);
dp.draw(dim);
}

—end sample]

Dim(DimLine InDim, Point3d pt1, Point3d pt2);
Dim(DimLine InDim, Point3d pt1, Point3d pt2, String txtMiddle);
Dim(DimLine InDim, Point3d pt1, Point3d pt2, int nChainModus);
Dim(DimLine InDim, Point3d[] pnts, String txtMiddle, String txtEnd, int nChainModus);
Dim(DimLine InDim, String txtMiddle, String txtEnd, int nChainModus);
Dim(DimLine InDim, Point3d[] pnts, String txtMiddle, String txtEnd, int nDisplayModusMiddle,
int nDisplayModusEnd);

```

These constructors of an actual dimension, Dim, requires at least a DimLine InDim. In the case multiple points are appended, different measures can be displayed depending on the nChainModus:

- \_kDimNone no values are displayed
- \_kDimDelta only the distances between the points are displayed
- \_kDimCumm only the cumulative distances from the first point are displayed
- \_kDimBoth both the distances between the points, as well as the cumulative distances from the first point are displayed
- \_kDimClassic only the distances between the points are displayed, but with pure Autocad dimensions

If the dimension text needs to be placed perpendicular to the dimension line, the nDisplayModusMiddle and nDisplayModusEnd need to be used:

- \_kDimNone the value is not displayed
- \_kDimPar the value is displayed parallel to the dim line
- \_kDimPerp the value is displayed perpendicular to the dim line

**setDeltaOnTop(**int** bOnTop);**

Sets the location of the delta dimension. The dimension can display both the cumulative and the incremental (delta) dimension of a collection of points. This delta dimension is displayed, by default, on the upper reading side of the dimension line. If bOnTop is set to FALSE, the delta dimension text will be displayed on the lower side of the dimension line.

**setReadDirection(**Vector3d** vecRead);**

Sets the read direction for the numerical and text values that are displayed. The vecRead is the head up direction from which all texts should be readable. Since the text could be parallel, as well as perpendicular to the dimension direction, it is important to specify the way to turn your head to read all dimensions.

**PLine[] getTextAreas(**Display** dp) const**

Gets the rectangles around the texts of the dimension, given the DimStyle which would be used in the Display.

---

[Example:

```

U(1,"mm");

Display dp(-1); // use default color
dp.dimStyle("ISO-25"); // specify dimensionstyle to use

Point3d pt1 = _Pt0; // start of polyline
Point3d pt2 = _PtG[0]; // second point is grippoint
Point3d pt3 = _Pt0+U(200)*_XU; // third is fixed
Point3d pt4 = (pt1+pt2)/2;
Point3d pt5 = (pt3+pt2)/2;

// specify and draw polyline
PLine pl1(pt1,pt2,pt3);
dp.color(5); // from now on draw with blue
dp.draw(pl1);

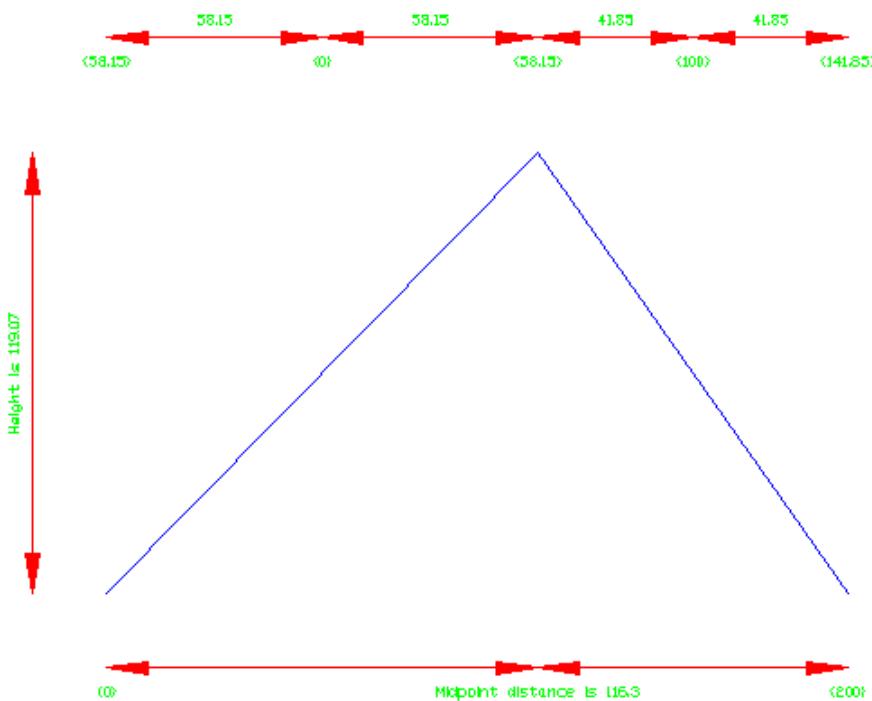
DimLine dl1(_Pt0-20*_XU, _YU, -_XU);
Dim dim1(dl1,pt1,pt2, "Height is <>"); // <> will be replaced by actual dimension
dp.draw(dim1);

DimLine dl2(_Pt0-20*_YU, _XU, _YU);
Dim dim2(dl2,"<>","{<>}",__kDimCumm);
dim2.append(pt1); // the sequence of the points is not important, just the first one
dim2.append(pt3);
dim2.append(pt2);
dim2.append(pt2,"<>","Midpoint distance is <>"); // points can be added multiple times.
Only the last text specification remains
dp.draw(dim2);

Point3d pnts[0];
pnts.append(pt4); // first point is the origin point for cumulative dimension
pnts.append(pt1);
pnts.append(pt2);
pnts.append(pt3);
pnts.append(pt5);
DimLine dl3(_Pt0+150*_YU, _XU, _YU);
Dim dim3(dl3,pnts,"<>","{<>}",__kDimBoth);
dp.draw(dim3);

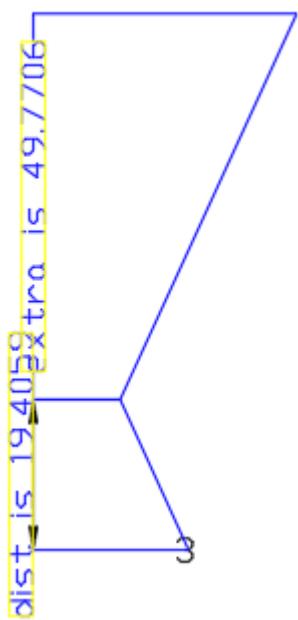
```

---



—end example]

[Example O-type with 2 extra grip points:



**U(1, "mm");**

---

```

String sTextHeightName=T("|Text Height|");
PropDouble dTextHeight(0, U(0), sTextHeightName);
dTextHeight.setDescription(T("|Defines the TextHeight|"));

String sDimStyleName=T("|DIMStyle|");
PropString sDimStyle(0, _DimStyles.sorted(), sDimStyleName);
sDimStyle.setDescription(T("|Defines the DIMStyle|"));

String sUseDisplayTextHeight=T("|Use display text height for dim|");

PropInt useDisplayTextHeight(0, 0, sUseDisplayTextHeight);

Display dp(-1); // use default color
dp.dimStyle(sDimStyle); // specify dimensionstyle to use

double textHeight = dTextHeight > 0 ? dTextHeight :
dp.textHeightForStyle("0", sDimStyle);

dp.textHeight(textHeight);

String strHeight = textHeight;
dp.draw(strHeight, _Pt0, _XW, _YW, 0, 0, _kDevice);

Point3d pt1 = _Pt0;
Point3d pt2 = _PtG[0];
Point3d pt3 = _PtG[1];

// specify and draw polyline
PLine pll(pt1,pt2,pt3);
dp.color(5); // from now on draw with blue
dp.draw(pll);

DimLine dl1(_Pt0-20*_XU, _YU, -_XU);
dl1.setUseDisplayTextHeight(useDisplayTextHeight);

Dim dim1(dl1,pt1,pt2, "dist is <>"); // <> will be replaced by actual
dimension
dp.draw(dim1);

Dim dim2(dl1, "extra is <>", "", _kDimClassic); // <> will be
replaced by actual dimension
dim2.append(pt2);
dim2.append(pt3);
dp.draw(dim2);

Display dpTA(2);
PLine pls[] = dim1.getTextAreas(dp);
PLine pls2[] = dim2.getTextAreas(dp);
pls.append(pls2);
for (int p = 0; p < pls.length(); p++)
dpTA.draw(pls[p]);

```

*—end example]*

### 16.3 DimAngular

Dimensioning of Angular type can be done by using the DimAngular type. A variable of DimAngular type needs to be defined, and then used in a display to draw. To change the color, linetype, text font, arrow size,... adapt the dimstyle, and make this dimstyle active for the display with the [dimStyle](#) call.

---

```
class DimAngular { // added 25.1.1535

    DimAngular(Point3d ptCenter, Point3d ptXline1, Point3d ptXline2, Point3d ptArc);

    void setTextLocation(Point3d ptLoc);
    void setText(String strText); // the default text is "<>" which serves as a placeholder for the
                                 actual dimension
    void setDimensionPlane(Point3d ptPlane, Vector vecInPlane, Vector vecNormal); // if not
                                 specified the plane is derived from ptCenter, ptXline1 and ptXline2
    void setDimScale(double dScale); // if dScale is 0, which is the default, the dimscale of the
                                 active dimstyle of the Display is used.

    void transformBy(CoordSys csTransformationMatrix);
    void transformBy(Vector vecTranslate);

    void setUseDisplayTextHeight(int bSet); // If TRUE (default FALSE), will force the text
                                         height to be the one from the Display instead of the text height of the
                                         DimStyle

    PLine[] getTextAreas(Display dp) const;
};
```

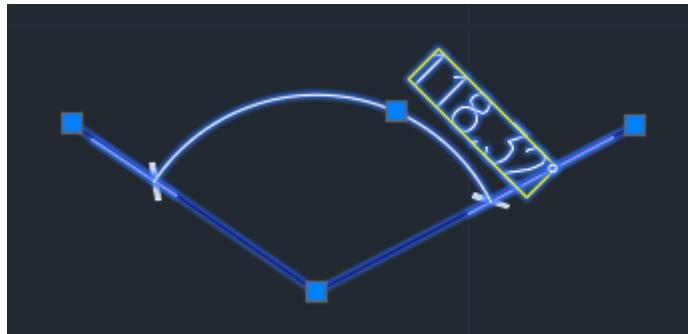
---

**PLine[] getTextAreas(Display dp) const**

Gets the rectangles around the texts of the dimension, given the DimStyle which would be used in the Display.

---

*[Example:*



```

(1, "mm");

String sTextHeightName=T("|Display text Height|");
PropDouble dTextHeight(0, u(0), sTextHeightName);
dTextHeight.setDescription(T("|Defines the TextHeight|"));

String sDimStyleName=T("|DIMStyle|");
PropString sDimStyle(0, _DimStyles.sorted(), sDimStyleName);
sDimStyle.setDescription(T("|Defines the DIMStyle|"));

String sUseDisplayTextHeight=T("|Use display text height for dim|");

PropInt useDisplayTextHeight(0, 0, sUseDisplayTextHeight);

PropString pText(1, "");
PropDouble pDimScale(1, 0);

String strSetTextLoc = T("|Set text location|");
addRecalcTrigger(_kContext, strSetTextLoc );
if (_bOnRecalc && _kExecuteKey==strSetTextLoc)
{
    Point3d pt = getPoint(T("|Select text location|"));
    _Map.setPoint3d("textLoc", pt);
}

String strRemoveTextLoc = T("|Remove text location|");
addRecalcTrigger(_kContext, strRemoveTextLoc );
if (_bOnRecalc && _kExecuteKey==strRemoveTextLoc)
{
    _Map.removeAt("textLoc", FALSE);
}

Display dp(-1); // use default color
dp.dimStyle(sDimStyle); // specify dimensionstyle to use

if (dTextHeight > 0.01)
    dp.textHeight(dTextHeight);

Point3d ptCent = _Pt0;
Point3d ptXLine1 = _PtG[0];

```

```

Point3d ptXLine2 = PtG[1];
Point3d ptArc = PtG[2];

DimAngular dimAng(ptCent, ptXLine1, ptXLine2, ptArc);
if (useDisplayTextHeight)
    dimAng.setUseDisplayTextHeight(TRUE);
if (pText.length() != 0)
    dimAng.setText(pText);

dimAng.setDimensionPlane(_PtU, _XU, _ZU);

if (pDimScale > 0.001)
    dimAng.setDimScale(pDimScale);

if (_Map.hasPoint3d("textLoc"))
{
    Point3d ptLoc = _Map.getPoint3d("textLoc");
    dimAng.setTextLocation(ptLoc);
}

dp.draw(dimAng);

// specify and draw polyline
PLine pll(ptXLine1, ptCent, ptXLine2);
dp.transparency(70);
dp.color(5); // from now on draw with blue
dp.draw(pll);

Display dpTA(2);
PLine pls[] = dimAng.getTextAreas(dp);
for (int p = 0; p < pls.length(); p++)
    dpTA.draw(pls[p]);

```

—end example]

## 16.4 DimRadial

Dimensioning of Radial type can be done by using the DimRadial type. A variable of DimRadial type needs to be defined, and then used in a display to draw. To change the color, linetype, text font, arrow size,... adapt the dimstyle, and make this dimstyle active for the display with the [dimStyle](#) call.

```
class DimRadial { // added 25.1.25  
  
    DimRadial(Point3d ptCenter, Point3d ptChord, double leaderLength);  
  
    void setTextLocation(Point3d ptLoc);  
    void setText(String strText); // the default text is "<>" which serves as a placeholder for the  
                                actual dimension
```

```

void setDimensionPlane(Point3d ptPlane, Vector vecInPlane, Vector vecNormal); // if not
    specified the plane is derived from ptCenter, ptXline1 and ptXline2
void setDimScale(double dScale); // if dScale is 0, which is the default, the dimscale of the
    active dimstyle of the Display is used.

void transformBy(CoordSys csTransformationMatrix);
void transformBy(Vector vecTranslate);

void setUseDisplayTextHeight(int bSet); // If TRUE (default FALSE), will force the text
    height to be the one from the Display instead of the text height of the
    DimStyle

PLine[] getTextAreas(Display dp) const;
};

```

---

#### **PLine[]** getTextAreas(**Display** dp) const

Gets the rectangles around the texts of the dimension, given the DimStyle which would be used in the Display.

---

[Example:



```

U(1, "mm");

String sTextHeightName=T("|Display text Height|");
PropDouble dTextHeight(0, U(0), sTextHeightName);
dTextHeight.setDescription(T("|Defines the TextHeight|"));

String sDimStyleName=T("|DimStyle|");
PropString sDimStyle(0, _DimStyles.sorted(), sDimStyleName);
sDimStyle.setDescription(T("|Defines the DimStyle|"));

String sUseDisplayTextHeight=T("|Use display text height for dim|");

PropInt useDisplayTextHeight(0, 0, sUseDisplayTextHeight);

```

---

```
PropString pText(1, "");  
PropDouble pDimScale(1, 0);  
  
PropDouble dLeaderLength(2, U(0), T("|Leader length|"));  
  
String strSetTextLoc = T("|Set text location|");  
addRecalcTrigger(_kContext, strSetTextLoc );  
if (_bOnRecalc && _kExecuteKey==strSetTextLoc)  
{  
    Point3d pt = getPoint(T("|Select text location|"));  
    _Map.setPoint3d("textLoc", pt);  
}  
  
String strRemoveTextLoc = T("|Remove text location|");  
addRecalcTrigger(_kContext, strRemoveTextLoc );  
if (_bOnRecalc && _kExecuteKey==strRemoveTextLoc)  
{  
    _Map.removeAt("textLoc", FALSE);  
}  
  
Display dp(-1); // use default color  
dp.dimStyle(sDimStyle); // specify dimensionstyle to use  
  
if (dTTextHeight > 0.01)  
    dp.textHeight(dTextHeight);  
  
Point3d ptCent = _Pt0;  
Point3d ptChord = _PtG[0];  
  
DimRadial dimAng(ptCent, ptChord, dLeaderLength);  
if (useDisplayTextHeight)  
    dimAng.setUseDisplayTextHeight(TRUE);  
if (pText.length() != 0)  
    dimAng.setText(pText);  
  
dimAng.setDimensionPlane(_PtU, _XU, _ZU);  
  
if (pDimScale > 0.001)  
    dimAng.setDimScale(pDimScale);  
  
if (_Map.hasPoint3d("textLoc"))  
{  
    Point3d ptLoc = _Map.getPoint3d("textLoc");  
    dimAng.setTextLocation(ptLoc);  
}  
  
dp.draw(dimAng);  
  
// specify and draw polyline  
PLine pll(ptChord, ptCent);
```

```

dp.transparency(70);
dp.color(5); // from now on draw with blue
dp.draw(pl1);

Display dpTA(2);
PLine pls[] = dimAng.getTextAreas(dp);
for (int p = 0; p < pls.length(); p++)
    dpTA.draw(pls[p]);

—end example]

```

## 16.5 DimDiametric

Dimensioning of Diametric type can be done by using the DimDiametric type. A variable of DimDiametric type needs to be defined, and then used in a display to draw. To change the color, linetype, text font, arrow size,... adapt the dimstyle, and make this dimstyle active for the display with the [dimStyle](#) call.

---

```

class DimDiametric { // added 25.1.25

    DimDiametric(Point3d ptChord, Point3d ptFarChord, double leaderLength);

    void setTextLocation(Point3d ptLoc);
    void setText(String strText); // the default text is "<>" which serves as a placeholder for the
        actual dimension
    void setDimensionPlane(Point3d ptPlane, Vector vecInPlane, Vector vecNormal); // if not
        specified the plane is derived from ptCenter, ptXline1 and ptXline2
    void setDimScale(double dScale); // if dScale is 0, which is the default, the dimscale of the
        active dimstyle of the Display is used.

    void transformBy(CoordSys csTransformationMatrix);
    void transformBy(Vector vecTranslate);

    void setUseDisplayTextHeight(int bSet); // If TRUE (default FALSE), will force the text
        height to be the one from the Display instead of the text height of the
        DimStyle

    PLine[] getTextAreas(Display dp) const;
}

```

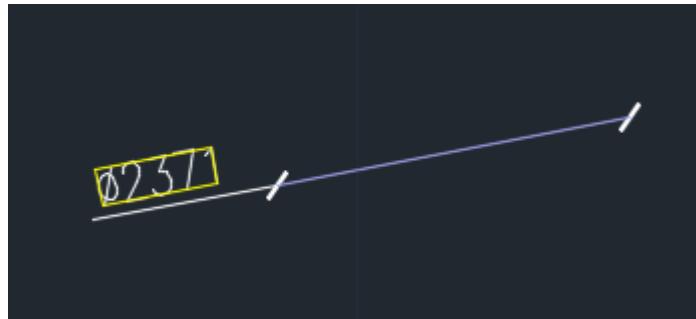
---

**PLine[] getTextAreas(Display dp) const**

Gets the rectangles around the texts of the dimension, given the DimStyle which would be used in the Display.

---

[Example:



```

(1,"mm");

String sTextHeightName=T("|Display text Height|");
PropDouble dTextHeight(0, U(0), sTextHeightName);
dTextHeight.setDescription(T("|Defines the TextHeight|"));

String sDimStyleName=T("|DimStyle|");
PropString sDimStyle(0, _DimStyles.sorted(), sDimStyleName);
sDimStyle.setDescription(T("|Defines the DimStyle|"));

String sUseDisplayTextHeight=T("|Use display text height for dim|");

PropInt useDisplayTextHeight(0, 0, sUseDisplayTextHeight);

PropString pText(1, "");
PropDouble pDimScale(1, 0);

PropDouble dLeaderLength(2, U(0), T("|Leader length|"));

String strSetTextLoc = T("|Set text location|");
addRecalcTrigger(_kContext, strSetTextLoc );
if (_bOnRecalc && _kExecuteKey==strSetTextLoc)
{
    Point3d pt = getPoint(T("|Select text location|"));
    _Map.setPoint3d("textLoc", pt);
}

String strRemoveTextLoc = T("|Remove text location|");
addRecalcTrigger(_kContext, strRemoveTextLoc );
if (_bOnRecalc && _kExecuteKey==strRemoveTextLoc)
{
    _Map.removeAt("textLoc", FALSE);
}

```

```

Display dp(-1); // use default color
dp.dimStyle(sDimStyle); // specify dimensionstyle to use

if (dTextHeight > 0.01)
    dp.textHeight(dTextHeight);

Point3d ptChord = _Pt0;
Point3d ptFarChord = _PtG[0];

DimDiametric dimAng(ptChord, ptFarChord, dLeaderLength);
if (useDisplayTextHeight)
    dimAng.setUseDisplayTextHeight(TRUE);
if (pText.length() != 0)
    dimAng.setText(pText);

dimAng.setDimensionPlane(_PtU, _XU, _ZU);

if (pDimScale > 0.001)
    dimAng.setDimScale(pDimScale);

if (_Map.hasPoint3d("textLoc"))
{
    Point3d ptLoc = _Map.getPoint3d("textLoc");
    dimAng.setTextLocation(ptLoc);
}

dp.draw(dimAng);

// specify and draw polyline
PLine pll(ptChord, ptFarChord);
dp.transparency(70);
dp.color(5); // from now on draw with blue
dp.draw(pll);

Display dpTA(2);
PLine pls[] = dimAng.getTextAreas(dp);
for (int p = 0; p < pls.length(); p++)
    dpTA.draw(pls[p]);

```

*—end example]*

## 16.6 Display text

The following properties of the display influence the appearance of text.

- color: the color of the text;
- dimStyle: the text style of the dimstyle is used;
- textHeight: the text height of the dimstyle can be overwritten.

```

draw(String strText, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag,
double dYFlag);
draw(String strText, Point3d ptLocation, Vector3d vecX, Vector3d vecY, double dXFlag,
double dYFlag, int nTextOrientation);

```

To draw text, the location point needs to be specified, as well as the text direction, and the text height direction. As with other tools, the location point can be specified with relative coordinates in the box surrounding the text by the dXFlag and dYFlag. If the flags are all equal to 0, the location point is located in the middle of the box. If all flags are equal to 1, the point is located at the corner point, in the -vecX and -vecY direction. So the flags are actually the relative coordinates of the centerpoint of the box, in the vecX, vecY coordinate system.

The text is by default oriented in the model space (nTextOrientation equals to **\_kModel**), but it can also be displayed parallel to the device/screen. By specifying the nTextOrientation equal to **\_kDevice**, the text will always be parallel to the screen. If specified **\_kDeviceX**, the text will always be displayed horizontal as well. The following values of nTextOrientation can be used:

- \_kModel ==0:** means aligned in the model
- \_kDevice ==1:** means parallel to the screen/device
- \_kDeviceX ==2:** means parallel to the screen/device, and also horizontal

Since hsbCAD2017 build 21.4.31, the strText can contain \P (new paragraph) to indicate a new line.

This is in accordance with MTEXT new line.

Beware \ is now an escape character, so to display it you need \\.

This escaping is on top of the literal Tsl string escaping, see example below.

---

[Example:

```
|||||||||||||||||||||||||||||||||||||||||||||||||||||||  
// Use the display system to draw some lines, and some texts  
  
Unit(1, "mm"); // define unit  
// define polylines and text strings  
Point3d pnt1 = _Pt0;  
Point3d pnt2 = _Pt0 + U(20)*_YU;  
Point3d pnt3 = _Pt0 + U(40)*_YU;  
Point3d pnt4 = _Pt0 + U(60)*_YU;  
PLine pl1(pnt1, pnt1+U(100)*_XU); // define PLine by 2 points  
PLine pl2(pnt2, pnt2+U(200)*_XU); // define PLine by 2 points  
PLine pl3(pnt3, pnt3+U(300)*_XU); // define PLine by 2 points  
PLine pl4(pnt4, pnt4+U(200)*_XU); // define PLine by 2 points  
String str1 = "First text !";  
String str2 = "Second text ?";  
String str3 = "Third text .";  
String str4 = "4th text .";  
  
// To display a simple line  
// with the default color of the instance  
// and the default linetype of the instance  
Display dp(-1); // choose default color: -1  
dp.draw(pl1); // draw the line  
dp.draw(str1,pnt1,_XU,_YU,1,1); // draw a string with default text style with lower left  
corner at pnt1  
// the default textstyle is taken from the default dimStyle, specified in the Hsb_settings.  
  
dp.color(4); // change the color to colorindex 4
```

```

dp.lineType("ACAD_ISO07W100"); // change linetype to specific linetype
dp.draw(pl2); // draw the line
dp.dimStyle("ISO-KR"); // specify dim and textstyle, also change the textHeight
dp.textHeight(10); // specify textHeight
dp.draw(str2,pnt2,_XU,_YU,0,0); // draw a string with center point at pnt2

dp.color(5); // change the color to colorindex 5
dp.lineType("ACAD_ISO10W100"); // change linetype to specific linetype
dp.draw(pl3); // draw the line
dp.dimStyle("ISO_KR2"); // specify dim and textstyle, also change the textHeight
dp.draw(str3,pnt3,_XU,_YU,0,-1); // draw a string with center X and top point at pnt3

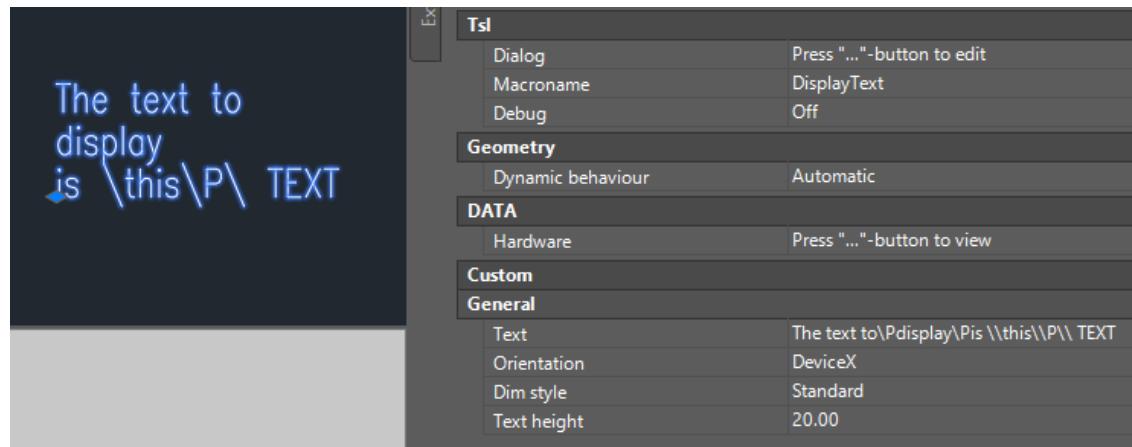
dp.color(-1); // change color back to default color
dp.lineType(""); // change linetype back to default
dp.draw(pl4); // draw the line
dp.dimStyle(""); // change dimstyle back to default, including the textHeight
dp.draw(str4,pnt4,_XU,_YU,1.3,1.3); // position text a little bit shifted to the right and top

```



—end example]

[Example showing multi-line support, since hsbCAD2017 build 21.4.31:



```
Unit(1, "mm");
```

```
// In a literal text inside tsl all \ characters need to be escaped
and become \\.
```

```
// Use \P, so literal \\P as a new line, conform the MTEXT autocad entity.
// To enter a \, escape it according to MTEXT, so this one becomes \\\
\.
PropString str(0, "The text to\\Pdisplay\\Pis \\\\"this\\\"\\P\\\" TEXT",
T("|Text|"));

String strOrientations[] = { "Model", "Device", "DeviceX"};
int nOrientations[] = { _kModel, _kDevice, _kDeviceX };
PropString strOrient(1, strOrientations, T("|Orientation|"));
int nOrient = nOrientations[strOrientations.find(strOrient, 0)];
PropString pDimStyle(2,_DimStyles , T("|Dim style|"));
PropDouble pTextHeight(0,U(20), T("|Text height|"));

Display dp(-1);
dp.showInDxa(TRUE);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
dp.draw(str, _Pt0, _xU, _yU,1,1, nOrient);
```

*—end example]*

## 16.7 Display on Element

All entities that belong to a certain Element (eg. a particular wall element), have a certain key in their primary layername. In other words, they belong to a certain Hsb\_Group. In addition to that, each element has a number of zones, with zone indices (-5,-4,...4,5). With each zone there is a particular character in the layername. A zone has different sets of entities: tools, dimensions,... Also for each set there is a character in the layername.

A TSL instance can display some graphics on an element, on a zone, with a particular entity character. In other words, the TSL can display some graphics on an Autocad Layer that contains all the required Hsb\_Group information. To do that the elemZone needs to be specified for the display. As with color, once specified, there is always one active element, with optionally a zone.

```
elemZone(Element el);
elemZone(Element el, int nZoneIndex, char cZoneCharacter);

el : the element that the following graphics will be drawn to
nZoneIndex : if specified, the zone index: (-5,-4,...4,5)
cZoneCharacter : specifies the entity set; should be equal to 'Z' for general items, 'T' for tools
```

But,... if the script is displaying on different zones and/or with different cZoneCharacters, or if the script adds different element tools to different zones, there is a difficulty, because the entity can only belong to one zone. Also, the entity must be put on a layer which is on, in order to be able to toggle the sublayers on which the element tools are visualized. Therefor it is needed that the tool is put on a layer that can be put on, but that essentially does not carry any visible items. The HSB-CAD software is designed that the E0 layer (cZoneCharacter 'E' with nZoneIndex 0) should be used for that.

So if the script adds different element tools to different zones, or if the script displays on different zone and/or with different cZoneCharacters, the assignToElementGroup (see also [Change Instance group](#)) should be called with 'E' and 0. So please assign the TSL instance to zone E0 with the call:

```
assignToElementGroup(eI, TRUE, 0, 'E');
```

[Example:

```
U(1,"mm");

if (_bOnInsert){
    _Pt0 = getPoint ();
    _Element.append(getElement());
    return;
}

// check if TSL is attached to element, if not, do nothing
if (_Element.length()==0) return; // 0 is a valid index now

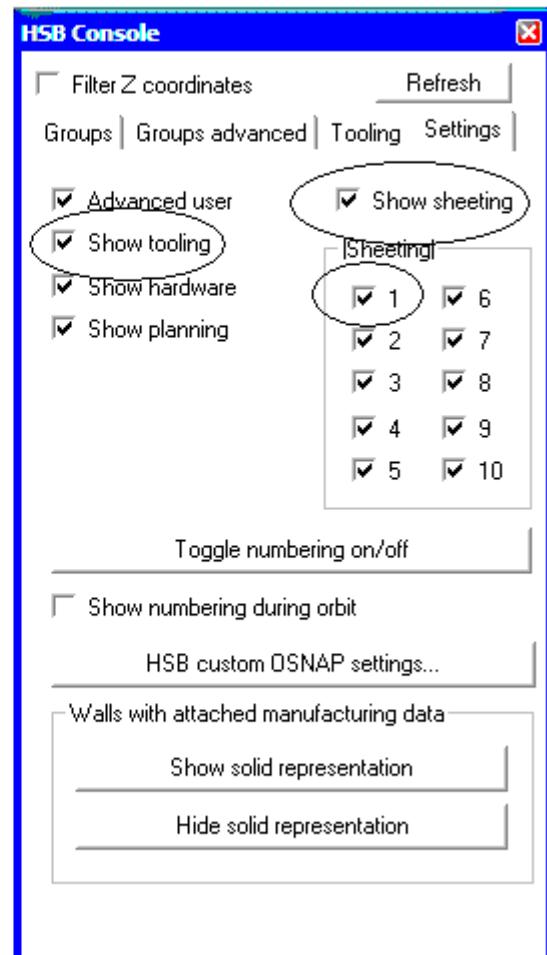
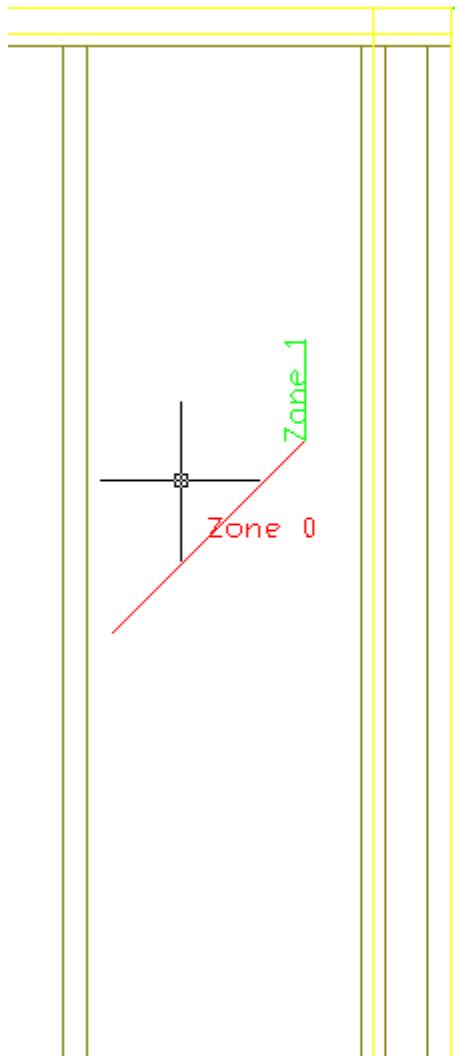
// use the elToUse element reference further on
Element elToUse = _Element[0];
CoordSys csEl = elToUse.coordSys(_Pt0); // coordSys of the element, but in point _Pt0
projected

// move the location point of the instance
_Pt0 = csEl.ptOrg();

// Define some points in element plane
Point3d pnt1 = _Pt0 - U(200)*csEl.vecX() - U(200)*csEl.vecY();
Point3d pnt2 = _Pt0 + U(200)*csEl.vecX() + U(200)*csEl.vecY();
Point3d pnt3 = pnt2 + U(200)*csEl.vecY();
PLine pl1(pnt1, pnt2); // define PLine by 2 points
PLine pl2(pnt2, pnt3); // define PLine by 2 points

Display dp(1); // choose color red
dp.textHeight(U(40)); // specify textHeight
dp.elemZone(elToUse,0,'Z'); // on zone 0 with character 'Z'
dp.draw(pl1); // draw the PLine
dp.elemZone(elToUse,0,'T'); // on zone 0 with character 'T'
dp.draw("Zone 0",_Pt0,csEl.vecX(),csEl.vecY(),1,1);

dp.color(3); // change color to green
dp.elemZone(elToUse,1,'Z'); // on zone 1 with character 'Z'
dp.draw(pl2); // draw the PLine
dp.elemZone(elToUse,1,'T'); // on zone 1 with character 'T'
dp.draw("Zone 1",pnt2,csEl.vecY(),-csEl.vecX(),1,1);
```



—end example]

[Example display element name:

```

U(1,"mm");

PropInt nZoneIndex(0,1,T("Zone index"));
PropString strZoneChar(0,"Z",T("Zone character"));

if (_bOnInsert){
    _Pt0 = getPoint ();
    _Element.append(getElement());
    return;
}

// check if TSL is attached to element, if not, do nothing

```

```

if (_Element.length()==0) return; // 0 is a valid index now

// use the elToUse element reference further on
Element elToUse = _Element[0];
CoordSys csEl = elToUse.coordSys(_Pt0); // coordSys of the element, but in point _Pt0
projected

// move the location point of the instance
Pt0 = csEl.ptOrg();

Display dp(1); // choose color red
dp.textHeight(U(40)); // specify textHeight

// add the script entity to the element group itself
assignToElementGroup(_Element0,TRUE,0,'E');

// dp.elemZone(elToUse,1,'Z'); // on zone 1 with character 'Z'
char cZone = strZoneChar.getAt(0); // take first character of string
dp.elemZone(elToUse,nZoneIndex,cZone);

String strToPrint = "Zone " + nZoneIndex + cZone + " - Element " + elToUse.number();
dp.draw(strToPrint ,_Pt0,csEl.vecX(),csEl.vecY(),1,1);

—end example]

```

## 16.8 Display in layout

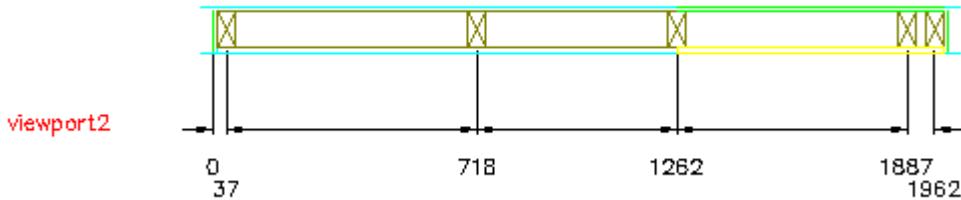
When adding a TSL in a paper space layout, the TSL will draw in paper space units. The mapping from paper space units to actual mm on your plotted paper is set through the Autocad "Page setup" -> "layout settings" -> "plot scale". Besides this global scaling, the following is worth mentioning:

- When a viewport displays something in the layout, there is a scaling involved from model space units to paper space units. This scaling is viewport dependent. It is this scale that changes, when you zoom inside the viewport. When the viewport is locked, you cannot change this scaling. When Hsb viewport data is added to the viewport, the scale is set, and the viewport is locked.
- When drawing a dimension in paper space to dimension distances shown in a viewport, the dimension style must be adjusted to the scale of the viewport. There are 2 scalings involved: the scaling of the measured distance, and the scaling of what is drawn (font, arrow size, line thickness,...). The scaling of the measured distance is set through in the dimension style, "Primary Units" -> "Measurement Scale" value. While the scaling of the graphics is set in the dimension style, "Fit" -> "Scale of Dimension Features" value.
- Drawing a dimension through TSL, through an Hsb\_Dimension, or directly through Autocad, should all react the same on the regarding scaling.
- If the TSL instance is not drawing any graphics, a default "marble" representation consisting of 3 circles is shown. To adjust the diameter of these circles to the paper space units, the setMarbleDiameter() function can be used.  
`setMarbleDiameter(double dDiam);`
- In TSL, to transform points from model space to paper space units, for a particular viewport, the coordSys() member function of a viewport must be used, see [Viewport](#).

- After a TSL has been added to paper space, additional viewports can be added to the TSL by using the hsb menu command "link tools".

The example below show a TSL that was added to the paper space in a layout. During insert is is added to a viewport. While the beams and the sheeting, shown from above are drawn inside the viewport, the dimension line is NOT. The TSL which is drawing the dimension line, checks the element of the viewport, collect the beams from it, and displays in paper space.

*[Example O-type with insert done in script. TSL should be added in Layout.]*



```

Unit(1,"mm"); // script uses mm
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Viewport vp = getViewPort(); // select viewport
    _Viewport.append(vp);

    return;
}

// set the diameter of the 3 circles, shown during dragging
setMarbleDiameter(U(4));

Display dp(1); // use color red
dp.dimStyle("HSB-DIM2"); // dimstyle was adjusted for display in paper space, sets
textHeight

// draw the scriptname at insert point
dp.draw(scriptName() ,_Pt0,_XW,_YW,1,1);

// do something for the last appended viewport only
if (_Viewport.length()==0) return; // _Viewport array has some elements
Viewport vp = _Viewport[_Viewport.length()-1]; // take last element of array
_Viewport[0] = vp; // make sure the connection to the first one is lost

// check if the viewport has hsb data
if (!vp.element().blsValid()) return;

CoordSys ms2ps = vp.coordSys();
CoordSys ps2ms = ms2ps; ps2ms.invert(); // take the inverse of ms2ps

```

```

Element el = vp.element();
int nZoneIndex = vp.activeZoneIndex();

DimLine InPs(_Pt0, _XW, _YW); // dimline in PS (Paper Space)
DimLine InMs = InPs; InMs.transformBy(ps2ms); // dimline in MS (Model Space)

Beam arBeams[] = el.beam(); // collect all beams from element
Point3d pntsMs[0]; // define array of points in MS
pntsMs.append(el.ptOrg()); // append origin of element
pntsMs.append(lnMs.collectDimPoints(arBeams, _kCenter));

Dim dim(lnMs,pntsMs,"<>","<>",_kDimCumm); // def in MS
dim.transformBy(ms2ps); // transfrom the dim from MS to PS
dp.draw(dim);

—end example]

```

## 16.9 BOM in layout

Another example of a TSL that is meaningful in a layout, is a TSL that displays a BOM. The following TSL displays some properties of all the beams that belong to the element at hand.

---

The following example uses a dimstyle to be set as property in the OPM, default: HSB-BOM. It uses beams and all sheetings + sort on posnum + added color prop + added material. The lunit and precision value of the length of the beams is set to 4 and 1, see first script lines.

*[Example O-type with insert done in script. TSL should be added in Layout.]*

Text can be changed in the OPM					
Number	Qty	Width	Height	Length	Material
5	16'-3"	3'-1"		364'-11"	Timber
5	5'-10"	3'-2"		100'	Timber
1	30'-10"	1'-6"		202'-9"	Timber
1	4'-2"	1'-6"		202'-9"	Timber
1	9'-3 1/2"	4'-4 3/4"		202'-9"	Timber
1	3'-1"	10'-5"		202'-9"	Timber
5	3'-1"	414'-10 3/4"			TENGEL
2	101'-8"	356'-2 1/4"			Spanplaat Groen

```

int nLUnit = 4; // architectural (only used for beam length)
int nPrec = 1; // precision (only used for beam length)

```

---

```
||||||||||||||||||||||||||||||||||||||||||||||||||
```

```

Unit(1, "mm"); // script uses mm

PropString strDimStyle(0, "HSB-BOM", "Dim style");
PropString propHeader(1, "Text can be changed in the OPM", "Table header");
PropInt nColorIndex(0,3, "Header color");

if (_bOnInsert) {

    _Pt0 = getPoint("Select upper left point of rectangle"); // select point
    Viewport vp = getViewport("Select the viewport from which the element is taken"); // select
    viewport
    _Viewport.append(vp);

    return;
}

// set the diameter of the 3 circles, shown during dragging
setMarbleDiameter(U(4));

Display dp(-1); // use color of entity for frame
dp.dimStyle(strDimStyle); // dimstyle was adjusted for display in paper space, sets
textHeight

double dCH = dp.textHeightForStyle("O",strDimStyle); // character height
double dCW = dp.textLengthForStyle("Een tekst.",strDimStyle)/10; // character width

if (_bOnDebug) {
// draw the scriptname at insert point
dp.draw(scriptName(), _Pt0, _XW, _YW, 1, 1);
}

// do something for the last appended viewport only
if (_Viewport.length()==0) return; // _Viewport array has some elements
Viewport vp = _Viewport[_Viewport.length()-1]; // take last element of array
_vp[0] = vp; // make sure the connection to the first one is lost

// check if the viewport has hsb data
if (!vp.element().blsValid()) return;

////////////////////

CoordSys ms2ps = vp.coordSys();
CoordSys ps2ms = ms2ps; ps2ms.invert(); // take the inverse of ms2ps
Element el = vp.element();
int nZoneIndex = vp.activeZoneIndex();

// build lists of items to display
int nNum = 0; // number of different items; make sure nNum is always the size of the
arrays
int arCount[0]; // counter of equal
String arW[0]; // width
String arL[0]; // length or height

```

```

int arPos[0]; // posnum
String arH[0]; // height
String arM[0]; // material

// collect the items
{ // beams
    Beam arBeam[0];
    arBeam = el.beam();
    for (int i=0; i<arBeam.length(); i++) {
        // loop over list items
        int bNew = TRUE;
        Beam bm = arBeam[i];
        for (int l=0; l<nNum; l++) {
            String strTim = String(bm.material());
            String strLength; strLength.formatUnit(bm.solidLength(),nLunit,nPrec);
            String strWidth; strWidth.formatUnit(bm.dW(),strDimStyle);
            String strHeight; strHeight.formatUnit(bm.dH(),strDimStyle);
            if (strTim=="") strTim = "Timber";
            if ((strHeight==arH[l])
                && (bm.posnum()==arPos[l])
                && (strWidth==arW[l])
                && (strLength==arL[l])
                && (String(strTim)==arM[l])
                )
            {
                bNew = FALSE;
                arCount[l]++;
                break; // out of inner for loop, we have found the equal one
            }
        }
        if (bNew) { // a new item for the list is found
            String strTim = String(bm.material());
            String strLength; strLength.formatUnit(bm.solidLength(),nLunit,nPrec);
            String strWidth; strWidth.formatUnit(bm.dW(),strDimStyle);
            String strHeight; strHeight.formatUnit(bm.dH(),strDimStyle);
            if (strTim=="") strTim = "Timber";
            arCount.append(1);
            arW.append(strWidth);
            arL.append(strLength);
            arPos.append(bm.posnum());
            arH.append(strHeight);
            arM.append(strTim);
            nNum++;
        }
    }
}
{ // sheeting
    Sheet arBeam[0];
    arBeam = el.sheet();
    for (int i=0; i<arBeam.length(); i++) {
        // loop over list items
        int bNew = TRUE;
        Sheet bm = arBeam[i];
        for (int l=0; l<nNum; l++) {

```

