



Scia Design Forms

Application manual

Scia Design Forms	1
Application manual	1
Release notes for version 5.0	44
User application	44
Builder application	44
Common settings	45
System requirements	46
Processor	46
RAM	46
Graphic card	46
Space on drive for projects and temp files	46
Minimum resolution	46
Operating System requirements	46
Installation	47
Scia Design Forms Installation	47
Installation of software protection (including tryout)	50
Protection	52
Installation of protection by local key	52
Licence activation - hardlock (dongle), network (floating) licence	54
Licence types	55
Try - out	55
Student	55
Standard	55
Application update	56
Language settings	57
Terminology and file types	58
File type in version 3	58
File formats in version 4	58
Error reports	58

User	60
About the application	61
User application settings	62
Zoom in the User application	62
Shortcuts in the User application	63
The Forms menu	64
List of calculations in Design Forms	65
Form menu editor	69
Add a folder to the tree	71
Add subfolder to the tree	71
Rename folder	71
Delete folder / subfolder / form	72
Add forms to the folder	72
Set icon for the folder / form	72
Filter of forms in the left part	72
Working with the dialogue	74
Inputting the values	75
Default value sets and the *.DEFAULT file	76
Loading default values and Manager of default value sets	80
How to load default sets (priorities)	83
Working with the form	84
Inserting the calculation form in the project	85
Renaming the forms	86
Delete a calculation from a project	87
Language selection	88
Layout selection - output details	89
Calculation output	90
Form Annotation	91
Working with projects	92

New, open, save	93
Header/footer definition	94
Inserting the picture	94
Moving pictures	95
Deleting pictures	95
Project export	96
Printing projects	98
FAQ for the User application	99
Message when a new form is added	99
Numeric variable with value NaN	99
Could not find file "... "	99
Builder application	100
About the application	101
Tools - Program settings	102
Using zoom in the code editor	105
Code editor shortcuts	106
What is form?	107
Source code creating	108
Code editor shortcuts	109
Table with special symbols for code editor	110
Source code of the calculation in the Code editor	112
Find	112
Replace	113
Table of variables	115
Variable names	116
Renaming variables	116
IDs	117
Description	117
Symbol	117

Value	117
Units	117
Precision	117
Type Double	120
Syntax:	120
Example	120
Type String	121
Syntax:	121
Example	121
Type Boolean	122
Syntax:	122
Example	122
Type Structured	123
Syntax:	123
Example	123
Nested variables in structured variable	123
Syntax:	123
Example	123
New Structure()	124
Example	124
<structured_variable>.Add()	124
Syntax:	124
Example	124
C# function POINT	125
Point()	125
Syntax:	125
Example	125
PointF()	126
PointD()	126

Predefined structured variables	126
ReinfBar	126
Forces1D	126
Forces2D	126
MaterialPoint	126
MaterialDiagram	126
Type Object	127
Syntax:	127
Example	127
Array	128
Syntax:	128
Example	128
Declaration of an array in the code	128
Example	129
Function PURGE	135
Import from MS Excel	136
The tool for debugging the code - Trace listener	138
Creating the Layout	139
Example:	139
Layout menu	143
Renaming a layout	143
Creating new layouts	143
Switching layouts	143
Calculation layout	145
Visibility	145
Font settings	146
Default component selection	146
Cancelling the connection to default component	146
Horizontal align	146

Vertical align	147
Equation settings	147
Predefined styles of components in layout	149
Copy formatting from one layout to another	151
Creating the Dialogue	153
Form dialogue	156
Drag and drop by the mouse	156
Press the '>>>' button	156
Drag and drop by the mouse (if not docked)	156
Using the keyboard (if docked)	156
Using the component tree	157
Form dialogue - adding and formatting components	157
Drag and drop by mouse	157
Double-click	158
Dialogue - table input	162
Dialogue item tree	163
Special variable IO - input/output	165
How to create a combo-box	167
Hide dialogue components by using script	169
Syntax:	169
Example	169
Inserting and using Libraries in the Dialogue	170
Inserting a library in the dialogue	170
Deleting a library from the dialogue.	170
Concrete section library	172
Steel section library	172
Steel library	173
Concrete library	173
Timber library	174

Bolts library	174
Custom library	175
Expanding/collapsing a library	177
Go through items	177
Available cross-sections in the Steel section library	178
List of ID codes	179
Steel cross sections:	179
Concrete cross sections:	180
New concrete cross sections:	180
IO values:	180
Stirrup characteristics:	180
Defined steel classes	181
The list of ID codes	182
Defined concrete classes	183
The list of codes ID	184
Defined timber classes	185
The list of codes ID	186
Bolts definition	187
Defined bolt classes	187
Defined bolt diameters	187
Selection of material and diameter	187
The list of ID codes	188
Example	189
Custom library displayed as combo-box in the Dialogue	191
Custom library Editor	191
A change, damage or loss of the definition file (XML)	192
How is the change displayed in the USER application:	192
How is the change displayed in the BUILDER application:	192
Custom library - manual definition and Editor	194

Custom library Editor	194
Category and Item	194
Displaying a custom library in the Dialogue	195
How to create a custom library	197
Columns definition in the Editor	197
Arrays in the Custom library	197
Column properties	198
Custom library - automatically defined	200
Inserting values in the library	200
Calculation header	201
Images	202
Form Annotation	203
Translations	204
CALC_xxxx	205
DIALOG_xxxxxx	205
Form Annotation	205
LAYOUT_xx	205
TEXT_xxxxxx	206
VARIABLE_xxx	206
Commands reference guide	208
Comments	209
Syntax:	209
Example 1	209
Example 2	209
Example 3	209
Standard commands	209
Syntax:	209
Example	210
Block visibility	211

Syntax:	211
Example	211
Syntax:	212
Example 1	212
Example 2:	213
EXIT()	213
Syntax:	213
Example	213
Syntax:	213
Example:	213
Continue(), Break()	214
Syntax:	214
EXIT()	214
Syntax:	214
Example	214
Syntax:	215
Example	215
Continue(), Break()	215
Syntax:	215
EXIT()	216
Syntax:	216
Example	216
Syntax:	216
Example	217
Using SWITCH CASE for a COMBO-BOX	217
Numeric (Number)	217
String (Text)	217
Syntax:	218
Example:	218

TEXT(<text>)	220
Syntax:	220
Example	220
VAL(<expression>, <accuracy>)	221
Syntax:	221
Example:	221
Description (<variable>)	221
Syntax:	221
Example	222
==, !=, >, >=, <, <=	223
Syntax:	223
Example	223
&&, , & , !(...)	224
Syntax:	224
Example	225
Mathematical operations - basic	226
Syntax:	226
Example	226
Syntax:	227
Example	227
SUM (<variable>, <variable>, <variable>, ...)	228
Syntax:	228
Example	228
Exponent and POWER command (<basic>, <exponent>)	228
Syntax:	228
Example	228
SQRT(<value>)	229
Syntax:	229
Example	229

MIN(<value>, <value>, ... <value>) and PARMIN(<value>, <value>, ... <value>)	229
Syntax:	229
Example	230
MAX(<value>, <value>, ... <value>) a PARMAX(<value>, <value>, ... <value>)	230
Syntax:	230
Example	230
SIN(<angle_in_degrees>)	231
COS(<angle_in_degrees>)	231
TG(<angle_in_degrees>)	231
COTG(<angle_in_degrees>)	231
ARCSIN(<ratio>) (from -1 to +1)	231
ARCCOS(<ratio>) (from -1 to +1)	231
ARCTG(<ratio>) (from -∞ to +∞)	231
ARCTG(<ratio>) (<value1>, <value2>)	231
ARCCOTG(<ratio>) (from -∞ to +∞)	231
Syntax:	231
Example	231
Syntax:	232
Example	232
Syntax:	232
Example	232
Syntax:	233
Example	233
DIV(<value>)	233
Syntax:	233
Example	233
MOD(<value>)	234
Syntax:	234
Example	234

Mathematical operations - advanced	235
LOG(<value>, <base>)	235
Syntax:	235
Example	235
LN(<value>)	235
Syntax:	235
Example	235
Syntax:	236
Tolerance	236
Example	236
Syntax:	237
Example:	237
Syntax:	237
Example:	238
Tools in Dialogue	238
Syntax:	239
Example	239
Syntax:	240
Example	240
Calculation and/or input from external files	241
Syntax:	241
Example	241
CALCULATE	242
Syntax:	242
(bool TransferVariables)	242
DRAW	242
Syntax:	242
(bool TransferVariables)	242
(int LayoutIndex)	242

DRAW(True), DRAW(False), DRAW()	243
Example	243
Syntax:	244
Syntax:	244
Example	245
CONVERT()	245
Syntax:	245
Example	245
GetStructure()	246
Syntax:	246
Example	246
Special EN functions for civil engineers	246
Syntax:	246
Example	247
Syntax:	247
Example	248
Syntax:	248
Example	248
ConcreteDiagramULS()	248
Syntax:	248
ConcreteDiagramSLS()	249
Syntax:	249
ReinfDiagramULS()	249
Syntax:	249
ReinfDiagramSLS()	249
Syntax:	249
Example	249
Enum	249
User class	250

Syntax:	250
Example	250
Usage:	250
User functions	250
Syntax:	251
Example	251
Static Math class	252
GRAPH	252
Syntax:	252
Example	252
Syntax:	253
Example	253
Line definition - an array of points (by coordinates)	253
Syntax:	253
Example	254
Syntax:	254
Example	254
Basic new Pen brush settings (Color.<colour_name>, <thickness>);	255
Syntax:	255
Example	255
LinearGradientBrush()	256
Syntax:	256
Example	256
HatchBrush()	257
Syntax:	257
Example	258
<variable>.Caption.Caption - Graph title	258
Syntax:	258
Example	259

<variable>.Caption.Font = new Font() - Graph title font	259
Syntax:	259
Example	259
<variable>.XAxis.Caption - Name of the X-axis	259
Syntax:	259
Example	259
<variable>.XAxis.Font= new Font() - X-axis label font	260
Syntax:	260
Example	260
Axis grid size and grid scale	260
<variable>.XAxis.MajorStep - X-axis grid size	260
Syntax:	260
Example	260
<variable>.YAxis.Scale - Axis-step scale	261
Syntax:	261
Example	261
Axis value format	261
<variable>.YAxis.Format - Axis value format	261
Syntax:	261
Example	262
Tables in the layout	262
Syntax:	262
Example	262
Syntax:	263
Example	263
<variable>[<row_index>][<column_index>].Value = "<text>"	263
Syntax:	263
Example	264
Wrap = true	264

Syntax:	264
Example	264
Column definition:	265
Row definition:	265
Alignment in tables - columns	265
<variable>.Columns[<index>].Alignment = ContentAlignment.<alignment>;	265
Syntax:	265
Example	266
Alignment in tables - rows	266
<variable>[<index>].Alignment = ContentAlignment.<alignment>;	266
Syntax:	266
Example	266
Pen for outside border lines	266
<variable>.BorderPen = new Pen(Color.<colour>, <thickness>)	266
Syntax:	266
Example	266
Pen for inside border lines	266
<variable>.GridPen = new Pen(Color.<colour>, <thickness>)	266
Syntax:	267
Example	267
Pen for outside border lines of a single row	267
<variable>[<index>].BorderPen = <variable>.BorderPen	267
Syntax:	267
Example	267
Font for a single row	267
<variable>[<index>].Font = new Font("", <size>)	267
Syntax:	268
Example	268
Cell padding	268

<variable>.Padding.All = <value>	268
Syntax:	268
Example	268
<variable>[i].Padding.All = <value>	268
Syntax:	268
Example	269
<variable>[i][y].Padding.All = <value>	269
Syntax:	269
Example	269
Graphics	270
Syntax:	270
Example	270
Draw(zoom)	271
Syntax:	271
Example	271
Draw(width, height, boolean of aspect ratio)	271
Syntax:	271
Example	272
Draw(enlarge horizontal, enlarge vertical, rotation angle)	272
Syntax:	272
Example	272
Draw(width, height)	273
Syntax:	273
Example	274
DrawGraphics(<other_graphics>, <movement_x>, <movement_y>)	274
Syntax:	274
Example	275
DrawGraphics(<other_graphics>, <movement_x>, <movement_y>, <angle>)	275
Syntax:	275

Example	275
DrawGraphics(<other_graphics>, <movement_x>, <movement_y>, <zoom_x>, <zoom_y>, <angle>); ...	276
Syntax:	276
Example	276
SaveUCS	277
Syntax:	277
Example	278
LoadUCS	278
Syntax:	278
Example	278
MoveUCS	278
Syntax:	278
Example	278
RotateUCS	278
Syntax:	278
Example	279
ScaleUCS	279
Syntax:	279
Example	279
Overall example	279
Syntax:	280
Example	280
Syntax:	282
Example	282
Syntax:	283
Example	283
DrawPolyline	284
Syntax:	284
Example	284

FillPolygon	285
Syntax:	285
Example	285
Syntax:	286
Example	287
Syntax:	288
Example	288
Syntax:	289
Example	289
DrawCircle	290
Syntax:	290
Example	291
FillCircle	291
Syntax:	291
Example	291
DrawEllipse	292
Syntax:	292
Example	292
FillEllipse	293
Syntax:	293
Example	293
Syntax:	294
Example	295
Syntax:	296
Example	296
DefaultDimStyle - brush	297
Syntax:	297
Example	297
DefaultDimStyle - font, text format, scale	297

Font	297
Syntax:	298
Example	298
Format	299
Syntax:	299
Example	299
Scale	300
Syntax:	300
Example	300
DefaultDimStyle - line, end mark, plotline	301
CaptionArrowShape	301
Syntax:	301
Example	302
CaptionArrowSize	302
Syntax:	302
Example	303
DimArrowShape	303
Syntax:	303
Example	303
DimArrowSize	304
Syntax:	304
Example	304
FixedDimLength	304
Syntax:	304
Example	304
LineOffset	305
Syntax:	305
Example	305
Overlap	305

Syntax:	306
Example	306
DefaultDimStyle - pen	306
Pen	306
Syntax:	307
Example	308
DefaultPen	308
Syntax:	308
Example	309
DrawDim	309
Syntax:	309
Example	309
DrawDimX (Y)	310
Syntax:	310
Example	311
DrawDimChainX(Y)	312
Syntax:	312
Example	313
DrawAngleDim	313
Syntax:	313
Example	314
DrawRadiusDim	315
Syntax:	315
Example	316
DrawDiameterDim	316
Syntax:	316
Example	316
Size.Width	317
Syntax:	317

Example	317
Size.Height	317
Syntax:	317
Example	318
TM.dX	318
Syntax:	319
Example	319
TM.dY	319
Syntax:	319
Example	319
TM.ZoomX or TM.ZoomY	319
Syntax:	320
Example	320
TR	320
Syntax:	321
Example	321
Variable types	323
Syntax:	325
Example	325
Syntax:	326
Example	326
Syntax:	327
Example	327
Syntax:	328
Example	328
Nested variables in structured variable	328
Syntax:	328
Example	328
New Structure()	329

Example	329
<structured_variable>.Add()	329
Syntax:	329
Example	329
C# function POINT	330
Point()	330
Syntax:	330
Example	330
PointF()	331
PointD()	331
Predefined structured variables	331
ReinfBar	331
Forces1D	331
Forces2D	331
MaterialPoint	331
MaterialDiagram	331
Syntax:	332
Example	332
Syntax:	333
Example	333
Declaration of an array in the code	333
Example	334
Library CONCRETE TOOLBOX	334
CompressedConcrete	334
Syntax:	334
Syntax:	335
CompressedReinf	335
Syntax:	335
Syntax:	335

TensionConcrete	336
Syntax:	336
Syntax:	336
TensionReinf	336
Syntax:	336
Syntax:	337
CompressedConcrete	337
Syntax:	337
Syntax:	338
Syntax:	338
Syntax:	338
Example:	339
CompressedReinf	339
Syntax:	339
Syntax:	339
Syntax:	340
Syntax:	340
Example:	340
TensionConcrete	341
Syntax:	341
Syntax:	341
Syntax:	341
Syntax:	342
Example:	342
TensionReinf	342
Syntax:	342
Syntax:	343
Syntax:	343
Syntax:	343

Example:	344
Outline	344
Syntax:	344
GetGravityCenterOfPolygon	344
Syntax:	344
CreateCS	345
Syntax:	345
Syntax:	345
Syntax:	345
Syntax:	346
Syntax:	346
InitialiseDesignAddData	346
Syntax:	346
Syntax:	347
SetDiameterAndCover	347
Syntax:	347
SetOffsetOfLayersOfReinforcement	347
Syntax:	347
SetCoefficientForReinforcementAreaForLayer	348
Syntax:	348
SetIfCanBeUseUserReinforcement	348
Syntax:	348
SetMaximalCompressedPortion	348
Syntax:	348
SetExtraRotationPoint	348
Syntax:	349
SetDesignMaterialDiagram	349
Syntax:	349
SetDesignUniaxialAddData	349

Syntax:	349
SetEdgeAddDataGen	349
Syntax:	350
SetCornerAddData	350
Syntax:	350
GetInfoAboutCorner	350
Syntax:	350
GetNumberOfEdgesWithDesignedReinforcement	350
Syntax:	350
GetDesignedReinforcementInfoForEdge	351
Syntax:	351
GetNumberOfLayersWithReinforcementForEdges	351
Syntax:	351
GetDesignedReinforcementInfoForLayerOfEdge	351
Syntax:	351
DesignReinforcementUniaxialAroundGeneralAxis	352
Syntax:	352
Syntax obsolete:	352
DesignReinforcementUniaxialAroundYAxis	352
Syntax:	352
DesignReinforcementUniaxialAroundZAxis	353
Syntax:	353
DesignReinforcementGeneralBiaxial	353
Syntax:	353
Syntax:	353
DesignReinforcementGeneral	354
Syntax:	354
Syntax:	354
DesignReinforcementCorner	354

Syntax:	354
DesignUniaxialGenLayers	355
Syntax:	355
DesignReinforcementGeneralBiaxialLayers	355
Syntax:	355
DesignReinforcementGeneralLayers	356
Syntax:	356
GetNumberOfEdgesOfWholeCrossSection	356
Syntax:	356
Syntax:	356
GetInfoAboutEdgeOfCrossSection	357
Syntax:	357
Syntax:	357
GetIndexesOfEdgesWithDesignedReinforcementFromUniaxialDesign	357
Syntax:	357
CalculateReinfRatioForEdge	357
Syntax:	357
CreatePracticalCalculator	358
Syntax:	358
SetAreasOfReinforcementForEdgeForPracticalCalculator	358
Syntax:	358
OptReinfDiamOnEdge	358
Syntax:	358
OptReinfDiamOnEdge1	359
Syntax:	359
MaxNumberBarsOnEdge	359
Syntax:	359
NumberBarsOnEdge	359
Syntax:	359

SpacingReqReinf	360
Syntax:	360
LengthOfLineForDesign	360
Syntax:	360
AssignReinfToEdge	360
Syntax:	360
ArrayReqReinf	361
Syntax:	361
SetAddData	361
Syntax:	361
SetDesignMaterial	361
Syntax:	361
ClearReinfDiagram	362
Syntax:	362
CrackingForces1	362
Syntax:	362
CrackingForces2	362
Syntax:	362
CentreOfPolygon	363
Syntax:	363
GetCountFibresConcrete	363
Syntax:	363
Syntax:	363
GetCountFibresReinforcement	364
Syntax:	364
Syntax:	364
GetCssHeight	364
Syntax:	364
Syntax:	364

GetFibrePositionConcrete	365
Syntax:	365
GetFibrePositionReinf	365
Syntax:	365
Syntax:	366
GetOutline	366
Syntax:	366
Syntax:	366
GetReinfBars	366
Syntax:	367
Syntax:	367
PointInPolygon	367
Syntax:	367
StressFromStrain	367
Syntax:	367
Characteristics	368
Syntax:	368
Syntax:	369
Syntax:	369
Syntax:	370
SetIntDiagramAddData	370
Syntax:	370
SetExtraRotationPoint	370
Syntax:	371
SetMaximalCompressedPortion	371
Syntax:	371
InteractionDiagramNM	371
Syntax:	371
InteractionDiagramNMyMz	371

Syntax:	372
CountPointsInteractionDiagram	372
Syntax:	372
CountPointsInteractionDiagramNMyMz	372
Syntax:	372
UltimateBorder3D	372
Syntax:	373
UltimateBorder	373
Syntax:	373
IntersectionsInteractionDiagramNM	373
Syntax:	373
ResistancelInteractionDiagramNM	374
Syntax:	374
HorizontalSectionInteractionDiagram3D	374
Syntax:	374
VerticalSectionInteractionDiagram3D	374
Syntax:	375
GeneralSectionInteractionDiagram3D	375
Syntax:	375
CountPointsHorizontalSectionInteractionDiagram3D	375
Syntax:	375
CountPointsVerticalSectionInteractionDiagram3D	376
Syntax:	376
CountPointsGeneralSectionInteractionDiagram3D	376
Syntax:	376
IntersectionsInteractionDiagram3D	377
Syntax:	377
ResistancelInteractionDiagram3D	377
AngleOfNeutralAxis	378

Syntax:	378
Syntax:	378
BalanceHeighCompressionZone	378
Syntax:	378
Syntax:	379
CompressivePartOfCss	379
Syntax:	379
CurvatureRes	379
Syntax:	379
CurvatureY	380
Syntax:	380
CurvatureZ	380
EffectiveHeight	380
Syntax:	380
Syntax:	381
EquilibriumPlane	381
Syntax:	381
Syntax:	381
GetIntersectionPoints	381
Syntax:	382
Syntax:	382
GetIntersectionPointsCount	382
Syntax:	382
Syntax:	382
HeightOfCompressionZone	383
Syntax:	383
Syntax:	383
InnerLevelArm	383
Syntax:	383

Syntax:	384
InnerLevelArm4Param	384
Syntax:	384
InnerLeverArmPartNeg	384
Syntax:	384
InnerLeverArmPartPos	385
Syntax:	385
LimitHeighCompressionZone	385
Syntax:	385
Syntax:	386
TensilePartOfCss	386
Syntax:	386
TranslationX	386
Syntax:	386
ForcesFromEquilibriumOfPlane	387
Syntax:	387
ArrayOfReinforcement	387
Syntax:	387
CssCharacteristicCon	388
Syntax:	388
CssCharacteristicReinf	388
Syntax:	388
RegionArrayOfReinforcement	388
Syntax:	389
RegionCharacteristicCon	389
Syntax:	389
RegionCharacteristicReinf	389
Syntax:	390
RegionCharacteristicTransf	390

Syntax:	390
ReinfBars	391
Syntax:	391
CompressedConcrete	391
Syntax:	391
Syntax:	391
CompressedReinf	392
Syntax:	392
Syntax:	392
Resultant	392
Syntax:	392
Syntax:	393
TensionConcrete	393
Syntax:	393
Syntax:	393
TensionReinf	393
Syntax:	393
Syntax:	394
WholeConcrete	394
Syntax:	394
Syntax:	394
WholeReinf	394
Syntax:	395
Syntax:	395
SetForces	395
Syntax:	395
SetForcesId	396
Syntax:	396
AngleShearForceResultant	396

Syntax:	396
CssDimension	397
Syntax:	397
EffectiveHeighProjection	397
Syntax:	397
Syntax:	398
InnerLeverArmPartNegProjection	398
Syntax:	398
InnerLeverArmPartPosProjection	398
Syntax:	399
InnerleverArmProjection	399
Syntax:	399
Syntax:	400
PositionWidthCssForShear	400
Syntax:	400
ShearForceResultant	400
Syntax:	401
WidthCssForShear	401
Syntax:	401
WidthCssGeneral	401
Syntax:	401
CoverReinf	402
Syntax:	402
MaxSpacingReinf	402
Syntax:	402
MinSpacingReinf	403
CreateStirrup	403
Syntax:	403
NumberOfStirrupLinks	404

Syntax:	404
ClearStirrupZones	404
Syntax:	404
CreateStirrupZone	405
Syntax:	405
GetCountFibresConcrete	405
Syntax:	405
GetCountFibresReinforcement	406
Syntax:	406
GetExtremeStrainOfConcreteMax	406
Syntax:	406
GetExtremeStrainOfConcreteMin	406
GetExtremeStrainOfReinfMax	407
Syntax:	407
GetExtremeStrainOfReinfMin	407
Syntax:	407
GetExtremeStressOfConcreteMax	407
Syntax:	408
GetExtremeStressOfConcreteMin	408
Syntax:	408
GetExtremeStressOfReinfMax	408
Syntax:	408
GetExtremeStressOfReinfMin	409
Syntax:	409
GetFiberReinfDiameter	409
Syntax:	409
GetFibreExtremeStrainMax	410
Syntax:	410
GetFibreExtremeStrainMin	410

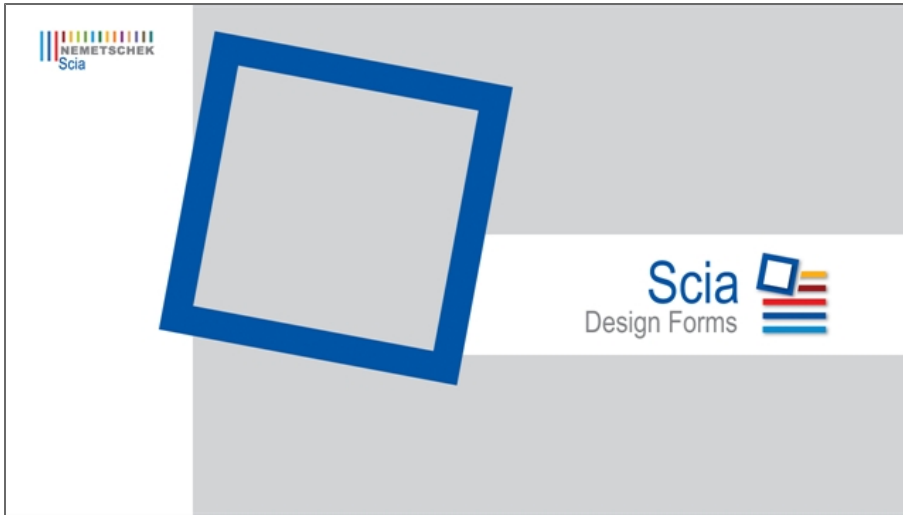
Syntax:	410
GetFibreExtremeStressMax	410
Syntax:	410
GetFibreExtremeStressMin	411
Syntax:	411
GetFibrePositionConcrete	411
Syntax:	411
Syntax:	411
GetFibrePositionReinf	412
GetLimitStrainOfConcrete	412
Syntax:	412
Syntax:	413
GetLimitStrainOfReinf	413
Syntax:	413
Syntax:	413
GetLimitStressOfConcrete	413
GetLimitStressOfReinf	414
Syntax:	414
Syntax:	414
GetReinfExtremeStrainMax	414
Syntax:	415
GetReinfExtremeStrainMin	415
Syntax:	415
GetReinfExtremeStressMax	415
Syntax:	415
GetReinfExtremeStressMin	415
Syntax:	416
GetStrainOfConcrete	416
Syntax:	416

Syntax:	416
GetStrainOfReinf	416
Syntax:	417
Syntax:	417
GetStressOfConcrete	417
Syntax:	417
Syntax:	418
GetStressOfReinf	418
Syntax:	418
Syntax:	418
AdditionalStrainReinf	419
Syntax:	419
CreateEffRectCssForTorsion	419
Syntax:	419
CreateGeneralCssForTorsion	419
Syntax:	420
LongReinAreaForTorsion	420
Syntax:	420
MinDistReinfInsideStirrup	420
Syntax:	420
ParameterAkFromEffRectCSS	421
Syntax:	421
ParameterAkFromEffRectCSS	421
Syntax:	421
ParameterAkFromTorGenCss	421
Syntax:	421
ParameterUkFromEffRectCss	422
Syntax:	422
ParameterUkFromPolygon	422

Syntax:	422
ParameterUkFromTorGenCss	422
Syntax:	422
PolygonCssForTorsion	423
Syntax:	423
Libraries	424
Steel cross sections:	426
Concrete cross sections:	427
New concrete cross sections:	427
IO values:	427
Stirrup characteristics:	427
Code editor directives	432
Syntax:	432
Example	432
Automatic tests	433
About automatic tests	434
Create automatic tests	435
Add test	437
Edit test	437
Deactivate test set	437
Delete test set	438
Run and evaluation of the test	439
Important rules when creating a new form	440
The dialogue	440
Long calculation	440
ID	441
Translation	441
Split texts	441
Pictures	441

Formatting	442
Page width	442
NaN	442
Predefined styles	443
Some additional information about the USER application	443
FAQ for the Builder application	444
Messages	444
FAQ	444
Error messages	445
Missing end curly bracket	445
Wrong variable type	445
Common error in syntax	446
Common error in syntax	446
Numeric variable with value NaN	446
Unknown variable	446
External reference was not found	446
Uninitialized object	447
Unknown method	447
Unknown property	447
Unknown property	447
Additional packages and applications	448
Installation/Un-installation of additional package	448
Installation	448
Un-installation	448
Installation/Un-installation of additional application	448
Installation	448
Un-installation	449
Installation Scia Concrete Section	449
First issue handling - hardlock (dongle) licence	456

First issue handling - network (floating) licence	456
Scia Concrete Section	456
CLS examples	457
Examples of forms (CLS) which are more complicated	457
Selection from CustomDataTable	457
How to create header and footer in User application	459
How to use Dialog default in User application	461
How to export project CLP in User application	462
How to use Forms menu in User application	464
How to use Forms menu editor in User application	465
How to create a condition in Builder application	468
How to create and use Predefined styles in Builder application	471
How to create and use combo-box in Builder application	473
How to use dialogue containers in Builder application	476
How to create and use Custom Library in Builder application	482
How to create and use Auto test in Builder application	489



Scia Design Forms is a well-arranged and effective tool for automation of engineering calculations. The application uses predefined forms where the user inserts input data (loads, geometry, material characteristics, ...). Scia Design Forms generates a clear calculation report which is based on the inserted variables and predefined equations. This report can be printed, saved to a drive or transferred to another text editor.

Scia Design Forms is distributed with a set of forms certified by **Nemetschek Scia** and thus can be used immediately after installation.

The application contains two modules: **User and Builder**. The User module is developed for the use of predefined/existing forms. Builder allows the user to edit the already existing forms and create new ones.

It is possible to link a specific version of Scia Design Forms with a specific version of Scia Engineer. Check more in the [Scia Engineer webhelp](#).

User application

 The screenshot displays the Scia Design Forms software interface. The window title is "Design - long. reinf., both side, N.". The active value set is "CZ jazyk, CZ anne.". The calculation is for "C 30/37".

Design of longitudinal reinforcement on rectangular cross section according to Eurocode 2 CSN EN 1992-1-1

Concrete:

Design value of concrete compressive strength
 $f_{cd} = \frac{\alpha_{cc} \cdot f_{ck}}{\gamma_c} = \frac{1 \cdot 30 \cdot 10^6}{1,5} = 20 \text{ MPa}$

Concrete coefficients
 $\alpha_{cc} = 1$

Compressive strain of concrete
 $\epsilon_{cs} = \frac{f_{cd}}{E_c} = \frac{20 \cdot 10^6}{33 \cdot 10^9} = 0,606 \text{ ‰}$

Concrete strength $f_{ck} < 50 \text{ MPa} \Rightarrow \eta = 1 \quad \lambda = 0,8$

Reinforcement:

Design value of steel strength
 $f_{sd} = \frac{f_{yk}}{\gamma_s} = \frac{500 \cdot 10^6}{1,15} = 435 \text{ MPa}$

Compressive strain
 $\epsilon_{st} = \frac{f_{sd}}{E_s} = \frac{435 \cdot 10^6}{210 \cdot 10^9} = 2,07 \text{ ‰}$

Lever arm of internal forces:
 For tension reinforcement
 $z_1 = \frac{h}{2} - d_1 = \frac{0,5}{2} - 0,046 = 0,204 \text{ m}$
 For compression reinforcement
 $z_2 = \frac{h}{2} - d_2 = \frac{0,5}{2} - 0,039 = 0,211 \text{ m}$

Maximum reinforcement compressive strain
 $\epsilon_{cs} = 1000000\% \Rightarrow$ unlimited value of reinforcement compressive strain is used in the calculation

Cross section dimensions

Cross section width
 $b = 0,3 \text{ m}$
 Cross section height
 $h = 0,5 \text{ m}$
 Tension reinf. center of gravity
 $d_1 = c + \frac{\Phi_1}{2} = 0,035 + \frac{0,022}{2} = 46 \text{ mm}$
 Tension reinf. center of gravity
 $d_2 = c + \frac{\Phi_2}{2} = 0,035 + \frac{8 \cdot 10^{-3}}{2} = 39 \text{ mm}$
 Effective cross section height
 $d = h - d_1 = 0,5 - 0,046 = 0,454 \text{ m}$

The interface also shows a list of input parameters on the left, such as National annex (CSN EN 1992-1-1), Userdefined headline (uživatelský popis), Loading (Bending moment $M_{ed} = 330 \text{ kNm}$, The acting axial force $N_{ed} = 90 \text{ kN}$), Cross section dimensions (Cross section height $h = 0,5 \text{ m}$, Cross section width $b = 0,3 \text{ m}$), Material characteristics (Concrete strain $\epsilon_{cs} = 2 \text{ ‰}$, Deformation module of steel $E_s = 210000 \text{ MPa}$, Characteristic tensile strength of reinforcement $f_{yk} = 500 \text{ MPa}$, Ultimate concrete strain $\epsilon_{su} = 3,5 \text{ ‰}$), Reinforcement design (Concrete cover $c = 35 \text{ mm}$, Tensile reinforcement profile $\Phi_1 = 22 \text{ mm}$, Compression reinforcement profile $\Phi_2 = 8 \text{ mm}$), and Coefficients.

At the bottom, there is a diagram of a rectangular cross-section with reinforcement. It shows the width b , height h , effective depth d , and distances to the center of gravity of the tension (d_1) and compression (d_2) reinforcement. The diagram also indicates the internal forces F_{s2} and F_{cc} , and the neutral axis depth x .

Builder application

The screenshot displays the Scia Engineer interface with the following components:

- Script Editor (Left):** Contains a calculation script for designing longitudinal reinforcement. It includes logic for material strength, safety factors, and strain calculations.
- Parameter Table (Bottom Left):** A table listing design parameters with their symbols, values, units, and frequencies.

ID	Description	Symbol	Value	Unit	Freq
	Coefficient of stress distribution	λ	0.73		1
	Coefficient of material strength	η	0.85		1
	Coefficient of action conditions	η_{sd}	1		2
	Partial safety factor for reinforcement	γ_s	1.15		2
	Partial safety factor for concrete	γ_c	1.5		2
	Cross section width	b	0.3	m	2
	Cross section height	h	0.5	m	2
	Effective height	d	0.454	m	2
CONCRETE_#	The flexural tensile strength of the co...	f_{ctm}	4.8	MPa	2
CONCRETE_#	Characteristic compressive strength of ...	f_{ck}	50	MPa	1
	Design value of concrete compressive str...	f_{cd}	53.3	MPa	2
	Characteristic tensile strength of rein...	f_{yk}	500	MPa	1
	Design tensile strength of reinforcement	f_{sd}	435	MPa	1
	Concrete cover	c	30	mm	2
CONCRETE_#	Ultimate concrete strain	ϵ_{cu}	2.6		2
- Design Results (Right):** Shows the design of longitudinal reinforcement according to Eurocode 2 (CSN EN 1992-1-1).
 - Concrete:** Design value of concrete compressive strength $f_{cd} = 53.3 \text{ MPa}$.
 - Concrete coefficients:** $\alpha_{ct} = 1$, $\alpha_{s1} = 1.27$.
 - Concrete strength $f_{td} > 50 \text{ MPa}$:** $\eta = 0.85$, $\lambda = 0.73$.
 - Reinforcement:** Design value of steel strength $f_{sd} = 435 \text{ MPa}$, Compressive strain $\epsilon_{s1} = 2.07 \text{ ‰}$.
 - Lever arm of internal forces:** For tension reinforcement $z_1 = 0.046 \text{ m}$, For compression reinforcement $z_2 = 0.211 \text{ m}$.
 - Maximum reinforcement compressive strain:** $\epsilon_{s2} = 1000000\%$.
 - Cross section dimensions:** $b = 0.3 \text{ m}$, $h = 0.5 \text{ m}$, $d = 0.454 \text{ m}$.
 - Effective cross section height:** $d = 0.454 \text{ m}$.
- Diagram (Bottom Right):** A schematic diagram of a rectangular cross-section showing internal forces, reinforcement, and geometric parameters like d_1 , d_2 , X , X_{c1} , X_{c2} , Z_1 , Z_2 , F_{s1} , F_{s2} , F_{c1} , and F_{c2} .

Web page [about Scia design Forms](#).

Web page about Nemetschek Scia: <http://nemetschek-scia.com/>

Forum about Scia Engineer with group about Scia Design Forms: <http://forum.nemetschek-scia.com/>

Used shortcuts:

- SDF – Scia Design Forms
- User - Scia Design Forms User application
- Builder - Scia Design Forms Builder application

Release notes for version 5.0

User application

- The new [Dialogue](#) in SDF Builder is now displayed in the User application as well;
- The choice of form is [extended](#);
- CLC file(s) can be opened by drag and drop to the User application;
- Possibility to exclude the Form name from the exported file (RTF and DOCX);
- The layout may be [zoomed](#);
- The Dialogue shows [Bookmarks](#);

Builder application

- New mathematical functions are included ([SUM](#), [AVERAGE](#), etc);
- Random number generator;
- Generalization of some variable types - some types have been converted to the [Object](#) type;
- [The text from table of variables](#) can be reused in the TEXT command;
- The functionality of [graphics](#) is enlarged by dimensions, arcs, ellipse, and more tools for better dynamically created graphic object in the source code;
- A totally new [Dialogue](#) - more component types, more options for display of variables, numerical and table input from the dialogue, better graphic design, possibility to [show or hide some part of the Dialogue](#);
- User-definition of the visible or invisible part of the layout by the [Collapsible blocks](#);
- [Custom data tables](#) - a function that allows to load data from an external XML file (the same type as for the Custom library) right from the source code;
- [Static class "Tools"](#) - e.g. annex dependent variables;
- Support for creating [Graphs](#) and figures;
- Support for creating [Tables in layout](#) - data can be organised in tables in the layout window;
- Command [TR](#) - for translations in Tables and Graphics
- Customisable user function and methods - the programming language of SDF has been extended, in order to work as standard object-oriented language; previously available simple commands still remain, yet users who want to have more opportunities can now create more.
- CLS file(s) can be opened by drag and drop to the Builder application;
- Pagebreak is visual component - it allows better layout definition;
- The new library for the [concrete cross section definition](#) (including different shapes, longitudinal reinforcement and stirrups);

Common settings

This chapter describes the common settings for both the User and Builder modules.

System requirements

Recommended system requirements for the Scia Design Forms installation:

Processor

Pentium IV - 3Ghz (recommended: CoreDuo2 3 Ghz or higher)

RAM

2 GB (recommended: > 4 GB)

Graphic card

256 MB, supporting OpenGL

Space on drive for projects and temp files

100 MB

Minimum resolution

1280 x 800

Operating System requirements

The Scia Design Forms application is based on Microsoft .NET Framework 4 Client Profile.

The operating systems from Microsoft (from OS Windows XP) are supported.

Which versions are tested by Scia test-team:

- Windows XP 32 bit
- Windows 2003 server 64 bit
- Windows 2008 server 64 bit
- Windows 2012 server 64 bit
- Windows 7 32/64 bit
- Windows 8 32/64 bit

Installation

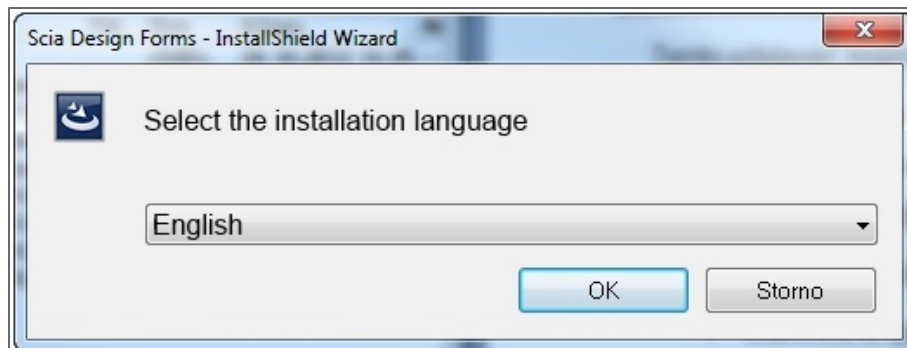
The installation process includes:

- Installation of Scia Design Forms;
- Installation of FlexNET Tryout protection;

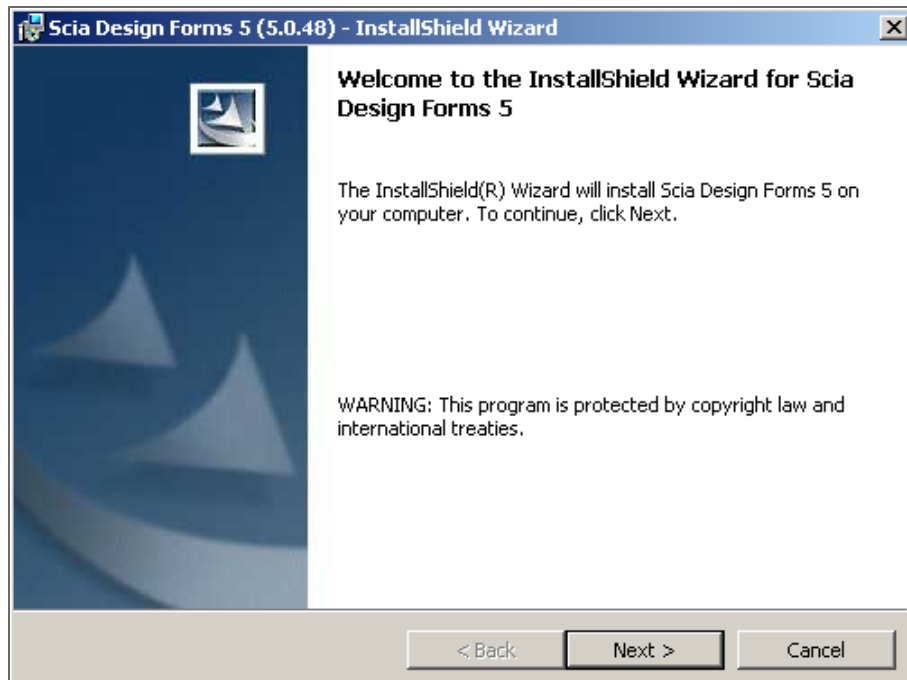
Scia Design Forms Installation

Some images are in CZ because of the system language.

1. Download and run the file SciaDesignForms_setup.exe (file name may vary depending on version)
2. Select the installation language from the dialogue:

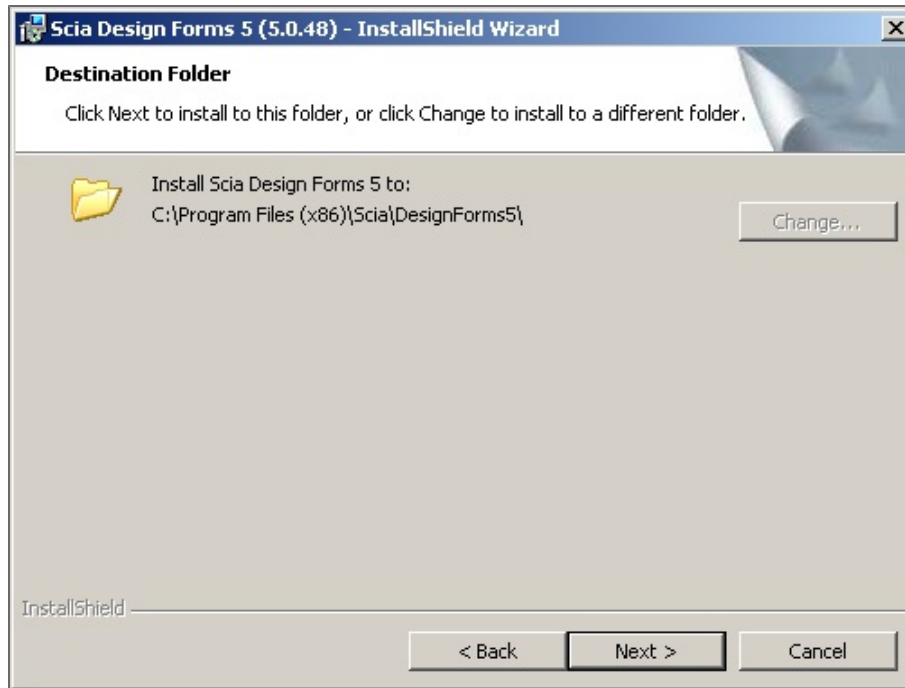


3. When welcome screen appears,



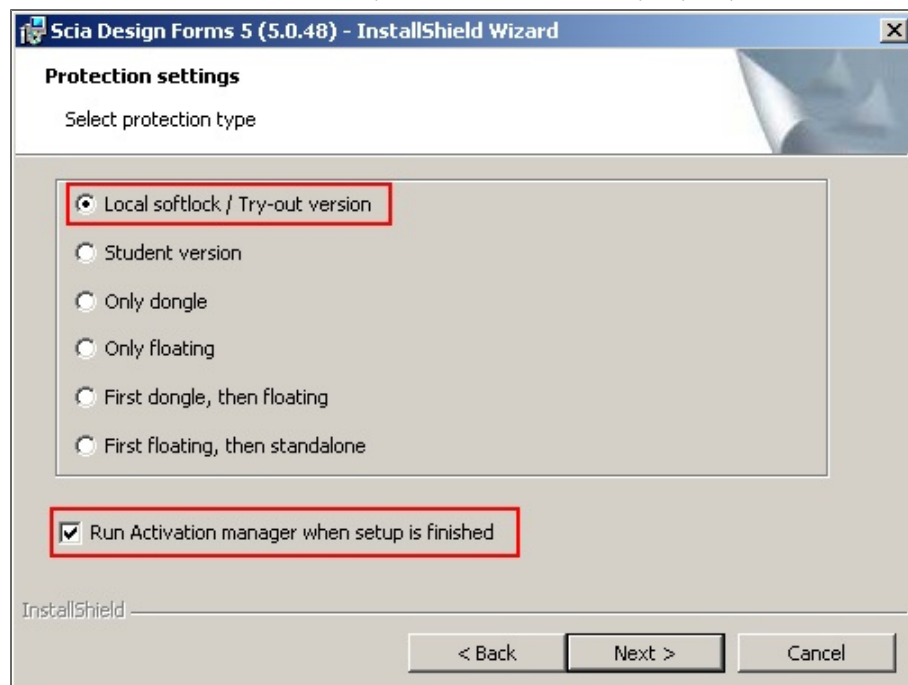
continue with [Next].

4. Select target installation folder. By default, the application is installed in X:\Program Files\. We recommend keeping the default installation path.



Continue with [Next].

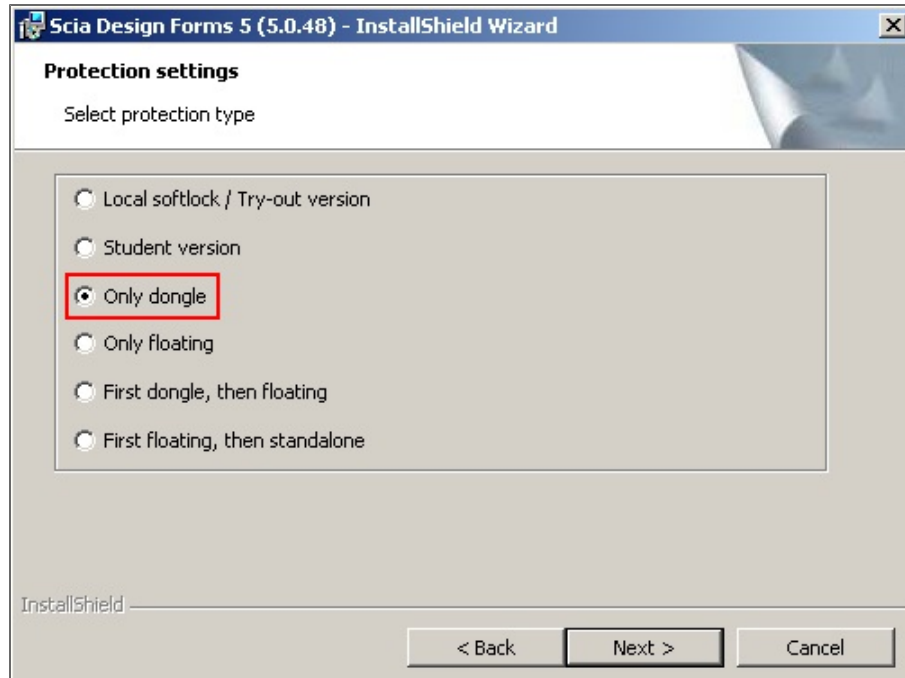
5. Protection settings - set the input parameters for protection:
 - a. for a commercial licence which is protected by local software protection or a tryout (demo) version:



Activate the Local software protection / Tryout version.

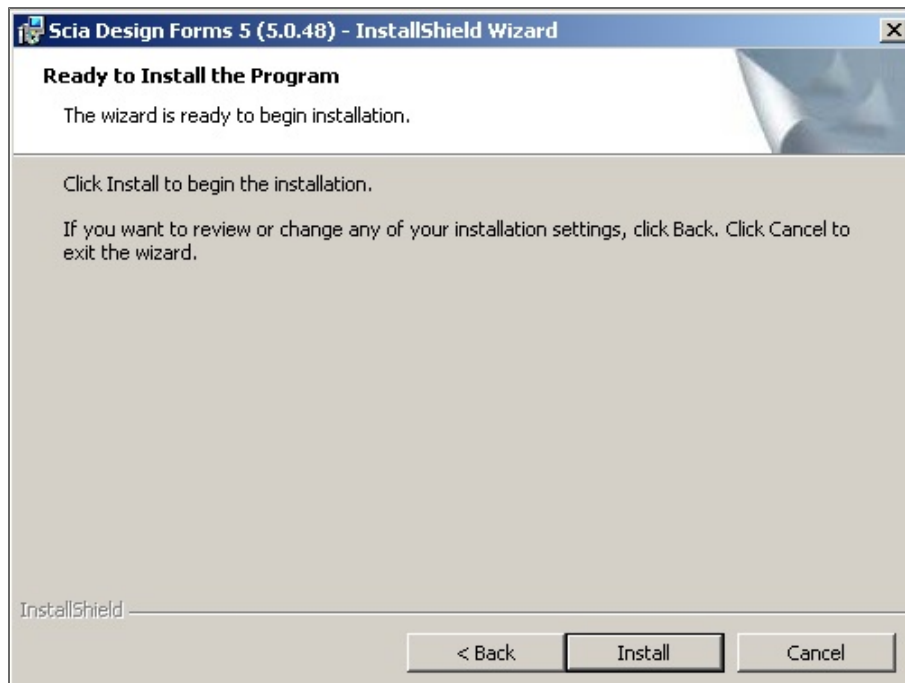
After completing the installation, do not change the setting (checkbox) for starting the Protection dialogue.

- b. for a commercial licence which is protected by a local key (dongle):



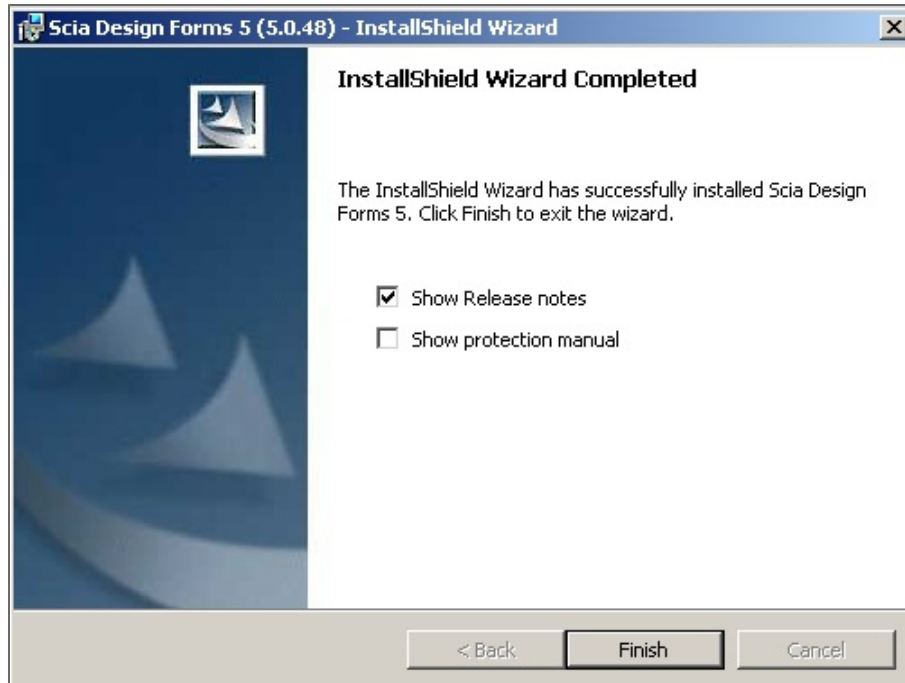
Continue with [Next].

6. The previous screen displays the selected parameters for the installation; click [Install] to start the installation process.



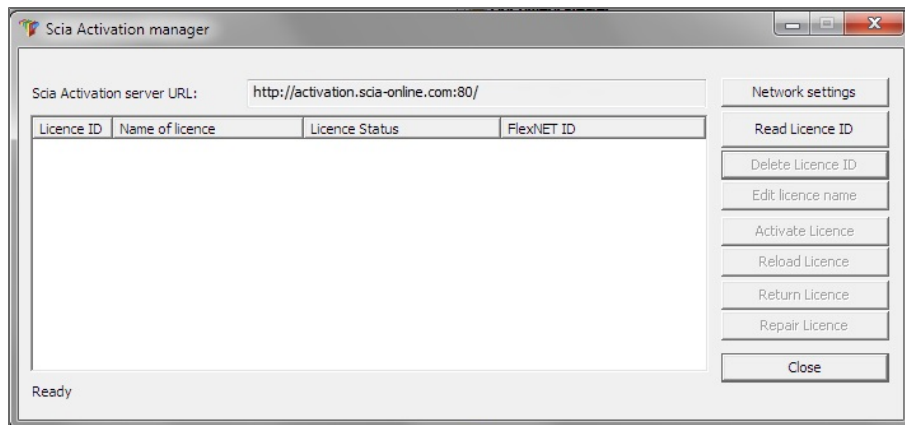
7. Continue by following the instructions for a specific protection type, as described in the following chapters.

8. The final screen displays the selected installation parameters. Click [Finnish] to finish the installation process



Installation of software protection (including tryout)

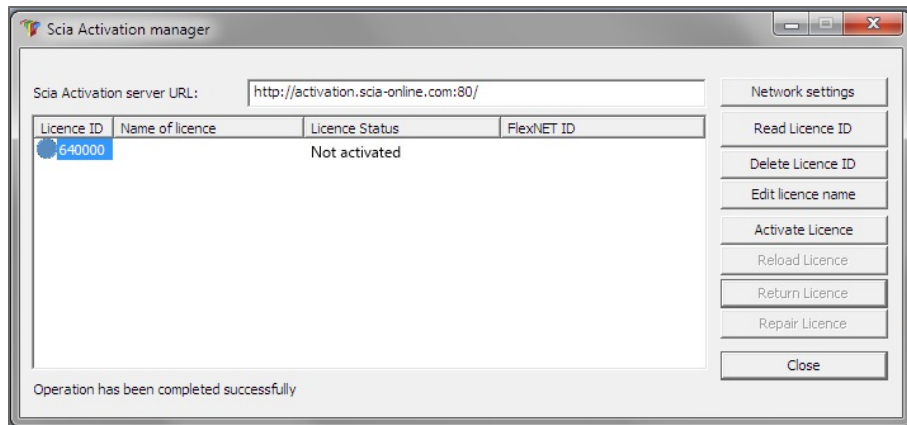
1. The **Scia Activation manager** dialog appears on your screen, at the end of the installation procedure.



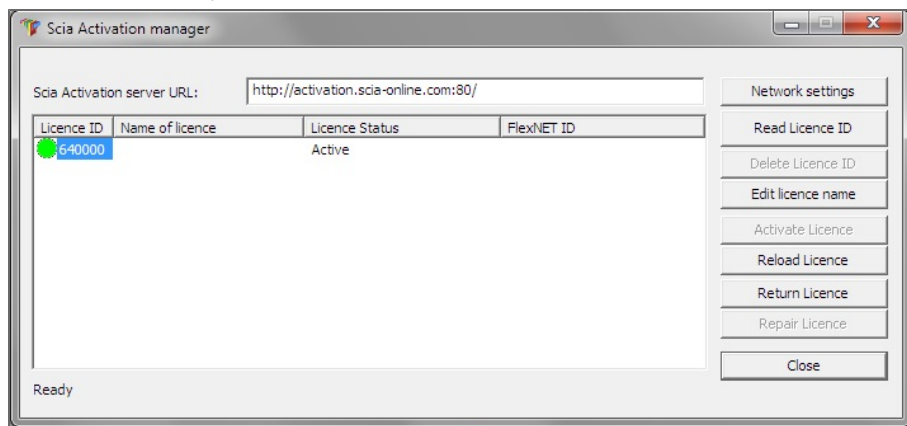
If this would not be the case, go to Windows Start menu > All Programs > Scia Licence Server > Activation Manager, and start the application. A shortcut with icon should be available on your desktop as well.

2. You received a *.LID file by e-mail, containing your authorization code. Download this file to your hard disk.

- Click **[Read Licence ID]** in the Scia Activation manager, and select the SCIAxxxxxx.LID file (xxxxxx representing your licence number). Click **[Open]**. Your licence number appears in the list.



- Select the licence number and click **[Activate licence]**. The circle in front of your licence number turns green when the licence has been correctly activated.



Click **[Close]**.

If you want to use the licence file on another computer, it is necessary to first deactivate it on the current one:

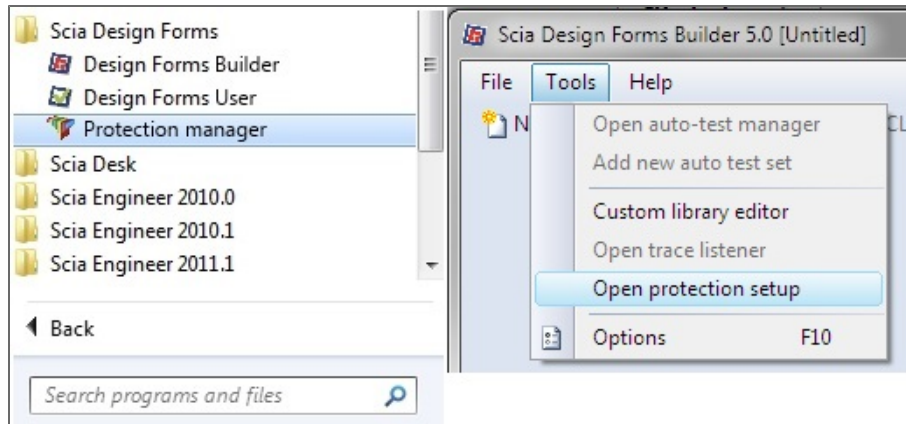
- Run the **Activation manager**.
- Select the licence number and click **[Return licence]**.
- Click **[Close]**.

Protection

Protection settings in Scia Design Forms

1. If the protection settings are correctly defined during installation, both application (USER and BUILDER) should run without further effort from the user. In case of problems with protection, the protection settings can be reviewed through the following procedure:

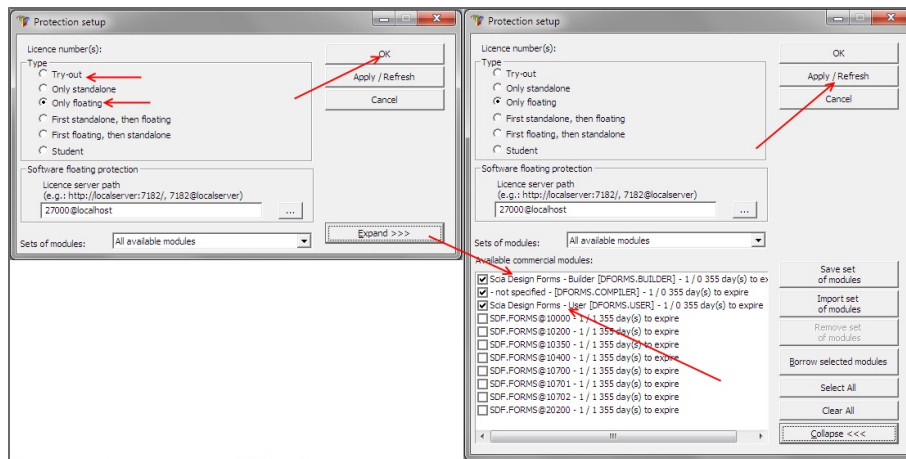
Run the protection setup from the Start Menu - All Applications > Scia Design Forms (or go to SDF Builder(User) / Tool / Open protection setup).



2. The protection settings should display a Tryout or an Only floating license. If this setting is different, select the correct one and use Apply.

At the bottom the commercial modules in your license are displayed.

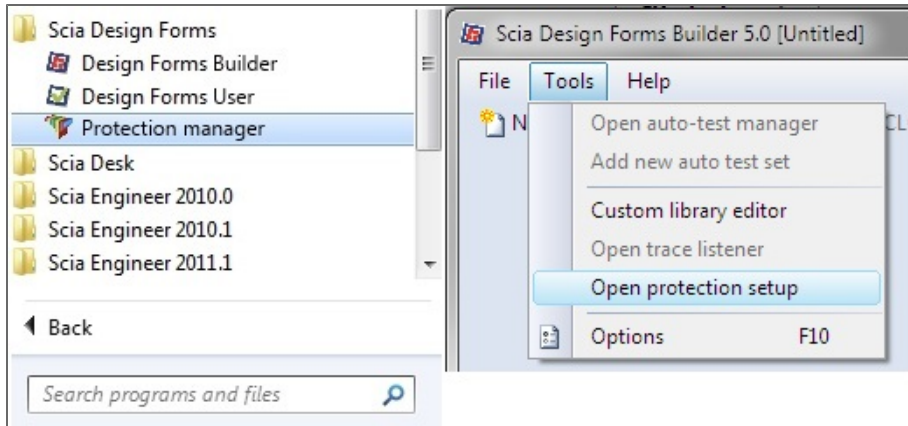
Modules which will be used must be checked. Any change must be confirmed by pressing the Apply and OK button.