```

String strLength; strLength.formatUnit(bm.solidLength(),strDimStyle);
String strWidth; strWidth.formatUnit(bm.solidWidth(),strDimStyle);
if ( (strLength==arH[l])
    && (bm.posnum()==arPos[l])
    && (strWidth==arW[l])
    && (String(bm.material())==arM[l])
    ) {
    bNew = FALSE;
    arCount[l]++;
    break; // out of inner for loop, we have found the equal one
}
}
if (bNew) { // a new item for the list is found
    String strLength; strLength.formatUnit(bm.solidLength(),strDimStyle);
    String strWidth; strWidth.formatUnit(bm.solidWidth(),strDimStyle);
    arCount.append(1);
    arW.append(strWidth);
    arPos.append(bm.posnum());
    arH.append(strLength);
    arM.append(String(bm.material()));
    arL.append("");
    nNum++;
}
}

// here is the time to sort
// bubble sort on posnum
int nD;
String sD;
for (int b1=1; b1<nNum; b1++) {
    int lb1 = b1;
    for (int b2 = b1-1; b2>=0; b2--) {
        if (arPos[lb1]<arPos[b2]) {
            nD = arPos[b2]; arPos[b2] = arPos[lb1]; arPos[lb1] = nD;
            nD = arCount[b2]; arCount[b2] = arCount[lb1]; arCount[lb1] = nD;
            sD = arW[b2]; arW[b2] = arW[lb1]; arW[lb1] = sD;
            sD = arH[b2]; arH[b2] = arH[lb1]; arH[lb1] = sD;
            sD = arL[b2]; arL[b2] = arL[lb1]; arL[lb1] = sD;
            sD = arM[b2]; arM[b2] = arM[lb1]; arM[lb1] = sD;
            lb1=b2;
        }
    }
}

// make lists to display
String arL0[] = {"Number"};
String arL1[] = {"Qty"};
String arL2[] = {"Width"};
String arL3[] = {"Height"};
String arL4[] = {"Length"};
String arL5[] = {"Material"};

```

```

for (int l=0; l<nNum; l++) {
    int nD = arPos[l];
    if (nD<0) arL0.append("");
    else if (nD<10) arL0.append("00" + String(nD));
    else if (nD<100) arL0.append("0" + String(nD));
    else arL0.append(String(nD));
    arL1.append(String(arCount[l]));
    arL2.append(arW[l]);
    arL3.append(arH[l]);
    arL4.append(arL[l]);
    arL5.append(arM[l]);
}
nNum++; // added the labels

// find out column widths
int nW0 = 0, nW1 = 0, nW2 = 0, nW3 = 0, nW4 = 0, nW5 = 0;
for (int l=0; l<nNum; l++) {
    if (nW0<arL0[l].length()) nW0 = arL0[l].length();
    if (nW1<arL1[l].length()) nW1 = arL1[l].length();
    if (nW2<arL2[l].length()) nW2 = arL2[l].length();
    if (nW3<arL3[l].length()) nW3 = arL3[l].length();
    if (nW4<arL4[l].length()) nW4 = arL4[l].length();
    if (nW5<arL5[l].length()) nW5 = arL5[l].length();
}

if (_bOnDebug) {
    reportMessage("\n");
    for (int l=0; l<nNum; l++) {
        reportMessage("\n");
        reportMessage(arL0[l] + " ");
        reportMessage(arL1[l] + " ");
        reportMessage(arL2[l] + " ");
        reportMessage(arL3[l] + " ");
        reportMessage(arL4[l] + " ");
        reportMessage(arL5[l] + " ");
    }
    reportMessage("\n");
}

// ----- Draw outer border -----
// calculate the total border width and height
int nWH = propHeader.length() + 2; // header width in number of characters
int nWC = (nW0 + nW1 + nW2 + nW3 + nW4 + nW5 + 12); // coloms width in number of
characters
int nWT = (nWH<nWC)?nWC:nWH; // total table width
double dWT = nWT*dCW;
double dHT = 2*nNum*dCH + 4*dCH; // 4 is the height of the header
Point3d ptLL = _Pt0 - dHT*_YW;
Point3d ptUR = _Pt0 + dWT*_XW;
Point3d ptLR( ptUR.X(), ptLL.Y(), 0 );
Point3d ptUL( ptLL.X(), ptUR.Y(), 0 );
PLine plBorder(ptLL,ptLR,ptUR,ptUL);
plBorder.addVertex(ptLL);

```

```

dp.draw(plBorder);

// ----- Draw header -----
Point3d ptL0 = ptUL; // start from upper left
Point3d pt = ptL0 + dCW*_XW - dCH*_YW;
pt.vis();
dp.color(nColorIndex);
dp.draw(propHeader,pt,_XW,_YW,1,-1);
dp.color(-1);
Point3d ptHL = ptL0 - 3*dCH*_YW;
PLine plHeaderLine(ptHL, ptHL + dWT*_XW );
dp.draw(plHeaderLine);

// ----- Draw inner vertical lines -----
// calculate the total border width and height
Point3d ptT = ptHL;
Point3d ptB = ptLL;
PLine plLineV(ptT,ptB);
Vector3d vecMove(0,0,0);
vecMove = 2*dCW*_XW + nW0*dCW*_XW; plLineV.transformBy(vecMove); dp.draw(plLineV);
vecMove = 2*dCW*_XW + nW1*dCW*_XW; plLineV.transformBy(vecMove); dp.draw(plLineV);
vecMove = 2*dCW*_XW + nW2*dCW*_XW; plLineV.transformBy(vecMove); dp.draw(plLineV);
vecMove = 2*dCW*_XW + nW3*dCW*_XW; plLineV.transformBy(vecMove); dp.draw(plLineV);
vecMove = 2*dCW*_XW + nW4*dCW*_XW; plLineV.transformBy(vecMove); dp.draw(plLineV);

// ----- Draw text -----
ptL0 = ptUL - 4*dCH*_YW; // start from upper left, subtract header
for (int l=0; l<nNum; l++) { // loop over each line.
    if (l==0) dp.color(nColorIndex);
    else dp.color(-1);

    Point3d ptL = ptL0 - 2*dCH*l*_YW;
    Point3d pt = ptL;

    /*
    // right alignment
    pt = pt + 1*dCW*_XW + nW0*dCW*_XW; pt.vis(); dp.draw(arL0[l],pt,_XW,_YW,-1,-1);
    pt = pt + 2*dCW*_XW + nW1*dCW*_XW; pt.vis(); dp.draw(arL1[l],pt,_XW,_YW,-1,-1);
    pt = pt + 2*dCW*_XW + nW2*dCW*_XW; pt.vis(); dp.draw(arL2[l],pt,_XW,_YW,-1,-1);
    pt = pt + 2*dCW*_XW + nW3*dCW*_XW; pt.vis(); dp.draw(arL3[l],pt,_XW,_YW,-1,-1);
    pt = pt + 2*dCW*_XW + nW4*dCW*_XW; pt.vis(); dp.draw(arL4[l],pt,_XW,_YW,-1,-1);
    pt = pt + 2*dCW*_XW + nW5*dCW*_XW; pt.vis(); dp.draw(arL5[l],pt,_XW,_YW,-1,-1);
    */

    // horizontal line.
    if (l!=0) {
        Point3d ptLn = ptL + 0.4*dCH*_YW;
        PLine plLineH(ptLn, ptLn + dWT*_XW );
        dp.draw(plLineH);
    }

    // left alignment

```

```

pt = pt + 1*dCW*_XW + nW0*dCW*_XW; pt.vis(); dp.draw(arL0[I],pt,_XW,_YW,-1,-1); // right
pt = pt + 2*dCW*_XW ; pt.vis(); dp.draw(arL1[I],pt,_XW,_YW,1,-1); // left
pt = pt + 2*dCW*_XW + nW1*dCW*_XW; pt.vis(); dp.draw(arL2[I],pt,_XW,_YW,1,-1);
pt = pt + 2*dCW*_XW + nW2*dCW*_XW; pt.vis(); dp.draw(arL3[I],pt,_XW,_YW,1,-1);
pt = pt + 2*dCW*_XW + nW3*dCW*_XW; pt.vis(); dp.draw(arL4[I],pt,_XW,_YW,1,-1);
pt = pt + 2*dCW*_XW + nW4*dCW*_XW; pt.vis(); dp.draw(arL5[I],pt,_XW,_YW,1,-1);
}

```

*—end example]*

## 16.10 Viewport

The Viewport type represents a reference to a viewport in the Autocad layout. Since a viewport is used to look to the modelspace, the coordinate transformation from model space to paper space is an important member function, named coordSys.

When you link the TSL instance with a viewport in a layout, it will be automatically added to the predefined array \_Viewport.

**Viewport \_Viewport[];**

During \_bOnInsert, the getViewport() function can be used to query the user to select a viewport.

```

Viewport getViewport();
Viewport getViewport(String strPrompt);

```

When the command "Set viewports in layout" is called from the console group tree, the element that is currently displayed inside the viewport might change. Therefor, when this command is fired, the TSL's that are appended to paperspace will be automatically reexecuted. During this execution phase, the **\_bOnViewportsSetInLayout** will be set to TRUE.

How use viewport information in paper space is also more explained in the topic "[Display in layout](#)".

---

```

class Viewport { // is not derived from Entity

    CoordSys coordSys() const;
    void setCoordSys(CoordSys cs);

    double dScale() const;

    Point3d ptCenPS() const;
    void setPtCenPS(Point3d ptVal);

    double widthPS() const;
    void setWidthPS(double dVal);

    double heightPS() const;
    void setHeightPS(double dVal);

    Element element() const;
}

```

---

```

int activeZoneIndex() const;

ViewData viewData() const; // see also ViewData
void setGeoFromViewData(ViewData viewData);

// The following methods are only available in command mode: on insert and on
// custom context commands.
static int switchToModelSpace(); // return TRUE if successful, added build v21.3.50,
// v22.0.35
static int switchToPaperSpace(); // return TRUE if successful, added build v21.3.50,
// v22.0.35
static int inLayoutTab(); // return TRUE if in a layout tab, return FALSE if in a model tab,
// added build v21.4.41, v22.0.74
// look also to Layout for currentLayout name.
static int inPaperSpace(); // return TRUE if in a layout tab, and not inside a viewport,
// added build v21.4.41, v22.0.74

Layout getLayout() const; // (added 24.1.91, 25.1.64) returns the Layout of the Viewport.

int setCurrent(); // will only have effect being in ModelSpace, so call
// switchToModelSpace before calling setCurrent. Return TRUE if successful
// v22.0.74

void turnGroupVisibilityOn(Group group); // added V23.4.4 see also Group
void turnGroupVisibilityOff(Group group); // added V23.4.4
int groupVisibilityIsOn(Group group); // added V23.4.4, return TRUE if all layers of group
// are thawed in viewport, else return FALSE
int groupVisibilityIsOff(Group group); // added V23.4.4, return TRUE if all layers of group
// are frozen in viewport, else return FALSE
};


```

---

**CoordSys** coordSys() const;  
**void** setCoordSys(**CoordSys** cs);

Get or set the transformation matrix from the model space to the paper space units. When you have a point, or any other geometrical entity in TSL, that is defined in model space, you can transformBy with this CoordSys to find the point in paper space. If the transformation is needed from paper space to model space, you need to invert the CoordSys that is returned, by calling the invert() function on it.

**double** dScale() const;

Return the custom scale of the viewport. The custom scale is the length in paper space units of a unit length in model space.

**Point3d** ptCenPS() const;  
**void** setPtCenPS(**Point3d** ptVal);

Return and set the center of the viewport in paper space coordinates.

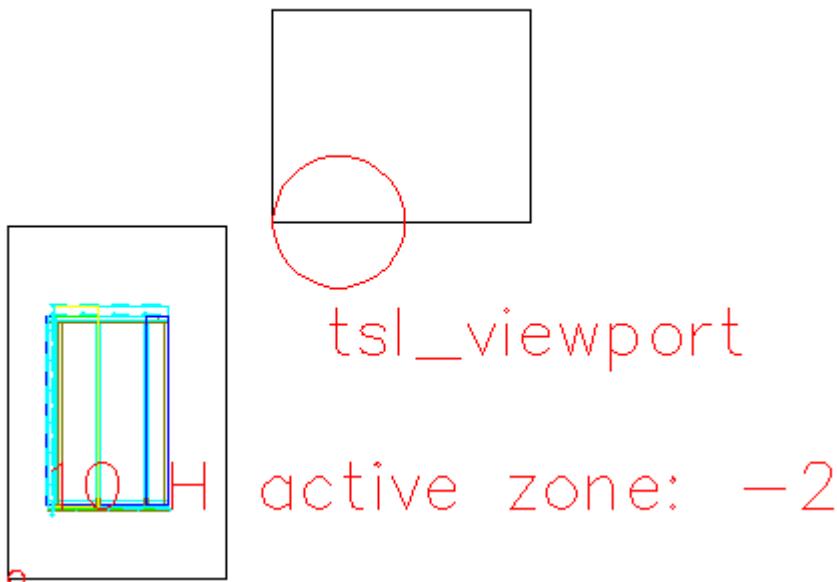
```
double widthPS() const;
void setWidthPS(double dVal);
double heightPS() const;
void setHeightPS(double dVal);
```

Return and set the width and the height in paper space dimensions of the viewport. So the widthPS() is the dimension in the \_XW direction, and heightPS() is the dimension in the \_YW direction.

```
Element element() const;
int activeZoneIndex() const;
```

If the viewport has Hsb data attached to it, containing an element reference, and it active zone, these two functions allows them to query them. When the command "Set viewports in layout" is called from the console group tree, the element might change. Therefor, when this command is fired, the TSL's that are appended to paperspace will be automatically reexecuted.

[Example of O-type with insert done in script, scriptname "tsl\_viewport", 2 viewports linked with the TSL and one of them showing element "10" of type "H":



```
Unit(1, "mm"); // script uses mm
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Viewport vp = getViewport(); // select viewport
    _Viewport.append(vp);
```

```

        return;
    }

    // set the diameter of the 3 circles, shown during dragging
    setMarbleDiameter(U(4));

    Display dp(1); // use color red
    dp.dimStyle("HSS-DIM2"); // dimstyle was adjusted for display in
    paper space, sets textHeight

    // draw the scriptname at insert point
    dp.draw(scriptName() ,_Pt0,_XW,_YW,1,1);

    // do something for all viewports
    for (int v=0; v<_Viewport.length(); v++) {
        CoordSys csVp = _Viewport[v].coordSys();
        Element el = _Viewport[v].element();
        int nZoneIndex = _Viewport[v].activeZoneIndex();
        double dX = _Viewport[v].widthPS();
        double dY = _Viewport[v].heightPS();
        double dS = _Viewport[v].dScale();
        Point3d ptCen = _Viewport[v].ptCenPS();

        Point3d ptA = ptCen - 0.5*dX*_XW - 0.5*dY*_YW; // lower left corner
        of viewport
        double dDist = dS*U(200); // scale distance 200 from model to paper
        space
        Point3d ptB = ptA + dDist*_XW;
        PLine plCir(_ZW); // define poly with nomal _ZW
        plCir.addVertex(ptA);
        plCir.addVertex(ptB,1); // bulge 1 means half a circle
        plCir.addVertex(ptA,1); // close polyline with first point
        dp.draw(plCir);

        if (el.bIsValid()) { // viewport has hsb data attached and set
            String str = el.number() + " " + el.code() + " active zone: " +
            nZoneIndex ;
            Point3d ptElinVp = el.ptOrg();
            ptElinVp.transformBy(csVp); // transform point from model to
            paper space
            dp.draw(str ,ptElinVp ,_XW,_YW,1,1);
        }
    }
}

```

*—end example]*

*[Example of O-type with insert done in script, the position of the viewport is set, the size is changed, and the transformation is taken from a second viewport.*

```

Unit(1,"mm"); // script uses mm
if (_bOnInsert) {

```

```

    _Pt0 = getPoint(); // select point
    _Viewport.append(getViewport()); // select viewport
    _Viewport.append(getViewport()); // select second viewport

    return;
}

// set the diameter of the 3 circles, shown during dragging
setMarbleDiameter(U(4));

Display dp(1); // use color red

// draw the scriptname at insert point
dp.draw(scriptName(), _Pt0, _XW, _YW, 1, 1);

// check running conditions
if (_Viewport.length()<2) return;
Viewport vp = _Viewport[0];
Viewport vp2 = _Viewport[1];

// set location in paper space
vp.setPtCenPS(_Pt0);

// switch width and height in paper space
double dWOrig = vp.widthPS();
vp.setWidthPS(vp.heightPS());
vp.setHeightPS(dWOrig);

// change the coordinate system: transformation from model to paper
CoordSys csVp2 = vp2.coordSys();
vp.setCoordSys(csVp2);

// now draw something
CoordSys csVp = vp.coordSys(); // last set coordSys
double dx = vp.widthPS();
double dy = vp.heightPS();
double ds = vp.dScale();
Point3d ptCen = vp.ptCenPS();

Point3d ptA = ptCen - 0.5*dx*_XW - 0.5*dy*_YW; // lower left corner of
viewport
double dDist = ds*U(200); // scale distance 200 from model to paper
space
Point3d ptB = ptA + dDist*_XW;
PLine plCir(_ZW); // define poly with nomal _ZW
plCir.addVertex(ptA);
plCir.addVertex(ptB,1); // bulge 1 means half a circle
plCir.addVertex(ptA,1); // close polyline with first point
dp.draw(plCir);

—end example]

```

[Example of O-type with insert done in script, calculate the viewport transformation from an element

```

Unit(1,"mm"); // script uses mm
if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Viewport vp = getViewport(); // select viewport
    _Viewport.append(vp);

    return;
}

// set the diameter of the 3 circles, shown during dragging
setMarbleDiameter(U(4));

Display dp(1); // use color red

// draw the scriptname at insert point
dp.draw(scriptName(), _Pt0, _XW, _YW, 1, 1);

// check running conditions
if (_Viewport.length()==0) return;
Viewport vp = _Viewport[0];
Element el = vp.element();

// check if this viewport has hsb_data
if (!el.bIsValid()) return;

// get the coordinate system of the element, and calculate a new
ModelSpaceToPaperSpace transformation
CoordSys csEl = el.coordSys();
Point3d ptCen = vp.ptCenPS();

CoordSys csMs2Ps;
double dScale = 0.02; // 1/50
csMs2Ps.setToAlignCoordSys(csEl.ptOrg(), csEl.vecX(), csEl.vecY(), csEl.
vecZ(), ptCen, _XW*dScale, _YW*dScale, _ZW*dScale);
vp.setCoordSys(csMs2Ps); // finally set the transformation to the
viewport

—end example]

```

[Example of O-type with insert done in script, switchToModelSpace switchToPaperSpace and setCurrent

```

if (_bOnInsert)
{
    _Pt0 = getPoint(); // select point
    return;
}

String strAddViewport = "add Viewport";
addRecalcTrigger(_kContext, strAddViewport );
if (_bOnRecalc && _kExecuteKey==strAddViewport)

```

```

{
    Viewport vp = getViewport(); // select viewport
    _Viewport.append(vp);
    reportMessage("amount of viewports: " + _Viewport.length());
}

String strSwitchToModelSpace = "switchToModelSpace";
addRecalcTrigger(_kContext, strSwitchToModelSpace );
if (_bOnRecalc && _kExecuteKey==strSwitchToModelSpace)
{
    int bSuccess = Viewport().switchToModelSpace();
    reportMessage("switchToModelSpace: " + bSuccess);
}

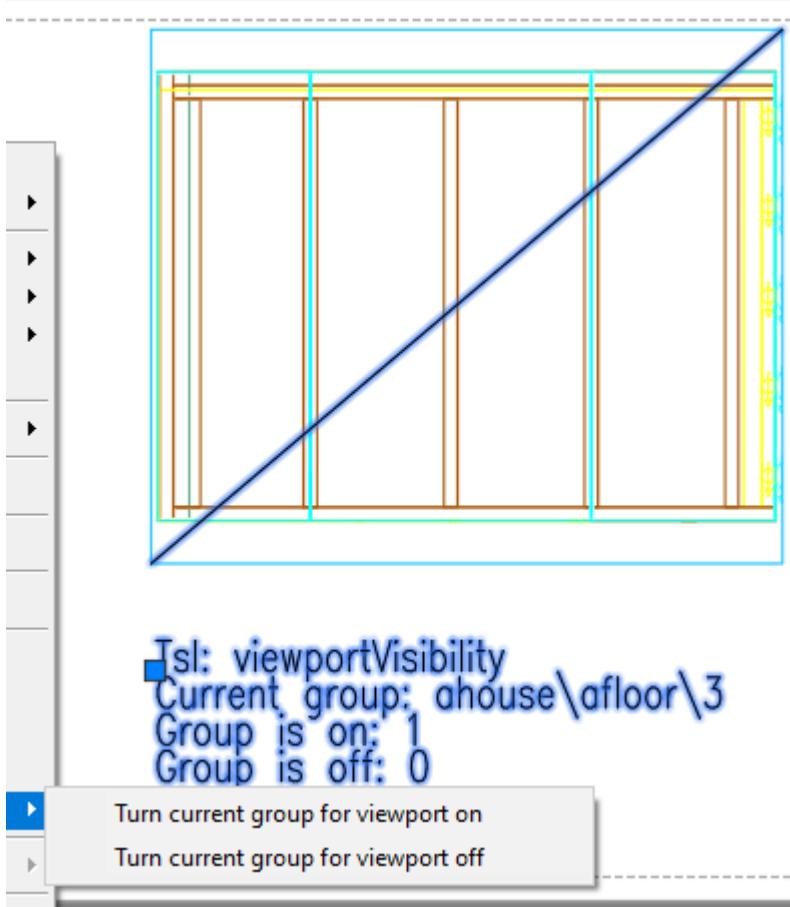
String strSwitchToPaperSpace = "switchToPaperSpace";
addRecalcTrigger(_kContext, strSwitchToPaperSpace );
if (_bOnRecalc && _kExecuteKey==strSwitchToPaperSpace)
{
    int bSuccess = Viewport().switchToPaperSpace();
    reportMessage("switchToPaperSpace: " + bSuccess);
}

String strViewportSetCurrent = "Viewport.setCurrent";
addRecalcTrigger(_kContext, strViewportSetCurrent );
if (_bOnRecalc && _kExecuteKey==strViewportSetCurrent)
{
    int bSuccess = Viewport().switchToModelSpace();
    reportMessage("switchToModelSpace: " + bSuccess);
    bSuccess = _Viewport[0].setCurrent(); // need to be in model
    space to make viewport current!
    reportMessage("Viewport.setCurrent: " + bSuccess);
}

```

*[end example]*

*[Example of O-type accessing the group visibility]*



```

Unit(1,"mm"); // script uses mm

PropString pDimStyle(0,_DimStyles , T("|Dim style|"));
PropDouble pTextHeight(0,U(5), T("|Text height|"));

if (_bOnInsert) {

    _Pt0 = getPoint(); // select point
    Viewport vp = getViewport(); // select viewport
    _Viewport.append(vp);

    return;
}

if (_Viewport.length() == 0)
{
    eraseInstance();
    return;
}

viewport vp0 = _Viewport[0];
Group grp = _kCurrentGroup;

```

```

String strTurnOn = T("|Turn current group for viewport on|");
addRecalcTrigger(_kContext, strTurnOn );
if (_bOnRecalc && _kExecuteKey==strTurnOn)
{
    vp0.turnGroupVisibilityOn(grp);
}

String strTurnOff = T("|Turn current group for viewport off|");
addRecalcTrigger(_kContext, strTurnOff );
if (_bOnRecalc && _kExecuteKey==strTurnOff)
{
    vp0.turnGroupVisibilityOff(grp);
}

String strLines[0];
strLines.append("Tsl: " +scriptName());
strLines.append("Current group: " + grp.name());
strLines.append("Group is on: " + vp0.groupVisibilityIsOn(grp));
strLines.append("Group is off: " + vp0.groupVisibilityIsOff(grp));

Display dp(-1);
dp.dimStyle(pDimStyle);
dp.textHeight(pTextHeight);
for (int l=0; l<strLines.length(); l++) {
    Vector3d vecO = -l*1.2*pTextHeight*_YW;
    dp.draw(strLines[l],_Pt0+vecO ,_XW, _YW, 1,1);
}

CoordSys csVp = vp0.coordSys();
double dx = vp0.widthPS();
double dy = vp0.heightPS();
Point3d ptCen = vp0.ptCenPS();
Point3d ptA = ptCen -0.5*dx*_XW - 0.5*dy*_YW; // lower left corner of
viewport
Point3d ptB = ptA + dx*_XW + dy*_YW;
dp.drawLine(ptA, ptB));

```

*—end example]*

### 16.10.1 ViewData

The ViewData type represents a description of a viewport in paperspace or modelspace. The ViewData has a coordinate transformation which represents the mapping from what is shown inside the view to the location in the space where the view is shown. So for a viewport in paperspace, the coordinate transformation represents the transformation from model to paperspace.

The ViewData information can be retrieved from a [Viewport](#) with the

```

ViewData Viewport::viewData() const;
void Viewport::setViewData(ViewData viewData);

```

See also [Shopdraw ViewData](#).

```

class ViewData { // is not derived from Entity

    CoordSys coordSys() const; // transformation from model space to world paper space
    void setCoordSys(CoordSys cs);

    CoordSys coordSysPS() const; // location of the View in paper space, so in fact
        // transformation from local view, to world paper space.
    void setCoordSysPS(CoordSys cs);

    double dScale() const;

    Point3d ptCenPS() const; // equal to coordSysPS().ptOrg()
    void setPtCenPS(Point3d ptVal);

    double widthPS() const; // width in coordSysPS().vecX() direction
    void setWidthPS(double dVal);

    double heightPS() const; // width in coordSysPS().vecY() direction
    void setHeightPS(double dVal);

    // The show set is the set of entities that is shown in the view. It could be an Element,
    // GenBeam, or a set defined by a group, module, subassembly,...
    String showSetName() const; // could be the group name or module name in case it is
        // that type of show set
    void setShowSetName(String strVal);
    int showSetIndex() const; // is the zone index in case of an element
    void setShowSetIndex(int nVal);

    Entity[] showSetDefineEntities() const; // Set of entities that defines the showset eg the
        // element for an element set, the subassembly tsl instance for a
        // subassembly,..
    void setShowSetDefineEntities(<Entity>[] arEntities);

    Entity[] showSetEntities() const; // Set of entities that represent the showset eg the beams
        // and sheet for an element, the entities of a subassembly tsl,..
    void setShowSetEntities(<Entity>[] arEntities);

    // The view parameters refer to the Entity or Viewport from which this ViewData came
    // from.
    String viewHandle() const; // the handle as string of the Viewport
    void setViewHandle(String strVal);

    String scaleGroup() const; // (added v21.2.8) only relevant in shopdraw context.
    void setScaleGroup(String strVal);

    String viewUserId() const; // the user id that was given to the show draw view.
    void setViewUserId(String strVal);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;
}

```

```

static ViewData[] convertFromSubMap(Map map, String strSubMapKey); // convert the
    submap to an array of ViewData
static ViewData[] convertFromSubMap(Map map, String strSubMapKey, int nVerboseMode);

static Map convertToMap(ViewData[] viewDataList); // convert to a map which can be
    added as a submap that could be read with convertFromSubMap
static Map convertToMap(ViewData viewData);

static int findDataForViewport(ViewData[] viewDataListToSearch, Entity entView); // returns -1
    or index in array
static int findDataForViewport(ViewData[] viewDataListToSearch, Viewport viewport); // returns
    -1 or index in array
};

```

---

[Example O-type TSL that works both in PaperSpace and in ShopDrawSpace:

```

Unit(1, "mm");

String sPaperSpace = T("|paper space|");
String sShopdrawSpace = T("|shopdraw multipage|");
String sArSpace[] = {sPaperSpace, sShopdrawSpace};
PropString sSpace(0, sArSpace, T("|Drawing space|"));

String sArDimStyles[0]; sArDimStyles= _DimStyles;
PropString sDimStyle(6, sArDimStyles, T("|Dimension style|"));

// ----

if (_bOnInsert) {
    if (insertCycleCount()>1) { eraseInstance(); return; } // only
    insert once
    showDialog();
    _Pt0 = getPoint(T("|Select location|")); // select point

    if (sSpace==sPaperSpace) {
        Viewport vp = getViewport(T("|Select the viewport|")); // select viewport
        _Viewport.append(vp);
    }
    else if (sSpace==sShopdrawSpace) {
        Entity ent = getShopDrawView(T("|Select the view entity|")); // select ShopDrawView
        _Entity.append(ent);
    }

    return;
}

// determine the space type depending on the contents of _Entity[0]
// and _Viewport[0]

```

---

```

{
    if (_viewport.length()>0)
        sSpace.set(sPaperSpace);
    else if (_Entity.length()>0 &&
_Entity[0].bIsKindOf(ShopDrawView()))
        sSpace.set(sShopdrawSpace);
    else {
        eraseInstance(); // this Tsl not allowed to be appended to
model space
        return;
    }
    sSpace.setReadOnly(TRUE);
}

int bError = 0; // 0 means FALSE = no error

// set of variables that change depending on the type of space
CoordSys ms2ps; // default to identity transformation

if (sSpace==sShopdrawSpace)
{
    ShopDrawView sv;
    if (_Entity.length()>0) sv = (ShopDrawView)_Entity[0];
    if (!bError && !sv.bIsValid()) bError = 1;

    // interprete the list of ViewData in my _Map
    ViewData arViewData[] = ViewData().convertFromSubMap(_Map,
_kOnGenerateShopDrawing + "\\" + _kViewDataSets,0); // 2 means
verbose
    int nIndFound = ViewData().findDataForViewport(arViewData, sv);//
find the viewData for my view
    if (!bError && nIndFound<0) bError = 2; // no viewData found

    if (!bError) {
        ViewData vwData = arViewData[nIndFound];
        ms2ps = vwData.coordSys(); // transformation to view
    }
}

// if it is a viewport
else if (sSpace==sPaperSpace)
{
    if (!bError && _viewport.length()==0) bError = 3;
    if (!bError) {
        Viewport vp = _viewport[0];
        ms2ps = vp.coordSys();
    }
}

CoordSys ps2ms = ms2ps; ps2ms.invert(); // take the inverse of ms2ps

```

```

setMarbleDiameter(U(4)); // set the diameter of the 3 circles, shown
during dragging

// Display
if (!bError)
{
    Display dp(-1);
    dp.dimStyle(sDimStyle);

    // dimstyle scale need to be set correct according to the space
    double dScale = ps2ms.scale();
    String strScale;
    strScale.formatUnit(dScale,sDimStyle); // the dimstyle specifies
the precision

    Vector3d vX = _xW, vY = _yW;
    dp.draw(T("|Scale factor|: ") + strScale, _Pt0, vX, vY, 1,1);
}

if (bError) {
    Display dp(-1);
    Vector3d vX = _xW, vY = _yW;
    dp.draw(scriptName() + " has error code: " + bError, _Pt0, vX, vY,
1,1);
}

—end example]

```

## 16.11 Hatch

The Hatch type holds the information to hatch a [PlaneProfile](#) while calling draw on [Display](#).

A list of all available hatch pattern names is available as [\\_HatchPatterns](#), see section on [general predefined variables](#).

---

```

class Hatch {

    Hatch(String strPatternName, double dPatternScale);

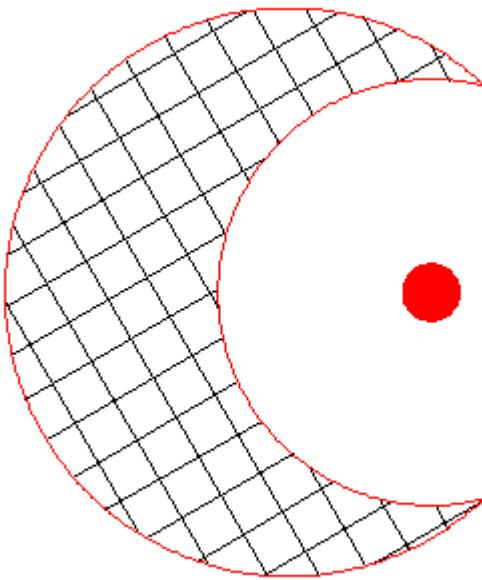
    void setAngle(double dAngleInDegrees);

}

```

---

*[Example O-type with insert implemented:*



```
U(1,"mm");

Vector3d vec = U(100)*_xu;
double dRefLength = vec.length();

PLine pCirBig;
pCirBig.createCircle(_Pt0,_zu,dRefLength*2);
//pCirBig.vis();
PlaneProfile profMoon(pCirBig); // define profile

PLine pCirSmall;
pCirSmall.createCircle(_Pt0+vec,_zu,dRefLength*1.5);
//pCirSmall.vis();
profMoon.joinRing(pCirSmall,TRUE); // add an opening (subtract ring)

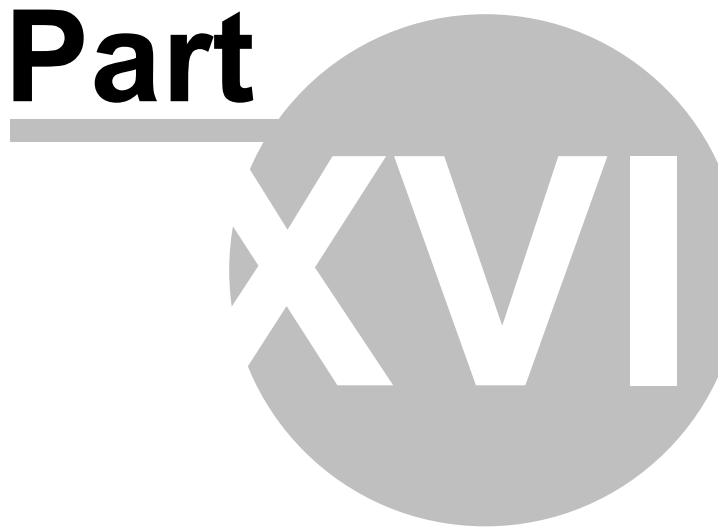
PLine pCirSmaller;
pCirSmaller.createCircle(_Pt0+vec,_zu,0.2*dRefLength);
//pCirSmaller.vis();
PlaneProfile profStar(pCirSmaller);

Hatch hatchNet("NET",U(10));
hatchNet.setAngle(30); // 30 degrees
Hatch hatchSolid("SOLID",U(1));

Display dp(-1); // keep color of entity
dp.draw(profMoon, hatchNet); // draw hatched inner of profile
dp.color(1); // red
dp.draw(profMoon); // draw profile
dp.draw(profStar, hatchSolid); // draw profile
```

*—end example]*

**Part**



## 17 Nester

### 17.1 NesterChild

NesterChild is a class that is part of the Nester Tsl Api. It represents a generic child shape to be placed inside a [NesterMaster](#).

---

```
class NesterChild { // (Do not use in versions before 17.0.21)

    NesterChild(String strOriginatorId, PlaneProfile ppShape);

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    PlaneProfile profShape() const; // see PlaneProfile
    void setProfShape(PlaneProfile prf); // Important: the vecZ of the profile needs to be
                                         codirectional with vecZ of the child panel.

    PlaneProfile profToolShape() const; // see PlaneProfile
    void setProfToolShape(PlaneProfile prf); // currently only the outer ring is used

    int nestInOpenings() const; // default TRUE
    void setNestInOpenings(int n); // set to TRUE to allow openings for placing smaller pieces

    String originatorId() const;
    void setOriginatorId(String str);

    double rotationAllowance() const; // value in degrees
    void setRotationAllowance(double n); // set value in degrees: 0 for any angle, 90 for rotations
                                         per 90 degrees, 360 for no rotations allowed
};
```

---

```
PlaneProfile profToolShape() const;
void setProfToolShape(PlaneProfile prf);
```

The difference between the profShape and profToolShape is the additional space that is needed to process its tools. The profShape expresses the shape of the solid that needs to be optimized. The profToolShape is typically bigger then the profShape, and has for each edge the required offset that needs to be present for the tools (saw, mill,...) to pass.

---

## 17.2 NesterMaster

NesterMaster is a class that is part of the Nester Tsl Api. It represents a generic master pline which is to contain [NesterChild](#) instances.

---

```
class NesterMaster { // (Do not use in versions before 17.0.21)

    NesterMaster(String strOriginatorId, PlaneProfile ppShape); // only outer ring of ppShape is
                                                               taken

    visualize(int indColor = -1) const;
    vis(int indColor = -1) const;

    transformBy(CoordSys csTransformationMatrix);
    transformBy(Vector vecTranslate);

    PlaneProfile profShape() const; // see PlaneProfile
    void setProfShape(PlaneProfile prf);

    String originatorId() const;
    void setOriginatorId(String str);

    int isStockPiece() const; // default FALSE
    void setIsStockPiece(int n); // set to TRUE to make this master a stock item

};
```

---

## 17.3 NesterData

NesterData is a class that is part of the Nester Tsl Api. It holds the parameters for the [NesterCaller](#).

The [NesterData::nesterToUse\(\)](#) returns one of the following predefined variables of type **int**:

[\\_kNTTestNester](#),  
[\\_kNTAutoNesterV4](#),  
[\\_kNTAutoNesterV6](#),  
[\\_kNTAutoNesterV5](#),  
[\\_kNTRectangularNester](#) (added to v21.0.6)

---

```

class NesterData { // (Do not use in versions before 17.0.21)

NesterData();

double allowedRunTimeSeconds() const;
void setAllowedRunTimeSeconds(double dTimeInSeconds); // maximum time that the nester is
allowed to run

double minimumSpacing() const;
void setMinimumSpacing(double dGap); // distance between the NesterChilds to keep

int generateDebugOutput() const;
void setGenerateDebugOutput(int n); // set to TRUE for debug output

double childOffsetX() const;
void setChildOffsetX(double dDist); // shift NesterChild inside NesterMaster
double childOffsetY() const;
void setChildOffsetY(double dDist); // shift NesterChild inside NesterMaster

int nesterToUse() const;
void setNesterToUse(int n);
static String nesterNameFromType(int nNesterType); // nNesterType should be:
_kNTTestNester, _kNTAutoNesterV4, _kNTAutoNesterV6, _kNTAutoNesterV5,
_kNTRectangularNester

};

```

---

[Example O-type TSL:

```

Unit(1, "mm");

NesterData nd;
int arIntNesters[] = {_kNTTestNester, _kNTAutoNesterV4,
_kNTAutoNesterV5, _kNTAutoNesterV6, _kNTRectangularNester};
String arStrNesters[arIntNesters.length()];
for (int i=0; i<arIntNesters.length(); i++)
    arStrNesters[i] = nd.nesterNameFromType(arIntNesters[i]);

PropString pNesterToUse(0, arStrNesters, T("|Nester to use|"));
int nNesterToUse = arIntNesters[arStrNesters.find(pNesterToUse, 0)];

nd.setAllowedRunTimeSeconds(U(60)); // seconds
nd.setMinimumSpacing(U(10));
nd.setGenerateDebugOutput(1);
nd.setChildOffsetX(U(10));

```

---

```

nd.setChildOffsetY(0(10));
nd.setNesterToUse(nNesterToUse);

reportMessage("\n allowedRunTimeSeconds:
"+nd.allowedRunTimeSeconds() );
reportMessage("\n minimumSpacing: "+nd.minimumSpacing() );
reportMessage("\n generateDebugOutput: "+nd.generateDebugOutput() );
reportMessage("\n childOffsetX: "+nd.childOffsetX() );
reportMessage("\n childOffsetY: "+nd.childOffsetY() );
reportMessage("\n nesterToUse:
"+nd.nesterNameFromType(nd.nesterToUse()) );

```

—end example]

## 17.4 NesterCaller

NesterCaller is a class that is part of the Nester Tsl Api. It is used to fire the nesting engine.

The **NesterCaller::nest()** returns one of the following predefined variables of type **int**:  
**\_KNROk, \_KNRNoDongle, \_KNRError**

---

```

class NesterCaller { // (Do not use in versions before 17.0.21)

    NesterCaller();

    int addChild(NesterChild child); // all items need to have a distinct originatorId, if not FALSE is
                                    // returned.
    int addMaster(NesterMaster master); // all items need to have a distinct originatorId, if not
                                    // FALSE is returned.

    int childCount(); // return total number of NesterChild in internal array.
    int masterCount(); // return total number of NesterMaster in internal array.

    NesterChild childAt(int nChildIndex);
    NesterMaster masterAt(int nMasterIndex);
    String childOriginatorIdAt(int nChildIndex);
    String masterOriginatorIdAt(int nMasterIndex);

    int nest(NesterData settings); // call the nester engine. Returns one of _KNROk,
                                // _KNRNoDongle, _KNRError...
    int nest(NesterData settings, int bSilent); // added 17.1.13. Default state of bSilent is
                                                // FALSE. If silent, the nester dialog will not show.

    // Nesting results are presented through indexes into the internal child and master arrays
    int[] nesterMasterIndexes();
    int[] leftOverChildIndexes();
    int[] leftOverMasterIndexes();

```

```

int[] childListForMasterAt(int nMasterIndex);
CoordSys[] childWorldXformIntoMasterAt(int nMasterIndex);
CoordSys[] childXformWithinMasterAt(int nMasterIndex);

};

```

---

*[Example O-type TSL that illustrates the nesting. It must be noted that hsbCAD does NOT have a real nester. There is only a test nester, which not really nests, but allows to check the communication protocol.*

```

// When added to the drawing, a number of sheets need to be selected,
// which are then stored in the _Sheet array.
// Once in the drawing, there is a custom action on the Tsl that
// allows to nest these sheets in a selected PLine.

Unit(1, "mm");

PropInt pAccept(0, 0, T("Accept nester result"));

if (_bOnInsert) {
    PrEntity ssE(T("|Select a set of sheets to be nested|"),
Sheet());
    if (ssE.go()) {
        _Sheet = ssE.sheetSet();
    }

    return;
}

NesterData nd;
nd.setAllowedRunTimeSeconds(U(320)); // seconds
nd.setMinimumSpacing(U(10));
nd.setGenerateDebugOutput(1);
nd.setNesterToUse(_kNTTestNester);

String strNestAction= T("|Nest in EntPLine|");
addRecalcTrigger(_kContext, strNestAction);
if (_bOnRecalc && _kExecuteKey==strNestAction)
{
    EntPLine entPline = getEntPLine();
    if (entPline.bIsValid()) {

        // construct a NesterCaller object adding masters and
        child�
        NesterCaller nester;
        for (int s=0; s<_Sheet.length(); s++) {
            Sheet sh = _Sheet[s];

```

```

        NesterChild nc(sh.handle(), sh.profShape());
        nester.addChild(nc);
    }
    NesterMaster nm(entPline.handle(),
PlaneProfile(entPline.getPLine()));
    nester.addMaster(nm);

    // report NesterCaller object content
    reportMessage("\nMasterList" );
    for (int m=0; m<nester.masterCount(); m++) {
        NesterMaster master = nester.masterAt(m);
        reportMessage("\nMaster "+m+
"+nester.masterOriginatorIdAt(m) + " == " + master.originatorId() );
    }
    reportMessage("\nChildList" );
    for (int c=0; c<nester.childCount(); c++) {
        NesterChild child = nester.childAt(c);
        reportMessage("\nChild "+c+
"+nester.childOriginatorIdAt(c) + " == " + child .originatorId() );
    }

    // do the actual nesting
    int nSuccess = nester.nest(nd);
    reportMessage("\nNestResult: "+nSuccess);
    if (nSuccess!=_kNRok) {
        reportNotice(T("|Not possible to nest|"));
        if (nSuccess==_kNRNoDongle)
            reportNotice(T("|No dongle present|"));
        return;
    }

    // examine the nester result
    reportMessage("\nLeft over MasterList" );
    int arLeftOverMaster[] =
nester.leftOverMasterIndexes();
    for (int m=0; m<arLeftOverMaster.length() ; m++) {
        reportMessage("\nMaster "+m+
"+nester.masterOriginatorIdAt(m));
    }
    reportMessage("\nLeft over ChildList" );
    int arLeftOverChild[] = nester.leftOverChildIndexes();
    for (int c=0; c<arLeftOverChild.length(); c++) {
        reportMessage("\nChild "+c+
"+nester.childOriginatorIdAt(c));
    }

    // loop over the nester masters
    int arMaster[] = nester.nesterMasterIndexes();
    for (int m=0; m<arMaster.length(); m++) {
        int nIndexMaster = arMaster[m];

```

```

        reportMessage("\nResult "+nIndexMaster +
"+nester.masterOriginatorIdAt(nIndexMaster) );

        int arChild[] =
nester.childListForMasterAt(nIndexMaster);
        CoordSys arWorldXformIntoMaster[] =
nester.childWorldXformIntoMasterAt(nIndexMaster);
        if
(arChild.length()!=arWorldXformIntoMaster.length()) {
            reportNotice("Nesterresult invalid");
            break;
        }

// show the childs within the master
for (int c=0; c<arChild.length(); c++) {
    int nIndexChild = arChild[c];
    String strChild =
nester.childOriginatorIdAt(nIndexChild);
    reportMessage("\n Child "+nIndexChild+
"+strChild );

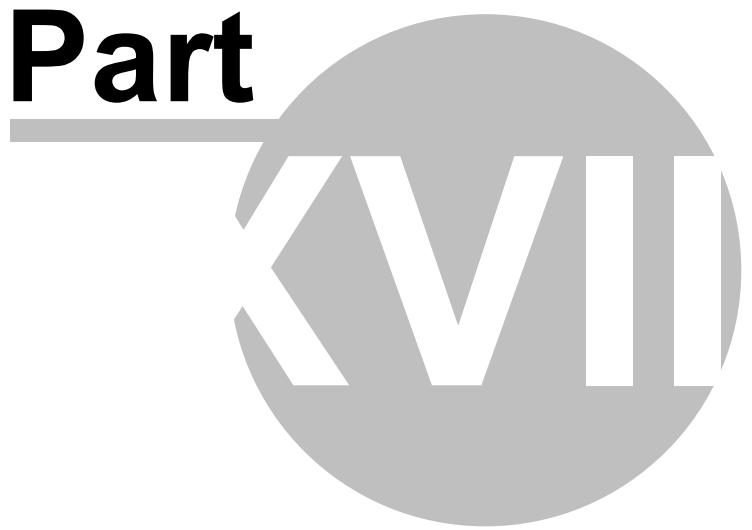
    if (pAccept==1) {
        Entity ent;
ent.setFromHandle(strChild);
        CoordSys cs =
arWorldXformIntoMaster[c];
        ent.transformBy(cs);
    }
}

}
}

```

*—end example]*

**Part**



## 18 dotNet

### 18.1 dotNet call

During insert, and custom events (see [context menu](#)), an external .Net function can be called from within the Tsl. The .Net dll has to be compiled for .Net1.1, working with Autocad 2004, 2005 and 2006. For Autocad series 2007, 2008 and 2009, it can be compiled for .Net1.1 or .Net2.0. For the Autocad series 2010 and 2011 it can be compiled with target .Net3.5.

Beware, the callDotNetFunction2 will only work from 2006 on and higher.

The following routine will call a specific routine of a .Net dll:

```
String[] callDotNetFunction1(String strDlIPath, String strClassName, String strFunction);
String[] callDotNetFunction1(String strDlIPath, String strClassName, String strFunction, String[] arArguments);

Map callDotNetFunction2(String strDlIPath, String strClassName, String strFunction);
Map callDotNetFunction2(String strDlIPath, String strClassName, String strFunction, Map mapArgument);
```

strDlIPath: the full path and name of the dll.  
 strClassName: the name of the class, including the namespace specifier  
 strFunction: the actual function name.  
 arArguments: an optional array of strings as list of arguments  
 mapArgument: an optional map argument

The callDotNetFunction1 function returns an array of strings, while the callDotNetFunction2 returns a Map value.

To make an assembly that contains a callable function you can start a new C# project of type "Class library".

Since hsbCAD2009+ build 14.1.43 and hsbCAD2010 build 15.0.18 the dll specified in the strDlIPath field is searched for automatically, if not fully specified. This means that if the dll does not contain a folder part, the [findFile](#) method is first applied on the dll. Then if found, the callDotNet method is launched.

Because Autocad determines the application context when loading the dll, the dll could be loaded into a 64 bit or a 32 bit context. For that reason you need to set your "Platform target" to "Any cpu". Also since hsbCAD is always loaded before your dll is being loaded, the referenced hsbCad dll's are found automatically from the <hsbCAD>\NetAc folder when loading. Therefore you should set the "local copy" of these referenced hsbCad dlls to false. In fact, if you set it to true, and the local copy is older than the one in NetAc, you might run into trouble.

*[Sample of TSL illustrating the callDotNetFunction1, with insert done in script:*

```
if (_bOnInsert) {
```

```

String strAssemblyPath = _kPathHsbInstall + "\\CustomApps\\
\TestDotNetFromTsl\\bin\\Release\\hsbTest.dll";
String strType = "hsbTestNS.hsbTestCL";
String strFunction = "testIT";

String strInAr[0];
strInAr.append("Input string 1");
strInAr.append("Input string 2");

String strOutAr[] = callDotNetFunction1(strAssemblyPath, strType,
strFunction, strInAr);

for (int i=0; i<strOutAr.length(); i++)
    reportMessage("\n"+i+": "+strOutAr[i]);

getPoint(); // be sure to request some input if you do not call
eraseInstance

eraseInstance();
return;
}

—end sample]

```

*[Sample of C# code that will compile into a dll that can work with the Tsl above. Make sure the project references the System.Windows.Forms.*

```

using System;
using System.Windows.Forms;

namespace hsbTestNS
{
    /// <summary>
    /// Summary description for hsbTestCL.
    /// </summary>
    public class hsbTestCL
    {
        public string[] testIT(string[] arIn)
        {
            string[] arOut = new string[3];
            for (int i=0; i < arIn.Length; i++)
            {
                MessageBox.Show(arIn[i]);
            }
            arOut[0] = "Hello";
            arOut[1] = "C Plus - Plus";
            arOut[2] = ";-)";

            return arOut;
        }
    }
}

```

---

—end sample]

---

[Sample of TSL illustrating the callDotNetFunction2, with insert done in script:

```
Unit (1, "mm");
if (_bOnInsert) {

    String strAssemblyPath = _kPathHsbInstall + "\CustomApps\
\TestDotNetFromTsl\bin\Release\hsbTest.dll";
    String strType = "hsbTestNS.hsbTestCL";
    String strFunction = "testMap";

    Map mapIn;
    mapIn.setInt("int", 3);
    mapIn.setDouble("dbl", 4, _kNoUnit);
    mapIn.setDouble("len", U(5));
    mapIn.setDouble("Area", U(5)*U(6), _kArea);
    mapIn.setDouble("Vol", U(5)*U(6)*U(1), _kVolume);
    mapIn.setString("key", "value");

    mapIn.setPoint3d("pt0", _Pt0);
    mapIn.setVector3d("vecXY", _XU+_YU);

    Map mapOut = callDotNetFunction2(strAssemblyPath, strType,
strFunction, mapIn);
    reportNotice("\nReturned map: "+mapOut);

    getPoint(); // be sure to request some input if you do not call
eraseInstance

    eraseInstance();
    return;
}
```

—end sample]

[Sample of C# code that will compile into a dll that can work with the Tsl above. Make sure the project references from the <hsbInstall>/NetAc folder the following dlls: hsbGeoLib.dll, hsbCollections.dll.

```
using System;
using System.Windows.Forms;
using hsbSoft.Collections;

namespace hsbTestNS
{
    /// <summary>
```

---

```
/// Summary description for hsbTestCL.  
/// </summary>  
public class hsbTestCL  
{  
    public bool testMap(HsbMap mapIn, out HsbMap mapOut)  
    {  
        mapOut = new HsbMap();  
        if (mapIn == null) return false;  
        MessageBox.Show(mapIn.ToString());  
        mapOut = mapIn;  
        return true;  
    }  
}  
}  
—end sample]
```

## 18.2 Managed API

In the distribution of hsbCAD you find a folder NetAC which contains the managed part of hsbCAD. In there you find also the assemblies that are part of our supported managed API. All methods in the namespace hsbSoft.Cad.API are supported.

In the following topics these methods are described:

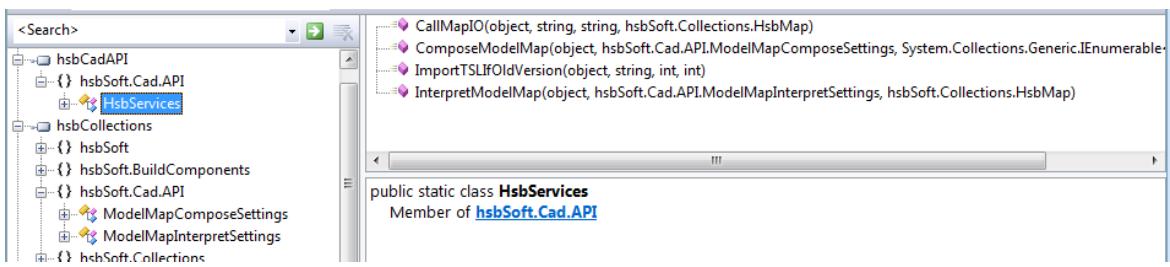
- [ModelXIO](#)
- [CallMapIO](#)
- [Modeless context](#)

The assembly hsbCadAPI uses other assemblies found in the NetAC folder. Also the methods and classes of this assembly will only work in an hsbCAD environment. This means that your assembly that references the hsbCadAPI dll will only work after loading it into Autocad.

Remember, to load a managed dll into Autocad you need to use the autocad command NETLOAD. If your assembly starts with hsb, then you can also use the HSB\_APPLOAD command.

The hsbCadAPI.dll (introduced version hsbCAD2011 build 16.0.3) is not dependent on the managed autocad assemblies (AcMgd.dll and AcDbMgd.dll). Object references are passed as either handle or IntPtr types. The database reference is passed as object.

The methods in the hsbCadAPI assembly are only portal methods. They do not work, unless the dll is loaded inside hsbCAD. hsbCAD will find the dlls automatically in its install folder structure. When your application references this assembly, as well as other assemblies from NetAC, it does NOT make sense to copy them to your application target folder. It is strongly advised to set the "Copy Local" state to False of the references to any NetAC dll.



### 18.2.1 ModelMapIO

The following C# application illustrates the use of the following methods: **ComposeModelMap** and **InterpretModelMap**. They are available in <hsbInstall>\NetAc\hsbCadAPI.dll (see [Managed API](#)) (in hsbCAD2011 build 16.0.3).

```
using hsbSoft.Collections;
using System;
using System.Collections.Generic;

namespace hsbSoft.Cad.API
{
    public static class HsbServices
    {

        public static HsbMap ComposeModelMap(object mgdObjAcadDatabase,
ModelMapComposeSettings composeSettings, IEnumerable<IntPtr> entities);
        public static IEnumerable<IntPtr> InterpretModelMap(object
mgdObjAcadDatabase, ModelMapInterpretSettings interpretSettings, HsbMap
inputMap);

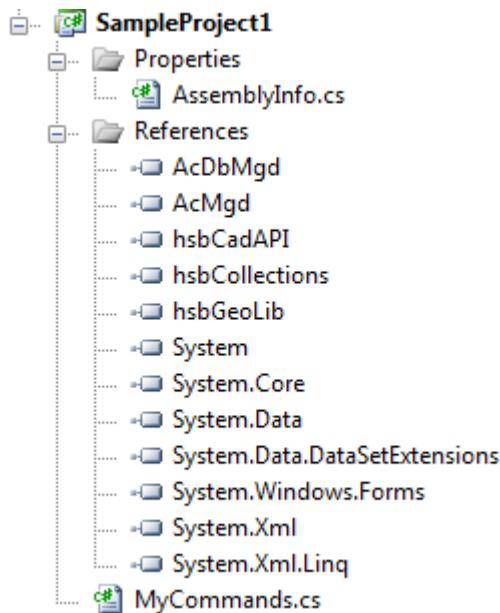
    }
}
```

The classes [ModelXComposeSettings](#) and [ModelXInterpretSettings](#) mimic the tsl classes which start with ModelX. They are available in the hsbCollections assembly.

*[Sample of C# class illustrating the use of ComposeModelMap and InterpretModelMap. The sample defines 2 commands inside autocad. The first one selects a number of entities, and exports these entities to a ModelX file. Then this ModelX file is opened by the DxExplorer. The second command imports the ModelX file into the autocad database.]*

*Below is the list of referenced assemblies. The autocad ones AcMgd and AcDbMgd can be referenced with local copy set to false. These are needed for the autocad references and are found in the Autocad install folder. The hsbCAD ones: hsbCadAPI, hsbCollections, and hsbGeoLib need to be set local copy false for the sample to work. They are found in the <hsbInstall>\NetAC folder.*

*The dll of this project can be loaded in autocad with the command NETLOAD. Having done that, there will be 2 commands available: myExportMM and myImportMM.*



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Reflection;
using System.IO;
using System.Diagnostics;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

using hsbSoft.Collections;
using hsbSoft.Cad.API;

namespace hsbSoft.Cad.API.Samples
{
    public class MyCommands
    {
        [CommandMethod("myExportMM", CommandFlags.UsePickSet)]
        static public void ExportMM()
        {
            // get autocad database and editor
            Document doc =
Autodesk
.AutoCAD
.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
            Editor ed = doc.Editor;
            Autodesk.AutoCAD.DatabaseServices.Database db =
doc.Database;
```

```

// query the user to select entities
PromptSelectionResult psr = ed.GetSelection();
if (psr.Status != PromptStatus.OK)
{
    ed.WriteMessage("\nCommand canceled");
    return;
}

// convert the selection set in to a list of IntPtr
var arEnts = new List<IntPtr>();
foreach (ObjectId idEnt in psr.Value.GetObjectIds())
{
    arEnts.Add(idEnt.OldIdPtr);
}

// call the ComposeModelMap with some settings, also pass in
the database
var flags = new ModelMapComposeSettings();
flags.AddCollectionDefinitions = true;
HsbMap mp = HsbServices.ComposeModelMap(db, flags, arEnts);
if (mp == null)
{
    ed.WriteMessage("\nModelMap could not be composed");
    return;
}

// ask the user to select a file location to save the file
var saveDialog = new SaveFileDialog();
saveDialog.Filter = "Model Map File (*.dxx)";
if (saveDialog.ShowDialog() != DialogResult.OK) { return; }

// output the composed modelmap as dxx file
mp.WriteToFile(saveDialog.FileName,
hsbSoft.Formatters.FormatterEngineType.Dxx);

// Locate and launch the DXExplorer app from the location of
the hsbCadAPI dll.
// For this to work, the hsbCadAPI should not be copied
locally.
var assembly =
Assembly.GetAssembly(typeof(hsbSoft.Cad.API.HsbServices));
var root = new
DirectoryInfo(assembly.Location).Parent.Parent.FullName;
var viewerPath = Path.Combine(root,
@"Utilities\DXxExplorer\DXExplorer.exe");
var processInfo = new ProcessStartInfo();
processInfo.Arguments = saveDialog.FileName;
processInfo.FileName = viewerPath;
Process.Start(processInfo);

}

[CommandMethod("myImportMM", CommandFlags.UsePickSet)]

```

```
    static public void ImportMM()
    {
        // get autocad database and editor
        Document doc =
Autodesk
.AutoCAD
.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
        Editor ed = doc.Editor;
        Autodesk.AutoCAD.DatabaseServices.Database db =
doc.Database;

        // ask user to select dxx file to be opened
        var openDialog = new OpenFileDialog();
        openDialog.CheckFileExists = true;
        openDialog.Filter = "Model Map File (*.dxx)";
        if (openDialog.ShowDialog() != DialogResult.OK) { return; }

        // the HsbMap constructor later on will throw an exception
        if the file does not exist.
        //if (!System.IO.File.Exists(openDialog.FileName))
        //{
        //    // check existence or catch exceptions later
        //    ed.WriteMessage("\nInvalid filename");
        //    return;
        //}

        // read the map specified by file
        var mp = new HsbMap(openDialog.FileName,
hsbSoft.Formatters.FormatterEngineType.Dxx);

        // call the InterpretModelMap with some settings also
        passing in the database
        var flags = new ModelMapInterpretSettings();
        flags.ResolveEntitiesByHandle = true;
        IEnumerable<IntPtr> arEnts =
HsbServices.InterpretModelMap(db, flags, mp);

        if (arEnts == null) { // something was wrong
            ed.WriteMessage("\nInvalid ModelMap");
            return;
        }

        // report the result of entities imported or modified
        ed.WriteMessage("\nSelected {0} entities.", arEnts.Count());
        foreach (IntPtr idInt in arEnts)
        {
            ObjectId id = new ObjectId(idInt);
            ed.WriteMessage("\nid handle: {0}",
id.Handle.ToString());
            }
            ed.WriteMessage("\n");
        }
    }
```

—end sample]

### 18.2.2 CallMapIO

The following C# application illustrates the use of the following methods: **ImportTSLIfOldVersion** and **CallMapIO**. They are available in <hsbInstall>\NetAc\hsbCadAPI.dll (see [Managed API](#)) (in hsbCAD2011 build 16.0.3).

```
using hsbSoft.Collections;
using System;
using System.Collections.Generic;

namespace hsbSoft.Cad.API
{
    public static class HsbServices
    {
        public static bool ImportTSLIfOldVersion(object
mgdObjAcadDatabase, string tslName, int majorVersionRequired, int
minorVersionRequired);
        public static bool CallMapIO(object mgdObjAcadDatabase, string
tslName, string executeKey, HsbMap map);
    }
}
```

The class HsbMap and related mimics the tsl class Map. They are available in the hsbCollections assembly.

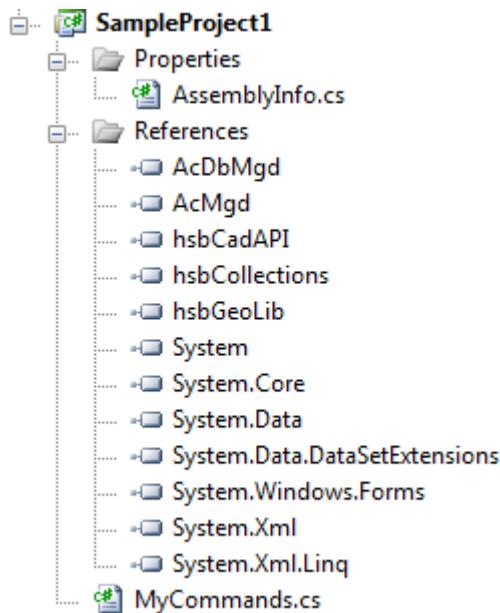
---

*[Sample of C# class illustrating the use of **ImportTSLIfOldVersion** and **CallMapIO**. The sample defines a command inside autocad which changes the color of an entity selected in the managed part and passed to the Tsl. The second CallMapIO calls another method in the same Tsl that returns a value from Tsl to managed.]*

*Below is the list of referenced assemblies. The autocad ones AcMgd and AcDbMgd can be referenced with local copy set to false. These are needed for the autocad references and are found in the Autocad install folder. The hsbCAD ones: hsbCadAPI, hsbCollections, and hsbGeoLib should also be set local copy false. They are found in the <hsbInstall>\NetAC folder.*

*Remark: for Autocad 2013 list of products you need also the AcCoreMgd.dll.*

*The dll of this project can be loaded in autocad with the command NETLOAD. Having done that, the following command becomes available: myChangeColor.*



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Reflection;
using System.IO;
using System.Diagnostics;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

using hsbSoft.Collections;
using hsbSoft.Cad.API;

namespace hsbSoft.Cad.API.Samples
{
    public class MyCommands
    {
        [CommandMethod("myChangeColor", CommandFlags.UsePickSet)]
        static public void ChangeColor()
        {
            // get active database
            Document doc =
Autodesk
.AutoCAD
.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
            Editor ed = doc.Editor;
            Autodesk.AutoCAD.DatabaseServices.Database db =
doc.Database;
```

```

// let user select an entity
PromptSelectionResult psr = ed.GetSelection();
if (psr.Status != PromptStatus.OK || psr.Value.Count==0)
{
    ed.WriteMessage("\nCommand canceled");
    return;
}
ObjectId idEnt = psr.Value[0].ObjectId;

HsbMap mapArg = new HsbMap(MapType.List);
mapArg.Add("dVal", 10, hsbSoft.Collections.UnitType.Length);
mapArg.Add("dVal2", 11,
hsbSoft.Collections.UnitType.Length);
    mapArg.Add("Entity", idEnt.Handle.ToString(), true); // pass
entity in map

string strTslName = "mapIOWorker";

// Load the tsl called "mapIOWorker" if the version in the
dwg is lower then 1.0.
if (!HsbServices.ImportTSLIfOldVersion(db, strTslName, 1,
0))
{
    ed.WriteMessage("\nCurrent Tsl too old, or not found,
and new one could not be loaded.");
    return; // the tsl named mapIOWorker was not found
}

// Call the Tsl with name "mapIOWorker" with execute key
"changeColor".
HsbServices.CallMapIO(db, strTslName, "changeColor",
mapArg);

// Call the Tsl with name "mapIOWorker" with execute key
"sumValues".
// The CallMapIO will return true if the tsl was executed
fine, and the tsl did not return eraseInstance.
bool bOk = HsbServices.CallMapIO(db, strTslName,
"sumValues", mapArg);

// report the result
string strRet = "<not set>";
if (bOk)
{
    // mapArg is not valid if CallMapIO returned false
    strRet = mapArg.Get("dValNew", 0.0).ToString();
}
MessageBox.Show(strTslName + " returned: " + bOk + " with
value " + strRet);
}
}

```

—end sample]

[Example O-type TSL that illustrates the called Tsl in the example above. The Tsl should be available with name "mapIOWorker" and version 1.0.

```

if ( _kExecuteKey=="changeColor" ) {

    String strKey = "Entity"; // key in _Map
    if (_Map.hasEntity(strKey)) {
        Entity ent = _Map.getEntity(strKey);
        ent.setColor(1);
        reportNotice("\nColor changed!");
    }
    else {
        reportNotice("\n_Map does not contain key: "+strKey);
    }

    return;
}

if ( _kExecuteKey=="sumValues" ) {

    String strKey = "dVal"; // key in _Map
    if (_Map.hasDouble(strKey)) {
        reportNotice("\n_Map contains "+strKey);
    }
    else {
        reportNotice("\n_Map does not contain key: "+strKey);
    }

    // interpret the arguments
    double dVal = _Map.getDouble("dVal");
    double dVal2 = _Map.getDouble("dVal2");
    _Map = Map(); // reset map contents

    // do the work
    double dValNew = dVal+dVal2;

    // return the results
    _Map.setDouble("dValNew", dValNew);

    // do not call eraseInstance(), otherwise no _Map changes are
    accepted
    return;
}

```

—end example]

### 18.2.3 Modeless context

In normal command context the document is locked, and a command-will-start event as well as a command-ended event is fired. Modeless document access does not have this, so the implementation of the modeless action must take care of this. To lock the document for changes, one should call the **LockDocument** method on Document, and Dispose the DockLock at the end.

If during the **CallMapIO** method or **InterpretModelX** entities are changed, some of the changes are postponed until the onCommandEnded event. If these calls happen during a modeless context the caller is responsible for notifying the changes can be done. For this purpose we have the **processRecalcEntitiesNow** method. It is available in <hsbInstall>\NetAc\hsbCadAPI.dll (see [Managed API](#)) (in hsbCAD2011 build 16.0.3). The following C# application illustrates this.

The following type of operations are (sometimes partly) executed during processRecalcEntitiesNow:

- genbeams of which the layer changed are recalculated (recalculation of genbeam means tools are updated from etools and solid is recalculated from tools).
- trigger tsI's and etools for recalculation in case of modification (through MM import, or transformBy)
- the recalc of the genbeams in case etools are modified
- automatic transformation of etools of which all the beams are in the transform operation
- viewport orientation is updated from requested data
- wall element arrow location is corrected in certain events
- roof section plates and windows are updated from roof section changes
- roof symbol update
- solids of modified beams are recalculated
- ...

The command name argument, strCmd is only used for error reporting and debugging. Only for some special commands like undo and redo it is important.

```
using hsbSoft.Collections;
using System;
using System.Collections.Generic;

namespace hsbSoft.Cad.API
{
    public static class HsbServices
    {
        public static void processRecalcEntitiesNow(string strCmd);
    }
}
```

---

*[Sample of C# class illustrating the use of **processRecalcEntitiesNow** and **CallMapIO**. The sample defines a command inside autocad which changes the color of an entity selected in the managed part and passed to the Tsl. For other info regarding references see the samples in [CallMapIO](#) and [ModelXIO](#). The dll of this project can be loaded in autocad with the command NETLOAD. Having done that, the following command becomes available: myChangeColor2.]*

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Reflection;
using System.IO;
using System.Diagnostics;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

using hsbSoft.Collections;
using hsbSoft.Cad.API;

namespace hsbSoft.Cad.API.Samples
{
    public class MyCommands
    {
        // set command flags to simulate modeless context:
        NoInternalLock and Transparent
        [CommandMethod("myChangeColor2", CommandFlags.UsePickSet |
        CommandFlags.NoInternalLock | CommandFlags.Transparent)]
        static public void ChangeColor2()
        {
            // get active database
            Document doc =
Autodesk
.AutoCAD
.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
            Editor ed = doc.Editor;
            Autodesk.AutoCAD.DatabaseServices.Database db =
doc.Database;

            // let user select an entity
            PromptSelectionResult psr = ed.GetSelection();
            if (psr.Status != PromptStatus.OK || psr.Value.Count==0)
            {
                ed.WriteMessage("\nCommand canceled");
                return;
            }
            ObjectId idEnt = psr.Value[0].ObjectId;

            HsbMap mapArg = new HsbMap(MapType.List);
            mapArg.Add("Entity", idEnt.Handle.ToString(), true); // pass
entity in map

            string strTslName = "mapIOWorker";

            // Modeless commands need to set the doclock.
            // Disposal of the lock will unlock the document at the end
            // of using block.
            DocumentLock loc = doc.LockDocument();
```

```

        using (loc)
        {
            // Load the tsl called "mapIOWorker" if the version in
            the dwg is lower than 1.0.
            if (!HsbServices.ImportTSLIfOldVersion(db, strTslName,
1, 0))
            {
                ed.WriteMessage("\nCurrent Tsl too old, or not
found, and new one could not be loaded.");
                return; // the tsl named mapIOWorker was not found
            }

            // Call the Tsl with name "mapIOWorker" with execute key
            "changeColor".
            HsbServices.CallMapIO(db, strTslName, "changeColor",
            mapArg);

            // Some parts of the tsl execution might be postponed
            until onCommandEnded.
            // For modeless commands, this event will never happen.
            So it is important to
            // trigger this cleanup by calling
            processRecalcEntitiesNow.
            HsbServices.processRecalcEntitiesNow("myChangeColor2");
        }

    }
}

—end sample]

```

#### 18.2.4 Extrusion Profiles

The following C# application illustrates the use of the following method:

**AssureExtrusionProfileIsPresent**. They are available in <hsbInstall>\NetAc\hsbCadAPI.dll (see [Managed API](#)) (in hsbCAD2011 build 16.0.102).

```

using hsbSoft.Collections;
using System;
using System.Collections.Generic;

namespace hsbSoft.Cad.API
{
    public static class HsbServices
    {
        public static bool AssureExtrusionProfileIsPresent(object
mgdObjAcadDatabase, string strProfileName);
    }
}

```

The method **AssureExtrusionProfileIsPresent** verifies if the extrusion profile with name strProfileName is already present in the drawing. If it is, it returns true. If it is not, the method does a

search through the dwg's in the <company>/Profile folder to search for the given extrusion profile definition inside the dwgs with the given name. If it is found, the profile is loaded into the current dwg, and true is returned. If the profile is not found in that location, the <content>/\*!/Profile folders are searched. If it is able to load the profile, true is returned. If the profile is not loaded, false is returned.

---

*[Sample of C# class illustrating the use of **AssureExtrusionProfileIsPresent**. The sample defines a command inside autocad which loads two extrusion profiles. To setup a C# project to run the sample, please look at the previous samples.*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Reflection;
using System.IO;
using System.Diagnostics;

using Autodesk.AutoCAD.Runtime;
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.EditorInput;

using hsbSoft.Collections;
using hsbSoft.Cad.API;

namespace hsbSoft.Cad.API.Samples
{
    public class MyCommands
    {
        [CommandMethod("myAssureExtrProfilePresent",
CommandFlags.UsePickSet)]
        static public void AssureExtrProfilePresent()
        {
            // get autocad database and editor
            Document doc =
Autodesk
.AutoCAD
.ApplicationServices.Application.DocumentManager.MdiActiveDocument;
            Editor ed = doc.Editor;
            Database db = doc.Database;

            string strName = "xPly";
            bool bOk = HsbServices.AssureExtrusionProfileIsPresent(db,
strName);
            ed.WriteMessage("\nPresent {0}: {1}.", strName, bOk);

            strName = "3Ply";
        }
    }
}
```

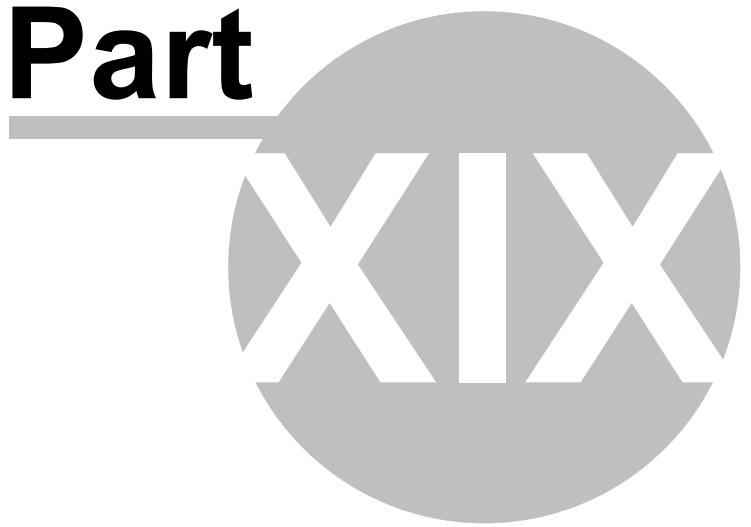
```
bOk = HsbServices.AssureExtrusionProfileIsPresent(db,
strName);
    ed.WriteMessage("\nPresent {0}: {1}.", strName, bOk);

}
}

}

—end sample]
```

**Part**



## 19 WebKit

### 19.1 WebKit

From hsbCAD v20 on, there is a javascript WebKit interface. WebKit is a runtime engine based on NodeJS and Chrome, that allows to run html with javascript. The WebKit supports html to appear as normal dialog.

The exposure from Tsl is similar to [callDotNet](#). Tsl can send to the WekKit application a [Map](#). The Map is automatically converted to JSON. The WebKit can return a map, which will be automatically converted back as well.

```
int callWebKit(String strFolderLocation, Map mapArgument, Map& mapRet);
```

---

*[Sample of TSL illustrating the callWebKit, with insert done in script:*

```
Map mpIn;
mpIn.readFromDxxFile("C:\\Hsb-Project\\Tsl\\WebKitMapInput.dxx");

String strFolder = _kPathHsbInstall + "\\Utilities\\WKMapEditor";

Map mpOut;
int nRetCode = callWebKit(strFolder, mpIn, mpOut);
reportMessage("\nTsl::callWebKit from " + strFolder + " returned: " +
nRetCode);

mpOut.writeToDxxFile("C:\\Hsb-Project\\Tsl\\WebKitMapOutput.dxx");
—end sample]
```

*[Sample of TSL illustrating the callWebKit, with insert done in script:*

```
if (_bOnInsert) {
    PrEntity ssE(T("|Select entities to be exported|"));
    if (!ssE.go()) {
        eraseInstance();
        return;
    }
    Entity entsOut[] = ssE.set();
    reportMessage (T("\n|Number of entities selected:| ") +
entsOut.length());

    // set some export flags
    ModelXComposeSettings mmFlagsOut;
    //mmFlagsOut.addSolidInfo(TRUE); // default FALSE
```

```

//mmFlagsOut.addAnalysedToolInfo(TRUE); // default FALSE
//mmFlagsOut.addElemToolInfo(TRUE); // default FALSE
//mmFlagsOut.addConstructionToolInfo(TRUE); // default FALSE
//mmFlagsOut.addHardwareInfo(TRUE); // default FALSE
//mmFlagsOut.addRoofplanesAboveWallsAndRoofSectionsForRoofs(TRUE);
// default FALSE
//mmFlagsOut.addCollectionDefinitions(TRUE); // default FALSE

// compose ModelX
ModelX mmOut;
mmOut.setEntities(entsOut);
mmOut.dbComposeMap(mmFlagsOut);

// write ModelX to dxx file
Map mapIn= mmOut.map();

String strFolder = _kPathHsbInstall + "\Utilities\WKMapEditor";

Map mapOut;
int nRetCode = callWebKit(strFolder, mapIn, mapOut);
//reportMessage("\nTsl::callWebKit from " + strFolder + "
returned: " + nRetCode);

if (nRetCode != 0) {return;}

ModelX mmIn;
mmIn.setMap(mapOut);

// set some import flags
ModelXInterpretSettings mmFlagsIn;
mmFlagsIn.resolveEntitiesByHandle(TRUE); // default FALSE
mmFlagsIn.resolveElementsByNumber(TRUE); // default FALSE
//mmFlags.setBeamTypeNameAndColorFromHsbId(TRUE); // default FALSE

// interpret ModelX
mmIn.dbInterpretMap(mmFlagsIn);

// report the entities imported/updated/modified
Entity entsIn[] = mmIn.entity();
reportMessage (T("\n|Number of entities imported:| ") +
entsIn.length());

eraseInstance();
return;
}

—end sample]

```

**Part**

**XXX**

## 20 Samples

### 20.1 Example of Slot and Mortise

```
Point3d pp = _L0.intersect(_Plf,-30);

PropDouble len(1,400,"Length");
PropDouble diam(2,40,"Diameter");
PropDouble depth(3,40,"Depth");
PropDouble T1(4,6,"T1");

 $_Y0.visualize(pp,3);$ 
//  $_X0.visualize(pp,3);$ 
 $_Z0.visualize(pp,3);$ 
//  $_Yf.visualize(_Pt0,4);$ 
//  $_Zf.visualize(_Pt0,4);$ 
 $_Z1.visualize(_Pt0,4);$ 

// Cut firstcut(pp,_Z1);
// Beam0.addTool(firstcut,1);

Mortise ms(pp,_Yf,_Zf,-_Z1, diam, _H0-20,depth, 1.3,0,1,1) ;
BeamCut bms(pp,_Yf,_Zf,-_Z1, diam, _H0-20,depth,-1.3,0,1) ;

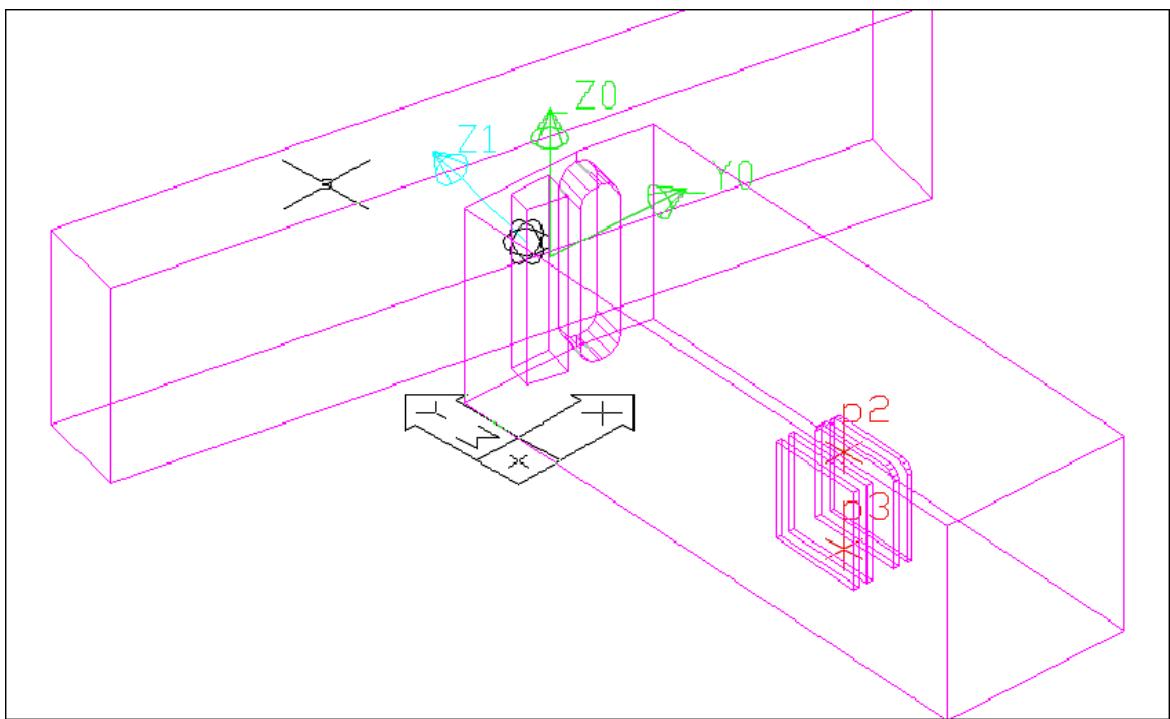
 $_Beam0.addTool(ms,1);$ 
 $_Beam0.addTool(bms);$ 

Point3d p2 = _L0.intersect(_Plf,-len);
Point3d p3 = p2 -  $_Z0*0.5*_H0;$ 

p3.visualize(10);
Slot sl2(p3,_X0,_Y0,_Z0, 100,T1,0.5*_H0,0, 5,1) ;
BeamCut bs2(p3,_X0,_Y0,_Z0, 100,T1,0.5*_H0,0,-5,1) ;

p2.visualize(10);
Slot sl1(p2,_X0,_Y0,_Z0, 100,T1,_H0,0, 10,-1) ;
BeamCut bs1(p2,_X0,_Y0,_Z0, 100,T1,_H0,0,-10,-1) ;

 $_Beam0.addTool(sl1);$ 
 $_Beam0.addTool(sl2);$ 
 $_Beam0.addTool(bs2);$ 
 $_Beam0.addTool(bs1);$ 
```



## 20.2 Examples Cut, Beam cut and Drills

Example 1:

```
Cut firstcut(_Pt0-30*_Z1,_Z1);
_Beam0.addTool(firstcut,1);
```

```
PropDouble a(1,120);
PropDouble b(2,80);
```

```
Point3d ptm1, ptm2, ptm3, ptm4;
ptm1 = _Ptc + _X1*a + _Y1*b;
ptm2 = _Ptc - _X1*a + _Y1*b;
ptm3 = _Ptc - _X1*a - _Y1*b;
ptm4 = _Ptc + _X1*a - _Y1*b;
```

```
Point3d ptn1, ptn2, ptn3, ptn4;
ptn1 = ptm1 + _Z1*_W1;
ptn2 = ptm2 + _Z1*_W1;
ptn3 = ptm3 + _Z1*_W1;
ptn4 = ptm4 + _Z1*_W1;
```

```
PropDouble diam(3,20);

Drill dr1(ptm1,ptn1,diam);
Drill dr2(ptm2,ptn2,diam);
Drill dr3(ptm3,ptn3,diam);
Drill dr4(ptm4,ptn4,diam);

_Beam1.addTool(dr1);
_Beam1.addTool(dr2);
_Beam1.addTool(dr3);
_Beam1.addTool(dr4);

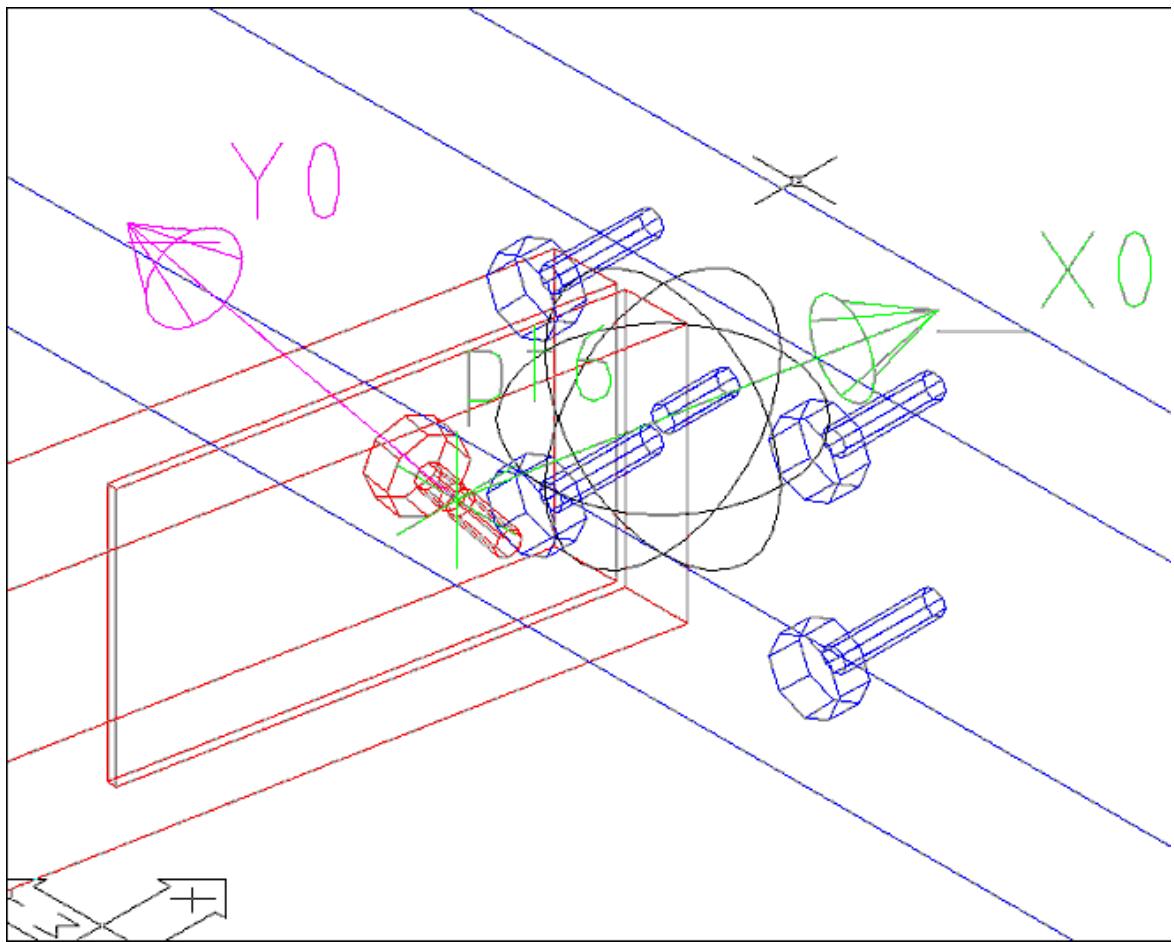
Point3d ptn5;
ptn5=_Pt0+_W1/2*_Z1;
Drill dr5(_Pt0,ptn5,diam);
_Beam1.addTool(dr5);

Point3d pt6,pt7,pt8;
pt6=_Pt0-_X0*150;
pt6.visualize(5);
_Y0.visualize(pt6,6);
pt7=pt6+_Y0*_W0*0.5;
pt8=pt6-_Y0*_W0*0.5;
Drill dr6(pt7,pt8,diam);
_Beam0.addTool(dr6);
PropDouble diasink(4,40);

Point3d pt9;
pt9=pt7-_Y0*25;
Drill dr7(pt7,pt9,diasink);
_Beam0.addTool(dr7);

Point3d pt10,pt11,pt12,pt13;
pt10=ptm1+_Z1*25;
pt11=ptm2+_Z1*25;
pt12=ptm3+_Z1*25;
pt13=ptm4+_Z1*25;
Drill dr8(ptm1,pt10,diasink);
Drill dr9(ptm2,pt11,diasink);
Drill dr10(ptm3,pt12,diasink);
Drill dr11(ptm4,pt13,diasink);
_Beam1.addTool(dr8);
_Beam1.addTool(dr9);
_Beam1.addTool(dr10);
_Beam1.addTool(dr11);