Installation of protection by local key

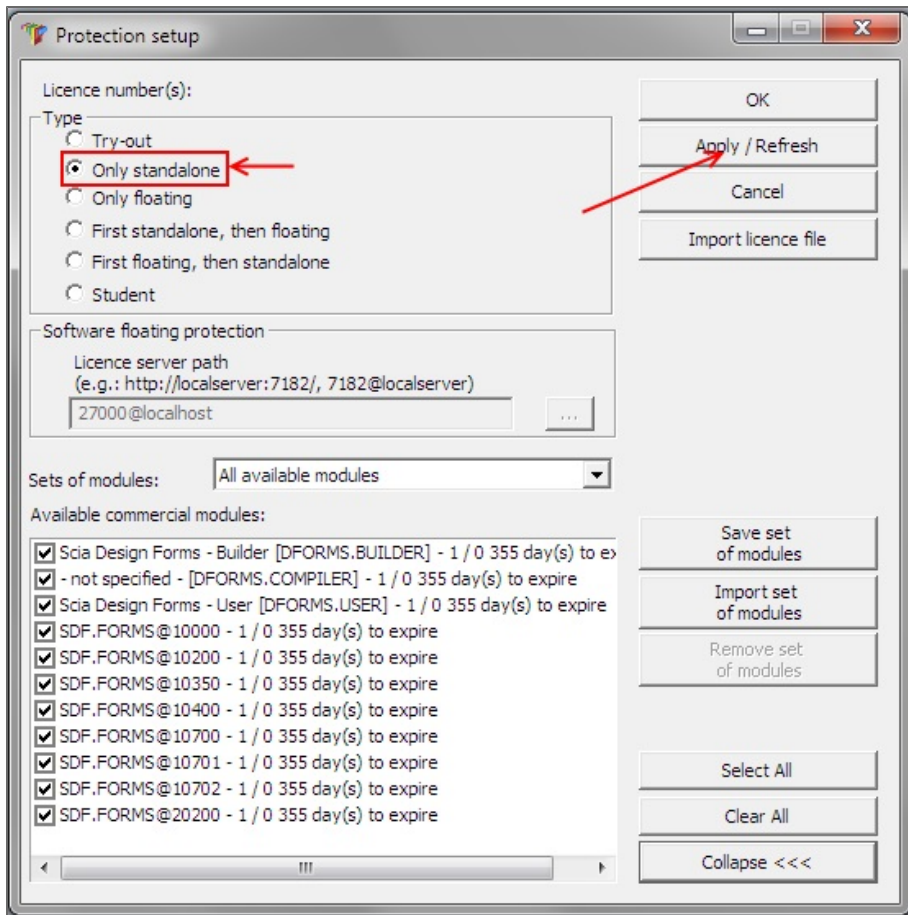
1. The license ID is saved in the hardware key (dongle). The license itself will be automatically loaded from the Nemetschek Scia server and other license files are not required.
2. If the Standalone license is checked during the installation, the Sentinel Protection is installed also. Connect your hardware key to the USB port.

3. Start the protection setup from the Start menu - All applications > Scia Design Forms (or go to SDF Builder(User) / Tool / Open protection setup).



4. Protection settings should be set to Standalone; if selected otherwise by default, select Standalone license type and click Apply.

Use import license. The current set of commercial modules will be loaded from Nemetschek Scia server.



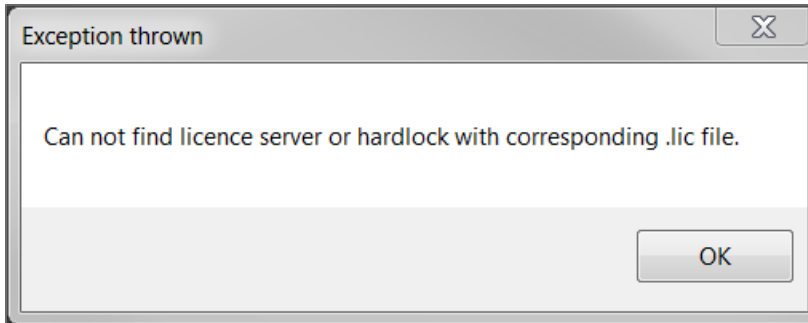
Online mode is required for loading commercial modules!
 If you do not have internet connection, ask you supplier for the LIC file.
 When the button Import license is used and the dialogue window is opened, select your license.
 The list of available modules is displayed in the bottom part.
 Check which modules will be used. If you make any changes, confirm it by Apply and OK.

Find more about Scia protection on [Scia Engineer webhelp](#).

Licence activation - hardlock (dongle), network (floating) licence

Scia Design Forms (and all additional applications) should run correctly without further intervention.

However, in case of a protection problem, you will receive next message when starting the application:



Go to Windows Start menu > All Programs > Scia Design Forms x > Protection Setup SDFx (x representing the current SDF version). Open the **Protection Setup**, check the selected protection type and change it if required. Click **[Apply / Refresh]** and close with **[OK]**.

First issue handling - hardlock (dongle) licence

1. Connect the hardware key (dongle) to your pc, and activate your internet connection.
2. In the **Protection setup**, click **[Import licence file]**. The licence file will be loaded from the Nemetschek Scia server.
3. Click **[OK]** and start the application.
4. In case the protection problem persists, please consult the [Protection page](#) of the Scia webhelp.

First issue handling - network (floating) licence

On the server pc:

New installation

1. Install the FlexNET Scia Licence Server.
2. Activate your *.LID file in the **Scia Activation manager**.

Update

1. Reload your *.LID file in the **Scia Activation manager**.
2. Restart the FlexNET Licensing Service.

On the client pc:

1. In the **Protection setup**, check the licence server path (port number@server name).
2. Click **[OK]** and start the application.
3. In case the protection problem persists, please consult the [Protection page](#) of the Scia webhelp.

Licence types

Try - out

This licence type is meant for testing, it has an expiration date.

Student

This type of licence is meant for students and teachers. To obtain, an academical e-mail address is required.

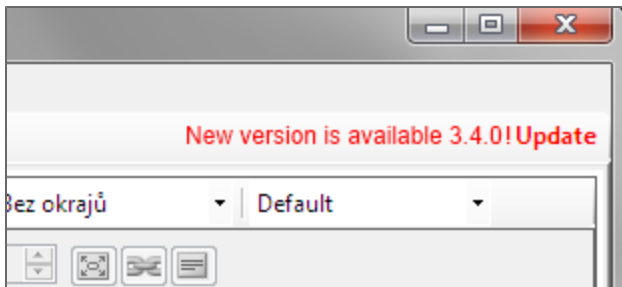
The student licence for Scia Engineer is used also as a student licence for Scia Design Forms.

Standard

This is the standard licence for all Scia Design Form customers.

Application update

A check for updates for the Scia Design Forms application is performed every time the programme is started. If a new version is made available the button "Update" appears in the top right corner.



When the user clicks the "Update" button, the application is closed and the update utility is started. The user has to confirm the start of the update function in some Windows versions.

The update utility loads and updates all required files and when finished, the Scia Design Forms applications are restarted.

Update of the application will not update existing forms (provided by Nemetschek Scia or by the user).

The update function is available in Main Menu > Help > Check for update.

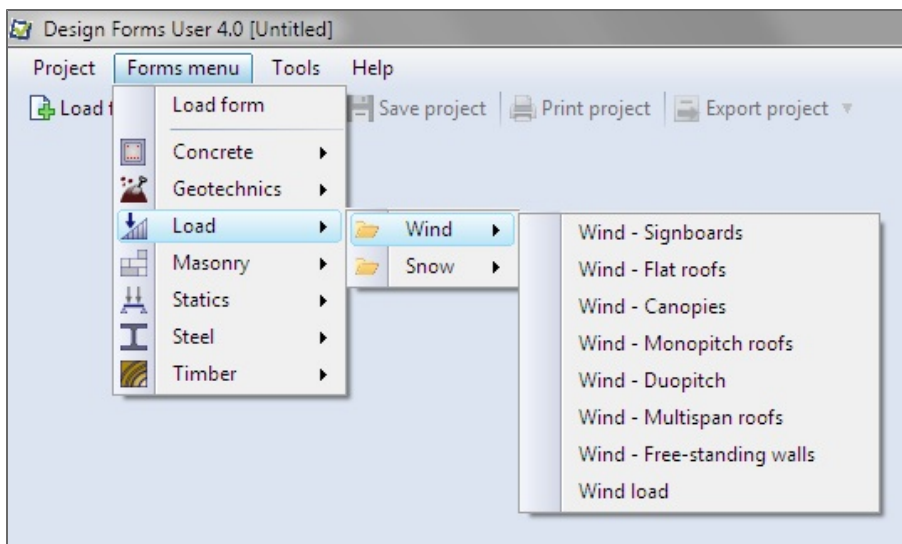
An update is performed for the User and Builder applications simultaneously. It is not necessary to update twice.

Language settings

The Scia Design Forms applications is delivered in Czech and English.

To change the default language:

- Go to Main Menu > Tools > Options;
- Use the combobox "Program language";
- Select the requested language;
- Restart the application;
- The groups and the names of calculations are translated also (USER application), the names of folders in Public Documents/Design Forms/Templates must be in English.



Terminology and file types

File type in version 3

CLC

This file is generated by the application Scia Design Forms Builder, version 3. The file contain binary data and the source code. It is unreadable for another application. CLC can be loaded to the User application (without editing) and to the Builder application (where editing is possible).

File formats in version 4

CLS

CLS files are generated by the Scia Design Forms Builder, version 4. CLS files can only be edited in the Scia Design Forms Builder, version 4 or higher. If this file is deleted, it cannot be substituted, the source code is lost, and it cannot be extracted from binary file (CLC).

This file is not readable by the User application.

The form cannot be edited without the source code in the CLS file! If you create forms, archive them carefully!

If the form is published using the binary format (CLC), it cannot not be edited further.

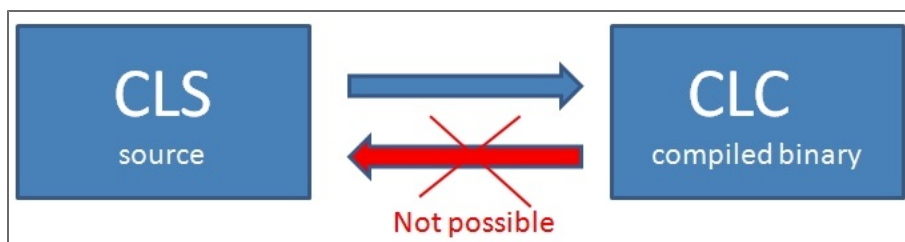
CLC

A *.CLC is a binary file (=unreadable for users), which is generated from the source code in the CLS file by pressing the 'Export CLC' button in the Scia Design Forms Builder, version 4 or higher. CLS can be used only in Builder application, it cannot be loaded in the User application.

CLC files contain the final definition of a form, input dialogue and Form Annotation.

If the CLS file is lost, no more changes in the form are possible. A CLS file cannot be extracted from a CLC file.

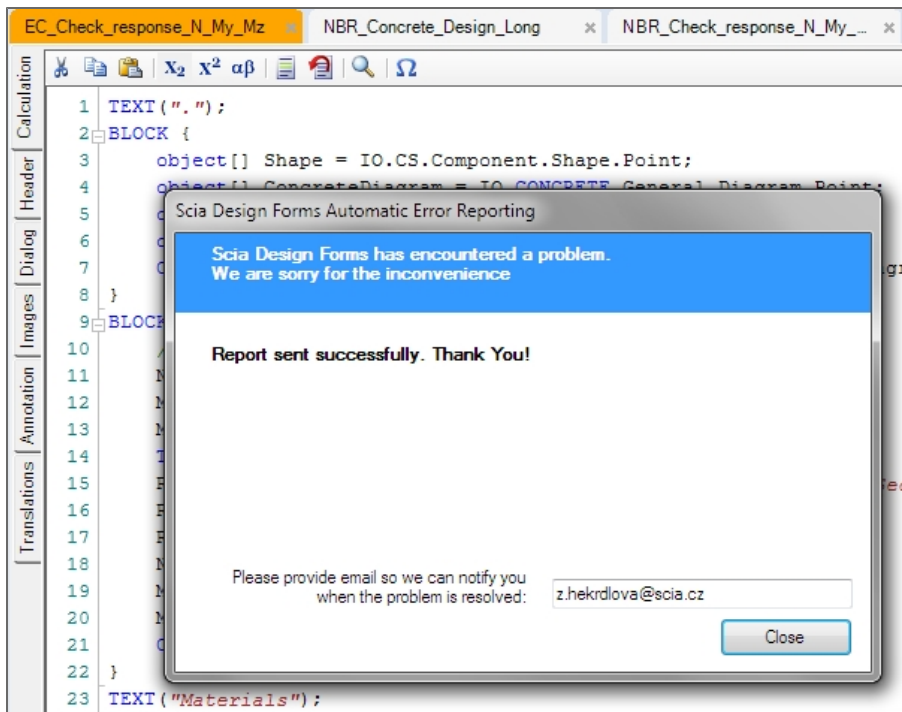
Binary CLC can be digitally signed and can be protected against unauthorized use.



Error reports

The Scia Design Forms application may experience exceptional circumstances in some extreme situations. In such cases, it is possible to send an error report to the developers.

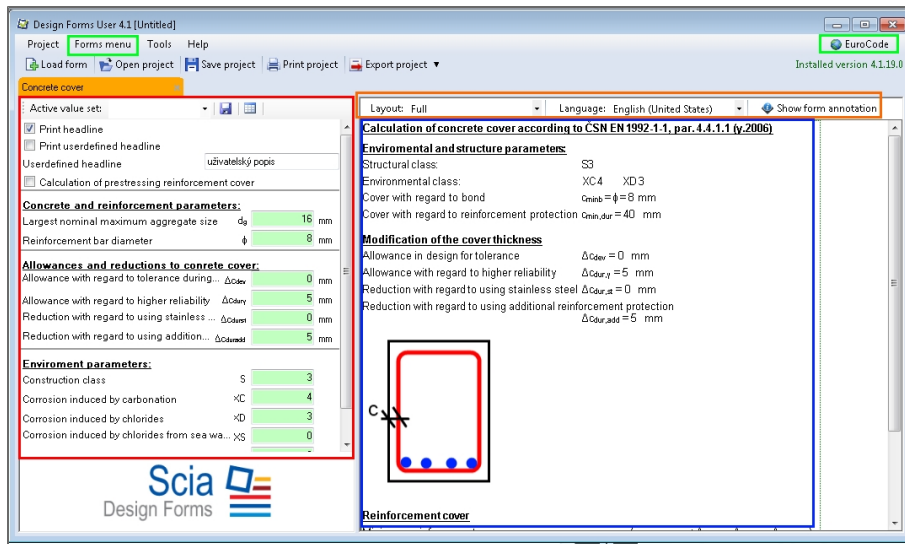
If this occurs, the application automatically displays a special Report dialogue:



- the user must fill in his email address and send the report by clicking on the button 'Send.'

User

The USER module is developed for the use of predefined forms. Some predefined forms are provided by Nemetschek Scia; other forms may be developed by users, university students or academics.



In the image above:

In green - menu for loading and adding forms to the project;

In red - input dialogue - for defining input variables before calculation;

In blue - output window - shows the report output in the selected layout;

In orange - settings for the language and layout for the report and displayed annotation to the calculation.

About the application

The USER application allows for the use of already existing forms (CLC files).

The user can load forms, enter values for the input variables defined in the form, and print or export results to MS Word, *.rtf, etc. Separate forms can be combined together as projects and saved on the drive.

The USER application is not intended for editing forms (changing the dialogue, equations or layouts); editing is possible in the BUILDER application.

The Scia Design Forms USER enables:

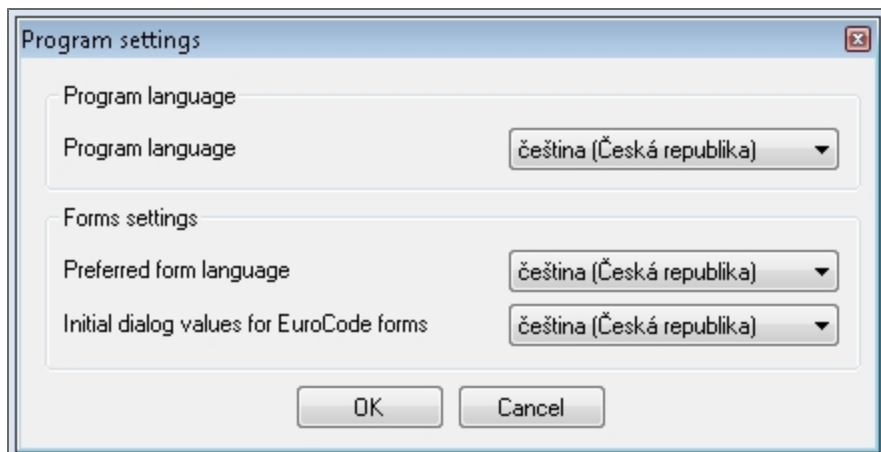
- the immediate execution of predefined calculation forms, form reuse and form grouping into projects;
- the production of well arranged reports, thanks to the clear presentation of used formulas, numerical substitutions and evaluation, including the possibility of logical verification of the obtained result (it satisfies / doesn't satisfy certain conditions);
- the selection of appropriate layouts, particularity out of a few levels of detail and out of available language variants;
- document print or export to text / pictures editors;
- optimization of the used cross-sections or materials due to the presence of built-in libraries;
- Exact and verifiable calculations according to the Eurocode, including explanations to entry values and calculation procedures;
- Helpful tables, graphs, etc. in the forms documentation;
- Quality protocol of common engineering calculations obtained with the minimal user effort.

web page about User:

http://sciadesignforms.com/en/sciadesignforms_user.html

User application settings

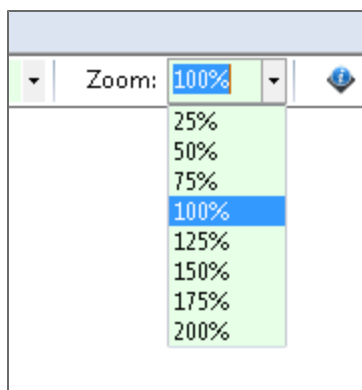
The USER application settings allow to select the default language, the preferred language for forms and preferred national annex for EUROCODE forms.



Zoom in the User application

The layout in the User application may be zoomed by the combobox.

The combobox is on the layout toolbar next to the Language.



Shortcuts in the User application

In the User application:

- CTRL+N - creates a new project
- CTRL+O - opens a project
- CTRL+S - saves a project
- CTRL+P - prints a calculation report
- ALT+X - closes the application
- CTRL+C - copies the calculation to the clipboard (by paragraphs)
- F1 - opens the help manual

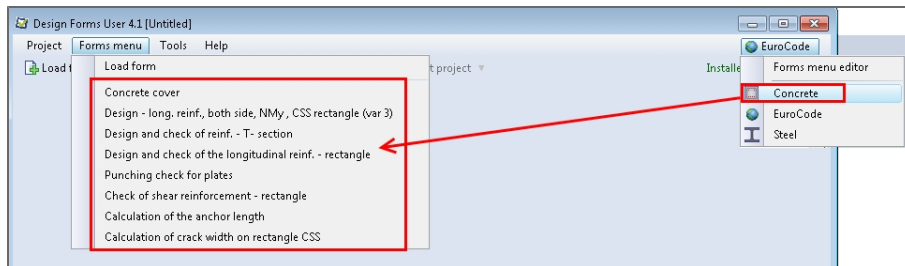
In the form list editor:

- CTRL + click on icon - selects the icon for the folder and all forms in it
- F2 - renames folder / sub folder
- INSERT - inserts a folder to the tree
- SHIFT + INSERT - insert a sub folder to the tree

The Forms menu

The forms menu allows to immediately select the form which should be loaded to the User application.

This menu can be edited by the Forms menu editor. Each set can be saved and loaded to the User application, in accordance with the user requirement.



List of calculations in Design Forms

File location

The standard installation of Design Forms comes with predefined calculations for concrete, timber, steel, and masonry structures and solutions for others structural problems. These calculations are saved in "C:\Users\Public\Documents\DesignForms\Templates\"

The USER module loads the file "Forms" on startup and displays its content in the main menu - Calculation.

When the user copies the calculations (*.clc) to the folder "Forms", it is added to the menu automatically after the application is restarted.

The folders in "Forms" can be changed; those changes are then displayed in the menu.

The list of calculations delivered by Scia

The links are continuously updated so that the most recently updated CLC files can be downloaded here.

Concrete:

- Concrete\Concrete cover.clc
- Concrete\Design of shear reinforcement.clc
- Concrete\Design of the reinforcement on the rectangular cross section.clc
- Concrete\Reinforcement design for a rectangular cross section.clc
- Concrete\Reinforcement design for T-section.clc
- Concrete\Punching check for plates.clc
- Concrete\Check of bending - CSS rectangle.clc
- Concrete\Calculation of the anchor length.clc
- Concrete\Calculation of crack width.clc

Timber:

- Timber\Check of steel connection.clc
- Timber\Check of tension.clc
- Timber\Check of rectangle - My+Mz.clc
- Timber\Check of rectangle -N+My+Mz.clc
- Timber\Check of rectangle - buckling.clc
- Timber\Check of shear.clc
- Timber\Connection timber-timber.clc
- Timber\Connection timber-steel.clc

Geotechnics:

- Geotechnics\Bearing capacity of foundation plate according to EN 1997-1, annex D.clc
- Geotechnics\Geostatic stress.clc

Geotechnics\Check of reinforced pad foundation - rectangle, simple.clc

Steel:

Steel\Check of circle tenon.clc

Steel\Check - N+My+Mz.clc

Steel\Check of shear and bending.clc

Steel\Check of simple bending.clc

Steel\Check of compression.clc

Steel\Check of shear.clc

Steel\Check of tension.clc

Steel\Check of bending with LTB.clc

Steel\Check of buckling on member.clc

Steel\Check of HS bolts.clc

Steel\Check of bolt connection cat. A - shear and deformation.clc

Statics:

Statics\Bracket - timber - F.clc

Statics\Bracket - timber - M.clc

Statics\Bracket - timber - q1.clc

Statics\Bracket - timber - q2.clc

Statics\Bracket - timber - q3.clc

Statics\Bracket - timber - q4.clc

Statics\Bracket - steel - F.clc

Statics\Bracket - steel - M.clc

Statics\Bracket - steel - q1.clc

Statics\Bracket - steel - q2.clc

Statics\Bracket - steel - q3.clc

Statics\Bracket - steel - q4.clc

Statics\Simple member - timber - 2F.clc

Statics\Simple member - timber - 2Fsym.clc

Statics\Simple member - timber - 3F.clc

Statics\Simple member - timber - 4F.clc

Statics\Simple member - timber - 5F.clc

Statics\Simple member - timber - F.clc

Statics\Simple member - timber - q.clc

Statics\Simple member - timber - q2.clc

Statics\Simple member - steel- 2F.clc

Statics\Simple member - steel - 2Fsym.clc

Statics\Simple member - steel- 3F.clc

Statics\Simple member - steel - 4F.clc

Statics\Simple member - steel - 5F.clc

Statics\Simple member - steel - F.clc

Statics\Simple member - steel - q.clc

Statics\Simple member - steel - q2.clc

Load - snowCZ:

Load\Snow\Snow - Drifting at projections and obstructions.clc

Load\Snow\Snow - Overhanging the edge of a roof.clc

Load\Snow\Snow - Monopitch roof.clc

Load\Snow\Snow - Duopitch roof.clc

Load\Snow\Snow - Multi span roof.clc

Load\Snow\Snow - Roof abutting and close to taller construction works.clc

Load\Snow\Snow - Cylindrical roof.clc

Load\Snow\Snow - Loads on snowguards and other obstacles.clc

Load\Snow\Coefficients for the snow load.clc

Load\Snow\Snow load.clc

Load - snowSK:

Load\Snow\Snow - Drifting at projections and obstructions.clc

Load\Snow\Snow - Overhanging the edge of a roof.clc

Load\Snow\Snow - Monopitch roof.clc

Load\Snow\Snow - Duopitch roof.clc

Load\Snow\Snow - Multi span roof.clc

Load\Snow\Snow - Roof abutting and close to taller construction works.clc

Load\Snow\Snow - Cylindrical roof.clc

Load\Snow\Snow - Loads on snowguards and other obstacles.clc

Load - wind:

Load\Wind\Wind - Signboards.clc

Load\Wind\Wind - Flat roof.clc

Load\Wind\Wind - Canopy.clc

Load\Wind\Wind - Monopitch roof.clc

Load\Wind\Wind - Duopitch roof.clc

Load\Wind\Wind - Multi span roof.clc

Load\Wind\Wind - Free-standing walls.clc

Load\Wind Wind load.clc

Masonry:

Masonry\Check of masonry in concentrated compression.clc

Masonry\Check of masonry in compression.clc

Masonry\Simplified check of shear perpendicular to wall plane.clc

Masonry\Simplified check of shear in wall plane.clc

Masonry\Simplified check of concentrated load.clc

Masonry\Simplified check of masonry in compression.clc

Form menu editor

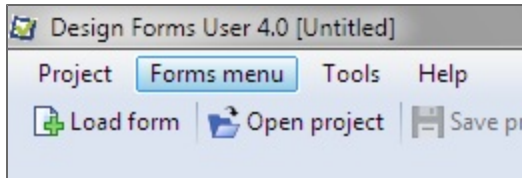
The SDF USER version 4 and higher allows to define more form lists which can be displayed in the main menu. The lists can be defined according to a logic groups - e.g. national codes, material National annex etc.

The form list can be adapted or created by user in the list editor.

Those forms are immediately available, the user doesn't have to search them on the drive.

Forms menu

The list of forms is accessed from the 'FORM MENU' on the main toolbar. The list shows forms which can be added immediately to the current project.



The list in this menu is filtered by the button in the application top left corner. The user can find predefined lists there, grouped according to e.g. material. The default list name is displayed on the button itself.

The default list:



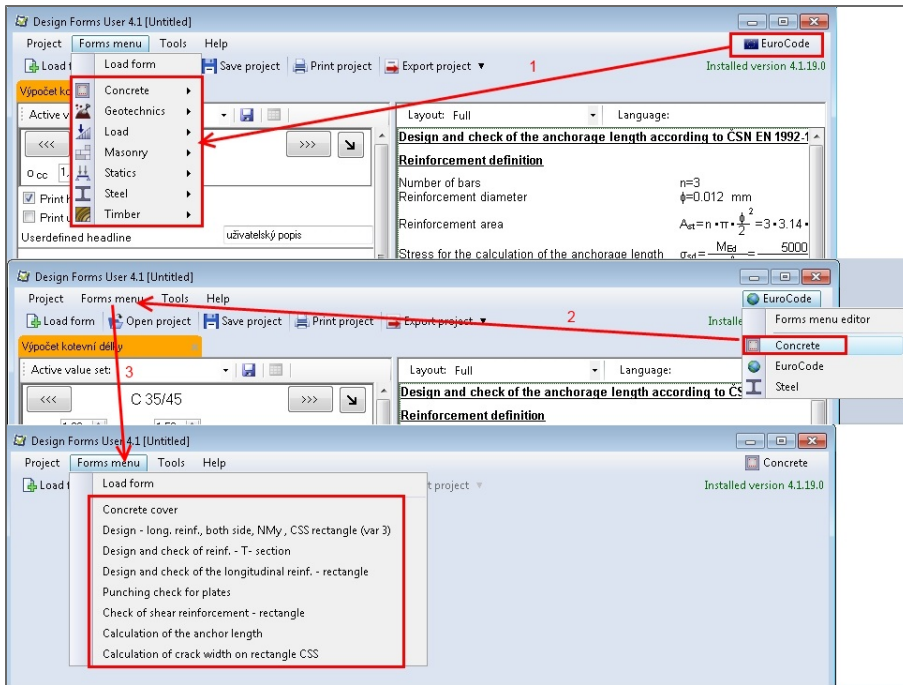
The list selection:



The forms menu selection

If the user wants to change the default list displayed in the Forms menu:

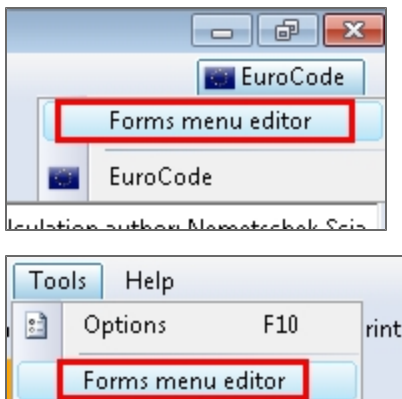
1. Click on the button showing the Default list (top left corner).
2. Select the requested Forms menu.
3. The Forms menu is automatically updated.



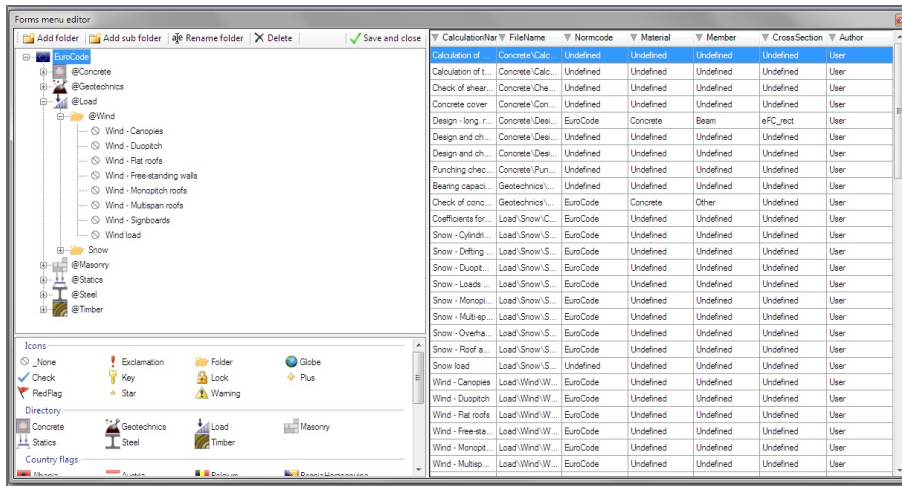
How to edit the menu

In the editor, it is possible to adapt the existing list or create a new one.

The editor can be opened by using the top right button or from Tools:



Editor:

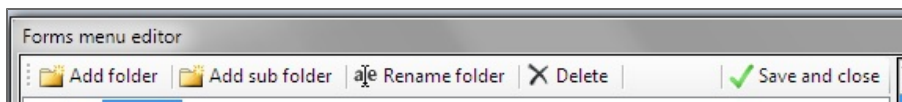


The left part displays a tree menu with defined lists of forms. The main items (the top level) are displayed in the top left corner. The inner items are displayed in the Calculation forms menu.

The right part displays the list of available forms which can be sorted by columns. Each column allows sort and filter (see the next chapter).

To adapt or create a new list:

1. Click on the button with default list (top left corner).
2. Select the second item "Adapt list of forms".
3. Adapt the list / create a new one - see next chapters.
4. Click "Save and close".



Changes will be lost if the dialogue is closed by using the cross icon (without saving)!

Add a folder to the tree

Select the folder in the tree. Use the button "Add folder." A new folder is automatically created at the same level.

Add subfolder to the tree

Select the folder in the tree. Use the button "Add subfolder." The sub folder is automatically created one level below.

Rename folder

1. Click twice on the item slowly.
2. Define a new name.
3. Confirm by Enter.

or

1. Select the item.
2. Click on button "Rename folder" (or use shortcut F2).
3. Define a new name.
4. Confirm by Enter.

Delete folder / subfolder / form

1. Select the item.
2. Click on the button "Delete" or use Delete from the context menu.

Add forms to the folder

1. Select the form (or more forms) in the left part.
2. Move them to the target folder.

Set icon for the folder / form

Each folder or form can have an icon.

1. Select the form or folder in the tree
2. Select the icon from the list on the bottom.

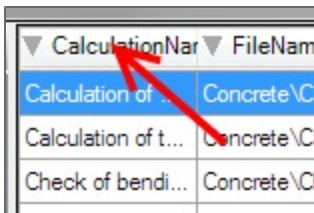
Multi-selection is not possible in the tree. If the same icon is required for the folder and all forms in it, hold CTRL key during icon selection.

Filter of forms in the left part

The list can be sorted and filtered.

Sort

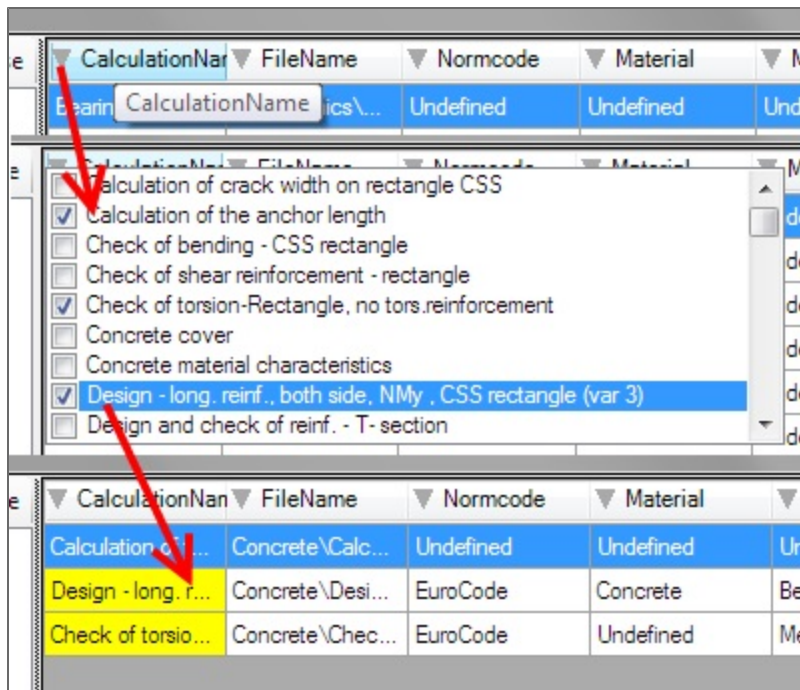
- click on the column header for sorting B->A, click for the second time to sort it A->B (click outside the arrow)



CalculationNar	FileNam
Calculation of ...	Concrete\C
Calculation of t...	Concrete\C
Check of bendi...	Concrete\C

Filter

- click on the arrow on the right part of the column header
- the list of available filters is displayed
- use the checkbox to mark selected filters
- click outside this dialogue
- the column with the filter is displayed in yellow, only filtered items are visible



Working with the dialogue

The Dialogue lets the user define the input values to be used in the calculation.

Concrete cover

Active value set: [icon] [icon]

Print headline

Print userdefined headline

Userdefined headline: uživatelský popis

Calculation of prestressing reinforcement cover

Concrete and reinforcement parameters:

Largest nominal maximum aggregate size d_g 16 mm

Reinforcement bar diameter ϕ 8 mm

Allowances and reductions to concrete cover:

Allowance with regard to tolerance during... ΔC_{dev} 0 mm

Allowance with regard to higher reliability ΔC_{dur} 5 mm

Reduction with regard to using stainless ... ΔC_{durst} 0 mm

Reduction with regard to using addition... ΔC_{duradd} 5 mm

Environment parameters:

Construction class S 3

Corrosion induced by carbonation x_C 4

Corrosion induced by chlorides x_D 3

Corrosion induced by chlorides from sea wa... x_S 0

Scia Design Forms

Inputting the values

Library items

The selection is done by using the left/right arrows or, when the cross-section library is used, the user selects the item from the detailed library view in the dialogue.

Boolean variables

Boolean variables can take up only TRUE/FALSE values.

The boolean variables are defined by checkboxes.

- Checked = TRUE
- Unchecked = FALSE

Text

Text variables (strings) are used as headlines, national code references, additional texts, etc.

Any text can be used as a value for string.

Numbers

Numerical variables are used in the calculation. The input is directly introduced by the user in the provided box (in green).

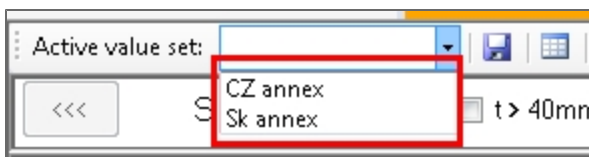
The SDF USER application does not distinguish between a dot and a coma as a decimal separator. The value of 123.456 is the same as of 123,456.

The field "Value" (in green) cannot contain any text characters. The only exception is the exponent, for example, "1e6" is the same as "1000000."

<<<	S 235 dle EC	<input type="checkbox"/> t > 40mm	>>>	↓
<<<	C 30/37		>>>	↓
α_{cc}	1,00	γ_c	1,50	
<input checked="" type="checkbox"/>	Print headline	National code <input type="text" value="ČSN EN 1992-1-1, §7.3.4 (20"/>		
<input type="checkbox"/>	Print userdefined headline	Userdefined headline <input type="text" value="uživatelský popis"/>		
<input type="checkbox"/>	The applied load is short term			
<u>Loading:</u>				
	Characteristic bending moment	M_{Ed}	<input type="text" value="30"/>	kNm
<u>Material characteristics:</u>				
	Modulus of elasticity of reinforcement	E_s	<input type="text" value="210"/>	GPa
	Secant modulus of elasticity of concrete	E_{cm}	<input type="text" value="33"/>	GPa

Default value sets and the *.DEFAULT file

It is possible to save one or more sets of default dialogue values for a form as a *.DEFAULT file. The active set is displayed in USER application on the toolbar above the Dialogue. This means that it is possible to load different value sets and calculate the form for each set. All sets are saved in one file (<Form_name>.DEFAULT), which is automatically loaded with the form.



It is possible to define the value set name with definition which annex it covered by this set.

Use special parameter to the name: CZ annex|CSN

- only "CZ annex" is saved as a name of the set

The list of annexes tags in .DEFAULT:

Austrian ONORM-EN ► ONORM

Belgian NBN-EN ► NBN

British BS-EN ► BS

Czech CSN-EN ► CSN

Dutch NEN-EN ► NEN

Finnish SFS-EN ► SFS

French NF-EN ► NF

German DIN-EN ► DIN

Greek ELOT-EN ► ELOT

Irish IS-EN ► IS

Luxembourgian LU-EN ► LU

Polish PN-EN ► PN

Romanian SR-EN ► SR

Slovak STN-EN ► STN

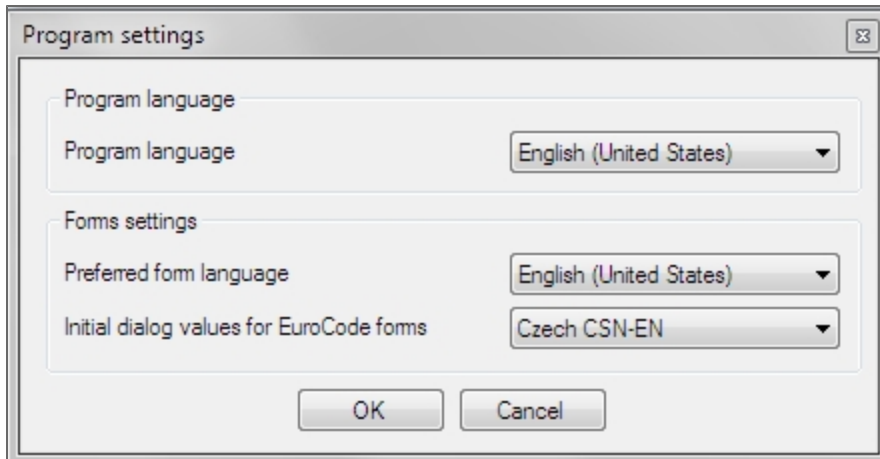
Slovenian SIST-EN ► SIST

Standard EN ► STD

User-default ► User - this is provided every-time when no parameter is used during saving the set

This menu allows to quickly select between saved sets. This is useful if the form takes into account parameters defined in national annexes of the EUROCODEs (certain values differ per country); also, if the same check may be performed (with certain modifications) for different member types (beam, column, rib); where differences in cross-section have to be taken into account (different dimensions for H, B, D, A, t_p , t_w ...), etc.

The preferred set of values settings is located in Main menu > Tools > Options.

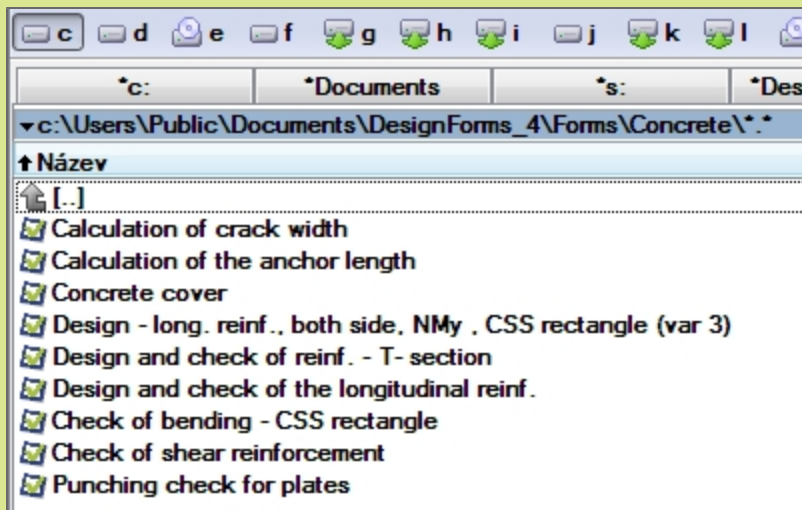


The default set can be selected and saved for each *.DEFAULT file containing multiple sets of values - more in chapter [Loading default values and Manager of default values sets](#).

The value sets are automatically saved during installation as *.DEFAULT files in C:\Users\\Documents\DesignForms_4\Forms\...

If the file *.DEFAULT is created for any form which is not saved (during installation or later) in C:\Users\\Documents\DesignForms_4\Forms\..., then the *.DEFAULT file is saved in the same folder as the form itself.

*.DEFAULT files for installed forms.



* DEFAULT files for forms which are saved elsewhere.

↑ Název	Přípona
↑ [..]	
Calculation of crack width	CLC
Calculation of crack width	CLS
Calculation of crack width	default
Calculation of the anchor length	CLC
Calculation of the anchor length	CLS
Calculation of the anchor length	default
Concrete cover	CLC
Concrete cover	CLS

Default sets, created for calculations according to EC in predefined forms, are delivered by Nemetschek Scia.

One set contains all values present in the dialogue - variables, checkboxes, strings and libraries. The sets are saved in the *.DEFAULT file under one another.

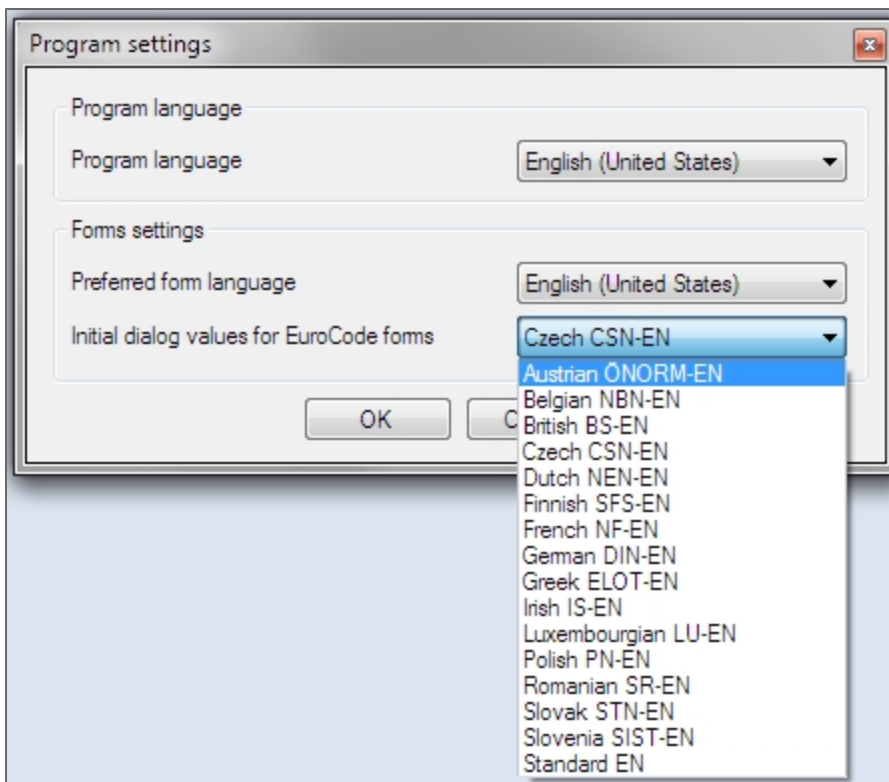
```

Concrete cover.default - Poznámkový blok
Soubor Úpravy Formát Zobrazení Nápověda
<Root>
  <User Description="Cz annex">
    <Variable Identifier="H1" value="True" />
    <Variable Identifier="NA" value="ČSN EN 1992-1-1, §4.4.1.1 (2006)" />
    <Variable Identifier="PrintCaption" value="False" />
    <Variable Identifier="Caption" value="uživatelský popis" />
    <Variable Identifier="B1" value="False" />
    <Variable Identifier="dlg-" value="16e-3" />
    <Variable Identifier="φ" value="8e-3" />
    <Variable Identifier="Δclddev-" value="0" />
    <Variable Identifier="Δcldury-" value="5e-3" />
    <Variable Identifier="Δcldurst-" value="0" />
    <Variable Identifier="Δclduradd-" value="5e-3" />
    <Variable Identifier="s" value="3" />
    <Variable Identifier="XC" value="4" />
    <Variable Identifier="XD" value="3" />
    <Variable Identifier="XS" value="0" />
    <Variable Identifier="XF" value="0" />
    <Variable Identifier="XA" value="0" />
  </User>
  <User Description="SK annex">
    <Variable Identifier="H1" value="True" />
    <Variable Identifier="NA" value="ČSN EN 1992-1-1, §4.4.1.1 (2006)" />
    <Variable Identifier="PrintCaption" value="False" />
    <Variable Identifier="Caption" value="uživatelský popis" />
    <Variable Identifier="B1" value="False" />
    <Variable Identifier="dlg-" value="16e-3" />
    <Variable Identifier="φ" value="8e-3" />
    <Variable Identifier="Δclddev-" value="0" />
    <Variable Identifier="Δcldury-" value="5e-3" />
    <Variable Identifier="Δcldurst-" value="0" />
    <Variable Identifier="Δclduradd-" value="5e-3" />
    <Variable Identifier="s" value="3" />
    <Variable Identifier="XC" value="4" />
    <Variable Identifier="XD" value="3" />
    <Variable Identifier="XS" value="0" />
    <Variable Identifier="XF" value="0" />
    <Variable Identifier="XA" value="0" />
  </User>
  <User Description="user 1">
    <Variable Identifier="H1" value="True" />
    <Variable Identifier="NA" value="ČSN EN 1992-1-1, §4.4.1.1 (2006)" />
    <Variable Identifier="PrintCaption" value="False" />
    <Variable Identifier="Caption" value="uživatelský popis" />
    <Variable Identifier="B1" value="False" />
    <Variable Identifier="dlg-" value="16e-3" />
    <Variable Identifier="φ" value="8e-3" />
    <Variable Identifier="Δclddev-" value="0" />
    <Variable Identifier="Δcldury-" value="5e-3" />
    <Variable Identifier="Δcldurst-" value="0" />
    <Variable Identifier="Δclduradd-" value="5e-3" />
    <Variable Identifier="s" value="3" />
    <Variable Identifier="XC" value="4" />
    <Variable Identifier="XD" value="3" />
    <Variable Identifier="XS" value="0" />
    <Variable Identifier="XF" value="0" />
    <Variable Identifier="XA" value="0" />
  </User>
</Root>

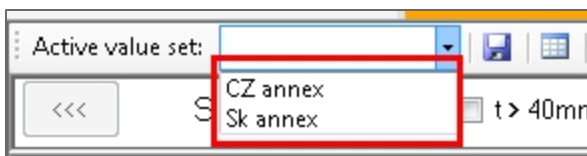
```

Loading default values and Manager of default value sets

When the calculation is loaded for the first time, the dialogue displays the default values for the calculation. The default values can be changed anytime. If the calculation supports more value sets, the user can select the preferred set in Main Menu > Tools.



The set can be defined by the combobox on the Dialogue toolbar.



More info about the file with default values to the dialogue - see chapter [Default value sets and the *.DEFAULT file.](#)

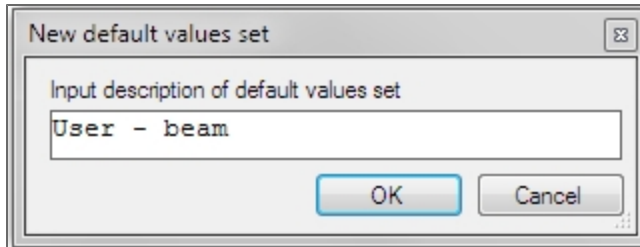
Create a new default set

If **new default values** should be defined (values displayed in the dialogue when the calculation is loaded to the project), then:

1. insert the required values to the Dialogue;
2. save the values by the button next to the combobox:



3. Define the set name:



4. Confirm by clicking 'OK.'

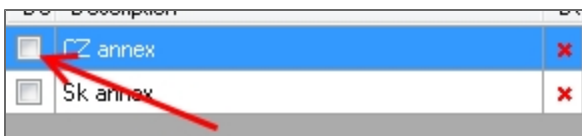
Manager functionality

If you want to mark any set as a **default set** for the form, use the Manager:

1. Open the Manager by the second button next to the combobox



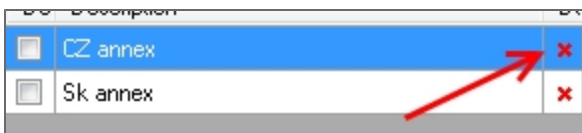
2. Select the set;
3. Check the checkbox;
4. Confirm.



This default set will always be used for the current form, as it has the highest priority. The rule for preferred default sets from Tools > Options will not be applied here.

If you want to **delete** a set of values, again use the Manager:

1. Open the Manager by the second button next to the combobox;
2. Select the set;
3. Use the button for deleting sets;
4. Confirm.



If you want to **rename** a set, use the Manager:

1. Open the Manager by the second button next to the combobox;
2. Select the set;
3. Click on the row and use shortcutkey F2;



4. Confirm.

If you want to **load** a set, use the Manager:

1. Open Manager by the second button next to the combobox;
2. Select the set;
3. Double-click on row;
4. Confirm.

The changes are loaded after clicking on the 'OK' button.

Default files

The default file determines safety factors, preferred cross-sections or library materials, calculation headlines, etc.

The default values are saved in <calculation_name.default>, in folder "C:\Users\<user>\Documents\DesignForms\Templates\".

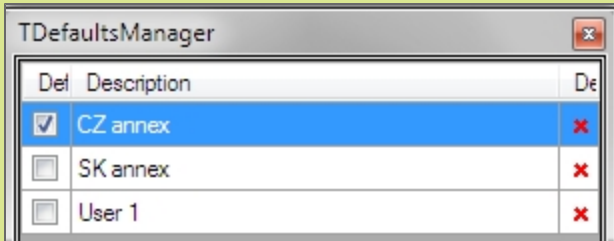
Název	↑ Přípona	Velikost	Datum	Atributy
[..]		<DIR>	22.11.2013 15:51	—
Connection timber-steel	default	2 051	26.09.2013 14:19	-a-
Connection timber-timber	default	2 299	26.09.2013 14:19	-a-
Check of rectangle on buckling	default	1 801	26.09.2013 14:19	-a-
Check of rectangle on My+Mz	default	1 909	26.09.2013 14:19	-a-
Check of rectangle on N+My+Mz	default	2 235	26.09.2013 14:19	-a-
Check of shear	default	1 509	26.09.2013 14:19	-a-
Check of steel connection	default	2 287	26.09.2013 14:19	-a-
Check of tension	default	1 505	26.09.2013 14:19	-a-

How to load default sets (priorities)

The workflow for loading a default set is the following:

1. If the file <form_name>.DEFAULT (sets of default values for the form to be displayed in the dialogue) doesn't exist, the USER application would load the values defined in the CLS file.
2. If any set is marked as 'Initial' in the Manager (the checkbox in the left column), it would loaded as default.

In this case, the CZ annex values is loaded as default set, as resulting from the described automatic process.



3. If the calculation is done according to the EUROCODE:
 - a. the system will check if the preferred annex is defined;
 - b. if yes, the calculation is loaded with it;
 - c. if no, the user is asked to define the set manually.
4. If the .DEFAULT file contains one initial set (marked in Manager), the forms is loaded with it.

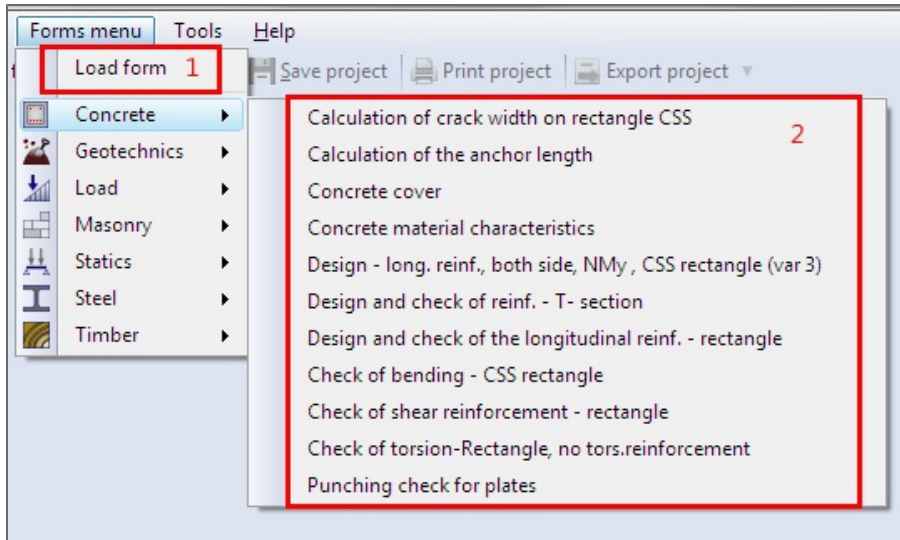
If the default set is not recognized, the Manager is automatically started and user is prompted to choose what set should be loaded.

Working with the form

The basic information about working with form is summarised in this chapter.

Inserting the calculation form in the project

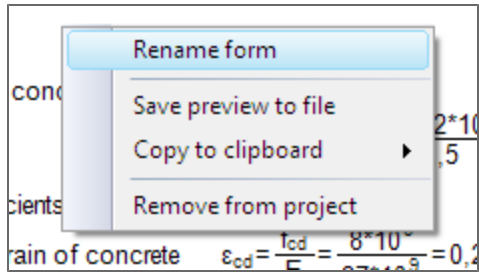
Go to the menu item 'Forms Menu'(2) and select the form from the thematic groups or select the *.cls file by using "Load calculation" (1).



The process of how to adapt the menu or add new calculations to a group is described in a separate chapter "[List of calculations in Design Forms](#)"

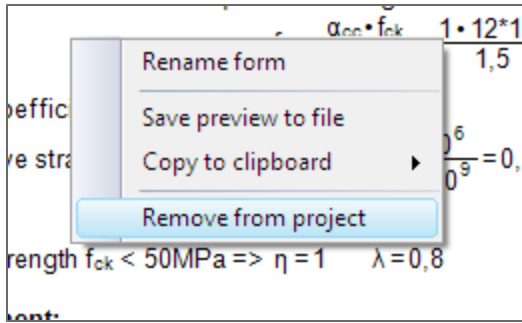
Renaming the forms

Display the pop-up menu and select the item "Rename calculation"; enter a new name and confirm.



Delete a calculation from a project

Select the function "Delete" from the pop-up menu; use the CTRL+F4 shortcut or the small cross on the tab.



Language selection

Calculations may have an arbitrary number of translations.

The language is selected by the combo box 'Language.'

Layout: Full	Language: English (United States)	Show form annotation
Calculation of concrete cover according to CSN	English (United States)	2006
Environmental and structure parameters	čeština (Česká republika)	
Structural class: S3	Slovak	

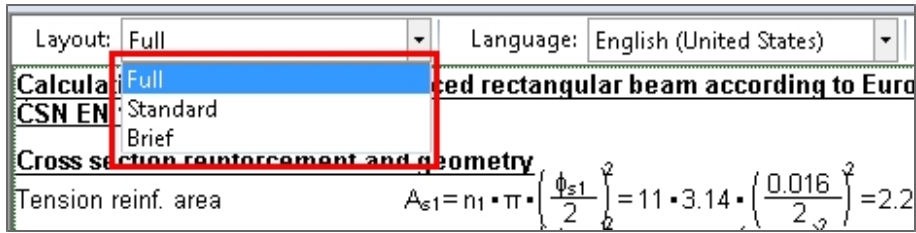
The translation is displayed as defined in the Builder*.CLS file. Common items that can be translated:

- The calculation report - text, descriptions and comments;
- Items in the calculation dialogue;
- The names of layouts;
- The annotation to a form.

See more about language definition in the Builder in the separate chapter - ["Calculation layout"](#).

Layout selection - output details

Each calculation may contain up to 6 layouts. The layout selection is done in the combobox above the component editor.



The layout definition is created in the Builder by the developer. Each layout contains the same calculation, but displayed in a different way. Differences between layouts are e.g. whether a formula is visible, whether images or descriptions are added. Layouts have no influence on the calculated results, only the graphical output is different.

Calculations delivered by Nemetschek Scia most often contain these layouts:

- Full - the most detailed output. All formulas, remarks, images, etc. are visible. Calculation steps are easily followed and controlled.
- Standard - all important formulas are visible. Some superfluous formulas, substitutions, remarks and images are hidden.
- Brief - only the most important results are visible. Remarks, images etc. are always hidden.

The layout definition is described in a separate chapter - [see "Calculation layout"](#).

Calculation output

Saving files to the calculation folder

The calculation can be saved in these formats:

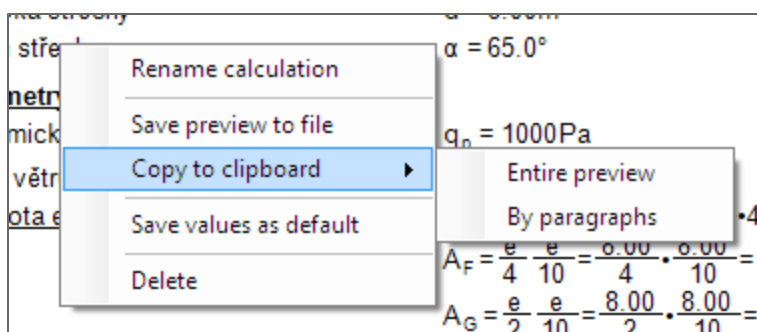
- Bitmaps: BMP, GIF, ICO, JPG, PNG, TIFF (when the bitmap is stretched, the file significantly loses quality.)
- Vector format: WMF (when the vector format is stretched, the file does not lose quality).

Copy the calculation to the clipboard

The fastest way to export output is by using the Windows clipboard.

There are two ways:

- Export the entire view - the whole calculation is exported to the clipboard as one image. This is useful for small calculations which fit on a single page.
- Export by paragraphs - hence, the output is saved in parts. The pagination of the text editor will be used. The calculation will be divided at the page ends (where the PAGEBREAK; command is used).
- Use left click in the layout and use the shortcut Ctrl+C on the keyboard; preview will be saved by paragraphs. The entire view can be copied only when using the menu (see below).



The files are saved to the clipboard in a WMF format; WMF files can be stretched without quality losses.

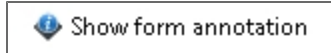
Form Annotation

The Form Annotation contains:

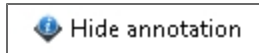
- The calculation header - defines the code which is used in the check, author, release date and a title for the calculation;
- Help for the calculation input data - includes brief description of the calculation and for example code tables, graphs, etc.

Find more about the Form Annotation in the section about the BUILDER application, in a separate chapter - "[Form Annotation](#)".

Display annotation in the USER application by using the button:



Hide annotation by using the button:



Working with projects

The standard file format for a Scia Design Forms project is *.CLP (calculation project). The project file contains a selection of calculations, their settings and default values.

Used forms, header, footer settings, default values are saved in the project file.

The binary copy (CLC files) of used forms is part of the CLP file. The form (CLC) stays the same and it will not be changed, even if a form is updated or deleted from the folder "c:\Users\\Documents\DesignForms\Tempates".

If an updated version of a form is needed in a SDF project (CLP file), then:

- remove the old calculation form;
- add the updated form;
- set the default values again.

New, open, save

The Scia Design Forms calculations can be grouped into projects. One project can contain an arbitrary number of calculations.

The application allows the user to open only one project.

New project

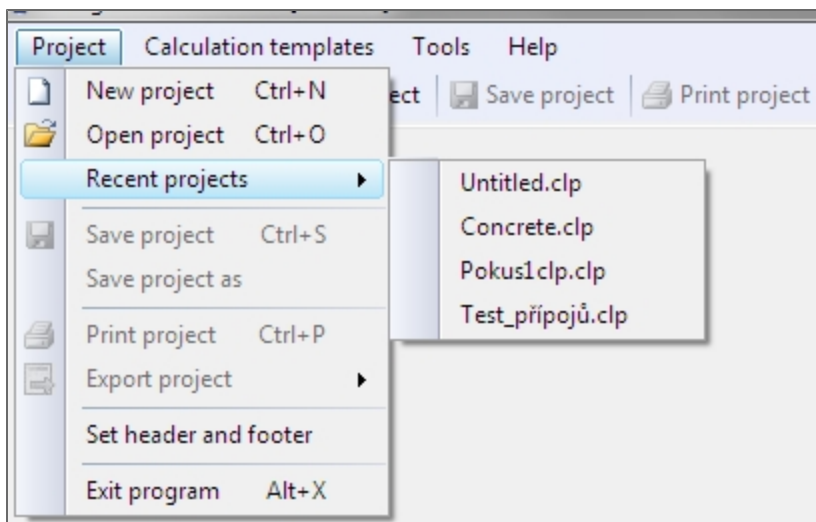
A new project is created automatically when the USER application is started. If a new calculation is needed during the working session, one of the described procedures can be used.

If the project is already open, the application will prompt the user to save the project. Then the new project will be created.

Open project

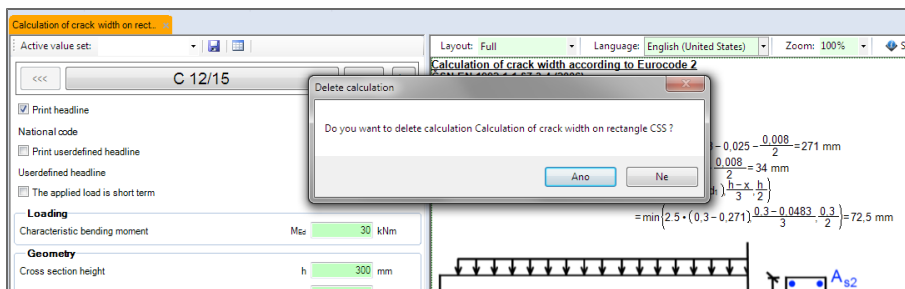
Go to Main Menu > Project > Open to open a saved project, or use the "Open project" button on the toolbar.

Recent projects are in Main Menu > Project > Recent projects.



Save project

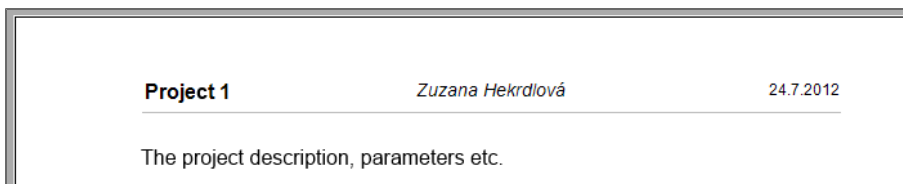
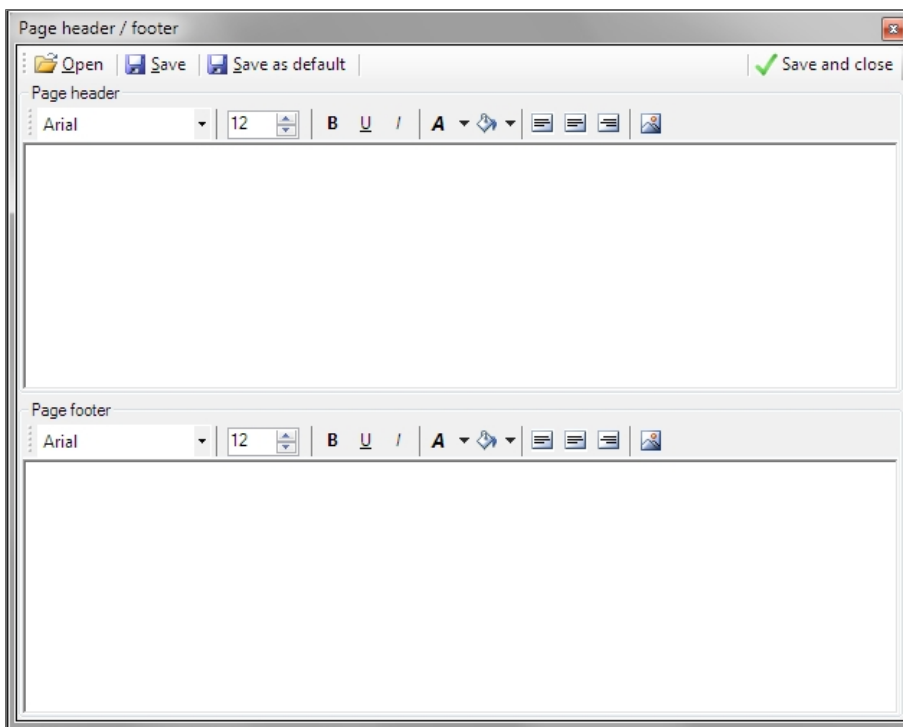
Go to Main Menu > Project > Save (or Save as) to save a project, or use the "Save Project" button on the toolbar.