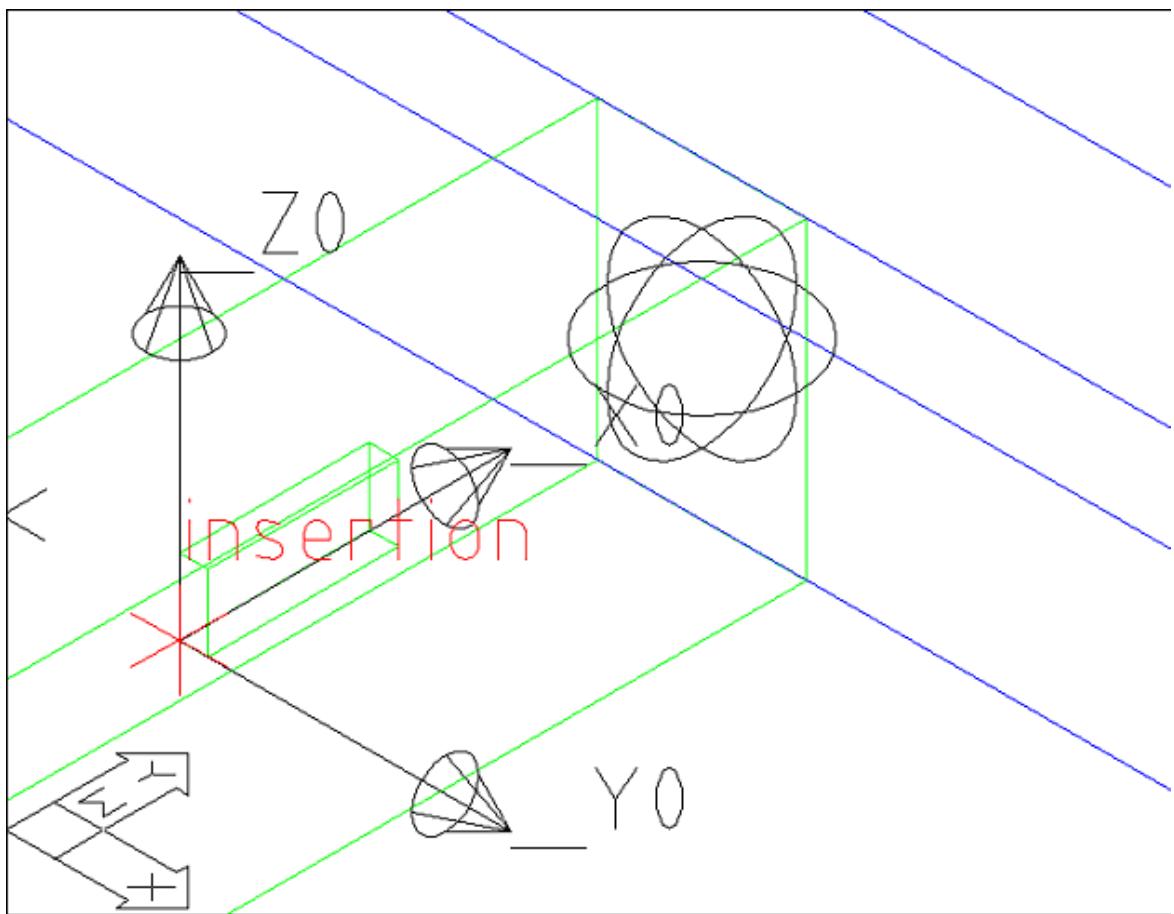
BeamCut b1(pt6,_X0,_Y0,_Z0, 500,8,_H1) ;
_Beam0.addTool(b1);
pt6.visualize(3);
_X0.visualize(pt6,3);
```

Example 2:

```
Cut firstcut(_Pt0,_Z1);
_Beam0.addTool(firstcut,1);
```

```
Point3d insertion;
insertion = _Pt0 - _X0*550;
_X0.visualize(insertion,7);
_Y0.visualize(insertion,7);
_Z0.visualize(insertion,7);
insertion.visualize(1);
```

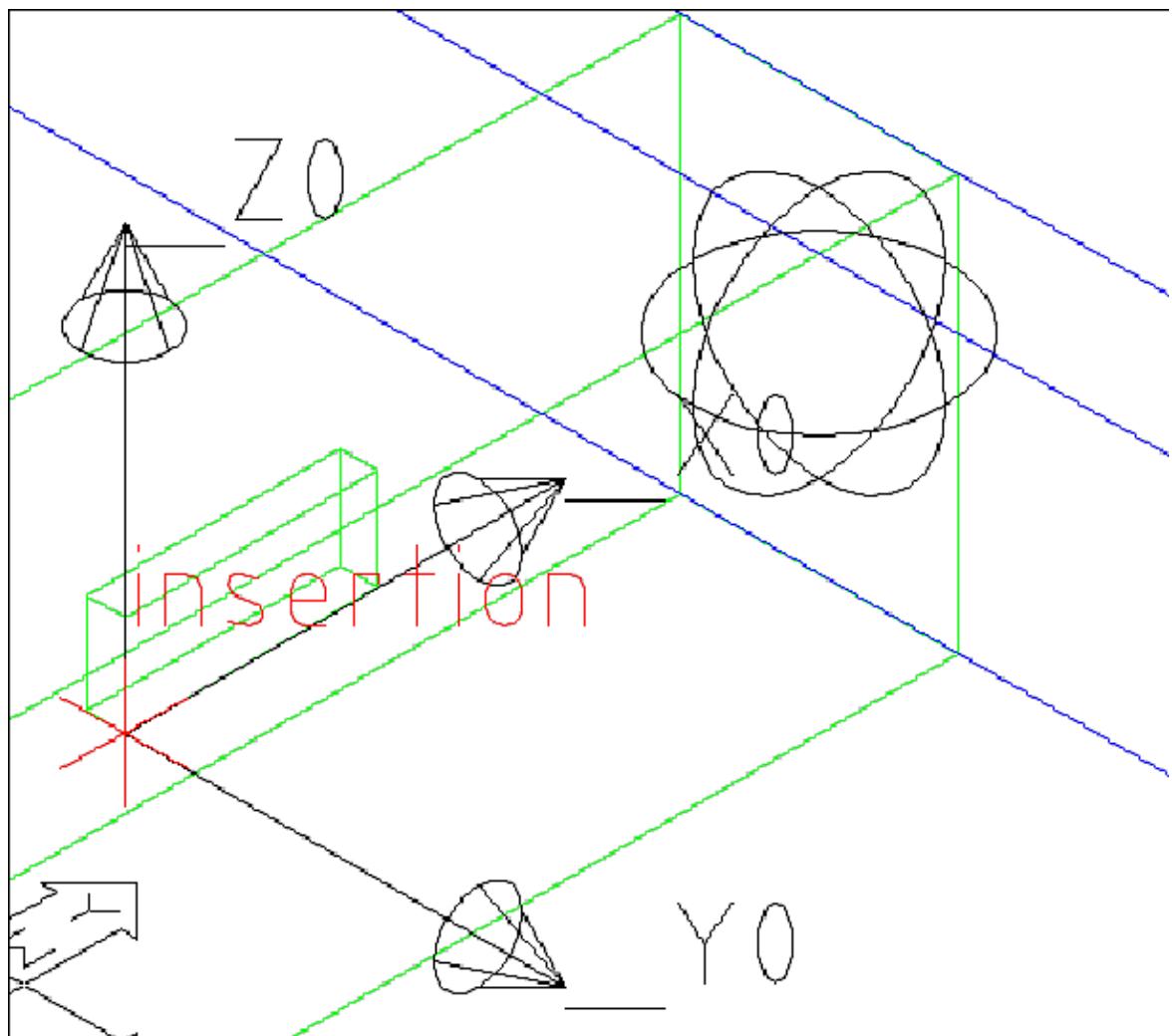
```
BeamCut bc(insertion,_X0,_Y0,_Z0,200,30,80,1,1,1);
_Beam0.addTool(bc);
```

Example 2:

```
Cut firstcut(_Pt0,_Z1);
_beam0.addTool(firstcut,1);

Point3d insertion;
insertion = _Pt0 - _X0*550;
_X0.visualize(insertion,7);
_Y0.visualize(insertion,7);
_Z0.visualize(insertion,7);
insertion.visualize(1);

BeamCut bc(insertion,_X0,_Y0,_Z0,200,30,80,1,-1,1);
_beam0.addTool(bc);
```

Example 2b:

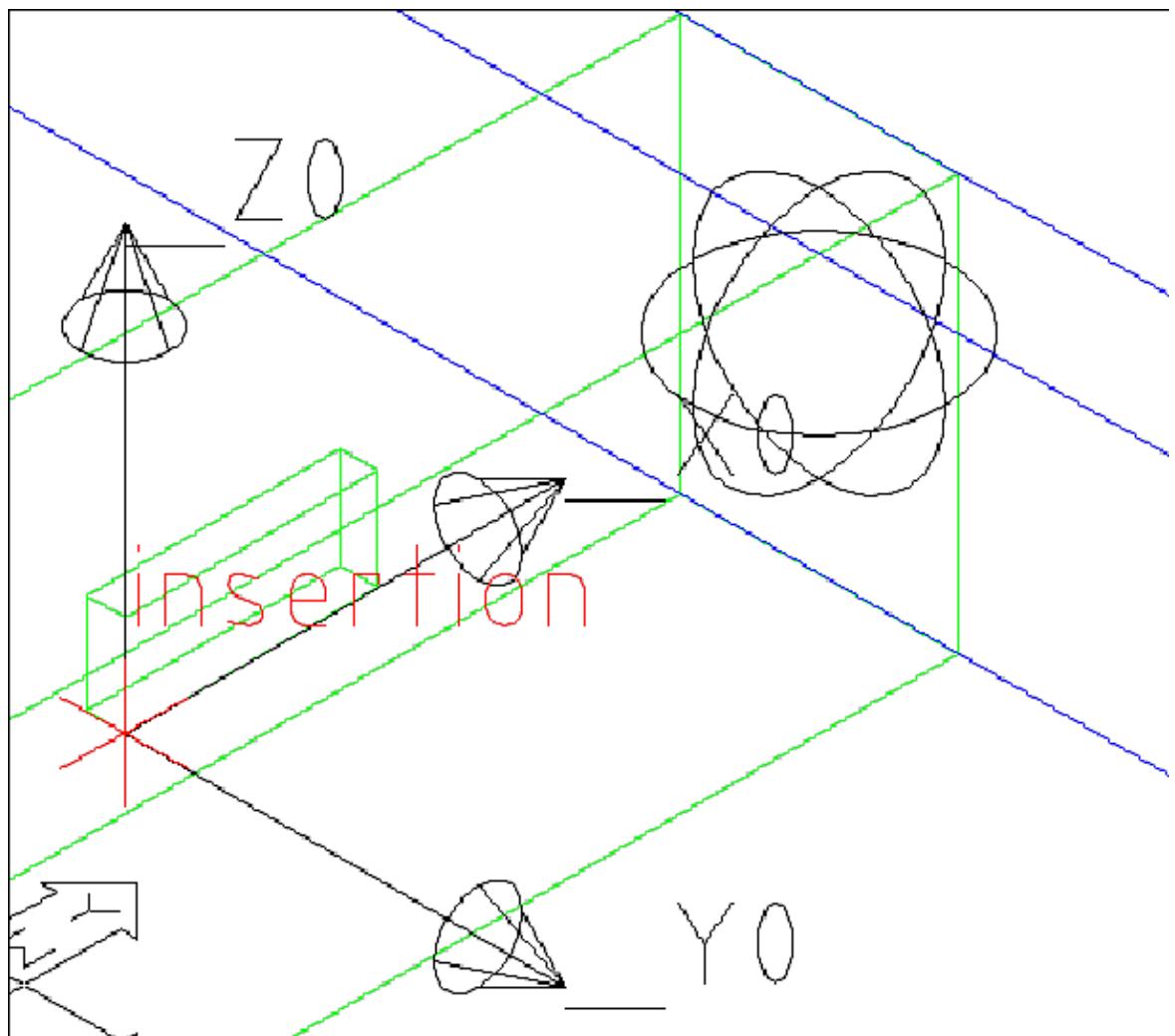
```

Cut firstcut(_Pt0,_Z1);
_Beam0.addTool(firstcut,1);

Point3d insertion;
insertion = _Pt0 - _X0*550;
_X0.visualize(insertion,7);
_Y0.visualize(insertion,7);
_Z0.visualize(insertion,7);
insertion.visualize(1);

BeamCut bc(insertion,_X0,_Y0,_Z0,200,30,80,1,-1,1);
_Beam0.addTool(bc);

```



### Example 3

```

PropDouble offset(2,200,"Cutplane offset");
Point3d pp1 = _Pt0-offset*_Z1;
Cut firstcut(pp1,_Z1);
_Beam0.addTool(firstcut,1);

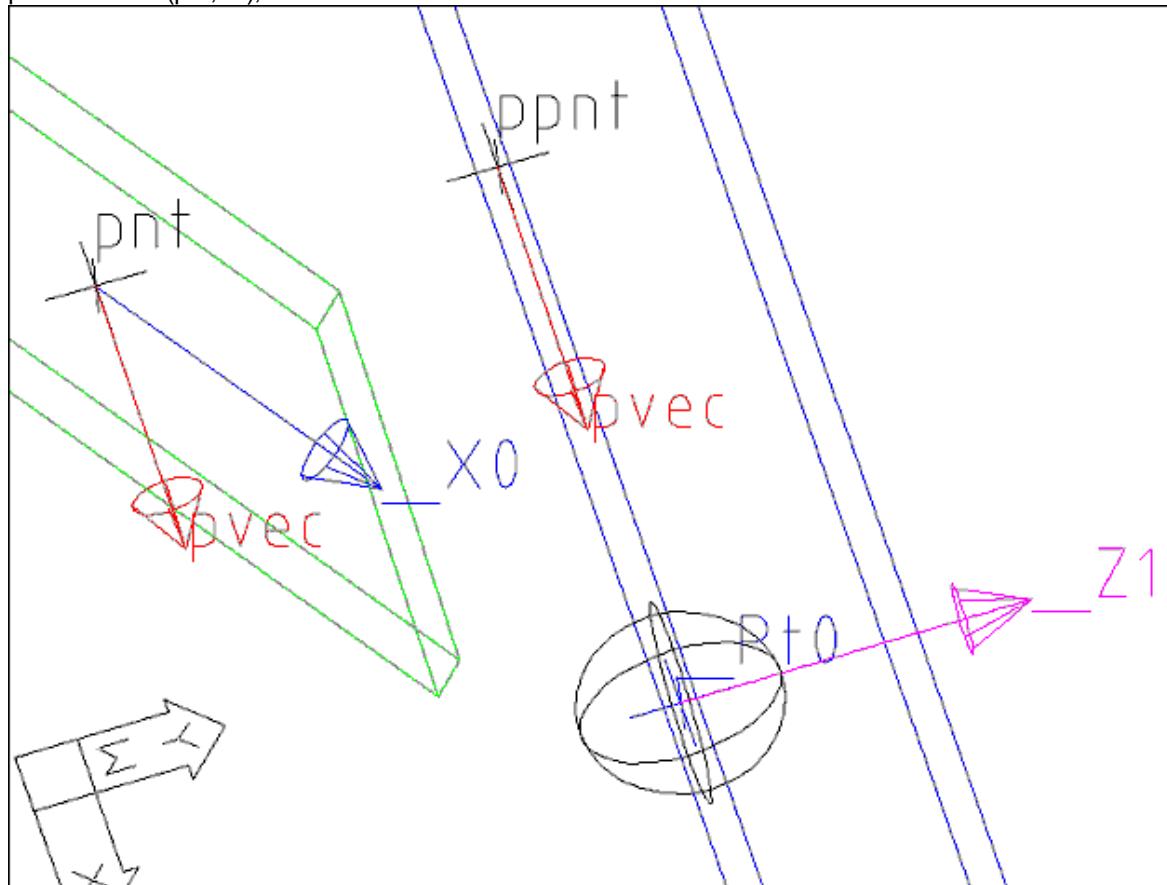
_Pt0.visualize(5);
_Z1.visualize(_Pt0,6);

Line ln(_Pt0,_X0);
Point3d pnt;
pnt = ln.intersect(_Plf, -2*offset);
pnt.visualize(7);
_X0.visualize(pnt,9);

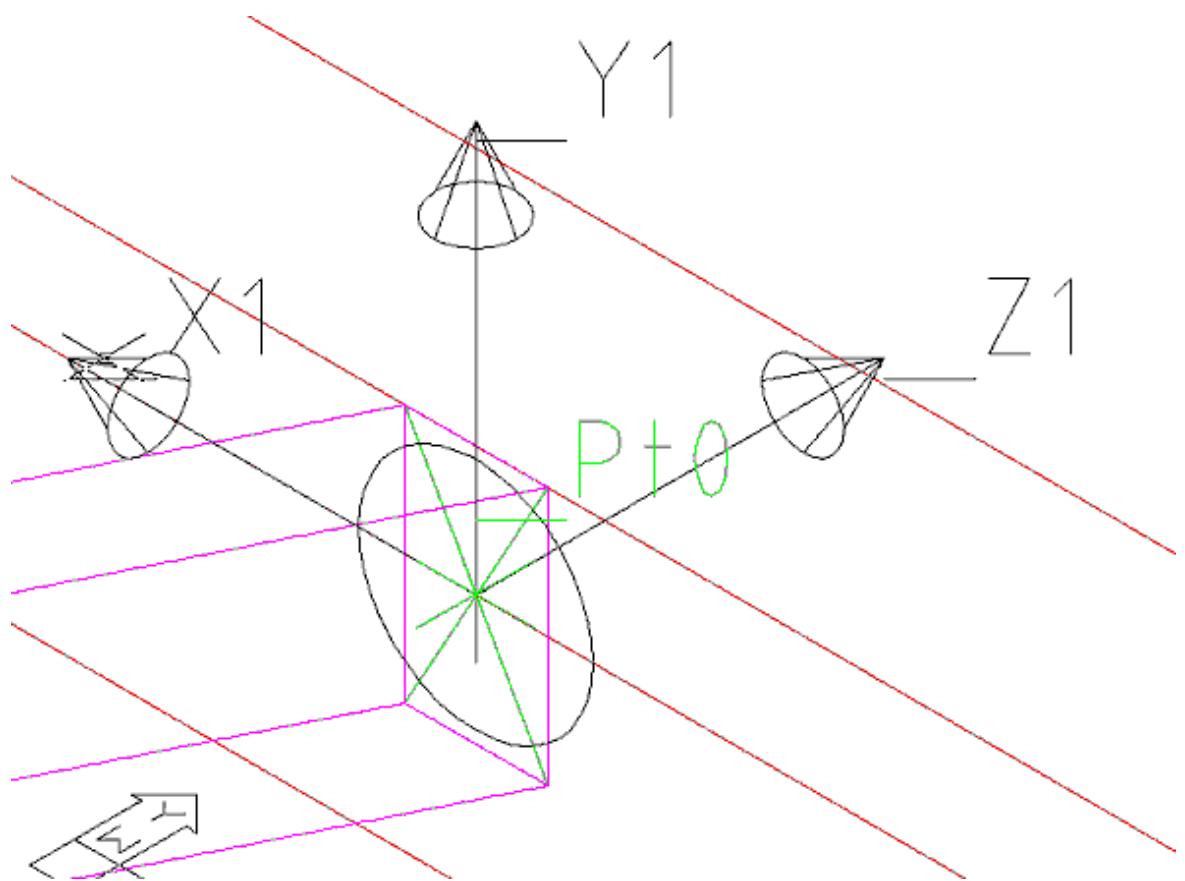
Point3d ppnt;
ppnt = pnt.projectPoint(_Plf,0);
ppnt.visualize(8);

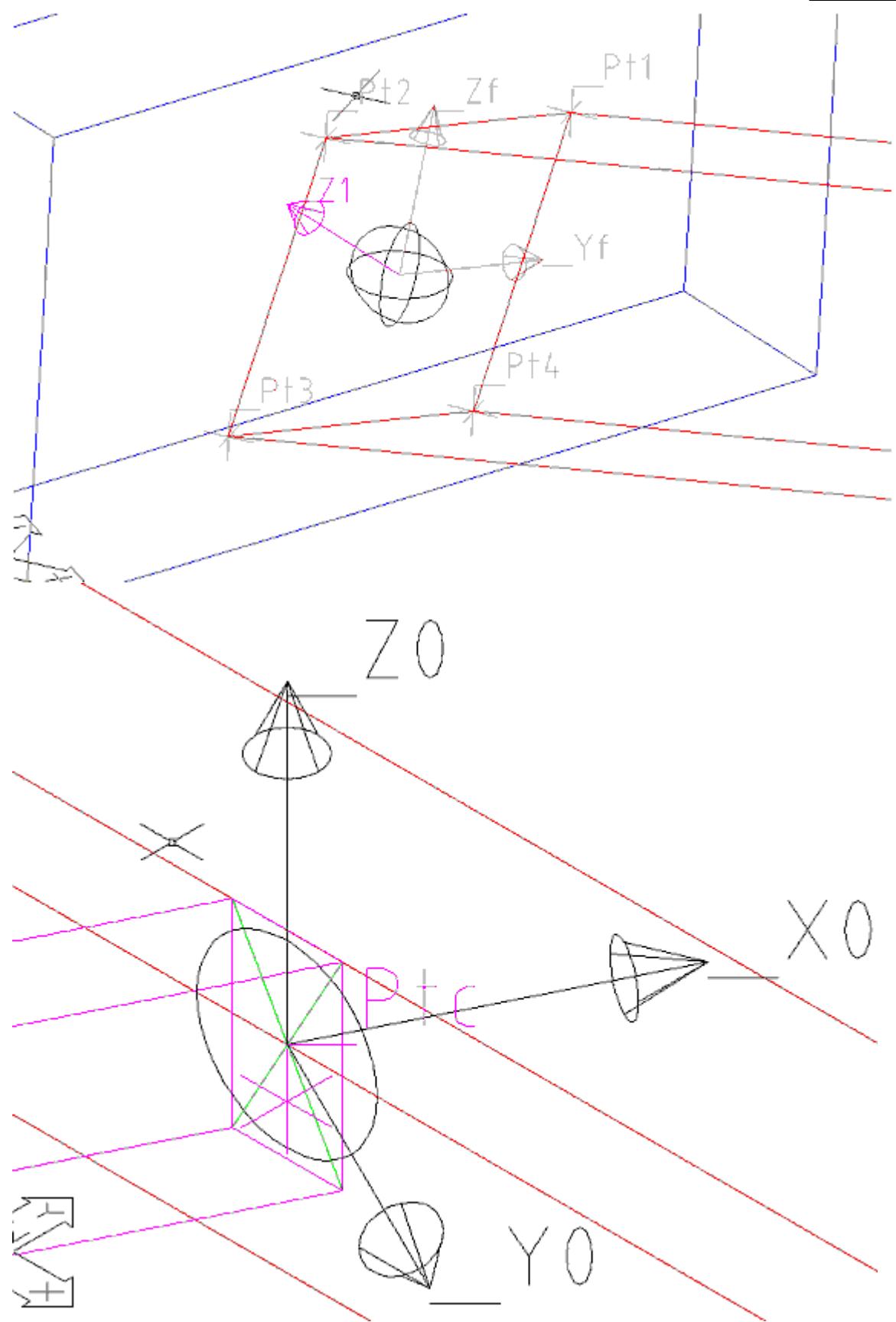
```

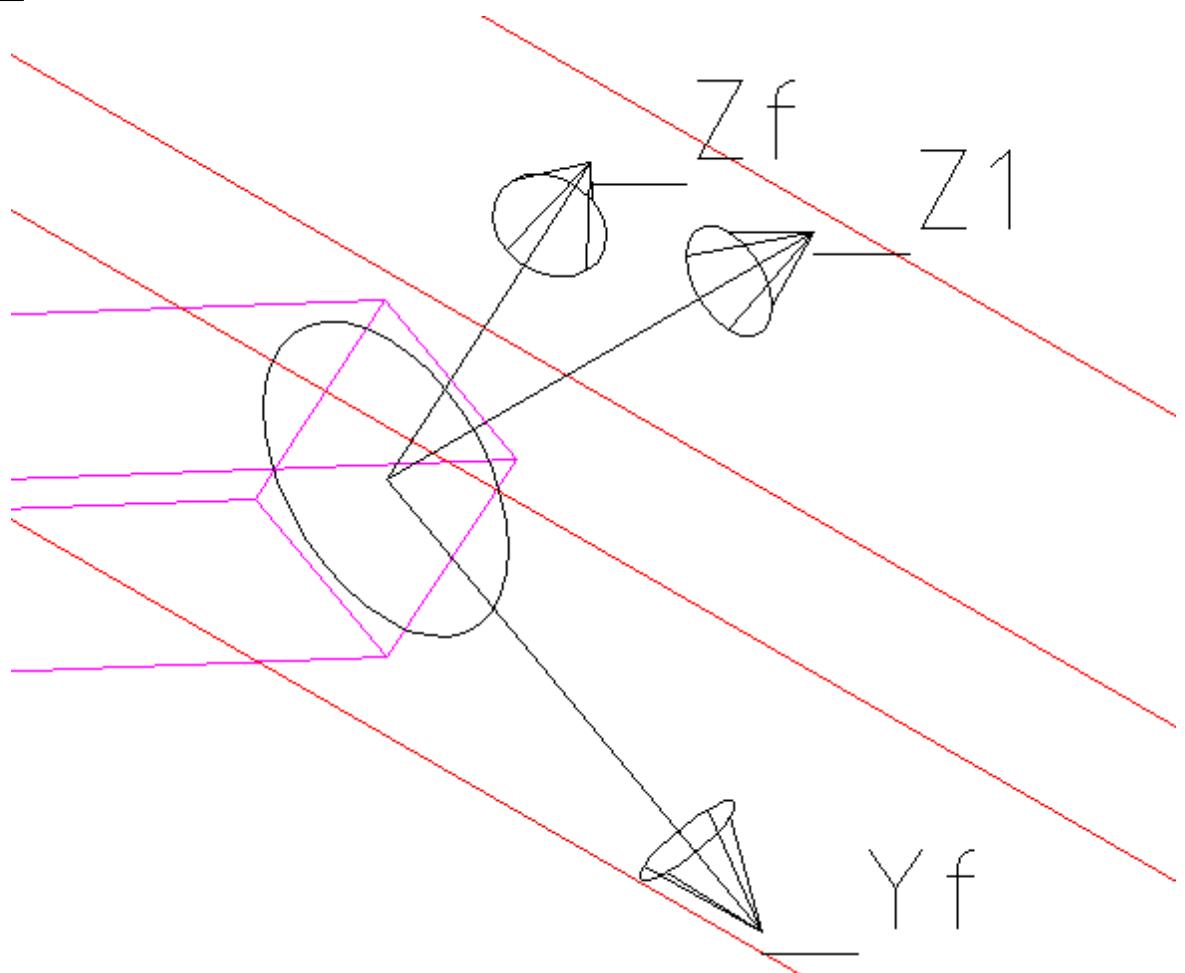
```
Vector3d pvec;  
pvec = _X0.projectVector(_Plf);  
pvec.visualize(ppnt,10);  
pvec.visualize(pnt,10);
```

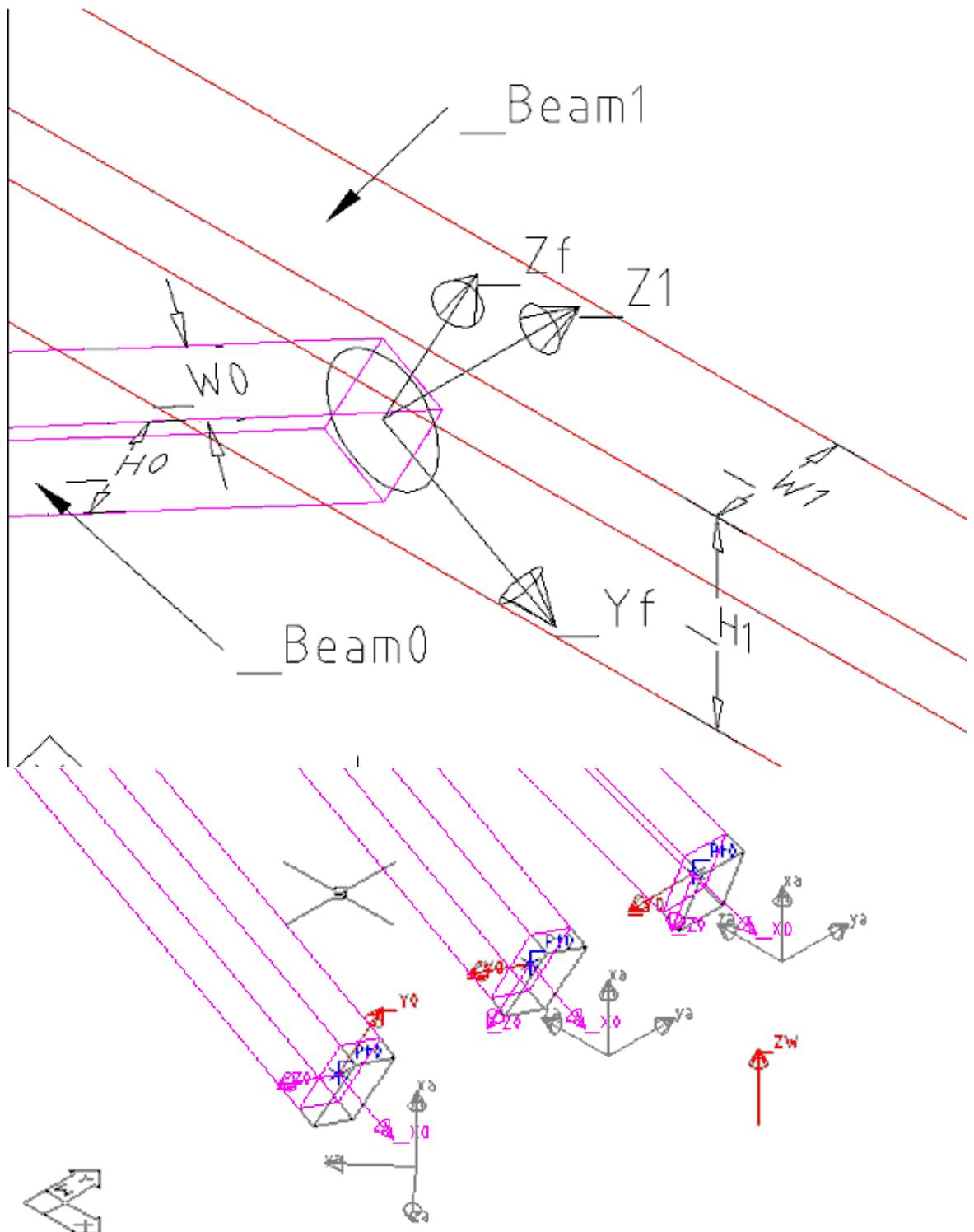


## 20.3 Example Manipulation of Vectors







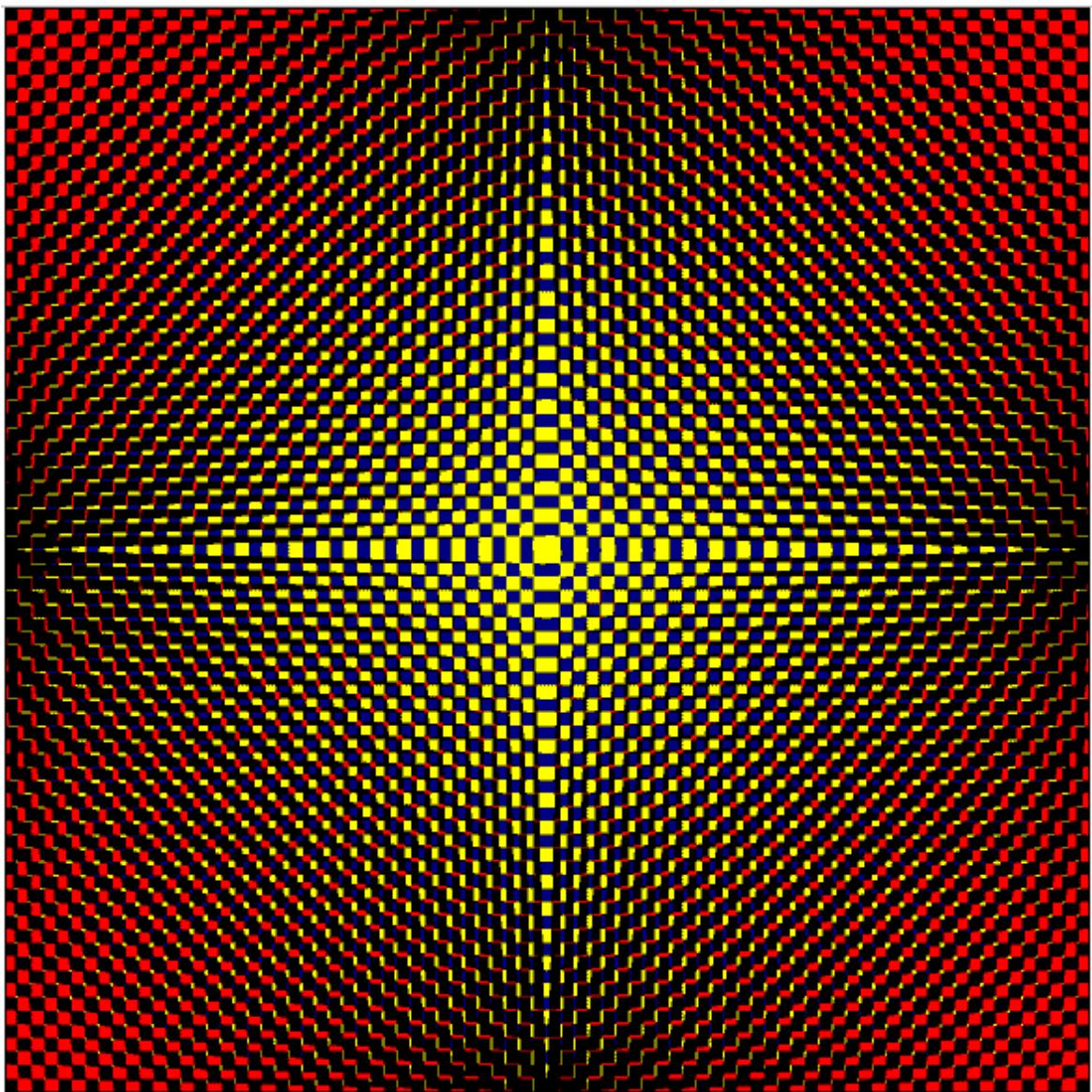


example of

```
Vector3d xa = _ZW.alignCoordSysX(_Y0);
Vector3d ya = _ZW.alignCoordSysY(_Y0);
Vector3d za = _ZW.alignCoordSysZ(_Y0);
```

## 20.4 TslArt1

Thanks to Thorsten Huck, Tsl has been taken over the edge of a pure technical application.



[ O-type:

```
// basics and props
U(1, "mm");

PropDouble dHeight(0, U(20), T("Height"));
PropInt nColumn(0, 20, T("Columns"));
PropInt nRow(1, 20, T("Rows"));
```

```

PropInt nColor1(2, 1, T("Color") + " 1");
PropInt nColor3(4, 176, T("Color") + " 2");
PropInt nColor4(5, 2, T("Color") + " 3");
// PropInt nColor2(3, 2, T("Color") + " 4");

PropDouble dRot(1, 0, T("Rotation"));

// declare standards
Vector3d vx,vy,vz;
vx = XW;
vy = YW;
vz = ZW;
CoordSys cs(_Pt0, vx, vy, vz);
CoordSys csRot;
csRot.setToRotation(dRot, vx, Pt0);

Point3d ptRef = Pt0;
PLine pl(vz),pl1(vz),pl2(vz),pl3(vz);

pl.addVertex(ptRef);
pl.addVertex(ptRef + vx * nColumn * dHeight);
pl.addVertex(ptRef + vx * nColumn * dHeight - vy * dHeight);
pl.close();

pl1.addVertex(ptRef);
pl1.addVertex(ptRef + vx * dHeight);
pl1.addVertex(ptRef + vx * dHeight - nRow * vy * dHeight);
pl1.close();

pl2.addVertex(ptRef);
pl2.addVertex(ptRef - vy * dHeight);
pl2.addVertex(ptRef + nColumn * vx * dHeight - vy * dHeight);
pl2.close();

pl3.addVertex(ptRef);
pl3.addVertex(ptRef - nRow * vy * dHeight);
pl3.addVertex(ptRef + vx * dHeight - nRow * vy * dHeight);
pl3.close();

PlaneProfile ppX(cs), ppH(pl),ppV(pl1),ppH1(pl2),ppV1(pl3),
pl(cs),pp2(cs),pp3(cs),pp4(cs);

// Display
Display dp(0),dp1(nColor1),dp3(nColor3),dp4(nColor4); //,dp2(nColor2)

for (int i = 0 ; i < nRow; i++) {
    for (int k = 0 ; k < nColumn; k++) {
        if ((i%2 == 1 && k%2 == 1) || (i%2 == 0 && k%2 == 0))
        {
            ppX = ppH;
        }
    }
}

```

```
ppX.intersectWith(ppV);
pp1.unionWith(ppX);

ppX = ppH1;
ppX.intersectWith(ppV1);
pp3.unionWith(ppX);
}

else if ((i%2 == 1 && k%2 == 0) || (i%2 == 0 && k%2 == 1))
{
    //ppX = ppH;
    //ppX.intersectWith(ppV);
    //pp2.unionWith(ppX);

    ppX = ppH1;
    ppX.intersectWith(ppV1);
    pp4.unionWith(ppX);
}

ppV.transformBy(vx * dHeight);
ppV1.transformBy(vx * dHeight);
}
ppV.transformBy(-vx * nColumn * dHeight);
ppH.transformBy(-vy * dHeight);

ppV1.transformBy(-vx * nColumn * dHeight);
ppH1.transformBy(-vy * dHeight);
}

/*uncomment if you want to draw a background
PLine plBox(vz);
plBox.addVertex(ptRef);
plBox.addVertex(ptRef + vx * nColumn * dHeight);
plBox.addVertex(ptRef + vx * nColumn * dHeight - nRow * vy * dHeight);
plBox.addVertex(ptRef - nRow * vy * dHeight);
plBox.close();

PlaneProfile ppBox(plBox);

ppBox.transformBy(csRot);
dp4.draw(ppBox, _kDrawFilled);

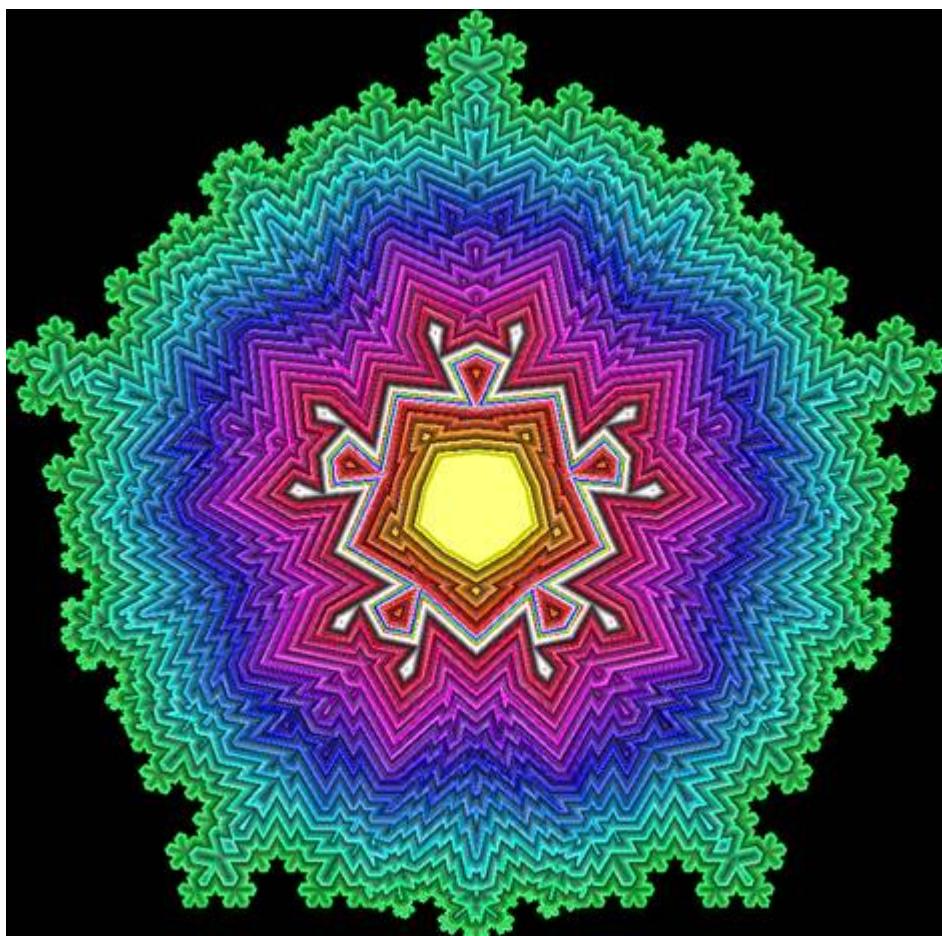
*/
pp1.transformBy(csRot);
pp3.transformBy(csRot);
pp4.transformBy(csRot);

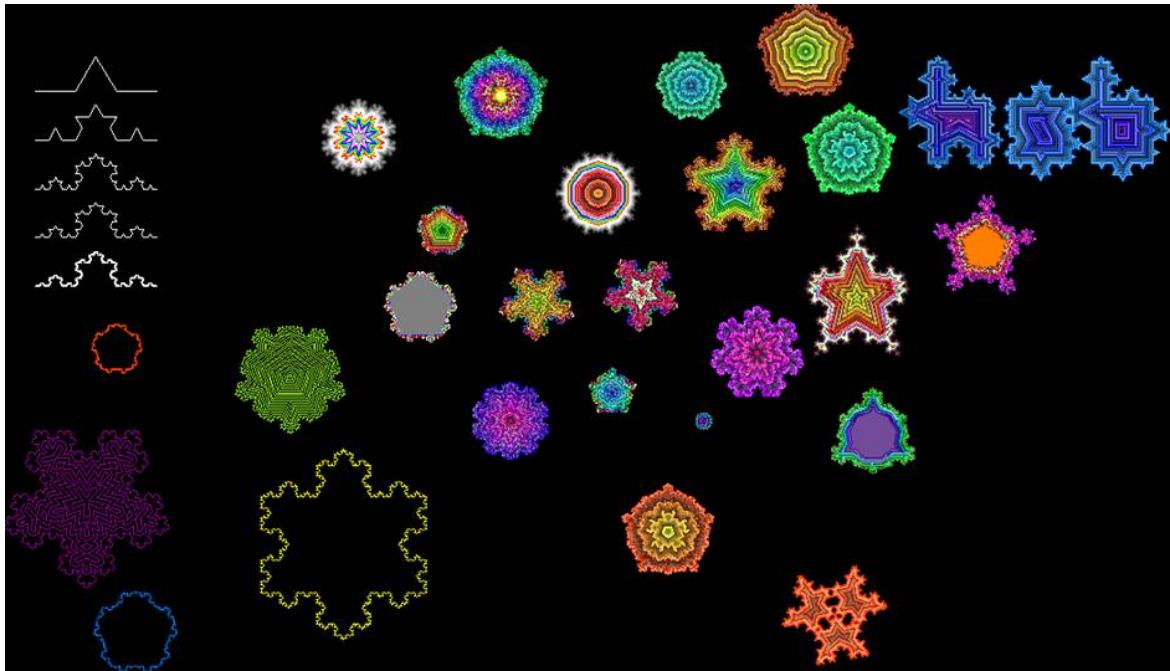
dp1.draw(pp1, _kDrawFilled);
//dp2.draw(pp2, _kDrawFilled);
dp3.draw(pp3, _kDrawFilled);
```

```
dp4.draw(pp4, _kDrawFilled);  
—end example]
```

## 20.5 TslArt2

Thanks to Thorsten Huck for this second Tsl art sample.





[ O-type:

```
//Standards
U(1, "mm");
PropInt nSteps(0, 3, T("Iteration Steps"));
PropInt nColor(1, 1, T("Start with Color"));
PropInt nSmoothen(2, 3, T("Smoothen"));
String sArMode [] = {T("Generate PLine"), T("Display Pattern")};
PropString sMode(0,sArMode,T("Mode"));
String sArNY [] = {T("No"), T("Yes")};
PropString sSwap(1,sArNY ,T("Swap Sides"));

// on insert
if (_bOnInsert)
{
    showDialog();
    EntPLine epl = getEntPLine();
    PLine pl = epl.getPLine();
    _Map.setPLine("pl",pl);
    epl.dbErase();
    return;
}

// mode
int nMode = sArMode.find(sMode);

// swap the rotation angle (inside/outside)
int nSwap = sArNY.find(sSwap);
```

```

// vecs
Vector3d vz = _ZW;

// get the source pline from map
PLine pl;
if ( _Map.hasPLine("pl"))
    pl = _Map.getPLine("pl");

// set some varias
double dLast3;
CoordSys cs;
LineSeg lsResult[0];
Point3d pt[0];
pt = pl.vertexPoints(false);

// collect linesegs from the source pline
LineSeg ls[0];
for (int p=0; p< pt.length()-1;p++)
{
    if (p==0) _Pt0 = pt[0];
    ls.append(LineSeg(pt[p], pt[p+1]));
}

// iterate
for (int c=0; c< nSteps;c++)
{
    LineSeg lsN[0]; // buffer for the segments
    for (int l=0; l< ls.length();l++)
    {
        // create new segments
        PLine c0(vz),c1(vz);
        Vector3d vx = ls[l].ptEnd()-ls[l].ptStart();
        double d3 = vx.length()/3;
        dLast3=d3;
        vx.normalize();

        // collect points to resolve one segment in to 4
        Point3d ptx[0];
        ptx.append(ls[l].ptStart());
        ptx.append(ls[l].ptStart()+vx*d3);
        double dAngle = 60;
        if (nSwap) dAngle *= -1;
        cs.setToRotation(dAngle,vz,ptx[ptx.length()-1]);
        ptx.append(ls[l].ptStart()+vx*d3*2);

        ptx[ptx.length()-1].transformBy(cs);

        ptx.append(ls[l].ptStart()+vx*d3*2);
        ptx.append(ls[l].ptStart()+vx*d3*3);
    }
}

```

```
// resolve
    for (int p=0; p< ptX.length()-1;p++)
        lsN.append(LineSeg(ptX[p],ptX[p+1]));

    }
    // rewrite source segments
    ls = lsN;
}
// store segments
lsResult.append(ls);

// recompose pline
PLine plNew(vz);
for (int p=0; p< lsResult.length();p++)
{
    lsResult[p].vis(p);
    plNew.addVertex(lsResult[p].ptStart());
    if (p==lsResult.length()-1)
        plNew.addVertex(lsResult[p].ptEnd());
}

// the modes_____
// generate pline mode
if (nMode == 0)
{
    EntPLine epl;
    epl.dbCreate(plNew);
    eraseInstance();
    return;
}

// the pattern mode
// Display
Display dp(nColor);

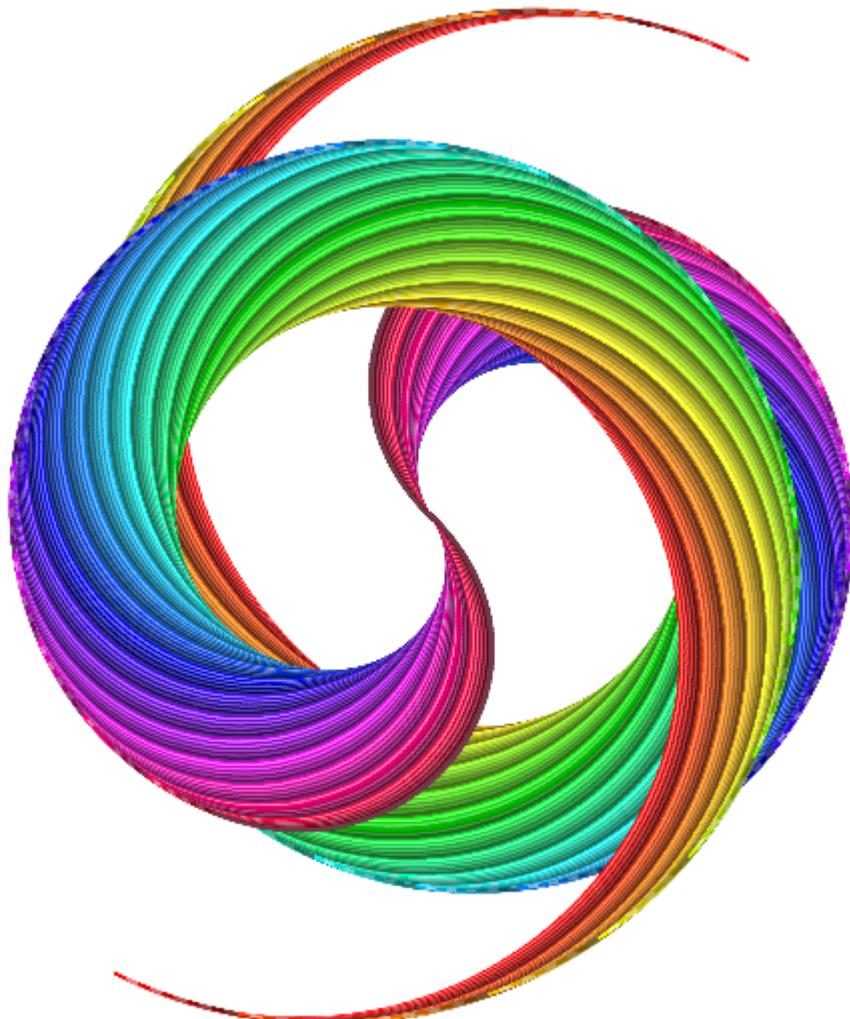
PlaneProfile pp[0];
pp.append(PlaneProfile(plNew));
dp.draw(pp[0],_kDrawFilled);
dp.color(nColor++);
int nMax = nSteps*nSmoothen*10;
for (int c = 0; c < nMax; c++)
{
    PlaneProfile ppNew = pp[pp.length()-1];
    ppNew.shrink(dLast3/nSmoothen);
    dp.color(nColor++);
    // subtract the last profile
    if (c==nMax-2)
    {
        PlaneProfile ppNewSub = ppNew;
        ppNewSub .shrink(dLast3/3);
```

```
    ppNew.subtractProfile(ppNewSub);
    dp.draw(ppNew, _kDrawFilled);
    break;
}
else
{
    pp.append(ppNew);
    dp.draw(ppNew, _kDrawFilled);
}
}

—end example]
```

## 20.6 TslArt3

Thanks to Thorsten Huck for this third Tsl art sample.



[ O-type:

1114

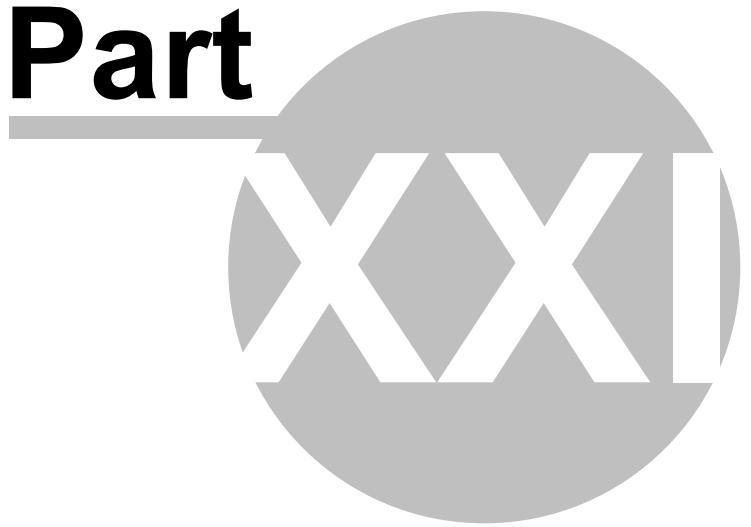
hsbCAD TSL

---

*—end example]*

---

**Part**



## 21 Appendix

### 21.1 Formal language specifications

Version 7 Oct 2003

---

```
***** LEXER *****/
S      [+ -]
D      [0-9]
E      ([DdEe][+ -]?{D}+)

// white space
[ \t]          {}
\n             {}

/*           { BEGIN CM1; }
<CM1>.        {}
<CM1>\n        {}
<CM1>/**/       { BEGIN 0; }
\\             { BEGIN CM2; }
<CM2>.         {}
<CM2>\n        { BEGIN 0; }

\"           { return STR_LIT; }
\"
<STR>([\"]|(\\\"))*
<STR>\\"         { BEGIN 0; }

// number
{D}+          { return INT_LIT; }

{D}+.{D}*({E})? | {D}*.{D}+({E})? |
{D}+{E}         { return DBL_LIT; }

'(\\"|[^\"])+' { return CHAR_LIT; }

// keywords
"true"         { return INT_LIT; }
"false"        { return INT_LIT; }
"TRUE"         { return INT_LIT; }
"FALSE"        { return INT_LIT; }
"NULL"         { return INT_LIT; }

"break"        { return BREAK; }
"continue"     { return CONTINUE; }
"return"        { return RETURN; }
```

```

"int"                                { return INT; }
"double"                             { return DOUBLE; }
"char"                               { return CHAR; }
"String"                             { return STRING; }

"Point3d"                            { return POINT3D; }
"Vector3d"                           { return VECTOR3D; }
"Line"                               { return LINE; }
"Plane"                              { return PLANE; }
"PLine"                             { return PLINE; }
"LineBeamIntersect"                  { return LBINTERSECT; }

"MetalPart"                          { return METALPART; }
"Body"                               { return BODY; }
"Unit"                               |
"U"                                   { return UNIT; }
"Beam"                               { return BEAM; }
"Sip"                                { return SIP; }
"Sheet"                             { return SHEET; }
"GenBeam"                           { return GENBEAM; }
"Hardware"                           { return HARDWARE; }
"Bolt"                               { return BOLT; }
"View"                               { return VIEW; }
"Block"                             { return BLOCK; }
"DimLine"                           { return DIMLINE; }
"Dim"                                { return DIM; }
"Entity"                            { return ENTITY; }
"Element"                           { return ELEMENT; }
"TsInst"                            { return TSLINST; }
"CoordSys"                          { return COORDSYS; }
"ElemZone"                           { return ELEMZONE; }
"Display"                            { return DISPLAY; }
"Viewport"                           { return VIEWPORT; }
"Opening"                            { return OPENING; }
"OpeningSF"                           { return OPENINGSF; }
"ElementLog"                         { return ELEMENTLOG; }

"Cut"                                 { return CUT; }
"Drill"                             { return DRILL; }
"BeamCut"                           { return BEAMCUT; }
"Slot"                               { return SLOT; }
"Arc"                                { return ARC; }
"Mortise"                           { return MORTISE; }
"House"                             { return HOUSE; }
"ParHouse"                           { return PARHOUSE; }
"Mark"                               { return MARK; }
"Dove"                               { return DOVE; }
"HousedDove"                         { return HOUSEDDOVE; }
"DoubleCut"                          { return DOUBLECUT; }
"Rabbit"                            { return RABBET; }
"Daddo"                             { return DADDO; }
"FreeProfile"                        { return FREEPROFILE; }
"SolidSubtract"                      { return SOLIDSUBTRACT; }

```

```

"ChamferedLap"           { return CHAMFEREDLAP; }
"SpecialMill"            { return SPECIALMILL; }
"CncMessage"             { return CNCMESSAGE; }
"DimTool"                { return DIMTOOL; }

"ELEMNAIL"               { return ELEMNAIL; }
"ELEMNONAIL"             { return ELEMNONAIL; }
"ELEMMILL"               { return ELEMMILL; }
"ELEMDRILL"              { return ELEMDRILL; }
"ELEMSAW"                { return ELEMSAW; }
"ELEMMARKER"             { return ELEMMARKER; }

"PROPINT"                { return PROPINT; }
"PROPDDOUBLE"             { return PROPDDOUBLE; }
"PROPSTRING"              { return PROPSTRING; }

"PRBASE"                 { return PRBASE; }
"PRPOINT"                { return PRPOINT; }
"PRENTITY"               { return PRENTITY; }

"if"                      { return IF; }
"else"                    { return ELSE; }
"while"                   { return WHILE; }
"for"                     { return FOR; }
"do"                      { return DO; }

// id
[a-zA-Z_][a-zA-Z0-9_]*      { return ID; }

// operators
"+="                      { return ADD_ASSIGN; }
"-="                      { return SUB_ASSIGN; }
"*="                      { return MUL_ASSIGN; }
"/="                      { return DIV_ASSIGN; }
"&&"                     { return AND_OP; }
"||"                      { return OR_OP; }
">>>"                     { return RIGHT_OP; }
"><<"                     { return LEFT_OP; }
"><="                      { return LE_OP; }
">>="                      { return GE_OP; }
"=="                      { return EQ_OP; }
"!="                      { return NE_OP; }
"++"                      { return INC_OP; }
"--"                      { return DEC_OP; }
"="                        { return EQUAL_OP; }
"+"                        { return PLUS_OP; }
"_"                        { return MINUS_OP; }
"**"                      { return POWER_OP; }
"/"                        { return DIVIDE_OP; }
"%"                        { return MOD_OP; }
","                        { return COMMA_OP; }
";"                        { return SEMICOLON_OP; }
"|"                        { return OR_OP; }
"|"                        { return NOT_OP; }
"^"                        { return EXPONENT_OP; }

```

```

"?"
":"
";"
","
"."
"&"
"!"
"~"
"<"
">"
"{"
"}"
"["
"]"
 "("
 ")"
{ return CHAR; }

```

\*\*\*\*\* PARSER \* Statements \*\*\*\*\*

```

input
: /* null input */
| Statements
;

Statements
: Statement
| Statement Statements
;

Statement
: Declaration ';'
| expr ':'
| ';'          /* null statement */
| BlockStatement
| IterationStatement
| SelectionStatement
| JumpStatement
;

BlockStatement
: '{' '}'
| '{' Statements '}'
;

IterationStatement
: WHILE bracketexpr Statement
| FOR '(' Statement ';' ')' Statement
| FOR '(' Statement ';' expr ')' Statement
| FOR '(' Statement expr ';' ')' Statement
| FOR '(' Statement expr ';' expr ')' Statement
| DO Statement WHILE bracketexpr ';'
;
```

## SelectionStatement

```
: IF bracketexpr Statement %prec ELSE
| IF bracketexpr Statement ELSE Statement
;
```

## JumpStatement

```
: BREAK
| CONTINUE
| RETURN
;
```

\*\*\*\*\* PARSER \* Declarations \*\*\*\*\*

## Declaration

```
: type_specifier newidlist
;
```

## type\_specifier

: INT	DOUBLE	CHAR	STRING	POINT3D
VECTOR3D	LINE	PLANE	METALPART	PLINE
UNIT	BEAM	HARDWARE	VIEW	PROPSTRING
CUT	DRILL	BEAMCUT	SLOT	MORTISE
HOUSE	MARK	PROPIINT	PROPDDOUBLE	
BODY	SIP	SHEET	ENTITY	ELEMENT
ELEMENTLOG		TSLINST	COORDSYS	DISPLAY
VIEWPORT	OPENING	OPENINGSF	GENBEAM	BLOCK
ARC	PARHOUSE	CHAMFEREDLAP		SPECIALMILL
CNCMESSAGE		DIMTOOL	DOVE	HOUSEDDOVE
DOUBLECUT	RABBET	DADDO	DIMLINE	DIM
LBINTERSECT		PRPOINT	PRENTITE	ELEMNAIL
ELEMDRILL	ELEMMILL	ELEMNONAIL	ELEMSAW	ELEMMARKER
ELEMZONE	FREEPROFILE		SOLIDSUBTRACT	

;

## newidlist

```
: newid
| newidlist ',' newid
;
```

## pointreflist

```
: pointref
| pointreflist pointref
;
```

## pointref

```
: **
| '&'
;
```

## newid

```
: ID
| ID '=' assignment_expr
```

```

| ID '[' conditional_expr ']'
| ID bracket_expr_list
| ID '[' ']' '=' accolbracket_expr_list
| ID '[' ']' '=' assignment_expr
| pointreflist ID
| pointreflist ID '=' conditional_expr
| pointreflist ID '[' conditional_expr ']'
| pointreflist ID bracket_expr_list
| pointreflist ID '[' ']' '=' accolbracket_expr_list
| pointreflist ID '[' ']' '=' assignment_expr
;

/****** PARSER * Expressions ******/
bracketexpr
: '(' expr ')'
;

expr
: assignment_expr
;

assignment_expr
: conditional_expr
| unary_expr assignment_operator assignment_expr
;

assignment_operator
: '='
| MUL_ASSIGN
| DIV_ASSIGN
| ADD_ASSIGN
| SUB_ASSIGN
;

conditional_expr
: logical_or_expr
| logical_or_expr '?' expr ':' conditional_expr
;

logical_or_expr
: logical_and_expr
| logical_or_expr OR_OP logical_and_expr
;

logical_and_expr
: inclusive_or_expr
| logical_and_expr AND_OP inclusive_or_expr
;

inclusive_or_expr
: exclusive_or_expr
;

```

```
| inclusive_or_expr "!" exclusive_or_expr  
;  
  
exclusive_or_expr  
: and_expr  
| exclusive_or_expr '^' and_expr  
;  
  
and_expr  
: equality_expr  
| and_expr '&' equality_expr  
;  
  
equality_expr  
: relational_expr  
| equality_expr EQ_OP relational_expr  
| equality_expr NE_OP relational_expr  
;  
  
relational_expr  
: shift_expr  
| relational_expr '<' shift_expr  
| relational_expr '>' shift_expr  
| relational_expr LE_OP shift_expr  
| relational_expr GE_OP shift_expr  
;  
  
shift_expr  
: additive_expr  
| shift_expr LEFT_OP additive_expr  
| shift_expr RIGHT_OP additive_expr  
;  
  
additive_expr  
: multiplicative_expr  
| additive_expr '+' multiplicative_expr  
| additive_expr '-' multiplicative_expr  
;  
  
multiplicative_expr  
: cast_expr  
| multiplicative_expr '*' cast_expr  
| multiplicative_expr '/' cast_expr  
| multiplicative_expr '%' cast_expr  
;  
  
cast_expr  
: unary_expr  
;  
  
unary_expr  
: postfix_expr  
| unary_operator cast_expr
```

---

```

;

unary_operator
: INC_OP
| DEC_OP
| '&'
| '**'
| '-'
| '+'
| '~'
| '!'
;

postfix_expr
: primary_expr
| postfix_expr '[' expr ']'
| postfix_expr INC_OP
| postfix_expr DEC_OP
| ID bracket_expr_list
| postfix_expr '.' ID bracket_expr_list
;

primary_expr
: bracketexpr
| typeSpecifier bracket_expr_list      /* eg.: int(2) or Point3d(1,2,3) */
| ID
| INT_LIT
| DBL_LIT
| CHAR_LIT
| STR_LIT
;

bracket_expr_list
: '(' ')'
| '(' argument_expr_list ')'
;

acolbracket_expr_list
: '{' '}'
| '{' argument_expr_list '}'
;

argument_expr_list
: assignment_expr
| argument_expr_list ',' assignment_expr
;

```

## 21.2 Hungarian notation

Type	Prefix	sample
<b>int</b>	n	nAmount
	b	bNoYes
	i	iVal
<b>Vector3d</b>	v	vX
	vec	vecX
<b>Point3d</b>	pt	ptInsert
<b>double</b> d		dSize
<b>String</b> s	sText	
	str	strText
<b>Properties</b>		
<b>PropDouble</b>	d	dValue
<b>PropInt</b>	n	nAmount
<b>PropString</b>	s	sText
array of		
<b>String</b> sAr		sArText
Type	Prefix	sample
<b>Arc</b>	arc	arc1
<b>Beam</b>	bm	bmToDim
<b>BeamCut</b>	bc	bcTop
<b>Body</b>	bd	bdBm
<b>Chamfer</b>	ch	chLeft
<b>ChamferedLap</b>	cl	cl1
<b>CncExport</b>	tl	tlSpecial
<b>ComplexProfile</b>	cp	cp1
<b>Cut</b>	ct	ctEnd
<b>Dado</b>	dd	ddGlas
<b>DiagonalNotch</b>	dn	dn1
<b>Dim</b>	dim	dim1
<b>DimLine</b>	dl	dl1
<b>DimTool</b>	dt	dtCenter
<b>Display</b>	dp	dpModel
<b>DoubleCut</b>	dc	dcEnd
<b>Dove</b>	dv	dvEnd
<b>Drill</b>	dr	drSink
<b>Drill</b>	dr	dr1
<b>ElemConstructionBeam</b>	ecb	ecb1
<b>ElemDrill</b>	edr	edrSink
<b>Element</b>	el	elToDim
<b>ElementLog</b>	el	elLog
<b>ElementMulti</b>	el	elMW
<b>ElementRoof</b>	el	elRoof
<b>ElementWall</b>	el	elWall
<b>ElementWallSF</b>	el	elWallSF
<b>ElemItem</b>	ei	eiSpecial
<b>ElemMarker</b>	emrk	emrkCenter
<b>ElemMill</b>	em	em1
<b>ElemNail</b>	en	en1
<b>ElemNailCluster</b>	enc	enc1
<b>ElemNoNail</b>	enn	enn1

ElemRoofEdge	edge	edgeList
ElemSaw	es	es1
ElemText	etxt	etxtLeft
ElemZone	ez	ez1
Entity	ent	entCircles
EntPLine	epl	eplContour
ERoofPlane	erp	erpMain
ERoofPlaneOpening	op	opInRoof
ExtrProfileCut	epc	epc1
FreeProfile	fp	fpMill
GenBeam	gb	gbToLabel
Grip	gd	gdWorld
Group	gr	grAll
HalfCut	hc	hc1
Hatch	hatch	hatchNet
House	hs	hsEnd
HousedDove	dv	dvEnd
Line	In	InSort
LogCourse	log	logCourses
LogNotch	lgn	lgn1
Mark	mrk	mrkCenter
MarkerLine	ml	mlCenter
Mortise	ms	ms1
Nailline	nl	nlZone1
Opening	op	opInEl
OpeningLog	op	opInEl
OpeningRoof	op	opInEl
OpeningSF	op	opInEl
PanelSplit	psp	pspCenter
PanelStop	ps	ps1
ParHouse	ph	phCenter
Plane	pn	pnToProject
PlaneProfile	pp	ppBmFront
PLine	pl	plShape
PrEnt	ss	ssE
PrPoint	ss	ssP
Rabbet	rb	rbGlas
ScarfJoint	sj	sj1
Sheet	sh	shToMill
Sip	sip	sipPanel
Slot	sl	slCenter
SolidSubtract	sosu	sosuShape
SpecialMill	sm	sm1
String	s	sName
ToolEnt	tent	tentDrill
TslInst	tsl	tslSatellite
ViewPort	vp	vpSelected
Wall	wl	wlMain

## 21.3 Special comments

The Tsl author is not the person to be responsible for the actual help information, but .... The tsl author should provide enough information to allow somebody else to compose this information.

To assist in composing that information, the tsl author should answer the following questions. The current suggestion is to use tags to format the answers. Any of the tags can have a Lang attribute specifying the language.

- What is the purpose of this tsl? (please add answer to description of tsl)

**<insert></insert>**

- Does this ts1 insert other ts1s, or is it inserted by other ts1.
- How does the insert work (what entities need to be selected, what is procedure).

**<property name="xx"></ property >**

- What are the important properties.

**<command name="xx"></command>**

- What are the custom commands.

**<remark></remark>**

- What could make the ts1 fail.
- Is this an Element ts1 that can be added to the list of auto attach ts1s.
- What is the (internal) result of the ts1: adds element/beam tools, changes properties of other entities,...
- Requirements and scope of applying this ts1 (display configuration, styles, hatch, )
- Dependency on other entities (maps, properties, property sets, auto properties,...)
- Dependency on other software components (dlls, xml, catalogs, ...)

**<version value="M.m" date="DDmonthYY" author="XX"></version>**

- What is the history of versions

These questions should be seen as triggers to the ts1 author to help him/her to provide enough information for the help composer. Not necessarily all questions need to be answered. There might be other questions relevant.

Please add these answers at the start of the ts1, in the ts1 code.

*[Example O-type with insert done in script. TSL should be added in Layout.]*

```

///<sum m ary Lang=en>
///Appended to an elem ent, this ts1m arks the none verticalbeam s of the
///elem entw ith the verticalbeam s thathave a contactcut.
///</sum m ary>

///<insertLang=en>
///This ts1can be appended to an elem ent. Itcan also be added to the list
///ofts1s to be added atgenerate construction tim e.
///</insert>

///<rem ark Lang=en>

```

```

/// * Because all none vertical beam s are marked, also the beam s of the modules are
marked.
/// * This ts1 is added to the elements zone 5 as tool, to easily toggle it on and off.
/// </acemark>

</version value="1.2" date="06m arch09"></version>

```

*—end example]*

## 21.4 Tsl editor shortcuts

Below is the list of the new Tsl editor shortcuts.

List of shortcuts:

1)	Ctrl + F	= Find
2)	F3	= Find next
3)	Ctrl + F3	= Find highlighted text and highlight next hit
4)	Shift + F3	= Find previous
5)	Ctrl + H	= Replace
6)	Ctrl + G	= Go to line
7)	Ctrl + Scroll wheel	= Zoom in/out
8)	Ctrl + Space	= Show auto complete window
9)	Hold Ctrl + Press M	= Collapse/expand regions
10)	F5	= Run to next break point or last line if it does not exist
11)	F6	= Init/Compile
12)	F10	= Step once
13)	Ctrl + Shift + M	= Maximise editor window (all other windows still use Esc.)
14)	Ctrl + K	= Comment
15)	Ctrl + U	= Uncomment
16)	Ctrl + S	= Emergency save
17)	Ctrl + L	= Emergency load
18)	F1	= Open help. If a word is highlighted it will put that in the search term

The ICSharp Avalon editor provides a host of shortcuts which are too many to list but I do find the following useful:

a)	Ctrl + End	= Go to end of document
b)	Ctrl + Home	= Got to top of document
c)	Ctrl + Side arrow	= Move cursor to next word
d)	Shift + Delete or Ctrl + X without highlighting	= Delete and copy line to clipboard
e)	Alt + Left mouse + Select multiple lines	= Allows you to edit/overwrite multiple lines in one go
f)	Ctrl + Z	= Undo
g)	Ctrl + D	= Delete
h)	Ctrl + X	= Cut
i)	Ctrl + V	= Paste
j)	Ctrl + C	= Copy
k)	Ctrl + A	= Select all

The snippet editor variables:

\$END\$ Cursor location after insert of snippet

\$VARIABLE\$ All appearances of \$VARIABLE\$ will be replaced after insert by a given value

```
$INDEX$      $INDEX$ expects the input of an integer. It will replace all occurrences
$COLLECTION$    $COLLECTION$ expects the input of an array name. It will replace all
                 occurrences
$TYPE$        $TYPE$ expects the input of a class name. It will replace all occurrences
```

The snippets are found in file %USERPROFILE%  
 \Documents\hsbCAD\TSL\Tsl.settings.

Special editor keys:

```
//region      sets the first line of a collapsible region
//endregion    sets the last line of a collapsible region
```

## 21.5 Notepad++

Since I often use Notepad++ as text editor (<http://notepad-plus.sourceforge.net>), I have investigated a bit how to configure it to view Tsl's similar as in the tsl editor.

There are basically 2 methods provided:

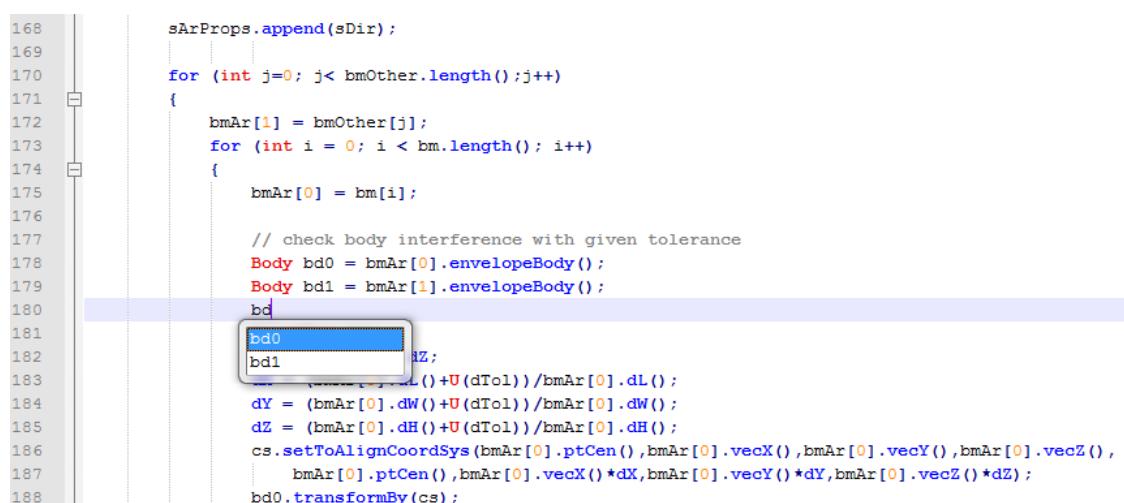
- Use an existing language, and add additional keywords
- Define a custom language

None of those 2 methods is completely satisfactory, but both are workable. The easiest way, (and for the folding option to work well the only way), is to start from an existing language. Starting from "Objective-C" gives a good result. Here are the modifications that I have done:

- Added user defined TYPE WORD, list below.
- Added user defined INSTRUCTION WORD
- Set the correct colour for strings, comments,...
- Set the mcr as known file extension for the language

And even better: if you enable the 'auto complete' in the settings you get the suggestions of all type words, instruction words and your own declarations (Thank you TH)!

The result looks like the following:



A screenshot of the Notepad++ code editor. The code is written in TSL and shows several foldable regions indicated by square icons on the left margin. The cursor is positioned at line 180, where the variable 'bd' is being typed. A dropdown auto-completion menu is open over the cursor, listing suggestions: 'bd0', 'bd1', 'bdL', 'bdR', 'bdT', and 'bdB'. The code itself includes various TSL constructs like loops, arrays, and class definitions.

```

168 sArProps.append(sDir);
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
  
```

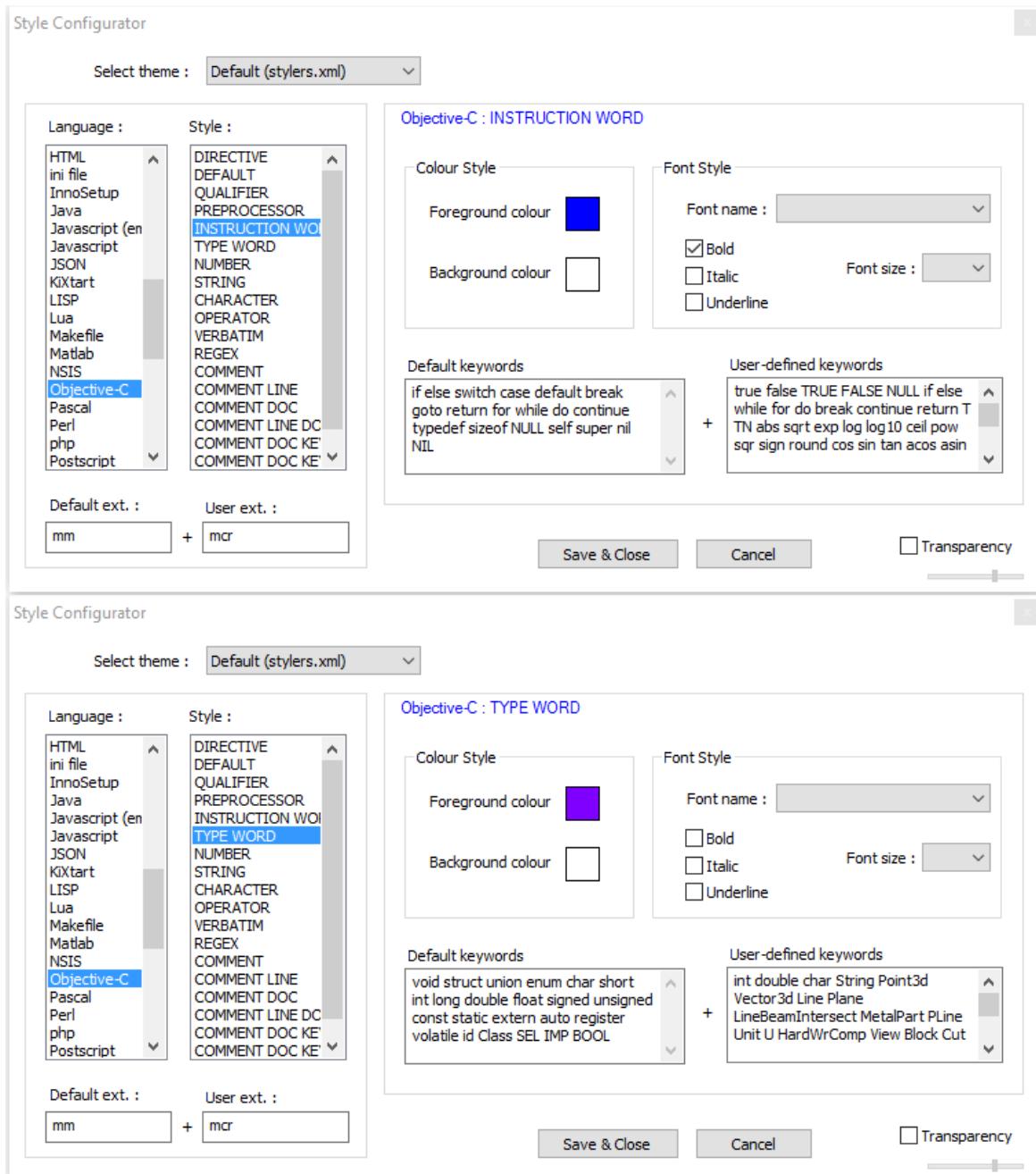
```

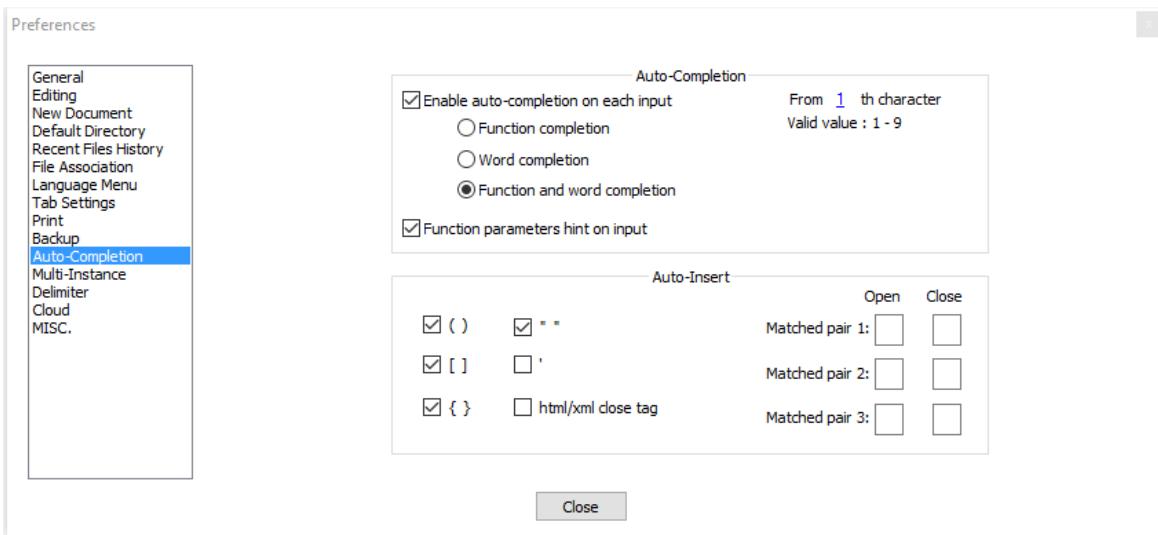
sArProps.append(sDir);

for (int j=0; j< bmOther.length();j++)
{
    bmAr[1] = bmOther[j];
    for (int i = 0; i < bm.length(); i++)
    {
        bmAr[0] = bm[i];

        // check body interference with given tolerance
        Body bd0 = bmAr[0].envelopeBody();
        Body bd1 = bmAr[1].envelopeBody();
        bd_
        bd0
        bd1
        bdL
        bdR
        bdT
        bdB
        bdL = (bmAr[0].dL() +U(dTol))/bmAr[0].dL();
        dY = (bmAr[0].dW() +U(dTol))/bmAr[0].dW();
        dZ = (bmAr[0].dH() +U(dTol))/bmAr[0].dH();
        cs.setToAlignCoordSys(bmAr[0].ptCen(),bmAr[0].vecX(),bmAr[0].vecY(),bmAr[0].vecZ(),
        bmAr[0].ptCen(),bmAr[0].vecX()*dX,bmAr[0].vecY()*dY,bmAr[0].vecZ()*dZ);
        bd0.transformBy(cs);
  
```

With the following configuration settings of Notepad++ you can achieve this:





There are 2 sets of "User-defined keywords" which need to be added to the configuration. These keywords can be retrieved from the running hsbCAD with the command:

"HSB\_LISTTSLWORDS", Option 2 (available in hsbCAD2011 build 16.0.31) gives the following output:

```
== User defined keywords for TYPE WORD ==
int double char String Point3d Vector3d Line Plane LineBeamIntersect MetalPart PLine Unit U
HardWrComp View Block Cut Drill BeamCut Slot Mortise House Dove Mark HousedDove Ari
RoundWoodMill DoubleCut Klingschrot PropellerSurfaceTool PropInt PropDouble PropString
DimLine Dim Arc ParHouse PrPoint PrEntity CoordSys ElemZone ElemNail ElemNoNail
ElemSaw ElemMill ElemDrill ElemMarker Rabbet Daddo FreeProfile Display SolidSubtract Body
Viewport ViewData EntityCollection ChamferedLap SpecialMill RevolutionMill CncMessage
Entity ToolEnt GenBeam Beam Sip Sheet Element TsInst Opening OpeningSF Wall Slab
ShopDrawView MasterPanel ChildPanel CollectionEntity TrussEntity MetalPartCollectionEnt
MvBlockRef FastenerAssemblyEnt MassElement MassGroup BlockRef ElementWall
ElementLog DimTool Map ModelX ModelXComposeSettings ModelXInterpretSettings
CncExport DiagonalNotch Elemltem MetalTKey ElemConstructionBeam ElemConstructionMap
NailLine CncCurveEnt EcsMarker CurvedDescription PressEnt PlaneProfile LineSeg ERoofPlane
Chamfer ScarfJoint SimpleScarf HalfCut PanelStop ElemText ElementWallSF EntPLine
PanelWirechase Group ElementRoof ElelRoofEdge OpeningRoof OpeningLog MarkerLine
LogNotch TirolerSchloss Grid DictObject ExtrProfile CurvedStyle ComplexProfile
ConvexConcaveProfile ERoofPlaneOpening ExtrProfileCut SipComponent SipStyle SipEdge
LogCourse Hatch PanelSplit ElelNailCluster ElementMulti SingleElementRef
MasterPanelStyle CollectionDefinition TrussDefinition MetalPartCollectionDef MvBlockDef
FastenerAssemblyDef CncCurveStyle ElelNumber MultiPageStyle SurfaceQualityStyle
RenderMaterial MapObject Quader DimRequest AnalysedTool AnalysedCut AnalysedBeamCut
AnalysedDoubleCut AnalysedDrill AnalysedHouse AnalysedMortise AnalysedSlot
AnalysedDove AnalysedSpecialMill AnalysedConvexConcaveProfile AnalysedComplexProfile
AnalysedLogNotch AnalysedDiagonalNotch AnalysedShoulderTenon AnalysedParHouse
AnalysedMarker AnalysedFreeProfile AnalysedChamferedLap AnalysedKamatsugi
AnalysedDovesugi AnalysedAri AnalysedSimpleScarf AnalysedScarfJoint AnalysedArc
AnalysedRabbet AnalysedLogDove AnalysedChamfer AnalysedPlaning AnalysedMarkerLine
```

AnalysedHalfCut NesterChild NesterMaster NesterData NesterCaller FastenerGuideline  
 FastenerComponentData FastenerArticleData FastenerSimpleComponent  
 FastenerListComponent TestReport PropellerSurface DimRequestLinear DimRequestPoint  
 DimRequestAngular DimRequestText DimRequestPitch DimRequestObholz  
 DimRequestHeightLevel DimRequestMultiViewLine DimRequestRadial DimRequestPLine  
 DimRequestChain DimRequestHatch

==== User defined keywords for INSTRUCTION WORD ====

true false TRUE FALSE NULL if else while for do break continue return T TN abs sqrt exp log  
 log10 ceil pow sqr sign round cos sin tan acos asin atan material model dxaout  
 dxaExportObject reportError reportWarning reportMessage reportNotice setCompareKey  
 setOPMKey scriptName eraseEntity eraseInstance colorFromLayer setProjectName  
 setProjectNumber setProjectStreet setProjectCity setProjectComment setProjectUser  
 setProjectRevision setProjectSpecial subMapXKeysProject subMapXProject  
 setSubMapXProject removeSubMapXProject projectName projectNumber projectStreet  
 projectCity projectComment projectUser projectRevision projectSpecial arxVersion bits  
 hsbOEVersion getTickCount length bIsZeroLength swap insertAt removeAt setLength append  
 addPart copyPart subPart normalize normal projectVector dotProduct crossProduct  
 alignCoordSysX alignCoordSysY alignCoordSysZ intersect projectPoint visualize vis  
 setRoundType setEndType setMustCut find isPerpendicularTo isParallelTo isCodirectionalTo  
 angleTo rotateBy coordSys dPosZ getPoint getString getInt getDouble getAt getElement  
 getElementFromEntity getTslInst getNailLine getCncCurveEnt getCurvedDescription  
 getEcsMarker getPressEnt getERoofPlane getElementRoof getElementMulti getViewPort  
 getOpening getOpeningRoof getERoofPlaneOpening getCollectionEntity  
 getFastenerAssemblyEnt getMassElement getMassGroup getTrussEntity  
 getMetalPartCollectionEnt getWall getMvBlockRef getBlockRef getSlab getShopDrawView  
 getMasterPanel getChildPanel getEntPLine getGrid getEntity getToolEnt getBeam  
 getGenBeam getSheet getSip showDialog showDialogOnce setPropValuesFromCatalog  
 setCatalogFromPropValues setPropValuesFromMap mapWithPropValues insertCycleCount  
 pushCommandOnCommandStack left right mid spanIncluding spanExcluding delete trimLeft  
 trimRight makeUpper makeLower format formatUnit formatTime atoi atof setPtOrg  
 activeZoneIndex setMarbleDiameter setExecutionLoops bLastExecutionLoop  
 setDependencyOnBeamLength setDependencyOnEntity setEraseAndCopyWithBeams  
 setKeepReferenceToGenBeamDuringCopy dX dY dScale setToAverage callDotNetFunction1  
 callDotNetFunction2 callWebKit addRecalcTrigger setDependencyOnDictObject  
 addDimRequest lAxis pt1 pt2 vecNrm1 vecNrm2 bHasContact nNumPoints  
 filterBeamsParallelUnique filterBeamsPerpendicularSort setRadius setDoSolid setVacuum  
 setAngle X Y Z setX setY setZ exportWithElementDxa exportToDxi setSubAssemblyName  
 subAssemblyName make collectDimPoints dwgFullName dwgName spawn spawn\_detach  
 findFile makeFolder setTextPosition draw color dimStyle lineType addViewDirection  
 addHideDirection textHeight elemZone token setJapaneseMarking allowMachineForCNC  
 excludeMachineForCNC cuttingBody addMeToGenBeams addMeToGenBeamsIntersect  
 setAutoDimInfo hasIntersection vecX ptOrg projectPoints orderPoints filterClosePoints  
 closestPointTo transformBy vecY vecZ projectLineSegs addTool addVertex getDistAtPoint  
 getPointAtDist isOn vertexPoints intersectPoints intersectPLine extend convertToLineApprox  
 reverse projectPointsToPlane setNormal close area ptStart ptEnd ptMid createCircle  
 createRectangle createConvexHull createSmoothArcsApproximation repType name bVisualize

subComponents articleNumber description manufacturer category group notes quantity  
linkedEntity countType dScaleX dScaleY dScaleZ dOffsetX dOffsetY dOffsetZ dAngleA dAngleB  
dAngleG setRepType setName setBVisualize setSubComponents setArticleNumber  
setDescription setManufacturer setModel setMaterial setCategory setGroup setNotes  
setQuantity setLinkedEntity setCountType setDScaleX setDScaleY setDScaleZ setDOffsetX  
setDOffsetY setDOffsetZ setDAngleA setDAngleB setDAngleG getListOfCatalogNames  
getHardWrCompsFromCatalog getExtents dbCreateFromDxf genBeam beam sheet sip entity  
tslInst mustCut setUseDirection setInnerCylinderDiameter innerCylinderDiameter dRadius  
dDiameter bUseThisDirection setModifySectionForCnC setXYOrthogonal setDepthCorrection  
setContinuousMortise setAddKeyhole setFlatWidth setTextHeight suppressLine  
setPosnumBeam setIsFemale setCncMode setMillDiameter setMillSide  
setMaximumDeviation set setReadOnly setFormat setDeltaOnTop setReadDirection value go  
beamSet sheetSet elementSet addAllowedClass getPickFirstSS setPickFirstSS allowNested  
setToRotation setToTranslation setToMirroring setToAlignCoordSys invert det scale  
transformPoints transformLineSegs setToProjection setToAlignWorldToPlane  
setToAlignPlaneToWorld getEulerAngles dH zoneIndex code distribution strVar dVar hasVar  
setDH setCode setColor setDVar setStrVar setSideToCenter setOverShoot setDepth  
showInDxa layer textLengthForStyle textHeightForStyle showInTslInst showInShopDraw  
showInDispRep writeToDxfFile writeToDwgFile fillDisplayFromEntity combine intersectWith  
volume ptCen lengthInDirection extremeVertices allVertices filterGenBeamsIntersect  
extractContactFaceInPlane shadowProfile getSlice hideDisplay decomposeIntoLumps  
dbCreateAsMassElement createSatFile dbCreateAs3dSolid createStlFile element ptCenPS  
setPtCenPS widthPS setWidthPS heightPS setHeightPS setCoordSys viewData  
setGeoFromViewData coordSysPS setCoordSysPS showSetName setShowSetName  
viewHandle setViewHandle viewUserId setViewUserId showSetIndex setShowSetIndex  
showSetEntities setShowSetEntities showSetDefineEntities setShowSetDefineEntities  
convertFromSubMap convertToMap findDataForViewport posnum setPosnum setEntities  
setToolIndex blsA dispRepNames blsKindOf blsValid myZoneIndex dbErase dbCopy  
layerName lineWeight setLineWeight isVisible setIsVisible handle setFromHandle  
tslInstAttached assignToLayer assignToGroups assignToElementGroup  
assignToElementFloorGroup gripPoints typeName typeDxfName groups grid realBody  
createRealBodySatFile createRealBodyStlFile dbCreate setDrawOrderToFront hyperlink  
setHyperlink refDocs setRefDocs numRefDocs appendRefDoc getRefDocAt getRefDocDescAt  
getClassificationMap getAutoPropertyMap attachedPropSetNames availablePropSetNames  
getAttachedPropSetMap attachPropSet removePropSet createPropSetDefinition  
setAttachedPropSetFromMap optionEvaluate optionAssignSameRuleSet hasSubMapContainer  
subMapKeys subMap setSubMap removeSubMap subMapXKeys subMapX setSubMapX  
removeSubMapX getPLine getEntitiesWithPosnum allBlockRefPaths xrefName renderMaterial  
setRenderMaterial removeHsbData setRenderMaterialMapping removeGenBeamConnection  
hardWrComps hardWrCompAt numHardWrComps setHardWrComps resetFastenerGuidelines  
addFastenerGuideline fastenerGuidelines getAttachedFasteners ptRef dW dD vecD dL dLM  
dLMMax ptCenSolid solidLength solidWidth solidHeight blsDummy setBlsDummy  
releasePosnum assignPosnum type setType addToolStatic removeToolsStaticOfType  
removeToolStatic label subLabel subLabel2 information grade beamCode hsblId setLabel  
setSubLabel setSubLabel2 setInformation setGrade setBeamCode setHsblId setPtCtrl  
dbCopyShape dimPoints module setModule panhand setPanhand envelopeBody  
eToolsConnected subAssembly subAssemblies filterGenBeamsNotInSubAssembly

filterGenBeamsNotThis getToolsOfTypeCncExport getToolsStaticOfTypeDrill  
 getToolsStaticOfTypeCut analysedTools analysedToolsFromEnvelope blsCutStraight dbSplit  
 dbJoin blsMyEnd filterBeamsContactCut filterBeamsContactCutHead findPlaneContactCut  
 findPlaneContactCutHead filterBeamsParallel filterBeamsPerpendicular  
 filterBeamsTConnection filterBeamsCapsuleIntersect hasTConnection setD extrProfile  
 setExtrProfile cncSplinterFree setCncSplinterFree quader hatchSection setHatchSection  
 hatchHidden setHatchHidden stretchDynamicTo stretchStaticTo stretchDynamicToMultiple  
 stretchStaticToMultiple filterBeamsHalfLineIntersectSort strCutN strCutP strCutNC strCutPC  
 dCutRib1P dCutRib2P dCutRib3P dCutRib4P dCutRib1N dCutRib2N dCutRib3N dCutRib4N  
 nCutsP nCutsN hasDimensions filterBeamsCenterDistanceYZRange filterBeamsBoxLocation  
 coordSysBeamsBoxLocation composeBeamPacks isEqualComparingPosnumCriteria  
 curvedStyle setCurvedStyle texture setTexture referenceFace setReferenceFace  
 analyseFirstBeamcut analyseFirstBeamcutAreas analyseFirstBeamcutDirs  
 findFoamRemovalPanelEdge style setStyle sipEdges stretchEdgeTo woodGrainDirection  
 setWoodGrainDirection setXAxisDirectionInXYPlane lumberInstallList plEnvelope  
 plEnvelopeCnc plShadowCnc plShadow plOpenings addOpening getWirechaseSegs  
 profCncNesting setProfCncNesting setSurfaceQualityOverrideTop surfaceQualityOverrideTop  
 setSurfaceQualityOverrideBottom surfaceQualityOverrideBottom realBodyOfComponentAt  
 profShape joinRing setPIEnvelope number dPosZOutlineFront dPosZOutlineBack  
 setDPosZOutlineFront setDPosZOutlineBack dBeamHeight setDBeamHeight dBeamWidth  
 setDBeamWidth beamExtrProfile setBeamExtrProfile opening plOutlineWall zone nailLine  
 setSheetCutLocation setCutLocation setRangeSheetGap setRoofEdgeDirective  
 setDistributionStudLocation setRangeNoDistributionStud setAreaStud setAreaNoSheet  
 setSpecial setOpeningOverwrite setDistributionReferences setZone setBlockingRun subType  
 setSubType definition profNetto profBrutto setProfNetto numElemTexts elemTextAt  
 elemTexts setElemTexts dFloorGroupHeight elementGroup lock setLock segmentMinMax  
 setNumber triggerGenerateConstruction triggerGenerateSheeting noNailProfile  
 aca2HsbCatalogList aca2HsbRunRuleSet map setMap callMapIO getListOfPropNames  
 propIntName propDoubleName propStringName hasPropInt hasPropDouble hasPropString  
 propInt propDouble propString setPropInt setPropDouble setPropString gripPoint opmName  
 version sequenceNumber setSequenceNumber setDependencyOnPosnumChanged  
 setAllowGripAtPt0 allowGripAtPt0 originator modelDescription materialDescription  
 additionalNotes setOriginator setModelDescription setMaterialDescription  
 setAdditionalNotes recalc recalcNow width setWidth height setHeight rise setRise sillHeight  
 setSillHeight headHeight setHeadHeight parentEntity releaseParentEntity setParentEntity  
 plShape openingType standardSizeNameInUse widthRough heightRough  
 bStoringRoughDimensions setBStoringRoughDimensions openingDescr setOpeningDescr  
 dBeam setDBeam dBeamRight setDBeamRight dSideBeamTop setDSideBeamTop  
 dSideBeamBottom setDSideBeamBottom dGapSide setDGapSide dGapTop setDGapTop dPlate  
 setDPlate dExtraPlateBottom setDExtraPlateBottom dExtraPlateTop setDExtraPlateTop  
 dGapBottom setDGapBottom descrPlate setDescrPlate constrDetail setConstrDetail  
 constrDetailTop constrDetailBottom constrDetailLeft constrDetailRight setConstrDetailTop  
 setConstrDetailBottom setConstrDetailLeft setConstrDetailRight constrDetailShimTop  
 constrDetailShimBottom constrDetailShimLeft constrDetailShimRight setConstrDetailShimTop  
 setConstrDetailShimBottom setConstrDetailShimLeft setConstrDetailShimRight  
 constrSheetingTop constrSheetingBottom constrSheetingLeft constrSheetingRight  
 setConstrSheetingTop setConstrSheetingBottom setConstrSheetingLeft

setConstrSheetingRight descrPacking setDescrPacking cutBottomBeam setCutBottomBeam  
continueSheeting setContinueSheeting dPackingMaxHeight setDPackingMaxHeight  
dSillPlateTopUp setDSillPlateTopUp dSillPlateTopDown setDSillPlateTopDown dSillPlateTop  
setDSillPlateTop numBeamsSupport setNumBeamsSupport numBeamsNoSupport  
setNumBeamsNoSupport numBeamsSupportRight setNumBeamsSupportRight  
numBeamsNoSupportRight setNumBeamsNoSupportRight bExtraBeamsTop  
setBExtraBeamsTop bExtraBeamsBottom setBExtraBeamsBottom bHeaderBelowPlate  
setBHeaderBelowPlate setNFloorsToCarry nFloorsToCarry setStyleNameSF styleNameSF  
dNonSuppBeamLeft dNonSuppBeamRight dShiftHeaderIntoTopPlate setDNonSuppBeamLeft  
setDNonSuppBeamRight setDShiftHeaderIntoTopPlate coordSysHsb instanceWidth  
zFaceOffset totalWidth setInstanceWidth baseHeight setBaseHeight setStartEnd  
shrinkWrapBody setPIShape nestedChildPanels dYield dLength dWidth updateYield  
myCncCurveEnts getMainGrainDirectionFromChildPanels setPtRef sipToMeTransformation  
sipEntity blsFlipped setBlsFlipped setDefinition definitionObject setDefinitionObject  
vecViewBlockOffset setScale lengthSelectionIsAutomatic setLengthSelectionIsAutomatic  
holdingDistance setHoldingDistance anchorTo guidelineToolEnt hasWidth depth hasDepth  
hasHeight radius hasRadius hasRise shapeType shapeType2String string2ShapeType operation  
setCoordSysOnly assignPosnums releasePosnums blsDynamic getDynBlockPropertyMap  
getConnectedElements setWallRoofLine ptArrow setPtArrow ptStartOutline ptEndOutline  
segStartOutline segEndOutline setPtStartOutline setPtEndOutline stretchOutlineTo  
cuttingType setCuttingType exposed setExposed cornerCleanup cornerCleanupMode  
dFirstLog dTongue dGroove dGap setDFirstLog setDTongue setDGroove setDGap  
bCutHalfLogTop bCutHalfLogBottom setBCutHalfLogTop setBCutHalfLogBottom dVisibleHeight  
arLogYValues setContourPtsLeft setContourPtsRight dHeightFromCourseNr logCourses  
addPoint addText setMapKey getMapKey setMapName getMapName setInt appendInt hasInt  
setDouble appendDouble hasDouble setString appendString hasString setPoint3d  
appendPoint3d getPoint3d hasPoint3d setEntity appendEntity hasEntity setEntityArray  
getEntityArray setVector3d appendVector3d getVector3d hasVector3d setPLine appendPLine  
hasPLine setPlaneProfile appendPlaneProfile getPlaneProfile hasPlaneProfile setBody  
appendBody getBody hasBody appendMap getMap hasMap setPoint3dArray  
appendPoint3dArray getPoint3dArray hasPoint3dArray getPoint3dPLine setPoint3dXYArray  
appendPoint3dXYArray getPoint3dXYArray hasPoint3dXYArray getPoint3dXYPLine keyAt  
indexAt moveLastTo swapLastWith writeToXmlFile readFromXmlFile writeToDxxFile  
readFromDxxFile getXmlContent setXmlContent getJsonContent setJsonContent  
getDxContent setDxContent showAdd dbComposeMap dbInterpretMap exporterGroups  
exporterShortcuts callExporter addSolidInfo addAnalysedToolInfo addOldToolInfo  
addElemToolInfo addHardwareInfo addConstructionToolInfo  
addRoofplanesAboveWallsAndRoofSectionsForRoofs addCollectionAndBlockDefinitions  
addAllGroups resolveEntitiesByHandle resolveElementsByNumber  
setBeamTypeNameAndColorFromHsbId toolName setShow setInfo setStretch toolIndex  
setZoneIndex spacing setSpacing plPath setPIPath filterNailLinesCloseToSheetingEdge  
calculateAllowedNailLineSegments filterGenBeamsBeingNailed  
removeGenBeamsWithNoNailingBeamCode presses ptInsert ptBottom ptTop curveParam  
setCurveParam size offset setSize setOffset curvedDescription allRings ringIsOpening  
numRings removeAllRings pointInProfile extentInDir shrink unionWith subtractProfile project  
getGripEdgeMidPoints moveGripEdgeMidPointAt getGripVertexPoints  
moveGripVertexPointAt splitSegments distanceTo findCapsuleIntersections plWallBoundary

roofNumber setRoofNumber planeType setPlaneType runCorrector setFeed  
 setEdgeRecessType setEdgeDetailCode vecDir text subCode autoErase indexRef idRef setPt1  
 setPt2 setVecDir setText setSubCode setAutoErase setIndexRef setIdRef spacingBeam  
 setSpacingBeam deltaDistribution setDeltaDistribution forceLevelDetail setForceLevelDetail  
 distributionType setDistributionType distributionZone setDistributionZone loadBearing  
 setLoadBearing constrDetailTopLeft constrDetailTopRight setConstrDetailTopLeft  
 setConstrDetailTopRight findDescriptionForDetCode detCodeMap namePart setNamePart  
 elementLinked bExists subGroups allExistingGroups allElementGroups collectEntities  
 addEntity dbRename dFloorHeight setDFloorHeight blsDeliverableContainer  
 setBlsDeliverableContainer nEquivalentStory setNEquivalentStory constrDetailRidge  
 setConstrDetailRidge constrDetailHip setConstrDetailHip constrDetailValley  
 setConstrDetailValley constrDetailOverhang setConstrDetailOverhang beamDistribution  
 setBeamDistribution dWallPlateHeight setDWallPlateHeight dWallPlateHeight2  
 setDWallPlateHeight2 constrDetailWallPlate setConstrDetailWallPlate constrDetailWallPlate2  
 setConstrDetailWallPlate2 dKneeWallHeight setDKneeWallHeight dKneeWallHeight2  
 setDKneeWallHeight2 constrDetailKneeWall setConstrDetailKneeWall constrDetailKneeWall2  
 setConstrDetailKneeWall2 dStrutHeight setDStrutHeight dStrutHeight2 setDStrutHeight2  
 constrDetailStrut setConstrDetailStrut constrDetailStrut2 setConstrDetailStrut2 insulation  
 setInsulation dBeamSpacing setDBeamSpacing dBeamDirection dReferenceHeight  
 setDReferenceHeight tileLathDistribution setTileLathDistribution counterLathDistribution  
 setCounterLathDistribution dCounterLathSpacing setDCounterLathSpacing  
 dCounterLathHeight setDCounterLathHeight dCounterLathWidth setDCounterLathWidth  
 bPurlin setBPurlin setVertexPoints elemRoofEdges setElemRoofEdges setVecZ setVecY  
 blsAFloor setValuesFromCatalog ptNode setPtNode ptNodeOther setPtNodeOther blsDefault  
 setBlsDefault vecBevelNormal setVecBevelNormal dOpeningHeight setDOpeningHeight  
 dTopHeight setDTopHeight constrDetailBottomAngled setConstrDetailBottomAngled  
 constrDetailTopAngled setConstrDetailTopAngled bAdjustPositionToTileLath  
 setBAdjustPositionToTileLath nExtraRaftersRight setNExtraRaftersRight nExtraRaftersLeft  
 setNExtraRaftersLeft dDistanceToFirstTileLath setDDistanceToFirstTileLath blsInFloor dGapLeft  
 setDGapLeft dGapRight setDGapRight dThicknessLeft setDThicknessLeft dThicknessRight  
 setDThicknessRight dWidthLeft setDWidthLeft dWidthRight setDWidthRight dDepthLeft  
 setDDepthLeft dDepthRight setDDepthRight dMillHeightTop setDMillHeightTop  
 dMillWidthTop setDMillWidthTop dMillExtraLengthTop setDMillExtraLengthTop  
 dMillHeightBottom setDMillHeightBottom dMillWidthBottom setDMillWidthBottom  
 dMillExtraLengthBottom setDMillExtraLengthBottom dLateralCutOutAngle  
 setDLateralCutOutAngle dLateralCutOutWidth setDLateralCutOutWidth dOffsetLateralMillings  
 setDOffsetLateralMillings dOffsetHorizontalMillings setDOffsetHorizontalMillings  
 bApplyNonSplittingTooling setBApplyNonSplittingTooling bSymmetrical setBSymmetrical  
 bMeasureInLogs setBMeasureInLogs coordinate blsOn hasClosestPointTo dSizeX dSizeY  
 blsInside flipAxisToOptimalGridOrientation blsSpecial entryName getReferencesToMe  
 styleDescription setStyleDescription importFromDwg getAllEntryNamesFromDwg  
 planeProfile blsScalable getAllEntries getAllEntryNames timberName timberMaterial  
 timberGrade dTongueHeight componentProfiles baseCurve closedCurve midCurve topCurve  
 beamWidth setBeamWidth lamThickness setLamThickness cuttingCurveOffset  
 setCuttingCurveOffset lamMinimumLength setLamMinimumLength lamExtraLength  
 setLamExtraLength lamExtraHeightRaw setLamExtraHeightRaw lamExtraWidthRaw  
 setLamExtraWidthRaw glueDensity setGlueDensity lamGroups setLamGroups woodClass

setWoodClass woodKind setWoodKind lamGrading setLamGrading dryJointLamIndex  
setDryJointLamIndex dryJointLamColor setDryJointLamColor dDisplacementRefPt2  
nLamIndexRefPt2 numLams woodClassAt lamSizeAt setPreMillType setMaxDepth  
setMinDepth setStartDepth setEndDepth setIsEndTool lineSegments dThickness  
setDThickness bAllowBevels setBAllowBevels dimColor setDimColor sipComponents  
setSipComponents sipComponentAt numSipComponents axisIndex surfaceQualityTop  
surfaceQualityBottom vecNormal plEdge detailCode recessType dRecessDepth dYMin dYMax  
setAngle edgeRecessType setEdgeDetailCodeLeft edgeDetailCodeLeft  
setEdgeDetailCodeRight edgeDetailCodeRight dSplineWidth dSplineHeight setDSplineWidth  
setDSplineHeight setDWidthSteg dWidthSteg setDWidthCenter dWidthCenter  
setDLumberOffset dLumberOffset nQtyLeft nQtyRight nQtyCenter  
bCenterStudsHaveFullPanelWidth setNQtyLeft setNQtyRight setNQtyCenter  
setBCenterStudsHaveFullPanelWidth setOpening singleElementRefs  
entitiesFromMultiElementBuild setContent clearContent getAllBlocksReferenced  
listComponent setListComponent headComponents setHeadComponents tailComponents  
setTailComponents mainComponent amountOfDigits setAmountOfDigits sequenceType  
setSequenceType prefix setPrefix composed setFromComposed objectCollectionType quality  
dictionaryName recalcAllReferencesToDictionary InEdgeD plFaceD  
modifyBoxUntilFaceCompletelyOut pointAt addAllowedView setStereotype addInternalSet  
toolType toolSubType toolEntTypeName mapInternal toolEnt questionGeneric  
filterToolsOfToolType filterToolsOfEToolType genBeamQuader findClosest vecSide  
bodyPointsInPlane questionIsCompoundCut dAngle dBevel bIsFreeD dTwist dDepth  
genBeamQuaderIntersectPoints vecN1 vecN2 vecLine dAngle1 dBevel1 dAngle2 dBevel2  
bodyPointsAlongLine ptStartExtreme ptEndExtreme vecFree dInnerCylinderDiameter  
bThrough ptOnSurface nRoundType dDoveAngle dXWidth dYHeight dZDepth dYWidth  
nToolIndex dBackcut nProfileType isEndTool dStartDepth dMaxDepth dMinDepth dEndDepth  
nPreMillType vecCut ptCut bHasCut plCurve dYDepthHouse dZDepthTop dZDepthBottom  
dZHeightBeam dYWidthBeam dXWidthSide dXWidthInterior dXAxisOffsetOtherBeam  
nShoulderType dZ dBraceAngle dBraceHeight dOuterDepth dInnerDepth strText dTextHeight  
iXPosText iYPosText textOrientation iLinesToShow bTextLiesDown nCncMode nMillSide  
plDefining plCuts vecSlope dChamferAngle dSeatLength dSeatHeight dTenonZDepth  
dTenonStep dTenonXWidth dTenonYHeight dDoveYHeight dDoveZDepth dDoveXWidth  
dAriAngle dTopXWidth dTopRadius dBottomRadius dXLength dZHeight dStepHeight  
nOverShoot bIsRabbit dZTop dZBottom dContactAngle nTopBottomProcessing nFeed  
vecChamferEdges vecPlaningSurfaces vecMarkerSide isCentered isInfinite nestInOpenings  
setNestInOpenings setProfShape profToolShape setProfToolShape originatorId  
setOriginatorId rotationAllowance setRotationAllowance isStockPiece setIsStockPiece  
allowedRunTimeSeconds setAllowedRunTimeSeconds minimumSpacing setMinimumSpacing  
generateDebugOutput setGenerateDebugOutput childOffsetX setChildOffsetX childOffsetY  
setChildOffsetY nesterToUse setNesterToUse nesterNameFromType addChild addMaster  
childCount masterCount childAt masterAt childOriginatorIdAt masterOriginatorIdAt nest  
leftOverChildIndexes nesterMasterIndexes leftOverMasterIndexes childListForMasterAt  
childWorldXformIntoMasterAt childXformWithinMasterAt addStep coating norm setCoating  
setNorm stackThickness sinkDiameter mainDiameter setStackThickness setSinkDiameter  
setMainDiameter fastenerLength threadLength minProjectionLength maxProjectionLength  
setFastenerLength setThreadLength setMinProjectionLength setMaxProjectionLength  
hasLengthInfo isBespoke setHasLengthInfo setIsBespoke componentData setComponentData

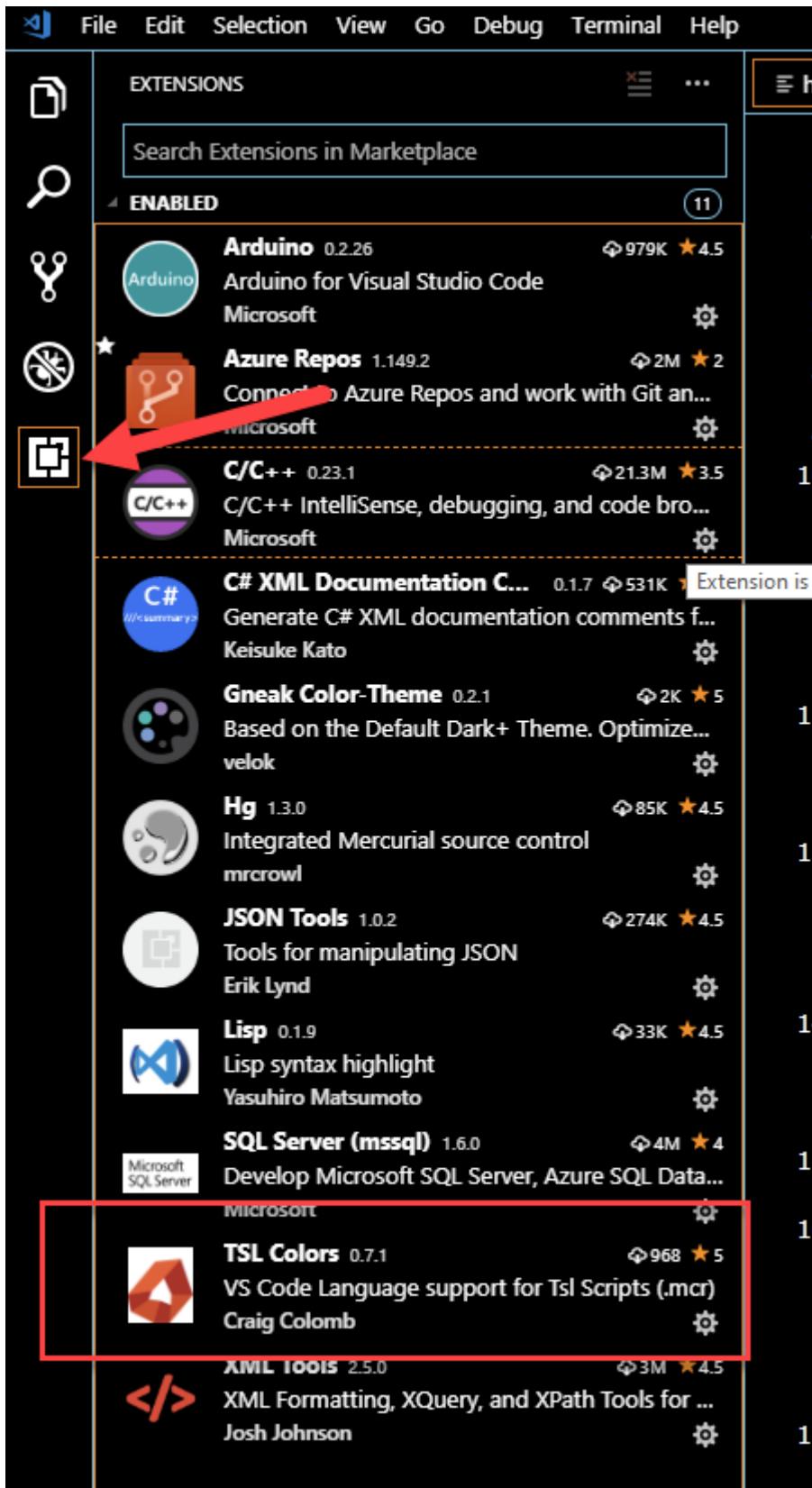
```
articleData setArticleData articleDataSet setArticleDataSet assertEquals assertEquals assertEquals assertEqualsCase assertEqualsNoCase pline1 pline2 intersectWithPlane setMinimumOffsetFromDimLine setIsChainDimReferencePoint setNodeTextDelta setNodeTextCumm addLine setShowLeaderLine setUseVerticalDimensions addToView addFromView setLineType addNode setDimStyle setHatch
```

## 21.6 VS Code

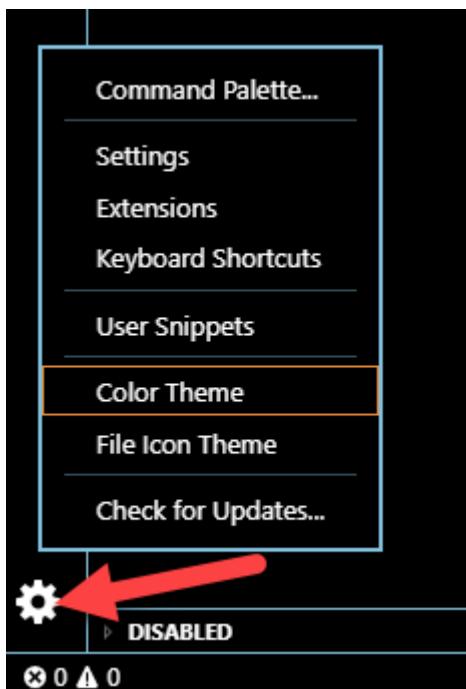
Another option for editing .mcr files outside of the ACA environment is VS Code. Thx to Craig, it is a bit more straightforward to configure for Tsl editing than Notepad++ as the marketplace has an extension for this, and you simply need to install it.

<https://code.visualstudio.com/>

After installation, search the Extension marketplace for 'Tsl Colors'.



This should allow VS Code to recognize Tsl files automatically and color code keywords, comments, and text. It will also install a new color theme which is designed to mimic the dark theme of the Tsl Editor in ACA. You can switch the color theme by clicking the gear icon bottom left of the VS Code window, then selecting Tsl Colors in the resulting pop-up dialog:



# Index

- ! -

!= 84

- : -

: 84

- ? -

? 84

- [ -

[] operator 80

- \_ -

\_Beam 443, 985

\_Beam0 443, 985

\_BeamTypes 423, 985

\_BlockNames 276, 985, 1006

\_bOnCheckConRepair0 988

\_bOnCheckConRepair1 988

\_bOnCheckSubAssembly0 221, 988

\_bOnCheckSubAssembly1 221, 988

\_bOnControlPropertyChanged 972

\_bOnDbCreated 988

\_bOnDbErase 2, 262

\_bOnDbUnErase 262

\_bOnDebug 988

\_bOnElementConstructed 196, 988

\_bOnElementDeleted 196, 988

\_bOnElementListModified 988

\_bOnElementRead 196, 988

\_bOnElementRecalc 196, 988

\_bOnGenerateShopDrawing 891, 988

\_bOnGripPointDrag 169

\_bOnInsert 961, 988

\_bOnJig 967

\_bOnMapIO 404, 883, 988

\_bOnOpeningListModified 988

\_bOnOptionChanged 233, 988

\_bOnRecalc 988

\_bOnRunTests 247

\_bOnViewportListModified 988

\_bOnViewportsSetInLayout 988, 1049

\_bOnWriteEnabled 988

\_DimStyles 985, 1006

\_Element 478, 985

\_Element0 478

\_Entity 371

\_GenBeam 985

\_Grip 255, 985

\_HatchPatterns 985, 1061

\_kAA5Axis 780

\_kAAPerpendicular 780

\_kAAri5Axis 783

\_kAAriHeadCompound 783

\_kAAriHeadPerpendicular 783

\_kAAriHeadSimpleAngled 783

\_kAAriHeadSimpleAngledTwisted 783

\_kAAriHeadSimpleBeveled 783

\_kAAriPerpendicular 783

\_kAAriRotated 783

\_kAAriTenonCompound 783

\_kAAriTenonPerpendicular 783

\_kAAriTenonSimpleAngled 783

\_kAAriTenonSimpleAngledTwisted 783

\_kAAriTenonSimpleBeveled 783

\_kAAriTilted 783

\_kABC5Axis 786

\_kABC5AxisBirdsmouth 786

\_kABC5AxisBlindBirdsmouth 786

\_kABCBirdsmouth 786

\_kABCBlindBirdsmouth 786

\_kABCClosedBirdsmouth 786

\_kABCDado 786

\_kABCDiagonalSeatCut 786

\_kABCHipBirdsmouth 786

\_kABCHouse5Axis 786

\_kABCHouseRotated 786

\_kABCHouseTilted 786

\_kABCHouse 786

\_kABCHouseThroughout 786

\_kABCJapaneseHipCut 786

\_kABCLapJoint 786

- \_kABCOpenDiagonalSeatCut 786  
\_kABCOpenSeatCut 786  
\_kABCRabbit 786  
\_kABCReversedBirdsmouth 786  
\_kABCRisingBirdsmouth 786  
\_kABCRisingSeatCut 786  
\_kABCSeatCut 786  
\_kABCSimpleHousing 786  
\_kABCValleyBirdsmouth 786  
\_kAbsolute 199  
\_kACCompound 806  
\_kACCP5Axis 803  
\_kACCPConcave 682, 803  
\_kACCPConvex 682, 803  
\_kACCPPerpendicular 803  
\_kACHExposed 793  
\_kACHip 806  
\_kACHOthersExposed 793  
\_kACL5Axis 796  
\_kACLPerpendicular 796  
\_kACP5Axis 799  
\_kACPerpendicular 806  
\_kACPPerpendicular 799  
\_kACSimpleAngled 806  
\_kACSimpleBeveled 806  
\_kAD5Axis 820  
\_kADCHead 811  
\_kADCSide 811  
\_kADCSimpleCut 811  
\_kADCValley 811  
\_kADHead 820  
\_kADN5Axis 809  
\_kADNPerpendicular 809  
\_kADo5Axis 814  
\_kADoHeadCompound 814  
\_kADoHeadPerpendicular 814  
\_kADoHeadSimpleAngled 814  
\_kADoHeadSimpleAngledTwisted 814  
\_kADoHeadSimpleBeveled 814  
\_kADoPerpendicular 814  
\_kADoRotated 814  
\_kADoTenonCompound 814  
\_kADoTenonPerpendicular 814  
\_kADoTenonSimpleAngled 814  
\_kADoTenonSimpleAngledTwisted 814  
\_kADoTenonSimpleBeveled 814  
\_kADoTilted 814  
\_kADPerpendicular 820  
\_kADRotated 820  
\_kADSFemale5Axis 817  
\_kADSFemalePerpendicular 817  
\_kADSMale5Axis 817  
\_kADSMalePerpendicular 817  
\_kADTilted 820  
\_kAFP5Axis 823, 857  
\_kAFPCenter 823, 857  
\_kAFPLeft 823, 857  
\_kAFPPerpendicular 823, 857  
\_kAFPRight 823, 857  
\_kAH5Axis 829  
\_kAHHeadCompound 829  
\_kAHHeadPerpendicular 829  
\_kAHHeadSimpleAngled 829  
\_kAHHeadSimpleAngledTwisted 829  
\_kAHHeadSimpleBeveled 829  
\_kAHPerpendicular 829  
\_kAHRotated 829  
\_kAHSimple 829  
\_kAHTenonCompound 829  
\_kAHTenonPerpendicular 829  
\_kAHTenonSimpleAngled 829  
\_kAHTenonSimpleAngledTwisted 829  
\_kAHTenonSimpleBeveled 829  
\_kAHTilted 829  
\_kAKSFemale5Axis 833  
\_kAKSFemalePerpendicular 833  
\_kAKSMale5Axis 833  
\_kAKSMalePerpendicular 833  
\_kALD5Axis 836  
\_kALDPerpendicular 836  
\_kAllBeams 195  
\_kAllSpaces 284  
\_kALN5Axis 839  
\_kALNPerpendicular 839  
\_kAM5Axis 848  
\_kAMHeadCompound 848  
\_kAMHeadPerpendicular 848  
\_kAMHeadSimpleAngled 848  
\_kAMHeadSimpleAngledTwisted 848  
\_kAMHeadSimpleBeveled 848  
\_kAMLInvisible 845  
\_kAMLSHadow 845

\_kAMLVisible 845  
\_kAMPerpendicular 848  
\_kAMRotated 848  
\_kAMShadow 842  
\_kAMSPerpendicular 875  
\_kAMTenonCompound 848  
\_kAMTenonPerpendicular 848  
\_kAMTenonSimpleAngled 848  
\_kAMTenonSimpleAngledTwisted 848  
\_kAMTenonSimpleBeveled 848  
\_kAMTilted 848  
\_kAMVisible 842  
\_kAnalysedTools 777  
\_kAngle 87, 181, 199  
\_kAnyMachine 664, 679  
\_kAOPOAdditive 607  
\_kAOPIIntersect 607  
\_kAOPOSubtractive 607  
\_kAPH5Axis 852  
\_kAPHPerpendicular 852  
\_kAPIExposed 855  
\_kAPIOthersExposed 855  
\_kArea 87, 181, 199  
\_kASJ5Axis 863  
\_kASJPerpendicular 863  
\_kASI5Axis 872  
\_kASIPerpendicular 872  
\_kASIRotated 872  
\_kASITited 872  
\_kASM5Axis 875  
\_kASMHeadCompound 875  
\_kASMHeadPerpendicular 875  
\_kASMHeadSimpleAngled 875  
\_kASMHeadSimpleBeveled 875  
\_kASMRotated 875  
\_kASMTilted 875  
\_kASS5Axis 869  
\_kAssembly 584  
\_kASSPerpendicular 869  
\_kAST5Axis 866  
\_kASTPerpendicular 866  
\_kASTRotated 866  
\_kASTTenonCompound 866  
\_kASTTenonPerpendicular 866  
\_kASTTenonSimpleAngled 866  
\_kASTTenonSimpleBeveled 866  
\_kASTTilted 866  
\_kAuto 195  
\_kBackward 730, 839  
\_kBeam0 195  
\_kBeam01 195  
\_kBespoke 263  
\_kBottom 713, 735, 842  
\_kB3dSolid 199  
\_kBTMassElement 199  
\_kBTSubDMesh 199  
\_kBTSurface 199  
\_kCancel 967  
\_kCCLeftHandShort 512  
\_kCCLogWallCrossing 512  
\_kCCMitre 512  
\_kCCMitreOpenEnd 512  
\_kCCNoCornerCleanup 512  
\_kCCConcave 773  
\_kCCConvex 773  
\_kCCRRightHandShort 512  
\_kCCSipOverlap 512  
\_kCCStretchLast 512  
\_kCCStretchThis 512  
\_kCDiagonal 773  
\_kCenter 705, 713, 735, 754, 842  
\_kCentered 730, 839  
\_kChamfer45 793  
\_kChamfer90 793  
\_kChamferRound 793  
\_kCompleted 679, 799  
\_kContext 65  
\_kContextRoot 65  
\_kCtrlKeyPressed 65  
\_kCurrentGroup 284  
\_kDevice 1006, 1035  
\_kDeviceX 1006, 1035  
\_kDimBoth 1021  
\_kDimClassic 1021  
\_kDimCumm 1021  
\_kDimDelta 1021  
\_kDimNone 1021  
\_kDimPar 1021  
\_kDimPerp 1021  
\_kDoor 584  
\_kDrawAsCurves 1006  
\_kDrawAsShell 1006

\_kDrawFilled 1006  
\_kErase 262  
\_kExecuteKey 65, 169, 218, 897, 972  
\_kExecutionLoopCount 193  
\_kExplicitRadius 746  
\_kExtrProfRectangular 321  
\_kExtrProfRound 321  
\_kExtrusionProfile2 705  
\_kExtrusionProifle1 705  
\_kFemaleEnd 696, 719, 771  
\_kFemaleSide 696, 719, 722, 771  
\_kFingerMill 705, 754, 823, 857  
\_kFixedSize 199  
\_kForward 730, 839  
\_kGripPointDrag 169  
\_kGSTAcad 255  
\_kGSTArrow 255  
\_kGSTCircle 255  
\_kGSTDiamond 255  
\_kGSTLine 255  
\_kGSTMinus 255  
\_kGSTPlus 255  
\_kGSTRotate 255  
\_kGSTSquare 255  
\_kGSTSquare45 255  
\_kGSTStar 255  
\_kGSTTriangle 255  
\_kGSTTriangleIso 255  
\_kHeaderRecess 748, 945  
\_kHidden 181  
\_kHundegger 664, 679  
\_kIsGray 284  
\_kIsNotDefined 284  
\_kIsOff 284  
\_kIsOn 284  
\_kIsotropic 423  
\_kKeyword 967  
\_kLeft 705, 713, 735, 754, 842  
\_kLength 87, 181, 199  
\_kLetIn 748, 945  
\_kList 199  
\_kLLDebug 225  
\_kLLError 225  
\_kLLFatal 225  
\_kLLInfo 225  
\_kLLNoLog 225  
\_kLLTrace 225  
\_kLLWarn 225  
\_kLogMessages 225  
\_kMaleEnd 696, 719, 722  
\_kMillWithout 679, 799  
\_kModel 1006, 1035  
\_kModelSpace 284  
\_kMPDoNotAllowMirror 296  
\_kMPUseXYPlane 296  
\_kMPUseYZPlane 296  
\_kMPUseZXPlane 296  
\_kMSTArch 604  
\_kMSTBarrelVault 604  
\_kMSTBox 604  
\_kMSTBrep 604  
\_kMSTCone 604  
\_kMSTCylinder 604  
\_kMSTDome 604  
\_kMSTDoric 604  
\_kMSTEtrusion 604  
\_kMSTGable 604  
\_kMSTIsocTriangle 604  
\_kMSTPyramid 604  
\_kMSTRevolution 604  
\_kMSTRightTriangle 604  
\_kMSTSphere 604  
\_kMultiPlexIsotropic 423  
\_kMySpace 284  
\_kNaigai 664, 679  
\_kNameLastChangedProp 181  
\_kNo 940  
\_kNoBeams 195  
\_kNoEdgeRecess 748, 945  
\_kNone 967  
\_kNonIsotropic 423  
\_kNoOpening 584  
\_kNoOverShoot 687  
\_kNotRound 746, 763  
\_kNoUnit 87, 181, 199  
\_kNTAutoNesterV4 1066  