Header/footer definition

To define a header and a footer, go to Main Menu > Project > Set Header and Footer.

The top text box defines the header, the bottom box refers to the footer. The font for these is defined in the standard dialogue menu.

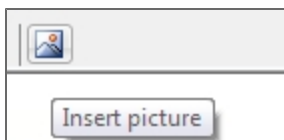


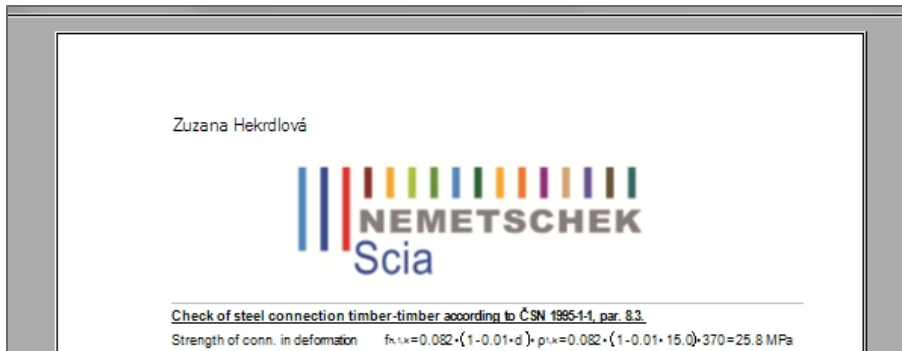
Working with pictures in headers / footers

Inserting the picture

The image can be inserted to the header/footer (e.g. logo). The formats BMP, GIF, ICO, JPG/JPEG, PNG, TIFF, and WMF are supported.

1. Use the icon "Insert picture";
2. Select the picture in the dialogue;
3. Confirm.





The inserted image is placed above the text, but the text is not automatically wrapped. It is necessary to move text next to the picture, in order for it to be visible.

Moving pictures

1. Select the picture; upon selection, it will be marked by a black rectangle.
2. Drag and drop the picture to the desired position.

Deleting pictures

1. Select the picture;
2. Use the 'Delete' Key.

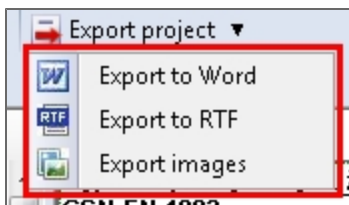
Header / footer active items

It is possible to add active items to the header or footer and change the content according to the document data:

- <PAGE> inserts page number;
- <PAGES> inserts the total number of pages in the document;
- <DATE> inserts the current date in format "DD.MM.YYYY"

The <PAGES> command is not correctly supported by the format RTF.

Project export



The output data can be exported to the following formats:

Export to .docx (MS Word)

The command exports the whole project to a MS Word document, version 2007 (*.docx).

Export to .rtf

The command exports the whole project to the RTF format (RichText).

RTF is a common format which is supported by numerous text editors (MS Word, OpenOffice, etc.).

The disadvantage of the RTF format is the larger size of files in comparison with DOCX.

Pictures export

Calculation reports can be exported to the following image formats:

- BMP - Bitmap;
- GIF - Compuserve GIF;
- ICO - Windows icon;
- JPG - JPEG format;
- PNG - Portable network graphics;
- TIFF - Tagged image file format;
- WMF - Windows metafile.

If batch export is required:

- use the button "Export Project" in the Main Menu > Project;
- select "Export images";
- In the dialogue "Export calculations":
 - Check calculations for export;
 - Define the destination folder;
 - Choose the exported file format;

Confirm by OK, and the output is saved.

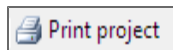
Export to .pdf

Use a virtual printer if a PDF export is required - e.g. PDF Creator, PDF Redirect etc..

Printing of help for the project is described in the chapter ["Printing projects"](#).

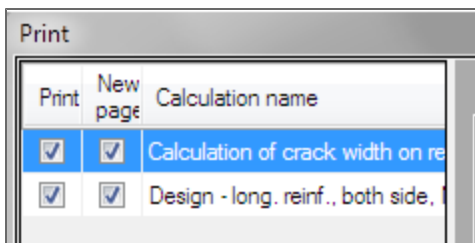
Printing projects

Go to the Main Menu > Project and use the function Print or use the 'Print Project' button on the toolbar.



Selection and settings of the printed calculations

The calculation list is displayed on the left side.



- Checkboxes allow to select whether a calculation will be printed or not.
- The second column defines if the calculation should start on a separate page or if it should be connected to the previous one. The calculation can be connected to the previous if it fits on the page. If not, the calculation is printed on a new page.

Print settings header/footer/borders

The page layout is defined by the checkboxes on the right side.

- Print borders - checked - the borders of pages, header and footer will be printed;
- Print header - checked - the header will be printed;
- Print footer - checked - the footer will be printed.

Print settings, print

The printer and print settings are defined on the right side.

- The button "Printer settings" displays the standard dialogue with settings.
- The button "Print" displays the standard dialogue for printer definition.

FAQ for the User application

Message when a new form is added

Error message: The variable ... cannot be loaded

Cause: The file DEFAULT doesn't contain the specified variable.

Solution: The form will be loaded without further problems; nevertheless, the inputs should be checked for wrong values.

Numeric variable with value NaN

Error message: NaN

Cause: A mathematical formula is not calculated due to an error argument of the function.

Example: Dividing by zero, Square root of negative number etc.

Solution: Check input values that participate in the mathematical formula.

Could not find file "..."

Error message: Could not find file "...".

Cause: The wanted file is not found in the defined location.

Solution: Check that the file is saved correctly.

The external file must be saved in the format *.CLC.

The external *.CLC file must be placed in the same folder as the opened CLS / CLC, or in the defined location.

Remark: If no path is predefined in the application settings, SDF searches for files in the folder where the current CLS / CLC is saved. If the current CLS is not saved, an empty path will be used!

Builder application

The Builder application allows the user to edit existing forms and create new ones.

The screenshot displays the Builder application interface for a concrete design. On the left, a sidebar lists various libraries: Steel section library, Steel library, Concrete library, Timber library, Bolts library, and Concrete section library. Below this is a table of design parameters with columns for ID, Description, Symbol, Value, Unit, Ex, and Pre. The table includes parameters such as the acting bending moment (M_{Ed}), coefficient of stress distribution (λ), coefficient of material strength (η), coefficient of action conditions (α_{cc}), and partial safety factors for reinforcement and concrete (γ_s and γ_c). On the right, a summary panel for a C 20/25 section shows design data: $\alpha_{cc} = 1.00$, $\gamma_c = 1.50$, and a checkbox for 'Print the headline of the design'. The 'Headline of the design' section lists: The acting bending moment ($M_{Ed} = 150$ kNm), The acting axial force ($N_{Ed} = -25.0$ kN), and Eccentricity of loading ($e_s = 0.00$ m). A 'Cross section dimensions' section lists: Cross section height ($h = 0.30$ m), Cross section width ($b = 0.20$ m), Effective distance for tension reinforcement ($d_t = 31.0$ mm), Effective distance for compression reinforcement ($d_c = 29.0$ mm), and Lever arm for tension reinforcement ($z_t = 0.119$ m).

ID	Description	Symbol	Value	Unit	Ex	Pre
Inter...	The acting bending moment	M_{Ed}	150	kNm	3	2
	Coefficient of stress distribution	λ	0.8		0	1
	Coefficient of material strength	η	1.0		0	1
Setu...	Coefficient of action conditions	α_{cc}	1.00		0	2
Setu...	Partial safety factor for reinforcement	γ_s	1.15		0	2
Setu...	Partial safety factor for concrete	γ_c	1.50		0	2
CS...	Cross section width	b	0.20	m	0	2
CS...	Cross section height	h	0.30	m	0	2

About the application

The BUILDER application allows for the creation of calculation forms (*.CLS files).

The user can create form, define and organise input dialogues, and export the calculation to a format which can be read by the USER application.

The BUILDER application allows users to change calculation forms - existing equations, dialogue, layouts, etc.

Scia Design Forms Builder:

- Creates new forms for arbitrary calculations; based on entering formulas and defining basic logical operations;
- Adapts existing forms to be used in the USER application;
- Converts calculations from spreadsheets into an application, where formulas can be displayed, substituted and calculated;
- Prepares reports based on a few predefined output levels of detail, following necessity for laconism or completeness;
- Supplements a calculation output with pictures (static and dynamic), charts, graphs, comments, code references etc.
- Is suitable for any calculation based on formulas;
- Allows the developer to quickly respond to code changes or new methods of calculation, thanks to possibility of form editing;
- Extend work presentations by the choice of more language version of forms
- Creates a builder community for developing new calculation forms.

Web page about the BUILDER:

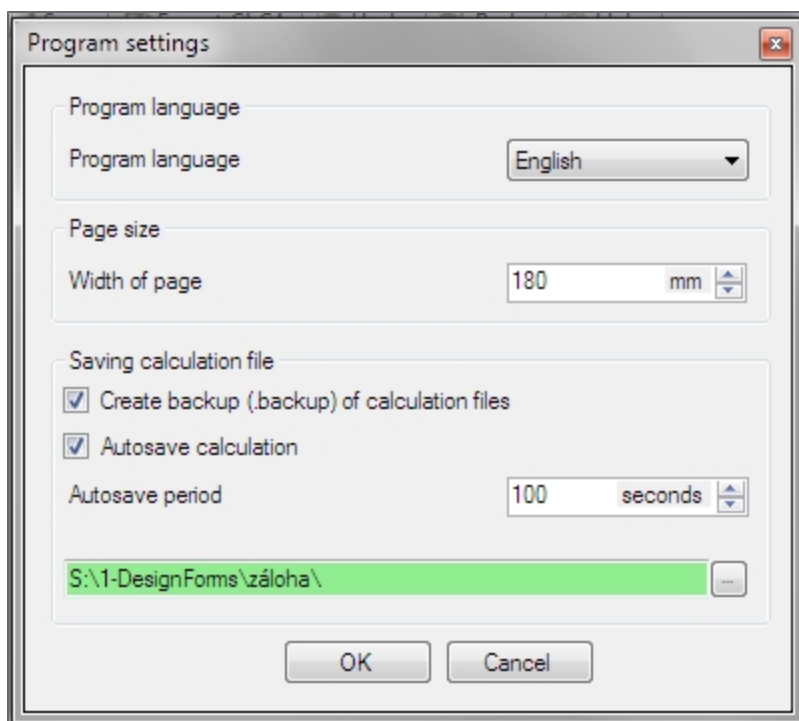
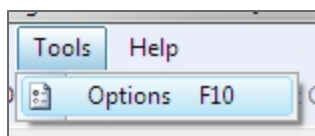
http://sciadesignforms.com/en/sciadesignforms_builder.html

Tools - Program settings

Application settings

Application settings are located on the Main Menu bar in the Tools Menu.

The options can also be opened by the keyboard shortcut F10.



For default language - see chapter [Common settings / Language settings](#)

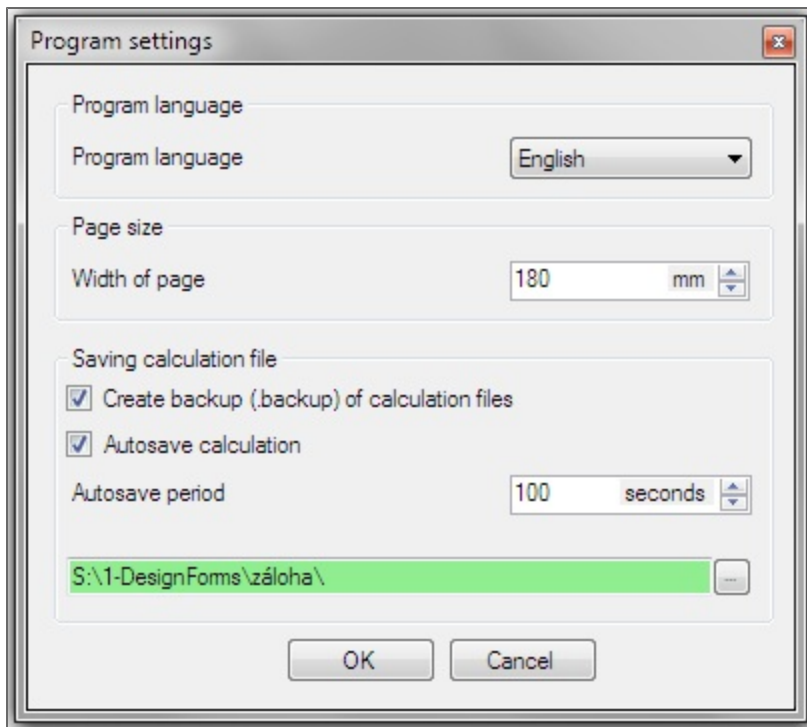
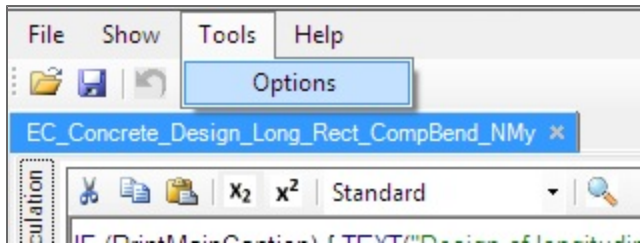
Page width settings

The page width is defined when creating a new form. The page width is indicated as a vertical line in the layout. It is recommended to set the page width with respect to the page offsets for printing - 1-2 cm.

Set the page width to 0.0 if the layout should be displayed without the vertical line.

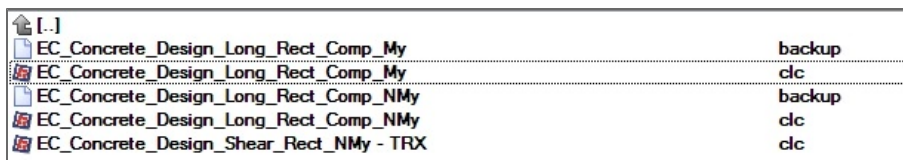
Data backup settings

The backup settings of the CLC files are in the application settings menu - Main Menu > Tools.

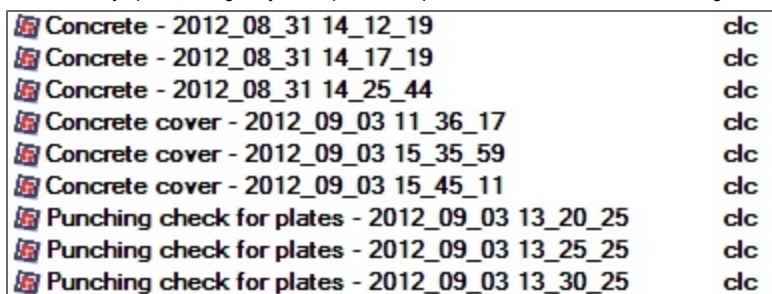


There are two kinds of file backup:

1. Backup - the application automatically saves the previous saved version of the file as a backup file. The file is saved with the .BACKUP extension to the same folder as the CLS file.



2. The application is furnished with an automatic saving option at predefined time intervals. This option allows the user to save currently open files regularly to the predefined path. The date and time of the saving are indicated.



Dialogue items:

-
- checkbox "Create backup of calculation files" - if this is checked, the backup of the original file is created when the file is manually saved. The backup function works separately from the Autosave Calculation.
 - checkbox "Autosave calculation" - if this is checked, the program will save the form automatically in predefined time intervals to the predefined folder. The name of the autosave contains date and time of the save.

The grid settings

The grid makes the graphical calculation output clearer. The grid is displayed horizontally in the top part of layout; the step is 10 mm.

Using zoom in the code editor

Zoom in code editor

The text size in the editor can be changed in two ways:

- hold the Ctrl key and roll the mouse wheel to set to the correct size;
- use the tool on the bottom bar:



Code editor shortcuts

The Code editor supports these shortcuts

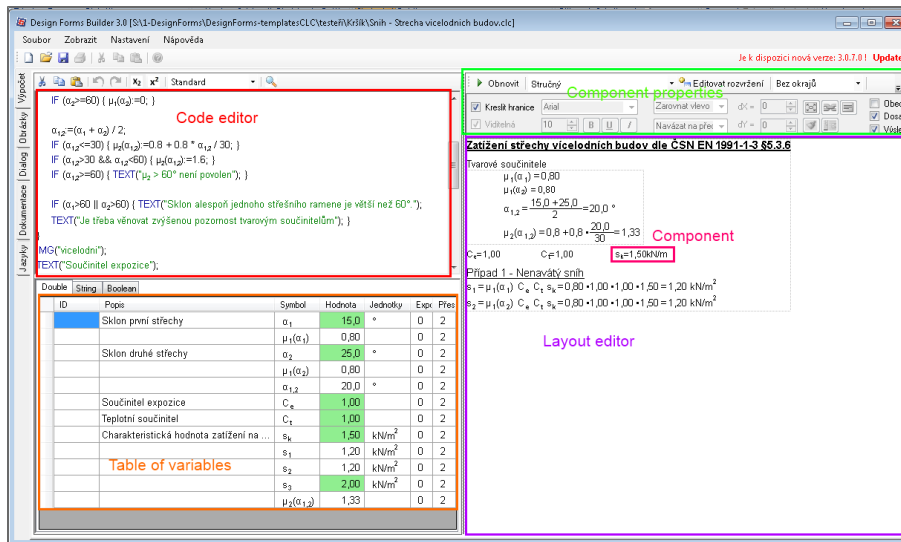
- Ctrl + A - select all
- Ctrl + C - copy to clipboard
- Ctrl + D - lower index / standard text
- Ctrl + F - find/replace
- Ctrl + G - greek / latin alphabet (switch)
- Ctrl + H - upper index / standard text
- Ctrl + V - paste from clipboard
- Ctrl + mouse wheel - change the size of the editor text
- F1 shows help
- F3 during find/replace search the next
- F5 - on Form Annotation tab -> Refresh of the web page, otherwise Refresh the calculation
- F9 Compile and save CLC
- F10 Program settings
- Alt active the work with block type - see the chapter Work with the block selection

The Layout editor supports these shortcuts

- Ctrl + B - component is bold (the result is bold in equation)
- Ctrl + U - component is underlined (the result is underlined in equation)
- Ctrl + I - component is italic (the result is italic in equation)
- Ctrl + 1 ... 8 - component is displayed according to the [predefined style 1 ... 8](#)

What is form?

The calculation components (equations, texts ...) are defined in the source code editor.



The layout of the components is defined by the Layout editor. The calculation output can be defined in one or more layouts. The differences are only graphical, the result is the same.

The Dialogue definition is in each calculation. The user selects the input data from the table with variables on this Tab. The user can add a library to the dialogue (cross section, material ...).

The calculation header contains the basic data about the calculation, author and used code. The header also contains the information used in Scia Engineer.

The calculation can be defined in one or more languages. The translation is defined on the Language tab.

The calculation contains a link to the web page with the Form Annotation.

It is not possible to load file, which was saved in the higher version of SDF.

Source code creating

The source code defines how will the calculation works:

1. displayed texts
2. order of equations - the process of calculations
3. conditions for calculation branches
4. which variables are needed as initial values - see table of variables

The screenshot shows a software interface with a source code editor on the left and a calculation result window on the right. The code is written in a programming language and includes logic for displaying text, calculating reinforcement area, and checking anchorage length. The result window shows a table of variables and detailed calculations for concrete tension strength, bond stress, and anchorage length.

Source Code (Left Panel):

```

1: IF (headline) TEXT ("Design and check of the anchorage length according to Eurocode 2");
2: TEXT ("Reinforcement definition");
3: IF (IF (condition)) TEXT ("Condition text");
4: double n = Math.PI;
5: TEXT ("Reinforcement definition");
6: IF (n > 0) {
7:   TEXT ("Number of bars");
8:   TEXT ("Reinforcement diameter");
9:   TEXT ("Reinforcement area");
10:  A_s = n * pi * (phi / 2)^2;
11:  TEXT ("Stress for the calculation of the anchorage length");
12:  sigma_s = M_ed / (z_s * A_s);
13:  TEXT ("Coefficient related to the quality of bond condition");
14:  TEXT ("Coefficients:");
15: }
16: TEXT ("Coefficients:");
17: TEXT ("n1=" & n1);
18: TEXT ("Coefficient of the bar diameter");
19: IF (phi <= 0.032) {
20:   TEXT ("value <= 0.032 mm");
21:   n2 = 1;
22: } ELSE {
23:   TEXT ("value > 0.032 mm");
24:   n2 = (132 - phi) / 100;
25: }
26: TEXT ("Coefficients:");
27: TEXT ("alpha1=" & alpha1 & ", alpha2=" & alpha2 & ", alpha3=" & alpha3 & ", alpha4=" & alpha4 & ", alpha5=" & alpha5);
28: PAGEBREAK();
29: IMG ("anchorage.png");
30: PAGEBREAK();
31: TEXT ("Design value of concrete tension strength");
32: f_tkd = alpha_s * f_tk;
33: TEXT ("Design value of the ultimate bond stress for ribbed bars");
34: f_bk = 2.25 * n1 * n2 * f_tkd;

```

Calculation Result (Right Panel):

Design and check of the anchorage length according to Eurocode 2
CSN EN 1992-1-1 §8.4.3, §8.4.4

Reinforcement definition

Number of bars	n = 3
Reinforcement diameter	φ = 8 mm
Reinforcement area	$A_{s1} = n \cdot \pi \cdot \left(\frac{\phi}{2}\right)^2 = 3 \cdot 3.14 \cdot \left(\frac{8 \cdot 10^{-3}}{2}\right)^2 = 151 \text{ mm}^2$
Stress for the calculation of the anchorage length	$\sigma_s = \frac{M_{ed}}{z_s \cdot A_{s1}} = \frac{50000}{0.3 \cdot 151 \cdot 10^{-6}} = 1105 \text{ MPa}$
Coefficient related to the quality of bond condition	n ₁ = 1
Coefficient of the bar diameter	n ₂ = 1
Coefficients:	α ₁ = 1, α ₂ = 1, α ₃ = 1, α ₄ = 1, α ₅ = 1

Design value of concrete tension strength
 $f_{tkd} = \frac{\sigma_{st} \cdot f_{tk}}{\gamma_c} = \frac{1 \cdot 1.3 \cdot 10^6}{1.5} = 0.867 \text{ MPa}$

Design value of the ultimate bond stress for ribbed bars
 $f_{bk} = 2.25 \cdot n_1 \cdot n_2 \cdot f_{tkd} = 2.25 \cdot 1 \cdot 1 \cdot 0.866667 = 1.95 \text{ MPa}$

Design of anchorage length:

Basic anchorage length
 $l_{b,req} = \frac{\phi \cdot \sigma_s}{4 \cdot f_{bk}} = \frac{8 \cdot 10^{-3} \cdot 1.11 \cdot 10^6}{4 \cdot 1.95 \cdot 10^6} = 1.13 \text{ m}$

assumption: f_{yk} is constant on the whole bar
is $\sigma_s \cdot z_s \geq 0.7$ => is SUFFICIENT

The designed anchorage length is:
 $l_{b,d} = \alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \alpha_5 \cdot l_{b,req} = 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1.13 = 1.13 \text{ m}$

Check of the anchorage length $l_{b,d} \geq l_{b,min}$

Tensile anchorage
 $l_{b,min} = \text{Max} \left\{ \begin{matrix} 0.3 \cdot l_{b,req} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.3 \cdot 1.13 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{matrix} \right\} = 0.34 \text{ m}$

Tensile anch. length must be bigger then 0.34 m => **is SUFFICIENT** => **0.34 m < 1.13 m**

Compression anchorage
 $l_{b,min} = \text{Max} \left\{ \begin{matrix} 0.6 \cdot l_{b,req} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.6 \cdot 1.13 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{matrix} \right\} = 0.68 \text{ m}$

Comp. anch. length must be bigger then 0.68 m => **is SUFFICIENT** => **0.68 m < 1.13 m**

Table of Variables (Bottom Left):

ID	Description	Symbol	Value	Unit	Preci
		m	3.14		2
	The tension bare diameter	φ	8	mm	2
	Number of reinforcement bars	n	3		0
	Reinforcement area	A _{s1}	151	mm ²	2
	Action bending moment	M _{ed}	50	kNm	2
	Inner lever arm of forces	z _s	300	mm	2
	Stress for calculation of an...	σ _s	1105	MPa	2

Orange part - if the H1 is fulfilled then the text is displayed as a calculation headline

Blue part - characteristics list and its calculation

Green part - displaying of image

Yellow part - table of variables

Code editor shortcuts

The Code editor supports these shortcuts

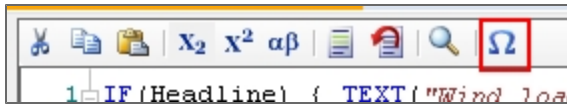
- Ctrl + A - select all
- Ctrl + C - copy to clipboard
- Ctrl + D - lower index / standard text
- Ctrl + F - find/replace
- Ctrl + G - greek / latin alphabet (switch)
- Ctrl + H - upper index / standard text
- Ctrl + V - paste from clipboard
- Ctrl + mouse wheel - change the size of the editor text
- F1 shows help
- F3 during find/replace search the next
- F5 - on Form Annotation tab -> Refresh of the web page, otherwise Refresh the calculation
- F9 Compile and save CLC
- F10 Program settings
- Alt active the work with block type - see the chapter Work with the block selection

The Layout editor supports these shortcuts

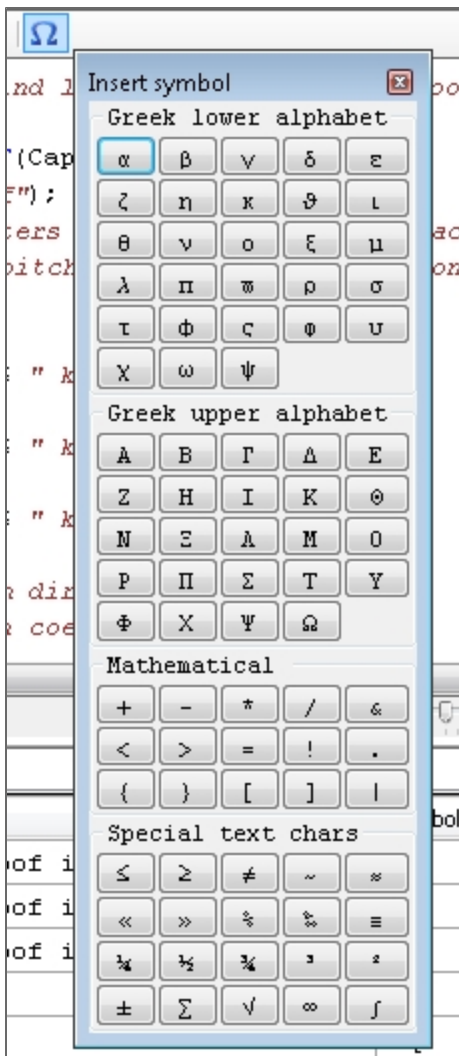
- Ctrl + B - component is bolt (the result is bolt in equation)
- Ctrl + U - component is underlined (the result is underlined in equation)
- Ctrl + I - component is italic (the result is italic in equation)
- Ctrl + 1 ... 8 - component is displayed according to the [predefined style 1 ... 8](#)

Table with special symbols for code editor

Some special symbols can be inserted by a table on the editor toolbar:



The button shows table which can be displayed while user works with editor:



When user clicks outside the table, to the application SDF, the table is displayed transparent. It is still displayed so it can be used when it is needed.

Insert symbol ✕

Greek lower alphabet

α	β	γ	δ	ϵ
ζ	η	κ	ϑ	ι
θ	ν	\omicron	ξ	μ
λ	π	ω	ρ	σ
τ	υ	ϕ	χ	ψ

Greek upper alphabet

Λ	B	Γ	Δ	E
Z	H	I	K	Θ
N	Ξ	Λ	M	O
P	Π	Σ	T	Y
Φ	X	Ψ	Ω	

Mathematical

$+$	$-$	$*$	$/$	$\frac{\square}{\square}$
$<$	$>$	$=$	$!$	\int

Font size:

Special text chars

$<$	$>$	\leftarrow	\rightarrow	\approx
-----	-----	--------------	---------------	-----------

Symbol	Value
$\frac{w_F}{w_H}$	-1.9
$\frac{w_G}{w_H}$	-1.0
w_H	-0.83

Source code of the calculation in the Code editor

Grouping parts of the code

The code parts in { ... } or between tags #region - #endregion can be packed and unpacked by the icon on the side. The function allows to hide code parts which are currently not needed for visualisation.

```
1 IF(headline) { TEXT("Design and check of the anchorage length according to Eurocode 2");
2   TEXT(NA); }
3 IF(PrintCaption) { TEXT(Captiontext); }
4 double n = Math.PI;
5 TEXT("Reinforcement definition");
6 IF(MEd > 0) {
16 TEXT("Coefficient related to the quality of bond condition");
17 TEXT("η1" & η1);
18 TEXT("Coefficient of the bar diameter");
19 IF(φ ≤ 0.032) {
26 TEXT("Coefficients:");
27 TEXT("α1" & α1 & ", α2" & α2 & ", α3" & α3 & ", α4" & α4 & ", α5" & α5);
28 PAGEBREAK();
```

The packed part is only hidden, it is still a component of the calculation. Any comment with description can be placed behind the tag #region.

Tags #region - #endregion is implemented only for packing code parts, it has no influence to the calculation itself.

Automatic highlighting of brackets

In the Code Editor brackets are automatically highlighted by pressing one of the brackets with the cursor.

Comment / Un-comment code parts

Current selection can be converted to a comment upon using the comment button on the editor toolbar (this will not affect the compile output). The second button (to the right of the first) un-comments the selected code part.

```
TEXT("Load extern:");
ExternM = LoadExternCLC("ExternM.CLC");
ExternM.DRAW(true);
```

Upper, lower index and Greek symbols

Upper index, lower index and Greek alphabet symbols can be obtained by using buttons on the editor toolbar, or by using the following keyboard shortcuts.



- lower index (CTRL+D)



- upper index (CTRL+H)



- Greek symbols (CTRL+G)

Find / replace

Find

Use CTRL + F or the context menu to start the functionality.

Procedure:

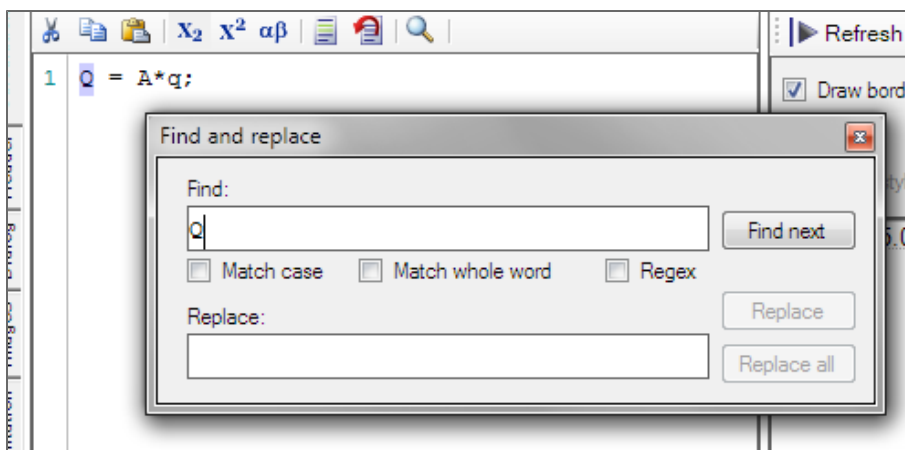
- Use CTRL + F or the context menu to display the 'Find and Replace' window.
- Write the text in the box provided.
- Use the button "Find next" to see other instances of the typed text in the code.
- User the button "Find all" to see all instances of the typed texts in the code.
- The text found is displayed in blue.

When searching for text strings in the code, it is possible to distinguish lower case and upper case letters, Greek symbols, to find whole words or only parts of words. Regex - provides searching the strings according to the syntax Regular Expression.

More about Regex (in English): <http://msdn.microsoft.com/cs-cz/library/hs600312.aspx>

Replace

- Check the checkbox "Replace with"
- Write the text.
- Write a new text in the provided text box.
- Use the button "Replace" to replace the instances of matching text one by one.
- User the button "Replace all" to replace all instances of matching texts at once.



Replace functionality respects the inserted upper/lower index inserted to the Replace field.

Source code syntax

General syntax rules:

- Variable names must fulfill the criteria for names;
- Each command must be ended by a semicolon (;);
- Brackets must be closed;
- The variables support both lower case and upper case letters - x_1 , X_1 , x_1 , x^1 are all recognized as different variables;
- Commands can be defined by lower case and upper case letters. The code editor will automatically convert them to upper case;
- Empty rows are ignored;
- Spaces outside the text strings are ignored;

- Numeric variables use dot instead of coma;
- Upper and lower index is accepted only for texts (command TEXT) and variable symbols;

```
TEXT("Compression area:"); TEXT("Required compression reinf. area");
Asy2req := (MEd1 - (λ * b * ξbal1 * d * η * fcd * 0.5 * (h - (ξbal1 * d)))) / (fyd * (z1 + z2));
```

Some strings are not accepted as a variable, as these have some specific function in Scia Design Forms.
For example, mathematical operations - MIN, MAX, LOG, etc, cannot be defined as variables.
This is also valid for other specific names - CONCRETE, STEEL, etc.

Work with the block selection type

The block selection type is activated by holding the SHIFT/ALT key .

This type of selection in the source code allows to edit more rows at the same time.

The block selection behaves in the same way as common selection. It allows for :

- Deleting
- Overwriting
- Copying to clipboard of multiple rows.

```
1 IF(headline) { TEXT("Design and check of the anchorage length according to Eurocode 2");
2   TEXT(NA); }
3 IF(PrintCaption) { TEXT(Captiontext); }
4 double π = Math.PI;
5 TEXT("Reinforcement definition");
6 IF(MEd > 0) {
7   TEXT("Number of bars");
8   TEXT("n = " & n);
9   TEXT("Reinforcement diameter");
10  TEXT("φ = " & φ);
11  TEXT("Reinforcement area");
12  Ast = n*π*[(φ/2)]2;
13  TEXT("Stress for the calculation of the anchorage length");
14  σsd = MEd / (zb*Ast);
15 }
16 TEXT("Coefficient related to the quality of bond condition");
17 TEXT("η1= " & η1);
18 TEXT("Coefficient of the bar diameter");
19 IF(φ <= 0.032) {
20   TEXT("Coefficients:");
21   TEXT("α1= " & α1 & ", α2= " & α2 & ", α3= " & α3 & ", α4= " & α4 & ", α5= " & α5);
22 }
```

Mathematical constants

- SDF has no predefined mathematical constants,
- Constants referred in equations are always converted to variables; these are taken from the equation and are automatically displayed in the table of variables. The developer can then type the value or include the respected variable in the Input Variables Dialogue (so that it can be defined in the USER application).

In $A = \pi * R^2$, the value of π has to be manually set to = 3,1415 in the table of variables or the dialogue in the USER application.

Table of variables

The SDF BUILDER generates variables in the Table of variables automatically, if these participate in equations, texts or conditions in the code. New variables are only added after compiling. The Table of variables should contain all the variables used in the calculation for the code to be executed. In certain cases, the variables need to be introduced in the Table of variables manually, either by using the available right-click menus in the Table of variables or through the code. Structured variables (vectors and matrices) and objects can only be defined through the code (see below).

If the application cannot recognize the variable type, the message "The given key was not present in the dictionary." is displayed in the layout. Manual definition is required in this case - see next chapter.

A green-coloured cell in the "Value" column in the Table of variables means that the number is not calculated in the code and the end-user should define it - it is not a result of any equation, but it is an input value for the calculations. The form developer should always include such variables in the dialogue (to be displayed in the SDF USER module) unless these are constants.

The SDF BUILDER does not automatically delete unused variables. These stay in the Table of variables even if they are no longer referred to in the code. Unused variables have no effects on the speed of form execution. Therefore, these can remain in the table without this deteriorating the form quality. Alternatively, it is possible to either delete them from the Table of variables, or to execute the [Purge Function](#) which deletes all unused variables automatically.

The assigned variable type can be changed through the context menu, by using the item 'Change Variable Type'.

Variables can be sorted by pressing on the column headers in the Table of variables, according to each of the listed properties.

Double String Boolean Structured						
ID	Description	Symbol	Value	Unit	Precision	
	Height	h	0.12	m	2	
	Width	b	0.058	m	2	

Inserting a new variable in the table

Variables can be defined manually:

- Select the proper table tab, corresponding to the needed type (Double/ String/ Boolean);
- Use the right click menu;
- Select "Insert new variable";
- Set the name for the new variable;

Structured variables cannot be defined in this way.

Inserting the new variable by the source code

The definition of variable type must be written in the code before the variable name, when the variable is used for the first time.

Examples of variable types:

- `double X = A+B;`
- `string S = "Hello";`
- `bool B = (A > B);`
- `struct P = Point(0,0);`
- `object Ext = LoadExternCLC("Extern.cls");`
- `double[] Arr= new double[];`

• `struct[] Poi = new struct[];`

An example of array definition: `double[] Pole = new double[];`
`= new double [];` When a new array is initialized, an empty vector/matrix has to be created, where values will be filled later. The command ensures that the array is not filled by other items from the previous compilation of the form. Thus, rerunning the form cannot deteriorate its quality. The object (array) definition must stay in the code, it is not saved in the CLS file as numeric or string variables.

`= new double [];` This part of the array definition formula cannot be used for variables introduced through [Table input](#). Table inputted variables are perpetually linked to the current values in the corresponding table. Using the `new double [];` command creates a conflict as the array values are set to zero.

Variables are created in the table of variables after the first time the code is compiled. When the variable has already been defined and is visible in the table, no further definition is needed. Nevertheless, it is recommended leaving the definition in the code.

`TEXT("Member length L = " & double L & " m");` //Variable L is defined and can be seen in the double tab of the Table of variables. Units should be defined additionally in the Table of variables.

Variable names

There are some rules and restrictions on the names that can be used for variables:

- The first letter of a variable name must be an upper case or lower case letter (no number are allowed);
- Symbols from both the Latin and Greek alphabet are allowed, as are numbers and the underscore symbol;
- Syntactic, mathematical, boolean and other operators are not allowed (dot, comma, semicolon, brackets, +, -, *, /, =, >, <, !, &, |, ", ' , ...);
- Operators are allowed only in the lower index (especially comma, dot and brackets).

Using other operators in the lower index is not recommended.

Renaming variables

For manual renaming of variables in the table and in the code:

- Select the variable in the table;
- Use the right click menu;
- Select "Rename selected variable";
- Define the new name.

The application will rename the variable in the table and in the source code.

NOTE: The variable name will not be changed in the TEXT strings.

Columns in the table of variables

IDs

IDs may be used to define a link to Scia Engineer or to a library (built-in or user-defined). When a variable is linked by means of an ID to an item in a library or to a value in Scia Engineer, any change in the Scia Engineer model, and any change in the currently selected library item, will result in automatic update of the value assigned to the variable. For example, if a variable refers to, through its ID, the cross-section height of the currently selected cross-section in the steel profile library in Scia Design Forms, changing the selected section will result in an immediate update and display of the referred section height in the table of variables.

Description

Variable descriptions can be added in order to assist the end-user in understanding how the calculation is built, and what is the meaning of each value being requested in the USER application dialogue. If a description is added in the table of variables, it will automatically appear together with the variable name when the variable is added to the dialogue.

Symbol

In the code and dialogue, variables are referred to by the symbol(s) assigned to them, or in other words, by their name. In predefined Design Forms provided by Nemetschek Scia, variable names are often the same as used in the corresponding structural design code - Eurocode, IBC, etc.

Value

Each variable has a current value, which is displayed in the table of variables in the default units, or in the units selected by the form developer.

Units

Physical units are defined in the 'Unit' column in the Table of variables. SI units are the basic units in Scia Design Forms; internally, all variables are recalculated back to SI units from the user-defined units shown in the 'Unit' column.

1 kN = 1000 N
1 mm = 0,001 m
1 GPa = 10^9 Pa

More about units in the chapter Units in [Table of variables](#).

Precision

The value specified in the column 'Precision' defines the variable precision.

Deleting a variable

A variable can be deleted when the corresponding row is selected in the Table of variables and the DEL key on the keyboard is used. In addition, the context menu could also be used instead.

Variable types

Variable types in SDF:

- **double** a number that ranges in value from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$; the precision is 15-16 decimal places;
- **string** text e.g.: "Sample text";
- **bool** a Boolean (logical) variable (takes up values TRUE / FALSE);
- **object** a structured variable (e.g. Point [X, Y], force [N, Vy, Vz, Mx, My, Mz], etc.) or object (external CLC, graphics, graphs).

A structured variable can be defined using a special constructor:

```
object S1 = new Structure();
```

- a structured variable named 'S1' is defined.

Double String Boolean Structured				
ID	Description	Symbol	Type	
		S1	Struct	

- **double[]** an array of numerical variables;
- **string[]** an array of text variables;
- **bool[]** an array of Boolean variables;
- **object[]** an array of objects, structured variables or arrays .

If you use the older syntax and you type `struct[]` or `array[]`, the command is automatically converted to **object []**.

```
double A;  
string B;  
bool C;  
object D;  
object E;  
double[] F;  
string[] G;  
bool[] H;  
object[] I;
```

The SDF BUILDER can automatically recognize the variable type from its use in the code. If this is not possible, the type must be defined manually - a message will be displayed in the layout window and the user can add the variable in the table of variables or using code.

The type of any variable can be manually changed - to do so, simply write the required type in front of the variable in the code. The variable is then re-declared in the table of variables and the original one must be deleted manually. The function [Purge calculation](#) can also be called for the purpose.

[Load an example: variable_types.cls](#)

How to find out the variable type from the code? Use the command:

`TEXT(<variable>.GetType().ToString());` - the variable type is printed in the layout as text.

```
System.String
System.Double
```

Type Double

Variables of the double type are numerical; these often serve as input data for calculations, as criteria for conditions, or as containers for intermediate or final results.

Syntax:

```
double <variable>;
```

- this script creates a variable with the name <variable> in the table of variables on tab Double; a zero value is assigned to <variable>.

Example

```
double A;
```

The SDF USER application does not distinguish between dot and coma as decimal separators. The value 123.456 is the same as the value 123,456.

The column "Value" cannot contain any text. The only exception is the exponent, for example: "1e6" is the same as "1000000"

All variables are automatically inserted as type Double. The user can change the type manually, or convert the type through the code by using the appropriate command (see previous chapter).

ID	Popis	Symbol	Hodnota	Jednotky	Přesnost
	Design wind load on the wall	q_{wzd}	0.5	kN/m ²	2
	Wall thickness	t	300	mm	2
	Wall height	h	3	m	2
	Length of wall loaded by axial force	b	1	m	2
		t_{min}	96	mm	2
	Coefficient of construction block type	k	0.1		2

Variables in cells coloured in green should be manually defined (in the dialogue or Table of variables); variables in white cells are defined or calculated in the code.

Type String

Variables of the string type take up text values, and can be used as headlines, additional explanations or descriptions. Strings may be built from text and number characters, punctuation marks and other operators.

Syntax:

```
string <variable>;
```

- this script creates a variable with the name <variable> in the table of variables on the tab 'String.'

Example

```
string B;
```

String variables can be edited in the SDF USER application, if these are included in the Dialogue, even if these have been assigned a value (text) in the code. String variables allow for the user to add additional user-defined texts in predefined locations in the layout.

Double String Boolean Structured			
ID	Popis	Symbol	Hodnota
	Userdefined headline	caption	uživatelský nadpis
	National code	NA	ČSN EN 1996-1-1, §6.2

Any text can be assigned as a value for a string.

Although Scia Design Forms allows for the translation of forms to any other language (by using the last vertical tab 'Translations' in the BUILDER application, see Chapter 'Translations'), string variables are not included in the automatic translation engine. There is a way around this though. When a string variable is inserted in the dialogue, the end-user is able to change the text in the string in the same way as double variables are assigned numerical values. The end-user can then manually translate the text, or even better, the translations can be added to the .DEFAULT file in the same way national annex specific parameters are taken into account in the USER application.

IF the string variable is used in a TEXT command, then it must be excluded from translation by the checkbox on the Translations tab.

<input type="checkbox"/>	TEXT_000590	NA
<input type="checkbox"/>	TEXT_000066	UserDefined

Double String Boolean Structured			
ID	Popis	Symbol	Hodnota
	Userdefined headline	caption	uživatelský nadpis
	National code	NA	ČSN EN 1996-1-1, §6.2

Strings are always defined manually, either in the Table of variables or in the code.

Type Boolean

Boolean variables can only assume the values TRUE or FALSE.

Syntax:

```
bool <variable>;
```

- this syntax creates a variable with the name <variable> on the Boolean tab in the table of variables, and sets it to TRUE.

Example

```
bool C;
```

The boolean variables are defined by checkboxes in the Table of variables or in the dialogue in both the BUILDER and USER applications.

- Checked = TRUE
- Unchecked = FALSE

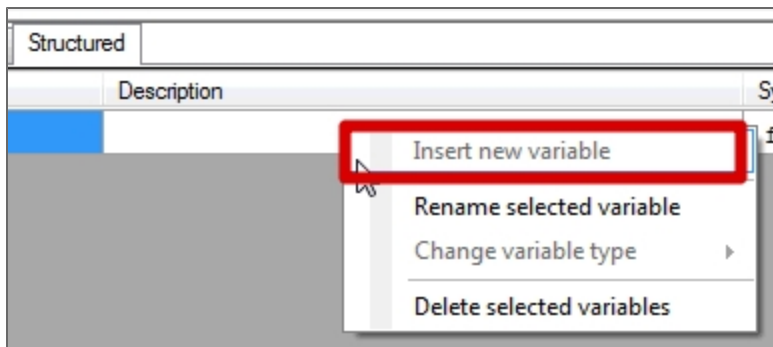
ID	Popis	Symbol	Hodnota
	Vertical gaps are filled by mortar	B1	<input checked="" type="checkbox"/>
	Print headline	Headline	<input checked="" type="checkbox"/>
	Print userdefined headline	PrintCaption	<input type="checkbox"/>

It is important to use the correct syntax for conditions with boolean variables (simple or chained) ([see more about conditions here](#)).

Type Structured

A structured variable is a container that stores a set of variables that do not have to be of the same type. A container can contain a numerical variable next to a boolean one, or it can, for example, store several numerical variables, each of different unit. A structured variable is declared using the same syntax as an object.

See more about [structured](#).



A special constructor `new Structure()`; is used for that purpose.

Syntax:

```
object <variable>;
```

- this notation declares a variable of the Object type on the Structured tab of the table of variables.

Example

```
object D;
```

Nested variables in structured variable

Nested variables can be used in the same way as any other variable.

Syntax:

The syntax for nested variables is:

```
<structured_variable_name> . <nested_variable_name>;
```

Example

`IntForce.N`; - `IntForce` is the name of a structured variable and `N` is the name of the nested variable.

New Structure()

Syntax:

```
object <variable> = new Structure();
```

- a new empty structured variable is defined

```
object <variable> = new Structure("<name>");
```

- a new empty structured variable with a key name is defined

Example

```
object <variable> = new Structure("X", 0.050, "Y", 0.050, "D", 0.020);
```

- a new structured variable is defined, it contains sub-variables X, Y, D, and these sub-variables are also assigned values X=0.005; Y=0.050; D=0.020

```
object S1 = new Structure();
```

- a structured variable named S1 is defined (in the table the name is listed in the column Symbol).

Double		String		Boolean		Structured	
ID	Description	Symbol	Type				
		S1	Struct				

[Load an example: Structure.cls](#)

<structured_variable>.Add()

The Add function adds a new sub-variable at the end of a structured variable. Exactly what type is the new sub-variable is specified by the function parameters. Generally, a variable of any type can be added.

Syntax:

```
<variable>.Add("<property_name>", <value>);
```

- a new sub-variable with assigned value is added to the variable <variable>.

Example

```
object S1 = new Structure("MyStructure");
```

- object S1 is a structured variable named MyStructure.


```
S1.Add("Item_number", 123);
TEXT("S1.MyProperty = " & S1.MyProperty);
```

- a new sub-variable named Item_number is added to variable S1 and this new sub-variable is assigned a value of 123;
- the second line displays this value in the layout.

```
S2.Add("BooleanProperty", true);
```

- a sub-variable BooleanProperty is added to variable S2 and is set to TRUE.

```
S3.Add("List", new object[]);
S3.List.Add(new Point(2, 0));
TEXT("S3.X = " & S3.X);
TEXT("S3.Y = " & S3.Y);
TEXT("S3.D = " & S3.D);
TEXT("S3.List[0] = " & S3.List[0]);
TEXT("S3.List[0] = " & S3.List[0].X);
```

- a new sub-variable named List is added to variable S3 and is assigned a new array;
- (object) sub-variable List is assigned a C# object of type Point (array with two values 2, 0);
- the three following TEXT commands display in the layout the values of X, Y and D;
- the fourth TEXT command displays in the layout the value of the List array with index 0 - i.e. Point with two sub-variables: X, Y;
- the fifth TEXT command writes number 2 that is stored in the Point object at index 0.

C# function POINT

Point()

Syntax:

```
Point(<X>, <Y>);
```

- a C# function, defines a structured variable that contains two coordinates X and Y. X and Y are two nested variables of the function Point (X, Y) (this function is primarily intended to define a point by means of two coordinates). This function contains Integer variables.

Example

```
How to get the X coordinate from the previous example:
struct[] POINT= Point(0,0);
POINT.X = ...
```

To navigate within the structured variable, the dot convention must be used (it is not possible to use indexes like with arrays).

More information can be found [here](#).

PointF()

Function Point with predefined variables of type float [X, Y]

More information can be found [here](#).

PointD()

Function Point with predefined variables of type double [X, Y] **This is the recommended type of the Point function.**

Predefined structured variables

ReinfBar

A structured variable with predefined properties [X, Y, D]

Forces1D

A structured variable with predefined properties [VN, Vy, Vz, Mx, My, Mz]

Forces2D

A structured variable with predefined properties [nx, ny, nxy, mx, my, vx, vy]

MaterialPoint

A structured variable with predefined properties [eps, sig]

MaterialDiagram

An array of points that form a material stress-strain diagram.

Type Object

This variable type is used for [arrays](#) (vectors, 2D/3D matrices, etc.), structured variables, objects, graphs and for links to external CLC files.

Syntax:

```
object <variable>;
```

- this script creates a variable in the table of variables with the name <variable>; the variable is displayed on the Structured tab with type Object.

Example

```
object D;
```

Loading an external CLC to the variable type object:

```
object Ext = LoadExternCLC("Extern.cls");
```

The array (or some other variable) length can be displayed by the using the string .Count or .Length. after the variable name, e.g. A = Ext.Count;

Array

An Array is a group of items of the same type. Variables in this group are called items of an array (array of numbers, array of strings ...), each item has its number (index) and the user can get the item using this number as reference.

Items in an array can be accessed via indexes. The index of the first item in an array is always zero (0).

Syntax:

```
double[] Array
```

- declares a variable of type Array, it is not filled with any data, it is not even an empty array.

```
double[] Array = new double [];
```

- the declared array is initialised, i.e. zeroes are assigned to its items.

This command at the beginning of a script ensures that the array is empty, in other words, this command makes sure that the array does not keep any values from a previous calculation cycle. This syntax guarantees a smooth run of the calculation.

We recommend that the second syntax be used for any new variables, **unless** the variable in question is a variable defined via the **Table input function**.

Example

```
X = MyArray[2]; - the value of MyArray with index 2 is stored to variable "X"
```

```
for(i, 0, 5) { TEXT(MyArray[i]); } - displays the first 6 items from MyArray (index 0 to 5)
```

Declaration of an array in the code

An Array can be declared directly in the source code. The declaration consists of the array type and empty square brackets.

- `double [] Numbers; // declares an array of real (double) numbers named "Numbers."`
- `string [] Texts; // declares an array of strings named "Texts."`
- `object [] Objects; // declares an array of objects, structured variables or arrays named "Objects."`
- `struct [] Points; // declares an array of structured variables named "Points."`
- `array [] Arrays // declares an array of arrays named "Arrays."`

If you use the older syntax (v.3) and you type `struct[]` or `array[]`, the command is automatically converted to **object []**.

An array has a whole set of additional functions. Count, Add, Remove ...

Example

An object variable is named Table

- 1) TEXT (Table[3]); - displays the value of the cell with index equal to 3 (indexes are numbered starting from 0);
- 2) TEXT(Table.Count); - displays in the layout the number of rows that are filled in the table input;
- 3) Table.Remove(7); - removes the value from the cell with index equal to 7 and the following values are moved forward by one cell.

object A = new Structure(); - a new structural variable A

A.Add("Array", new object[]); - an array named Array is added to the structured variable from the line above

A.List.Add(new Point(2, 0)); - a C# Point object is added to the array (point is in fact an array of two values: X and Y

TEXT(A.List[0]); - Point is displayed in the layout as it is the item with index = 0

TEXT(A.List[0].X); - value 2 is displayed in the layout as it is the value of sub-variable X

The Count command (that returns size) cannot be used with some variables; in such cases, it is replaced by the command Length.

If values are assigned to an array variable using indexing, it is not recommended to skip indexes; this is because, internally, all the skipped indexes must be declared. If a cell is not declared by indexing, it is still created automatically by the application.

[Load an example: TableInputExample.cls](#)

Units in the table of variables

The Unit functionality in Scia Design Forms allows the user to assign physical measurement units (meters, Newtons, Pascals, etc.) to variables of the type double.

Units are used in the:

- Dialogue - user input is interpreted in the units assigned in the table of variables;
- Layout - the units are mentioned in the layout window and report:
 - inside an equation, when the "Print units" checkbox is checked;
 - on the right part of the equation, where result with unit is displayed. For this function to work, all variables that participate in the equation should be assigned units in the Table of variables, including the result.

Scia Design Forms recognises all [common physical units](#) - in addition to the SI, Gaussian and other metric unit systems, imperial units are also available. User-defined unit systems can be created by combinations of the internally defined units.

Example: kg/m, t/m², kg*m*s⁻², mm/m, 10⁻⁴*m²

Physical units are used mainly in the Dialogue and Layouts. The values are saved in basic SI units internally. It is allowed to combine different units in one formula or equation, as the conversion is performed automatically.

Example:
there are X (mm), a (mm), c (cm), d (m) in table of variables
 $X = a + b + c = 1 \text{ mm} + 2 \text{ cm} + 3 \text{ m} = 3021 \text{ mm}$

Scia Design Forms DOES NOT check if the equation contains compatible units. It is possible to use wrong unit combinations in the equation and SDF will calculate it internally by converting to SI units and summing up the obtained basic SI values.

Example:
 $X = 1 \text{ kPa} + 2 \text{ dm} + 3 \text{ s} = 1000 + 0,2 + 3 = 1003,2$

The list of internally defined units

Symbol	Coefficient	Exponent SI units						
		m	kg	s	A	K	cd	mol
Length:								
mm	0,001	1	0	0	0	0	0	0
cm	0,01	1	0	0	0	0	0	0
dm	0,1	1	0	0	0	0	0	0
m	1	1	0	0	0	0	0	0
km	1000	1	0	0	0	0	0	0
in	0,0254	1	0	0	0	0	0	0
ft	0,3048	1	0	0	0	0	0	0
yd	0,9144	1	0	0	0	0	0	0
mi	1609,344	1	0	0	0	0	0	0
nmi	1852	1	0	0	0	0	0	0
AU	1,50E+11	1	0	0	0	0	0	0
pc	3,09E+16	1	0	0	0	0	0	0
ly	9,46E+15	1	0	0	0	0	0	0
Weight:								
kg	1	0	1	0	0	0	0	0
t	1000	0	1	0	0	0	0	0
lb	1000	0	1	0	0	0	0	0
Time:								
ns	1,00E-06	0	0	1	0	0	0	0
ms	0,001	0	0	1	0	0	0	0
s	1	0	0	1	0	0	0	0
min	60	0	0	1	0	0	0	0
h	3600	0	0	1	0	0	0	0
den	86400	0	0	1	0	0	0	0
Electric current:								
A	1	0	0	0	1	0	0	0

Temperature:								
K	1	0	0	0	0	1	0	0
°C	1	0	0	0	0	1	0	0
°F	5,0/9,0	0	0	0	0	1	0	0
Luminous intensity:								
cd	1	0	0	0	0	0	1	0
Number of moles:								
mol	1	0	0	0	0	0	0	1
Area:								
ar	1,00E+02	2	0	0	0	0	0	0
ha	1,00E+04	2	0	0	0	0	0	0
Moment								
Nm	1	2	1	-2	0	0	0	0
kNm	1,00E+03	2	1	-2	0	0	0	0
MNm	1,00E+06	2	1	-2	0	0	0	0
Compression:								
Pa	1	-1	1	-2	0	0	0	0
kPa	1,00E+03	-1	1	-2	0	0	0	0
MPa	1,00E+06	-1	1	-2	0	0	0	0
GPa	1,00E+09	-1	1	-2	0	0	0	0
bar	1,00E+05	-1	1	-2	0	0	0	0
atm	101325	-1	1	-2	0	0	0	0
Torr	133,3224	-1	1	-2	0	0	0	0
psi	6894,757	-1	1	-2	0	0	0	0
ksi	6894757	-1	1	-2	0	0	0	0
Force:								
N	1	1	1	-2	0	0	0	0
kN	1,00E+03	1	1	-2	0	0	0	0

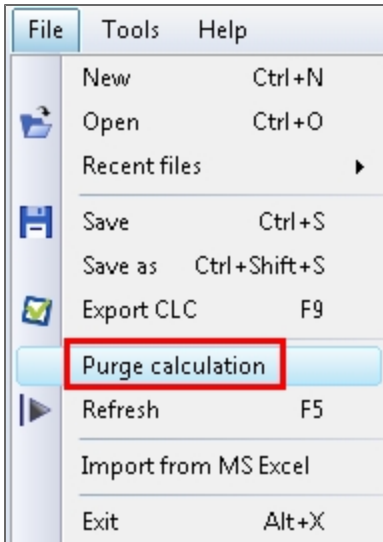
MN	1,00E+06	1	1	-2	0	0	0	0
GN	1,00E+09	1	1	-2	0	0	0	0
kip	4448,2216	1	1	-2	0	0	0	0
Cubic capacity:								
ml	1,00E-06	3	0	0	0	0	0	0
cl	1,00E-05	3	0	0	0	0	0	0
dl	1,00E-04	3	0	0	0	0	0	0
l	1,00E-03	3	0	0	0	0	0	0
hl	0,1	3	0	0	0	0	0	0
Angle:								
rad	1	0	0	0	0	0	0	0
ster radian	1	0	0	0	0	0	0	0
degree	1	0	0	0	0	0	0	0
Percentage:								
%	0,01	0	0	0	0	0	0	0
‰	0,001	0	0	0	0	0	0	0
Frequency:								
Hz	1	0	0	-1	0	0	0	0
kHz	1,00E+03	0	0	-1	0	0	0	0
MHz	1,00E+03	0	0	-1	0	0	0	0
GHz	1,00E+09	0	0	-1	0	0	0	0
Work:								
eV	1,60E-19	2	1	-2	0	0	0	0
mJ	1,00E-03	2	1	-2	0	0	0	0
J	1	2	1	-2	0	0	0	0
kJ	1,00E+03	2	1	-2	0	0	0	0
MJ	1,00E+06	2	1	-2	0	0	0	0
GJ	1,00E+09	2	1	-2	0	0	0	0
cal	4,1868	2	1	-2	0	0	0	0
kcal	4186,8	2	1	-2	0	0	0	0

Power:								
mW	1,00E-03	2	1	-3	0	0	0	0
W	1	2	1	-3	0	0	0	0
kW	1,00E+03	2	1	-3	0	0	0	0
MW	1,00E+06	2	1	-3	0	0	0	0
GM	1,00E+09	2	1	-3	0	0	0	0
Electric voltage:								
mV	1,00E-03	2	1	-3	-1	0	0	0
V	1	2	1	-3	-1	0	0	0
kV	1,00E+03	2	1	-3	-1	0	0	0
Electric capacitance:								
F	1	-2	-1	4	2	0	0	0
Electric charge:								
C	1	0	0	1	1	0	0	0
Electrical resistance:								
Ω	1	2	1	-3	-2	0	0	0
Illuminance:								
lx	1	-2	0	0	0	0	1	0
Magnetic flux density:								
T	1	0	1	-2	-1	0	0	0

The user can define units as basic units to the power x : $m^{x=2}$. SDF can then convert correctly from e.g. mm^2 to m^2 .

Function PURGE

The function allows to delete unused parts of the calculation - variables, graphical settings. The function is applied after the calculation has been compiled.



The function deletes:

- Unused variables;
- Unused definitions of visual components.
- Unused translations - e.g. commented parts of code

Attention: The PURGE function deletes all unused components, including the commented ones!

Call the function PURGE:

- Select "File" in the main menu;
- Select "Purge";
- Confirm with 'OK'.

If there are some unused translations, they are during purge also.

Import from MS Excel

Calculation in MS Excel can be converted to SDF code through a tool called 'Import from MS Excel'.

The MS Excel import tool converts variables, formulas and dependencies between cells, and descriptions (if defined) from Excel.

Start the Import wizard:

- Go to Main Menu > Project;
- Select the function "Import from MS Excel".

The wizard is divided into the 3 parts.

- The left panel shows the recognized the excel table - only non-zero rows and columns are included;
- The list of used variables in the middle panel;
- Preview of imported formulas and texts on the right panel.

Import procedure:

- Load the excel calculation by using the button "Load excel file";
- Select the cells which should be transferred to SDF;
- Cells with formulas (these will automatically be converted to equations) or cells with plain text (these will automatically be converted to text by the text command) are selectable;
- Select more cells by using the CTRL or SHIFT key;
- Use the "Select cells" button when the selection is finished;
- The table with variables, displayed equations and texts are refreshed;
- the wizard may be closed;
- The selected data are converted and appended in the Code editor.

Functions which can be converted

Mathematical functions which can be taken from MS Excel to Scia Design Forms are described in the Commands reference guide. The name of the function must match the name used in SDF.

The exception is the command IF; SDF recognizes KDYŽ + and other languages in excel.

	A	B	C	D
1	Weight of the paint on the beam:			
2	Beam length	L	5 m	
3	Area per m	a	0.75 m ² /bm	
4	Area of beam	A	3,75 m ²	
5	Paint per m ²	q	0,5 l/m ²	
6	Paint on beam	Q	=C5*C4	

Calculation panel:
 1 Q = A * q;
 Q = A * q = 5.00 * 2.00 = 10.0 m

Renaming the used variables:

- Select the variable which should be renamed in the table;
- Select more cells by using the CTRL or SHIFT key;
- Link the relevant cell address with the correct name in the column "Variable name" (see next chapter)

or

- Right click in the table of variables;
- Select item "Variable name" from the context menu;
- Select the position with the variable name

Defining the variable descriptions:

- Select the variable which should have a description in the table;
- Select more cells using CTRL or SHIFT key;
- Write the relevant address of the cell with the correct description to column "Variable description" (see next chapter)

or

- Use the right mouse click in the table with variables.
- Select item "Variable description" from the context menu
- Select the position with the variable description

Setting the relevant address of the cell

The address is defined as X;Y, it is relevant to the current cell. the X coordinate grows from the left to the right; Y from the top to the bottom

Example:

The cell on the left from the current one is: -1;0

The cell under the current one is: 0;1

The cell 3 columns to the right from the current one is: 3;0

The cell 1 columns to the right and 2 rows below the current one is: 1;-2

Removing cells from the selection

- Select the cells which should be removed in the preview;
- Press the "Unselect cells" button.