\_kNTAutoNesterV5 1066  
\_kNTAutoNesterV6 1066  
\_kNTRectangularNester 1066  
\_kNTTestNester 1066  
\_kNYNZ 674  
\_kNYPZ 674

\_kObject 199  
\_kOCTChildPanel 343  
\_kOCTCollectionEntity 343  
\_kOCTElement 343  
\_kOCTGroup 343  
\_kOCTIndividualShopDrawing 343  
\_kOCTMassElement 343  
\_kOCTMasterPanel 343  
\_kOCTMetalPartCollectionEnt 343  
\_kOCTModuleElementDrawing 343  
\_kOCTSubassembly 343  
\_kOCTTrussEntity 343  
\_kOCTTsIDefined 343  
\_kOk 967  
\_kOnGenerateShopDrawing 888, 891, 897  
\_kOnlyBrdMth 679, 799  
\_kOpening 584  
\_kOsModeCen 967  
\_kOsModeCentroid 967  
\_kOsModeEnd 967  
\_kOsModelIns 967  
\_kOsModeMid 967  
\_kOsModeNear 967  
\_kOsModeNode 967  
\_kOsModePerp 967  
\_kOsModeQuad 967  
\_kOsModeTan 967  
\_kOverShoot 687  
\_kPathAppData 214, 217  
\_kPathCurrentDir 199, 214, 217  
\_kPathDwg 214, 217  
\_kPathHsbCompany 214, 217  
\_kPathHsblInstall 214, 217, 1073  
\_kPathHsbRoofDetail 214, 217  
\_kPathHsbWallDetail 214, 217  
\_kPathPersonal 214, 217  
\_kPathPersonalTemp 214, 217  
\_kPathProgramFiles 214, 217  
\_kPathRoamableRoot 214, 217  
\_kPathWindowsInstall 214, 217  
\_kPointInProfile 124  
\_kPointOnRing 124  
\_kPointOutsideProfile 124  
\_kPYNZ 674  
\_kPYPZ 674  
\_kRandek 664, 679  
\_kRCCollectionEntity 886  
\_kRCElement 886  
\_kRCElementModule 886  
\_kRCEntity 886  
\_kRCGroup 886  
\_kRCSubassembly 886  
\_kRCTool 886  
\_kRCToolsFromETool 886  
\_kRCTrussEntity 886  
\_kReadOnly 181  
\_kRelative 199  
\_kRelief 763  
\_kReliefSmall 763  
\_kRight 705, 713, 735, 754, 842  
\_kRound 746, 763  
\_kRounded 746  
\_kRoundSmall 763  
\_kRPTCeiling 561  
\_kRPTFloor 561  
\_kRPTRoof 561  
\_kRTBlock 398  
\_kRTBuildIn 398  
\_kRTTsl 398  
\_kRTUser 398  
\_kRunTestsEvent 247  
\_kScalable 199  
\_kShiftKeyPressed 65  
\_kShopDrawChildPageArea 892, 893  
\_kShopDrawChildPageCreate 892, 895  
\_kShopDrawEvent 892, 893, 895  
\_kShopDrawTemplateRule 908  
\_kShopDrawViewDataShowSet 897  
\_kSimpleModel 263  
\_kSpline2xLumber 748, 945  
\_kSplineBlock 748, 945  
\_kSplineCustom 748, 945  
\_kSplineDouble 748, 945  
\_kSplineJoist 748, 945  
\_kSplineSingleLumber 748, 945  
\_kSplineSingleTopSkin 748, 945  
\_kSplitAboveOpening 945  
\_kSplitBelowOpening 945  
\_kSTDigits 510  
\_kSTExcelColumn 510  
\_kSTOddCharAABB 510  
\_kStraight 313

\_kStretchNot 423  
\_kStretchOnInsert 423  
\_kStretchOnToolChange 423  
\_kTEConcave 773  
\_kTEConvex 773  
\_kTEDiagonal 773  
\_kTELTCeiling 653  
\_kTELTLast 653  
\_kTELTRoof 653  
\_kTELTRoofOverhang 653  
\_kTELTUdefined 653  
\_kTENTBearing 653  
\_kTENTGeneric 653  
\_kTENTLast 653  
\_kTENTUndefined 653  
\_kTESGenerated 653  
\_kTESInvalid 653  
\_kTESLast 653  
\_kTESUndefined 653  
\_kTESValid 653  
\_kTETGirder 653  
\_kTETHipValley 653  
\_kTETStandard 653  
\_kTETUndefined 653  
\_kThread 263  
\_kTIConcave 773  
\_kTIConvex 773  
\_kTIDiagonal 773  
\_kTileBottomOpening 566  
\_kTileEave 566  
\_kTileEdgeTooSmall 566  
\_kTileHip 566  
\_kTileHorizontalPlaneEdge 566  
\_kTileLeft 566  
\_kTileLeftOpening 566  
\_kTileLeftSloped 566  
\_kTileNotSet 566  
\_kTileOther 566  
\_kTileRidge 566  
\_kTileRight 566  
\_kTileRightOpening 566  
\_kTileRightSloped 566  
\_kTileTopOpening 566  
\_kTileValley 566  
\_kTool 199  
\_kTop 713, 735, 842  
\_kTorwegge 664, 679  
\_kTOXN 842  
\_kTOXP 842  
\_kTOYN 842  
\_kTOYP 842  
\_kTTAllTools 928  
\_kTTDrill 928  
\_kTTItem 928  
\_kTTMarker 928  
\_kTTMill 928  
\_kTTNail 928  
\_kTTNailCluster 928  
\_kTTSaw 928  
\_kUndefinedIsotropic 423  
\_kUniversalMill 705, 754, 823, 857  
\_kUpdate 972  
\_kVersion2000 1006  
\_kVersion2004 1006  
\_kVersion2007 1006  
\_kVersion2010 1006  
\_kVersionCurrent 1006  
\_kVerticalFingerMill 705, 754, 823, 857  
\_kViewDataSets 888, 897, 1057  
\_kVolume 87, 181, 199  
\_kWeinmannSaege 664, 679  
\_kWindow 584  
\_kWithUF 679, 799  
\_kXN 151  
\_kXP 151  
\_kYes 940  
\_kYN 151  
\_kYP 151  
\_kZN 151  
\_kZP 151  
\_LineTypes 921, 985, 1006  
\_Map 199, 985  
\_Opening 584  
\_Pt0 985  
\_PtE 985  
\_PtG 169, 985  
\_PtG[] 169  
\_PtG0 169  
\_PtG1 169  
\_PtG2 169  
\_PtU 985  
\_PtW 985

\_Sheet 459, 985  
 \_Sip 985  
 \_ThisInst 404, 985  
 \_Viewport 180, 218, 985, 1041, 1049  
 \_X0 985  
 \_XE 985  
 \_XF0 985  
 \_XU 985  
 \_XW 985  
 \_Y0 985  
 \_YE 985  
 \_YU 985  
 \_YW 985  
 \_Z0 985  
 \_ZE 985  
 \_ZU 985  
 \_ZW 985

## - < -

<MYPOS> 735

## - A -

abs 83  
 aca2HsbCatalogList 478  
 aca2HsbRunRuleSet 478  
 AcadDatabase 270  
 acceptObject 371  
 acos 84  
 ActiveRule 880  
 activeZoneIndex 1049  
 addAllGroups 243  
 addAllowedClass 961  
 addAllowedViewDir 912  
 addAnalysedToolInfo 243  
 addCollectionAndBlockDefinitions 243  
 addConstructionToolInfo 243  
 addDimRequest 160, 912  
 addElemToolInfo 243  
 addEntity 276  
 addFastenerGuideline 395, 402  
 addFromView 920  
 addHardwareInfo 243  
 addHideDirection 255, 1006  
 addInternalSet 912

additionalNotes 404  
 addLine 917  
 addLumberToInstallList 465  
 addMeToGenBeams 664  
 addMeToGenBeamsIntersect 664  
 addNode 921  
 addOldToolInfo 243  
 addOpening 465  
 addPart 132, 165  
 addPoint 693  
 addRecalcTrigger 65, 160, 169, 247, 262, 892, 893, 895  
 addRing 124, 465  
 addRoofplanesAboveWallsAndRoofSectionsForRoofs 243  
 addSolidInfo 243  
 addStep 402  
 addText 693  
 addTool 132, 165, 423, 443, 478, 610  
 addToolStatic 423  
 addToView 920  
 addUniqueIdsAsHandle 243  
 addVertex 114  
 addViewDirection 255, 693, 1006  
 AecFacetDev 371  
 alignCoordSysX 103  
 alignCoordSysY 103  
 alignCoordSysZ 103  
 alignment 650  
 allBlockRefPaths 371, 478  
 allDatabases 270  
 allDwgNames 270  
 allElementGroups 284  
 allExistingGroups 284  
 allowedOversize 364  
 allowedRunTimeSeconds 1066  
 allowFlippingForComparison 357  
 allowGripAtPt0 404  
 allowMachineForCNC 664  
 allowRotationForComparison 357  
 allowXrefContentToBeModified 244  
 allRings 124  
 allVertices 132  
 amountOfDigits 510  
 AnalysedArc 780  
 AnalysedAri 783

AnalysedBeamCut 786  
 AnalysedChamfer 793  
 AnalysedChamferedLap 796  
 AnalysedComplexProfile 799  
 AnalysedConvexConcaveProfile 803  
 AnalysedCut 806  
 AnalysedDiagonalNotch 809  
 AnalysedDoubleCut 811  
 AnalysedDove 814  
 AnalysedDovesugi 817  
 AnalysedDrill 820  
 AnalysedFreeProfile 823  
 AnalysedHouse 829  
 AnalysedKamatsugi 833  
 AnalysedLogDove 836  
 AnalysedLogNotch 839  
 AnalysedMarker 842  
 AnalysedMarkerLine 845  
 AnalysedMortise 848  
 AnalysedParHouse 852  
 AnalysedPlaning 855  
 AnalysedPropellerSurface 857  
 AnalysedRabbet 860  
 AnalysedScarfJoint 863  
 AnalysedShoulderTenon 866  
 AnalysedSimpleScarf 869  
 AnalysedSlot 872  
 AnalysedSpecialMill 875  
 AnalysedTool 423, 777, 786, 796, 799, 803, 806, 809, 811, 814, 820, 829, 839, 842, 848, 852, 866, 872, 875  
 analysedTools 423, 777  
 analysedToolsFromEnvelope 423  
 anchorTo 571  
 and 84  
 angle 930  
 angleA 545  
 angleB 545  
 angleTo 103  
 append 80, 114, 1021  
 appendBody 199  
 appendDouble 199  
 appendEntity 199  
 appendInt 199  
 appendMap 199  
 appendOpening 610  
 appendPLine 199  
 appendPoint3d 199  
 appendPoint3dArray 199  
 appendPoint3dXYArray 199  
 appendRefDoc 371  
 appendString 199  
 appendVector3d 199  
 Arc 666  
 area 114, 124, 132  
 Ari 668  
 arLogYValues 518  
 array 80  
 articleData 331  
 articleDataSet 331  
 articleNumber 330, 398, 571  
 arxVersion 160  
 asin 84  
 ASMHeadSimpleAngled 875  
 assertEquals 247  
 assertFalse 247  
 assertTrue 247  
 assignPosnum 404, 423  
 assignPosnums 607  
 assignToElementFloorGroup 160, 188, 371  
 assignToElementGroup 160, 188, 371, 1038  
 assignToGroups 160, 188, 371  
 assignToLayer 160, 188, 371  
 AssureExtrusionProfilesPresent 1087  
 assureRealSolidIsCalculated 423  
 atan 84  
 atof 87  
 atoi 87  
 attachedPropSetNames 293, 371  
 attachPropSet 371  
 auto 100  
 autoErase 502  
 automatic tool insertion 187  
 availablePropSetNames 371  
 axisIndex 357

## - B -

bAdjustPositionToTileLath 599  
 bAllowBevels 360  
 bApplyNonSplittingTooling 592  
 baseCurve 313

baseHeight 657  
 basicFoamBody 465  
 bCenterStudsHaveFullPanelWidth 945  
 bCutHalfLogBottom 518  
 bCutHalfLogTop 518  
 beam 236, 276, 296, 395, 443, 478, 522  
 Beam cut  
     Shop drawing 177  
 beamcode 423  
 BeamCut 672  
 beamDistribution 527  
 BeamTypes 985  
 beamWidth 313  
 bExclusive 188, 371  
 bExists 284  
 bExtraBeamsBottom 595  
 bExtraBeamsTop 595  
 bHasContact 146  
 bHasCut 799  
 bHeaderBelowPlate 595  
 bIncludeExtrusionProfileTongueHeight 423  
 blsA 293, 371  
 blsAFloor 527  
 blsCutStraight 443  
 blsDefault 532  
 blsDeliverableContainer 284  
 blsDummy 423  
 blsDynamic 538  
 blsFlipped 542  
 blsFreeD 786  
 blsInFloor 599  
 blsInside 578  
 blsKindOf 293, 371  
 blsMyEnd 443  
 blsOn 578  
 blsSpecial 293  
 blsValid 124, 270, 276, 291, 293, 371, 777, 912  
 blsZeroLength 103  
 bits 160  
 bLastExecutionLoop 160, 193  
 Block 276  
 BlockNames 985, 1006  
 BlockRef 371, 538  
 blockRefs 371  
 blocks 619  
 bMeasureInLogs 592  
 body 132, 698  
 bodyExtents 371  
 bodyPointsAlongLine 811  
 Bolt 398  
 bOnCheckConRepair0 988  
 bOnCheckConRepair1 988  
 bOnCheckSubAssembly0 988  
 bOnCheckSubAssembly1 988  
 bOnDbCreated 988  
 bOnDbErase 262  
 bOnDbUnErase 262  
 bOnDebug 988  
 bOnElementConstructed 196, 988  
 bOnElementDeleted 196, 988  
 bOnElementListModified 988  
 bOnElementRead 196  
 bOnElementRecalc 196, 988  
 bOnGenerateShopDrawing 891, 988  
 bOnGripPointDrag 169  
 bOnInsert 988  
 bOnJig 967  
 bOnMapIO 404, 883, 988  
 bOnOpeningListModified 988  
 bOnOptionChanged 233, 988  
 bOnRecalc 988  
 bOnRunTests 247  
 bOnViewportListModified 988  
 bOnViewportsSetInLayout 988  
 bOnWriteEnabled 988  
 box 132  
 break 82  
 bStoringRoughDimensions 584  
 bSymmetrical 592  
 bThrough 820  
 Btl 225  
 bUseThisDirection 698, 820  
 bVisualize 398  
 BvxStrategyKey 230  
 BvxUserAttribute1 230  
 BvxUserAttribute2 230  
 BvxUserAttribute3 230

## - C -

C# 1073  
 calculateAllowedNailLineSegments 630

callDotNetFunction 1073  
callDotNetFunction1 160, 1073  
callDotNetFunction2 1073  
callExporter 236  
CallMapIO 404, 1081, 1085  
canHaveValidGeom 657  
category 329, 398  
Chamfer 674  
ChamferedLap 675  
childOffsetX 1066  
childOffsetY 1066  
ChildPanel 230, 542  
childPanelFlipPrefix 357  
cleanFolder 214  
clearContent 296  
clippingBody 545  
clipVolume 545, 645  
close 114, 270  
closedCurve 313  
closestPointTo 103, 108, 110, 114, 124  
closestPointTolfCoplanar 114  
CncCurveEnt 552  
CncCurveStyle 295  
CncExport 423, 678  
CncMessage 679  
cncSplinterFree 443  
coating 329  
code 496, 502, 561  
collectDimPoints 1021  
collectEntities 284  
CollectionDefinition 296, 303, 308  
CollectionEntity 296, 550, 617, 650  
color 255, 371, 496, 1006  
colorFromLayer 160  
combine 132  
combinedClippedBodyOfEntities 545  
comment 74  
comments 74  
ComplexProfile 679  
componentData 331  
componentMaterials 321  
componentProfiles 321  
componentToExcludeFromOpenings 357  
composeBeamPacks 393, 443  
composed 510  
ComposeModelMap 1077  
compound 75  
cone 132  
ConeDrill 681  
Connection  
    Predefined variable E-connection 1001  
    Predefined variables G-connection 995  
    Predefined variables T - connection 991  
connectionTypeName 655  
connectTrusses 655  
const 79, 80  
constrDetail 532, 595  
constrDetailBottom 524, 595, 599  
constrDetailBottomAngled 599  
constrDetailHip 527  
constrDetailKneeWall 527  
constrDetailKneeWall2 527  
constrDetailLeft 524, 527, 595, 599  
constrDetailOverhang 527  
constrDetailRidge 527  
constrDetailRight 524, 527, 595, 599  
constrDetailShimBottom 595  
constrDetailShimLeft 595  
constrDetailShimRight 595  
constrDetailShimTop 595  
constrDetailStrut 527  
constrDetailStrut2 527  
constrDetailTop 524, 595, 599  
constrDetailTopAngled 599  
constrDetailTopLeft 524  
constrDetailTopRight 524  
constrDetailValley 527  
constrDetailWallPlate 527  
constrDetailWallPlate2 527  
constrSheetingBottom 595  
constrSheetingLeft 595  
constrSheetingRight 595  
constrSheetingTop 595  
Construction directives 505  
context menu 65  
continue 82  
continueSheeting 595  
Conversion to HSB Units 175  
convertFromSubMap 777, 888, 1057  
convertToLineApprox 114  
convertToMap 777, 1057  
ConvexConcaveProfile 682

coordinate 578  
 Coordinate System  
     Point and Vector manipulation 103  
 coordSys 114, 124, 148, 151, 371, 393, 423, 478, 496, 538, 542, 545, 550, 554, 557, 561, 569, 571, 578, 584, 599, 610, 619, 625, 635, 644, 645, 648, 653, 657, 780, 783, 786, 796, 799, 803, 809, 814, 817, 829, 833, 836, 839, 842, 848, 852, 860, 863, 866, 869, 872, 875, 1021, 1049, 1057  
 coordSysBeamsBoxLocation 443  
 coordSysHsb 657  
 coordSysScaled 538  
 copyMembersFrom 199  
 copyPart 132, 165  
 copyToolsFrom 423  
 cornerCleanup 512  
 cornerCleanupMode 512  
 correctTextNormalForViewDirection 1021  
 cos 84  
 counterLathDistribution 527  
 countFailed 247  
 countSuccess 247  
 countType 398  
 create 657  
 createCircle 114  
 createConvexHull 114  
 createMissingAttributes 538  
 createNewGuid 160  
 createPropSetDefinition 371  
 createRealBodySatFile 371  
 createRealBodyStlFile 371  
 createRectangle 114, 124  
 createSatFile 132, 371  
 createSmoothArcsApproximation 114, 155, 754  
 createStlFile 132, 371  
 cross product 154  
 crossProduct 103  
 currentLayout 291  
 CurvedDescription 554, 637  
 curvedStyle 313, 443  
 curveParam 637  
 Custom 65  
 CustomConstruction 948  
 CustomConstructionTool 948  
 Cut 685  
     Shop drawing 177  
 cutBottomBeam 595  
 cuttingBody 395, 664, 666, 672, 675, 679, 681, 687, 698, 705, 719, 752, 754, 758, 769, 829, 848, 872  
 cuttingBodyOfToolEnt 423  
 cuttingCurveOffset 313  
 cuttingType 512  
 cylinder 132  
 cZoneCharacter 188, 371

## - D -

Daddo 687  
 dAngle 783, 786, 796, 799, 814, 820, 829, 848, 872, 875  
 dAngle1 811  
 dAngle2 811  
 dAngleA 398  
 dAngleB 398  
 dAngleG 398  
 dAriAngle 783  
 database 293, 371  
 dBackcut 803  
 dbComposeMap 236, 243  
 dbCopy 371  
 dbCopyEntitiesFrom 478  
 dbCopyShape 423  
 dbCreate 276, 284, 296, 324, 333, 352, 357, 364, 371, 404, 443, 459, 465, 527, 533, 538, 542, 545, 550, 554, 557, 558, 560, 561, 571, 584, 592, 595, 599, 610, 619, 625, 630, 635, 637, 640  
 dbCreateAs3dSolid 132  
 dbCreateAsMassElement 132  
 dbCreateBodyEntity 199  
 dbCreateFromDwg 276  
 dbCreateFromDxf 276  
 dBeam 595  
 dBeamDirection 527  
 dBeamHeight 478  
 dBeamRight 595  
 dBeamSpacing 527  
 dBeamWidth 478  
 dbErase 276, 284, 293, 371  
 dBevel 783, 786, 814, 820, 829, 848, 872, 875  
 dBevel1 811  
 dBevel2 811  
 dblInterpretMap 236, 244  
 dbJoin 443, 459, 465

dBottomRadius 783  
dBraceAngle 866  
dBraceHeight 866  
dbRename 276, 284  
dbSplit 443, 459, 465, 657  
dChamferAngle 796  
dContactAngle 836  
dCornerRadius 829, 848  
dCounterLathHeight 527  
dCounterLathSpacing 527  
dCounterLathWidth 527  
dCutRib1N 443  
dCutRib1P 443  
dCutRib2N 443  
dCutRib2P 443  
dCutRib3N 443  
dCutRib3P 443  
dCutRib4N 443  
dCutRib4P 443  
dD 151, 423  
dDepth 786, 793, 820, 823, 829, 848, 872  
dDepthLeft 592  
dDepthRight 592  
dDiameter 681, 698, 820  
dDisplacementRefPt2 313  
dDistanceToFirstTileLath 599  
dDoveAngle 814, 817  
dDoveXWidth 817  
dDoveYHeight 817  
dDoveZDepth 817  
debug 404  
declaration 75, 79  
    Maths on double and int 83  
declarations 79  
decomposeIntoLumps 132  
defineSet 619  
definition 478, 538, 550, 571, 617, 625, 650  
definitionObject 650  
delete 87  
deleteFile 214  
deltaDistribution 524  
dEndDepth 799  
depth 604, 930, 936  
description 313, 324, 330, 398, 569, 571, 584, 599, 625, 648, 657  
descrPacking 595  
descrPlate 595  
descrSF 595  
det 148  
detailCode 476  
detCodeMap 524  
dExtraPlateBottom 595  
dExtraPlateTop 595  
dFirstLog 518  
dFlatWidth 833  
dFloorGroupHeight 478  
dFloorHeight 284  
dGap 518, 532, 866, 945  
dGapBottom 592, 595  
dGapLeft 592  
dGapRight 592  
dGapSide 595  
dGapTop 592, 595  
dGroove 518  
dH 423, 496, 502  
dHeightFromCourseNr 518  
diameter 936  
DictObject 293, 296, 313, 321, 341, 345, 357, 362, 364  
Dim 1021  
DimAngular 1028  
dimCollisionAreas 622  
dimColor 360  
DimDiametric 1033  
DimLine 1021  
dimPoints 423  
DimRadial 1030  
DimRequest 912, 916, 917, 918, 919, 920, 921  
DimRequestAngular 918  
DimRequestChain 921  
DimRequestHatch 922  
DimRequestHeightLevel 919  
DimRequestLinear 917  
DimRequestMultiViewLine 920  
DimRequestObholz 919  
DimRequestPitch 918  
DimRequestPLine 921  
DimRequestPoint 916  
DimRequestRadial 920  
DimRequestText 917  
dimStyle 1006  
DimStyles 985, 1006

DimTool 693  
dInnerCylinderDiameter 820  
dInnerDepth 866  
direction 936  
Display 364, 1006  
displayLineSegments 569  
displayModusCummulative 1021  
displayModusDelta 1021  
displaySetName 622  
dispRepNames 371  
distanceTo 108  
distribution 496  
distributionType 524  
distributionZone 524  
dKneeWallHeight 527  
dKneeWallHeight2 527  
dL 423  
dLateralCutOutAngle 592  
dLateralCutOutWidth 592  
dLength 610, 799  
dLMax 423  
dLMin 423  
dMaxDepth 799  
dMillExtraLengthBottom 592  
dMillExtraLengthTop 592  
dMillHeightBottom 592  
dMillHeightTop 592  
dMillWidthBottom 592  
dMillWidthTop 592  
dMinDepth 799  
dNonSuppBeamLeft 595  
dNonSuppBeamRight 595  
do 77  
DockLock 1085  
DocumentLock 1085  
dOffsetHorizontalMillings 592  
dOffsetLateralMillings 592  
dOffsetX 398  
dOffsetY 398  
dOffsetZ 398  
dOpeningHeight 599  
dot product 155  
DotNet 1073  
dotProduct 103  
double 79  
Conversion to HSB Units 175

Maths on double and int 83  
DoubleCut 693  
dOuterDepth 866  
Dove 696  
dPackingMaxHeight 595  
dPlate 595  
dPosZ 496  
dPosZOutlineBack 478  
dPosZOutlineFront 478  
dRadius 681, 698, 803, 820  
draw 1006, 1035  
drawHsbLogo 1006  
drawImage 87, 1006  
drawQR 87, 1006  
dRecessDepth 476  
dReducedTipWidth 833  
Drill 698  
    Shop drawing 177  
dryJointLamColor 313  
dryJointLamIndex 313  
dScale 1057  
dScaleX 398, 538, 625  
dScaleY 398, 538, 625  
dScaleZ 398, 538, 625  
dSeatHeight 817, 833  
dSeatLength 817, 833  
dShiftHeaderInTopPlate 595  
dSideBeamBottom 595  
dSideBeamTop 595  
dSillPlateTop 595  
dSillPlateTopDown 595  
dSillPlateTopUp 595  
dSizeX 578  
dSizeY 578  
dSplineHeight 945  
dSplineWidth 945  
dStartDepth 799  
dStrutHeight 527  
dStrutHeight2 527  
dTenonStep 833  
dTenonXWidth 833  
dTenonYHeight 833  
dTenonZDepth 833  
dTextHeight 842  
dThickness 341, 357, 360, 610  
dThicknessLeft 592

dThicknessRight 592  
 dTongue 518  
 dTongueHeight 321  
 dTopHeight 599  
 dTopRadius 783  
 dTopToBackstep 833  
 dTopXWidth 783  
 dTwist 783, 786, 814, 829, 848, 872, 875  
 dVar 496  
 dVisibleHeight 518  
 dW 423  
 dWallPlateHeight 527  
 dWallPlateHeight2 527  
 dwg 1006  
 dwgFullName 160  
 dwgName 160, 270  
 dWidth 610  
 dWidthCenter 945  
 dWidthLeft 592  
 dWidthRight 592  
 dWidthSteg 945  
 dX 124, 151, 635, 796, 842, 852, 860  
 dxaExportObject 178  
 dxaout 160, 178  
 dXAxisOffsetOtherBeam 839  
 DXF 1006  
 dXLength 836, 869  
 dXMax 533  
 dXWidth 814  
 dXWidthInterior 809, 839  
 dXWidthSide 839  
 dY 124, 151, 635, 796, 842, 852, 860  
 dYDepthHouse 839  
 dYHeight 783, 814  
 dYield 610  
 dYMax 522, 533  
 dYMin 522  
 dYNegDepth 809  
 dYPosDepth 809  
 dYWidth 875  
 dYWidthBeam 809, 839  
 dZ 151, 796, 852, 860  
 dZBottom 836  
 dZDepth 783, 814, 875  
 dZDepthBottom 809, 839  
 dZDepthTop 809, 839

dZHeight 869  
 dZHeightBeam 809, 839  
 dZTop 836  
**- E -**  
 E  
 Predefined variables E-connection 1001  
 EcsMarker 557  
 edgeDetailCodeLeft 945  
 edgeDetailCodeRight 945  
 EdgeName 252  
 edgeRecessType 945  
 EdgeTileData 561, 566  
 edgeTileTopology 561, 566  
 Edit script 65  
 ElemConstructionBeam 943  
 ElemConstructionMap 948  
 ElemDrill 936  
 element 371, 478, 533, 584, 1049  
 elementGroup 478  
 elementLinked 284  
 ElementLog 518  
 ElementMulti 533  
 ElementRoof 527  
 ElementWall 512, 518, 524  
 ElementWallSF 524  
 Elemltem 940  
 ElemMarker 939  
 ElemMill 933  
 ElemNail 926  
 ElemNailCluster 941  
 ElemNoNail 928  
 ElemNumber 510  
 ElemRoofEdge 532  
 elemRoofEdges 527, 532  
 ElemSaw 640, 930, 951  
 ElemText 478, 502  
 elemTexts 478, 502  
 elemTool 640  
 elemZone 496, 1006, 1038  
 else 76  
 encodedImageHeightOverWidth 87  
 encodeImageFromFile 87  
 endNodeIndex 653  
 EntCircle 560

entitiesFromMultiElementBuild 538  
 entitiesInClipVolume 545  
 Entity 236, 276, 296, 371, 395, 478, 607, 657  
 EntityCollection 393  
 Entity-derived 404  
 EntPLine 558  
 entryName 276, 291, 293  
 envelopeBody 423  
 eraseAttributes 538  
 eraseEntity 187  
 eraseInstance 160, 187  
 ERoofPlane 561  
 ERoofPlaneOpening 569  
 errorMessage 371  
 eToolsConnected 423  
 Example of Slot and Mortise 1094  
 excludeMachineForCNC 664, 678  
 exclusionEntities 545  
 expandEnvironmentVariables 214  
 export 68  
 exportAsMark 738  
 exporterGroups 236  
 exporterShortcuts 236  
 exportToDxi 160, 178  
 exportWithElementDxa 160, 178, 221  
 exposed 512  
 expression 75  
 extentInDir 124  
 extentsWcs 371  
 extra beams 68  
 extractContactFaceInPlane 132  
 extremeVertices 132  
 extrProfile 321, 443, 518, 703, 945  
 ExtrProfileCut 321, 703  
 extrusion 132

**- F -**

false  
   Logical operations 84  
 FastenerArticleData 330, 331  
 FastenerAssemblyDef 324  
 FastenerAssemblyEnt 571  
 FastenerComponentData 329, 331  
 FastenerGuideline 395, 402  
 fastenerGuidelines 395, 402

fastenerLength 330  
 FastenerListComponent 324, 331  
 FastenerSimpleComponent 324  
 filter 352  
 filterAcceptedEntities 352, 371  
 filterBeamsBoxLocation 443  
 filterBeamsCapsuleIntersect 443  
 filterBeamsCenterDistanceYZRange 443  
 filterBeamsContactCut 443  
 filterBeamsContactCutHead 443  
 filterBeamsHalfLineIntersectSort 443  
 filterBeamsParallel 103, 443  
 filterBeamsParallelUnique 103  
 filterBeamsPerpendicular 103, 443  
 filterBeamsPerpendicularSort 103  
 filterBeamsTConnection 443  
 filterClosePoints 110  
 filterEntitiesOfType 371  
 filterGenBeamsBeingNailed 630  
 filterGenBeamsIntersect 132  
 filterGenBeamsNotInSubAssembly 423  
 filterGenBeamsNotThis 423  
 filterNailLinesCloseToSheetingEdge 630  
 filterToolsOfEToolType 777  
 filterToolsOfToolType 777, 780, 783, 786, 793, 796, 799, 803, 806, 809, 811, 814, 817, 820, 823, 829, 833, 836, 839, 842, 845, 848, 852, 855, 857, 860, 863, 866, 869, 872, 875  
 filterValid 80  
 find 80, 87  
 findClosest 780, 783, 786, 793, 796, 799, 803, 806, 809, 811, 814, 817, 820, 823, 829, 833, 836, 839, 842, 845, 848, 852, 855, 857, 860, 863, 866, 869, 872, 875  
 findClosestEdgeIndex 465  
 findClosestRingIndex 465  
 findContainingRoofplanes 561  
 findDataForViewport 888, 1057  
 findDescriptionForDetCode 524  
 findFile 160, 217  
 findFoamRemovalPanelEdge 465  
 findNoCase 80  
 findParentRoofplane 561  
 findPlaneContactCut 443  
 findPlaneContactCutHead 443  
 first 80  
 flags 96

flipAlignZ 404  
 flipAxisToOptimalGridOrientation 578  
 flipNormal 114  
 foamComponentIndex 465  
 for 77  
 forceLevelDetail 524  
 format 87, 96, 352  
 formatObject 181, 371  
 formatObjectVariables 371  
 formatTime 87  
 formatUnit 87  
 freeDirectionFromRealSolid 672  
 FreeProfile 705  
 FreeText 713  
 frontCutPerpToRoof 561  
 frozen 395  
 Function 181

**- G -**

**G**  
 Predefined variables G-connection 995  
 genBeam 2, 236, 276, 296, 395, 423, 465, 478, 777  
 GenBeam-derived 404  
 genBeamQuader 777  
 genBeamQuaderIntersectPoints 786, 811, 829, 848, 872  
 generateDebugOutput 1066  
 genericQuestion 777  
 geomSupport 657  
 getAllBlocksReferenced 345  
 getAllEntries 291, 293, 295, 296, 303, 308, 313, 321, 324, 333, 341, 343, 345, 352, 357, 361, 362, 364  
 getAllEntryNames 276, 291, 295, 296, 303, 308, 313, 321, 324, 333, 341, 343, 345, 352, 357, 361, 362, 364  
 getAllEntryNamesFromDwg 296, 303, 308, 324, 333, 341, 343, 345, 357, 361  
 getAllRoofPlanes 561  
 getAt 87  
 getAttachedFasteners 395  
 getAttachedPropSetMap 293, 371  
 getAttributeList 538  
 getAttributeMap 538  
 getAutoPropertyMap 371  
 getBackgroundTrueColor 160  
 getBeam 972  
 getBlockRef 538  
 getBody 199  
 getBodyAsPlaneProfilesList 199  
 getBodyFaceLoops 199  
 getClassificationMap 293, 371  
 getClipVolume 545  
 getCollectionEntity 550  
 getConnectedElements 512  
 getCurvedDescription 554  
 getDistAtPoint 114  
 getDouble 199, 972  
 getDrawingPropertiesMap 270  
 getDxContent 199  
 getDynBlockPropertyMap 538  
 getElement 524, 972  
 getElementFromEntity 972  
 getElementMulti 533  
 getElementRoof 527, 972  
 getEntCircle 560  
 getEntitiesWithPosnum 371, 404, 423  
 getEntity 199, 972  
 getEntityArray 199  
 getEntPLine 558, 972  
 getERoofPlane 561, 972  
 getERoofPlaneOpening 569, 972  
 getEulerAngles 148  
 getExtents 276  
 getFastenerAssemblyEnt 571  
 getFilesInFolder 214  
 getFoldersInFolder 214  
 getGenBeam 972  
 getGrid 578  
 getGripEdgeMidPoints 124  
 getGripVertexPoints 124  
 getHardWrCompsFromCatalog 398  
 getInt 199, 972  
 getJsonContent 199  
 getLineTypeString 653  
 getListOfCatalogNames 398, 404, 527, 599  
 getListOfPropNames 404  
 getMainGrainDirectionFromChildPanels 610  
 getMap 181, 199  
 getMapKey 199  
 getMapName 199  
 getMassElement 604  
 getMassGroup 607

getMasterPanel 542, 610  
 getMillSidePolyNormal 705  
 getMultiPage 619  
 getMvBlockRef 625  
 getNailLine 630, 972  
 getNode.TypeString 653  
 getOpening 584, 592, 595, 972  
 getOpeningRoof 599, 972  
 getPLine 199, 371, 558  
 getPlotViewport 635  
 getPoint 972  
 getPoint3d 199  
 getPoint3dArray 199  
 getPoint3dPLine 199  
 getPoint3dXYArray 199  
 getPoint3dXYPLine 199  
 getPointAtDist 114  
 getPressEnt 637  
 getRefDocAt 371  
 getRefDocDescAt 371  
 getReferencesToMe 293  
 getResultBlock 538  
 getSawLine 640  
 getSection2d 645  
 getSheet 972  
 getShopDrawView 644, 888  
 getSip 972  
 getSlab 648  
 getSlice 132  
 getString 199, 972  
 getTangentAtPoint 114  
 getTextAreas 1021, 1028, 1030, 1033  
 getTickCount 160  
 getToolEnt 395  
 getToolsOfTypeCncExport 423  
 getToolsOfTypeDrill 478, 936  
 getToolsOfTypeFreeProfile 423, 705  
 getToolsOfTypeMill 478, 933  
 getToolsOfTypeNail 478, 926  
 getToolsOfTypeNoNail 478, 928  
 getToolsOfTypeSaw 478, 930  
 getToolsStaticOfTypeBeamCut 423, 672  
 getToolsStaticOfTypeCut 423, 685  
 getToolsStaticOfTypeDrill 423, 698  
 getTrussStatusString 653  
 getTrussTypeString 653  
 getTsIIInst 404, 972  
 getUcs 160, 284  
 getVarDouble 160  
 getVarInt 160  
 getVarString 160  
 getVector3d 199  
 getViewCenter 160  
 getViewDirection 160  
 getViewHeight 160  
 getViewport 972, 1049  
 getWall 657, 972  
 getWirechaseSegs 465  
 getXmlContent 199  
 glueDensity 313  
 go 961, 967  
 goJig 967  
 grade 329, 423  
 Grid 284, 371, 578  
 Grip 255  
 gripPoint 404  
 grip-point  
     Move grip-point to axis of beam 103  
 gripPoints 68, 169, 371  
 grips 255, 404  
 Group 284, 329, 398, 578  
 groups 371  
 groupVisibility 284  
 groupVisibilityIsOff 1049  
 groupVisibilityIsOn 1049  
 guidelineToolEnt 571

## - H -

HalfCut 716  
 handle 291, 293, 371  
 Hardware 398  
 HardWrComp 2, 395, 398  
 hardWrCompAt 395, 398  
 hardWrComps 395, 398  
 hasBody 199  
 hasClosestPointTo 578  
 hasDepth 604  
 hasDimensions 443  
 hasDouble 199  
 hasEntity 199  
 hasHeight 604

hasInt 199  
hasIntersection 110, 132  
hasLengthInfo 330  
hasMap 199  
hasPLine 199  
hasPoint3d 199  
hasPoint3dArray 199  
hasPoint3dXYArray 199  
hasPropDouble 404  
hasPropInt 404  
hasPropString 404  
hasRadius 604  
hasRise 604  
hasString 199  
hasSubMapContainer 371  
hasTConnection 443  
hasTrueColor 371  
hasVar 496  
hasVector3d 199  
Hatch 922, 1006, 1061  
hatchHidden 443  
HatchPatterns 985, 1061  
hatchSection 443  
headComponents 324  
headHeight 584  
height 364, 545, 584, 604  
heightPS 1049, 1057  
heightRough 584  
hideDirections 255  
hideDisplay 132  
holdingDistance 571  
hostId 293, 371  
House 719  
HousedDove 722  
HSB\_APPLOAD 1076  
Hsb\_GenerateConstruction 196  
-Hsb\_GenerateConstruction 196  
Hsb\_HundeggerPba 230  
HSB\_LISTTSLWORDS 1128  
HSB\_LOGSHEDWIZARD 225  
HSB\_MAPTEXTURE 354  
HSB\_READCONSTRUCTION 988  
hsb\_recalcTslWithKey 65  
HSB\_RESETOPM 65  
HSB\_SCRIPTDEF 65  
Hsb\_ScriptInsert 218, 972  
Hsb\_ScriptInsertNoPrompt 218  
Hsb\_Store3d 196  
-Hsb\_Store3d 196  
Hsb\_StoreTslStateInDwg 224  
HSB\_TIMETRACKEND 245  
HSB\_TIMETRACKSTART 245  
HSB\_TIMETRACKSTARTPROFILING 245  
hsbAnalysedToolsOut 777  
hsbCadAPI 1076, 1077, 1081  
hsbld 423  
hsbOEVersion 160  
hsbOptions 233  
HsbServices 1077, 1081  
hsbShareProjectId 160  
HundeggerPba 230  
hyperlink 371

- | -

idRef 502  
if 76  
iLinesToShow 842  
imgEncodeImageFromFile 1006  
importFromDwg 296, 303, 308, 324, 333, 341, 343, 345, 357, 361  
importProjectInfo 244  
ImportTSLIfOldVersion 1081  
includedEntities 545  
indexAt 199  
indexOfMovedGrip 255  
indexRef 502  
information 423, 478, 610  
inLayoutTab 160, 1049  
innerCylinderDiameter 698  
inPaperSpace 160, 1049  
inQrEmbeddedImageSizeFactor 87  
insert 87  
Insert done in script 68  
insertAt 80  
insertCycleCount 972  
insertSingleElement 533  
instanceWidth 657  
insulation 527  
int 79  
    Conversion to HSB Units 175  
    Maths on double and int 83

InterpretModelMap 1077, 1085  
 intersect 110  
 intersectPLine 114  
 intersectPLineAsDistances 114  
 intersectPoints 114, 124, 132, 578  
 intersectWith 132  
 intersectWithPlane 155, 754  
 invert 148  
 isActive 657  
 isBespoke 330  
 isCircle 114  
 isClosed 114  
 isCodirectionalTo 103  
 isEqualComparingPosnumCriteria 443  
 isMoved 255  
 isNull 132  
 isOn 114  
 isotropic 423  
 isParallelTo 103  
 isPerpendicularTo 103  
 isRelativeToEcs 255  
 isSectionLine 545  
 isSelfIntersecting 114  
 isStretchPoint 255  
 isValid 132  
 isVisible 371  
 iteration 75, 77  
 iTextOrientation 842  
 iXPosText 842  
 iYPosText 842

**- J -**

joinRing 459  
 jump 75, 82

**- K -**

kAA5Axis 780  
 kAAPerpendicular 780  
 kAAri5Axis 783  
 kAAriHeadCompound 783  
 kAAriHeadPerpendicular 783  
 kAAriHeadSimpleAngled 783  
 kAAriHeadSimpleAngledTwisted 783  
 kAAriHeadSimpleBeveled 783  
 kAAriPerpendicular 783  
 kAAriRotated 783  
 kAAriTenonCompound 783  
 kAAriTenonPerpendicular 783  
 kAAriTenonSimpleAngled 783  
 kAAriTenonSimpleAngledTwisted 783  
 kAAriTenonSimpleBeveled 783  
 kAAriTilted 783  
 kABC5Axis 786  
 kABC5AxisBirdsmouth 786  
 kABC5AxisBlindBirdsmouth 786  
 kABCBirdsmouth 786  
 kABCBlindBirdsmouth 786  
 kABCClosedBirdsmouth 786  
 kABCDado 786  
 kABCDiagonalSeatCut 786  
 kABCHipBirdsmouth 786  
 kABCHouse5Axis 786  
 kABCHouseRotated 786  
 kABCHouseTilted 786  
 kABC Housing 786  
 kABC Housing Throughout 786  
 kABCJapaneseHipCut 786  
 kABC Lap Joint 786  
 kABC Open Diagonal Seat Cut 786  
 kABC Open Seat Cut 786  
 kABC Rabbet 786  
 kABC Reversed Birdsmouth 786  
 kABC Rising Birdsmouth 786  
 kABC Rising Seat Cut 786  
 kABC Seat Cut 786  
 kABC Simple Housing 786  
 kABC Valley Birdsmouth 786  
 kACCCompound 806  
 kACCP5Axis 803  
 kACCPConcave 803  
 kACCPConvex 803  
 kACCPPerpendicular 803  
 kACHExposed 793  
 kACHip 806  
 kACHOthersExposed 793  
 kACL5Axis 796  
 kACLPPerpendicular 796  
 kACP5Axis 799  
 kACPPerpendicular 806  
 kACPPPerpendicular 799

- kACSimpleAngled 806  
kACSimpleBeveled 806  
kAD5Axis 820  
kADCHead 811  
kADCSide 811  
kADCSimpleCut 811  
kADCValley 811  
kADHead 820  
kADN5Axis 809  
kADNPerpendicular 809  
kADo5Axis 814  
kADoHeadCompound 814  
kADoHeadPerpendicular 814  
kADoHeadSimpleAngled 814  
kADoHeadSimpleAngledTwisted 814  
kADoHeadSimpleBeveled 814  
kADoPerpendicular 814  
kADoRotated 814  
kADoTenonCompound 814  
kADoTenonPerpendicular 814  
kADoTenonSimpleAngled 814  
kADoTenonSimpleAngledTwisted 814  
kADoTenonSimpleBeveled 814  
kADoTilted 814  
kADPerpendicular 820  
kADRotated 820  
kADSFemale5Axis 817  
kADSFemalePerpendicular 817  
kADSMale5Axis 817  
kADSMalePerpendicular 817  
kADTilted 820  
kAfp5Axis 823, 857  
kAfpCenter 823, 857  
kAfpLeft 823, 857  
kAfpPerpendicular 823, 857  
kAfpRight 823, 857  
kAH5Axis 829  
kAHHeadCompound 829  
kAHHeadPerpendicular 829  
kAHHeadSimpleAngled 829  
kAHHeadSimpleAngledTwisted 829  
kAHHeadSimpleBeveled 829  
kAHPerpendicular 829  
kAHRotated 829  
kAHSimple 829  
kAHTenonCompound 829  
kAHTenonPerpendicular 829  
kAHTenonSimpleAngled 829  
kAHTenonSimpleAngledTwisted 829  
kAHTenonSimpleBeveled 829  
kAHTilted 829  
kAKSFemale5Axis 833  
kAKSFemalePerpendicular 833  
kAKSMale5Axis 833  
kAKSMalePerpendicular 833  
kALD5Axis 836  
kALDPerpendicular 836  
kAllBeams 195  
kAllSpaces 284  
kALN5Axis 839  
kALNPerpendicular 839  
kAM5Axis 848  
Kamatsugi 725  
kAMHeadCompound 848  
kAMHeadPerpendicular 848  
kAMHeadSimpleAngled 848  
kAMHeadSimpleAngledTwisted 848  
kAMHeadSimpleBeveled 848  
kAMLInvisible 845  
kAMLSHadow 845  
kAMLVisible 845  
kAMNone 842  
kAMPerpendicular 848  
kAMRotated 848  
kAMShadow 842  
kAMSPerpendicular 875  
kAMTenonCompound 848  
kAMTenonPerpendicular 848  
kAMTenonSimpleAngled 848  
kAMTenonSimpleAngledTwisted 848  
kAMTenonSimpleBeveled 848  
kAMTilted 848  
kAMVisible 842  
kAnalysedTools 777  
kAngle 87, 181, 199  
kAOPAdditive 607  
kAOPIIntersect 607  
kAOPSubtractive 607  
kAPH5Axis 852  
kAPHPerpendicular 852  
kAPIExposed 855  
kAPIOthersExposed 855

kArea 87, 181, 199  
kASJ5Axis 863  
kASJPerpendicular 863  
kASI5Axis 872  
kASIPerpendicular 872  
kASIRotated 872  
kASITilted 872  
kASM5Axis 875  
kASMHeadCompound 875  
kASMHeadPerpendicular 875  
kASMHeadSimpleBeveled 875  
kASMRotated 875  
kASMTilted 875  
kASS5Axis 869  
kAssembly 584  
kASSPerpendicular 869  
kAST5Axis 866  
kASTPerpendicular 866  
kASTRotated 866  
kASTTenonCompound 866  
kASTTenonPerpendicular 866  
kASTTenonSimpleAngled 866  
kASTTenonSimpleBeveled 866  
kASTTilted 866  
kAuto 195  
kBackward 730, 839  
kBeam0 195  
kBeam01 195  
kBespoke 263  
kBottom 713, 735, 842  
kBT3dSolid 199  
kBTMassElement 199  
kBTSubDMesh 199  
kBTSurface 199  
kCancel 967  
kCCLeftHandShort 512  
kCCLogWallCrossing 512  
kCCMitre 512  
kCCMitreOpenEnd 512  
kCCNoCornerCleanup 512  
kCCConcave 773  