Delete variable in the table

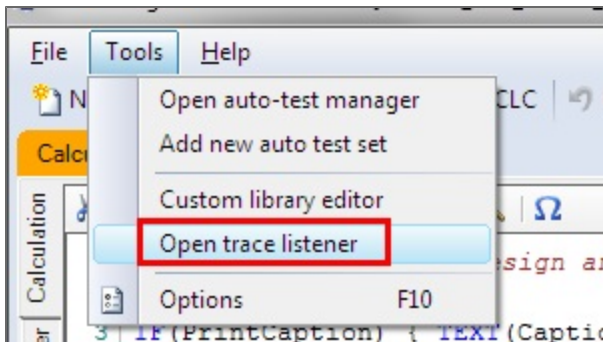
- Select the variable which should be deleted;
- Select more cells by using CTRL or SHIFT key;
- Press the button "Delete variables".

The tool for debugging the code - Trace listener

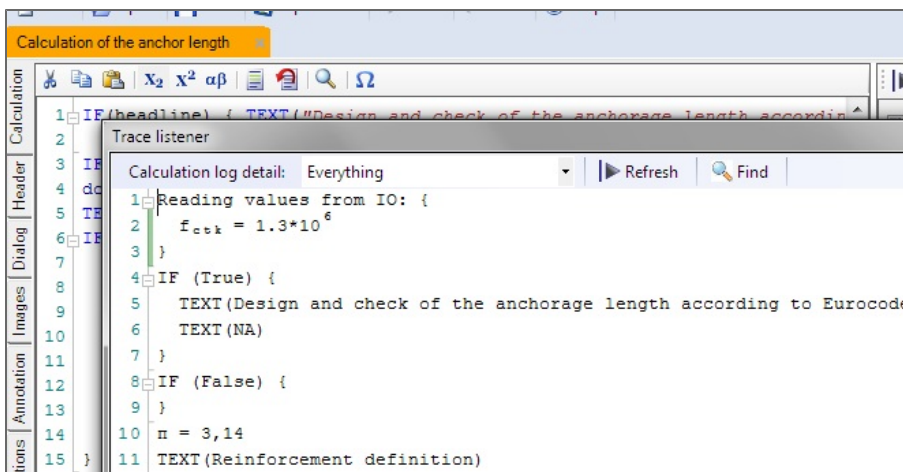
The trace listener is a debugging tool in the SDF BUILDER.

It shows the list of commands which are called from the script and it displays the basic info for debugging.

Go to Main Menu > Tools > Open trace listener.



The trace listener is displayed in a separate window.



Creating the Layout

A preview of the calculation report is displayed in the layout window of the Scia Design Forms BUILDER; the layout (report) is what is shown in the USER application as an output of the form. The report can be exported from the USER to a number of image formats, *.rtf and MS Word formats.

Calculations in the layout should be presented professionally and therefore, these are usually formatted with texts styles, tables, graphs and images.

Usually, several layouts are created for the same calculation, in order to foresee the different needs of users - reports sometimes need to be as detailed as possible, and other times only a summary is sufficient.

Example:

The following screenshots are taken from the same Design Form - the calculation method and results are the same, yet the output is different.

The full layout - contains all descriptions, images and remarks:

Refresh Full Edit layouts English (United States)

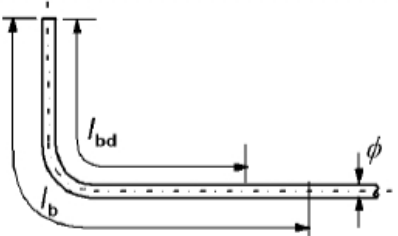
Draw borders Arial X = 0
 Visible 10 B U / Y = 0 Print units
 Predefined styles:

Design and check of the anchorage length according to Eurocode 2

CSN EN 1992-1-1 §8.4.3, §8.4.4

Reinforcement definition

Number of bars	$n=3$
Reinforcement diameter	$\phi=8 \cdot 10^{-3}$ mm
Reinforcement area	$A_{st}=n \cdot \pi \cdot \frac{\phi^2}{4}=3 \cdot 3.14 \cdot \frac{8 \cdot 10^{-3}{}^2}{4}=151 \cdot 10^{-6}$ m ²
Stress for the calculation of the anchorage length	$\sigma_{sd}=\frac{M_{Ed}}{z_b \cdot A_{st}}=\frac{50000}{0.025 \cdot 151 \cdot 10^{-6}}=13270$ MPa
Coefficient related to the quality of bond condition	$\eta_1=1$
Coefficient of the bar diameter value $\leq \phi \cdot 32$ mm	$\eta_2=1$
Coefficients:	$\alpha_1=1, \alpha_2=1, \alpha_3=1, \alpha_4=1, \alpha_5=1$



Design value of concrete tension strength	$f_{ctd}=\frac{\alpha_{ct} \cdot f_{ctk}}{\gamma_c}=\frac{1 \cdot 2.2 \cdot 10^6}{1.5}=1.47$ MPa
Design value of the ultimate bond stress for ribbed bars	$f_{bd}=2.25 \cdot \eta_1 \cdot \eta_2 \cdot f_{ctd}=2.25 \cdot 1 \cdot 1 \cdot 1.47 \cdot 10^6=3.3$ MPa

Design of anchorage length:

Basic anchorage length	$l_{brqd}=\frac{\phi}{4} \cdot \frac{\sigma_{sd}}{f_{bd}}=\frac{8 \cdot 10^{-3}}{4} \cdot \frac{13.3 \cdot 10^9}{3.3 \cdot 10^6}=8.04$ m
<i>assumption: f_{bd} is constant on the whole bar</i>	
$\alpha_1 \alpha_2 \alpha_3 \geq 0.7 \Rightarrow$ is SUFFICIENT	
The designed anchorage length l_{bd} :	$l_{bd}=\alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \alpha_5 \cdot l_{brqd}=1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 8.04=8.04$ m

Check of the anchorage length $l_{bd} \geq l_{b,min}$:

Tensile anchorage	$l_{bmin1}=\text{Max} \left\{ \begin{matrix} 0.3 \cdot l_{brqd} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.3 \cdot 8.04 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{matrix} \right\} = 2.41$ m
<i>Tensile anch. length must be bigger then 2.41 m \Rightarrow is SUFFICIENT $\Rightarrow 2.41$ m < 8.04 m</i>	
Compression anchorage	$l_{bmin2}=\text{Max} \left\{ \begin{matrix} 0.6 \cdot l_{brqd} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.6 \cdot 8.04 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{matrix} \right\} = 4.83$ m
<i>Comp. anch. length must be bigger then 4.83 m \Rightarrow is SUFFICIENT $\Rightarrow 4.83$ m < 8.04 m</i>	

The standard layout - contains a basic description, no remarks, not all images:

Refresh Standard Edit layouts English (United States)

Draw borders Arial X = 0 Y = 0 Print units

Visible 10 B U /

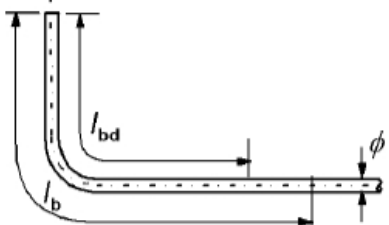
Predefined styles:

Design and check of the anchorage length according to Eurocode 2

CSN EN 1992-1-1 §8.4.3, §8.4.4

Reinforcement definition

$n=3$
 $\phi=8 \cdot 10^{-3} \text{ mm}$
 $A_{st}=151 \cdot 10^{-6} \text{ m}^2$
 $\sigma_{sd} = \frac{M_{Ed}}{Z_b \cdot A_{st}} = \frac{50000}{0.025 \cdot 151 \cdot 10^{-6}} = 13270 \text{ MPa}$
 $\eta_1=1$
 $\eta_2=1$
 $\alpha_1=1, \alpha_2=1, \alpha_3=1, \alpha_4=1, \alpha_5=1$



$f_{ctd} = \frac{\alpha_{ct} \cdot f_{ctk}}{\gamma_c} = \frac{1 \cdot 2.2 \cdot 10^6}{1.5} = 1.47 \text{ MPa}$
 $f_{bd} = 2.25 \cdot \eta_1 \cdot \eta_2 \cdot f_{ctd} = 2.25 \cdot 1 \cdot 1 \cdot 1.47 \cdot 10^6 = 3.3 \text{ MPa}$

Design of anchorage length:

$l_{brqd} = \frac{\phi}{4} \cdot \frac{\sigma_{sd}}{f_{bd}} = \frac{8 \cdot 10^{-3}}{4} \cdot \frac{13.3 \cdot 10^9}{3.3 \cdot 10^6} = 8.04 \text{ m}$
assumption: f_{bd} is constant on the whole bar
 ~~$\alpha_1 \alpha_2 \alpha_3 \geq 0.7 \Rightarrow$~~ **is SUFFICIENT**
 $l_{bd} = \alpha_1 \cdot \alpha_2 \cdot \alpha_3 \cdot \alpha_4 \cdot \alpha_5 \cdot l_{brqd} = 1 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdot 8.04 = 8.04 \text{ m}$

Check of the anchorage length $l_{bd} \geq l_{b,min}$:

$l_{bmin1} = \text{Max} \left\{ \begin{array}{l} 0.3 \cdot l_{brqd} \\ 10 \cdot \phi \\ 0.1 \end{array} \right\} = \text{Max} \left\{ \begin{array}{l} 0.3 \cdot 8.04 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{array} \right\} = 2.41 \text{ m}$
Tensile anch. length must be bigger then 2.41 m \Rightarrow is SUFFICIENT $\Rightarrow 2.41 \text{ m} < 8.04 \text{ m}$

$l_{bmin2} = \text{Max} \left\{ \begin{array}{l} 0.6 \cdot l_{brqd} \\ 10 \cdot \phi \\ 0.1 \end{array} \right\} = \text{Max} \left\{ \begin{array}{l} 0.6 \cdot 8.04 \\ 10 \cdot 8 \cdot 10^{-3} \\ 0.1 \end{array} \right\} = 4.83 \text{ m}$
Comp. anch. length must be bigger then 4.83 m \Rightarrow is SUFFICIENT $\Rightarrow 4.83 \text{ m} < 8.04 \text{ m}$

The brief layout - contains a brief summary of what is calculated and what are the results, no images, equations or remarks:

Refresh Brief Edit layouts English (United States)

Draw borders Arial X = 0
 Visible 10 B U / Y = 0 Print units
 Predefined styles:

Design and check of the anchorage length according to Eurocode 2
CSN EN 1992-1-1 §8.4.3, §8.4.4

Reinforcement definition

$n=3$ $\phi=8 \cdot 10^{-3}$ mm
 $A_{st}=151 \cdot 10^{-6}$ m²
 $\sigma_{sd}=13270$ MPa
 $\eta_1=1$ $\eta_2=1$
 $\alpha_1=1$, $\alpha_2=1$, $\alpha_3=1$, $\alpha_4=1$, $\alpha_5=1$
 $f_{ctd}=1.47$ MPa
 $f_{bd}=3.3$ MPa

Design of anchorage length:

$l_{brqd}=8.04$ m
 $\alpha_1 \alpha_2 \alpha_3 \geq 0.7 \Rightarrow$ **is SUFFICIENT**
 $l_{bd}=8.04$ m

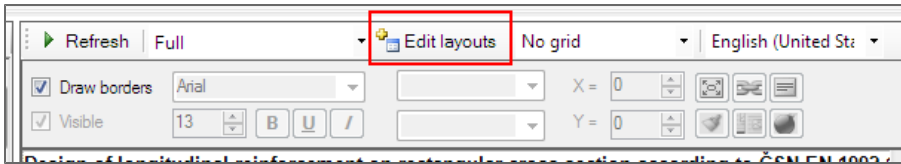
Check of the anchorage length $l_{bd} \geq l_{b,min}$:

$l_{bmin1}=2.41$ m
Tensile anch. length must be bigger then 2.41 m => is SUFFICIENT => 2.41 m < 8.04 m
 $l_{bmin2}=4.83$ m
Comp. anch. length must be bigger then 4.83 m => is SUFFICIENT => 4.83 m < 8.04 m

Layout menu

Editing layouts

The red box indicates a button used for managing layouts in the Scia Design Forms BUILDER. The 'Edit layouts' function opens a list of active/inactive layouts, where names can be assigned to layouts, and layouts can be rearranged or deleted.



Renaming a layout

- Open the layout list by clicking on the button "Edit layouts";
- Select the target layout to be renamed;
- Click twice on the layout name;
- Set a new name and confirm with 'OK.'

Creating new layouts

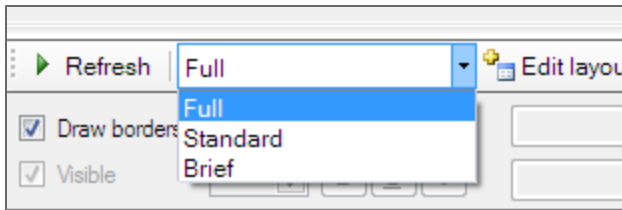
A new layout is defined by checking the checkbox in front of the layout name in the list.

- Open the layout list by clicking on the button "Edit layouts";
- Select the layout which should be activated;
- Check the checkbox;
- Set a name of the layout and confirm with 'OK.'

Switching layouts

The user can switch between layouts by using the combobox on the toolbar.

Changes which are made to the report style (displayed in the layout window) are saved to the currently active layout. If the user wants to apply those changes to other layouts, it is useful to edit one layout and copy its visual setting to the other ones by using the 'Copy properties of selected components from one layout to another'.



Design of longitudinal reinforcement on rectangular

Refresh Standard Edit layouts English (United States)

Draw borders Arial None X = 0

Visible 17 B U J Follow dY = 20 Print units

1 2 3 4 5 6 7 8 9 0

Fatigue design of steel components according to EN 1993-1-9:2005
Heavy gauge steel members and connections

Basic information Copy layout settings

	Template layout	Target layouts
Number of	Standard	Standard
Partial fact	Detailed	Detailed
Partial fact	Brief	Brief
Stress s		
Nominal st	Layout 4	Layout 4
Stress ran	Layout 5	Layout 5
Stress ran	Layout 6	Layout 6
Stress ran		
Stress ran		
Stress ran		
Referen		
Please, inc		
Classificati		
Stress r		
Calculated		

OK

Calculation layout

Each calculation form may contain up to 6 layouts.

The order of components (texts, equations, conditions, cycles, etc.) in the layout is generated following to the order of commands in the code. Newly defined or edited components are displayed after refresh.

In the beginning (or in other words by default), all components are displayed below one another with the same formatting.

Component selection

Use left click to select a component. The selected component is marked by a red rectangle (optional) and its properties are displayed in the toolbar.

Use SHIFT to select multiple components.

Selecting components from the code

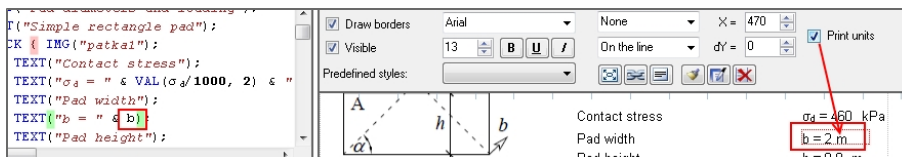
Components can be selected from the code as well. Hidden components can only be selected in this way.

Select the whole command to select the targeted component. Use CTRL+A to select the whole code. The properties of all components will be displayed. All components can be now set to 'visible'.

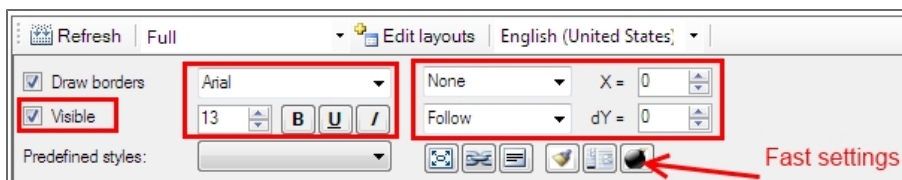
Selected visual component are highlighted by red background in the code. Use double-click on the component in the layout preview window to locate the source code in the code editor.

Variables in the TEXT command

When a variable is used inside a TEXT command, it is displayed without units by default. Units can be displayed by checking the checkbox 'Print units'.



Component properties



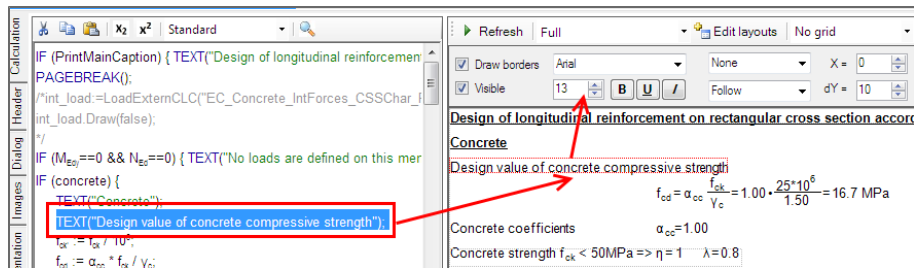
Visibility

Visibility of components is controlled by the checkbox 'Visible.'

- Select the target component.
- Check or uncheck the visibility checkbox.

Font settings

- Select the component which should be edited.
- Set the font, font size and font properties on the toolbar.



Default component

The default component is used to define the position of visual components (next to it, under it ..). The default component is usually the previously defined visible component in the code, unless the user manually specifies it.

Attention: The last calculated component is not necessarily the default component. If a visual component is set to invisible, then it cannot be the default component. Also, conditions (IF, FOR, SWITCH commands) make it more complicated to determine which is the default component, as all possibilities and calculation outcomes should be considered.

Default component selection

The default component can be defined manually:

- Select the dependent component;
- Hold the ALT key and click on the default component in the layout editor.

The default component is marked by a blue rectangle.

Cancelling the connection to default component

The link to a default component can be cancelled by holding the ALT key and clicking on an empty area.

Position settings

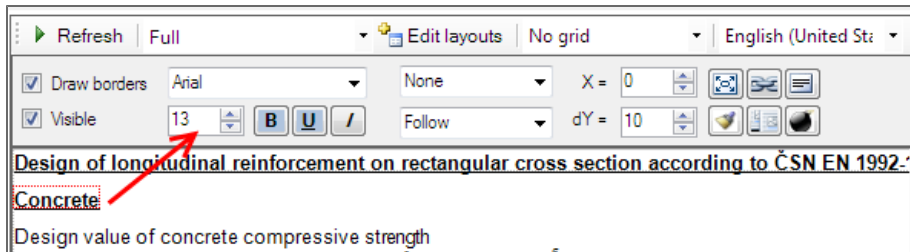
The component (text, equation, formula, image) position in the graphical output is defined relatively to its default component or by global coordinates of the insertion point. Both methods may be combined independently for the X- and Y-direction.

Horizontal align

The horizontal align setting defines the position in horizontal direction (X-direction).

The alignment can be absolute or relative to the default component:

- None - an absolute value for the X-coordinate should be defined;
- Align left - the left side will match with the left side of the default component;
- Centre - the component is aligned to the centre of the default component;
- Align right - the right side will match with the right side of the default component;
- Follow - the component is put behind the default component.



Vertical align

It defines the position in vertical direction (Y-direction).

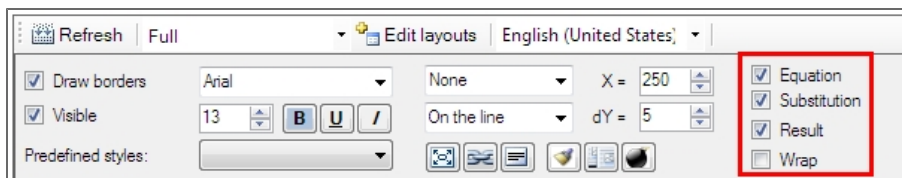
The alignment can be absolute or relative to the default component:

- None - an absolute value for Y-coordinate should be defined;
- Align top - the top side will match with the top side of the default component;
- On the line - the component is aligned on the same line with the default component;
- Align bottom - the bottom side will match with the bottom side of the default component;
- Follow - the component is put behind the default component.

Equation settings

On the right side of the properties area, settings are provided to define the display of equations.

- Equation - displays the equation;
- Substitution - displays the substitution;
- Result - displays the result;
- Wrapping - wraps the equation, carries over long equations to the next line, so that these fit easily on a page.



- Print units - allows to display units inside the equation and in the result.

Refresh Full Edit layouts English (United States)

Draw borders Arial Align left $dX = 250$ Equation
 Visible 13 **B** **U** **I** On the line $dY = 0$ Substitution Print units
 Predefined styles: Result Wrap

Check bending moment on concrete pad

Reinforcement design

Effective cross section height $d = h - c - \frac{\phi}{2} = 0.9 - 0.06 - \frac{8 \cdot 10^{-3}}{2} = 0.836 \text{ m}$

Lever arm of inner forces $z = 0.9 \cdot d = 0.9 \cdot 0.836 = 0.752 \text{ m}$

Refresh Full Edit layouts English (United States)

Draw borders Arial Align left $dX = 250$ Equation
 Visible 13 **B** **U** **I** On the line $dY = 0$ Substitution Print units
 Predefined styles: Result Wrap

Check bending moment on concrete pad

Reinforcement design

Effective cross section height $d = h - c - \frac{\phi}{2} = 0.9 \text{ m} - 60 \text{ mm} - \frac{8 \text{ mm}}{2} = 0.836 \text{ m}$

Lever arm of inner forces $z = 0.9 \cdot d = 0.9 \cdot 0.836 = 0.752 \text{ m}$

Required reinf. area $A_{s,req} = \frac{M_{Ed}}{z} = \frac{340216}{0.752} = 1.06 \cdot 10^{-3} \text{ m}^2$

Predefined styles of components in layout

Predefined styles are used for quick setting of component properties. These allow the user to organise the final calculation layout easily.

Predefined styles are saved under 10 buttons in the property panel; all 10 styles may be edited by the user. The definition file for predefined styles is saved in UserDocuments>\DesignForms_4\PreDefinedStyles.xml.

Download [PreDefinedStyles.xml](#)

To assign a style to a component:

- Select one or more components;
- Click on the button with corresponding to the desired style.

How to save a new style:

- Select the sample component which has already been formatted;
- Hold the Control Key (Ctrl) and click on target style button;
- The formatting style of the selected component is saved under the selected button.

Predefined styles:

Style	Horizontal align	Vertical align	Font				Equation			
			Font	B- o- lt	Under- line	Italic	Equation	Sub- stitution	Result	W- ra- p
Standard 1	Align left 0	Follow 5	Arial 13				X	X	X	
Standard 2	Align left 10	Follow 5	Arial 13				X	X	X	
Headline 1	None 0	Follow 0	Arial 13	X	X		X	X	X	
Headline 2	None 0	Follow 10	Arial 13	X	X		X	X	X	
Formula 1	None 200	On line 0	Arial 13				X	X	X	
Formula 2	Follow 30	On line 0	Arial 13						X	
Remark 1	None 50	Follow 5	Arial 13			X	X	X	X	
Remark 2	Follow 5	On line 0	Arial 13	X		X	X	X	X	



If a component is manually edited after a style has been assigned to it, the selected predefined style is no longer active, although its settings are used as a starting point for further style adjustments.

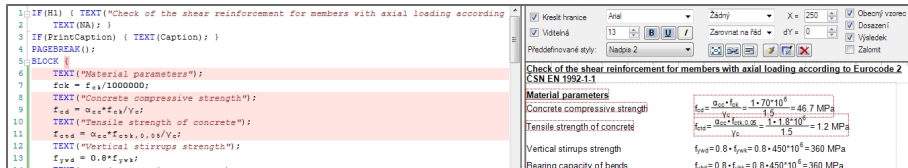
It is also possible to easily copy styles from one calculation form to another. If a visual component (text, equation, etc.) is copied to another form (using the code), the style of the component is automatically preserved in the destination form. This allows the user to use the clipboard functionality for the sole purpose of defining new styles in another form.

Copy formatting from one layout to another

Formatting can also be copied from one layout to another. The functionality allows for copying formatting to selected part of the code or to the whole layout.

How to copy to a selected part:

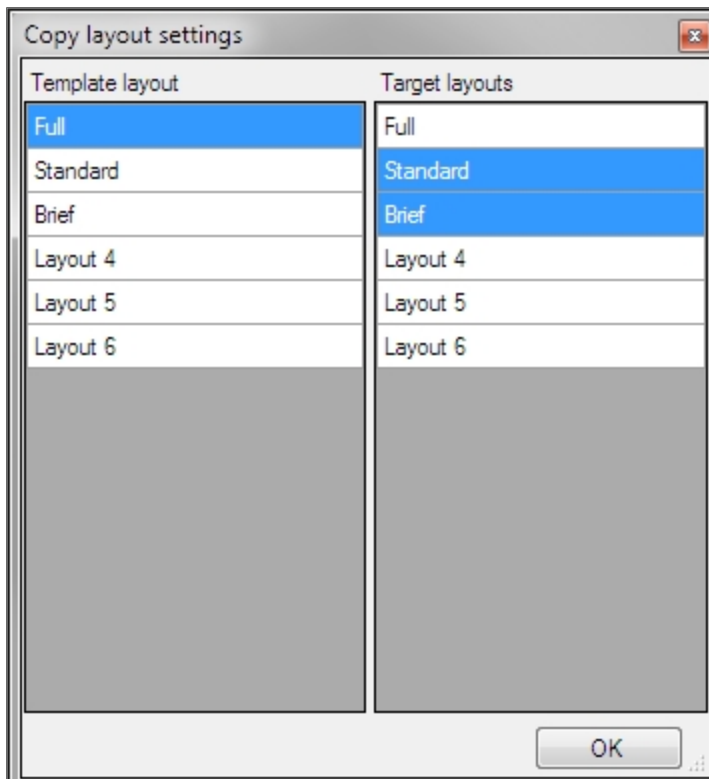
1. Select the components in the layout (or in the source code), use Shift key for selecting multiple components;



2. Click on the button "Copy properties ..." in the properties settings area;



3. Select the layout to the left as source layout;



4. Select layout(s) to the right as target layout(s);
5. All formatting should be copied from the source to target layouts.

How to copy formatting to a whole layout:

1. Click on the source code and use CTRL+A - the whole code is highlighted and all components in the layout are selected;

The screenshot shows a software interface with a code editor on the left and a properties panel on the right. The code editor contains the following code:

```

1: IF (N1) { TEXT("Check of the shear reinforcement for members with axial loading according
2:
3: IF (PrintCaption) { TEXT(Caption); }
4: PAGEBREAK();
5: BLOCK (
6:
7: TEXT("Material parameters");
8: fck = fck/1000000;
9: TEXT("Concrete compressive strength");
10: fcd = ηc * fck / γc;
11: TEXT("Tensile strength of concrete");
12: fctd = ηct * fctk * scs / Vcs;
13: TEXT("Vertical stirrups strength");
14: fyd = 0.8 * fyk;
15: TEXT("Bearing capacity of bonds");
16: fbd = 0.8 * fyk;
17: TEXT("Cross section axial stress") }

```

The properties panel on the right shows the following table:

Check of the shear reinforcement for members with axial loading according to Eurocode 2 CSN EN 1992-1-1	
Material parameters	
Concrete compressive strength	$f_{cd} = \frac{1.70 \cdot 10^6}{1.5} = 46.7 \text{ MPa}$
Tensile strength of concrete	$f_{ctd} = \frac{1.5}{1.5} = 1.2 \text{ MPa}$
Vertical stirrups strength	$f_{yd} = 0.8 \cdot f_{yk} = 0.8 \cdot 450 \cdot 10^6 = 360 \text{ MPa}$
Bearing capacity of bonds	$f_{bd} = 0.8 \cdot f_{yk} = 0.8 \cdot 450 \cdot 10^6 = 360 \text{ MPa}$
Cross section axial stress	$\sigma_{cp} = \frac{N_{ed}}{A_c} = \frac{20000}{4700} = 0.417 \text{ MPa}$
Coefficient of strain in comp. chord	$\eta_c = 1.0$

2. Click on the mentioned button in the component properties;
3. Select the layout to the left as source layout;
4. Select layout(s) to the right as target layout(s);
5. The target layout is now formatted in the same way as source layout.

Creating the Dialogue

The calculation Dialogue is one of the most important parts of the form. It contains the list of input variables which have to be determined by the end-user in the USER application.

All variables of the types double, string and boolean can be added to the Dialogue from the Table of variables. Variables that are not defined or calculated in the code itself will be displayed in green and will be editable. Other variables will be displayed only as additional information that cannot be edited.

Calculation of the anchor length

Concrete section library
Steel section library
Steel library
Concrete library
Timber library
Bolts library
Custom library

Double String Boolean >>>

ID	Description	Symbol	Value	Dim
		m	3.14	
	The tension bars diameter	ϕ	8	mm
	Number of reinforcement bars	n	3	
	Reinforcement area	A_{s1}	151e-6	m ²
	Action bending moment	M_{Ed}	50	kNm
	Inner lever arm of forces	z_b	25	mm
	Stress for calculation of ancho...	σ_{Ed}	13270	MPa
	Design value of concrete tensio...	$f_{ct,d}$	1.47	MPa
	Coefficient related to the qual...	η_1	1	
	Coefficient of bar diameter	η_2	1	
	Design value of cohesion of ste...	f_{bd}	3.3	MPa
	Coefficient of bar form	α_1	1	
	Coefficient of concrete cover	α_2	1	
	Coefficient of confinement by t...	α_3	1	
	Coefficient of welded transvers...	α_4	1	
	Coefficient of pressure transve...	α_5	1	
	Basic anchorage length	l_{brgd}	8.04	m
	The designed anchorage length	l_{bd}	8.04	m
	Coefficient of the concrete str...	σ_{cc}	1	
CONC...	Characteristic axial tensile st...	f_{ctk}	2.2e6	
CONC...	Partial safety factors for conc...	γ_c	1.5	
		γ	1	
		l_{brgd}	2.41	m
		l_{brgd}	4.83	m

<<< C 35/45 >>>

σ_{cc} 1.00 γ_c 1.50

Print headline
National code: ČSN EN 1992-1-1 §8.4.3. §8
 Print userdefined headline
Userdefined headline: uživatelský popis

Loading:
Action bending moment: M_{Ed} 50 kNm
Inner lever arm of forces: z_b 25 mm

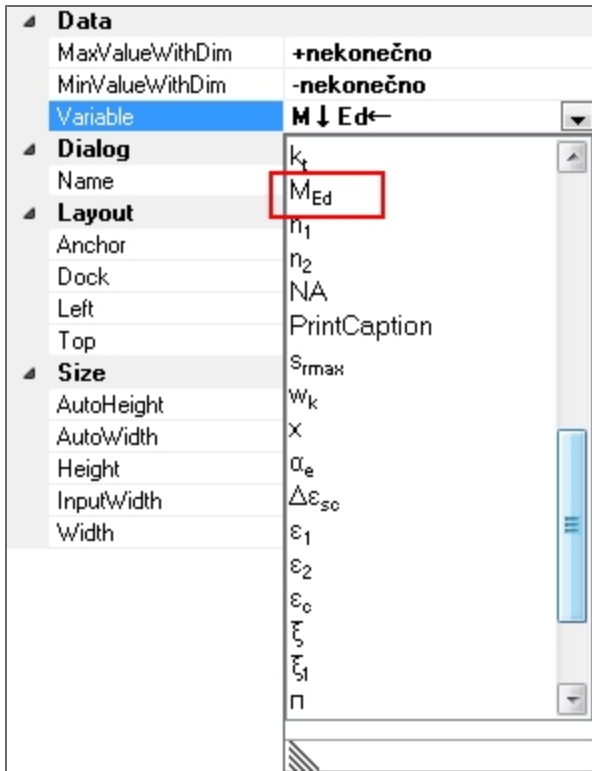
Reinforcement:
Number of reinforcement bars: n 3
The tension bars diameter: ϕ 8 mm

Reinforcement stress:
Stress for calculation of anchorage length: σ_{Ed} 13270 MPa

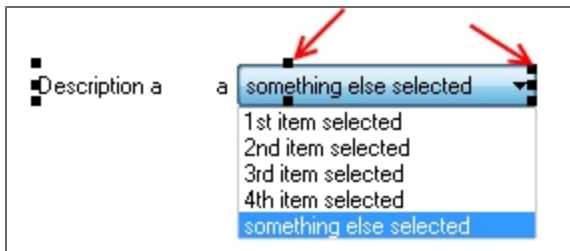
Coefficient of anchorage length:
Coefficient of the concrete strength: α_{cc} 1
Coefficient related to the quality of bond condition: η_1 1
Coefficient of bar diameter: η_2 1

Coefficients:
Coefficient of bar form: α_1 1
Coefficient of concrete cover: α_2 1
Coefficient of confinement by transverse reinforcement: α_3 1
Coefficient of welded transverse bars: α_4 1
Coefficient of pressure transverse to the plane: α_5 1
 n 3.14

An already existent component in the dialogue can be linked to any variable from the table of variables. The list of available variables can be found in the properties of dialogue components.



The size of each component can be edited by the black dots on the component borders or it can be edited numerically in the properties, by defining the size, location, docking, etc.



Size	
AutoHeight	True
AutoWidth	False
Height	642
Width	318

The dialogue is displayed in the left pane of the User application:

Design and check of the anchorage length according to ČSN EN 1992-1.1 par. 8.4.3 and 8.4.4

Reinforcement definition

Number of bars $n=3$
 Reinforcement diameter $\phi=0.012$ mm
 Reinforcement area $A_{st}=n \cdot \pi \cdot \frac{\phi^2}{4} = 3 \cdot 3.14 \cdot \frac{0.012^2}{4} = 339 \cdot 10^{-6} \text{ m}^2$

Stress for the calculation of the anchorage length $\sigma_{sd} = \frac{M_{Ed}}{z_B \cdot A_{st}} = \frac{50000}{0.25 \cdot 339 \cdot 10^{-6}} = 590 \text{ MPa}$

Coefficient related to the quality of bond condition $\eta_1=1$
 Coefficient of the bar diameter value $\leq \phi \cdot 32 \text{ mm}$ $\eta_2=1$
 Coefficients: $\alpha_1=1, \alpha_2=1, \alpha_3=1, \alpha_4=1, \alpha_5=1$

Design of anchorage length:

Design value of concrete tension strength $f_{ctd} = \frac{0.4 \cdot f_{ctk}}{\gamma_c} = \frac{1 \cdot 2.2 \cdot 10^6}{1.5} = 1.47 \text{ MPa}$
 Design value of the ultimate bond stress for ribbed bars $f_{bd} = 2.25 \cdot \eta_1 \cdot \eta_2 \cdot f_{ctd} = 2.25 \cdot 1 \cdot 1 \cdot 1.47 \cdot 10^6 = 3.3 \text{ MPa}$

Basic anchorage length $l_{Bd} = \frac{\phi}{4} \cdot \frac{\sigma_{sd}}{f_{bd}} = \frac{0.012}{4} \cdot \frac{590 \cdot 10^6}{3.3 \cdot 10^6} = 0.536 \text{ m}$
assumption: f_{bd} is constant on the whole bar

Check of the anchorage length $l_{Bd} \geq l_{Bd, min}$

Tensile anchorage $l_{Bd} = \text{Max} \left\{ \begin{matrix} 0.3 \cdot l_{Bd} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.3 \cdot 0.536 \\ 10 \cdot 0.012 \\ 0.1 \end{matrix} \right\} = 0.161 \text{ m}$
 Tensile anchorage length must be bigger than 0.161 m \Rightarrow is SUFFICIENT $\Rightarrow 0.161 \text{ m} < 0.536 \text{ m}$

Compression anchorage $l_{Bd} = \text{Max} \left\{ \begin{matrix} 0.6 \cdot l_{Bd} \\ 10 \cdot \phi \\ 0.1 \end{matrix} \right\} = \text{Max} \left\{ \begin{matrix} 0.6 \cdot 0.536 \\ 10 \cdot 0.012 \\ 0.1 \end{matrix} \right\} = 0.322 \text{ m}$
 Compression anchorage length must be bigger than 0.322 m \Rightarrow is SUFFICIENT $\Rightarrow 0.322 \text{ m} < 0.536 \text{ m}$

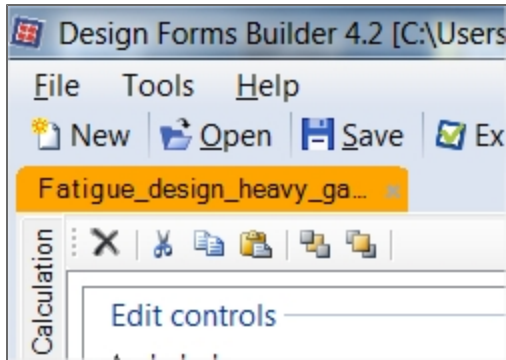
The dialogue has been improved in version 5.0 but remains fully compatible with versions 4.0 and 4.1.

Shortkeys:

- Ctrl + C - copy a component;
- Ctrl - V - paste a component;
- Ctrl + X - cut a component;
- PageUp - move component one level higher in the list;
- PageDown - move component one level lower in the list;
- Home - move component to the beginning;
- End - move component to the end;

Only docked components can be moved by shortkeys.

Buttons on the top ribbon allow for components to be deleted, cut, copied, pasted, moved to front and moved to back.



Form dialogue

Components in the dialogue - variables, text descriptions, labels and comments, pictures, etc. can be arranged in many ways. Since version 5.0, these can be placed in containers, organized in tabs, groups, tables; they can be docked or aligned, arranged in columns, or moved freely in the dialogue space.

Inserting variables in the dialogue.

Variables can be added to the Dialogue in two ways:

Drag and drop by the mouse

- Select the variable which should be added to the Dialogue;
- Click and hold the left mouse key;
- Drag it to the appropriate part of the Dialogue.
- The component is added as un-docked

Press the '>>>' button

- Select the variable which should be added to the Dialogue in the table of variables;
- Press the '>>>' button.
- The component is added as un-docked

Select multiple dialogue items by the CTRL or SHIFT key (Windows standard). The selection is marked in blue.

Changing the order of variables:

The order of variables can be changed in three ways:

Drag and drop by the mouse (if not docked)

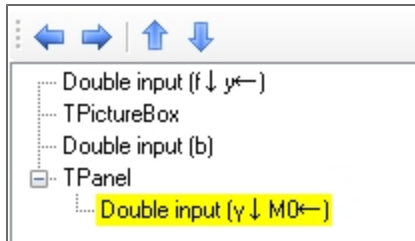
- Select the variable which should be moved;
- Click and hold the left mouse key;
- Drag the variable to the appropriate part of the Dialogue

Using the keyboard (if docked)

- Select the variable which should be moved;
- Press the keys PageUp / PageDown on the keyboard to move the variable one step or use Home / End to move it to Top / Bottom (components must be docked for this type movement)

Using the component tree

- Select the component/components in the tree (dialogue outline) or in the dialogue;



- Use the arrows on the top bar:



Components within the visual layers

Since version 5.0, the concept of layers is introduced to the input dialogue. This means that variables can be stacked on top of each other with only the top one being accessible. There are ways to avoid unwanted stacking - using the dialogue outline, docking, etc. Nevertheless, if a component is hidden behind other items in the dialogue, then there are buttons to bring it to front view, or put it back, behind the other components.



The layer concept is useful for un-docked components. The same buttons can be used for docked components, but will achieve a different result. The button 'Bring to front' will put a docked component on top of the list, whereas 'Bring to back' will send it to the bottom of the list. The buttons work in a similar way for components linked to containers - group boxes, panels, etc.



When the buttons move component to the top / bottom (using buttons)

Here is a list of cases when this function moves components to the top/bottom:

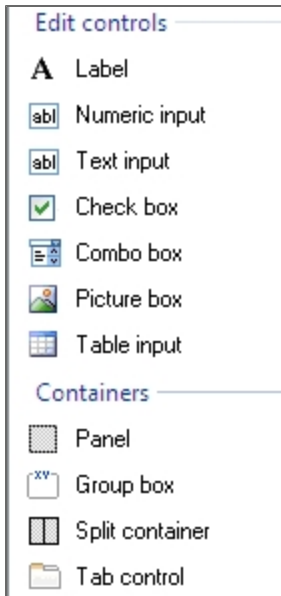
- If the component is **docked**, is it moved to the top / bottom of the list of all docked components.
- If components is on the **component typu container** - panel, group box ...

Form dialogue - adding and formatting components

Adding components to the dialogue

Drag and drop by mouse

- Select the component(s) which should be added to the Dialogue from the list on the left side;
- Press and hold the left mouse button;
- Drag and drop the variables to the Dialogue.



Double-click

- Select a component from the list on the left;
- Use double-click;
- The component is added to the Dialogue.

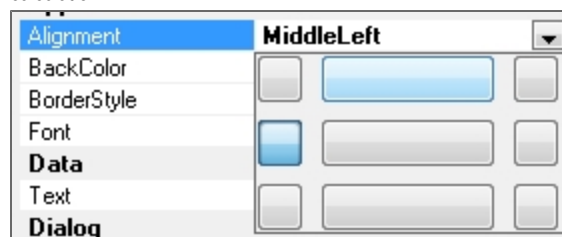
Component types

Edit controls - simple component types which can be filled by any available variable from the table of variables or have some special function (picture box, table input). Standard settings are available for these items.



Label

A simple description; it can be aligned anywhere in the dialog space; it is not binded to a variable and has no effect on the calculation.



Numeric input

Simple numeric input - it can contain a description of the variable, a symbol and box for input with units. An additional feature is that limits - minimal and maximal values, can be defined in the properties window.

Text input

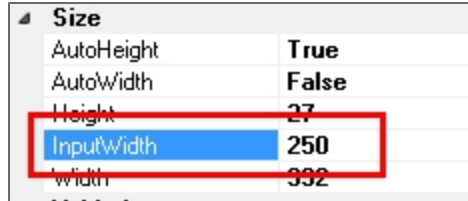
Simple text input - it can contain variable description and box for text input. It is meant for the definition of string variables.

Checkbox

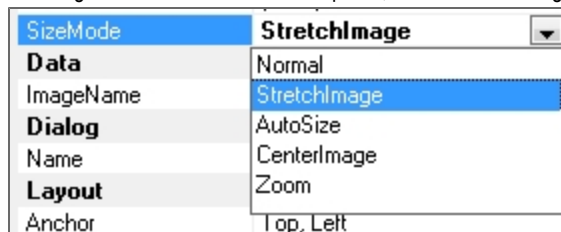
Check-box with standard options. It is best used for boolean variables.

Combobox

A combobox menu with standard options - an index and description, the width of the input box can be edited, see more in a [separate chapter here](#).

**Picture box**

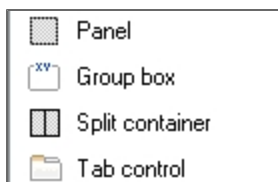
Allows for insertion of pictures - the picture name is defined in properties window; the image must be added to the vertical tab 'Image'. In addition to the standard options, sizes can be edited graphically.

**Table input**

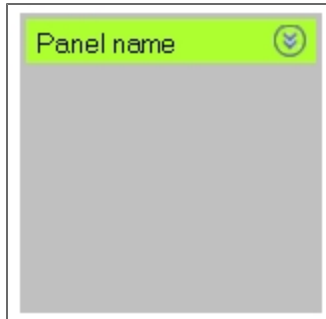
Inputs values to a 2D table, see [the separate chapter](#).

	Active	Normal force [kN] [kN]	Bending moment [kNm] [kNm]	Note
▶ 1	<input type="checkbox"/>			
* 2	<input type="checkbox"/>			

Containers - special type of components which cannot be binded to a variable, but allow user to arrange and represent variables in more ways.

**Panel**

Background for components - has a reference name; a special functionality can convert it to a combobox.



Group box

Background under components with borders and a name. Use property padding to set offset for all inserted variables.

Split container

Background under components, vertically split into parts. Each part can be hidden separately, the width of both parts is editable in the properties. Split containers just split the dialog area, these do not contain any text or other settings.

Tab control

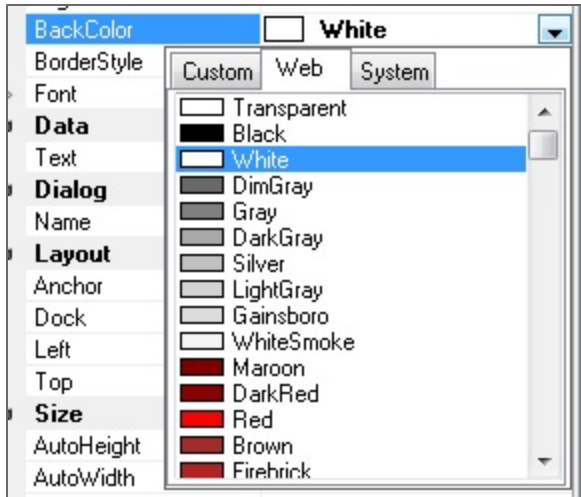
Background under components, organized into tabs. Variables can be grouped on the tabs.

Containers can be inserted into one another infinitely.

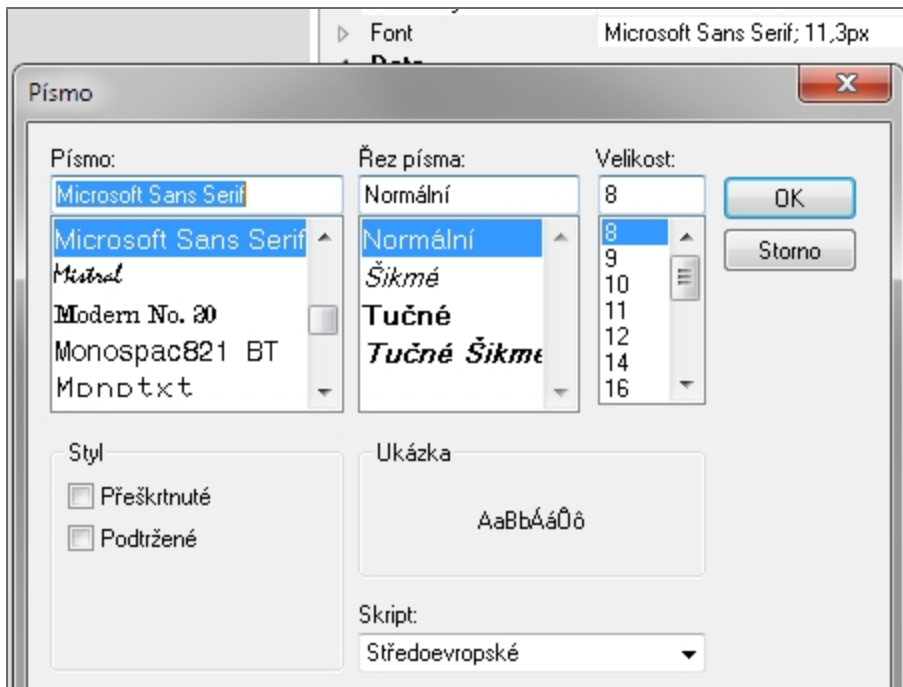
Standard component properties

Appearance	
Alignment	MiddleLeft
BackColor	<input type="checkbox"/> White
BorderStyle	None
Font	Microsoft Sans Serif; 11,3px
Data	
Text	label1
Dialog	
Name	
Layout	
Anchor	Top, Left
Dock	None
Left	147
Top	843
Size	
AutoHeight	True
AutoWidth	False
Height	23
Width	300

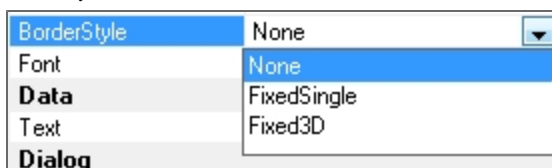
- background colour;



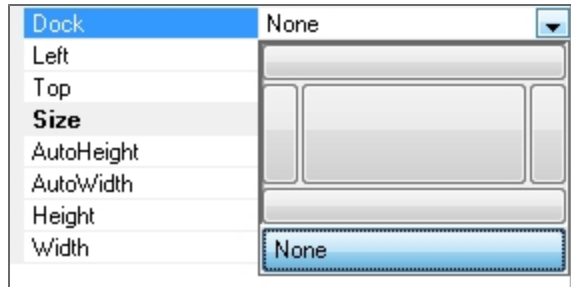
- colour of input box;
- variable description - dock to the left, top or invisible;
- font (the picture is in CZ because of the system language);



- variable symbol visibility - True, False;
- border style;



- alignment and docking;



- size (this property is active only when the variable is not docked = dock setting 'None'), when the automatic height / width is used, the size is calculated automatically, it cannot be changed manually;

Size	
AutoHeight	True
AutoWidth	False
Height	23
Width	300

- width of the input box;

Size	
AutoHeight	True
AutoWidth	False
Height	29
InputWidth	250
Width	392

Dialogue - table input

It is possible to insert into a dialogue a special component for definition of calculation input data - table input. This component is displayed as a table in which the user can insert an arbitrary number of rows, i.e. input values. Number and names of the columns are specified by the user during creation of the dialogue.

The values input by the user in the table (dialogue in the User application) can be later processed in the code editor using the FOR command.

When a new variable for the table input is being declared, it is not possible to use **initialisation** (=new Object[]) as for other variables of the object type. Reset of the variable would be performed after the input data are read from the dialogue and the user would thus lose the originally input data!

Table input example:

1. A new object-type variable is defined in the code editor.

```
object[] Combi;
```

2. A table input component is inserted in the dialogue.

	Active	Normal force [kN] [kN]	Bending moment [kNm] [kNm]	Note
▶ 1	<input type="checkbox"/>			
* 2	<input type="checkbox"/>			

3. The table has columns and descriptions defined.

Description	Symbol	Type	Unit
Active	Active	Bool	
Normal fo...	N	Numeric	kN
Bending m...	M _y	Numeric	kNm
Note	Note	String	

4. Table variables are in the code called using the dot convention.

```
Combi[i].Active) {
  σ = Combi[i].N/A+Com
```

5. The FOR cycle takes all the input values from the table and calculates the required result.

```
FOR(i, 0, Combi.Count-1) {
  IF (Combi[i].Active) {
    σ = Combi[i].N/A+Combi[i].My/I;
    σmax = max(σmax, σ);
  }
}
```

[Load an example: TableInputExample.cls](#)

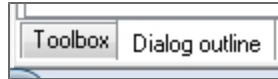
Table input function:

- <variable_name>.Count - displays number of table rows (as they were filled in in the dialogue in SDF User)
- <variable_name>[<index>].N - displays the value from the table for the given index and column N

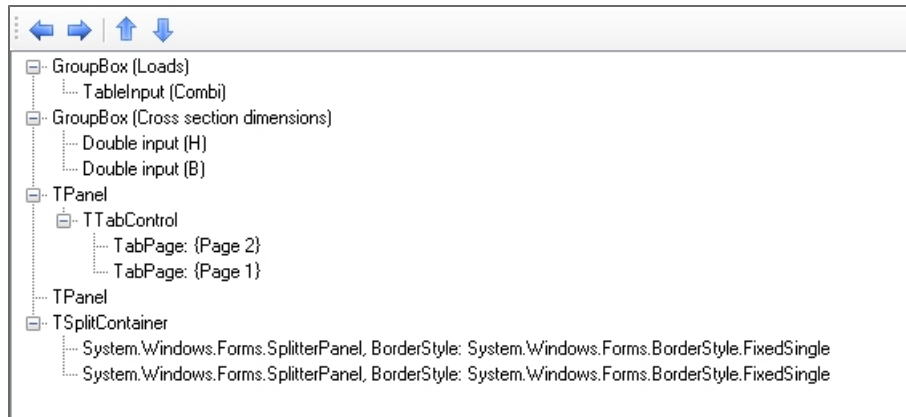
Dialogue item tree

Dialogue components can be displayed in the form of a tree that shows relations and connections of individual components.

1. This preview can be displayed by switching the tab at the bottom part of the variable table.



2. The tree structure of the dialogue is displayed instead of the component types.



3. Components can be embedded or moved to another level using the arrows on the bar.



Special variable IO - input/output

From the 'Dialogue' tab of the BUILDER application, different libraries can be included in the calculation form - Steel sections, Concrete sections, Concrete material library acc. to EC 1992, etc. After a library has been inserted in the dialogue, all data instances in it can be referenced by variables in the form.

Since version 4.1, access to library items is improved. The libraries included in the form dialogue save their data to a new type of system variable, referred to as 'IO' (InputOutput).

How to load values from the IO variable:

1. Directly in the source code, for example:
 - $H = IO.CS.Geometry.H$; - this command saves the cross-section height to the variable H;
 - $Polygon = IO.CS.Component.Shape.Point$; - this command saves an array of nodes which define the selected cross-section shape;
 - $Reinf = IO.Reinf.General$; - this command saves the structured variable with general characteristics of the selected reinforcement;

Available IO values are automatically suggested by an Intellisense function when the user writes "IO."

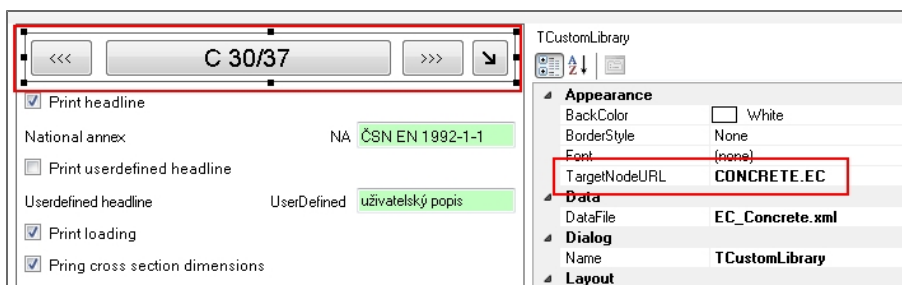
More information can be found here: ["Target IO node - how to have one library appear multiple times in the dialogue."](#)

2. By ID - IO values can be accessed by ID in the same way as in previous versions. The system checks all variables with IDs when the calculation starts. The system automatically adds an "IO.<ID>" to each ID. Thus, the syntax of the formula $V = IO.<ID_variable_V>$ in the source code gives the same result as when using the ID.
 - If the IO value linked to an ID is not found, the default value remains unchanged.
 - Variables with ID defined as "Result...." are skipped during a check.

Inserted libraries in the dialogue also have properties. A 'Target IO node' in the properties window defines how the library in question will be referred by the IO-syntax. For example, the concrete library items - characteristic compressive strength, elastic modulus, can be referred by the syntax $\langle variable \rangle = IO.Concrete.EC.<item_symbol \rangle$; . All library items from the concrete material library (EC 1992) will be saved to the node CONCRETE.EC.

To view or change the target IO node per library:

- Insert the library in the dialogue and select it;
- Write the path to the node in the cell "Target IO node".



If a CLS contains only one library, the IO node reference is not required, as all values are store at the highest level of the IO variable. The user may use the dot convention and write the required value symbol right after IO.

The screenshot shows a software interface with two main components. On the left, a window titled 'IO.' displays a list of variables: Add, Contains, Diagram, Ecm, eps_c1, eps_c2, eps_c3, eps_cu1, eps_cu2, eps_cu3, fck, and fckcube. The 'Ecm' variable is highlighted. Below this list is a text editor window with a toolbar and a 'Refresh' button. The text editor contains the following code: `1 TEXT("Ecm from concrete library: " & IO.Ecm);`. To the right of the text editor is a properties panel with the following settings: 'Draw borders' checked, 'Visible' checked, font set to 'Arial' size '13', and 'Predefined styles' set to 'Ecm from concrete library: 27*10⁹'. The text editor also shows a preview of the output: 'E_{cm} from concrete library: 27*10⁹'.

How to create a combo-box

Definition of combo-box items

The combo-box component can be used for variable types Numeric and String.

A numeric variable is defined by the index of the selected item in a list (indexes are numbered from 0).

The combo-box component can be inserted into the dialogue and its content can be then defined via its properties. Each list item is defined by its index and description.

We recommend that a variable of the Numeric type be used as it has wider capabilities including a direct use in the calculation or in a condition.

If a variable of the Numeric type is used, the list returns the index of the selected item that is defined in the combo-box definition.

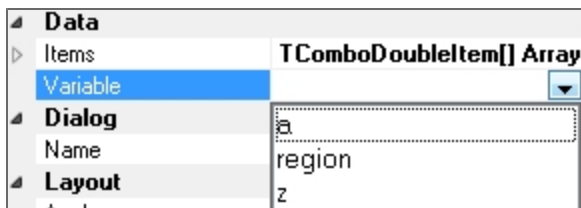
A String variable is defined as a text string of the selected item in the list.

How to define a combo-box menu

- When you add a combo-box component into the dialogue (by dragging or double-click), both the list and description are empty.



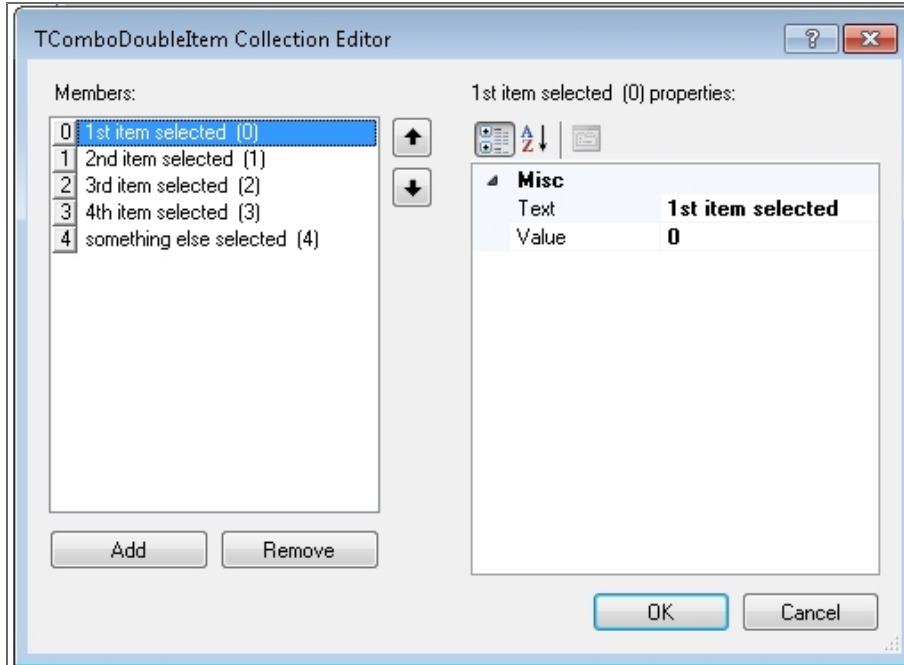
- In the properties, select the variable that is to be input through this component.



- Define the list - to display the dialogue, use button  that appears on the item in the properties.



- Type the list items in the dialogue - define the index and description.



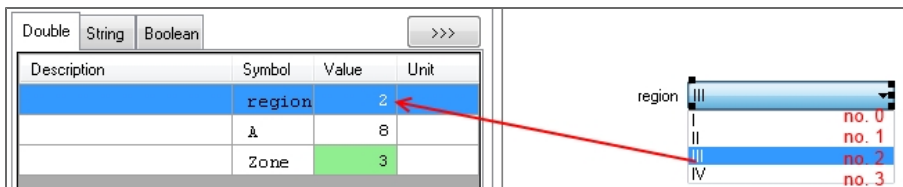
- Use the bottom button to add or delete individual items, if required.



- Use the arrows at the side of the list to change the order of the items in the list, if necessary.



The list item that is selected in the dialogue is indicated in the table of variables as an index. If you change the index in the table of variables, you also change the selection of the combo-box.



The list items do not have to be numbered in ascending order.

[Load an example containing four different types of combo-boxes.](#)

The items are translated.

ID	English (United States)	čestina (Česká republika)
CALC_NAME	Wind - Monopitch roofs	Větr - Pultové střechy
DIALOG_00001	0° 90° 180° 0° and roof loaded by pressure	0° 90° 180° 0° a střecha zatížena tlakem
DIALOG_000003	Parameters of the wind load:	Parametry zatížení větrem:
DIALOG_000016	Geometry of the roof:	Geometrie střechy:

How to use the items from a combo-box in the source code is described in chapter [Commands reference guide / Switch case](#).

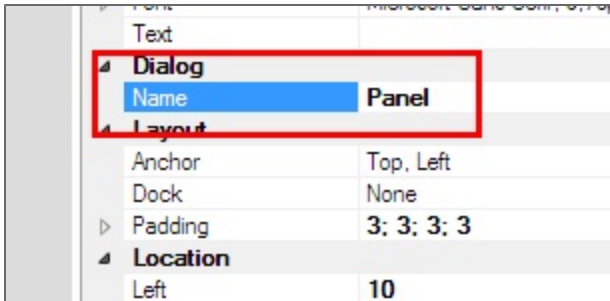
Hide dialogue components by using script

Syntax:

```
Dialog.GetComponentByName("<name of component>").Visible = false;
```

- this command sets the component <name of component> as hidden from within the source code.

The name of a component can be found in its properties window in the dialogue.



Example

```
IF(visible){ Dialog.GetComponentByName("Panel").Visible = true;
} ELSE { Dialog.GetComponentByName("Panel").Visible = false; }
- if the condition "visible" is true -> the dialogue component named "Panel" is visible, otherwise not
```

NOTE: Put all components which should be hidden in one panel. Switch off visibility for this panel.

Inserting and using Libraries in the Dialogue

Libraries contain organised sets of commonly used values that help the user increase productivity. The availability of libraries often eliminates the need to use external data sources, such as catalogues, brochures, design codes, etc. In Scia Design Forms, libraries are provided for characteristics of commonly used materials - steel, concrete, timber, for cross-section properties for commonly used profiles, for bolt geometrical and material properties.

Libraries must first be added to the dialogue of the calculation form before these can be linked by ID to variables in the code.

Library items are referred to by IDs. The latter contain a reference to the library where the item is located, as well as a symbol to denote the property of interest in the library. In the screenshot below, STEEL.EC refers to the EC 1993 Steel library, whereas f_y links automatically to the yield strength.

STEEL.EC.fy	Yield strength	f_y	235	MPa	2
CS.Geometry.Wz		$W_{e1,z}$	8.65e-6		2
CS.Geometry.Wy	Elastic cross section modulus...	$W_{e1,y}$	53e-6	m ³	2
CS.Geometry.Wplz		$W_{p1,z}$	13.6e-6		2
CS.Geometry.Wply	Plastic cross section modulus...	$W_{p1,y}$	60.7e-6	m ³	2

More details are available in the chapter about IDs [here](#).

Inserting a library in the dialogue

- Select the library from the list (on the left panel);
- Use double click to add it to the dialogue.

or

- Select the library from the list (on the left panel);
- User drag and drop to add it to the dialogue.

The screenshot displays the Scia Design Forms dialogue interface. On the left, a tree view shows the 'Steel library' selected. Below it is a table of library items with columns for ID, Description, Symbol, Value, and Dim. The 'Yield strength' item is highlighted, showing a value of 235 MPa. On the right, the 'IPE 120' section properties dialogue is open, showing various parameters like 'National code' (ČSN EN 1993) and 'Loading' (Bending moment - axis y: 10 kNm, Shear force: 10 kN). A red arrow points from the 'Steel library' in the left panel to the 'IPE 120' dialogue on the right.

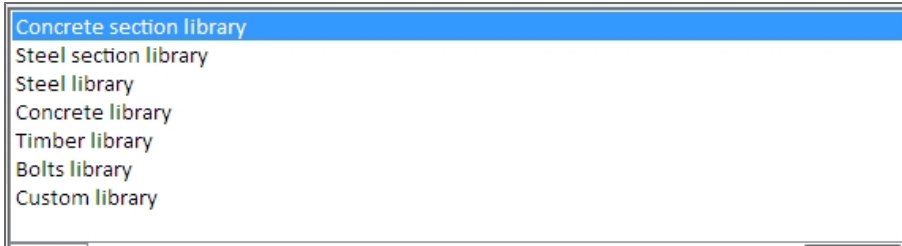
Libraries can be moved freely in the dialogue space, or can be docked and groups, as other components in the dialogue.

Deleting a library from the dialogue.

- Select the library that is already included in the dialogue;
- Select item "Delete" button from the top bar.

C 20/25		>>>	Library
α_{cc}	1,00	γ_c	1,50
Anal	10,00	B	U
<input type="checkbox"/> Print the headline of the design	Boolean		
Headline of the design	Reinforcement calculation		
The acting bending moment	M_{Ed}	15,0	kNm
Cross section dimensions:			
Cross section height	h	300	mm
Cross section width	b	200	mm
Effective distance for tension reinforcement	d_1	31,0	mm
Effective distance for compression reinforcement	d_2	29,0	mm
Material characteristics:			
Deformation module of concrete	E	29000	Mpa
The flexural tensile strength of the concrete	f_{ctm}	2,20	MPa
Characteristic compressive strength of concrete	f_{ck}	20	MPa
Deformation module of steel	E_s	210000	MPa
Characteristic tensile strength of reinforcement	f_{yk}	500	MPa
Limit value for tensile reinforcement strain	ϵ_{yk}	0,00	‰
Ultimate concrete strain	ϵ_{cu3}	3,50	‰
Reinforcement design:			
Reinforcement cover	c	25,0	mm
Coefficients:			
Partial safety factor for concrete	γ_c	1,50	
Partial safety factor for reinforcement	γ_s	1,15	
Coefficient of action conditions	α_{cc}	1,00	

Standard libraries in the Dialogue



The standard libraries are predefined databases of cross-sections, material characteristics or other properties and parameters. Each item in a library can be linked to a variable by ID. ID references can be defined in both the table of variables and the code.

Concrete section library

The library defines the shape and reinforcement of a concrete [cross section](#).

A screenshot of the configuration dialogue for the Concrete section library. It features a dropdown menu for 'Cross section shape' with options: Rectangle (selected), T-Shape, and Custom. Below this are six input fields for reinforcement parameters, each with a numeric value and a unit of 'mm':

Cross section shape	Rectangle
Height of cross section	
Width of cross section	
Bars count at top surface	0
Diameter of bars at top surface	20,0 mm
Reinforcement cover at top surface	30,0 mm
Bars count at bottom surface	0
Diameter of bars at bottom surface	20,0 mm
Reinforcement cover at bottom surface	30,0 mm

Steel section library

The library defines the shape and bolt positions of steel [cross sections](#).

IPE 140			
Description	Symbol	Value	Units
Weight	g_{VL}	12,90	kg/m
Height	h	140,00	mm
Width	b	73,00	mm
Web thickness	t_w	4,70	mm
Flange thickness	t_f	6,90	mm
Radius of web-flange connection	r_1	7,00	mm
Radius of flange end	r_2	0,00	mm
Effective section height	d	112,20	mm
Area	A	1,64	$\cdot 10^3 \text{ mm}^2$

Steel library

The library defines the material characteristics of [steel](#) that are most often used in the design according to the standard EN 1993.

S 235 dle EC 1993-1-1			
Description	Symbol	Value	Units
Yield strength	f_y	235,00	MPa
Tensile strength	f_u	360,00	MPa
Elastic modulus	E	210,00	GPa
Shear elastic modulus	G	80,77	GPa
Poisson's ration	ν	0,30	-
Density	γ	7 850,00	kg/m ³
Coefficient of linear thermal expansion	α	12,00	10^{-6} K^{-1}

Concrete library

The library defines the material characteristics of [concrete](#) that are most often used in the design according to the standard EN 1992.

C 12/15			
α _{cc} 1,00		γ _c 1,50	
Description	Symbol	Value	Unit
Characteristic compressive cylinder strength	f _{ck}	12,00	MPa
Characteristic compressive cube strength	f _{ck,cube}	15,00	MPa
Average cylinder compressive strength	f _{cm}	20,00	MPa
Average axial tensile strength	f _{ctm}	1,60	MPa
5% fractile of cylinder tensile strength	f _{ctk, 0.05}	1,10	MPa
95% fractile of cylinder tensile strength	f _{ctk, 0.95}	2,00	MPa
Elastic modulus	E _{cm}	27,00	GPa
Concrete strain at characteristic strength	ε _{ct1}	1,80	‰
Concrete strain at ultimate strength	ε _{cu1}	3,50	‰
Concrete strain at ultimate strength	ε _{c2}	2,00	‰
Concrete strain at maximum strength	ε _{cu2}	3,50	‰
Concrete strain at characteristic strength (using...)	ε _{c3}	1,75	‰
Concrete strain at characteristic strength (using...)	ε _{cu3}	3,50	‰
Exponent according to Table 3.1 EC2	n	2,00	-

Timber library

The library defines the material characteristics of [timber](#) that are most often used in the design according to the standard EN 1995.

C14 (EN 338)			
Description	Symbol	Value	Units
Bending strength parallel to grain	f _{mk}	14,00	MPa
Tension strength parallel to grain	f _{t,0,k}	8,00	MPa
Tension strength perpendicular to the grain	f _{t,90,k}	0,30	MPa
Compression strength parallel to grain	f _{c,0,k}	16,00	MPa
Compression strength perpendicular to the grain	f _{c,90,k}	4,30	MPa
Shear strength	f _{v,k}	1,70	MPa
Youngs modulus parallel to grain	E _{0,mean}	7,00	GPa
5% fractile of Youngs modulus	E _{0,05}	4,70	GPa
Youngs modulus perpendicular to the grain	E _{90,mean}	0,23	GPa
Shear modulus	G _{mean}	0,44	GPa
Density	ρ _k	290,00	kg/m ³

Bolts library

The library defines the parameters of [bolts](#) in the metric system (ISO 898-1).

Bolt material according to EN 1993	8.8
Nominal bolt diameter	3.6
	4.6
	4.8
	5.6
	5.8
	6.8
	8.8
	10.9
	12.9

Custom library

A library that can be defined manually or can be loaded from an XML file.

See the chapter [Custom library](#).

Libraries

Please, find the chapter about adding and modifying libraries [here](#).

IDs in standard libraries consist of two parts:


1. **blue part** - target IO node - determines from which library the value should be read;
2. **red part** - indicates which property/parameter from the library should be used (see chapter about IDs in standard libraries).

The screenshot shows a software interface with a table of material properties and a detailed view of a specific property. The table has columns for Description, Symbol, Value, and Unit. The detailed view shows a table with columns for ID, Description, Symbol, Value, Unit, and Preci.

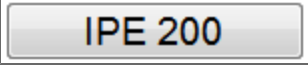
Description	Symbol	Value	Unit
Characteristic compressi...	f _{ck}	2,00	MPa
Characteristic compressi...	f _{ck,cube}	5,00	MPa
Average cylinder compre...			
Average axial tensile str...			
5% fractile of cylinder te...			
95% fractile of cylinder t...			
Elastic modulus			
Concrete strain at chara...			

ID	Description	Symbol	Value	Unit	Preci
CONCRETE . EC . f _{ck}		f _{ck}	12e6		2
CONCRETE . EC . f _{ck,cube}		f _{ck,cube}	15e6		2
CONCRETE . EC . f _{cm}		f _{cm}	20e6		2

Expanding/collapsing a library

- Use the buttons  /  to expand or collapse a library;

or

- Click on the cross section name  - the library is expanded automatically.

Go through items

- Use the buttons ">>>" or "<<<" to browse through the library.

Available cross-sections in the Steel section library

The steel cross-section library contains common rolled sections:

- I, IPE, IPN, HEA, HEB, HEM profiles;
- U, UPE, UPN, 2xU boxes;
- Circular tubes, square (box), rectangular sections (cold-formed, hot-rolled);
- L-sections - equal-leg angles;
- L-sections - unequal-leg angles;
- T-sections (rolled or cut from I-sections);
- Circular bars.

List of ID codes

Steel cross sections:

CS.SectionName	Cross section name
CS.SectionImage	Cross section type image
CS.Geometry.FormCode	Cross section type: 0 = NotDefined 1 = I, IPE, HEA, HEB, HEM 2 = Tube 3 = U, UPE 4 = square/ rectangular tube 5 = L 6 = T 7 = bar
CS.Geometry.H	Height
CS.Geometry.B	Width
CS.Geometry.tw	Web width
CS.Geometry.tf	Flange width
CS.Geometry.r1	Diameter of the connection between web and flange
CS.Geometry.r2	Diameter of the flange ends
CS.Geometry.d	Effective height of the web
CS.Geometry.A	Area
CS.Geometry.Avz	Effective shear area
CS.Geometry.Iy	Moment of inertia - y axis
CS.Geometry.Wy	Elastic cross section modulus y
CS.Geometry.Wply	Plastic cross section modulus y
CS.Geometry.iy	Radius of gyration - y axis
CS.Geometry.Iz	Moment of inertia - z axis
CS.Geometry.Wz	Elastic cross section modulus z
CS.Geometry.Wplz	Plastic cross section modulus z
CS.Geometry.iz	Radius of gyration - z axis
CS.Geometry.IT	Torsion moment of inertia
CS.Geometry.Iw	Warping constant
CS.Geometry.ss	Length of the rigid part of a flange
CS.Geometry.ay	Buckling coefficient - y axis
CS.Geometry.az	Buckling coefficient - z axis

Concrete cross sections:

CS.Geometry.H	Height
CS.Geometry.B	Width/Effective width
CS.Geometry.th	Web width
CS.Geometry.sh	Slab thickness
CS.Geometry.FormCode	Cross section type - according to Scia Engineer
CS.Reinf.n1	Number of bars on top
CS.Reinf.Cover1	Top bars concrete cover
CS.Reinf.φ1	Top bars diameter
CS.Reinf.n2	Number of bars on bottom
CS.Reinf.Cover2	Bottom bars concrete cover
CS.Reinf.φ2	Bottom bars diameter

New concrete cross sections:

The new concrete cross section library adds some functionality to the old one. It allows to use various cross section shapes, reinforcement templates or stirrup shapes.

[Load the example: ConcreteSectionLibrary.cls](#)

IO values:

CS.Component.Shape.Point	Array of outline points = cross section shape
CS.Geometry	Set of values which defines the cross section shape (dimensions)
Beam.Reinforcement.Bar	Array of bars of longitudinal reinforcement
Beam.Reinforcement.c_min	Min cover of longitudinal reinforcement
Beam.Reinforcement.c_max	Max cover of longitudinal reinforcement
Beam.Reinforcement.StirrupChars	Stirrups characteristics

Stirrup characteristics:

Beam.Reinforcement.StirrupChars.Asw	Area of all stirrups in one layer
Beam.Reinforcement.StirrupChars.ss	Average distance between layers
Beam.Reinforcement.StirrupChars.Asw_ss	Average area of stirrups per meter [m ² /m]
Beam.Reinforcement.StirrupChars.D	Average diameter of stirrup bar
Beam.Reinforcement.StirrupChars.Ns	Number of stirrup bars in one layer
Beam.Reinforcement.StirrupChars.Angle	Average angle of the stirrups (0°= horizontal, 90°= vertical)
Beam.Reinforcement.StirrupChars.Branch	Array of arrays of points which defines the stirrup shape. Example: Beam.Reinforcement.StirrupChars.Branch[0] contains an array of points which defines the shape of the first stirrup (index 0).

Defined steel classes

The library contains steel classes according to EN 1992-1-3, tab. 3.1 and EN 10025-2.

The list of ID codes

STEEL.EC.fy	Yield stress
STEEL.EC.fu	Ultimate strength
STEEL.EC.Es	Modulus of elasticity (compression, tension)
STEEL.EC.G	Shear modulus
STEEL.EC. ν	Factor of transverse deformation
STEEL.EC. γ	Density
STEEL.EC. α	Coefficient of linear thermal expansion

Defined concrete classes

The library contains concrete class definition according to EN 1992-1-1, tab. 3.1.

The list of codes ID

CONCRETE.EC.fck	Characteristic compressive cylindrical strength
CONCRETE.EC.fckcube	Characteristic compressive cubical strength
CONCRETE.EC.fcm	Mean value of compressive cylindrical strength
CONCRETE.EC.fctm	Mean value of axial tensile strength of concrete
CONCRETE.EC.fctk005	5 % fractile of the tensile strength
CONCRETE.EC.fctk095	95 % fractile of the tensile strength
CONCRETE.EC.Ecm	Modulus of elasticity
CONCRETE.EC.eps_c1	Compressive strain in concrete at limit stress f_c
CONCRETE.EC.eps_cu1	Ultimate compressive strain in concrete at stress f_c
CONCRETE.EC.eps_c2	Compressive strain in concrete at limit strength
CONCRETE.EC.eps_cu2	Ultimate compressive strain in concrete at maximum strength
CONCRETE.EC.eps_c3	Compressive strain in concrete at the limit strengths - according to bilinear diagram
CONCRETE.EC.eps_cu3	Ultimate compressive strain in concrete at maximum strengths - according to bilinear diagram
CONCRETE.EC.n	Exponent according to table 3.1 ČSN EN 1992-1-1

Defined timber classes

The library contains timber class definitions according to EN 338.

The list of codes ID

TIMBER.EC.fmk	Bending strength
TIMBER.EC.ft0k	Tensile strength parallel to grain
TIMBER.EC.ft90k	Tensile strength perpendicular to grain
TIMBER.EC.fc0k	Compression strength parallel to grain
TIMBER.EC.fc90	Compression strength perpendicular to grain
TIMBER.EC.fvk	Shear strength
TIMBER.EC.E0	Modulus of elasticity parallel to grain
TIMBER.EC.E005	5 percent fractile of the modulus of elasticity
TIMBER.EC.E90	Modulus of elasticity perpendicular to grain
TIMBER.EC.G	Shear modulus
TIMBER.EC. ρ	Density

Bolts definition

Defined bolt classes

The library contains material class definition according to EN 1993-1-8.

Defined bolt diameters

M8, M10, M12 M16, M20, M24, M27, M30, M36 (ISO 898-1).

Selection of material and diameter

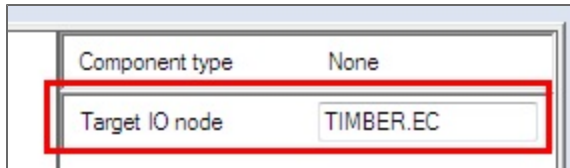
- Select the required bolt material from the combobox menu.
- Select the required bolt diameter from the combobox menu.

The list of ID codes

Bolt_D	Bolt diameter
Bolt_A	Bolt area
Bolt_Anet	Bolt area on threads
Bolt_fyb	Yield strength of the bolt material
Bolt_fub	Ultimate strength of the bolt material
Bolt_Material	Bolt material (double)

Target IO node - how to have one library appear multiple times in the dialogue

Since Scia Design Forms version 4.1, it is possible to use one type of library multiple times in the calculation. If, for example, two or more steel profiles need to be used in the calculation of a steel connection, the properties of each profile may be linked to a separate instance of the library and no conflicts will appear in the calculation form. The 'Target IO node' displayed in the properties window is used to distinguish between those two (or more) steel cross-section libraries.



All library values in Scia Design Forms are stored in the structured system variable IO (InputOutput). The target IO node is used to navigate in the structure of the IO variable. The user can define a path to each library item by using the target IO node name, and the name of the item itself. Therefore, the target IO node name must be unique.

This functionality allows for the user of two libraries of the same type in the same calculation form - two concrete material libraries, for example, where C12/15 is selected in the first library and C25/30 is selected in the second.

How to create/change an IO target node in the Dialogue:

- Go to the Dialogue (vertical tab of the BUILDER application);
- Select the library which has to be already added to the dialogue;
- Set the name of the target IO node in the properties.

Example

Two steel material libraries and two steel cross-section libraries are required for a connection check:

- the library for the COLUMN cross-section has Column in its Target IO node;
- the library for the BEAM cross-section has Beam in its Target IO node;
- the library for the COLUMN material has Column.material in its Target IO node;
- the library for the BEAM material has Beam.material in its Target IO node.

The access to each library and item is:

1. Directly in the source code (variable H, fy)
 $H_{col} = IO.Column.Geometry.H$; - The value from a library named Column is inserted to the variable H_{col}
2. By ID (variable E)
 ID is filled by $Column.Material.E$ - The value of the modulus of elasticity from the library named Column is inserted to the variable.

[Load the example: Libraries.CLS](#)

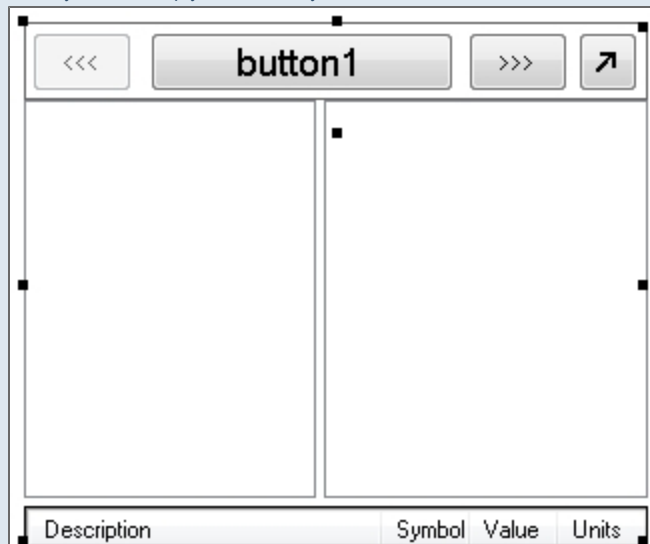
The custom library returns a value even for the required columns: Item and Category;
 e.g.. $IO.custom.Item$ or $IO.custom.Category$ (for the library which has target IO node = "custom").

Custom library

The custom library is a special component of the Dialogue, which allows the user to work with user-defined libraries and databases.

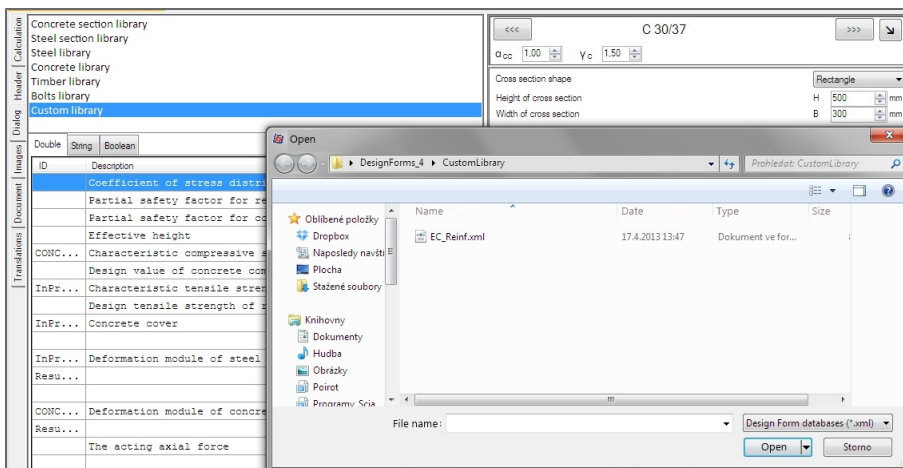
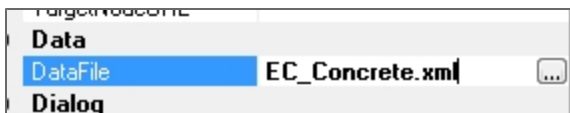
Example: A library which contains soil types, aluminium classes, imperial bolts types and so on.

A newly inserted empty custom library:



A custom library is loaded to the dialogue from an external database. Such a database should be saved at the location (a library cannot be loaded from a different folder): C:\Users\Public\Documents\DesignForms_4\CustomLibrary).

The XML database from where the data are loaded to the library is displayed in the library properties. By this same field in the properties, links to any XML can be defined.



Custom libraries can be used in the same way as the standard, hard-coded libraries. The library must be saved in the folder "Public\Documents\DesignForms_4\CustomLibrary\" in XML format.

If the user want to publish a calculation form with a custom library in it, the XML database file should also be included. The end-user should then save the XML file in C:\Users\Public\Documents\DesignForms_4\CustomLibrary\.

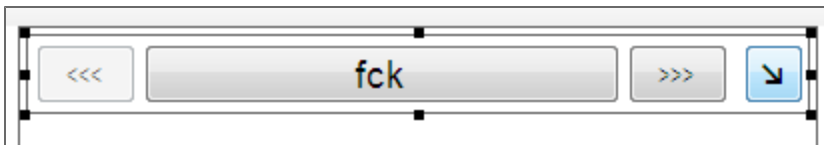
The SDF User application will automatically recognise the library and load it from the XML definition file.

The strings from a custom library is not translated in the Builder application.

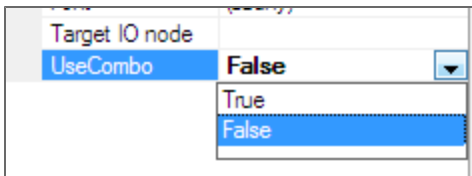
Custom library displayed as combo-box in the Dialogue

There is an option in the dialogue properties which defines the custom library shape. It may be displayed as library or as "combo-box".

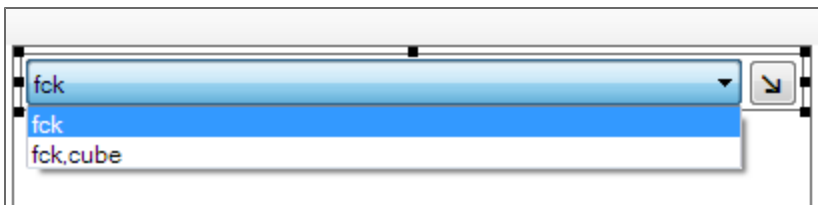
The library is displayed as "library":



The UserCombo property - False = displayed as "library", True = displayed as "combobox"



The library is displayed as "combobox":



Custom library Editor

The content of a custom library can be filled in from within the Scia Design Forms BUILDER. The Custom library Editor is used to introduced the data, and the created library is saved in XML format in the location C:\Users\Public\Documents\DesignForms_4\CustomLibrary\.

The interface of the Custom library Editor looks like a table - the user can simply fill in the data or paste it from another database, e.g. MS Excel. One row in the table refers to a single item in the library (e.g. in a section library - IPE 80, IPE 100, etc.). The columns in the table correspond to the different item properties (e.g. strength, density, etc.).

Category	Item	Characteristic yield strength	Coefficient k = f _{tk} /f _{yk}	Characteristic strain at maximum force	Design yield strength	Design yield strength	Limit strain	Class	Inclined plastic branch	Material diagram point	Material diagram point
Inclined	B 400 A	400	1.5	0.025	347.83	365.22	0.02	A	True	-0.0225;-365.22;	-0.0166;-347.83;
Inclined	B 500 A	500	1.5	0.025	434.78	456.52	0.02	A	True	-0.0225;-456.52;	-0.0207;-434.78;
Inclined	B 600 A	600	1.5	0.025	521.74	547.83	0.02	A	True	-0.0225;-547.83;	-0.0248;-521.74;
Inclined	B 400 B	400	1.8	0.05	347.83	375.65	0.05	B	True	-0.045;-375.65;	-0.0166;-347.83;
Inclined	B 500 B	500	1.8	0.05	434.78	469.57	0.05	B	True	-0.045;-469.57;	-0.0207;-434.78;
Inclined	B 600 B	600	1.8	0.05	521.74	563.48	0.05	B	True	-0.045;-563.48;	-0.0248;-521.74;

A change, damage or loss of the definition file (XML)

The definition file of the Custom library may accidentally be changed, completed or deleted.

How is the change displayed in the USER application:

The calculation is saved to the project (*.CLP file) with a value from the custom library. If the value is removed from the custom library, the project is loaded with this value, but the Dialogue will show a warning message. The user will see the value in the output and exporting or printing will not be hampered.

If the user would like to select a different value from the library, the original value will be lost and it would not be possible to load it again, as it is not in the library any more. Changes in the custom library are displayed upon opening the project after the change.

Example - a value is deleted in the custom library and the CLP project has to be repaired:

1 - the layout in the USER application displays the original value, which has already been deleted from the custom library;

2 - the user selects a different value from the library;

3 - the new value f_{yk} is loaded for the variable x.

How is the change displayed in the BUILDER application:

If a value used in a calculation form is deleted from the definition file, the layout would display a warning message that the value is no longer not found. User must select a different value from the dialogue.

Changes in a custom library are displayed when the *.CLS file is opened after the change.

Example - a value is deleted in the custom library and the CLS file has to be repaired:

The screenshot shows a software interface with a CLS layout editor. The top window displays a text field with the content: "1 TEXT ('check of custom library').", "2 x = IO.beam.fyk;". Below this, a warning message is shown: "check of custom library", "x=IO.beam.fyk=IO.beam.fyk = fyk - not such a field or property". A red box highlights this message with a red arrow pointing to the number 1.

The middle window shows a tab dialogue with a list of values: "B 600 A", "B 600 A", "B 600 A", "B 600 B", "B 600 B", "B 600 B", "B 600 C", "B 600 C", "B 600 C". A red arrow points to the first "B 600 A" entry with the number 2.

The bottom window shows the same list of values, but with "B 600 A" selected. A green box highlights the selected entry with the number 3.

The bottom right window shows a table with the following data:

Description	Symbol	Value	Units
Characteristic yield strength	f_k	600	MPa
Coefficient $k = f_t/f_{yk}$	k	1.5	
Characteristic strain at maximum force	ϵ_{uk}	0.025	
Design yield strength	f_{d1}	521.74	MPa
Design strength	f_{d2}	547.83	MPa
Limit strain	ϵ_{lim}	0.02	
Class	Class	A	
Inclined plastic branch	Inclined	True	
Material diagram point	Point	-0.0225-...	
Material diagram point	Point	-0.00248...	

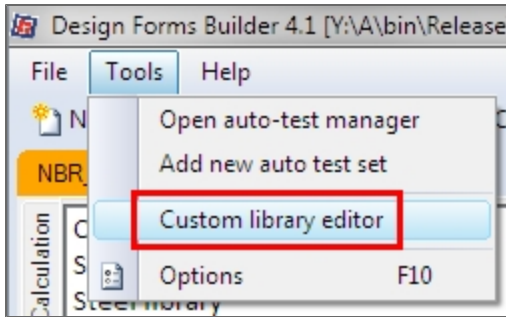
- 1 - A warning message appears in the CLS layout; the value is not found - the original value was B400 A;
- 2 - on the tab dialogue in the BUILDER application, a warning message shows that a value is missing;
- 3 - the user must select a different value to be used.

Custom library - manual definition and Editor

Custom library Editor

The Custom library Editor allows for new custom libraries to be created or existing ones to be edited. The Editor is accessible in the Scia Design Forms BUILDER from the menu item Main Menu > Tools > Custom library Editor.

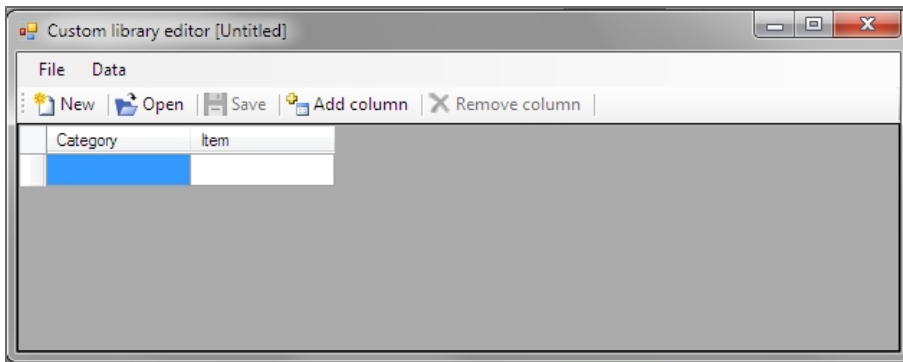
More about custom libraries can be found [here](#).



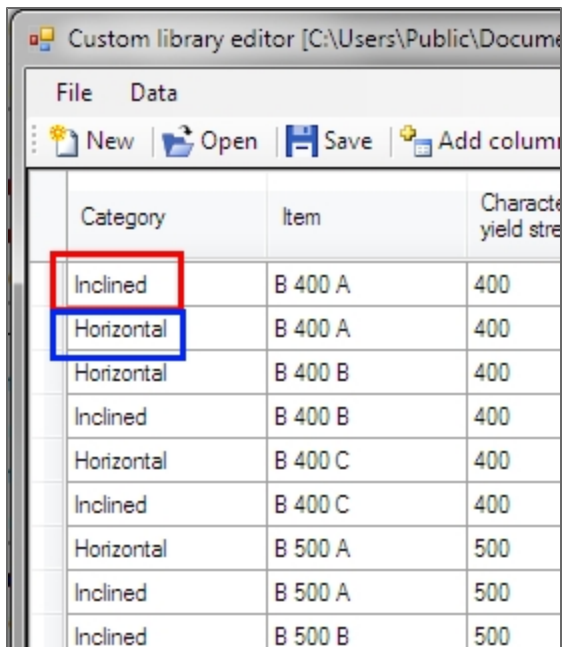
Existing custom libraries in XML files can be loaded to the Editor, where these can be edited, expanded and saved.

Category and Item

The two columns "Category" and "Item" are always displayed when the Editor is opened. These two columns define the distribution and names of library items. The columns are **required** for each custom library.



In a custom library, items are distributed automatically according to the values in the 'Category' column. The names of items are defined by the values in the column 'Item.' If the 'Category' values are the same for all items, the library will only display one list in the dialogue.



The screenshot shows a window titled "Custom library editor [C:\Users\Public\Docume...]" with a menu bar containing "File" and "Data". Below the menu bar is a toolbar with icons for "New", "Open", "Save", and "Add column". The main area contains a table with three columns: "Category", "Item", and "Character yield strength". The table has 11 rows. The first two rows have "Inclined" and "Horizontal" in the "Category" column, respectively. The "Item" column contains values like "B 400 A", "B 400 B", "B 400 C", "B 500 A", and "B 500 B". The "Character yield strength" column contains values like "400" and "500". A red box highlights the "Inclined" cell in the first row, and a blue box highlights the "Horizontal" cell in the second row.

Category	Item	Character yield strength
Inclined	B 400 A	400
Horizontal	B 400 A	400
Horizontal	B 400 B	400
Inclined	B 400 B	400
Horizontal	B 400 C	400
Inclined	B 400 C	400
Horizontal	B 500 A	500
Inclined	B 500 A	500
Inclined	B 500 B	500

The 'Category' and 'Item' columns cannot be deleted, these are always required.

The IO reference method returns values also for the obligatory 'Item' and 'Category' columns in the case of custom libraries, e.g.. IO.custom.Item or IO.custom.Category (for library with target IO node = "custom").

Displaying a custom library in the Dialogue

A custom library with **one category** inserted in the Dialogue:

B 400 A			
<ul style="list-style-type: none"> B 400 A B 500 A B 600 A B 400 B B 500 B B 600 B B 400 C B 500 C B 600 C 			
Description	Symbol	Value	Units
Characteristic yield strength	f_k	400	MPa
Coefficient $k = f_k/f_y$	k	1.5	
Characteristic strain at maximum force	ϵ_{uk}	0.025	
Design yield strength	f_{d1}	347.83	MPa
Design yield strength	f_{d2}	365.22	MPa
Limit strain	ϵ_{lim}	0.02	
Class	Class	A	
Inclined plastic branch	Inclined	True	
Material diagram point	Point	-0.0225; -...	
Material diagram point	Point	-0.00166...	

A custom library with **two categories** inserted in the Dialogue:

B 400 A			
<ul style="list-style-type: none"> Inclined Horizontal 		<ul style="list-style-type: none"> B 400 A B 500 A B 600 A B 400 B B 500 B B 600 B B 400 C B 500 C B 600 C B 400 A B 500 A 	
Description	Symbol	Value	Units
Characteristic yield strength	f_k	400	MPa
Coefficient $k = f_k/f_y$	k	1.5	
Characteristic strain at maximum force	ϵ_{uk}	0.025	
Design yield strength	f_{d1}	347.83	MPa
Design yield strength	f_{d2}	365.22	MPa
Limit strain	ϵ_{lim}	0.02	
Class	Class	A	
Inclined plastic branch	Inclined	True	
Material diagram point	Point	-0.0225; -...	
Material diagram point	Point	-0.00166...	

- Categories are displayed to the left, whereas items to the right.

How to create a custom library

Columns definition in the Editor

All columns (corresponding to properties) must be defined before the user starts filling the data.

Use button "Add column" to create a new property. These parameters must be defined in the pop-up dialogue:

- Description - descriptions are displayed in the custom library when it is inserted in the dialogue;
- Symbol - symbols are displayed in the custom library next to the description. Also, the symbol is displayed in the layout when an array is created from the custom library (see below). This field is required;
- Target node path - this is a reference name for the property, used when called using the IO variable. The target node path is suggested when the IO syntax is used (see more info about IO [here](#)). This field is required;

How is a path to a variable composed:

The screenshot illustrates the process of defining a column property. The main window shows a list of columns with properties like 'Characteristic yield strength' (symbol f_k , value 500 MPa). A dialog box 'TColumnHeaderEditor' is open, showing the configuration for a new column: Description 'Characteristic yield strength', Symbol ' f_k ', Target node path ' f_yk ', Property type 'Numeric', and Units 'MPa'. Red arrows indicate the flow from the 'Reinf_Eurocode' target IO node in the library to the 'Target node path' field in the dialog. Blue arrows show the variable assignment 'X = IO.Reinf_Eurocode.fyk;' in the calculation window and the resulting output 'X = IO.Reinf_Eurocode.fyk = 500*10⁶ = 500*10⁶' in the layout preview.

the part in red - the first part of the path is the 'Target IO node' name of the library (shown in the properties of the library in the Dialogue), the second part is the 'Target node path' in the column definition in the Editor (refers to a property in the custom library);

the part in blue - source code for assigning the property f_yk (from the library with target IO node name Reinf_Eurocode) to the variable X; the output in the layout preview is also shown.

Arrays in the Custom library

Under certain circumstances, the items in a custom library may be saved to an array.

When is an array created from inserted values in a custom library:

If there are more columns defined in the Editor with the same 'Target node path' then the items in these columns are converted to an array. The data are accessible from the course code. This functionality may be used in some special cases; the user may access the array through common array indexing.

Example of such an array: the Target node path of columns Catalogue number 1, Catalogue number 2, and Catalogue number n, etc. are the same. All variables are saved to one array Catalogue, and the user can access the data by index. See more information about [arrays](#).

Column properties

- Type:
 - number
 - text
 - boolean
 - structured
- Units - [physical unit](#) of the variable (activated only for numbers)

If units are not inserted, the variable is defined in SI units.

If the selected type is "structured", a new pop-up dialogue appears. The sub-properties are defined by this table. The items in the table are similar to the column properties (definition of the column).

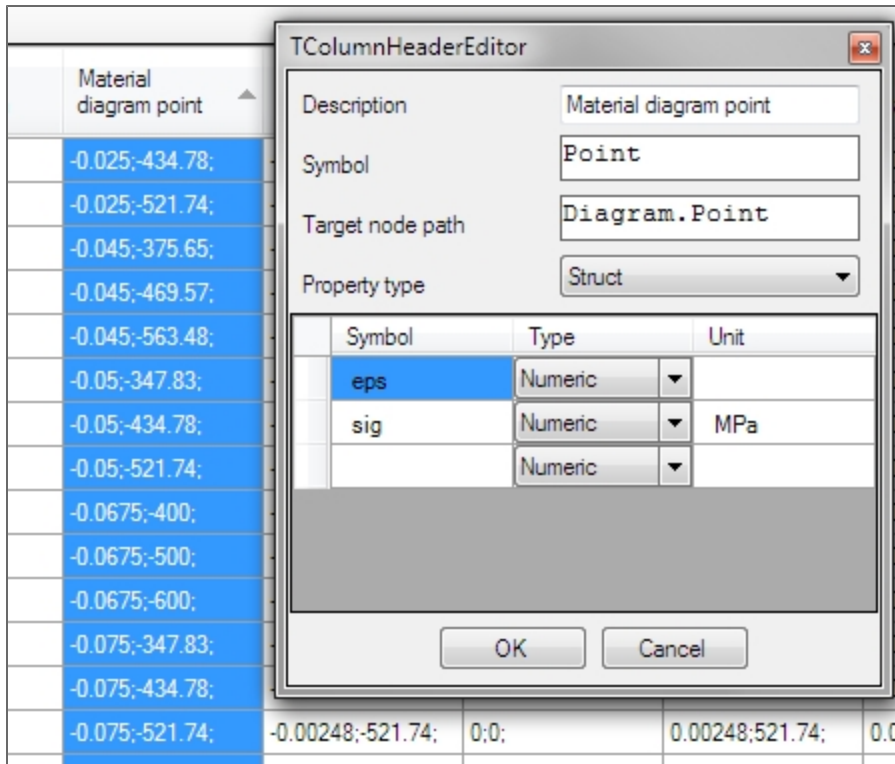
Example of a structured variable for a stress-strain diagram:

The properties are "eps" and "sig":

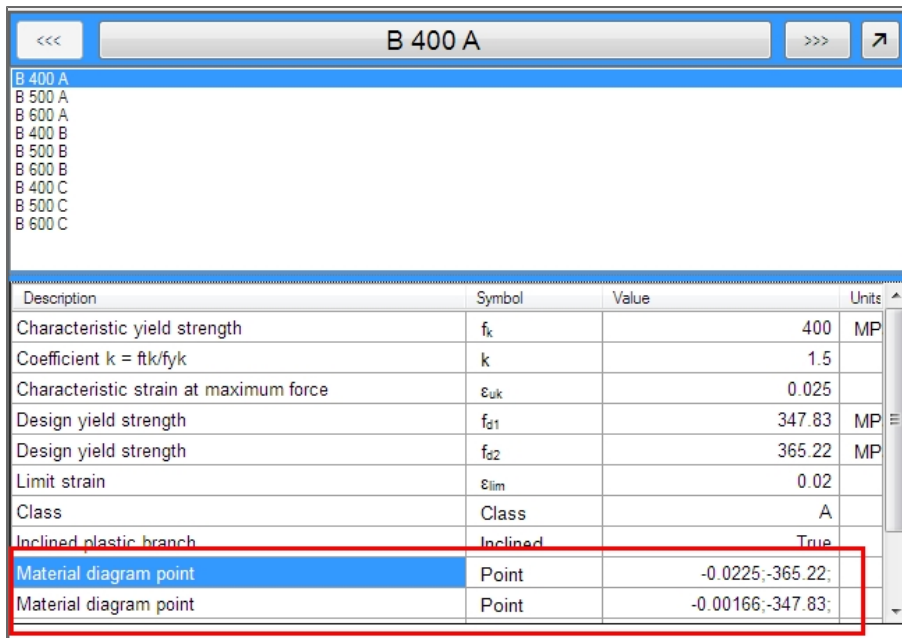
- 1) the property "eps" is inserted without units, therefore, it is interpreted in SI units.
- 2) the property "sig" has units defined as MPa, all inserted values for "sig" must be in MPa.

Description	Material diagram point		
Symbol	P		
Target node path	Diagram.Point		
Property type	Struct		
	Symbol	Type	Unit
	eps	Numeric	
	sig	Numeric	MPa
		Numeric	

Values should be split by **semicolons** when filled in a structured column.



The item displays the semicolons also in the inserted custom library in the Dialogue.

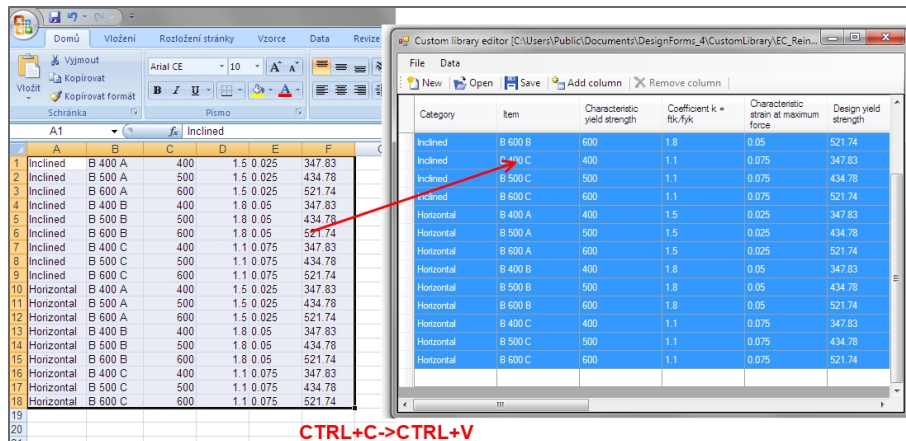


Custom library - automatically defined

Inserting values in the library

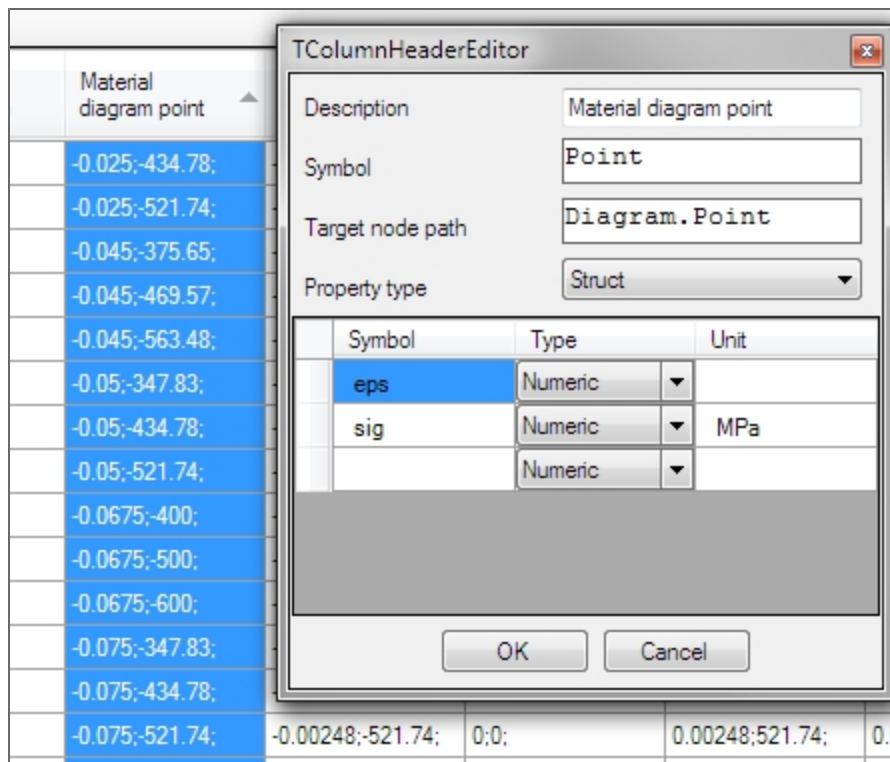
It is possible to insert the content of a library to the [Editor](#) if all properties (columns) are defined.

The content can be inserted manually or copied through the clipboard (CTRL+C, CTRL+V) from the external grid (e.g. MS Excel).



The following formats are required:

- Numbers - the **dot** decimal separator is required (not comma), exponential syntax can be used (e.g. 1.234e6)
- Texts - no limits
- Boolean - must be **"TRUE"** or **"FALSE"**
- Structured values - the values of sub-properties must be separated by **semicolon**. The semicolon is required after the last value also.



Calculation header

The calculation header contains basic data about the calculation, author and the used national standard.

- The calculation name - a name which is displayed in the SDF USER module;
- Author - the author of the calculation form. "Nemetschek SCIA" is reserved for forms developed by the Nemetschek Scia company;
- National annex (NA) - the NA which was used in the calculation;
- Calculation version - version of the form.

Properties for a link with Scia Engineer:

- Code - the code (standard) used for the calculation form;
- Applicable member - member type which is supported by this calculation - 1D/2D member;
- Applicable material - material type which is supported by this calculation - steel/concrete, etc.;
- Applicable cross section - section which is supported by this calculation.

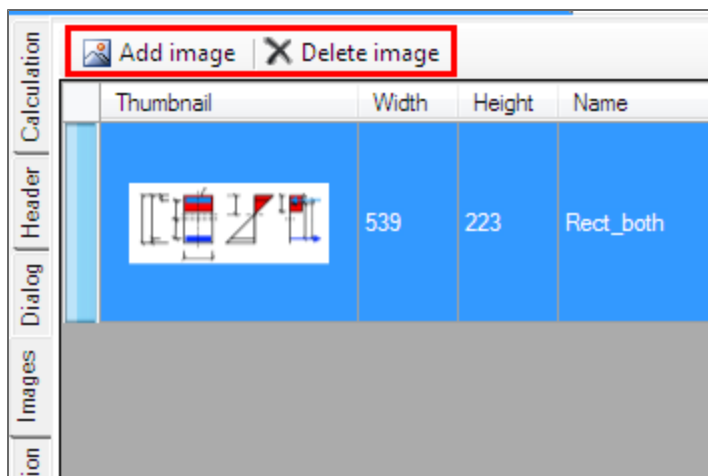
Calculation	Form name	Calculation
	GUID	73e76244-09b9-4293-9c
Header	Author	User
	Form version	-
Dialog	Norm code	Undefined
	Applicable member	Undefined
Images	Applicable material	Undefined
	Applicable cross-section	Undefined
Documentation	Result settings	
	ESA_ID	Name

Images

All images used in the calculation are saved in the calculation file (*.cls4).

The user has to import the image to the calculation library before it can be used.

Images are inserted in the layout by the command `IMG("image_name")` in the code; the `<image_name>` must match the name defined on the tab 'Image'.



Loading images to the calculation

- Use the button "Add image" on the 'Image' tab in the BUILDER application;
- Select image(s) which should be loaded to the form and confirm.

Deleting images from the calculation

- Select image(s) on the 'Image' tab that should be deleted;
- Use the "Delete image" button.

Renaming an image

- Define a new name in the column "Name".

Form Annotation

Inserting the Form Annotation link

The Form Annotation can be a standard web page in HTML format.

Write the URL address for the Form Annotation in the "Form Annotation" tab header.

Each language can have its URL address of Form Annotation.

Translations

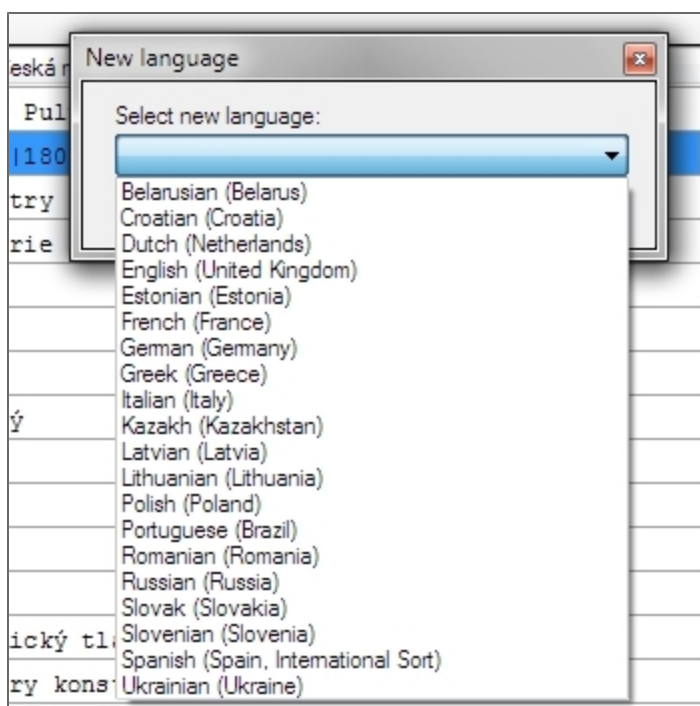
The same calculation form may be provided in two or more languages without having to change the code or calculation layout. Scia Design Forms has been developed keeping the need for multiple languages in mind. A built-in functionality is provided in order to assist a form developer in translating forms easily and fast.

Translations are handled on the 'Translations' tab of the BUILDER application. Automatically, all items that can be translated in the form are listed on the 'Translations' tab - contents of TEXT commands, names of layouts, variable descriptions, dialogue components, etc. A form developer can add languages by adding columns to the right, and type the translations in the provided boxes.

A form has a default language, which is defined by right-clicking anywhere on the 'Translations' tab. The user can select which items should be translated and which not. If a translation cell is left empty, the application will always use the default language instead. This is useful in the case an item is created by the TEXT command, but does not contain any language dependent parts, e.g. `TEXT("A = " & A & ">0);`.

Add a new language to the form

- Display the context menu and select the item "Add new language";
- Select the required language from the provided list and confirm with 'OK;'
- Languages can be renamed and deleted in the same way.



Remark: The list of languages is loaded from the Windows system and is filtered to the list Scia Design Forms currently supports.

Exporting and importing translations

Two additional buttons are provided on the tab 'Languages:'

- The Export button exports the whole Translations table to an XLS file where the user can type translations;
- The Import function reads the content of an XLS file back to the Languages tab.

Calculation		
ID	English (United States)	čeština (Česká republika)
VARIABLE_e _{css}	Ultimate concrete strain	Mezní poměrné přetvoření betonu
VARIABLE_f _{cm}	The flexural tensile strength of the concrete	Pevnost betonu v tahu za ohybu
VARIABLE_M _{es}	The acting bending moment	Působící ohybový moment
LAYOUT_1	Standard	Standard
VARIABLE_c	Reinforcement cover	Krytí výztuže
VARIABLE_PrintC	Print the headline of the design	Tisknout nadpis posudku
VARIABLE_Y _s	Partial safety factor for reinforcement	Parciální součinitel bezpečnosti pro výztuž
VARIABLE_Y _c	Partial safety factor for concrete	Parciální součinitel bezpečnosti pro beton
VARIABLE_f _{yk}	Characteristic tensile strength of reinforcement	Charakteristická pevnost výztuže v tahu
VARIABLE_f _{ck}	Characteristic compressive strength of concrete	Charakteristická pevnost betonu v tlaku

	A	B	C	D
ID	English (United States)	ENU	čeština (Česká republika)	
CALC_NAME	Calculation of crack width on rectangle CSS		Výpočet šířky trhliny	
LAYOUT_0	Full		Plný	
LAYOUT_1	Standard			
LAYOUT_2	Brief		Stručný	
LAYOUT_3	Layout 3			
LAYOUT_4	Layout 4			
LAYOUT_5	Layout 5			
VARIABLE_φ _p	The compensatory diameter for prestressing steel		Náhradní průměr předpínací výztuže	
VARIABLE_A _{p1}	Cross-sectional area of the tendon which can be calcu		Započítatelná plocha předpínací výztuže	
VARIABLE_E _{cm}	Secant modulus of elasticity of concrete		Sečnový modul pružnosti betonu	
VARIABLE_E _s	Modulus of elasticity of reinforcement		Návrhová hodnota modulu pružnosti betonářské výztuže	
VARIABLE_A _{ceff}	Effective area of tensile concrete		Účinná plocha taženého betonu	
VARIABLE_k ₁	Coefficient of bonded reinforcement		Součinitel soudržnosti výztuže	
VARIABLE_k ₂	Coefficient of strain distribution		Součinitel typu namáhání	
VARIABLE_k ₃	Coefficient		Součinitel	
VARIABLE_k ₄	Coefficient		Součinitel	
VARIABLE_b	Cross section width		Šířka průřezu	
VARIABLE_h	Cross section height		Výška průřezu	
VARIABLE_ξ	Ratio of bond strength of prestressing steel		Poměr pevnosti v soudržnosti předpínací a betonářské výztuže	
VARIABLE_c ₁	Reinforcement cover		Krytí tažené výztuže	
VARIABLE_n ₁	Number of reinforcement bars		Počet prutů tažené výztuže	
VARIABLE_φ _{s1}	Diameter of tension reinforcement bar		Průměr prutu tažené výztuže	
VARIABLE_c ₂	Cover of compression reinforcement		Krytí tlačené výztuže	
VARIABLE_n ₂	Number of compression reinforcement bars		Počet prutů tlačené výztuže	

Translation items

CALC_xxxx

Contains properties of the calculation:

- CALC_NAME- calculation name. It is used in the main menu of the SDF User application

DIALOG_xxxxxx

Additional texts in the Dialogue (it is not the variable description). For more additional texts [see chapter "Calculation dialogue"](#).

Form Annotation

URL address with the Form Annotation.

Each language can have a different URL address = different Form Annotation.

LAYOUT_xx

Layout names.

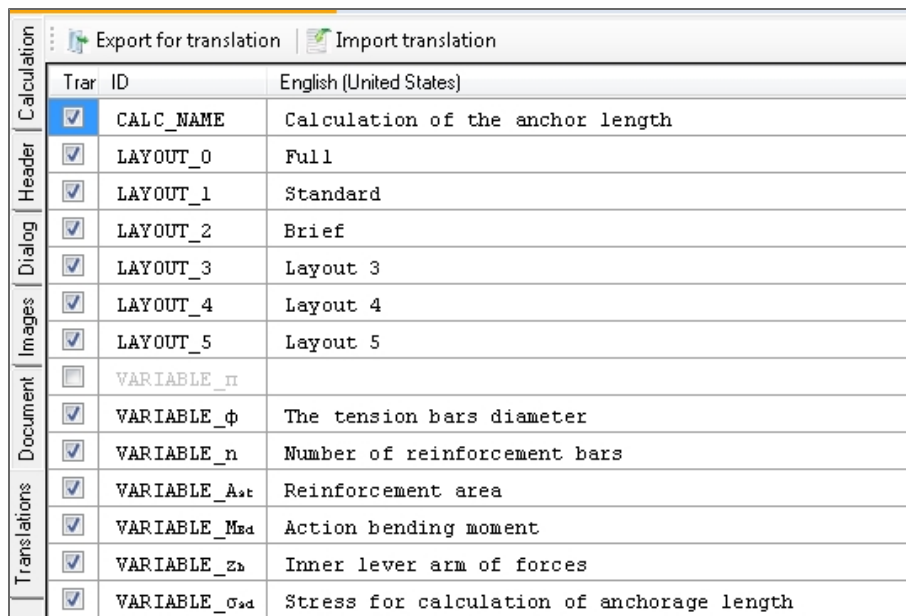
TEXT_XXXXXX

Text defined by the TEXT command.

VARIABLE_XXX

Variable descriptions from the Calculation Dialogue. For more about variable descriptions [see chapter "Table with variables"](#).

Lock the cell for translation



Tran ID	English (United States)
<input checked="" type="checkbox"/> CALC_NAME	Calculation of the anchor length
<input checked="" type="checkbox"/> LAYOUT_0	Full
<input checked="" type="checkbox"/> LAYOUT_1	Standard
<input checked="" type="checkbox"/> LAYOUT_2	Brief
<input checked="" type="checkbox"/> LAYOUT_3	Layout 3
<input checked="" type="checkbox"/> LAYOUT_4	Layout 4
<input checked="" type="checkbox"/> LAYOUT_5	Layout 5
<input type="checkbox"/> VARIABLE_π	
<input checked="" type="checkbox"/> VARIABLE_φ	The tension bars diameter
<input checked="" type="checkbox"/> VARIABLE_n	Number of reinforcement bars
<input checked="" type="checkbox"/> VARIABLE_A _s t	Reinforcement area
<input checked="" type="checkbox"/> VARIABLE_M _{Ed}	Action bending moment
<input checked="" type="checkbox"/> VARIABLE_z _b	Inner lever arm of forces
<input checked="" type="checkbox"/> VARIABLE_σ _s d	Stress for calculation of anchorage length

Since version 4.1, the Translations table contains a column entitled "Translate." This column contains check-boxes that define whether a string will be translated or not.

If the check-box is checked:

- The item has a "TranslateID" in CLS => it is included in the Scia Design Forms automatic translation;
- The item is exported to XLSX when the export for translation function is called;
- Translations defined in the table are active - changes in the string are visible in the source code and in the layout, when the language is changed.

If the check-box is unchecked:

- The item does not have a "TranslateID" in CLS => it is not included in the Scia Design Forms automatic translation;
- The item is not exported to XLSX;
- Translations defined in the table are inactive - these are not visible in the source code or in the layout, when the language is changed. The text as defined in the column of the default language will be used instead.

If an item had some translations before the check-box was unchecked, the translations are still saved in the *.CLS file - therefore, the translation can be inactivated without losing data.

The translation cannot be unchecked for:

1. CALC_NAME - form name
2. LAYOUT_# - layout name
3. DOCUMENT - html page for annotations

Those items are important for the translation and it cannot be unchecked for translations.

The translation can be deactivated for:

1. VARIABLE_### - variable description
2. DIALOG_##### - dialogue texts
3. TEXT_##### - strings from the command TEXT in source code

Translation of graphics and table parts

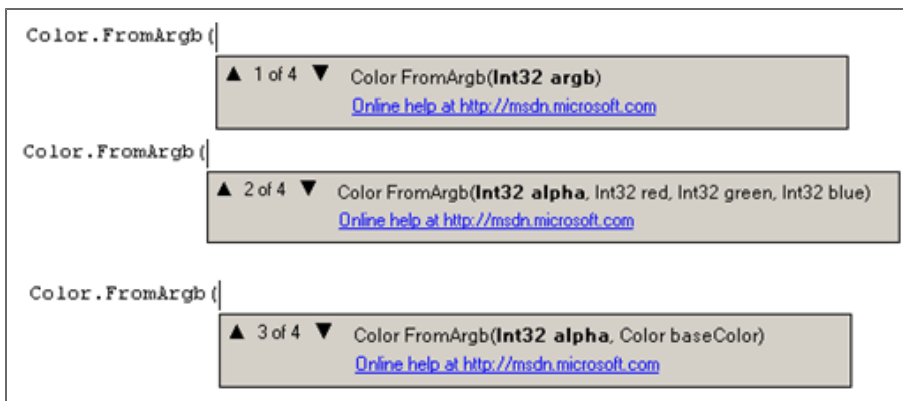
If the string from graphics should be added to the translation, use [command TR](#).

Commands reference guide

A significant part of the command list can be displayed in the SDF BUILDER by using the dot convention. To show the list, use the shortcut key CTRL+Spacebreak.



The help from XML file in the Builder application:



Comments

Syntax:

```
// adds a comment, starting at the // and ending at the end of the line; closing the comment is not required.
```

```
/* ... */ limits the comment between the indicated symbols /* and */.
```

Example 1

```
TEXT("An example of using comments in Scia Design Forms"); //this is a comment at the end of the line
```

Example 2

```
TEXT("begin");  
/*  
This is a comment over several lines. All commands within it are ignored, only the text "begin" and "end" are displayed.  
A = 10;  
TEXT("Some text");  
*/  
TEXT("end");
```

Example 3

```
TEXT("An example of a comment inside a command:");  
A = 10 + /*the text is skipped*/20;
```

[Load the example: REMARKS.cls](#)

Standard commands

Zde je seznam standardních příkazů v Scia Design Forms Builder.

BLOCK

Syntax:

```
BLOCK {  
  <code>  
}
```

or

```
BLOCK(<Block_Caption>){  
  <Block_Body>  
}
```

This command groups components into a block (paragraph). A block is handled as a single component in the layout - block contents can be moved together, and their position in the report may be set by a single setting. The location of each component within the block may be set in the usual way. A group created by the BLOCK command can be a default component.

The BLOCK command can be useful when the following components in the layout should be placed in relation to the previous components, but the size of previous components is unknown.

The block caption allows for collapsing parts of the layout to a single string (same as the <Block_Caption>). This syntax allows for more concise and manageable calculation reports.

Example

The user does not know which would be the last component of the "R" calculation. This is often the case when conditions are used in the code. If the user would like to place the following text string correctly, he should pack the "R" calculation into a block; then, the position of the following components would no longer be ambiguous and there would no longer be a danger of text overlaps or white spaces in the layout.

$$R = \frac{1}{\sqrt{a^2 - b^2}} = \frac{1}{\sqrt{5,00^2 - 2,00^2}} = 0,218 \quad \Rightarrow \quad R = 1,00$$

The default component of this text is a block.
This is the reason why the text is correctly placed.

```
BLOCK {  
  IF (a>b){  
    R = 1 / SQRT(a2 - b2);  
    IF (R<1) { TEXT("=>"); R = 1; }  
  } ELSE {  
    TEXT("The value must be bigger then b !");  
  }  
}  
TEXT("The default component of this text is a block.");  
TEXT("This is the reason why the text is correctly placed.");
```

The red part - the Block is used to encapsulate the code which may be a bit different according to the condition in IF. The blue part is Pagebreak which is always generated after the Block and that why it is always placed correctly.

```

1 BLOCK {
2   IF (fck < 50000000) {
3     TEXT ("Concrete strength fck < 50MPa =>")
4     η = 1;
5     λ = 0.8;
6   } ELSE {
7     TEXT ("Concrete strength fck > 50MPa =>")
8     η = 1 - [fck - 50] / 200;
9     λ = 0.8 - [fck - 50] / 400;
10  }
11 }
12 PAGEBREAK ();

```

1	2	3	4	5	6	7	8	9
Design of longitudinal reinforcement								
CSN EN 1992-1-1								
Concrete:								
f _{cd} = 53,3 MPa								
σ _{cc} = 1 η = 0,85								
ε _{cd} = 1,27 ‰ λ = 0,73								
Reinforcement:								
f _{yd} = 435 MPa z ₁ = 0,204 m								
σ _{yd} = 2,07 ‰ z ₂ = 0,211 m								
ε _u = 1000000%								

[Load the example: BLOCK.cls](#)

Block visibility

A block can be visible or hidden in the calculation layout.

Attention! All components in a hidden block are invisible, regardless if some of them are set to 'visible' individually. Block components could be visible only if the block is visible.

Collapsible block

Collapsible blocks are used for collapsing parts of the layout in the User application. Standard BLOCK component functionality is extended by the possibility of collapsing - by the part Block caption. The caption is displayed in the layout with the "+" (=display) or "-" (=collapse) symbol. The block without <Block_Caption> remains unchanged.

<Block_Caption> is automatically added into Translation tab.

Syntax:

```

BLOCK(<Block_Caption>){
  <Block_Body>
}

```

Example

The text Input values is visible in layout with "+" which is used to display or collapse the part which is defined inside the Block command.

```

BLOCK("Input values:") {
  TEXT("):");
  TEXT("Axial force"): TEXT("Nax = " & VAL(Nax/1000, 2) & " kN");
  TEXT("Bending moment (y)": TEXT("My, z = " & VAL(My, z/1000, 2) & " kNm");
  TEXT("Bending moment (z)": TEXT("My, z = " & VAL(My, z/1000, 2) & " kNm");
  TEXT("Buckling length (y)": TEXT("Ly = " & Ly & " m");
  TEXT("Buckling length (z)": TEXT("Lz = " & Lz & " m");
}
TEXT("Some text");

```

[Load the example: CollapsibleBlock.cls](#)

IF – ELSE

Syntax:

```
IF (<boolean_expression>){<code1> }
```

or

```
IF (<boolean_expression>){<code1> } ELSE {<code2> }
```

This command is used for defining conditions and branching the calculation.

If the condition (boolean expression) is met, the commands in the first part (<code1>) are executed. If the condition is not met, the second part (<code2>) is executed. The command ELSE is not required; if only the IF part is provided, after controlling the condition and executing the code if needed, the calculation continues after the IF command (see example 2)

Example 1

```

IF (R>0){
  V = (4 / 3) * π * R3;
  A = 4 * π * R2;
} ELSE {
  TEXT("The sphere radius must be larger then zero!");
}

```

[Load the example: IF.cls](#)

If the sphere radius (R) is larger then zero, the volume (V) and surface area (A) will be calculated. The relevant formulas are displayed in the layout. Otherwise (if the radius is equal to zero or has a negative value) the text will be displayed instead.

Example 2:

```

IF (α<=30) {μ1=0.8;}
IF (30<α && α<60) {μ1=0.8 * (60 - α) / 30;}
IF (α>=60) {μ1=0}

IF (v==3) {TEXT ("variable v is equal to 3");}

```

EXIT()**Syntax:**

```
EXIT();
```

- the EXIT command is used without any parameters in the brackets; it stops the execution of the code on the line where it is stated. The command is useful when the IF condition is used.

Example

```
IF (X==0) {EXIT();} – if the value of X equals 0, then the program will be stopped, no additional components will be generated in the layout.
```

[Load the example: Break, Continue, Exit.cls](#)

FOR**Syntax:**

```
FOR (<index>, <initial value>, <final_value>) {<code>}
```

FOR conditionally performs a command several times. The command FOR runs the <code> (see syntax), while the variable <index> changes value from <initial value> to <final_value> (including). Optionally, the <index> can be used as part of the command.

Example:

```

TEXT("Example of cycle FOR");
FOR(i, Start, End) {
TEXT("i = "&i);
}
TEXT("End of example");

```

- the code starts with initial value, the calculation is provided for each next value $i = \text{Start}$, $i = \text{Start} + 1$, $i = \text{Start} + 2$ etc. until i is bigger or equal to End.