kCCConvex 773  
kCCRRightHandShort 512  
kCCSipOverlap 512  
kCCStretchLast 512  
kCCStretchThis 512  
kCDiagonal 773  
kCenter 713, 735, 754, 842  
kCentered 730, 839  
kChamfer45 793  
kChamfer90 793  
kChamferRound 793  
kCompleted 679, 799  
kContext 65  
kCtrlKeyPressed 65  
kCurrentGroup 284  
kDevice 1006  
kDeviceX 1006  
kDoor 584  
kDrawAsCurves 1006  
kDrawAsShell 1006  
kDrawFilled 1006  
kErase 262  
kExecuteKey 65, 169, 218, 897, 972  
kExplicitRadius 746  
kExtrProfRectangular 321  
kExtrProfRound 321  
keyAt 199  
key-value 199  
keywordIndex 967  
kFemaleEnd 696, 771  
kFemaleSide 696, 722, 771  
kFingerMill 754, 823, 857  
kFixedSize 199  
kForward 730, 839  
kGripPointDrag 169  
kGSTAcad 255  
kGSTArrow 255  
kGSTCircle 255  
kGSTDiamond 255  
kGSTLine 255  
kGSTMinus 255  
kGSTPlus 255  
kGSTRotate 255  
kGSTSquare 255  
kGSTSquare45 255  
kGSTStar 255  
kGSTTriangle 255  
kGSTTriangleIso 255  
kHeaderRecess 476, 748, 945  
kIsGray 284  
kIsNotDefined 284

kIsOff 284  
kIsOn 284  
kIsotropic 423  
kKeyword 967  
kLeft 713, 735, 754, 842  
kLength 87, 181, 199  
kLetIn 476, 748, 945  
kList 199  
kLLDebug 225  
kLLError 225  
kLLFatal 225  
kLLInfo 225  
kLLNoLog 225  
kLLTrace 225  
kLLWarn 225  
kLogMessages 225  
kMaleEnd 696, 722  
kMillWithout 679, 799  
kModel 1006  
kModelSpace 284  
kMPDoNotAllowMirror 296  
kMPUseXYPlane 296  
kMPUseYZPlane 296  
kMPUseZXPlane 296  
kmSTArch 604  
kmSTBarrelVault 604  
kmSTBox 604  
kmSTBrep 604  
kmSTCone 604  
kmSTCylinder 604  
kmSTDome 604  
kmSTDoric 604  
kmSTExtrusion 604  
kmSTGable 604  
kmSTIsocTriangle 604  
kmSTPyramid 604  
kmSTRevolution 604  
kmSTRightTriangle 604  
kmSTSphere 604  
kMultiPlexIsotropic 423  
kMySpace 284  
kNameLastChangedProp 181  
kNoBeams 195  
kNoEdgeRecess 476, 748, 945  
kNone 967  
kNonIsotropic 423  
kNoOpening 584  
kNotRound 746  
kNoUnit 87, 181, 199  
kNTAutoNesterV4 1066  
kNTAutoNesterV5 1066  
kNTAutoNesterV6 1066  
kNTRectangularNester 1066  
kNTTestNester 1066  
kNYNZ 674  
kNYPZ 674  
kObject 199  
kOCTChildPanel 343  
kOCTCollectionEntity 343  
kOCTElement 343  
kOCTGroup 343  
kOCTIndividualShopDrawing 343  
kOCTMassElement 343  
kOCTMasterPanel 343  
kOCTMetalPartCollectionEnt 343  
kOCTModuleElementDrawing 343  
kOCTSSubassembly 343  
kOCTTrussEntity 343  
kOCTTslDefined 343  
kOk 967  
kOnGenerateShopDrawing 888, 891, 897  
kOnlyBrdMth 679, 799  
kOpening 584  
kOsModeCen 967  
kOsModeCentroid 967  
kOsModeEnd 967  
kOsModelns 967  
kOsModeMid 967  
kOsModeNear 967  
kOsModeNode 967  
kOsModePerp 967  
kOsModeQuad 967  
kOsModeTan 967  
kPathAppData 214, 217  
kPathCurrentDir 199, 214, 217  
kPathDwg 214, 217  
kPathHsbCompany 214, 217  
kPathHsbInstall 214, 217, 1073  
kPathHsbRoofDetail 214, 217  
kPathHsbWallDetail 214, 217  
kPathPersonal 214, 217  
kPathPersonalTemp 214, 217

kPathProgramFiles 214, 217  
kPathRoamableRoot 214, 217  
kPathWindowsInstall 214, 217  
kPointInProfile 124  
kPointOnRing 124  
kPointOutsideProfile 124  
kPYNZ 674  
kPYPZ 674  
kRCCollectionEntity 886  
kRCElement 886  
kRCElementModule 886  
kRCEntity 886  
kRCGroup 886  
kRCSubassembly 886  
kRCTool 886  
kRCToolsFromETool 886  
kRCTrussEntity 886  
kRight 713, 735, 754, 842  
kRound 746  
kRounded 746  
kRPTCeiling 561  
KRPTFloor 561  
KRPTRoof 561  
kRTBlock 398  
kRTBuildIn 398  
kRTTsl 398  
kRTUser 398  
kRunTestsEvent 247  
kScalable 199  
kShiftKeyPressed 65  
kShopDrawChildPageArea 892, 893  
kShopDrawChildPageCreate 892, 895  
kShopDrawEvent 892, 893, 895  
kShopDrawTemplateRule 908  
kShopDrawViewDataShowSet 897  
kSimpleModel 263  
kSpline2xLumber 476, 748, 945  
kSplineBlock 476, 748, 945  
kSplineCustom 476, 748, 945  
kSplineDouble 476, 748, 945  
kSplineJoist 476, 748, 945  
kSplineSingleLumber 476, 748, 945  
kSplineSingleTopSkin 476, 748, 945  
kSplitAboveOpening 945  
kSplitBelowOpening 945  
kSTDigits 510  
kSTEExcelColumn 510  
kSTODCharAABB 510  
kStraight 313  
kTEConcave 773  
kTEConvex 773  
kTEDiagonal 773  
kTELTCeiling 653  
kTELTLast 653  
kTELTRoof 653  
kTELTRoofOverhang 653  
kTELTUdefined 653  
KTENTBearing 653  
KTENTGeneric 653  
KTENTLast 653  
KTENTUndefined 653  
kTESGenerated 653  
kTESInvalid 653  
kTESLast 653  
kTESUndefined 653  
kTESValid 653  
kTETGirder 653  
KTETHipValley 653  
KTETStandard 653  
kTETUndefined 653  
kThread 263  
kTIConcave 773  
kTIConvex 773  
kTIDiagonal 773  
kTileBottomOpening 566  
kTileEave 566  
kTileEdgeTooSmall 566  
kTileHip 566  
kTileHorizontalPlaneEdge 566  
kTileLeft 566  
kTileLeftOpening 566  
kTileLeftSloped 566  
kTileNotSet 566  
kTileOther 566  
kTileRidge 566  
kTileRight 566  
kTileRightOpening 566  
kTileRightSloped 566  
kTileTopOpening 566  
kTileValley 566  
kTool 199  
kTop 713, 735, 842

kTOXN 842  
 kTOXP 842  
 kTOYN 842  
 kTOYP 842  
 kTTAllTools 928  
 kTTDrill 928  
 kTTItem 928  
 kTTMarker 928  
 kTTMill 928  
 kTTNail 928  
 kTTNailCluster 928  
 kTTSaw 928  
 kUndefinedIsotropic 423  
 kUniversalMill 754, 823, 857  
 kVersion2000 1006  
 kVersion2004 1006  
 kVersion2007 1006  
 kVersion2010 1006  
 kVersionCurrent 1006  
 kVerticalFingerMill 754, 823, 857  
 kViewDataSets 888, 897, 1057  
 kVolume 87, 181, 199  
 kWindow 584  
 kWithUF 679, 799  
 kXN 151  
 kXP 151  
 kYN 151  
 kYP 151  
 kZN 151  
 kZP 151

## - L -

label 423, 637  
 lamExtraHeightRaw 313  
 lamExtraLength 313  
 lamExtraWidthRaw 313  
 lamGrading 313  
 lamGroups 313  
 lamMinimumLength 313  
 lamSizeAt 313  
 lamThickness 313  
 last 80  
 layer 423, 1006  
 layerName 371  
 Layout 291, 635

left 87  
 length 80, 87, 103, 108, 114, 199, 364  
 lengthA 545  
 lengthB 545  
 lengthInDirection 132  
 lengthSelectionIsAutomatic 571  
 Line 110  
 lineAt 653  
 LineBeamIntersect 146  
 LineSeg 108  
 lineType 1006  
 LineTypes 921, 985, 1006  
 lineWeight 371  
 linkedEntity 398  
 listComponent 324  
 InEdged 151  
 loadBearing 423, 524  
 loadProfiles 364  
 location 936  
 lock 478  
 lockDatabase 253  
 lockDatabaseOf 253  
 LockDocument 1085  
 LogCourse 518, 522  
 logCourses 518, 522  
 Logical operations 84  
 LogNotch 730  
 lowerExtension 545  
 lumberInstallList 465

## - M -

machinePathOnly 823  
 mainDiameter 329  
 makeFolder 214, 217  
 makeLower 87  
 makeUpper 87  
 manufacturer 329, 398  
 map 199, 218, 236, 330, 333, 404, 678, 948  
 mapInternal 777  
 MAPMM 218  
 MapObject 333  
 mapWithPropValues 404, 883, 972  
 mapWithPropValuesFromCatalog 404  
 Mark 735  
 MarkerLine 738

markFlippedChildPanels 357  
 MassElement 604  
 MassGroup 607  
 master 981  
 MasterPanel 230, 610  
 MasterPanelStyle 341  
 material 160, 178, 329, 360, 398, 423, 496  
 materialDescription 404  
 Maths on double and int 83  
 maximumHeight 512  
 maxProjectionLength 330  
 MetalPart 165  
 MetalPartCollectionDef 303  
 MetalPartCollectionEnt 617  
 Metal TKey 740  
 mid 87  
 midCurve 313  
 MillAround 719, 746  
 millDiameter 705, 823  
 millSide 705  
 minimumSpacing 1066  
 minProjectionLength 330  
 mirrorPlane 296  
 mirrorStyle 296  
 mm2GermanDimensionFormat 87  
 mod 83  
 model 160, 178, 329, 398  
 modelDescription 404  
 modeless 1085  
 ModelMap 236  
 ModelMapComposeSettings 236, 1077  
 ModelMapInterpretSettings 236, 1077  
 modelToSection 645  
 modelToView 622  
 ModelX 236, 243, 244  
 ModelXComposeSettings 236, 243  
 ModelXInterpretSettings 236, 244  
 modifyBoxUntilFaceCompletelyOut 151  
 modifySectionForCnC 672  
 module 423  
 Mortise 746  
     Example of Slot and Mortise 1094  
 moveGripEdgeMidPointAt 124  
 moveGripVertexPointAt 124  
 moveLastTo 199  
 MultiPage 619, 622  
 MultiPageStyle 343  
 MultiPageView 619, 622  
 mustCut 685  
 MvBlockDef 345, 625  
 MvBlockRef 625  
 myCncCurveEnts 610  
 MYPOS 735  
 myRoofplane 566  
 myZoneIndex 371

**- N -**

NailLine 478, 630  
 name 181, 255, 284, 329, 352, 360, 398, 423, 545, 569, 578, 610, 635  
 nameFromType 566  
 namePart 284  
 nCncMode 823, 857  
 nCutsN 443  
 nCutsP 443  
 nEquivalentStory 284  
 nestedChildPanels 610  
 NesterChild 1065  
 NesterData 1066  
 NesterMaster 1066  
 nesterNameFromType 1066  
 nesterToUse 1066  
 nestInOpenings 1065  
 NETLOAD 1076  
 nExtraRaftersLeft 599  
 nExtraRaftersRight 599  
 nFeed 793  
 nFloorsToCarry 595  
 nLamIndexRefPt2 313  
 nMillSide 823, 857  
 nNumPoints 146  
 nodeAt 653  
 noNailProfile 478  
 norm 329  
 normal 103, 110, 560, 685, 806  
 normalize  
     Point and Vector Manipulation 103  
 normalizeVectors 151  
 not 84  
 Notepad 1128  
 Notepad++ 1128

notes 330, 371, 398  
 nOverShoot 860  
 nPreMillType 799  
 nProfileType 803  
 nQtyCenter 945  
 nQtyLeft 945  
 nQtyRight 945  
 nRoundType 829, 848, 852  
 nShoulderType 839  
 nTextOrientation 1035  
 nToolIndex 783, 875  
 nTopBottomProcessing 836  
 numBeamsNoSupport 595  
 numBeamsNoSupportRight 595  
 numBeamsSupport 595  
 numBeamsSupportRight 595  
 number 478, 510, 538, 610, 635  
 numHardWrComps 395, 398  
 numLams 313  
 numLines 653  
 numNodes 653  
 numRefDocs 371  
 numRings 124  
 numSipComponents 357  
 nZoneIndex 188, 371

**- O -**

objectCollectionType 343  
 offset 114, 637  
 OnCheckConRepair0 988  
 OnCheckConRepair1 988  
 OnCheckSubAssembly0 988  
 OnCheckSubAssembly1 988  
 onCommandEnded 1085  
 OnDbCreated 988  
 OnDebug 988  
 OnElementConstructed 988  
 OnElementDeleted 988  
 OnElementRecalc 988  
 OnGenerateShopDrawing 988  
 OnInsert 988  
 OnMapIO 883, 988  
 OnOptionChanged 988  
 OnRecalc 988  
 OnViewportsSetInLayout 988

openFromFile 270  
 Opening 478, 584, 595  
 openingAngle 681  
 openingCount 610  
 openingDescr 592, 595  
 OpeningLog 592  
 OpeningRoof 599  
 OpeningSF 595  
 openingType 584  
 openNew 270  
 operator- 132  
 operator+ 132  
 operator+= 132  
 operator-= 132  
 OPM 181  
 OPM functions 181  
 opmName 404  
 optionAssignSameRuleSet 371  
 optionEvaluate 233, 371  
 options 233  
 or 84  
 orderPoints 110  
 originalName 276  
 originator 404  
 originatorId 1065, 1066  
 OutlineEdgeInfo 252  
 overShoot 930, 933  
 oversizeQuader 364

**- P -**

pageIndex 622  
 PainterDefinition 352  
 PanelSplit 945  
 PanelStop 748  
 PanelWirechase 750  
 panhand 423  
 paperLowerLeft 291  
 paperSizeX 291  
 paperSizeY 291  
 parentEntity 584  
 ParHouse 752  
 partnerRoofPlane 566  
 path 217  
 Performance 245  
 Plane 110

PlaneProfile 124, 1006  
planeType 561  
plBevel 857  
plCurve 799  
plCuts 705, 823  
plDefining 823, 857  
plEdge 476  
plEnvelope 459, 465, 478, 561, 569  
plEnvelopeCnc 465, 542  
plFaceD 151  
PLine 114, 561  
pline1 155  
pline2 155  
plInternal 705  
plOpeningAt 610  
plOpenings 459, 465  
plotRotation 291  
PlotViewport 635  
plOutlineWall 478  
plPath 630, 640  
plShadowCnc 465  
plShape 584, 599, 610, 622, 926, 928, 930, 933  
plWallBoundary 561  
Point 79  
    Point and Vector manipulation 103  
Point and Vector manipulation 103  
Point3d 79  
pointInProfile 124  
position 653  
posnum 393, 404, 423, 607  
posnumandtext 423  
posnumCriteria 160  
posnumLocation 645  
precision 96  
Predefined variables E-connection 1001  
Predefined variables G-connection 995  
Predefined variables T-connections 991  
prefix 510  
PrEntity 961  
PressEnt 554, 637  
presses 554  
processRecalcEntitiesNow 1085  
profBrutto 478  
profCncNesting 465  
profile 423  
profiling 245  
profNetto 478  
profShape 459, 1065, 1066  
project 124  
    Point and Vector Manipulation 103  
projectCity 160  
projectComment 160  
projectLineSegs 110  
 projectName 160  
projectNumber 160  
projectPoint 103  
projectPoints 110  
projectPointsToPlane 114  
projectRevision 160  
projectSpecial 160  
projectStreet 160  
projectUser 160  
projectVector 103  
propDouble 181, 404  
propDoubleName 404  
PropellerSurface 155  
PropellerSurfaceTool 754  
propInt 181, 404  
propIntName 404  
propString 181, 404  
propStringName 404  
PrPoint 967  
pt1 146, 502, 793, 845, 855  
pt2 146, 502, 793, 845, 855  
ptArrow 512  
ptBottom 637  
ptCen 132, 423, 560  
ptCenPS 1049, 1057  
ptCenSolid 423  
ptCenter 681  
ptCentreOfGravity 423  
ptCut 799  
ptEnd 108, 114, 402, 476, 653, 657, 698  
ptEndExtreme 820  
ptEndOutline 512  
PtG 169  
PtG0 169  
PtG1 169  
PtG2 169  
ptInsert 637  
ptLoc 255  
ptMid 108, 114, 124, 476

ptNode 532  
 ptNodeOther 532  
 ptOnSurface 829, 848, 872  
 ptOrg 110, 148, 404, 478, 496, 502, 685, 780, 783, 786, 799, 803, 806, 809, 811, 814, 817, 823, 829, 833, 836, 839, 842, 848, 852, 857, 860, 863, 866, 869, 872, 875  
 ptRef 313, 423, 610  
 ptStart 108, 114, 402, 476, 653, 657, 698, 820  
 ptStartExtreme 820  
 ptStartOutline 512  
 ptTop 637  
 pushCommandOnCommandStack 160, 196, 246

## - Q -

qrDecode 87  
 qrEncode 87  
 quader 151, 364, 423, 672, 786, 829, 848, 872  
 quality 361  
 quantity 398, 478  
 questionIsCompoundCut 806

## - R -

Rabbit 758  
 radius 560, 604  
 rayIntersection 132  
 readFromDxxFile 236  
 readFromXmlFile 199, 217  
 readTextFile 214  
 realBody 371  
 realBodyOfComponentAt 465  
 realBodyTry 371  
 recalc 404  
 recalcAllReferencesToDictionary 333  
 recalcNow 404  
 recessType 476  
 reconstructArcs 114  
 refDocs 371  
 referenceFace 443  
 refPositionHost 655  
 refPositionOther 655  
 regenerate 619  
 releaseParentEntity 584  
 releasePosnum 404, 423

releasePosnums 607  
 removeAllOpeningRings 124  
 removeAllRings 124  
 removeAt 80, 199  
 removeCatalogEntry 404  
 removeEmptyFolder 214  
 removeGenBeamConnection 395  
 removeHsbData 371  
 removeLumberFromInstallList 465  
 removeOpeningAt 610  
 removePropSet 371  
 removeSubMap 181, 371  
 removeSubMapX 291, 371, 398, 664  
 removeSubMapXProject 160  
 removeToolsStatic 423  
 removeToolsStaticOfType 423  
 removeToolStatic 423  
 renamePropertiesInSet 371  
 renderMaterial 354, 371  
 replace 87  
 reportError 160, 187  
 reportMessage 87, 160, 187  
 reportNotice 160  
 reportWarning 160, 187  
 repType 398  
 Reset OPM 65  
 resetFastenerGuidelines 395, 402  
 resolveElementsByNumber 244  
 resolveEntitiesByHandle 244  
 resolveHandlesAsUniqueIds 244  
 resolveSelfIntersect 132  
 return 82, 187  
 reverse 80, 114  
 RevolutionMill 760  
 rgb 160, 371, 1006  
 rgbB 160  
 rgbG 160  
 rgbR 160  
 right 87  
 ringIsOpening 124  
 rise 584, 604  
 roofCodeExpression 512  
 roofNumber 561  
 RoofOverhang 225  
 rotateBy 103  
 rotationAllowance 1065

RoundWoodMill 765  
runCorrector 561

## - S -

SawLine 478, 640  
scale 255  
scaleGroup 1057  
ScaleType 199  
ScarfJoint 766  
ScriptInsert 218  
ScriptInsertNoPrompt 218  
scriptName 160, 404  
seConstrDetailRight 524, 595  
seConstrDetailShimRight 595  
seConstrSheetingRight 595  
Section2d 645  
segEndOutline 512  
segment 566  
segmentMinMax 478, 538  
segStartOutline 512  
select dialog 60  
Select Metalpart/Tools description 60  
selection 75, 76  
sequenceNumber 404  
sequenceType 510  
set 181, 961  
setAdditionalNotes 404  
setAddKeyhole 696  
setAllowedOversize 364  
setAllowedRunTimeSeconds 1066  
setAllowFlippingForComparison 357  
setAllowGripAtPt0 404  
setAllowRotationForComparison 357  
setAmountOfDigits 510  
setAngle 679, 930, 951, 1061  
setAngleA 545  
setAngleB 545  
setAreaNoSheet 478, 505  
setAreaStud 478, 505  
setArticleData 331  
setArticleDataSet 331  
setArticleNumber 330, 398, 571  
setAttachedPropSetFromMap 371  
setAttributesFromMap 538  
setAutoDimInfo 664  
setAutoErase 502  
setBAdjustPositionToTileLath 599  
setBAllowBevels 360  
setBApplyNonSplittingTooling 592  
setBaseHeight 657  
setBCenterStudsHaveFullPanelWidth 945  
setBCutHalfLogBottom 518  
setBCutHalfLogTop 518  
setBeamCode 423  
setBeamDistribution 527  
setBeamTypeNameAndColorFromHsBld 244  
setBeamWidth 313  
setBExtraBeamsBottom 595  
setBExtraBeamsTop 595  
setBHeaderBelowPlate 595  
setBIsDefault 532  
setBIsDeliverableContainer 284  
setBIsDummy 423  
setBIsFlipped 542  
setBlockingRun 478, 505, 508  
setBMeasureInLogs 592  
setBody 199  
setBStoringRoughDimensions 584  
setBSymmetrical 592  
setBVisualize 398  
setCatalogFromPropValues 404, 972  
setCategory 329, 398  
setChildOffsetX 1066  
setChildOffsetY 1066  
setChildPanelFlipPrefix 357  
setCloneDuringBeamSplit 195  
setCncMode 705, 754  
setCncSplinterFree 443  
setCoating 329  
setCode 496, 502, 561  
setColor 255, 371, 496  
setCompareKey 160, 177  
setComponentData 331  
setComponentToExcludeFromOpenings 357  
setConnectionTypeName 655  
setConstrDetail 532, 595  
setConstrDetailBottom 524, 595, 599  
setConstrDetailBottomAngled 599  
setConstrDetailHip 527  
setConstrDetailKneeWall 527  
setConstrDetailKneeWall2 527

setConstrDetailLeft 524, 527, 595, 599  
setConstrDetailOverhang 527  
setConstrDetailRidge 527  
setConstrDetailRight 527, 599  
setConstrDetailShimBottom 595  
setConstrDetailShimLeft 595  
setConstrDetailShimTop 595  
setConstrDetailStrut 527  
setConstrDetailStrut2 527  
setConstrDetailTop 524, 595, 599  
setConstrDetailTopAngled 599  
setConstrDetailTopLeft 524  
setConstrDetailTopRight 524  
setConstrDetailValley 527  
setConstrDetailWallPlate 527  
setConstrDetailWallPlate2 527  
setConstrSheetingBottom 595  
setConstrSheetingLeft 595  
setConstrSheetingTop 595  
setContent 296  
setContinueSheeting 595  
setContinuousMortise 696  
setContourPtsLeft 518  
setContourPtsRight 518  
setControlsOtherProperties 181  
setCoordSys 393, 443, 625, 1057  
setCoordSysOnly 607  
setCounterLathDistribution 527  
setCountType 398  
setCurrent 1049  
setCurrentLayout 291  
setCurvedStyle 443  
setCurveParam 637  
setCutBottomBeam 595  
setCutDefiningAsOne 705  
setCutLocation 505  
setCuttingCurveOffset 313  
setCuttingType 512  
setD 443  
setDAngleA 398  
setDAngleB 398  
setDAngleG 398  
setDBeam 595  
setDBeamRight 595  
setDBeamSpacing 527  
setDCounterLathHeight 527  
setDCounterLathSpacing 527  
setDCounterLathWidth 527  
setDDepthLeft 592  
setDDepthRight 592  
setDDistanceToFirstTileLath 599  
setDebug 404  
setDefineSet 619  
setDefinesFormatting 181  
setDefinition 538, 550, 571, 617, 625, 650  
setDefinitionObject 650  
setDeltaDistribution 524  
setDeltaOnTop 921, 1021  
setDeltaTextX 918  
setDeltaTextY 918  
setDependencyOnBeamLength 160, 193  
setDependencyOnDictObject 160, 293  
setDependencyOnEntity 160, 193  
setDependencyOnEntityPropSet 193, 371  
setDependencyOnPosnumChanged 177, 404  
setDepth 604, 705, 930, 933, 936  
setDepthCorrection 696  
setDescription 181, 313, 324, 330, 398, 569, 584, 599, 625, 648, 657  
setDescrPacking 595  
setDescrPlate 595  
setDescSF 595  
setDExtraPlateBottom 595  
setDExtraPlateTop 595  
setDFirstLog 518  
setDFloorHeight 284  
setDGap 518, 532, 945  
setDGapBottom 592, 595  
setDGapLeft 592  
setDGapRight 592  
setDGapSide 595  
setDGapTop 592, 595  
setDGroove 518  
setDH 459, 496, 502  
setDiameter 936  
setDimColor 360  
setDimensionPlane 1028, 1030, 1033  
setDimScale 1028, 1030, 1033  
setDimStyle 918, 921  
setDirection 936  
setDisplay 364  
setDistributionReferences 505

setDistributionStudLocation 478, 505  
setDistributionType 524  
setDistributionZone 524  
setDKneeWallHeight 527  
setDKneeWallHeight2 527  
setDLateralCutOutAngle 592  
setDLateralCutOutWidth 592  
setDMillExtraLengthBottom 592  
setDMillExtraLengthTop 592  
setDMillHeightBottom 592  
setDMillHeightTop 592  
setDMillWidthBottom 592  
setDMillWidthTop 592  
setDNonSuppBeamLeft 595  
setDNonSuppBeamRight 595  
setDOffsetHorizontalMillings 592  
setDOffsetLateralMillings 592  
setDOffsetX 398  
setDOffsetY 398  
setDOffsetZ 398  
setDOpeningHeight 599  
setDoSolid 664  
setDouble 199  
setDPackingMaxHeight 595  
setDPlate 595  
setDPosZOutlineBack 478  
setDPosZOutlineFront 478  
setDrawingPropertiesFromMap 270  
setDrawOrderToFront 371  
setDryJointLamColor 313  
setDryJointLamIndex 313  
setDScaleX 398  
setDScaleY 398  
setDScaleZ 398  
setDShiftHeaderInTopPlate 595  
setDSideBeamBottom 595  
setDSideBeamTop 595  
setDSillPlateTop 595  
setDSillPlateTopDown 595  
setDSillPlateTopUp 595  
setDSplineHeight 945  
setDSplineWidth 945  
setDStrutHeight 527  
setDStrutHeight2 527  
setDThickness 360  
setDThicknessLeft 592  
setDThicknessRight 592  
setDTongue 518  
setDTopHeight 599  
setDVar 496  
setDWallPlateHeight 527  
setDWallPlateHeight2 527  
setDWidthCenter 945  
setDWidthLeft 592  
setDWidthRight 592  
setDWidthSteg 945  
setDxContent 199  
setDXMax 533  
setDYMax 533  
setEdgeDetailCode 748  
setEdgeDetailCodeLeft 945  
setEdgeDetailCodeRight 945  
setEdgeRecessType 748, 945  
setElemRoofEdges 527, 532  
setElemTexts 478, 502  
setElemTool 640  
setEndDepth 679  
setEndType 696, 719, 722, 725, 746, 771  
setEntities 236  
setEntity 199  
setEntityArray 199  
setEnumValues 181  
setEraseAndCopyWithBeams 195  
setExclusionEntities 545  
setExecutionLoops 160, 193  
setExplicitRadius 746  
setExposed 512  
setExtraTopNeckWidth 725  
setExtrProfile 443, 518, 943, 945  
setFastenerLength 330  
setFeed 674  
setFlatWidth 696, 722, 725  
setFloorGroup 478  
setForceLevelDetail 524  
setFormat 181  
setFreeDirectionsFromRealSolid 672  
setFromComposed 510  
setFromHandle 284  
setFromMap 664  
setFrontCutPerpToRoof 561  
setFrozen 395  
setGenerateDebugOutput 1066

setGlueDensity 313  
setGrade 329, 423, 943  
setGrips 255, 404  
setGroup 329, 398  
setHardWrComps 395, 398  
setHasLengthInfo 330  
setHatch 922  
setHatchHidden 443  
setHatchSection 443  
setHeadComponents 324  
setHeadHeight 584  
setHeight 364, 545, 584, 604  
setHeightPS 1049, 1057  
setHoldingDistance 571  
setHsBld 423  
setHsBShareProjectId 160  
setHyperlink 371  
setIdRef 502  
setIncludedEntities 545  
setIndexRef 502  
setInfo 943  
setInformation 423, 478, 610  
setInnerCylinderDiameter 698  
setInstanceWidth 657  
setInsulation 527  
setInt 199  
setIsActive 657  
setIsBespoke 330  
setIsChainDimReferencePoint 916  
setIsEndTool 682  
setIsFemale 668  
setIsotropic 423  
setIsRelativeToEcs 255  
setIsStretchPoint 255  
setIsVisible 371  
setJapaneseMarking 664  
setJsonContent 199  
setKeepReferenceToGenBeamDuringCopy 195  
setLabel 423, 637, 943  
setLamExtraHeightRaw 313  
setLamExtraLength 313  
setLamExtraWidthRaw 313  
setLamGrading 313  
setLamGroups 313  
setLamMinimumLength 313  
setLamThickness 313  
setLayout 635  
setLength 80, 364, 679, 716  
setLengthA 545  
setLengthB 545  
setLengthSelectionIsAutomatic 571  
setLineType 921  
setLineWeight 371  
setLinkedEntity 398  
setListComponent 324  
setLoadBearing 524  
setLoadProfiles 364  
setLocation 936  
setLock 478  
setLowerExtension 545  
setMachinePathOnly 705  
setMainDiameter 329  
setManufacturer 329, 398  
setMap 181, 199, 236, 330, 333, 404  
setMapKey 199  
setMapName 199  
setMarbleDiameter 160  
setMarkFlippedChildPanels 357  
setMaterial 329, 360, 398, 423, 496, 943  
setMaterialDescription 404  
setMaxDepth 679  
setMaximumDeviation 754  
setMaximumHeight 512  
setMaxProjectionLength 330  
setMillAround 719, 746  
setMillDiameter 754  
setMillSide 754  
setMillSidePolyNormal 705  
setMinDepth 679  
setMinimumOffsetFromDimLine 917, 921  
setMinimumSpacing 1066  
setMinProjectionLength 330  
setMirrorPlane 296  
setMirrorStyle 296  
setModel 329, 398  
setModelDescription 404  
setModifySectionForCnC 672  
setModule 423  
setMustCut 685  
setName 255, 284, 329, 360, 398, 423, 545, 569, 578, 610, 635, 943  
setNamePart 284

setNEquivalentStory 284  
setNesterToUse 1066  
setNestInOpenings 1065  
setNExtraRaftersLeft 599  
setNExtraRaftersRight 599  
setNFloorsToCarry 595  
setNodeTextCumm 916  
setNodeTextDelta 916  
setNorm 329  
setNormal 114, 560  
setNotes 330, 371, 398  
setNQtyCenter 945  
setNQtyLeft 945  
setNQtyRight 945  
setNumBeamsNoSupport 595  
setNumBeamsNoSupportRight 595  
setNumBeamsSupport 595  
setNumBeamsSupportRight 595  
setNumber 478, 510, 610, 635  
setOffset 637  
setOpening 945  
setOpeningAngle 681  
setOpeningDescr 592, 595  
setOpeningOverwrite 478, 505, 518  
setOPMKey 160, 181  
setOriginator 404  
setOriginatorId 1065, 1066  
setOverShoot 687, 758, 930, 933  
setPanhand 423  
setParentEntity 584  
setPlaneType 561  
setPIEnvelope 459  
setPLine 199, 558  
setPIPPath 630, 640  
setPIShape 599, 610, 926, 930, 933  
setPoint3d 199  
setPoint3dArray 199  
setPoint3dXYArray 199  
setPosnum 393  
setPosnumBeam 735  
setPosnumCompareVertices 664, 705  
setPosnumCriteria 160  
setPrefix 510  
setPreMillType 679  
setProfCncNesting 465  
setProfNetto 478  
setProfShape 1065, 1066  
setProjectCity 160  
setProjectComment 160  
setProjectName 160  
setProjectNumber 160  
setProjectRevision 160  
setProjectSpecial 160  
setProjectStreet 160  
setProjectUser 160  
setPropDouble 404  
setPropInt 404  
setPropString 404  
setPropValuesFromCatalog 65, 404, 527, 972  
setPropValuesFromMap 404, 883, 972  
setPt1 502  
setPt2 502  
setPtArrow 512  
setPtCen 560  
setPtCenPS 1049, 1057  
setPtCtrl 423  
setPtEndOutline 512  
setPtInsert 313  
setPtLoc 255  
setPtNode 532  
setPtNodeOther 532  
setPtOrg 404, 502  
setPtRef 610  
setPtStartOutline 512  
setQuantity 398, 478  
setRadius 560, 604, 681, 698  
setRangeNoDistributionStud 478, 505  
setRangeSheetGap 505  
setReadDirection 1021  
setReadOnly 181  
setReducedTipWidth 725  
setRefDocs 371  
setReferenceFace 443  
setRenderMaterial 354, 371  
setRenderMaterialMapping 354  
setRepType 398  
setRise 584, 604  
setRoofCodeExpression 512  
setRoofEdgeDirective 505  
setRoofNumber 561  
setRotationAllowance 1065  
setRoundType 719, 746, 752, 763

setScale 255, 538, 625  
setScaleGroup 1057  
setScriptName 404  
setSequenceNumber 404  
setSequenceType 510  
setShapeType 255  
setSheetCutLocation 478, 505  
 setShow 940  
setShowChildPanelFlipMark 357  
setShowElementTools 404  
setShowLeaderLine 917  
setShowSetDefineEntities 1057  
setShowSetEntities 1057  
setShowSetIndex 1057  
setShowSetName 1057  
setSideToCenter 930, 933, 951  
setSideToLeft 930, 933  
setSideToRight 930, 933  
setSillHeight 584  
setSinkDiameter 329  
setSipComponents 357  
setSize 637  
setSnapMode 967  
setSolidMillDiameter 705  
setSolidPathOnly 705  
setSpacing 630, 926  
setSpacingBeam 524  
setStackThickness 329  
setStartDepth 679  
setStartEnd 657  
setStereotype 912  
setStretch 943  
setString 199  
setStrVar 496  
setStyle 465, 552, 554, 610, 619  
setStyleDescription 293  
setStyleNameSF 595  
setSubAssemblyName 160, 221  
setSubCode 502  
setSubComponents 398  
setSubLabel 423, 943  
setSubLabel2 423, 943  
setSubMap 181, 371  
setSubMapX 291, 371, 398, 664  
setSubMapXProject 160  
setSubType 329, 478  
setSupportTypeName 657  
setSurfaceQualityOverrideBottom 465, 610  
setSurfaceQualityOverrideTop 465, 610  
setTailComponents 324  
setText 502, 1028, 1030, 1033  
setTextHeight 735  
setTextLocation 1028, 1030, 1033  
setTextPosition 735  
setTexture 443  
setThickness 341  
setThreadLength 330  
setTileLathDistribution 527  
setToAlignCoordSys 148  
setToAlignPlaneToWorld 148  
setToAlignWorldToPlane 148  
setToleranceLumber 357  
setToolIndex 630, 640, 760, 771, 926, 928, 930, 933, 936  
setToolTip 255  
setToolType 928  
setToProjection 148  
setToRotation 148  
setToTranslation 148  
setTransparency 371  
setTrueColor 371  
setTurningDirectionWith 930, 933  
setType 329, 423, 595, 943  
setUcs 284  
setUseDirection 698  
setUseDisplayTextHeight 1021, 1028, 1030, 1033  
setUseModelExtentsForHeight 545  
setUseVerticalDimensions 919  
setVacuum 930, 933, 936, 951  
setValuesFromCatalog 527, 599  
setVecBevelNormal 532  
setVecDir 502, 532  
setVector3d 199  
setVecX 255  
setVecY 255, 527  
setVecZ 527  
setVertexPoints 527  
setVerticalTolerance 657  
setVertices 545, 561  
setViewData 1049, 1057  
setViewHandle 1057  
setViewUserId 1057

setWallRoofLine 512  
setWidth 364, 584, 604  
setWidthPS 1049, 1057  
setWoodClass 313  
setWoodGrainDirection 465  
setWoodKind 313  
setX 103  
setXAxisDirectionInXYPlane 459, 465  
setXmlContent 199  
setY 103  
setZ 103  
setZone 478  
setZoneIndex 502, 630, 640, 926, 930, 933, 936  
shadowProfile 132, 550  
shapeType 255, 604  
shapeType2String 604  
ShedWizard 225  
sheet 236, 276, 296, 395, 459, 478  
Shop drawing 177  
shopdraw 880  
ShopDrawView 644, 888  
showAdd 199  
showChildPanelFlipMark 357  
showDialog 398, 404, 972, 981  
showDialogOnce 972  
showDuringSnap 1006  
showElementTools 404  
showInDispComp 263, 1006  
showInDispRep 1006  
showInDxa 1006  
showInShopDraw 1006  
showInTsInst 1006  
showSet 619, 622  
showSetDefineEntities 1057  
showSetEntities 1057  
showSetIndex 1057  
showSetName 1057  
shrinkWrapBody 657  
sidelsCenter 930, 933  
sidelsLeft 930, 933  
sidelsRight 930, 933  
sillHeight 584  
SimpleScarf 768  
simplify 114, 124  
sin 84  
SingleElementRef 533, 538  
singleElementRefs 533, 538  
sinkDiameter 329  
sip 236, 276, 296, 395, 465, 478  
SipComponent 360  
sipComponentAt 357  
sipComponents 357  
SipEdge 465, 476  
sipEdges 465  
sipEntity 542  
SipStyle 357, 465  
sipToMeTransformation 542  
size 637  
Slab 648  
sliceBody 132  
Slot 769  
    Example of Slot and Mortise 1094  
snapPlaneInModel 622  
solidBox 443  
solidHeight 423  
solidLength 423  
SolidSubtract 769  
solidWidth 423  
sorted 80  
spacing 630, 926  
spacingBeam 524  
spanExcluding 87  
spanIncluding 87  
spawn 160, 217  
spawn\_detach 160, 217  
SpecialMill 771  
splitPLine 124  
splitSegments 124, 478  
sqrt 83  
stackThickness 329  
standardSizeNameInUse 584  
startNodeIndex 653  
statement 75  
status 653  
Store state in dwg 224  
StoreTslStateInDwg 224  
storeUniqueIdsOfCreatedEntities 244  
strCutN 443  
strCutNC 443  
strCutP 443  
strCutPC 443  
stretchDynamicTo 443

stretchDynamicToMultiple 443  
 stretchEdgeTo 465  
 stretchOutlineTo 512  
 stretchStaticTo 443  
 stretchStaticToMultiple 443  
 String 79, 87  
 string2ShapeType 604  
 Strings 87  
 strText 842  
 strVar 496  
 style 465, 552, 554, 610, 619  
 styleDescription 293  
 styleNameSF 595  
 subAssemblies 423  
 subAssembly 423  
 subAssemblyName 160, 221  
 subCode 502  
 subComponents 398  
 subGroups 284  
 sublabel 423  
 sublabel2 423  
 subMap 181, 371  
 subMapKeys 181, 371  
 subMapX 291, 371, 398, 664  
 subMapXKeys 291, 371, 398, 664  
 subMapXKeysProject 160  
 subMapXProject 160  
 subPart 132, 165  
 subType 329, 478  
 supportTypeName 657  
 suppressLine 735  
 surfaceQualityBottom 341, 357  
 surfaceQualityOverrideBottom 465, 610  
 surfaceQualityOverrideTop 465, 610  
 SurfaceQualityStyle 341, 361  
 surfaceQualityTop 341, 357  
 swap 80  
 swapLastWith 199  
 switchToModelSpace 160, 1049  
 switchToPaperSpace 160, 1049

Predefined variables T-connection 991  
 tailComponents 324  
 tan 84  
 terminator 74  
 TestReport 247  
 text 502  
 textHeight 1006  
 textHeightForStyle 1006  
 textLengthForStyle 1006  
 texture 443  
 threadLength 330  
 thumbnail 68  
 tileLathDistribution 527  
 tileMap 566  
 tileName 566  
 tileType 566  
 tileTypePartner 566  
 timberGrade 321  
 timberMaterial 321  
 timberName 321  
 TirolerSchloss 773  
 TN 87  
 token 87  
 tokenize 87  
 toleranceLumber 357  
 toolEnt 395, 550, 561, 571, 777  
 toolEntTypeName 777  
 toolIndex 630, 640, 926, 928, 930, 933, 936  
 toolName 678, 875, 948  
 toolSubType 777, 780, 786, 799, 806, 811, 842, 852, 855, 860, 863, 866  
 toolTip 255  
 toolType 777, 928  
 totalWidth 657  
 transformBy 103, 108, 110, 114, 124, 132, 151, 165, 199, 371, 402, 1021, 1028, 1030, 1033, 1065, 1066  
 transformContent 296  
 transformLineSegs 148  
 transformPoints 148  
 transparency 371, 1006  
 TransType 199  
 triggerGenerateConstruction 478  
 triggerGenerateSheeting 478  
 trim 114  
 trimLeft 87

## - T -

T 87, 160  
 Predefined variables T-connection 991  
 T macro

trimRight 87  
 TruckDefinition 364, 1006  
 true  
     Logical operations 84  
 trueColor 160, 371  
 TrussConnection 655  
 TrussDefinition 308  
 TrussEntity 650  
 TrussEnvelope 650, 653  
 TrussEnvelopeLine 653  
 TrussEnvelopeNode 653  
 TrussSupport 657  
 TSL 57  
 TSL editor 68  
 TSL editor window 60  
 tsInst 236, 276, 291, 296, 404, 478  
 tsInstAttached 371  
 TsIScript 362  
 turnGroupVisibilityOff 284, 1049  
 turnGroupVisibilityOn 284, 1049  
 turningDirectionWith 930, 933  
 type 57, 329, 352, 423, 595, 653  
 type of the TSL 68  
 typeDxfName 371  
 typeName 291, 293, 371

**- U -**

U 160, 175  
 uniqueId 371  
 Unit 160, 175  
 Units  
     Conversion to HSB Units 175  
 updateYield 610  
 useModelExtentsForHeight 545

**- V -**

vacuum 930, 933, 936  
 value 967  
 Variable  
     Predefined variables E-connection 1001  
 Variables  
     Predefined variables G-connection 995  
     Predefined variables T-connections 991  
 vecBevelNormal 532

vecCenterLogOffset 443  
 vecCenterOffset 443  
 vecChamferEdges 793  
 vecCut 799  
 vecD 151, 423  
 vecDir 502, 532, 681  
 vecFree 820  
 vecFromDir 545  
 vecLine 811  
 vecMarkerSide 845  
 vecN1 811  
 vecN2 811  
 vecNormal 476  
 vecNrm1 146  
 vecNrm2 146  
 vecOffsetApplied 255  
 vecPlaningSurfaces 855  
 vecSide 780, 783, 786, 796, 799, 803, 806, 809, 811, 814, 817, 820, 823, 829, 833, 836, 839, 842, 845, 848, 852, 855, 857, 860, 863, 866, 869, 872, 875  
 vecSlope 796  
 Vector 79  
     Point and Vector Manipulation 103  
 Vector3d 79  
 vecViewBlockOffset 625  
 vecX 110, 148, 255, 423, 478  
 vecY 110, 148, 255, 423, 478  
 vecZ 110, 148, 423, 478, 496, 820, 823, 857  
 vecZOutwards 705  
 version 404  
 vertexPoints 114  
 verticalTolerance 657  
 vertices 545, 561  
 View  
     Shop drawing 177  
 ViewData 622, 888, 1049, 1057  
 viewDirections 255  
 viewHandle 1057  
 Viewport 291, 1049, 1057  
 views 619  
 viewScale 622  
 viewSet 545  
 viewUserId 1057  
 vis 151, 393, 912, 1057  
 visualize 108, 124, 151, 393, 398, 402, 912, 1057  
 volume 132, 423

**- W -**

Wall 657  
while 77  
width 96, 364, 584, 604, 650  
widthPS 1049, 1057  
widthRough 584  
woodClass 313  
woodClassAt 313  
woodGrainDirection 465, 542  
woodKind 313  
writeEntitySetIntoDwg 276  
writeTextFile 214  
writeToDwg 276  
writeToDwgFile 1006  
writeToDxfFile 1006  
writeToDxxFile 236  
writeToXmlFile 199, 217

**- X -**

X 103  
xrefDatabases 270  
XrefLocker 253  
xrefName 371, 478

**- Y -**

Y 103

**- Z -**

Z 103  
zFaceOffset 657  
zone 478  
zoneIndex 478, 496, 502, 630, 640, 926, 928, 930, 933, 936

---

1178

hsbCAD TSL

---

**Done!**