[Load the example: FOR.cls](#)

Continue(), Break()

The commands are best implemented in cycles (WHILE, FOR).

- The command Continue() ends the current cycle step and starts the next step of the cycle.
- The command Break() ends the whole cycle.

When the commands Continue and Break are used in nested cycles, these commands always affect the innermost one (the last stated).

Syntax:

```
Continue();
```

```
Break();
```

The commands have no parameters.

[Load the example: Break, Continue, Exit.cls](#)

EXIT()

Syntax:

```
EXIT();
```

The command EXIT has no parameters (in the brackets); it stops the code execution on the line where it is stated. It is useful in the IF condition.

Example

```
IF (X==0) {EXIT();} – if the value of X equals 0, the program will be stopped, no additional components will be generated in the layout.
```

[Load the example: Break, Continue, Exit.cls](#)

WHILE

Syntax:

```
WHILE (<condition>) {  
  <code>  
}
```

The <code> is executed while the <condition> is fulfilled.

Example

```
x = 0;  
WHILE(x<5) {  
  TEXT("Example of a WHILE command.");  
  x = x + 1;  
}  
TEXT("End of WHILE cycle. x = "&x);
```

[Load the example: WHILE.cls](#)

The WHILE command performs the action <code> while the given condition is met. In this case, it writes the value of "x" in the appropriate loop. The command stops when x gets to a value of 5.

Continue(), Break()

The commands are best implemented for cycles (WHILE, FOR).

- The command Continue() ends the current cycle step and starts the next cycle step.
- The command Break() ends the whole cycle.

When the commands Continue and Break are used in two nested cycles, these affect always the innermost stated one (the last stated).

Syntax:

```
Continue();
```

```
Break();
```

The commands have no parameters.

[Load the example: Break, Continue, Exit.cls](#)

EXIT()

Syntax:

```
EXIT();
```

EXIT has no parameters (in the brackets); it stops the code execution on the line where it is stated. It is useful in the IF condition.

Example

```
IF (X==0){EXIT();} – if the value of X equals 0, then the code execution will be stopped, no additional components will be generated in the layout
```

[Load the example: Break, Continue, Exit.cls](#)

SWITCH - CASE

Syntax:

```
SWITCH (<variable>){  
CASE <value1> : { <code1> }  
CASE <value2> : { <code2> }  
...  
CASE <valueN> : { <codeN> }  
DEFAULT: { <code_default> }  
}
```

or

```
SWITCH (<variable>){  
CASE <value1> : { <code1> }  
CASE <value2> : { <code2> }  
...  
CASE <valueN> : { <codeN> }  
}
```

The command is meant for creating branches. If the value of <variable> is equal to <value_X>, the <code_X> will be executed. Then, the following step from SWITCH-CASE block will be checked.

If the value of <variable> is not equal to <value_X>, the <code_X> will not be executed. If the value of <variable> is not equal to neither of the listed <value1>, <value2>, ..., or <valueN>, the code after the keyword DEFAULT will be executed instead. If the branch DEFAULT is not defined, the code continues to the next command after the SWITCH block.

Example

```

TEXT("Example of SWITCH - CASE");
TEXT("The text changes according to the variable Index.");
SWITCH(Index){
CASE 0: { TEXT("The zero value is selected."); }
CASE 1: { TEXT("The first value is selected."); }
CASE 2: { TEXT("The second value is selected."); }
CASE 3: { TEXT("The third value is selected."); }
DEFAULT: { TEXT("Anything else is selected"); }
}

```

[Load the example: SwitchCase.cls](#)

Using SWITCH CASE for a COMBO-BOX

Combo-boxes in the dialogue may contain Numeric or String variables from the table of variables.

Numeric (Number)

In the case of a numeric variable, the function searched and identifies the index of the selected item only (indexes from 0), not the value. The combobox itself must be again defined in the Dialogue.

It is much shorter to defined the combo-box only by the Dialogue component.

```

SWITCH(<variable_name>){
CASE 0: { TEXT("item 1"); }
CASE 1: { TEXT("item 2"); }
CASE 2: { TEXT("item 3"); }
CASE 3: { TEXT("item 4"); }
}

```

String (Text)

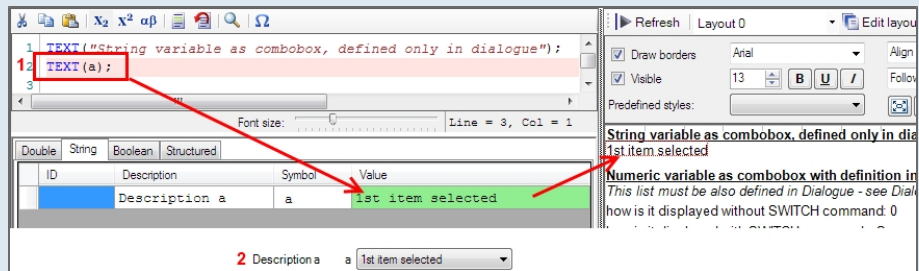
In the case of a string variable, the function considers the string from the combo-box list; quotation marks must be used. This combination of commands and dialogue items is slightly more complicated than the one with numeric variables, as the items often have to be translated. Working with numeric variables is recommended in this case, because then any possible usage is supported.

```

SWITCH(<variable_name>){
CASE "A": { TEXT("A"); }
CASE "B": { TEXT("B"); }
CASE "C": { TEXT("C"); }
}

```

This SWITCH block is not needed, the string combo-box returns its strings directly if : TEXT("<variable_name>"); is defined in the code; the displayed value depends on the item selected in the combo-box.



- 1 - Script in the source code that requests the display of the value of the variable, linked to the combo-box;
- 2 - In the dialogue - current selection in the combo-box.

[Load the example: combo.cls](#)

See more about combo-box [here](#).

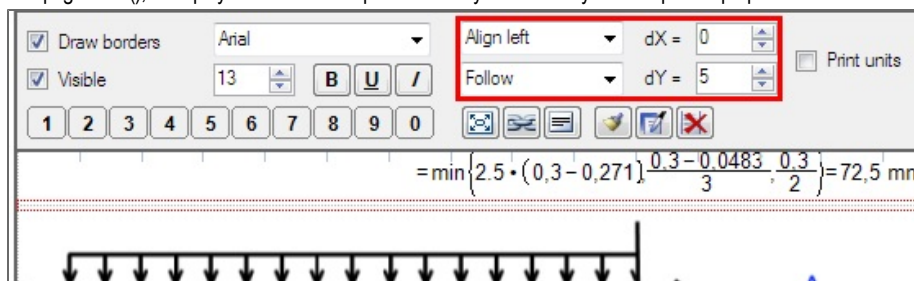
PAGEBREAK

Syntax:

PAGEBREAK();

This command cuts the report layout into pages. Every page can then be exported as a separate image/metafile, when the export functions of SDF are called.

The pagebreak(); is displayed as visual component - it may be moved by the component properties.



Example:

```
PAGEBREAK();
BLOCK {
  IMG("Rect_both");
  TEXT("Acc"); TEXT("As1"); TEXT("As2");
  TEXT("d1"); TEXT("<d1");
```

```
TEXT<("d2"); TEXT("b");  
TEXT("z1"); TEXT("z2");  
}  
PAGEBREAK();
```

This example displays an image and text which are separated from the rest of the layout (these will be shown on a separate page upon export).

When the PAGEBREAK is placed into the text because of format (e.g. shorter Brief layout where two columns of variables are created), in some cases user has to create a BLOCK around the whole part so the pagebreak won't split this group to two parts.

Text operations

TEXT(<text>)

Syntax:

```
TEXT(<text>);
```

The <text> argument may contain strings, equations or formulas. Several rules should be followed within the brackets of the command:

- "..." - prints a fixed string. The string between the quotation marks will be displayed as it is;
- '...' - prints a formula without values or substitution. The content between quotation marks will be displayed as a mathematical formula; variables will not be replaced by values;
- no quotation marks - the introduced code is a formula with values. The content will be converted to a mathematical formula using the up-to-date values of the referred variables;
- & - concatenates two parts of text/string/formulas together.

This command prints explanatory text, remarks, captions or formulas in the calculation output. The "&" symbol is used to concatenate parts of the TEXT argument, which are of different types (string, value, formula, ...).

Example

```
TEXT("This is simple text.");
```

```
TEXT("The following commands will not change the value of A. These are pure text outputs.");  
TEXT("This is a sample formula without substituting values: A = "&LOG(B) + C / D");  
TEXT("This is a sample formula with substituting values: A = "&LOG(B) + C / D);
```

```
TEXT("This equation changes the value of A.");  
A = LOG(B) + C / D;  
TEXT("Value of variable A = "&A);
```

This is a plain text

The next part wont change A, it is only text

Example of equation without substitution $A = \log(B) + \frac{C}{D}$

Example of equation with substitution $A = \log(100) + \frac{2,00}{3,00}$

The example change value A

$A = \log(B) + \frac{C}{D} = \log(100) + \frac{2,00}{3,00} = 2,67$

The value of variable A = 2,67

[Load example: TEXT.cls](#)

VAL(<expression>, <accuracy>)

Syntax:

VAL(<expression>, <accuracy>)

- it returns the value of <expression> with a given <accuracy>. The function displays the resulting value (usually a variable) in other than default units - see example.

Example:

L = 12,34 m
L = 123 dm
L = 1234 cm
L = 12340 mm
L = 12,34*10 ⁻³ km

```
TEXT("L = "&L&" m");
TEXT("L = "&VAL(10 * L,2)&" dm");
TEXT("L = "&VAL(100 * L,1)&" cm");
TEXT("L = "&VAL(1000 * L,0)&" mm");
TEXT("L = "&VAL(L / 1000,3)&" km");
```

[Load the example: VAL.cls](#)

Description (<variable>)

Syntax:

Description (<variable>);

- it returns the description of the variable <variable> from the table of variables. In this way, the variable description can be called from the table of variables; therefore, the description only needs to be typed once - it is not needed to retype it in a TEXT command. This is also useful for translations, as these can be done only once and be reused in the source code for every Description command.

The description of a variable is linked to the symbol of the variable (one row in the table of variables). The description can be simply typed text, or it may be also combined.
The description of any variable type may be used. The common usage is the repeating texts.

Remove this command from table for translations.

<input checked="" type="checkbox"/>	DOCUMENT	http://sciadesignforms.com/
<input type="checkbox"/>	TEXT_000002	Description("A")
<input checked="" type="checkbox"/>	VARIABLE_S1	

Example

```
TEXT("The first text comes from double variable A and it is loaded to string variable S1");  
string S1 = Description("A");
```

```
string S1 = Description("A");  
TEXT(Description("A"));
```

[Load the example: Description.cls](#)

Boolean operations

==, !=, >, >=, <, <=

Syntax:

A == B – is equal

A != B – is not equal

A > B – A is greater than B

A >= B – A is greater than or equal to B

A < B – A is lower than B

A <= B – A is lower than or equal to B

-this syntax is used for comparison of values.

Remark: „||“ – means OR; „&&“ – means and simultaneously

Example

```
IF (A==0) { TEXT("A is equal 0"); }
IF (!(A==0)) { TEXT("A is not equal 0"); }
IF (B!=0) { TEXT("B is not equal 0"); }
IF (C>=0) { TEXT("C is not negative"); }
```

All those conditions can be used together when using operators for combining conditions (see below):

```
IF (A>B && A>C) { TEXT("A is bigger"); }
IF (A<B || A<C) { TEXT("A is not the biggest"); }
IF (A<0 || B<0 || C<0) { TEXT("One of values is negative"); }
IF (A>0 && (B<0 || C<0)) { TEXT("A is positive, but B or C is negative"); }
```

```
IF ((A>0 && B>0 && C>0) || (A<0 && B<0 && C<0)) { TEXT("All numbers are positive, or all numbers are negative."); }
```

[Load the example: LOGIC.cls](#)

&&, ||, &|, !(...)

The operators are used to combine boolean expressions.

Syntax:

<boolean_expression> && <boolean_expression>- AND

<boolean_expression> || <boolean_expression>- OR

<boolean_expression> &| <boolean_expression>- XOR

!(<boolean_expression>)- NOT

True table for operation "&&":

A	B	Y=A&&B
0	0	0
0	1	0
1	0	0
1	1	1

True table for operation "||":

A	B	Y=A B
0	0	0
0	1	1
1	0	1
1	1	1

True table for operation "&|":

A	B	Y=A& B
0	0	0
0	1	1
1	0	1
1	1	0

True table for operation "!=":

A	Y=!A
0	1
1	0

Example

```
IF (A==0) { TEXT("A is equal 0"); }  
IF (!(A==0)) { TEXT("A is not equal 0"); }  
IF (B!=0) { TEXT("B is not equal 0"); }  
IF (C>=0) { TEXT("C in not negative"); }
```

```
IF (A>B && A>C) { TEXT("A is larger"); }  
IF (A<B || A<C) { TEXT("A is not the largest"); }  
IF (A<0 || B<0 || C<0) { TEXT("One of the values is negative"); }  
IF (A>0 && (B<0 || C<0)) { TEXT("A is positive, but B or C is negative"); }
```

```
IF ((A>0 && B>0 && C>0) || (A<0 && B<0 && C<0)) { TEXT("All numbers are positive, or all numbers are negative."); }
```

[Load the example: LOGIC.cls](#)

Mathematical operations - basic

Here is a list of basic mathematical operations in Scia Design Forms Builder.

Brackets

Syntax:

(...)

- brackets control the order of mathematical operations and parameters of functions; these are not displayed in the output;

[...]

- brackets control the order of mathematical operations; these are always displayed in the output.

Example

```
a = 1 + 2 3 - 4 = 1 + 2 • 3 - 4 = 3,00
b = 1 + 2 3 - 4 = 1 + 2 • 3 - 4 = 5,00
c = 1 + 2 3 - 4 = 1 + 2 • 3 - 4 = -1,00
d = 1 + 2 3 - 4 = 1 + 2 • 3 - 4 = -3,00
e = (1 + 2) 3 - 4 = (1 + 2) • 3 - 4 = 5,00
f = 1 + 2 (3 - 4) = 1 + 2 • (3 - 4) = -1,00
g = (1 + 2) (3 - 4) = (1 + 2) • (3 - 4) = -3,00
```

```
a = 1 + 2 * 3 - 4;
b = (1 + 2) * 3 - 4;
c = 1 + 2 * (3 - 4);
d = (1 + 2) * (3 - 4);
e = [1 + 2] * 3 - 4;
f = 1 + 2 * [3 - 4];
g = [1 + 2] * [3 - 4];
```

[Load the example: MATH.cls](#)

Number constants are always displayed with a dot instead of a comma as decimal separator in the layout.
3,1415 -> 3.1415

+, -, *, /

Syntax:

$$X = A + B;$$

$$X = A - B;$$

$$X = A * B;$$

$$X = A / B;$$

Standard mathematical operations

Example

$$a = 1 + 2 + 3 + 4 = 10,0$$

$$b = 3 - 4 = -1,00$$

$$c = 5 * 6 = 30,0$$

$$d = \frac{7}{8} = 0,875$$

$$e = (1 + 2) * (3 - 4) = -3,00$$

$$f = 2 * 3 - \frac{3}{2} = 4,50$$

$$g = \frac{2 + 5 - 1}{3} = 2,00$$

$$a = 1 + 2 + 3 + 4;$$

$$b = 3 - 4;$$

$$c = 5 * 6;$$

$$d = 7 / 8;$$

$$e = [1 + 2] * [3 - 4];$$

$$f = (2 * 3) - (3 / 2);$$

$$g = (2 + 5 - 1) / 3;$$

[Load the example: MATH.cls](#)

SUM (<variable>,<variable>, <variable>, ...)

Syntax:

```
Sum = Sum (<variable>, <variable>, <variable> ...);
```

- the command calculates the sum of all inserted variables and values.

Example

```
A = Sum (a, b, c, d);  
A = Sum (1, 2, 3, 4);  
A = Sum (1, 2, c, d);
```

[Load the example: Average_Sum.cls](#)

Exponent, POWER, SQRT

Exponent and POWER command (<basic>,<exponent>)

Syntax:

```
X = AB - A standard mathematical operation.
```

or

```
X = POWER(A, B)
```

- This calculates an exponent with a base equal to A, raised to the power B.

Example

```
Ea = xa = 01 = 0  
Pa = power(x, a) = power(0, 1) = 0  
Eb = xb+c = 02+4 = 0  
Pb = power(x, b+c) = power(0, 2+4) = 0  
Ec = xa +  $\frac{b}{c}$  = 01 +  $\frac{2}{4}$  = 0  
Pc = power(x, a +  $\frac{b}{c}$ ) = power(0, 1 +  $\frac{2}{4}$ ) = 0
```

```
Ea = xa;  
Pa = POWER(x, a);  
Eb = xb+c;  
Pb = POWER(x, b+c);
```



```
Ec = xa + b / c;  
Pc = POWER(x, a + b / c);
```

[Load the example: POWER.cls](#)

SQRT(<value>)

Syntax:

```
X = SQRT(<value>);
```

The command calculates the square root of the given <value>. For the calculation of the cube root (or n-th root) use the command POWER with corresponding arguments.

Example

$$\begin{aligned} A &= \sqrt{100} = \sqrt{100} = 10,0 \\ B &= 100^{\frac{1}{2}} = 100^{\frac{1}{2}} = 10,0 \\ C &= 1000^{\frac{1}{3}} = 1000^{\frac{1}{3}} = 10,00 \\ D &= 1000000^{\frac{1}{6}} = 1000000^{\frac{1}{6}} = 10,00 \end{aligned}$$

```
A = SQRT(100);  
B = POWER(100, 1 / 2);  
C = POWER(1000, 1 / 3);  
D = POWER(1000000, 1 / 6);
```

MIN, MAX, PARMIN, PARMAX

MIN(<value>, <value>, ... <value>) and PARMIN(<value>, <value>, ... <value>)

Syntax:

```
X = MIN(<value>, <value>, ... <value>);
```

- the command returns the minimal value among the listed arguments; the values are listed one after another (horizontally) in the layout.

```
X = PARMIN(<value>, <value>, ... <value>);
```

- the command returns the minimal value among the listed arguments; the values are listed one below another (vertically) in the layout.

Example

```
A = MIN(a/b,c+d,e*f);  
B = MAX(a/b,c+d,e*f);  
C = PARMIN(a/b,c+d,e*f);  
D = PARMAX(a/b,c+d,e*f);
```

A = MIN (125, a, b, c); - the minimum is selected from the value and variables

The screenshot shows a code editor on the left with four lines of code: 1. A = MIN(a/b, c+d, e*f); 2. B = MAX(a/b, c+d, e*f); 3. C = PARMIN(a/b, c+d, e*f); 4. D = PARMAX(a/b, c+d, e*f);. To the right is a control panel with 'Draw borders' and 'Visible' checked, and a 'Predefined styles' dropdown. Below the control panel, the execution results are displayed: A = min(a/b, c+d, e*f) = min(1/2, 3+4, 5*6) = 0.5; B = max(a/b, c+d, e*f) = max(1/2, 3+4, 5*6) = 30; C = Min(a/b, c+d, e*f) = Min(1/2, 3+4, 5*6) = 0.5; D = Max(a/b, c+d, e*f) = Max(1/2, 3+4, 5*6) = 30.

[Load the example: MIN_MAX.cls](#)

MAX(<value>,<value>, ... <value>) a PARMAX(<value>,<value>, ... <value>)

Syntax:

```
X = MAX(<value>,<value>, ... <value>);
```

- the command returns the maximal value among the listed arguments; the values are listed one after another (horizontally) in the layout.

```
X = PARMAX(<value>,<value>, ... <value>); A standard mathematical operation
```

- the command returns the minimal value among the listed arguments; the values are listed one below another (vertically) in the layout.

Example

```
A = MIN(a/b,c+d,e*f);  
B = MAX(a/b,c+d,e*f);  
C = PARMIN(a/b,c+d,e*f);  
D = PARMAX(a/b,c+d,e*f);
```

```
A = MAX (125, a, b, c); - the maximum is selected from the value and variables
```

[Load the example: MIN_MAX.cls](#)

Trigonometric functions

SIN(<angle_in_degrees>)

COS(<angle_in_degrees>)

TG(<angle_in_degrees>)

COTG(<angle_in_degrees>)

ARCSIN(<ratio>) (from -1 to +1)

ARCCOS(<ratio>) (from -1 to +1)

ARCTG(<ratio>) (from $-\infty$ to $+\infty$)

ARCTG(<ratio>) (<value1>, <value2>)

ARCCOTG(<ratio>) (from $-\infty$ to $+\infty$)

Syntax:

```
X = SIN(<angle_in_degrees>);
```

- the function argument must be introduced in degrees (angle).

```
X = ARCTG(<value1>, <value2>);
```

- ARCTG may be used with two arguments; in this case, it returns the arctan of the proportion $\frac{\text{<value2>}}{\text{<value1>}}$.

Example

```
A = SIN( $\alpha$ );  
B = COS( $\alpha$ );  
C = TG( $\alpha$ );  
D = COTG( $\alpha$ );
```

$$A = \sin(\alpha) = \sin(30,0) = 0,50$$

$$B = \cos(\alpha) = \cos(30,0) = 0,866$$

$$C = \operatorname{tg}(\alpha) = \operatorname{tg}(30,0) = 0,577$$

$$D = \operatorname{cotg}(\alpha) = \operatorname{cotg}(30,0) = 1,73$$

The function ARCSIN may result in NotANumber (NaN) when the input is not an exact number, e.g. (1,00003 > 1.0). The application automatically rounds off numbers to 10 decimal places; therefore, when the number is exact after being rounded off, the result will be calculated correctly.

The picture shows an example when the input value is not rounded off and it is in fact -1,0000000012 (<1.0).

[Load the example: GONIOMETRIC.cls](#)

ABS<value>

Syntax:

```
X = ABS(<value>);
```

- the function calculates the absolute value of the defined argument.

Example

```
A = ABS(123,456); // returns 123,456
A = ABS(0); // returns 0
A = ABS(-123,456); // returns 123,456
```

ROUND(<value>, <number_of_dec_places>)

Syntax:

```
X = Round(<value>, <number_of_dec_places>);
```

- the function rounds off the given value to the given number of decimal places.

Example

```
A = Round(123.456, 0); // returns 123.000
A = Round(123.456, 2); // returns 123.450
```

```
A = Round (123.456, 5); // returns 123.45600  
A= Round (B,2); // returns variable B rounded off to 2 decimal places
```

[Load the example: ROUND.cls](#)

AVERAGE (<variable>,<variable>, <variable>, ...)

Syntax:

```
Avg = Average (<variable1>, <variable2>, <variable3> ...);
```

- the function calculates the average from the defined arguments - values or variables.

Example

```
A = Average (a, b, c, d);  
A = Average (1, 2, 3, 4);  
A = Average (1, 2, c, d);
```

[Load example: Average_Sum.cls](#)

DIV, MOD

DIV(<value>)

Syntax:

```
X = DIV(<value>, <value>);
```

- the function returns the integer result of division of the two arguments.

Example

```
A = DIV(10, 3); // returns 3,0  
A = DIV(16, 4); // returns 4,0  
A = DIV(27, 5); // returns 5,0
```

MOD(<value>)

Syntax:

```
X = MOD(<value>, <value>);
```

- the function returns the remainder of division of the two arguments.

Example

```
A = MOD(10, 3); // returns 1,0  
A = MOD(16, 4); // returns 0,0  
A = MOD(27, 5); // returns 2,0
```

Mathematical operations - advanced

Here is a list of advanced mathematical operations in Scia Design Forms Builder.

LOG, LN

LOG(<value>, <base>)

Syntax:

```
X = LOG(<value>);  
X = LOG(<value>, <base>);
```

When the function is used with a single argument (first syntax), it returns the result of a logarithmic function with a base 10. If two arguments are defined, then the logarithmic function is calculated for a base equal to the second argument <base>.

Example

```
A = LOG(100); // returns 2,0  
A = LOG(1); // returns 0,0  
A = LOG(0.001); // returns -3,0  
A = LOG(25, 5); // returns 2,0  
A = LOG(1, 5); // returns 0,0  
A = LOG(0.25, 2); // returns -2,0
```

LN(<value>)

Syntax:

```
X = LN(<value>);
```

- the function returns the natural algorithm (base = e = 2,71828).

Example

```
A = LN(7.389056); // returns 2,0  
A = LN(1); // returns 0,0
```

COMPARE

Syntax:

```
X = COMPARE(<object1>, <object2>); The function compares the defined objects.  
X = COMPARE(<number1>, <number2>, <precision> ); The function compares the defined numbers.
```

The results from the COMPARE command:

- 0 - the objects / numbers are equal;
- 1 - the first object is larger than the second;
- -1 - the first object is smaller than the second.

Replace IF conditions with COMPARE when two doubles need to be compared.

If values are compared using the IF command, it is not possible to define the precision of the comparison (is 0.0001 equal to or larger than 0?). This is not like that for the COMPARE command. Using the IF command is still acceptable when you are dealing with continuous functions, but pay attention to with discontinuous functions.

Tolerance

If two numbers are compared using a defined precision, the function returns 0 (numbers are equal) if:

$$\text{Max}(\text{number1}, \text{number2}) / \text{Min}(\text{number1}, \text{number2}) < 1 + 10^{-\text{precision}}$$

i.e. the ratio of the two numbers is not greater than 1, increased by ten to -(the defined precision).

The precision must be defined as a positive non-zero number, the bigger the number the higher the precision.

Example

```
X = COMPARE("A", "B"); // returns 0  
X = COMPARE("A", "A"); // returns 1  
X = COMPARE("A", "a"); // returns 0  
X = COMPARE(true, true); // returns 1  
X = COMPARE(true, false); // returns 0  
X = COMPARE(false, false); // returns 1  
X = COMPARE(0, 1); // returns 0  
X = COMPARE(1, 1); // returns 1  
X = COMPARE(2, 1); // returns 0  
X = COMPARE(1.0005, 1, 3); // returns 0  
X = COMPARE(1.0005, 1, 4); // returns 1
```

Precision equal to 3 means that the relative ratio of two numbers is less than $10e-3$.

[Load the example: COMPARE.CLS](#)

The example of using command Compare to prevent the division by zero value is in [Coding rules here](#).

LININTERP

Syntax:

```
doubleD = LinInterp (doubleX, struct Point1, struct Point2 [struct Point3...]);
```

- it returns a linear interpolation for value X (on the x-axis) between points Point 1, Point 2, etc.

If the value X is not between Point 1 and Point N, the results is NaN (Not A Number).

Points 1 to N must be sorted in ascending order for their X coordinates.

Example:

```
X = LinInterp(-1, point(0, 0), point(1, 10));
X = LinInterp(0.2, point(0, 0), point(1, 10));
X = LinInterp(0.6, point(0, 0), point(1, 10));
X = LinInterp(2, point(0, 0), point(1, 10));
X = LinInterp(0.8, point(0, 0), point(1, 10), point(2, 200));
X = LinInterp(1.5, point(0, 0), point(1, 10), point(2, 200));
```

```
X = LinInterp (- 1,Point{0,0},Point{1,10})= NaN
X = LinInterp (0.2,Point{0,0},Point{1,10})= 2
X = LinInterp (0.6,Point{0,0},Point{1,10})= 6
X = LinInterp (2,Point{0,0},Point{1,10})= NaN

X = LinInterp (0.8,Point{0,0},Point{1,10},Point{2,200})= 8
X = LinInterp (1.5,Point{0,0},Point{1,10},Point{2,200})= 105
```

[Load the example: LININTERP.CLS](#)

CONTAINS

Syntax:

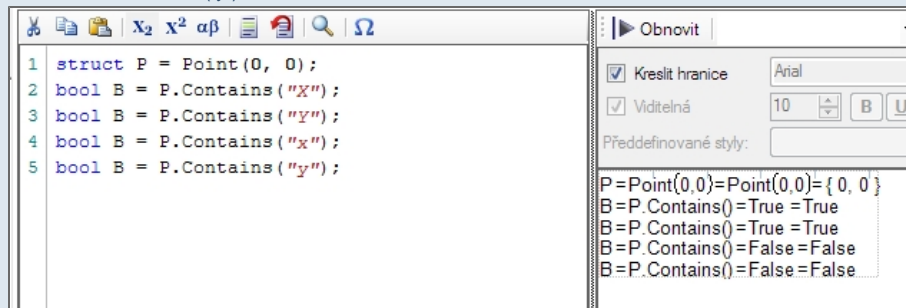
```
bool B = CONTAINS(stringChild);
```

CONTAINS(stringChild) is a method of addressing a structured variable. It can be used to verify whether a structured variable contains the given variable (Child). This sub-variable can be declared only as text. If, for example, a structured variable is of the Force type, it contains six sub-variables - N, Vy, Vz, Mx, My, Mz - such a variable may be asked whether it contains My and it should return TRUE.

The function CONTAINS(string <text>) returns TRUE if the variable contains the sub-variable of the given name <text>. Otherwise, it returns FALSE.

Example:

```
struct P = Point(0, 0); - a structured variable contains a point with coordinates X and Y; therefore, the function CONTAINS finds X, but not x  
bool B = P.CONTAINS("X");  
bool B = P.CONTAINS("Y");  
bool B = P.CONTAINS("x");  
bool B = P.CONTAINS("y");
```



RANDOM()

The Random class is a generator of pseudo-random numbers. It generates a sequence of numbers that meet certain static requirements for randomness.

Methods:

- Next() returns a nonnegative random number.
- Next(Int32) returns a nonnegative random number, smaller than the specified maximum.
- Next(Int32, Int32) returns a random number within a specified range.
- NextDouble() returns a random number between 0.0 and 1.0.

[More information can be found here \(MSDN\).](#)

Tools in Dialogue

Here is a list of commands used in the Dialogue in Scia Design Forms Builder.

Hide dialogue components by using script

Syntax:

```
Dialog.GetComponentByName("<name of component>").Visible = false;
```

- this command sets the component <name of component> as hidden from within the source code.

The name of a component can be found in its properties window in the dialogue.



Example

```
IF(visible) { Dialog.GetComponentByName("Panel").Visible = true;
} ELSE { Dialog.GetComponentByName("Panel").Visible = false; }
- if the condition "visible" is true -> the dialogue component named "Panel" is visible, otherwise not
```

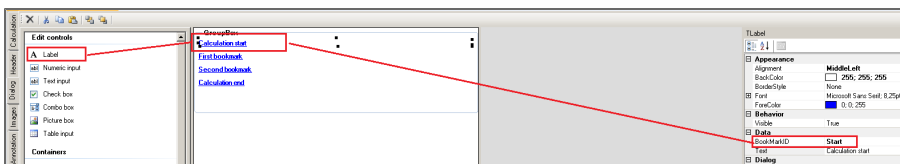
NOTE: Put all components which should be hidden in one panel. Switch off visibility for this panel.

Bookmark

Bookmarks command is used for the fast scrolling in the layout in the User application. This command is displayed as a shortcut (blue label as default) in the Dialogue which moves the layout to the predefined position (Bookmark command in the source code). The bookmark component is displayed with the blue colour in the Dialogue.

Definition in the Builder:

1. Define the bookmark in the source code (see the part Syntax).
2. Add the component Edit control type "Label" to the Dialogue.
3. Fill the property BookMarkID by the string from the source code.
4. Define the text in the properties - the text will be displayed in the Dialogue.



The bookmark must be placed carefully in the source code, so it will always lead to the position which is displayed. It is possible to combine bookmarks and visible part of the dialogue and layout.

Syntax:

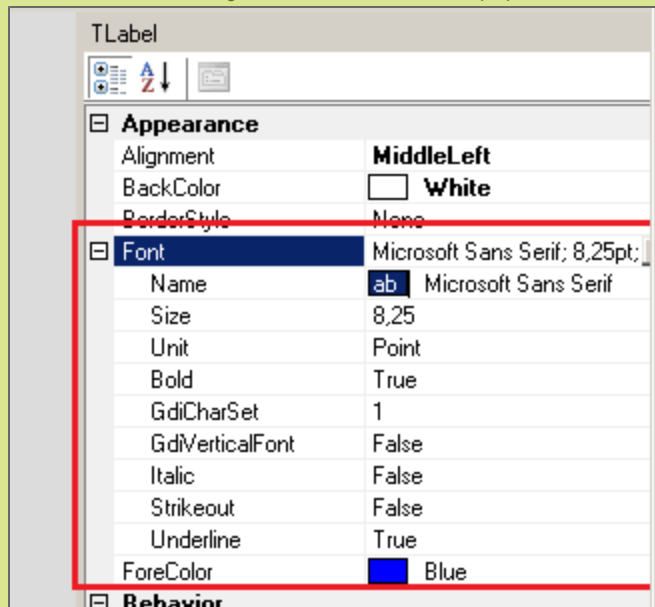
```
Bookmark("Bookmark_ID");
```

-Bookmark_ID can be any string but just one word (without spaces).

Example

```
12 BOOKMARK("First");
```

The blue colour in the Dialogue is editable in the bookmark properties.



[Load the example: Bookmark.cls](#) and [Bookmark.CLC](#)

Calculation and/or input from external files

LoadExternCLC

Syntax:

```
<variable>=LoadExternCLC("file_path/file_name.clc");
```

The command LoadExternCLC will load a new component to the layout (if the following command is DRAW) or calculate (if the following command is CALCULATE). The component should be defined by code in a previously prepared *.CLC file. Both files are linked and changes in the loaded file are visible in the current layout after refresh. External files may be displayed in the layout or these can be calculated in the background (see chapters DRAW and CALCULATE).

External files are always loaded to a object variable. If more external *.CLC files are loaded to the current *.CLS file, it is necessary to use different names for the variables in order to avoid conflicts in the loaded data.

If the current *.CLS and external *.CLC files are saved in the same folder, then the name of the *.CLC is sufficient to call it. If the CLC file is stored somewhere else, a full path is required.

An external *.CLC file is loaded to a variable of type **object**.

The next command defines what should be done with the external file; it is not possible to make use of the LoadExternCLC command without the definition of either the DRAW or CALCULATE command. See chapters DRAW and CALCULATE.

Example

```
LoadExternCLC("Extern.clc");
TEXT ("Load extern:");
ExternM = LoadExternCLC ("ExternM.CLC");
ExternM.DRAW (true);
```

[Load the example: EXTERN.zip](#)

Save these files to the same folder and open them in Scia Design Forms. The file Extern.CLS contains code for loading the files ExternN.CLC and ExternM.CLC; changes in ExternM.clc and ExternN.clc will become visible when Extern.CLS is compiled.

ExternA.CLS contains ExternB.CLS.

CALCULATE

Syntax:

```
<ExternCLC>.Calculate();  
<ExternCLC>.Calculate(bool TransferVariables);
```

- the function runs the code in an external CLC without displaying results in the layout.

(bool TransferVariables)

The boolean variable TransferVariables defines whether values should also be transferred between the current and external *.CLC files. A TRUE value of the boolean variable means that the values from the external CLC will be loaded. FALSE means that the current values, if defined, will be used.

```
extern2.Calculate(true);  
TEXT(" Value x from extern2 :");  
ext1 := extern2.x;
```

If the boolean variable TransferVariables is not set then it is automatically set to False.

[Load the example: EXTERN.zip](#)

Problems may occur when a variable has the same name in the current and external CLC; if a variable is transferred from one file to another when the boolean parameter is set to True, the variable in the current CLC will be overwritten.

DRAW

Syntax:

```
< ExternCLC>.Draw();  
< ExternCLC>.Draw(bool TransferVariables);  
< ExternCLC>.Draw(bool TransferVariables, int LayoutIndex);
```

- the command displays the external *.CLC file in the layout of the current form.

(bool TransferVariables)

The boolean variable TransferVariables defines whether values should also be transferred between the current and external *.CLC files. A TRUE value of the boolean variable means that the values from the external *.CLC will be loaded. FALSE means that the current values, if defined, will be used.

(int LayoutIndex)

The variable LayoutIndex defines which layout defined in the external *.CLC file will be displayed in the current layout. If the variable is not set, the layout of the current calculation will be used. Values that are allowed for this argument are the layout names in the

external *.CLC file, or the consecutive number of a layout.

```
TEXT("Load 2nd layout from ExternM:");
extern := LoadExternCLC("ExternM.clc");
extern.Draw(True,2);
```

The first argument takes up a boolean value - TRUE, and the second argument is the integer 2. It means that values WILL BE loaded to the external *.CLC file and the second layout will be displayed in the layout of the current file.

[Load the example: EXTERN.zip](#)

DRAW(True), DRAW(False), DRAW()

Draw(True) - Variables with the same name (symbol) are synchronized in current and external calculations. The current calculation displays the component of the external calculation; it uses values from the current CLC and rewrites the values which come from the external *.CLC file.

ATTENTION!. The current *.CLC could thus have values of variables, used in the calculation, that are different from the values defined and displayed in it.

Draw (False) - variables from the current file are not transferred to the external file and current values are not changed.

Draw() - the default value for the function argument is set to False, variables are not transferred.

Access to variables

Variables can be referred to only if the external CLC has been loaded.

The dot convention is used to access the external variables.

```
TEXT("Extern2 will send value x if A>4");
extern2 := LoadExternCLC("extern2.clc");
extern2.A := 5;
extern2.Calculate(true);
TEXT(" Value x from extern2 :");
ext1 := extern2.x;
```

Please, use the correct format for SDF version 4 - the command is extern2= LoadExternCLC ("extern2.cls);

Example

$M_{Rd,y} = \text{ExternCLC.MRd,y}$; - the value $M_{Rd,y}$ is loaded from the external *.CLC file.
 extern2.A = 5; - the value 5 is set to the variable A of extern2.cls.

ExternCLC without predefined variables (type object)

The external CLC can be displayed or calculated without using the variable type object.

Syntax:

```
LoadExternCLC("<path to the extern file>").Draw(<parameters>);
```

- load and display external CLC, or

```
LoadExternCLC("<path to the extern file>").Calculate(<parameters>);
```

- load and calculate external CLC.

When the shorter syntax is used (shown in this paragraph) the link to the external CLC is not saved to a variable. The user cannot access the variables which are calculated in the external file.

Reading values from external XML files (CustomDataTable)

Values from an *.XML file, which are often used by [Custom libraries in the Dialogue](#), can be loaded without inserting the Custom library to the Dialogue. Values are saved in the CLS in an object variable.

*.XML files must be saved in C:\Users\Public\Documents\DesignForms_version\CustomLibrary\... It is possible to save the

*.XMLs to a subfolder and use the subfolder in the path to the file in the source code.

*.XML files could also be created in the [Custom library editor](#), and these will be readable by SDF BUILDER application. It is possible to create any user-defined database and use the values in a calculation.

Values from external XML files are loaded without any conversion (as text), this saves time during the loading process. If data should be converted, then use this command: `new CustomDataTable(<file_name>.xml>).Convert();` This method, however, is quite time-consuming.

Syntax:

```
object <variable_contains_XML> = new CustomDataTable("<file_name>.xml");
```

- the whole table from the *.XML file is loaded to the new variable of the type 'object'.

```
object <variable_name> = <variable_contains_XML>.Rows[i].ItemArray[y];
```

- the value from the i^{th} row and y^{th} column is loaded to the variable <variable_name>, it is loaded from the object with XML data (created by the previous syntax); the value is saved as text.

Example

```
object DT = new CustomDataTable("EC_1992-1-1_Table_4.4N.xml");
object X = DT.Rows[0].ItemArray[5];
```

- the whole XML is loaded to the object variable DT as text;
- variable X takes up the text value from DT in row zero and column 5.

[Load the example: CustomDataTable_simple.zip](#)

[Load the example: CustomDataTable.zip](#)

More information on the web can be found [here](#).

CONVERT()**Syntax:**

```
object <variable_contains_converted_XML> = <variable_contains_XML>.Convert();
```

- the command converts all data from text to numbers; if the variable in *.XML has units, then the initial form or unit is used during the converting process (see the example below).

```
double <variable> = <variable_contains_converted_XML>.Rows[i].ItemArray[y];
```

- the value from row i and column y from the object with converted data from XML is saved to a new variable; the variable is saved as a number.

Example

If the *.XML file contains a variable in MPa and the user uses the command CONVERT(), the values is converted to Pa. Units are not automatically added.

e.g.:

```
object DT = new CustomDataTable("EC_1992-1-1_Table_4.4N.xml");
object CONVERTED = DT.Convert();
double D = Converted.Rows[0].ItemArray[5];
```

- The DT variable assumes the XML data without conversion (loaded as text); if e.g. 20 MPa is defined in the *.XML file, the value is displayed as 20 in the layout;
- The CONVERTED object assumes all data in DT, after these are converted to numbers; in the same example, the value in the CONVERTED variable will be $20 \cdot 10^6$.
- D is number taken from the CONVERTED variable from row 0 and column 5.

If a unit should be defined, it must be defined in the variable which loads the value from the main variable with XML data.

[Load a simple example: CustomDataTable_simple.zip](#)

GetStructure()

Syntax:

```
object <structured> = <variable_contains_XML>.GetStructure(<variable_contains_XML>.Rows[i]);
```

- it loads a row with values and save it to a structured variable <structured>.

```
double <variable> = <structured>.<column_name_in_XML>;
```

- it saves a single value from this row to the variable <variable>.

Example

```
object B = DT.GetStructure(DT.Rows[3]);  
double A = B.XC4;
```

- DT contains data from the *.XML file without conversion (stored as text);
- variable B is a converted structured variable from the 3rd row (contains all data from this row);
- variable A loads the value from the column named "XC4" in the *.XML file.

[Load a simple example: CustomDataTable_simple.zip](#)

Special EN functions for civil engineers

EuroCode.SteelSectionPressureClass(struct CS, double fy);

EuroCode.SteelSectionBendingClass(struct CS, double fy)

This class of functions are used to refer to commonly used properties defined in the Eurocodes.

Syntax:

```
double D = EuroCode.SteelSectionPressureClass(struct CS, double fy) - Returns the cross-section class according to EN 1993-1-1:2005, for pure compression. Arguments:
```

- CS - a structured variable which contains cross-section parameters (included in the steel cross-section library);
- fy - yield strength of the selected steel.

`double D = EuroCode.SteelSectionBendingClass(struct CS, double fy)` - Returns the cross-section class according to EN 1993-1-1:2005, for pure bending. Arguments:

- CS - a structured variable which contains cross-section parameters (included in the steel cross-section library);
- fy - yield strength of the selected steel.

Example

```
double SectionClass = EuroCode.SteelSectionBendingClass(IO.CS.Geometry, fy);
```

- the value calculated by the source code is loaded to the variable SectionClass - it contains a number defining the cross-section class (1-4);

- IO.CS.Geometry - the first argument defines which cross-section should be used; the cross-section is defined by selecting it in the steel cross-section library in the dialogue. The default library target node name is CS; other names can be defined for multiple section libraries in the same calculation form;

- fy - the second argument is loaded from variable fy

[Load the example: Check of bending with.CLS](#)

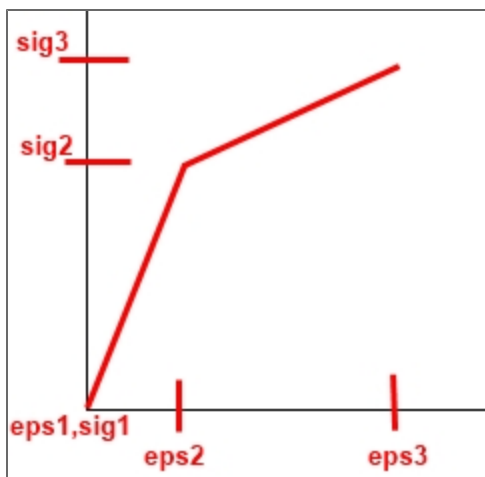
This example is also available in the setup of Scia Design Forms.

MaterialDiagram()

This command creates an object variable that describes the stress-strain curve of a material. The object variable contains points (also structured variables), which define the material constitutional law, if connected with lines.

Syntax:

```
object <variable_name> = new MaterialDiagram(<ε1>, <σ1>, <ε2>, <σ2>, ...);
```



Example

```
object MD1 = new MaterialDiagram(0, 0, 0.001, 20000000);  
- the new material diagram is loaded to the object MD1; it consists of two points: (0,0) and (0.001,20*106).
```

[Load the example: MaterialDiagram.CLS](#)

ReinfBar()

The function ReinfBar constructs a structured variable that requires 3 parameters. The first two parameters are the position of the reinforcement bar (coordinates); the third parameter is the bar diameter. Units are not defined, all values remain in the default Scia Design Forms units. This allows for the parameters to be properly referred to afterwards. Units may be defined later on.

Syntax:

```
ReinfBar(double X, double Y, double D);
```

- the command creates a new reinforcement bar at coordinates (X,Y) with D as diameter.

Example

```
ReinfBar(x1, y1, 16); - the position is defined by x1 a y1, the diameter is 16 (when the parameter is used in source code, the unit "mm" should be defined).
```

Stress-strain diagram functions

Here are some additional functions for creating stress-strain diagrams for concrete and reinforcement:

ConcreteDiagramULS()

Syntax:

```
object[] ConcreteDiagramULS = ConcreteULSdiagram(double  $\alpha_{cc}$ , double  $\gamma_c$ , bool Char);
```

α_{cc} - the coefficient takes into account long-term effects on the compression strength and negative effects resulting from the manner of load application;

γ_c - the coefficient corresponds to the partial safety factor for concrete strength;

Char – the boolean variable defines whether characteristics or design values will be used.

ConcreteDiagramSLS()

Syntax:

```
object[] ConcreteDiagramSLS = ConcreteSLSdiagram(double kfck, double φ, bool Tension)
```

kf_{ck} - the coefficient defines the maximal concrete strength;

φ - corresponds to the creep coefficient;

Tension – the boolean variable defines whether the tensile or compressive portion of the stress-strain diagram will be used.

ReinfDiagramULS()

Syntax:

```
object[] ReinfDiagramULS = ReinfULSdiagram(γs, double CoeffεSud)
```

γ_s - corresponds to the partial safety factor for the reinforcement material;

Coeffε_{Sud} – coefficient for the calculation of design stress in the reinforcement.

ReinfDiagramSLS()

Syntax:

```
object[] ReinfDiagramSLS = ReinfSLSdiagram(double kfyk)
```

kf_{yk} - coefficient of maximal strength limit for the reinforcement;

The rest of values are loaded from the material library.

Example

A function from an external file is loaded:

```
object Extern = LoadExternCLC("StressStrain_Diagram.clc");
object [] Diagram = Extern.ConcreteULSdiagram(αcc, γc, Char);
```

or use this command:

```
object [] Diagram = Extern.ConcreteULSdiagram(1,0, 1,5, False);
```

[Load example: StressStrain_Diagrams.CLS](#)

Enum

Enums:

-
- **ArrowShape** - defines settings for per drawn arrow. Used in Graphics (dimensions, leader, arrow, ...)
 - **ContentAlignment** - defines text alignment, alignment of cell content, etc. [See more here.](#)
 - **DashStyle** - defines the style of a dashed line. [see more here.](#)
 - **FontStyle** - defines font style (Bold/Underline/Italic). [see more here.](#)
 - **HatchStyle** - defines the style of HatchBrush. [see more here.](#)
 - **PointStyle** - defines point style in graphs drawn with the GRAPH command.

User class

Find more information on classes [here](#) on Wikipedia.

Syntax:

```
class <Class_Name> {  
  <Class_Properties>  
  <Class_Methods>  
}
```

- these commands describe the user-defined class in the source code.

Example

```
class Test {  
  double X;  
  double Y;
```

```
  Initialise:  
  object <Variable_Name> = new <Class_Name>();
```

[Load an example: Class.cls](#)

Usage:

A user class can be used to encapsulate a logic block of properties (values) and functions. The user may create multiple instances of one class type to make the script more clear, organised and using the form more efficient.

See also: [MSDN online help.](#)

User functions

Find more information about functions [here](#) on wikipedia.

User functions can be created in a user class or these may be loaded from an external form (see more about the LoadExternCLC command in this [separate chapter](#)).

User functions may be used (1) in equations or (2) to export data to the output.

Syntax:

```

<type> <function_name><parameters>{
<function_body>
return <value>;
}

```

- this script creates a new function in the source code.

A user-defined function is described using (see notation in 'sybtax'):

- **type** - type of the returned value: double, string, boolean, or object;
- **function_name** - name of the function; it may contain letters, numbers and underscore; the first character must be a letter;
- **parameters** - an array of parameters separated by commas; a parameter is defined by parameter type and parameter name;
- **functions_body** - standard SDF script;
- **return** - the output value is called by the command "return"; when a value is not defined, a "null" value is returned.

Example

```

double Test(double X){
TEXT("X = " & X);
A = log(X);
return A;
}
Test(213);
R = Test(213);

```

- the function calculates $\log(X)$ and returns the value - $R = \log(213)$;

- if a command is introduced in the function after the RETURN command, it will not be displayed; the return command terminates the function;

- the returned value is stored in variable R.

The screenshot shows a code editor on the left and a preview window on the right. The code editor contains the following code:

```

1 double Test(double X) {
2     TEXT("X = " & X);
3     A = log(X);
4     return A;
5 }
6 TEXT("Call local procedure:");
7 Test(213);
8

```

The preview window shows the output of the function call:

Call local procedure: X = 213
A=log(X)=log(213)=2.33

[Load the example of an easy function: Functions-easy.cls](#)

[Load the example with two functions: Functions2.cls](#)

[Load an example of loading external user function: user_function_extern.zip](#)

[Load a more complex example: Functions.cls](#)

Static Math class

Static class "Math" contains:

- Math.E = 2,7182818284590452353
- Math.PI = 3,14159265358979323846
- Math.GetPolynomRoots(IList Coefficients) - find roots of general polynomial (level up to 100)

GRAPH

In Scia Design Forms, a graph is a type of visual component generated by means of commands from the source code. In the layout, the entire graph may be selected as a single component. In addition, a [POINT](#) or a [LINE](#), defined by an array of points (by coordinates), can be drawn inside a graph.

A graph is composed of two principal parts:

- An object defined by the NEW GRAPH command - contains graph axes with labels and gridlines;
- Point or line objects drawn in the graph.

How to define a graph:

1. Define a graph object using the NEW GRAPH(); command;
2. Prepare the variables that will contain the function to be displayed (coordinates of points that will define the graph);
3. Define the line style;
4. Define additional points on the graph, if needed;
5. Define additional visual elements of the graph - axes, labels, fonts, etc.
6. Draw the graph (to display in layout).

object <variable> = new Graph();

The graph object can later be drawn using the command object <variable>.Draw(<width>, <height>);

Syntax:

```
object <variable> = new Graph();
```

- it defines an object of the GRAPH type; the object properties are object name, names of axes, division of axes, scale for axes division, and formatting of the labels and values on axes.

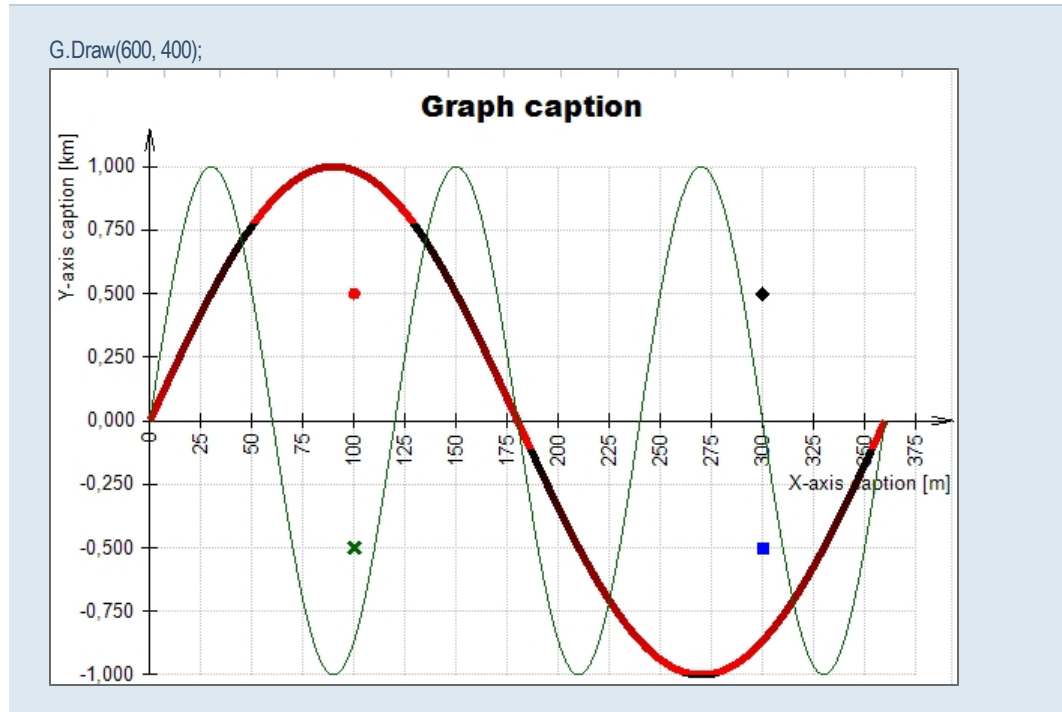
Example

```
object G = new Graph();
```


object <variable>.Draw()**Syntax:**

```
object <variable>.Draw(<width>, <height>);
```

- this command draws a previously defined graph into a visual component in the layout with the given size <width>x<height>.

Example

[Load an example: Graph_line.cls](#)

Definition of geometry in graphs - array of points**Line definition - an array of points (by coordinates)**

To draw lines in a graph, a variable of type array has to be defined. The function used to define a line specifies an array of points.

Syntax:

```
object[] <variable> = new object[];
```

- it defines an container object that will store point coordinates. The object will later on be used to define the line in the graph.

Example

```
object[] Pts1 = new object[];
```

Points (point is defined by the X and Y coordinates) must be assigned to the object - e.g. using a FOR cycle:

```
object[] Pts1 = new object[];
object[] Pts2 = new object[];

// create data points
FOR(i, 0, Count) {
    Pts1[i] = new PointD(i, 1000*sin(i));
    Pts2[i] = new PointD(i, 1000*sin(3*i));
}
```

- the example contains a FOR cycle that draws two different sin curves (the X-coordinate i is paired with the Y-values $\sin(i)$ and $\sin(3i)$ respectively).

[Load an example: Graph_line.cls](#)

Definition of geometry in graphs - points

Syntax:

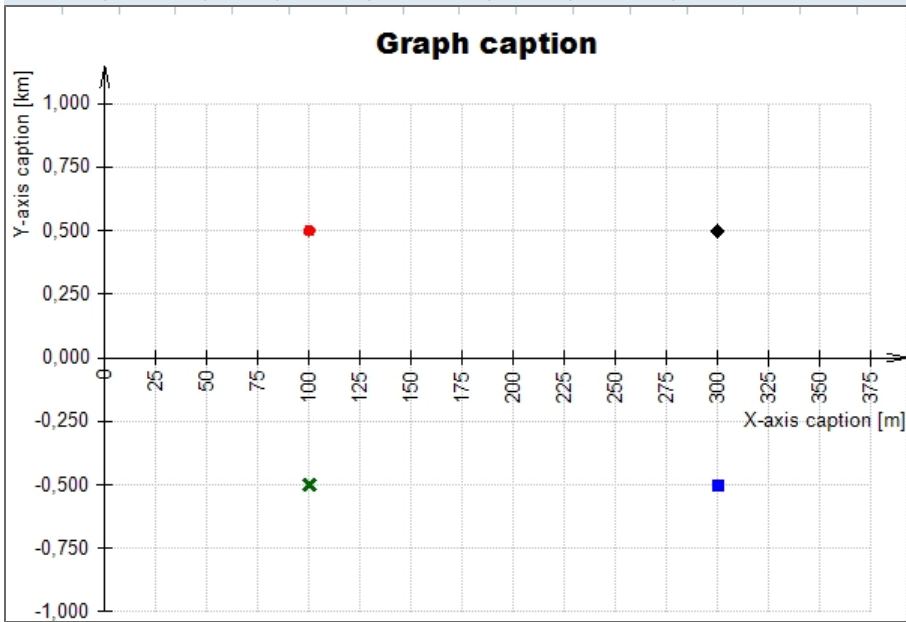
```
object <variable>.AddData(new PointF(<X>, <Y>), new Pen(Color<colour_name>, <thickness>));
```

The function parameters are: two points defined by means of coordinates using the POINT C# function (more [here](#)), colour definition and thickness, optionally also a drawing style (see the example below).

Example

```
G.AddData(new PointF(100, 500), new Pen(Color.Red, 2));
G.AddData(new PointF(100, -500), new Pen(Color.DarkGreen, 3), PointStyle.Cross);
G.AddData(new PointF(300, -500), new Pen(Color.Blue, 0), PointStyle.Rect);
```

```
G.AddData(new PointF(300, 500), new Pen(Color.Black, 0), PointStyle.Diamond);
```



[Load an example: Graph_point.cls](#)

Basic brush of new Pen type

Basic new Pen brush settings (Color.<colour_name>, <thickness>);

Syntax:

```
<variable>.AddData(<co-ordinates_object>, new Pen(Color.<colour_name>, <thickness>);= <value>;
```

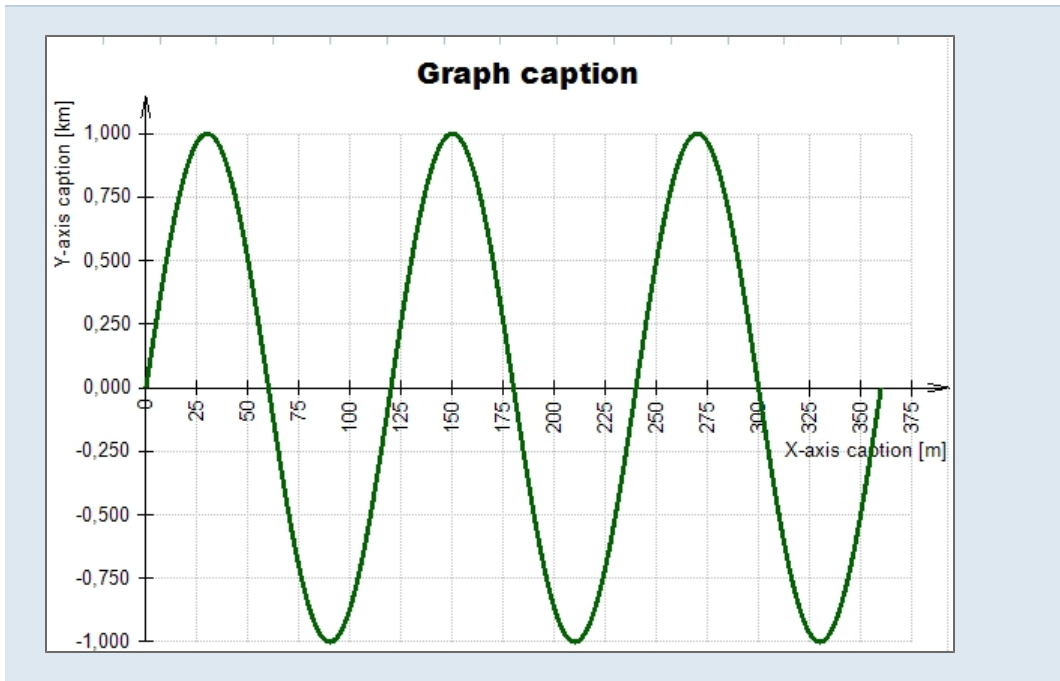
- it draws a line defined by the coordinates object using a pen with fixed colour and thickness;

If the thickness is set to 0 - the thinnest possible line is drawn.

Example

```
G.AddData(Pts2, new Pen(Color.DarkGreen, 3));
```

- graph G object with coordinates from Pts2 is drawn in dark green with a thickness of 3.



[Load an example: Graph_brushes.cls](#)

Brush of Gradient type

LinearGradientBrush()

GRADIENT is one of the two options for brush adjustment when drawing lines in graphs. Using the brush, the user can draw lines in graphs in two-colour gradient. The gradient angle may be defined by using one of the two command arguments. This brush can be used to filling the polyline ([see a separate chapter](#)).

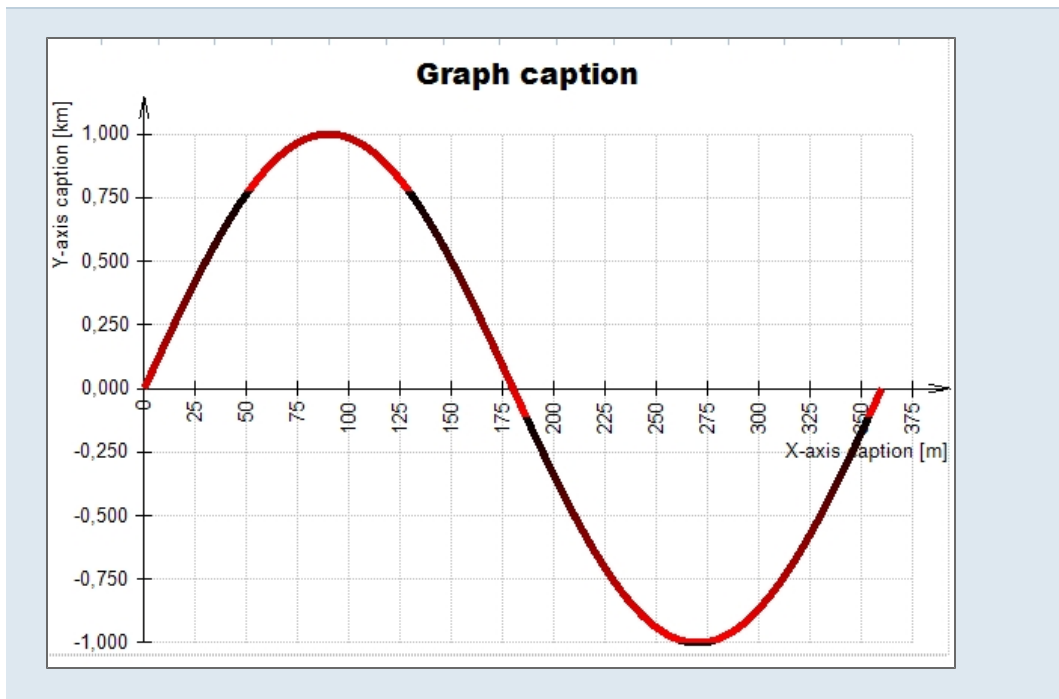
Syntax:

```
object <variable> = new LinearGradientBrush(new Point(X, Y), new Point(X, Y), Color.<colour_name>, Color.<colour_name>);
```

- the parameters of this function are: the two points defining the colour-gradient angle, and the two gradient colours (in the included example these are black and red).

Example

```
object B = new LinearGradientBrush(new Point(0, 0), new Point(0, 150), Color.Black, Color.Red);  
- point 0,0 and point 0,150 define the gradient direction, the line is filled with black and red in gradient.
```



[Load an example: Graph_brushes.cls](#)

Brush of Hatch type

HatchBrush()

This is another option for brush adjustment for drawing lines in graphs. Instead of a solid colour, the user can use a hatch pattern to fill a line. The function allows the user to adjust the pattern of the used hatch, as well as the foreground and background colours.

Syntax:

```
object <variable> = new HatchBrush(HatchStyle.<hatch_pattern>, Color.<colour_name>, Color.<colour_name>);
```

- a hatch is used for the graph line.

The parameters are:

- hatch patterns;
- hatch (foreground) colour;
- background colour.

A list of patterns can be found [here](#).

For example:

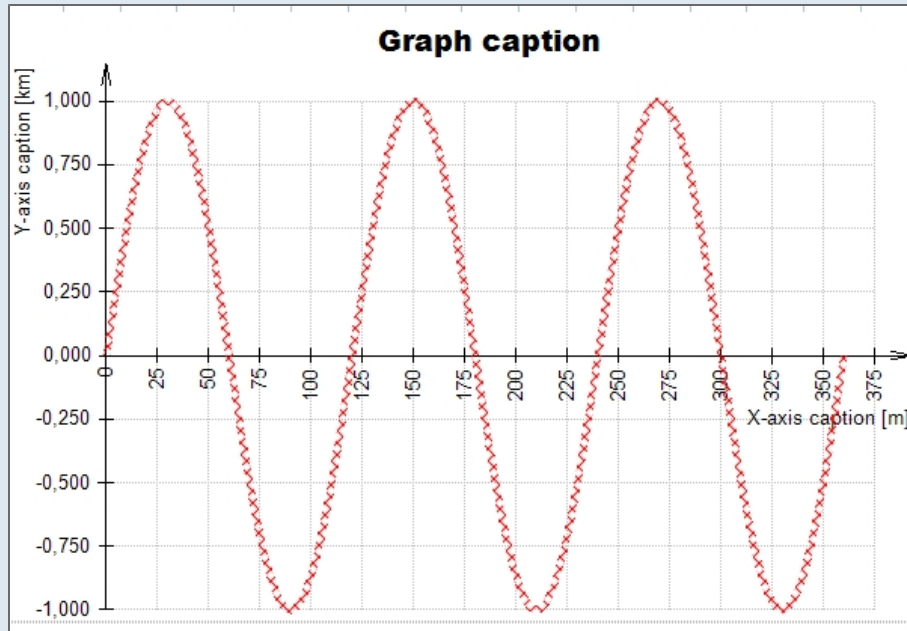
- Horizontal - a horizontal pattern;
- Vertical - a vertical pattern
- Cross - both horizontal and vertical patterns combined in one pattern;

- DiagonalCross - a diagonally oriented Cross pattern;
- Percent25 - The ratio of foreground to background colour is 25:75.

Example

```
object B = new HatchBrush(HatchStyle.DiagonalCross, Color.Red, Color.Transparent);
```

- the hatching combines two perpendicular sets of lines, the foreground colour is red, the background is set to transparent



[Load an example: Graph_brushes.cls](#)

Definition of graph properties - labels

All following steps must be set as invisible in the layout, as these define visualization settings and are of no interest to the user of the calculation form.

`<variable>.Caption.Caption - Graph title`

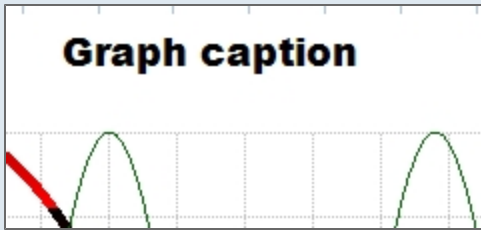
Syntax:

```
<variable>.Caption.Caption = "<graph_name>";
```

- it writes a title above the graph.

Example

```
G.Caption.Caption = "Graph caption";
```



```
<variable>.Caption.Font = new Font() - Graph title font
```

Syntax:

```
<variable>.Caption.Font = new Font("<font>", <font_size>);
```

- it sets the font type and size of the graph title.

Example

```
G.Caption.Font = new Font("Arial Black", 12);
```

```
<variable>.XAxis.Caption - Name of the X-axis
```

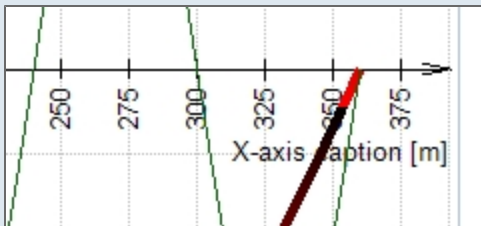
Syntax:

```
<variable>.XAxis.Caption = "<x_axis_name>";
```

- it writes a name (label) on the x-axis.

Example

```
G.XAxis.Caption = "X-axis caption [m]";
```



An alternative notation for the Y-axis: `G.YAxis.Caption = "Y-axis caption [m]";`

<variable>.XAxis.Font= new Font() - X-axis label font

Syntax:

```
<variable>.XAxis.Font = new Font("<font>",<font_size>);
```

- it sets the font type and size for the X. axis.

Example

```
G.XAxis.Font = new Font("Calibri", 12);
```

An alternative notation for the Y-axis: G.YAxis.Font = new Font("Calibri", 12);

Definition of graph properties - values

All following steps must be set as invisible in the layout, as these define visualization settings and are of no interest to the user of the calculation form.

Axis grid size and grid scale

<variable>.XAxis.MajorStep - X-axis grid size

Syntax:

```
<variable>.XAxis.MajorStep = "<x_axis_step>";
```

- it sets the X-axis grid size

Example

```
G.XAxis.MajorStep = 25;
```



An alternative notation for the Y-axis: G.YAxis.MajorStep = 250;

<variable>.YAxis.Scale - Axis-step scale**Syntax:**

```
<variable>.YAxis.Scale = <value>; // sets the Y-axis grid size
```

Example

```
G.YAxis.Scale = 0.001;
```

- the Y-axis step is 1000 times smaller than the real dimension



An alternative notation for the X-axis: G.XAxis.Scale = 0.001;

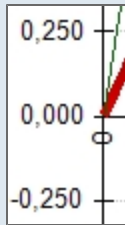
Axis value format**<variable>.YAxis.Format - Axis value format****Syntax:**

```
<variable>.YAxis.Format = <value>;
```

- it sets the Y-axis value format (0.000, 00.00 ...)

Example

```
G.YAxis.Format = "0.000";
```



```
An alternative notation for the X-axis: G.XAxis.Format = "0.000";
```

Tables in the layout

In order to create clear calculation reports it is often useful to organize data (input, descriptions, images, output values, etc.) into tables in the layout. A table displayed in the layout is declared as an object variable. Each cell of the table can be then assigned a value through the script using indexes.

A table in the layout has:

- Table style - pen, colours, etc.;
- Cell contents.

To define a table:

1. define a new Table() object;
2. define the cell content by indexing to each cell;
3. define the line style, colour, font (for cells, rows or columns);
4. define individual table parts - axes, labels, etc.;
5. display the table;

Syntax:

```
object <variable> = new Table(<number_of_columns>);
```

- it defines a Table object. The object is later displayed using the command `object <variable>.Draw()`;

Example

```
object T = new Table(4); - it defines a new table with 4 columns.
```


object <variable>.Draw()**Syntax:**

```
object <variable>.Draw(<width>, <height>);
```

- the command displays the predefined table in the layout as a visual component .

Example

```
T.Draw();
```

Column one	Second column	Column with text and formula	Image
1	10	Hello world! $X = X + 1 = 0 + 1 = 1$	
2	20	Hello world! $X = X + 1 = 1 + 1 = 2$	
3	30	Hello world! $X = X + 1 = 2 + 1 = 3$	
4	40	Hello world! $X = X + 1 = 3 + 1 = 4$	
5	50	Hello world! $X = X + 1 = 4 + 1 = 5$	

[Load the more complex example: Table_complex.CLS](#)

Table - content definition

Each cell in a table is defined by indexes of rows columns.

```
<variable>[<row_index>][<column_index>].Value = "<text>"
```

This defines the text to be displayed in the table cell with row number <row_index> and column number <column_index>.

Syntax:

```
<variable>[<row_index>][<column_index>].Value = "<text>";
```

- it displays the text in the defined cell.

Example

T[0][0].Value = "Column one"; - the text "Column one" will be displayed in the first row of the first column

Column one	Second column
1	10

Use the commands for Width and Wrap to define a multiline text, see the chapter below.

If the cell has predefined width, the content maybe automatically wrapped, see the command Wrap = true.

```
FOR(i, 1, 5){  
  T[i][2].Value = "Hello world!";  
}
```

- each cell of the second column is assigned the text "Hello world!"

[Load an example: Table_layout_content.cls](#)

Wrap = true

The cell content with this property will be automatically wrapped. Text is divided in white-space characters.

Conditions:

- Width of the cell is greater than zero (= is set);
- The wrapping property is set to true.

Syntax:

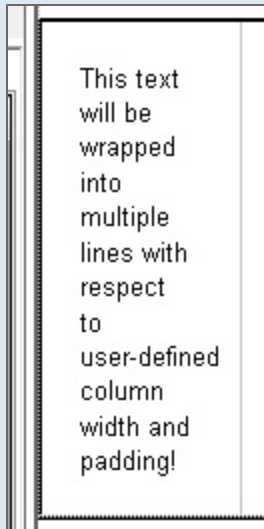
```
<variable>.Columns[i].Width = <value>;  
<variable>.Columns[i].Wrap = true;
```

- the text in the i column will be wrapped. The defined width (<value>) will be used.

Example

```
T.Columns[0].Width = 100;  
T.Columns[0].Wrap = true;
```

```
T[0][0].Value = "This text will be wrapped into multiple lines with respect to the user-defined column width and padding!";
```



Auto-wrapping is applied only to cells which contains values. Cell with instructions (TEXT, IMG, ...) are not affected!

Table - shape definition

A table can be defined in the layout:

- Using a pen for outside border lines - `<variable>.BorderPen = new Pen(Color.<colour>, <thickness>);`
- Using a pen for inside border lines - `<variable>.GridPen = new Pen(Color.<colour>, <thickness>);`
- Using the font for any row - `<variable>[<index>].Font = new Font("[<font_name>]", <size>);`
- Using a pen for any row - `<variable>[<index>].BorderPen = T.BorderPen;`
- Using column alignment - `<variable>.Columns[<index>].Alignment = ContentAlignment.MiddleCenter;`

Column definition:

```
<variable>.Columns[<index>] ...
```

Row definition:

```
<variable>[<index>] ...
```

Alignment in tables - columns

```
<variable>.Columns[<index>].Alignment = ContentAlignment.<alignment>;
```

Adjusts alignment of table columns.

Syntax:

```
<variable>.Columns[<index>].Alignment = ContentAlignment.<alignment>;
```

Example

```
T.Columns[2].Alignment = ContentAlignment.MiddleCenter;
```

Alignment in tables - rows

```
<variable>[<index>].Alignment = ContentAlignment.<alignment>;
```

Adjusts alignment of table rows.

Syntax:

```
<variable>[<index>].Alignment = ContentAlignment.<alignment>;
```

Example

```
T[0].Alignment = ContentAlignment.MiddleCenter;
```

Pen for outside border lines

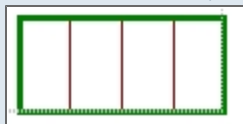
```
<variable>.BorderPen = new Pen(Color.<colour>, <thickness>)
```

Syntax:

```
<variable>.BorderPen = new Pen(Color.<colour>, <thickness>);
```

Example

```
T.BorderPen = new Pen(Color.Green, 3);
```



[Load an example: Table_pen.cls](#)

Pen for inside border lines

```
<variable>.GridPen = new Pen(Color.<colour>, <thickness>)
```

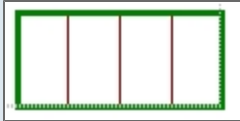
Adjusts the pen for the inside border lines of a table.

Syntax:

```
<variable>.GridPen = new Pen(Color.<colour>, <thickness>);
```

Example

```
T.GridPen = new Pen(Color.DarkRed, 1);
```



[Load an example: Table_pen.cls](#)

Pen for outside border lines of a single row

```
<variable>[<index>].BorderPen = <variable>.BorderPen
```

Adjusts the pen for any table row.

Syntax:

```
<variable>[<index>].BorderPen = <variable>.BorderPen;
```

```
<variable>[<index>].BorderPen = new Pen(Color.<colour>, <thickness>);
```

Example

```
T.BorderPen = new Pen(Color.DarkRed, 3);
```

```
T[0].BorderPen = T.BorderPen;
```

- The previous pen settings for outer border lines will be used also for the inside border lines of the row with index 0

[Load an example: Table_to_layout.cls](#)

Font for a single row

```
<variable>[<index>].Font = new Font ("<font>", <size>)
```

Adjusts the font for any table row.

Syntax:

```
<variable>[<index>].Font = new Font("<font>", <size>);
```

Example

```
T[0].Font = new Font("Arial Black", 12);  
- adjusts the font for the first table row.
```

Column one	Second column	Column with text and formula
1	10	Hello world! $X = X + 1 = 0 + 1 = 1$

Cell padding

```
<variable>.Padding.All = <value>
```

The cell padding specifies the offset (in px) from the cell border to the cell content to the whole table.

Syntax:

```
<variable>.Padding.All = <value>;
```

Example

```
T.Padding.All = 15; - it defines padding 15 px for all cells in the table.
```

```
<variable>[i].Padding.All = <value>
```

The cell padding specifies the offset (in px) from the cell border to the cell content to the whole i-row.

Syntax:

```
<variable>[i].Padding.All = <value>;
```


Example

```
T[1].Padding.All = 15; - it defines padding 15 px for all cells in the second row (index 1).
```

<variable>[i][y].Padding.All = <value>

The cell padding specifies the offset (in px) from the cell border to the cell content to the cell with indexes i, y.

Syntax:

```
<variable>[i][y].Padding.All = <value>;
```

Example

```
T[1][0].Padding.All = 15; - it defines padding 15 px for the first cells on the second row (index 1,0).
```

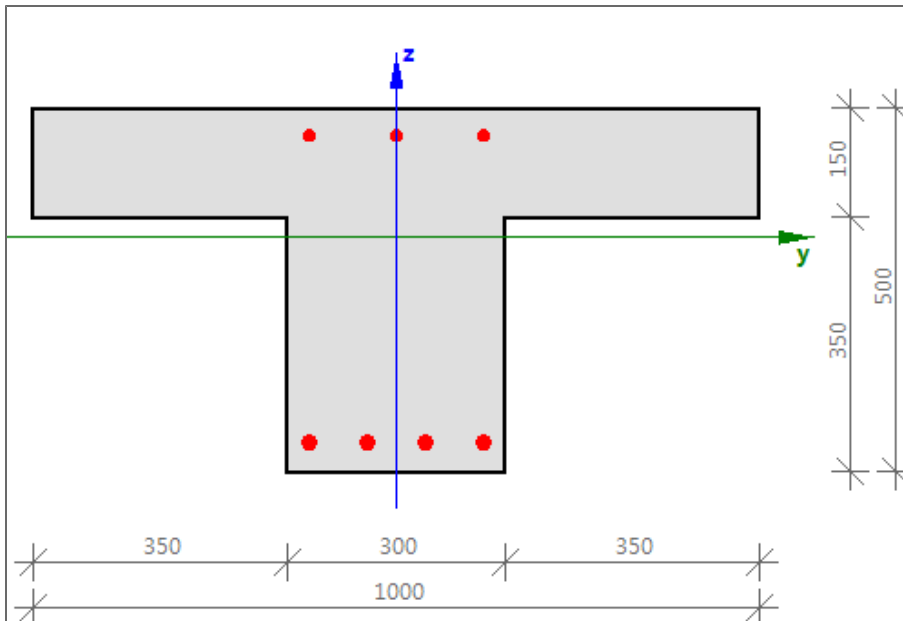
[Load an example: Cell_padding.cls](#)

Graphics

Graphic operations provide vector graphics in the calculation layout.

A graphic object is saved in the variable type "Graphics" (command new Graphics()).

The graphics allows to create polygons, lines, dimensions, hatched areas and similar.



new Graphics()

The command creates the Graphics object. This object will contain all graphics data. All other commands for graphics uses dot-convention which starts with this object (e.g. G.DrawArc(...)) - the new arc is visible when graphics object G is displayed in the layout).

Syntax:

```
object <variable> = new Graphics();
```

Example

```
object G = new Graphics();
```

Draw() - draw graphics, parameters

The graphics in the object is drawn to the layout. The drawing size has different parameters (width, height, zoom, angle ...).

Draw(zoom)

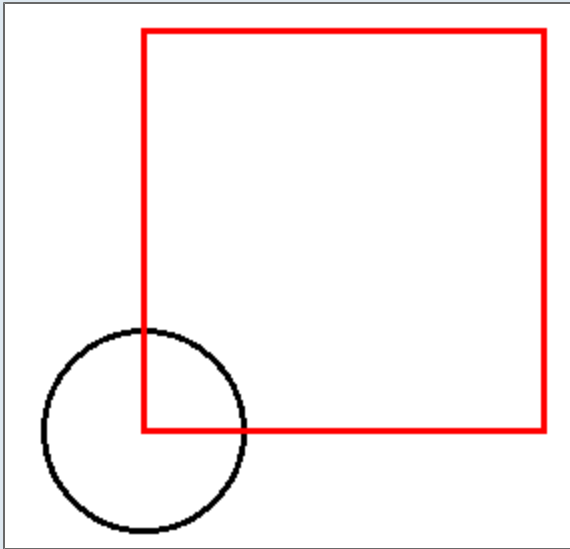
Syntax:

```
<variable>.Draw(<zoom_value>);
```

- it draws all elements which are defined in the graphics object with the predefined zoom, this syntax is the fastest one

Example

G1.Draw(1); - object G1 is drawn with no zoom.



Draw(width, height, boolean of aspect ratio)

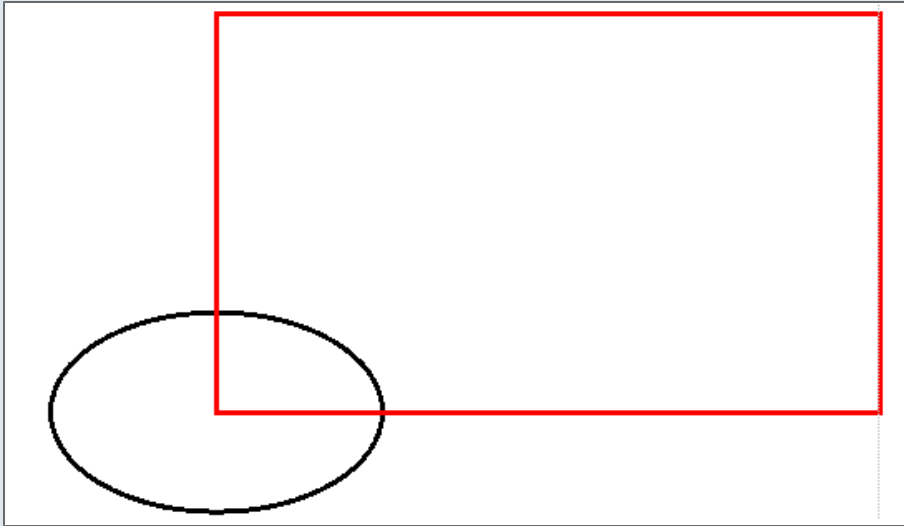
Syntax:

```
<variable>.Draw(<width>, <height>, <boolean_of_aspect_ratio>);
```

- it draws all elements to the bounding box with predefined sizes, the boolean parameter defines if the aspect ratio is True or False. This syntax is much slower than the first one (about 50-times slower).

Example

G1.Draw(500, 300, false); - object G1 is displayed in the bounding-box 500x300, the sizes don't respect its original ratio.



G1.Draw(500, 300, true); - object G1 is displayed in the bounding-box 500x300, the sizes respect its original ratio, if the sizes of the bounding-box is not in the correct ratio, the default value is the smaller one, the second size is recalculated according to the ratio. In this case 300 will stay and 500 will be changed.

Draw(enlarge horizontal, enlarge vertical, rotation angle)

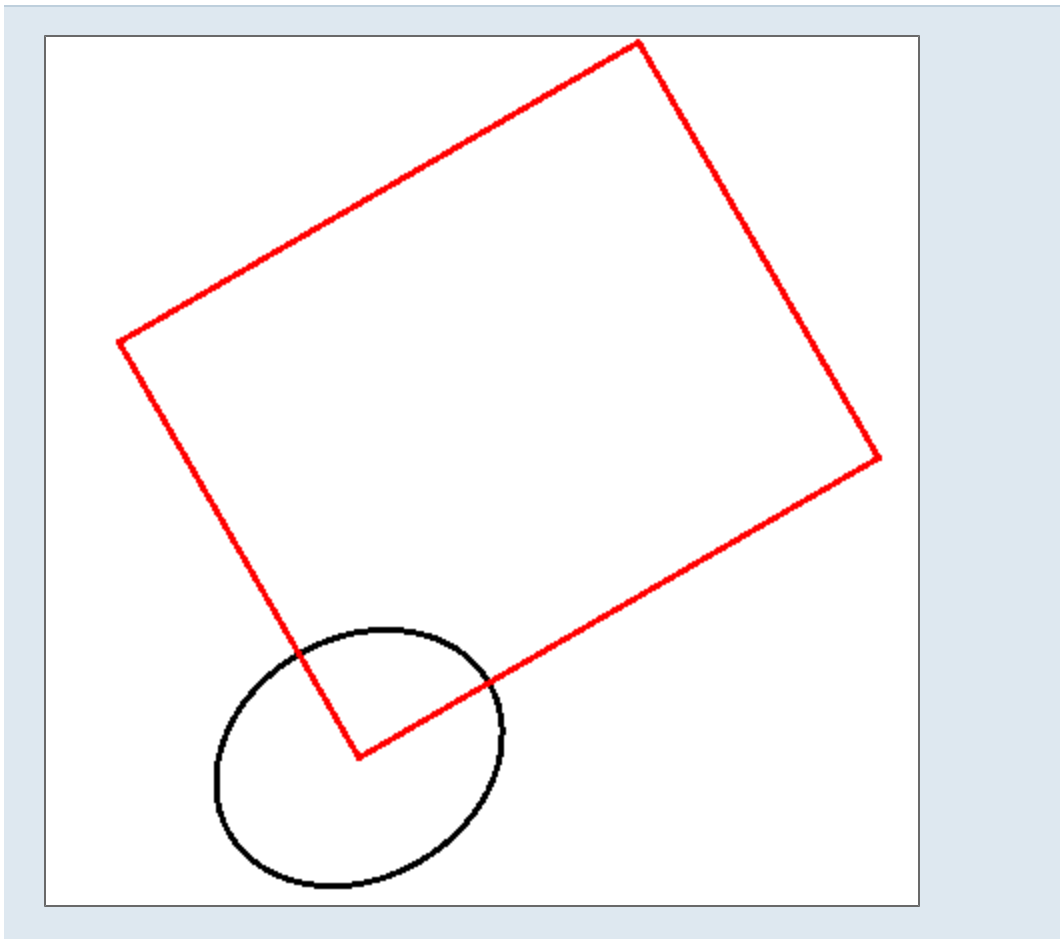
Syntax:

```
<variable>.Draw(<enlarge_horizontal>, <enlarge_vertical>, <rotation_angle>);
```

- it draws all elements, it allows to enlarge the size of both sides and rotate the graphics according to the predefined angle.

Example

G1.Draw(1.5, 1.2, 30); - object G1 is displayed 1,5x bigger in the horizontal direction, 1,2x bigger in the vertical direction and rotated by 30°.



Draw(width, height)

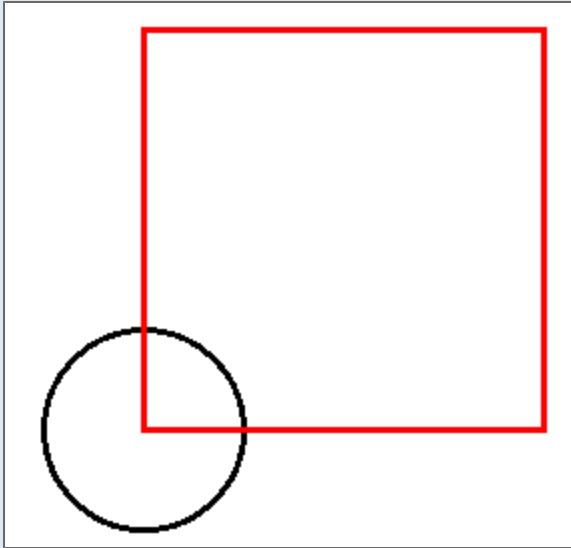
Syntax:

```
<variable>.Draw(<width>, <height>);
```

- it draws all elements to the bounding-box with predefined sizes. This syntax is much slower than the first one (about 50-times slower).

Example

G1.Draw(500,300); - object G1 is displayed in the bounding-box 500x300 px.



[Load the example with all parameter types: Draw.cls](#)

DrawGraphics() - one graphics to another

This command draws one graphic object to another graphic object. It allows user to prepare some part separately and then add it by one command.

DrawGraphics(<other_graphics>, <movement_x>,<movement_y>)

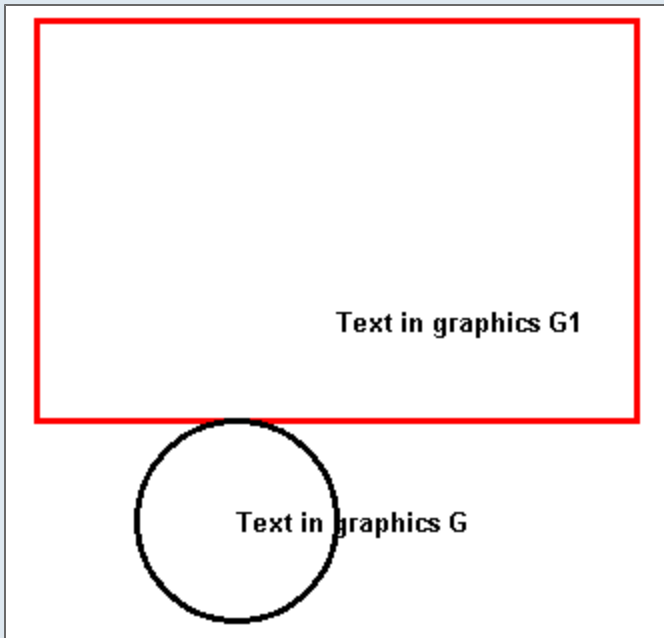
Syntax:

```
<variable>.DrawGraphics(<other_graphics_variable>, <movement_x>, <movement_y>);
```

- it draws external graphics to the predefined position to the current one.

Example

G.DrawGraphics(G1, 0, 50); - it draws external graphics G1 to the graphics G on position X = 0, Y = 50. No zoom is used.



DrawGraphics(<other_graphics>, <movement_x>, <movement_y>, <angle>)

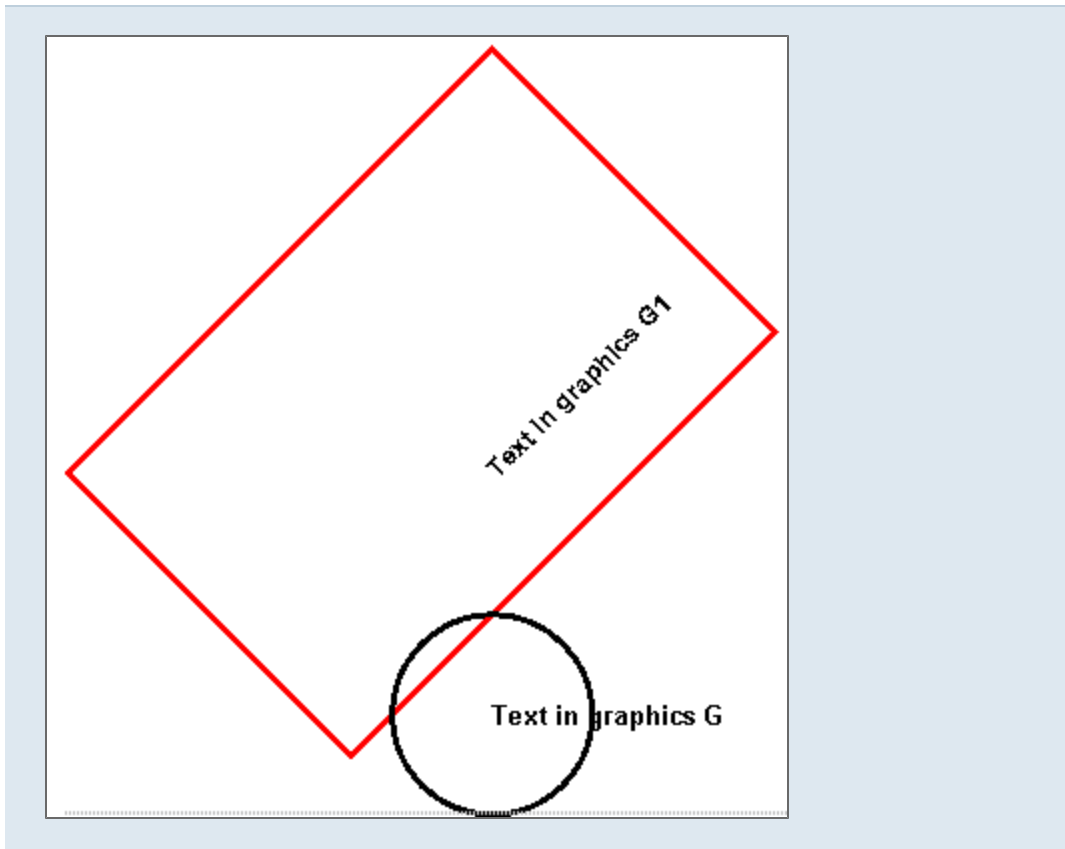
Syntax:

```
<variable>.DrawGraphics(<other_graphics_variable>, <movement_x>, <movement_y>, <angle>);
```

- it draws external graphics to the predefined position and with the predefined rotation to the current one.

Example

G.DrawGraphics(G1, 0, 50, 45); - it draws external graphics G1 to graphics G on position X = 0, Y = 50 and rotate it by 45°. No zoom is used.



DrawGraphics(<other_graphics>, <movement_x>, <movement_y>, <zoom_x>, <zoom_y>, <angle>);

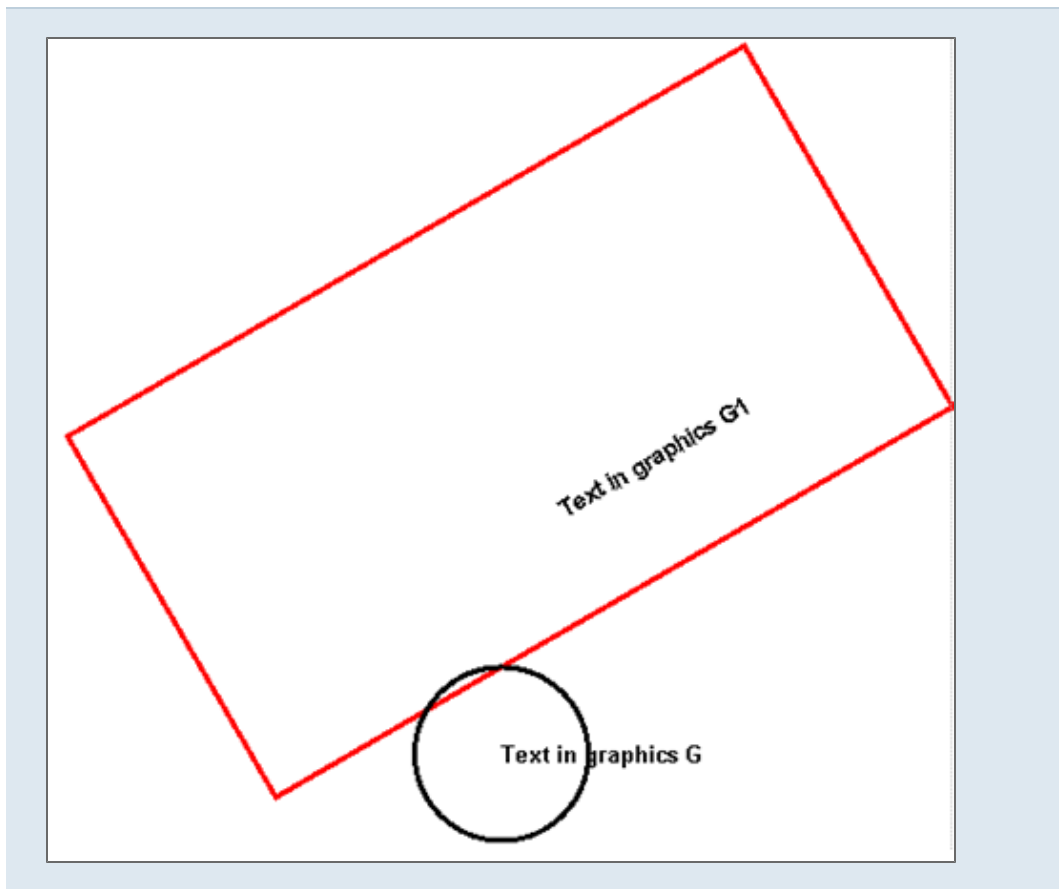
Syntax:

```
<variable>.DrawGraphics(<other_graphics_variable>, <movement_x>, <movement_y>, <zoom_x>, <zoom_y>, <angle>);
```

- it draws external graphics to the predefined position in the current one. Parameters are zoom, position and rotation.

Example

```
G.DrawGraphics(G1, 0, 50, 1.5, 1.2, 30);- it draws external graphics G1 to graphics G on position X = 0, Y = 50, its bigger 1,5x in horizontal direction, 1,2x in vertical direction and it rotates G1 by 30°.
```

[Load the example with all parameter types: DrawGraphics.cls](#)

UCS operations

Each graphics is drawn in User coordinate system. This system is defined by its origin and directions of axis X and Y. This UCS may be moved and modified by the following commands. The graphic objects is then drawn step by step - see overall example.

SaveUCS

This command saves the current used UCS. All UCS systems are saved to the memory with the specified index. The index is unique identifier of the UCS system.

Syntax:

```
<variable>.SaveUCS(<index>);
```

- it saves the current user coordinate system under the defined index.

Example

```
G.SaveUCS(0); - it saves the current used UCS to position 0.
```

LoadUCS

This command loads the previously saved UCS.

Syntax:

```
<variable>.LoadUCS(<index>);
```

- it loads a previously saved UCS which is defined by the index.

Example

```
G.LoadUCS(0); - it loads UCS with index 0.
```

MoveUCS

This command moves the current UCS to the specific location. The new position of the system origin is defined by coordinates X and Y (double values).

Syntax:

```
<variable>.MoveUCS(<x>, <y>);
```

- Move UCS to defined location - x, y.

Example

```
G.MoveUCS(150, 0); - it moves UCS to the new origin 150,0.
```

RotateUCS

This command rotates the current UCS by a defined angle (in °).

Syntax:

```
<variable>.RotateUCS(<angle>);
```

- it rotates UCS by a specific angle.

Example

```
G.RotateUCS(30); - it rotates UCS by 30°.
```

ScaleUCS

This command scales the current UCS (the drawn graphics) in directions X and Y. If one axis is scaled by 2 it means that the graphics will be displayed twice longer in this direction.

Syntax:

```
<variable>.ScaleUCS(<scale_x>, <scale_y>);
```

- it scales UCS in X and Y direction.

Example

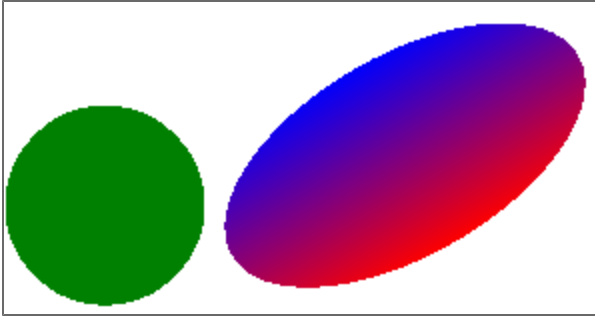
```
G.ScaleUCS(2, 1);
```

- Scale actual graphics by 2 in X-axis and by 1 in Y-axis.

Overall example

```
object G = new Graphics();
object LinGradBrush = new LinearGradientBrush(new PointF(250, -100), new PointF(250, 0), Color.Blue, Color.Red);
G.FillCircle(0, 0, 100, Color.Green);
G.SaveUCS(0);
G.MoveUCS(150, 0);
G.RotateUCS(30);
G.ScaleUCS(2, 1);
G.FillCircle(0, 0, 100, LinGradBrush);
G.LoadUCS(0);
G.Draw(1);
```

- it draws a green circle with diameter 100 with the center in 0,0. Its UCS is saved under index 0.
- the origin of USC with index 0 is moved to 150,0, the UCS is rotated by 30°, the scale in X is 2 and in Y is 1.
- the second circle is drawn with linear gradient brush from blue to red with diameter 100, the center of the second circle is 0,0 but this is already moved and modified UCS. The second UCS is scaled in Y direction, so the circle is displayed as ellipse.
- the original UCS with index 0 is then loaded back.
- the whole graphics is drawn with zoom 1.



[Load the example: UCSOperations.cls](#)

Angle

This command changes the rotation of the graphic object from 0° to the defined value.

Syntax:

```
<variable>.Angle = <value>;
```

- value in °

Example

```
G.Angle = 30; - it rotates the object G by 30°.
```

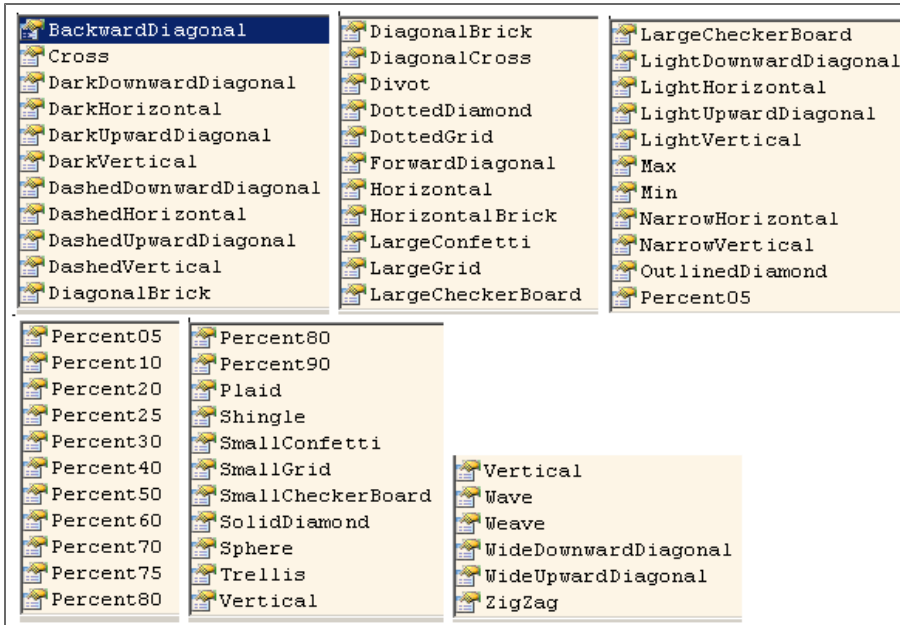
Brushes

This command defines the hatch pattern for a Brush which is used for filling the graphics object. The command `LinearGradientBrush` is used for filling the polygon graphics.

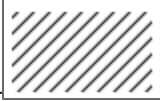

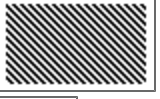





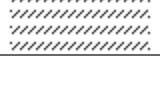
The parameters are:

- hatch patterns;
- hatch (foreground) colour;
- background colour.

A list of patterns can be found [here](#).



For example:

- BackwardDiagonal - 
- Cross - 
- DarkDownwardDiagonal - 
- DarkHorizontal - 
- DarkUpwardDiagonal - 
- DarkVertical - 
- DashedDownwardDiagonal - 
- DashedHorizontal - 
- DashedUpwardDiagonal - 

Syntax:

```
object <name> = new HatchBrush( HatchStyle.<hatch_pattern>, Color.<colour_name>, Color.<colour_name>);
```

- it defines an object for a hatch brush type with two colours - the first colour is foreground colour and the second is background colour.

```
object <NameOfObject> = new SolidBrush(Color.<colour_name>);
```

- it defines an object for a solid brush type with defined colour.

```
object <NameOfObject> = new LinearGradientBrush(<begin_object>, <end_object>, Color.<colour_name>, Color.<colour_name>);
```

- it defines an object with Linear gradient brush with predefined colours and vector for a gradient.

The begin object - starting point of the gradient vector, it may be defined by Point, PointF or struct[X,Y].

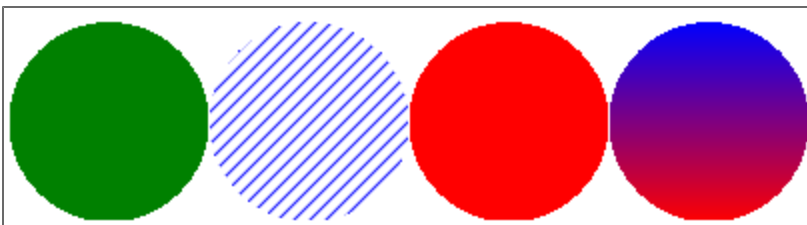
The end object - finishing point of the gradient vector, it may be defined by Point, PointF or struct[X,Y].

Example

```
object HatchBrush = new HatchBrush(HatchStyle.BackwardDiagonal, Color.Blue, Color.White); - it defines an object HatchBrush, the hatch style is backward diagonal lines, the foreground colour is blue and the background is white.
```

```
object SolidBrush = new SolidBrush(Color.Red); - it defines an object SolidBrush with solid red colour.
```

```
object LinGradBrush = new LinearGradientBrush(new PointF(250, -100), new PointF(250, 0), Color.Blue, Color.Red); - it defines an object LinGradBrush, the vector for the gradient is defined by two points [250,-100] and [250,0]. The first colour is blue and the second colour is red.
```



[Load the example: DrawCircle.cls](#)

DrawLine

One of the most used command. This command draws a line between two points.

The points are defined:

1. by their coordinates (double value)
2. by objects, the command accepts object type Point, PointD, PointF, struct[X,Y]

Syntax:

```
<Variable>.DrawLine(<x1>, <y1>, <x2>, <y2>);
```

- it draws a line between two points. The default pen is used.

```
<Variable>.DrawLine(<x1>, <y1>, <x2>, <y2>, <pen_style>);
```

- it draws a line between two points. The predefined pen style is used.

```
<Variable>.DrawLine(<begin_point_object>, <end_point_object>);
```

- it draws a line between points which are defined by objects. The default pen is used.

```
<Variable>.DrawLine(<begin_point_object>, <end_point_object>, <pen_style>);
```

- it draws a line between points which are defined by objects. The predefined pen is used.

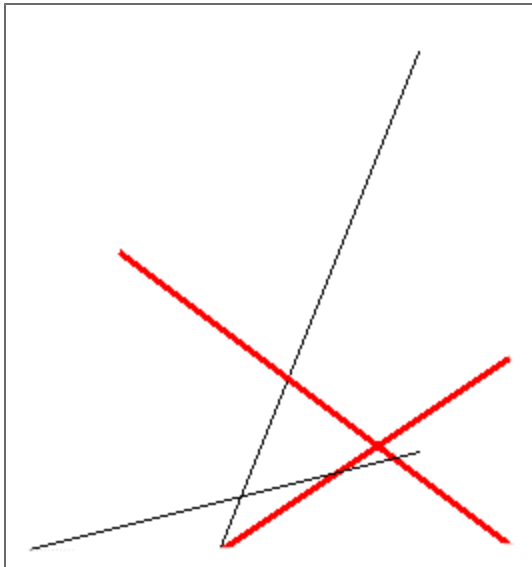
Example

```
G.DrawLine(0, 0, 100, 250); - it draws a line from 0, 0 to 100, 250. The default pen style is used.
```

```
G.DrawLine(0, 0, 150, 100, Pen); - it draws a line from 0, 0 to 150, 100. The predefined style Pen is used.
```

```
G.DrawLine(new PointD(-100, 0), new PointD(100, 50)); - it draws a line from point -100,0 to point 100, 50. The default pen style is used.
```

```
G.DrawLine(new PointD(-50, 150), new PointD(150, 0), Pen); - it draws a line from point -100, 0 to point 100, 50. The predefined style Pen is used.
```



[Load the example: DrawLine.cls](#)

DrawPolyline, FillPolygon

DrawPolyline

This command draws a polygon defined by array of points.

The array of points is defined by Point, PointD, PointF or struct[X, Y].

Syntax:

```
<Variable>.DrawPolyLine(<array_of_points>);
```

- it draws a polygon defined by an array of points. The default pen is used.

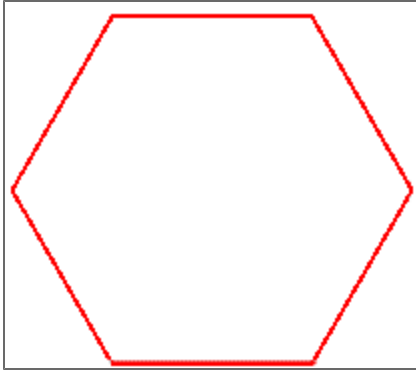
```
<Variable>.DrawPolyLine(<array_of_points>, <pen_style>);
```

- it draws a polygon defined by an array of points. The predefined pen style is used.

Example

```
G.DrawPolyLine(Pts); - it draws a polygon for array of points named Pts. The default pen style is used.
```

```
G.DrawPolyLine(Pts, Pen); - it draws a polygon for array of points named Pts. The predefined style Pen is used.
```

[Load the example: DrawPolyline.cls](#)

FillPolygon

This command fills an polygon with solid colour or brush.

The polygon may be defined by the object - list of points, the command accepts Point, PointD, PointF or struct[X,Y].

Syntax:

```
<Variable>.FillPolygon(<array_of_points>, Color.<colour_name>);
```

- it fills an polygon defined by the array of points with solid colour.

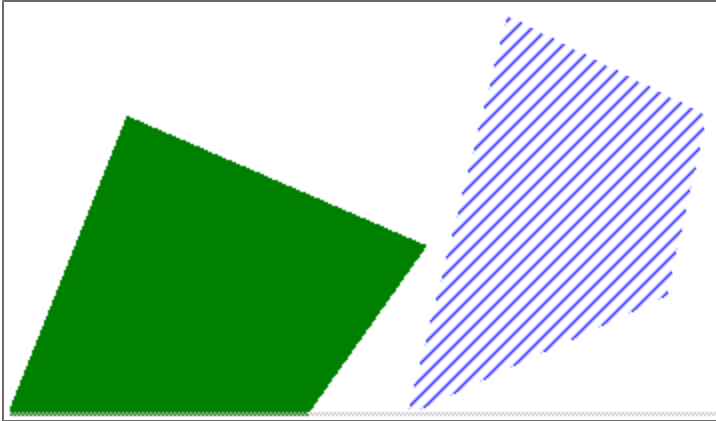
```
<Variable>.FillPolygon(<array_of_points>, <brush_name>);
```

- it fills an polygon defined by the array of points with the defined brush.

Example

```
G.FillPolygon(Pts1, Color.Green); - it fills an polygon from array Pts1 with the green colour.
```

```
G.FillPolygon(Pts2, HatchBrush); - it fills an polygon from array Pts2 with the brush named HatchBrush.
```



[Load the example: FillPolygon.cls](#)

DrawArrow

This command draws the arrow from the defined begin to the end point.

Syntax:

```
<variable>.DrawArrow(<begin_point> , <end_point> , ArrowShape.<shape> , <size> , <pen>);
```

- it draws arrow from the begin to the end point with predefined shape, predefined size and pen.

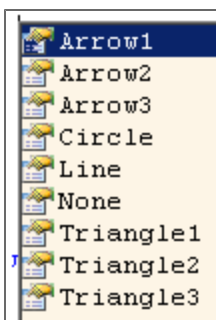
```
<variable>.DrawArrow(<begin_point> , <end_point>);
```










- it draws arrow from the begin to the end point with default settings.

```
<variable>.DrawArrow(<begin_point> , <end_point> , ArrowShape.<shape> , <size>);
```

- it draws arrow from begin the to the end point with predefined shape and predefined size. The default pen is used.

Possible arrow shapes:



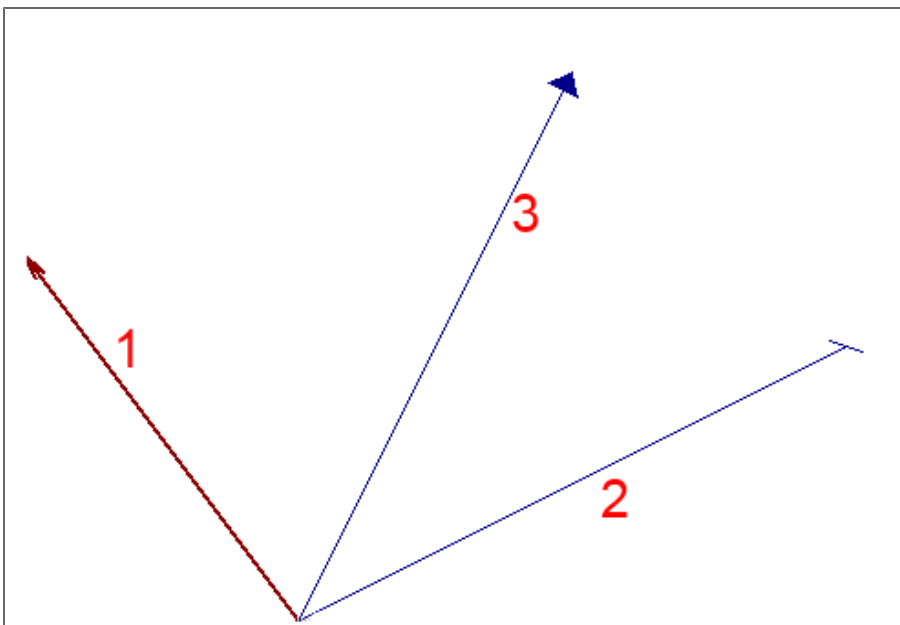
- **Arrow1** - blank arrow with angle 20° 
- **Arrow2** - blank arrow with angle 45° 
- **Arrow3** - blank arrow with angle 90° 
- **Circle** - filled circle 
- **Line** - slash 
- **None** - no arrow 
- **Triangle1** - filled arrow with angle 20° 
- **Triangle2** - filled arrow with angle 45° 
- **Triangle3** - filled arrow with angle 90° 

Example

`G.DrawArrow(new PointD(-150, 200), new PointD(0, 0), ArrowShape.Arrow1, 15, Pen);` - it defines arrow from 0,0 to 150, 200 with shape Arrow1, size is 15 px, the Pen style of pen is used.

`G.DrawArrow(new PointD(300, 150), new PointD(0, 0));` - it defines arrow from 0, 0 to 300, 150 with shape and size as default.

`G.DrawArrow(new PointD(150, 300), new PointD(0, 0), ArrowShape.Triangle3, 15);` - it defines arrow from 0,0 to 150, 300 with shape Triangle3, size is 15 px, the default pen style is used.



[Load the example: DrawArrow.cls](#)

DrawLeader

This command draws a leader with a specified text. The begin and end point is defined by objects.

Point object - the command accepts types Point, PointD, PointF, struct[X,Y].

Alignment styles:

- BottomCenter – Bottom, at the center
- BottomLeft – Bottom, left
- BottomRight – Bottom, right
- MiddleCenter – At the center, at the center
- MiddleLeft – At the center, left
- Middle Right – At the center, right
- TopCenter – Top, at the center
- TopLeft – Top, left
- TopRight – Top, right

Syntax:

```
<Variable>.DrawLeader(<begin_point_object>,<end_point_object>,<text>);
```

- it draws a leader with a predefined text. The text is automatically oriented according to leader line. The default dimension style is used.

```
<Variable>.DrawLeader(<begin_point_object>,<end_point_object>,<text>, ContentAlignment.<alignment>);
```

- it draws a leader with a predefined text. The text is oriented according to the defined alignment. The default dimension style is used.

```
<Variable>.DrawLeader(<begin_point_object>,<end_point_object>,<text>, ContentAlignment.<alignment>,  
TDimStyle.<dimension_style>);
```

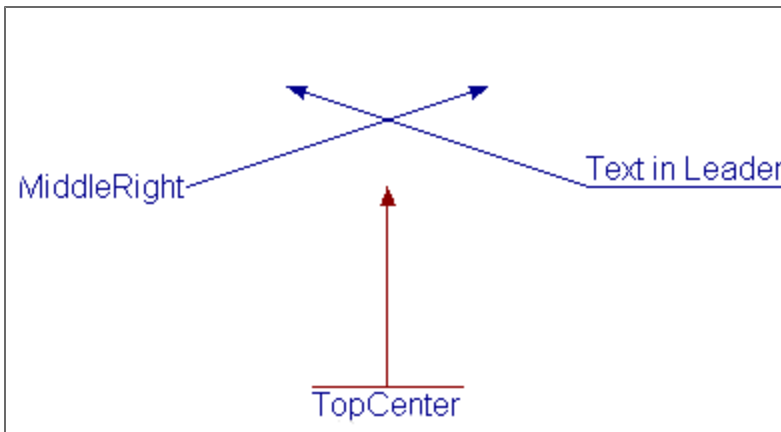
- it draws a leader with a predefined text. The text is oriented according to the defined alignment. The predefined dimension style is used.

Example

```
G.DrawLeader(new PointD(0, 50), new PointD(150, 0), "Text in Leader"); - it draws a leader with begin point [0,50] and  
end point [150,0] with text "Text in leader". The default alignment and default dimension style are used.
```

```
G.DrawLeader(new PointD(100, 50), new PointD(-50, 0), "MiddleRight", ContentAlignment.MiddleRight); - it draws a  
leader with begin point [100,50] and end point [-50,0] with text "Text in leader". The middle right alignment and default  
dimension style are used.
```

`G.DrawLeader(new PointD(50, 0), new PointD(50, -100), "TopCenter", ContentAlignment.TopCenter, DimStyle);` - it draws a leader with begin point [50,0] and end point [50, -100] with text "TopCenter". The top center alignment and DimStyle dimension style are used.



[Load the example: DrawLeader.cls](#)

DrawArc

This command draws an arc which represents the part of the ellipse.

Syntax:

```
<variable>.DrawArc(<x>, <y>, <width_of_circumscribed_rectangle>, <height_of_circumscribed_rectangle>, <begin_angle>, <end_angle>);
```

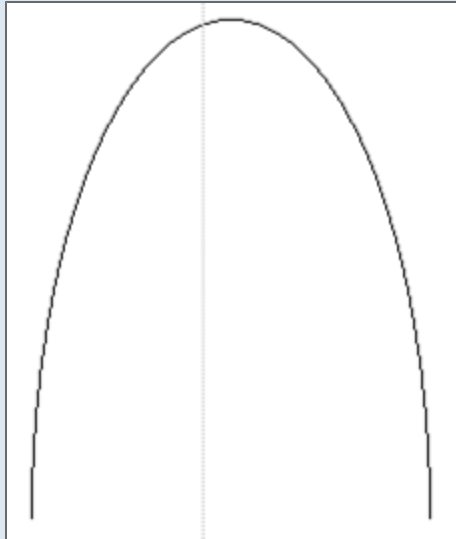
- it defines the coordinates X and Y of the left lower corner, width and height of the circumscribed rectangle, begin and end angle of from the initial angle.

```
<variable>.DrawArc(<x>, <y>, <width_of_circumscribed_rectangle>, <height_of_circumscribed_rectangle>, <begin_angle>, <end_angle>, <pen_style>);
```

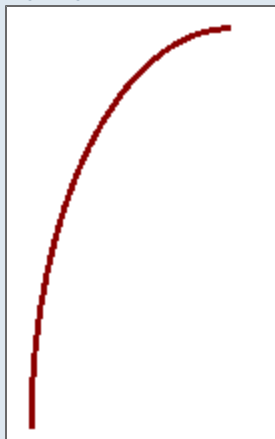
- it defines the coordinates X and Y of the left lower corner, width and height of the circumscribed rectangle, begin and end angle of from the initial angle and the pen style.

Example

`G.DrawArc(100, 100, 200, 500, 0, 180);` - the left lower corner is 100, 100, the width is 200 and the height is 500. The begin angle calculated anticlockwise from the initial angle is 0°, the end angle is 180°. The default style is used.



G.DrawArc(300, 300, 200, 400, 90, 90, Pen); - the left lower corner is 300, 300, the width is 200 and the height is 400. The begin angle calculated anticlockwise from the initial angle is 90°, the end angle is 90°. The predefined Pen style is used.



[Load the example: DrawArc.cls](#)

DrawCircle, FillCircle

DrawCircle

The command draws a circle.

Syntax:

```
<variable>.DrawCircle(<x> , <y>, <diameter>, <pen>);
```

- it draws circle with X, Y center, defined diameter and predefined pen style.

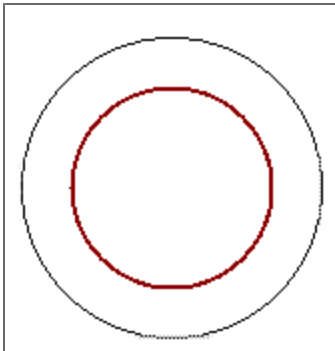
```
<variable>.DrawCircle(<x> , <y> , <diameter>);
```

- it draws circle with X, Y center, defined diameter and default pen style.

Example

```
G.DrawCircle(150, 150, 100, Pen); - the circle has center 150, 150, the diameter is 100 and predefined style Pen is used.
```

```
G.DrawCircle(150, 150, 150); - the circle has center 150, 150, the diameter is 150 and default pen style is used.
```



[Load the example: DrawCircle.cls](#)

FillCircle

Fill the drawn circle by colour or brush. The brush has to be predefined before it is used.

Syntax:

```
<variable>.FillCircle(<x> , <y> , <diameter> , Color.<colour_name>);
```

- it fills the drawn circle with given center and diameter using solid brush with predefined colour.

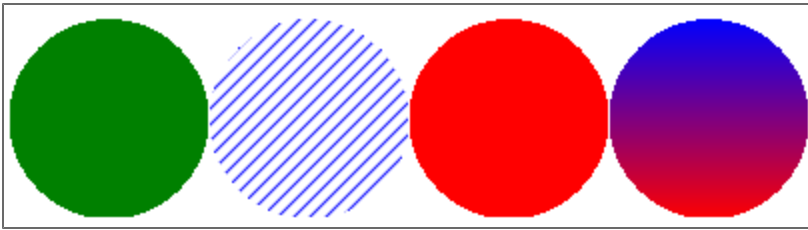
```
<variable>.FillCircle(<x> , <y> , <diameter> , <predefined_brush>);
```

- it fills the drawn circle with given center and diameter using predefined brush.

Example

```
G.FillCircle(0, 0, 100, Color.Green); - it fills a circle with center at coordinates 0, 0 and with diameter 100. The brush colour is green.
```

`G.FillCircle(100, 0, 100, HatchBrush);` - it fills a circle with center at coordinates 100, 0 and with diameter 100. The HatchBrush is used for the filling.



[Load the example: FillCircle.cls](#)

DrawEllipse, FillEllipse

DrawEllipse

This command draws ellipse with defined parameters.

Syntax:

```
<variable>.DrawEllipse(<x>, <y>, <main_axis_length>, <secondary_axis_length>);
```

- it draws ellipse with X, Y center, main axis and secondary axis are defined, the default pen style is used.

```
<variable>.DrawEllipse(<x>, <y>, <main_axis_length>, <secondary_axis_length>, <pen>);
```

- it draws ellipse with X, Y center, main axis and secondary axis are defined, the predefined pen style is used.

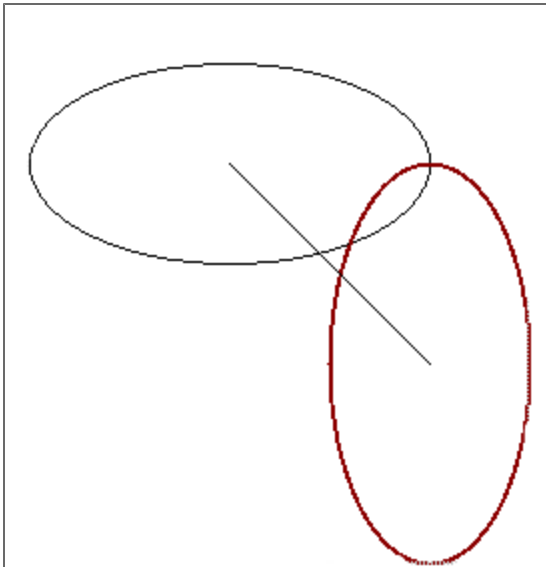
Example

```
G.DrawEllipse(0, 100, 200, 100);
```

- it defines ellipse, center is 0, 100, the main axis is 200 and the secondary is 100. The default pen is used.

```
G.DrawEllipse(100, 0, 100, 200, Pen);
```

- it defines ellipse, center is 100, 0, the main axis is 100 and the secondary is 200. The predefined Pen pen is used.



[Load the example: DrawEllipse.cls](#)

FillEllipse

This command fills the predefined ellipse by a colour or by a defined brush.

Syntax:

```
<Variable>.FillEllipse(<x>, <y>, <main_axis_length>, <secondary_axis_length>, Color.<colour_name>);
```

- it fills an ellipse with the solid brush.

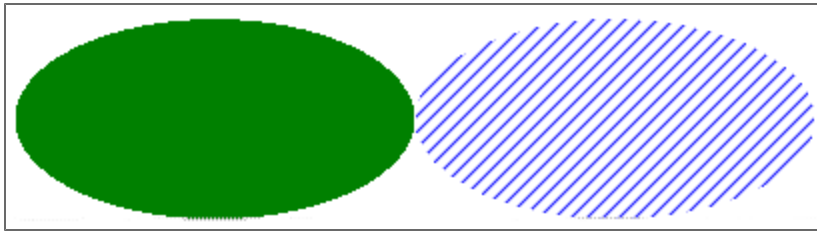
```
<Variable>.FillEllipse(<x>, <y>, <main_axis_length>, <secondary_axis_length>, <brush_object>);
```

- it fills an ellipse with the predefined brush.

Example

```
G.FillEllipse(0, 0, 200, 100, Color.Green); - it fills an ellipse with center [0,0], major axis 200 and minor axis 100. Colour of the brush is green.
```

```
G.FillEllipse(200, 0, 200, 100, HatchBrush); - it fills an ellipse with center [200,0], major axis 200 and minor axis 100. The predefined brush named HatchBrush is used.
```



[Load the example: FillEllipse.cls](#)

DrawText

This command draws a text line. The inserting point is its coordinates. Alignment styles:

Syntax:

```
<Variable>.<text>,<font>,<x>,<y>;
```

- it draws a text with a predefined inserting point and font. The default alignment and colour are used.

```
<Variable>.<text>,<font>,<x>,<y>, ContentAlignment.<alignment>;
```

- it draws a text with a predefined inserting point and font. The default colour and predefined alignment are used.

```
<Variable>.<text>,<font>,<x>,<y>, ContentAlignment.<alignment>, <angle>;
```

- it draws a text with a predefined inserting point and font. The default colour and predefined alignment are used. The specified angle for the text is used.

```
<Variable>.<text>,<font>,<x>,<y>, ContentAlignment.<alignment>, <angle>;
```

- it draws a text with a predefined inserting point and font. The default colour and predefined alignment are used. The specified angle for the text is used.

```
<Variable>.<text>,<font>,<x>,<y>, ContentAlignment.<alignment>, <angle>, Color.<color_name>;
```

- it draws a text with a predefined inserting point, font, colour and alignment. The specified angle for the text is used.

```
<Variable>.<text>,<font>,<x>,<y>, ContentAlignment.<alignment>, Color.<color_name>;
```

- it draws a text with a predefined inserting point, font, colour and alignment.

Example

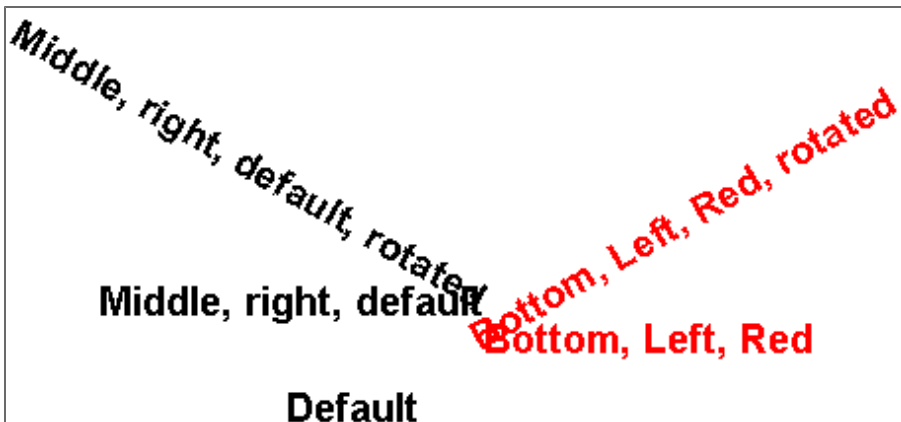
G.DrawText("Default", Font, -50, -35); - it draws a text "Default" with coordinates -50 and -35. The predefined text font Font is used. The default alignment (bottom, left) and colour (black) are used.

G.DrawText("Middle, right, default", Font, 50, 30, ContentAlignment.MiddleRight); - it draws a text "Middle, right, default" with coordinates 50 and 30. The predefined text font Font and alignment to the middle right side are used. The default colour (black) is used.

G.DrawText("Middle, right, default, rotated", Font, 50, 30, ContentAlignment.MiddleRight, 30); - it draws a text "Middle, right, default, rotated" with coordinates 50 and 30. The predefined text font Font and alignment to the middle right side are used. The default colour (black) is used. The text is rotated to 30°.

G.DrawText("Bottom, Left, Red, rotated", Font, 50, 0, ContentAlignment.BottomLeft, -30, Color.Red); - it draws a text "Bottom, Left, Red, rotated" with coordinates 50 and 0. The predefined text font Font and alignment to the bottom left side are used. The colour is set to red and the text is rotated to -30°.

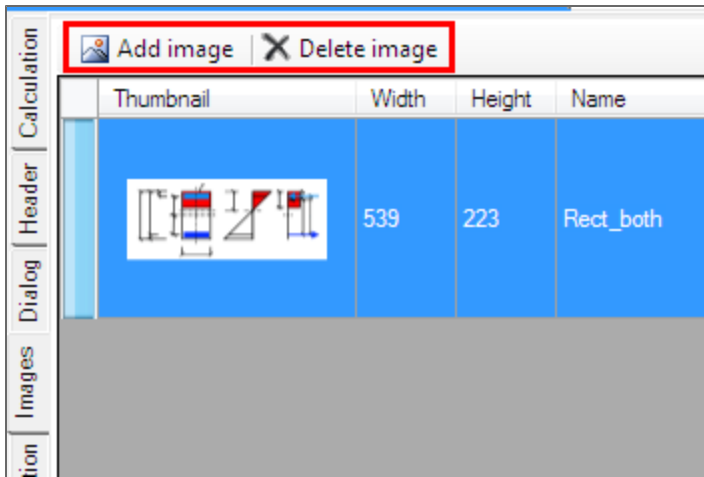
G.DrawText("Bottom, Left, Red", Font, 50, 0, ContentAlignment.BottomLeft, Color.Red); - it draws a text "Bottom, Left, Red" with coordinates 50 and 0. The predefined text font Font and alignment to the bottom left side are used. The colour is set to red.



[Load the example: DrawText.cls](#)

DrawImage

The command draws bitmap which is added to the tab Image in the Builder application.



Syntax:

```
<Variable>.DrawImage(<image_name>,<x>,<y>);
```

- it draws image from the tab Images to the position X, Y (double values) - its inserting point.

```
<Variable>.DrawImage(<image_name>,<x>,<y>,<width>,<height>);
```

- it draws image from the tab Images to the position X, Y with predefined size (width, height).

Example

```
G.DrawImage("SL", 0, 0); - it draws image "SL" at coordinates [0,0].
```

```
G.DrawImage("SL", 0, -100, 300, 100); - it draws image "SL" at coordinates [0,0] with width 300 and height 100.
```



[Load the example: DrawImage.cls](#)

DefaultDimStyle - brush

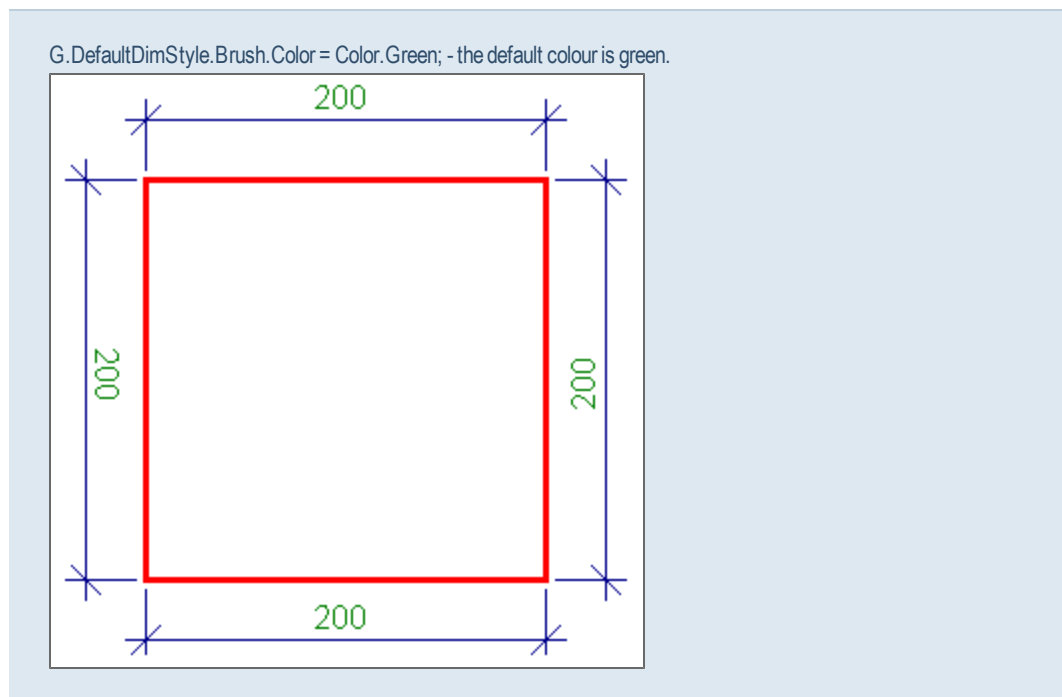
This command defines the colour for the dimension text.

Syntax:

```
<variable>.DefaultDimStyle.Brush.Color = Color.<colour_name>;
```

- it defines the default text colour.

Example



DefaultDimStyle - font, text format, scale

The group of commands for default dimension text style definition.

Font

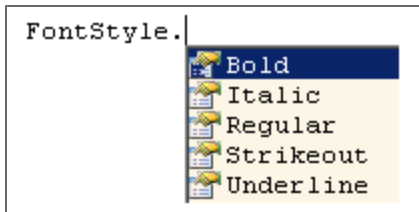
This command defines the dimension text font (size, thickness, type) which will be used for all dimensions.

There are two ways how to define the font:

1. Define it by variable - the variable is more clear and it may be reused in more graphics objects
2. Define it directly - this option is faster, but it is defined only for this one use case

More about fonts on web: <http://msdn.microsoft.com/en-us/library/system.drawing.font.aspx>

Possible font styles are: Bolt, Italic, Regular, Strikeout and Underline.



Syntax:

The variable definition:

```
object <font_variable> = new Font("<font_name>", <font_size>, <font_style>);
```

- it defines the font variable, the font type, the font size and the font style

```
<variable>.DefaultDimStyle.Font = <font_variable>;
```

- it uses the font variable in the dimension style

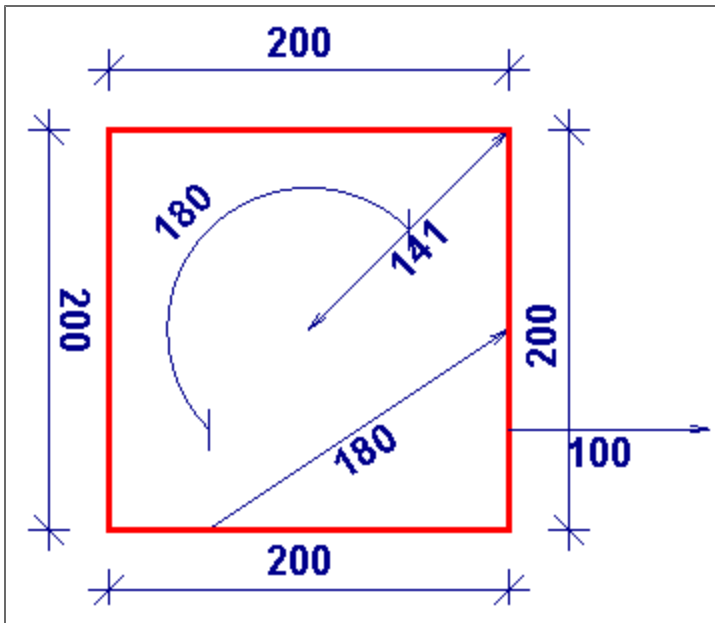
Direct definition:

```
<variable>.DefaultDimStyle.Font = new Font("<font_name>", <font_size>, <font_style>);
```

- it defines only the font type, the font size and the font style

Example

```
object DimFont = new Font("Arial", 15, FontStyle.Bold);  
G.DefaultDimStyle.Font = DimFont;  
- the variable DimFont contains the definition for all dimension texts - font Arial, size 15 px, bold
```



Format

This command defines the minimum number of digits which will be used in dimension text.

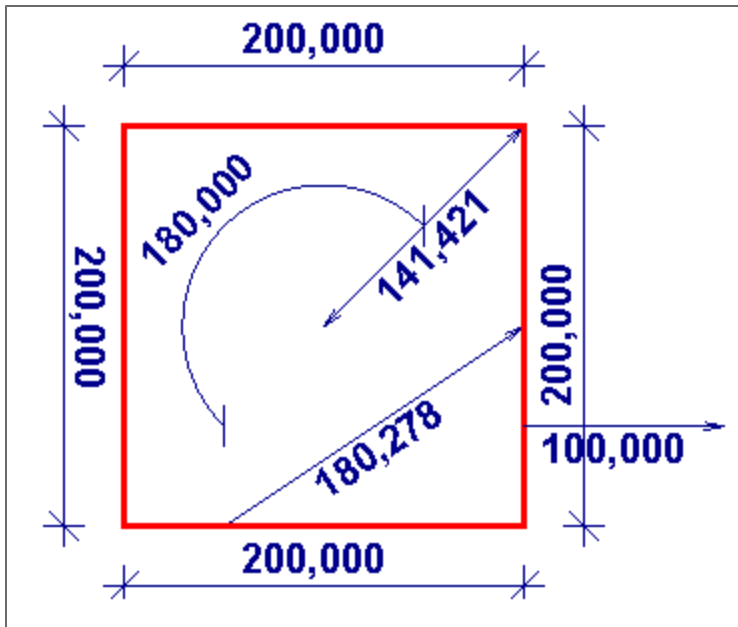
Syntax:

```
<variable>.DefaultDimStyle.Format = "<format>";
```

- it defines the format of texts (0,000, 000,00, ...)

Example

```
G.DefaultDimStyle.Format = "00,000"; - it defines the format with two places before delimiter and three decimal places after delimiter
```



Scale

This command multiplies all measured lengths on dimension lines by the defined scale. It is useful when user need different units on the picture than he has in the code.

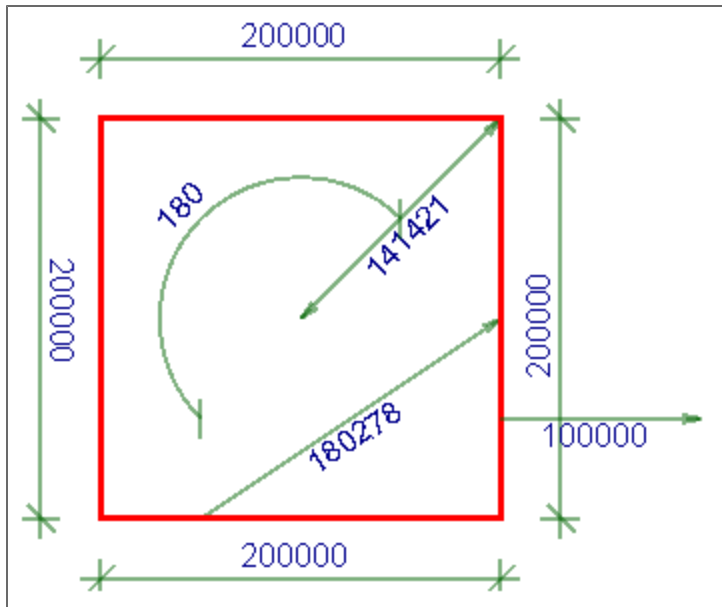
Syntax:

```
<variable>.DefaultDimStyle.Scale = <scale_value>;
```

- it defines the value used for scale (1000, 0,001, ...)

Example

```
G.DefaultDimStyle.Scale = 103; - all dimensions will display the number multiplied by 1000.
```

[Load the example: DefaultDimStyle.cls](#)

DefaultDimStyle - line, end mark, plotline

The group of commands for default dimension style definition.

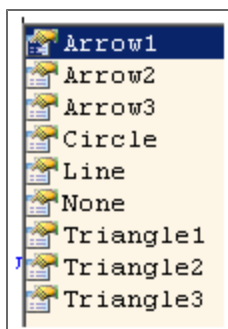
CaptionArrowShape

This command defines the dimension end mark for radius and diameter.

Syntax:

```
<variable>.DefaultDimStyle.CaptionArrowShape = ArrowShape.<arrow_type>;
```

- the arrow types are displayed on the picture:



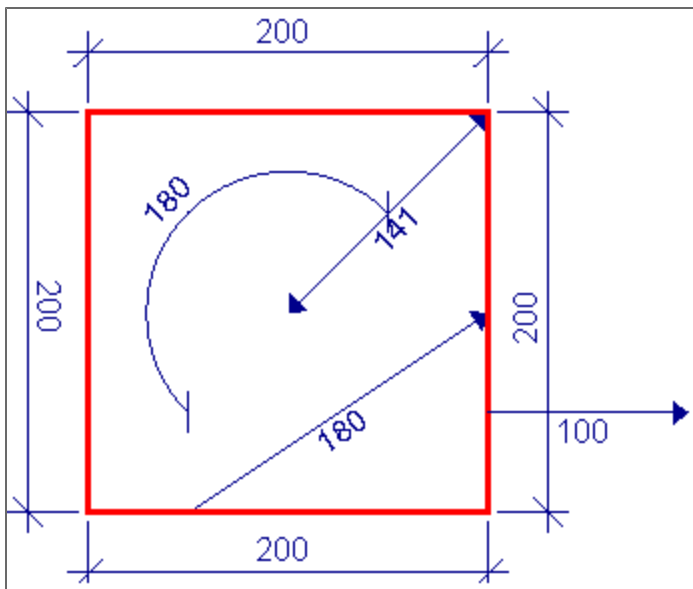
Arrow types:

- **Arrow1** - Blank arrow with angle 20°
- **Arrow2** - Blank arrow with angle 45°

- **Arrow3** - Blank arrow with angle 90°
- **Circle** - Filled circle
- **Line** - dimension line
- **None** - no mark
- **Triangle1** - Filled arrow with angle 20°
- **Triangle2** - Filled arrow with angle 45°
- **Triangle3** - Filled arrow with angle 90°

Example

G.DefaultDimStyle.CaptionArrowShape = ArrowShape.Triangle3; - this command defines the end mark as filled arrow with angle 90°.



CaptionArrowSize

This command defines the size of the end mark for the radius and diameter.

Syntax:

```
<variable>.DefaultDimStyle.CaptionArrowSize = <end_mark_size>;
```

- it defines the end mark size in px

Example

```
G.DefaultDimStyle.CaptionArrowSize = 20; - the end mark size is 20 px.
```

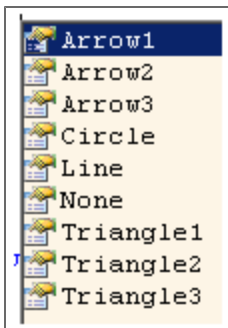
DimArrowShape

This command defines shape of the end mark for line and arc dimensions.

Syntax:

```
<variable>.DefaultDimStyle.DimArrowShape = ArrowShape.<arrow_type>;
```

- it defines the arrow shape according to the picture:

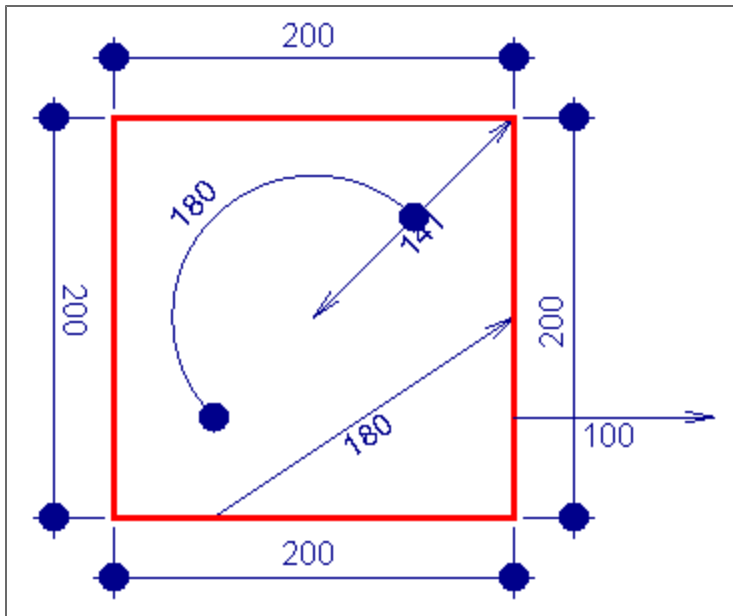


Arrow types:

- **Arrow1** - Blank arrow with angle 20°
- **Arrow2** - Blank arrow with angle 45°
- **Arrow3** - Blank arrow with angle 90°
- **Circle** - Filled circle
- **Line** - dimension line
- **None** - no mark
- **Triangle1** - Filled arrow with angle 20°
- **Triangle2** - Filled arrow with angle 45°
- **Triangle3** - Filled arrow with angle 90°

Example

```
G.DefaultDimStyle.DimArrowShape = ArrowShape.Circle; - the end mark shape is filled circle.
```



DimArrowSize

This command defines the size of the end mark for line and arc dimensions.

Syntax:

```
<variable>.DefaultDimStyle.DimArrowSize = <end_mark_size>;
```

- it defines the arrow size in px

Example

```
G.DefaultDimStyle.DimArrowSize = 15; - the end mark size is 15 px.
```

FixedDimLength

This command defines the plotline length.

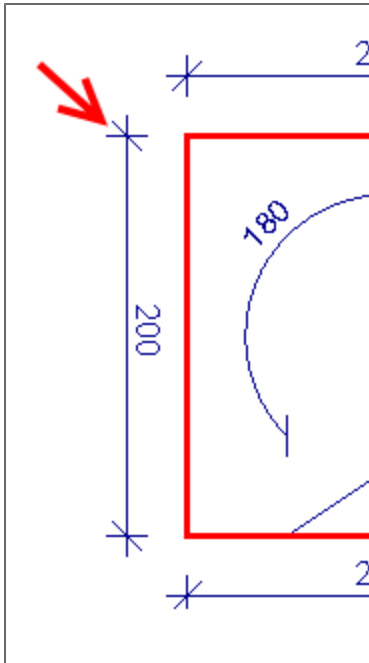
Syntax:

```
<variable>.DefaultDimStyle.FixedDimLength = <plotline_length>;
```

- it defines the plotline length in px

Example

```
G.DefaultDimStyle.FixedDimLength = 10; - plotline length is 10 px.
```



LineOffset

This command defines the offset of the plotline from the object. The size is defined in px.

Syntax:

```
<variable>.DefaultDimStyle.LineOffset = <offset>;
```

- it defines the plotline offset from the object in px

Example

```
G.DefaultDimStyle.LineOffset = 50; - the dimension is displayed with offset 50 px from the measured object
```



Overlap

This command defines the length of the plotline beyond the end mark. The size is in px.

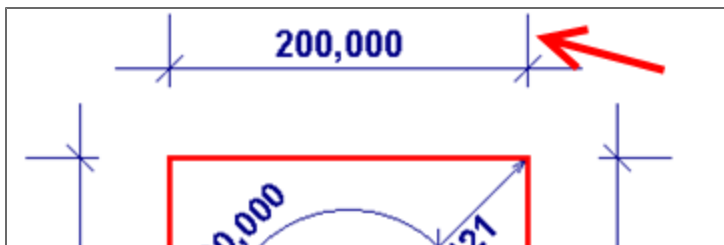
Syntax:

```
<variable>.DefaultDimStyle.Overlap = <length_beyond_mark>;
```

- it defines the length beyond the end mark

Example

```
G.DefaultDimStyle.LineOffset = 30; - the length of the plotline beyond the end mark is 30 px.
```



DefaultDimStyle - pen

Pen

This command defines the dimension line type. It is possible to define the colour and the thickness.

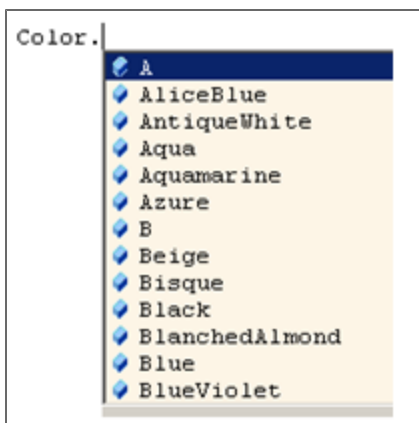
The command supports all standard colours and ARGB format.

There are two ways of Pen definition.

1. Define it by variable - the variable is more clear and it may be reused in more graphics objects
2. Define it directly - this option is faster, but it is defined only for this one use case






See more about pens on web: <http://msdn.microsoft.com/en-us/library/system.drawing.pen.aspx>

Description of colours:



The help in SDF Builder:

Dash styles:

- Solid 
- Dot 
- DashDotDot 
- DashDot 
- Dash 

Syntax:The variable definition:

```
object <pen_variable> = new Pen(Color.<colour_name>, <pen_thickness>);
```

- it defines the pen variable, the pen colour according to the list and pen thickness in px

```
<variable>.DefaultDimStyle.Pen = <pen_variable>;
```

- it uses the pen variable in the dimension style

```
<variable>.DashStyle = DashStyle.<dash_style>;
```

- it defines the dash style for the pen object.

Direct definition:

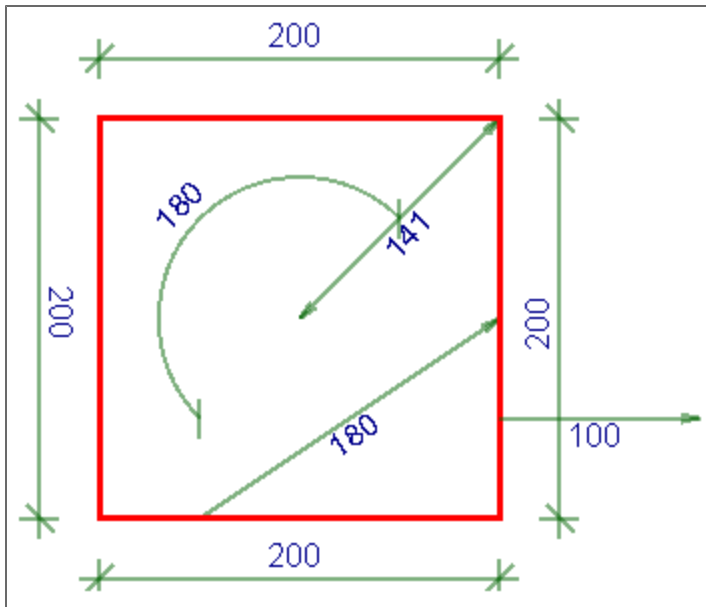
```
<variable>.DefaultDimStyle.Pen = new Pen(Color.<colour_name>, <pen_thickness>);
```

- it defines only the pen colour according to the list and pen thickness in px

Example

```
object DimPen = new Pen(Color.FromArgb(128, Color.DarkGreen), 2);  
G.DefaultDimStyle.Pen = DimPen; - the variable DimPen contains the pen definition - dark green colour from ARGB with  
transparency 128 (means 50%) and thickness 2 px.
```

```
Pen.DashStyle = DashStyle.Dash; - the pen style is set to Dash
```



DefaultPen

This command defines the default colour and thickness of the pen for the dimension lines which doesn't have its own definition. The standard setting is a black colour and thickness 1 px, it may be changed by this option.

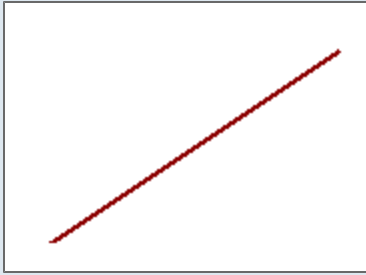
Syntax:

```
<variable>.DefaultPen = new Pen(Color.<colour_name>, <thickness>);
```

- it defines the colour and thickness in px

Example

G.DefaultPen = new Pen(Color.DarkRed, 2); - the default pen is dark red with thickness 2 px



[Load the example: DefaultPen.cls](#)

DrawDim

This command defines the general dimension line between two points.

Syntax:

```
<variable>.DrawDim(<begin_point> , <end_point>);
```

- it defines the begin and end node of the dimension line, the default dimension style is used.

```
<variable>.DrawDim(<begin_point> , <end_point> , <offset>);
```

- it defines the begin and end node of the dimension line and the offset of the dimension line.

```
<variable>.DrawDim(<begin_point> , <end_point> , <dimension_style>);
```

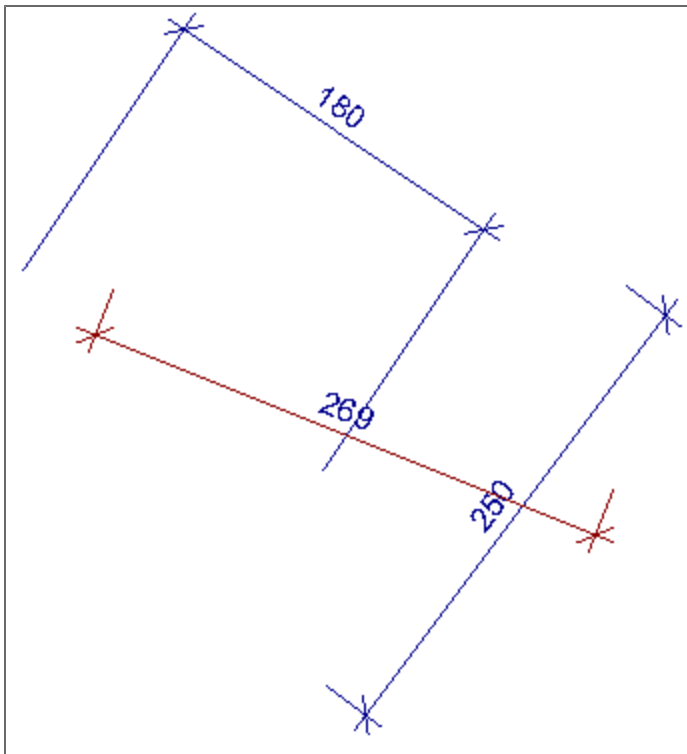
- it defines the begin and end node of the dimension line and the predefined dimension style is used.

Example

G.DrawDim(new PointD(0, 0), new PointD(150, 200)); - it draws the dimension line between 0, 0 and 150, 200. The default dimension style is used.

G.DrawDim(new PointD(0, 100), new PointD(-150, 200), 150); - it draws the dimension line between 0, 100 and -150, 200, the dimension offset is 150 px. The default dimension style is used.

G.DrawDim(new PointD(-100, 200), new PointD(150, 100), DimStyle); - it draws the dimension line between -100, 200 and 150, 100. The predefined dimension style DimStyle is used.



[Load the example: DrawDim.cls](#)

DrawDimX (Y)

This command draws the linear dimension line between two points in X direction (=vertical).

Syntax:

```
<variable>.DrawDimX(<begin_point> , <end_point> , <offset_from_object>);
```

- it defines begin and end point and offset of the dimension from the object. The default dimension style is used.

```
<variable>.DrawDimX(<begin_point> , <end_point>);
```

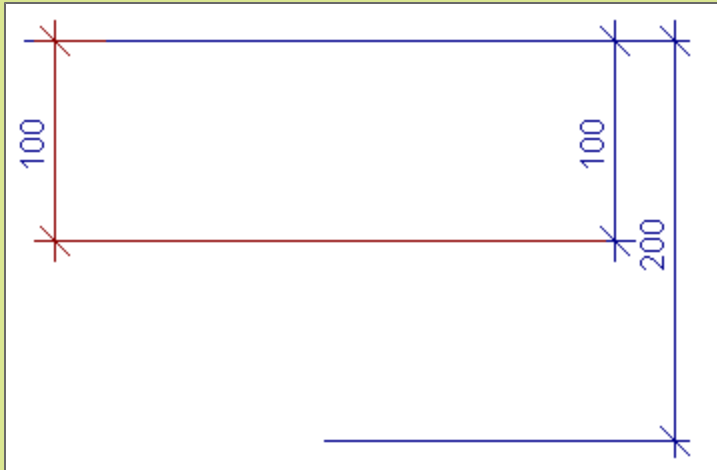
- it defines begin and end point, the default dimension style is used.

```
<variable>.DrawDimX(<begin_point> , <end_point> , <dimension_style>);
```

- it defines begin and end point, the predefined dimension style is used.

The alternative syntax is used for dimension line in Y direction (=horizontal).

e.g. `<variable>.DrawDimY(begin, end);`



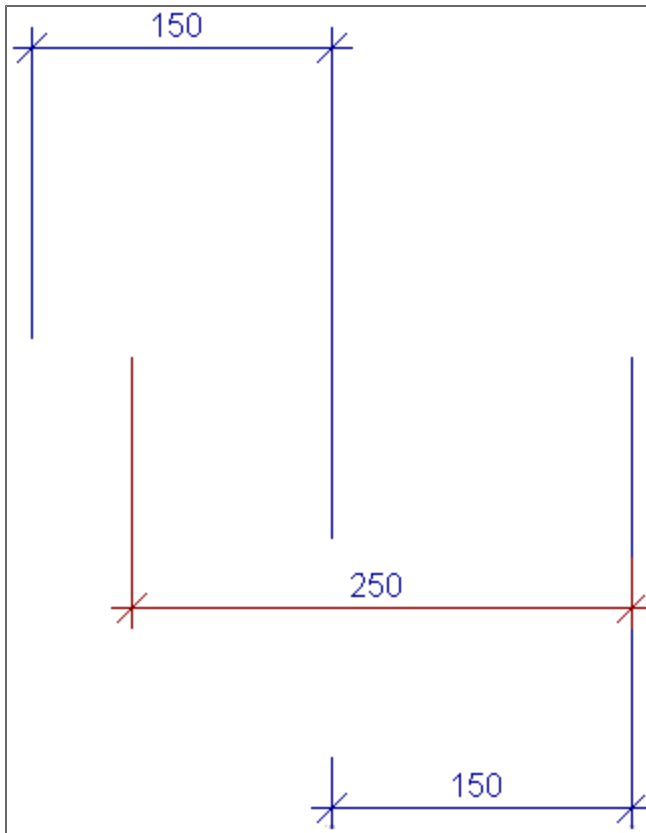
[Load the example: DrawDimY.cls](#)

Example

`G.DrawDimX(new PointD(0, 100), new PointD(-150, 200), 150);` - it defines the linear dimensions between 0, 100 and -150, 200 with offset 150 px from the object. The default dimension style is used.

`G.DrawDimX(new PointD(0, 0), new PointD(150, 200));` - it defines the linear dimensions between 0, 0 and 150, 200 with offset 150 px from the object. The default dimension style is used.

`G.DrawDimX(new PointD(-100, 200), new PointD(150, 100), DimStyle);` - it defines the linear dimensions between -100, 200 and 150, 100 with offset 150 px from the object. The predefined `DimStyle` dimension style is used.



[Load the example: DrawDimX.cls](#)

DrawDimChainX(Y)

This command draws the horizontal or vertical chain dimension line over the array of points.

Array of points - array of points which are measured by the dimension line. The command accepts types Point, PointD, PointF or struct[X, Y].

Syntax:

```
<variable>.DrawChainDimX(<array_of_points>);
```

- it draws a horizontal chain dimension over given set of points. The default dimension style is used.

```
<variable>.DrawChainDimX(<array_of_points>, <dimension_style>);
```

- it draws a horizontal chain dimension over given set of points. The predefined dimension style is used.

The same command may be used for the vertical (Y) dimension line:

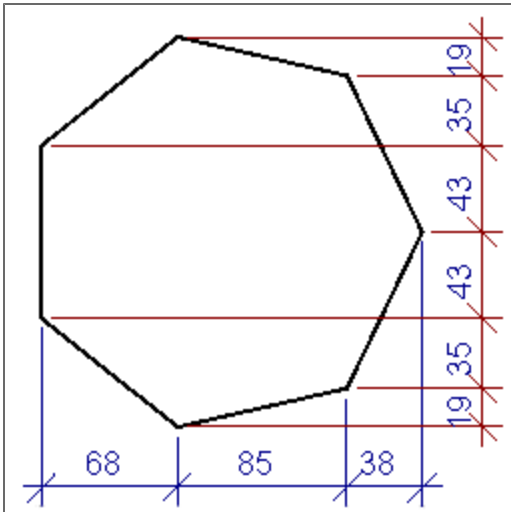
```
<variable>.DrawChainDimY(<array_of_points>);
```

```
<variable>.DrawChainDimY(<array_of_points>, <dimension_style>);
```

Example

G.DrawChainDimX(Pts); - it draws a horizontal dimension over points from array Pts. The default dimension style is used.

G.DrawChainDimX(Pts, DimStyle); - it draws a horizontal dimension over points from array Pts. The DimStyle dimension style is used.



[Load the example: DrawChainDim.cls](#)

DrawAngleDim

This command draws an angle dimension line.

Syntax:

```
<variable>.DrawAngleDim(<center_point>, <begin_point>, <end_point>);
```

- it defines center, begin and end point. The default dimension style is used.

```
<variable>.DrawAngleDim(<center_point>, <begin_point>, <end_point>, <dimension_style>);
```

- it defines center, begin and end point. The predefined dimension style is used.

```
<variable>. DrawAngleDim(<1st_point>, <2nd_point>, <3rd_point>, <4th_point>);
```

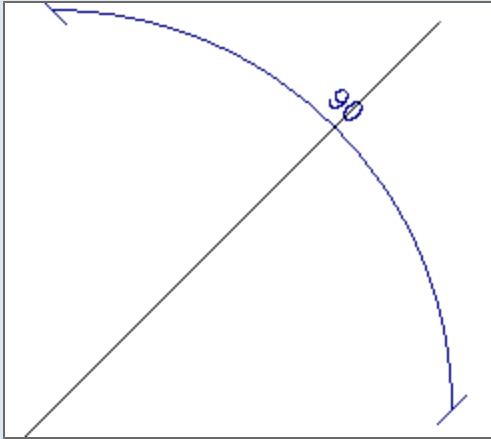
- it defines 4 points for the dimension. The default dimension style is used.

```
<variable>. DrawAngleDim(<1st_point>, <2nd_point>, <3rd_point>, <4th_point>, <dimension_style>);
```

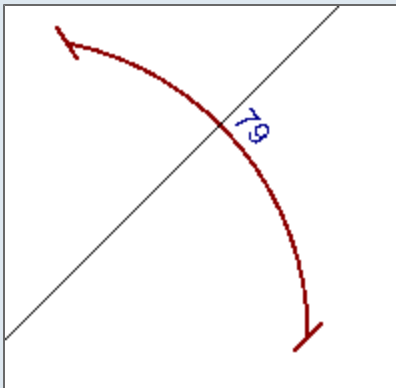
- it defines 4 points for the dimension. The predefined dimension style is used.

Example

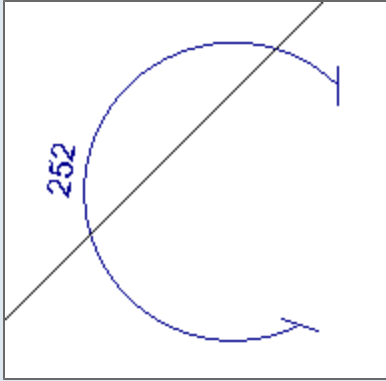
G.DrawAngleDim(new PointD(100, 100), new PointD(300, 100), new PointD(100, 300)); - it draws the dimension line defined by two points (begin, end) which are on measured lines. The begin node (100, 100) is between point 300, 100 and point 100, 300. The default dimension style is used.



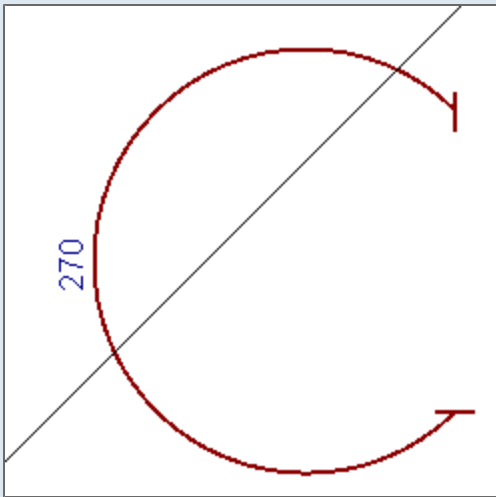
G.DrawAngleDim(new PointD(50, 50), new PointD(200, 50), new PointD(100, 300), DimStyle); - it draws the dimension line which is defined by begin and end node which are on measured lines. The begin node (50, 50) is between point 200, 50 and point 100, 300. The predefined dimension style DimStyle is used.



G.DrawAngleDim(new PointD(200, 50), new PointD(150, 150), new PointD(250, 200), new PointD(300, 250)); - it draws the dimension line which is defined by 2+2 points. The default dimension style is used.



G.DrawAngleDim(new PointD(250, 50), new PointD(150, 150), new PointD(250, 200), new PointD(300, 250), DimStyle); - it draws the dimension which is defined by 2+2 points. The predefined dimension style DimStyle is used.



[Load the example: DrawAngleDim.cls](#)

DrawRadiusDim

This command draws a radius dimension (line with one arrow). The begin and end point are defined as objects.

The accepted type of objects are Point, PointD, PointF and struct[X, Y].

Syntax:

```
<variable>.DrawRadiusDim(<begin_point_object>, <end_point_object>);
```

- it defines the begin and end point object, the default dimension style is used.

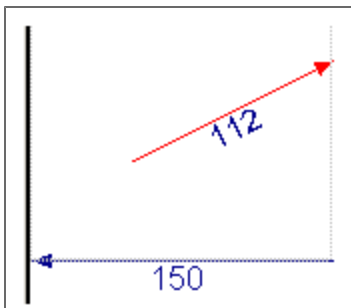
```
<variable>.DrawRadiusDim(<begin_point_object>, <end_point_object>, TDimStyle.<dimension_style>);
```

- it defines the begin and end point object, the predefined dimension style is used.

Example

```
G.DrawRadiusDim(new PointD(150, 0), new PointD(0, 0)); - it draws the radius dimension line between 150, 0 and 0, 0.  
The default dimension style is used.
```

```
G.DrawRadiusDim(new PointD(50, 50), new PointD(150, 100), DimStyle); - it draws the radius dimension line between  
50, 50 and 150, 100. The predefined dimension style DimStyle is used
```



[Load the example: DrawRadiusDim.cls](#)

DrawDiameterDim

This command draws a diameter dimension (line with two arrows).

Syntax:

```
<variable>.DrawDiameterDim(<begin_point>, <end_point>);
```

- it defines the begin and end node, the default dimension style is used.

```
<variable>.DrawDiameterDim(<begin_point>, <end_point>, <dimension_style>);
```

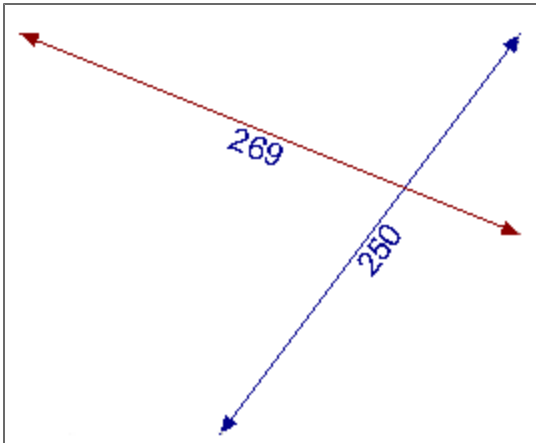
- it defines the begin and end node, the predefined dimension style is used.

Example

```
G.DrawDiameterDim(new PointD(0, 0), new PointD(150, 200)); - it draws the diameter line between 0, 0 and 150, 200.  
The default dimension style is used.
```



```
G.DrawDiameterDim(new PointD(-100, 200), new PointD(150, 100), DimStyle); - it draws the diameter line between -100, 200 and 150, 100. The predefined dimension style DimStyle is used
```



[Load the example: DrawDiameterDim.cls](#)

Size

Commands which may be used for measuring the graphics sizes. The system creates a bounding box around the graphics. The bounding box is displayed as a red rectangle in the layout. The command will return the size value (its double).

Size.Width

This command returns the width of the bounding box.

Syntax:

```
double <variable> = <graphics_variable>.Size.Width;
```

Example

```
double Width = G.Size.Width; - it saves the width to the variable Width.
```

Size.Height

This command returns the height of the bounding box.

Syntax:

```
double <variable> = <graphics_variable>.Size.Height;
```

Example

```
double Height = G.Size.Height; - it saves the height to the variable Height .
```



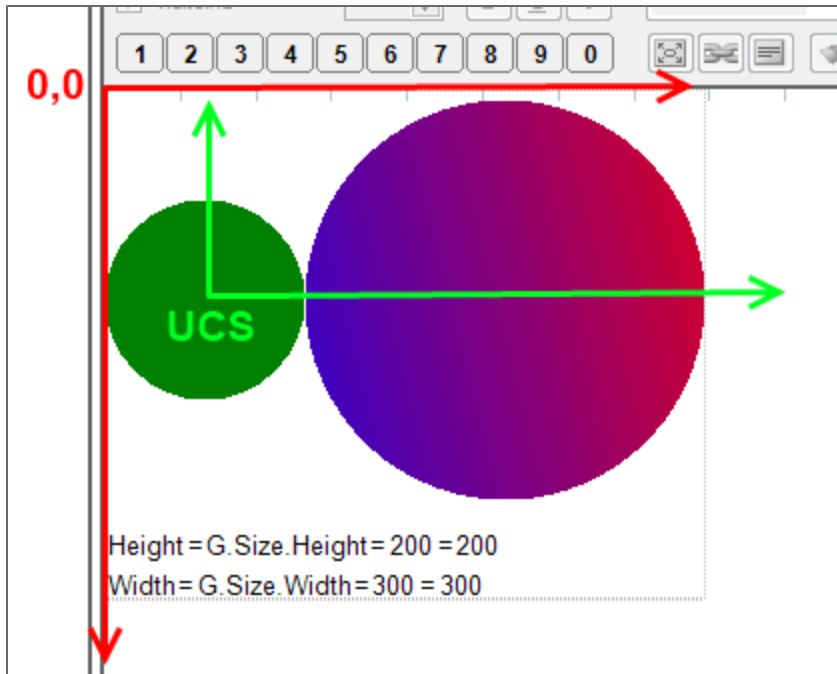
[Load the example: Size.cls](#)

TM

The group of commands which returns another property of graphics object. Property is returned as double value.

TM.dX

The command returns X coordinate of current UCS (of graphics) in the layout. The layout origin is in the top left corner- it has coordinate 0,0.

**Syntax:**

```
double <variable> = <graphics_variable>.TM.dX;
```

Example

```
double dX = G.TM.dX; - the coordinate X of the UCS origin is saved to the variable dX.
```

TM.dY

The command returns Y coordinate of current UCS (of graphics) in the layout. The layout origin is in the top left corner- it has coordinate 0,0.

Syntax:

```
double <variable> = <graphics_variable>.TM.dY;
```

Example

```
double dY = G.TM.dY; - the coordinate X of the UCS origin is saved to the variable dX.
```

TM.ZoomX or TM.ZoomY

This command returns the value of scale in X(Y)-axis which is used in the graphics.

Syntax:

```
double <variable> = <graphics_variable>.TM.ZoomX;
```

```
double <variable> = <graphics_variable>.TM.ZoomY;
```

or

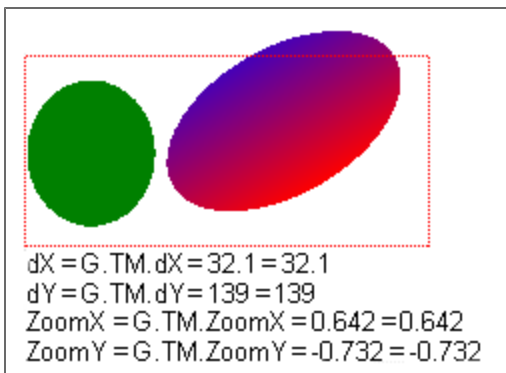
```
double <variable> = <graphics_variable>.ZoomX;
```

```
double <variable> = <graphics_variable>.ZoomY;
```

Example

```
double ZoomX = G.TM.ZoomX; - the value of scale in X-axis which is used in the graphics is saved to the variable ZoomX.
```

```
double ZoomY = G.TM.ZoomY; - the value of scale in Y-axis which is used in the graphics is saved to the variable ZoomY.
```



[Load the example: TM.cls](#)

TR

The TR command may be used for string from the graphics (table in layout, or other objects with texts). This command add the string to the table in the TAB Translations, so it may be translated in the standard way. The string is added to the Translations with ID STRING_HASH.

Syntax:

```
....= TR("<string>");
```

- the defined string is added to the translation table

The same syntax may be used in e.g. graphics.

The command `...= TEXT ("...");` will also add the text to the table, but then the string in the table layout is **not formatted** by other table format commands! It must be formatted manually.

Example

```
T[0][0].Value = TR("Column one");
T[0][1].Value = TR("Second column");
```

- strings "Column one" and "Second column" are added to the table.

```

1 object T = new Table(2);
2 T[0][0].Value = "Column one";
3 T[0][1].Value = "Second column";
4 T[1][0].Value = ("this is added on the second row ");
5 T[1][1].Value = "2 * " & a;
6
7 T.Draw();

```

Přeloženo	ID	English (United States)	čeština (Česká republika)
<input checked="" type="checkbox"/>	CALC_NAME	Calculation	
<input checked="" type="checkbox"/>	LAYOUT_0	Layout 0	
<input checked="" type="checkbox"/>	LAYOUT_1	Layout 1	
<input checked="" type="checkbox"/>	LAYOUT_2	Layout 2	
<input checked="" type="checkbox"/>	LAYOUT_3	Layout 3	
<input checked="" type="checkbox"/>	LAYOUT_4	Layout 4	
<input checked="" type="checkbox"/>	LAYOUT_5	Layout 5	
<input type="checkbox"/>	VARIABLE_T		
<input type="checkbox"/>	VARIABLE_a		
<input checked="" type="checkbox"/>	DOCUMENT	http://sciadesignforms.com/	

```

1 object T = new Table(2);
2 T[0][0].Value = TR("Column one");
3 T[0][1].Value = TR("Second column");
4 T[1][0].Value = ("this is added on the second row ");
5 T[1][1].Value = "2 * " & a;
6
7 T.Draw();

```

Přeloženo	ID	English (United States)	čeština (Česká republika)
<input checked="" type="checkbox"/>	CALC_NAME	Calculation	
<input checked="" type="checkbox"/>	LAYOUT_0	Layout 0	
<input checked="" type="checkbox"/>	LAYOUT_1	Layout 1	
<input checked="" type="checkbox"/>	LAYOUT_2	Layout 2	
<input checked="" type="checkbox"/>	LAYOUT_3	Layout 3	
<input checked="" type="checkbox"/>	LAYOUT_4	Layout 4	
<input checked="" type="checkbox"/>	LAYOUT_5	Layout 5	
<input type="checkbox"/>	VARIABLE_T		
<input type="checkbox"/>	VARIABLE_a		
<input checked="" type="checkbox"/>	DOCUMENT	http://sciadesignforms.com/	
<input checked="" type="checkbox"/>	STRING_1B509	"Column one"	
<input checked="" type="checkbox"/>	STRING_72308	"Second column"	

[Load the example: Table_layout_TR.cls](#)

Variable types

Variable types in SDF:

- **double** a number that ranges in value from $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$; the precision is 15-16 decimal places;
- **string** text e.g.: "Sample text";
- **bool** a Boolean (logical) variable (takes up values TRUE / FALSE);
- **object** a structured variable (e.g. Point [X, Y], force [N, Vy, Vz, Mx, My, Mz], etc.) or object (external CLC, graphics, graphs).

A structured variable can be defined using a special constructor:

```
object S1 = new Structure();
```

- a structured variable named 'S1' is defined.

Double String Boolean Structured				
ID	Description	Symbol	Type	
		S1	Struct	

- **double[]** an array of numerical variables;
- **string[]** an array of text variables;
- **bool[]** an array of Boolean variables;
- **object[]** an array of objects, structured variables or arrays .

If you use the older syntax and you type `struct[]` or `array[]`, the command is automatically converted to `object []`.

```
double A;
string B;
bool C;
object D;
object E;
double[] F;
string[] G;
bool[] H;
object[] I;
```

The SDF BUILDER can automatically recognize the variable type from its use in the code. If this is not possible, the type must be defined manually - a message will be displayed in the layout window and the user can add the variable in the table of variables or using code.

The type of any variable can be manually changed - to do so, simply write the required type in front of the variable in the code. The variable is then re-declared in the table of variables and the original one must be deleted manually. The function [Purge calculation](#) can also be called for the purpose.

[Load an example: variable_types.cls](#)

How to find out the variable type from the code? Use the command:

TEXT(<variable>.GetType().ToString()); - the variable type is printed in the layout as text.

System.String

System.Double

Type Double

Variables of the double type are numerical; these often serve as input data for calculations, as criteria for conditions, or as containers for intermediate or final results.

Syntax:

```
double <variable>;
```

- this script creates a variable with the name <variable> in the table of variables on tab Double; a zero value is assigned to <variable>.

Example

```
double A;
```

The SDF USER application does not distinguish between dot and coma as decimal separators. The value 123.456 is the same as the value 123,456.

The column "Value" cannot contain any text. The only exception is the exponent, for example: "1e6" is the same as "1000000"

All variables are automatically inserted as type Double. The user can change the type manually, or covert the type through the code by using the appropriate command (see previous chapter).

ID	Popis	Symbol	Hodnota	Jednotky	Přesnost
	Design wind load on the wall	q_{Ewd}	0.5	kN/m ²	2
	Wall thickness	t	300	mm	2
	Wall height	h	3	m	2
	Length of wall loaded by axial force	b	1	m	2
		t_{min}	96	mm	2
	Coefficient of construction block type	k	0.1		2

Variables in cells coloured in green should be manually defined (in the dialogue or T table of variables); variables in white cells are defined or calculated in the code.

Type String

Variables of the string type take up text values, and can be used as headlines, additional explanations or descriptions. Strings may be built from text and number characters, punctuation marks and other operators.

Syntax:

```
string <variable>;
```

- this script creates a variable with the name <variable> in the table of variables on the tab 'String.'

Example

```
string B;
```

String variables can be edited in the SDF USER application, if these are included in the Dialogue, even if these have been assigned a value (text) in the code. String variables allow for the user to add additional user-defined texts in predefined locations in the layout.

ID	Popis	Symbol	Hodnota
	Userdefined headline	caption	uživatelský nadpis
	National code	NA	ČSN EN 1996-1-1, §6.2

Any text can be assigned as a value for a string.

Although Scia Design Forms allows for the translation of forms to any other language (by using the last vertical tab 'Translations' in the BUILDER application, see Chapter 'Translations'), string variables are not included in the automatic translation engine. There is a way around this though. When a string variable is inserted in the dialogue, the end-user is able to change the text in the string in the same way as double variables are assigned numerical values. The end-user can then manually translate the text, or even better, the translations can be added to the .DEFAULT file in the same way national annex specific parameters are taken into account in the USER application.

IF the string variable is used in a TEXT command, then it must be excluded from translation by the checkbox on the Translations tab.

<input type="checkbox"/>	TEXT_000590	NA
<input type="checkbox"/>	TEXT_000066	UserDefined

ID	Popis	Symbol	Hodnota
	Userdefined headline	caption	uživatelský nadpis
	National code	NA	ČSN EN 1996-1-1, §6.2

Strings are always defined manually, either in the Table of variables or in the code.

Type Boolean

Boolean variables can only assume the values TRUE or FALSE.

Syntax:

```
bool <variable>;
```

- this syntax creates a variable with the name <variable> on the Boolean tab in the table of variables, and sets it to TRUE.

Example

```
bool C;
```

The boolean variables are defined by checkboxes in the Table of variables or in the dialogue in both the BUILDER and USER applications.

- Checked = TRUE
- Unchecked = FALSE

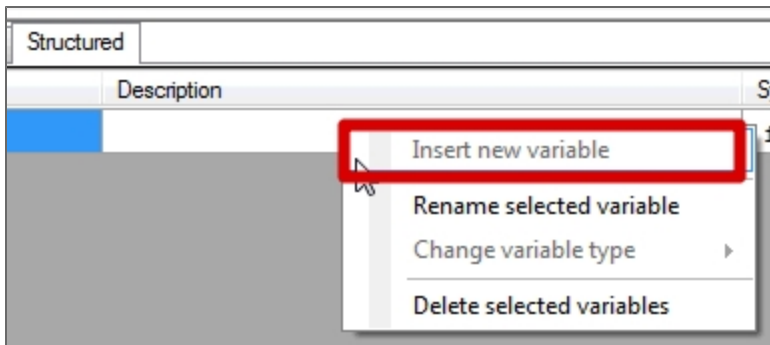
Double			
String			
Boolean			
Structured			
ID	Popis	Symbol	Hodnota
	Vertical gaps are filled by mortar	B1	<input checked="" type="checkbox"/>
	Print headline	Headline	<input checked="" type="checkbox"/>
	Print userdefined headline	PrintCaption	<input type="checkbox"/>

It is important to use the correct syntax for conditions with boolean variables (simple or chained) ([see more about conditions here](#)).

Type Structured

A structured variable is a container that stores a set of variables that do not have to be of the same type. A container can contain a numerical variable next to a boolean one, or it can, for example, store several numerical variables, each of different unit. A structured variable is declared using the same syntax as an object.

See more about [structured](#).



A special constructor `new Structure()`; is used for that purpose.

Syntax:

```
object <variable>;
```

- this notation declares a variable of the Object type on the Structured tab of the table of variables.

Example

```
object D;
```

Nested variables in structured variable

Nested variables can be used in the same way as any other variable.

Syntax:

The syntax for nested variables is:

```
<structured_variable_name> . <nested_variable_name>;
```

Example

```
IntForce.N; - IntForce is the name of a structured variable and N is the name of the nested variable.
```

New Structure()

Syntax:

```
object <variable> = new Structure();
```

- a new empty structured variable is defined

```
object <variable> = new Structure("<name>");
```

- a new empty structured variable with a key name is defined

Example

```
object <variable> = new Structure("X", 0.050, "Y", 0.050, "D", 0.020);
```

- a new structured variable is defined, it contains sub-variables X, Y, D, and these sub-variables are also assigned values X=0.005; Y=0.050; D=0.020

```
object S1 = new Structure();
```

- a structured variable named S1 is defined (in the table the name is listed in the column Symbol).

Double		String		Boolean		Structured	
ID	Description	Symbol	Type				
		S1	Struct				

[Load an example: Structure.cls](#)

<structured_variable>.Add()

The Add function adds a new sub-variable at the end of a structured variable. Exactly what type is the new sub-variable is specified by the function parameters. Generally, a variable of any type can be added.

Syntax:

```
<variable>.Add("<property_name>", <value>);
```

- a new sub-variable with assigned value is added to the variable <variable>.

Example

```
object S1 = new Structure("MyStructure");
```

- object S1 is a structured variable named MyStructure.

```
S1.Add("Item_number", 123);
TEXT("S1.MyProperty = " & S1.MyProperty);
```

- a new sub-variable named Item_number is added to variable S1 and this new sub-variable is assigned a value of 123;
- the second line displays this value in the layout.

```
S2.Add("BooleanProperty", true);
```

- a sub-variable BooleanProperty is added to variable S2 and is set to TRUE.

```
S3.Add("List", new object[]);
S3.List.Add(new Point(2, 0));
TEXT("S3.X = " & S3.X);
TEXT("S3.Y = " & S3.Y);
TEXT("S3.D = " & S3.D);
TEXT("S3.List[0] = " & S3.List[0]);
TEXT("S3.List[0] = " & S3.List[0].X);
```

- a new sub-variable named List is added to variable S3 and is assigned a new array;
- (object) sub-variable List is assigned a C# object of type Point (array with two values 2, 0);
- the three following TEXT commands display in the layout the values of X, Y and D;
- the fourth TEXT command displays in the layout the value of the List array with index 0 - i.e. Point with two sub-variables: X, Y;
- the fifth TEXT command writes number 2 that is stored in the Point object at index 0.

C# function POINT

Point()

Syntax:

```
Point(<X>, <Y>);
```

- a C# function, defines a structured variable that contains two coordinates X and Y. X and Y are two nested variables of the function Point (X, Y) (this function is primarily intended to define a point by means of two coordinates). This function contains Integer variables.

Example

```
How to get the X coordinate from the previous example:
struct[] POINT= Point(0,0);
POINT.X = ...
```

To navigate within the structured variable, the dot convention must be used (it is not possible to use indexes like with arrays).

More information can be found [here](#).

PointF()

Function Point with predefined variables of type float [X, Y]

More information can be found [here](#).

PointD()

Function Point with predefined variables of type double [X, Y] **This is the recommended type of the Point function.**

Predefined structured variables

ReinfBar

A structured variable with predefined properties [X, Y, D]

Forces1D

A structured variable with predefined properties [VN, Vy, Vz, Mx, My, Mz]

Forces2D

A structured variable with predefined properties [nx, ny, nxy, mx, my, vx, vy]

MaterialPoint

A structured variable with predefined properties [eps, sig]

MaterialDiagram

An array of points that form a material stress-strain diagram.

Type Object

This variable type is used for [arrays](#) (vectors, 2D/3D matrices, etc.), structured variables, objects, graphs and for links to external CLC files.

Syntax:

```
object <variable>;
```

- this script creates a variable in the table of variables with the name <variable>; the variable is displayed on the Structured tab with type Object.

Example

```
object D;
```

Loading an external CLC to the variable type object:
object Ext = LoadExternCLC("Extern.cls");

The array (or some other variable) length can be displayed by the using the string .Count or .Length. after the variable name, e.g. A = Ext.Count;

Array

An Array is a group of items of the same type. Variables in this group are called items of an array (array of numbers, array of strings ...), each item has its number (index) and the user can get the item using this number as reference.

Items in an array can be accessed via indexes. The index of the first item in an array is always zero (0).

Syntax:

```
double[] Array
```

- declares a variable of type Array, it is not filled with any data, it is not even an empty array.

```
double[] Array = new double [];
```

- the declared array is initialised, i.e. zeroes are assigned to its items.

This command at the beginning of a script ensures that the array is empty, in other words, this command makes sure that the array does not keep any values from a previous calculation cycle. This syntax guarantees a smooth run of the calculation.

We recommend that the second syntax be used for any new variables, **unless** the variable in question is a variable defined via the **Table input function**.

Example

```
X = MyArray[2]; - the value of MyArray with index 2 is stored to variable "X"
```

```
for(i, 0, 5){ TEXT(MyArray[i]); } - displays the first 6 items from MyArray (index 0 to 5)
```

Declaration of an array in the code

An Array can be declared directly in the source code. The declaration consists of the array type and empty square brackets.

- `double [] Numbers; // declares an array of real (double) numbers named "Numbers."`
- `string [] Texts; // declares an array of strings named "Texts."`
- `object [] Objects; // declares an array of objects, structured variables or arrays named "Objects."`
- `struct [] Points; // declares an array of structured variables named "Points."`
- `array [] Arrays // declares an array of arrays named "Arrays."`

If you use the older syntax (v.3) and you type `struct[]` or `array[]`, the command is automatically converted to `object []`.

An array has a whole set of additional functions. Count, Add, Remove ...

Example

An object variable is named Table

- 1) TEXT(Table[3]); - displays the value of the cell with index equal to 3 (indexes are numbered starting from 0);
- 2) TEXT(Table.Count); - displays in the layout the number of rows that are filled in the table input;
- 3) Table.Remove(7); - removes the value from the cell with index equal to 7 and the following values are moved forward by one cell.

```
object A = new Structure(); - a new structural variable A
A.Add("Array", new object[]); - an array named Array is added to the structured variable from the line above
A.List.Add(new Point(2, 0)); - a C# Point object is added to the array (point is in fact an array of two values: X and Y
TEXT(A.List[0]); - Point is displayed in the layout as it is the item with index = 0
TEXT(A.List[0].X); - value 2 is displayed in the layout as it is the value of sub-variable X
```

The Count command (that returns size) cannot be used with some variables; in such cases, it is replaced by the command Length.

If values are assigned to an array variable using indexing, it is not recommended to skip indexes; this is because, internally, all the skipped indexes must be declared. If a cell is not declared by indexing, it is still created automatically by the application.

[Load an example: TableInputExample.cls](#)

Library CONCRETE TOOLBOX

Concrete toolbox is a set of functionalities which may be used in the source code. Those functionalities are using the power of the Scia Engineer inside the Scia Design Forms.

Intelisense can be used for inserting - it is activated after writing the command and "." or Ctrl+Space (so called dot convention)

Concrete toolbox - Area

Help for methods under group "Area"

CompressedConcrete

Method calculates area of concrete in compression.

Syntax:

```
double CompConcArea = CONCRETE.Area.CompressedConcrete(CharType CharType);
```

where:

- CharType - Char type CONCRETE.CharType.CompressionCharConcrete

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double CompConcArea = CONCRETE.Area.CompressedConcrete(Double CSIndex, Double ForcelIndex);
```

where

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CompressedReinf

Method calculates area of non-prestressed bars in compression

Syntax:

```
double CompReinfArea = CONCRETE.Area.CompressedReinf(CharType CharType);
```

Where:

- CharType - Char type CONCRETE.CharType.CompressionCharReinf

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double CompReinfArea = CONCRETE.Area.CompressedReinf(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TensionConcrete

Method calculates area of concrete in tension.

Syntax:

```
double TenConcArea = CONCRETE.Area.TensionConcrete(CharType CharType);
```

Where:

- CharType - CharType CONCRETE.CharType.TensionCharConcrete

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double TenConcArea = CONCRETE.Area.TensionConcrete(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TensionReinf

Method calculate area of non-prestressed bars in tension.

Syntax:

```
double TenReinfArea = CONCRETE.Area.TensionReinf(CharType CharType);
```

Where:

- CharType - Char type CONCRETE.CharType.TensionCharReinf

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double TenReinfArea = CONCRETE.Area.TensionReinf(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: Area](#)

Concrete toolbox - CenterOfGravity

Help for methods under group "CenterOfGravity"

CompressedConcrete

Method calculates coordinates for center of gravity of compressed concrete.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedConcrete(CharType CharType);
```

Where :

- CharType - Char type CONCRETE.CharType.CompressionCharConcrete

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedConcrete(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Following method calculates coordinates for center of gravity of compressed concrete for reference point different from [0,0].

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedConcrete(CharType CharType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CharType - Char type CONCRETE.CharType.CompressionCharConcrete
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedConcrete(Double CSIndex, Double ForcelIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method `CreateCS` (with appropriate ID, viz "CreateCS") and forces have to be set with method `SetForcesId` (with appropriate ID, viz "SetForces").

Example:

```
//Activation of Concrete toolbox
double IdCss = 0;
CONCRETE.CreateCS(IdCSS, IO.CS.Component.Shape.Point, IO.CONCRETE.EC.Diagram.Point, IO.Beam.Reinforcement.Bar, IO.REINF.EC.Diagram.Point);
CONCRETE.SetForcesId(0, double N, double Vy, double Vz, double Mx, double My, double Mz); //CenterOfGravity
object Pointa = CONCRETE.CenterOfGravity.CompressedConcret (CONCRETE.CharType.CompressionCharConcrete);
object b = CONCRETE.CenterOfGravity.CompressedConcrete(0, 0);
object c = CONCRETE.CenterOfGravity.CompressedConcrete(CONCRETE.CharType.CompressionCharConcrete, 1, 1, 0);
object d = CONCRETE.CenterOfGravity.CompressedConcrete(0, 0, 1, 1, 0);
```

CompressedReinf

Method calculates coordinates for center of gravity of compressed reinforcement

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedReinf(CharType CharType);
```

Where :

- CharType - Char type CONCRETE.CharType.CompressionCharReinf

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedReinf(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method `CreateCS` (with appropriate ID, viz "CreateCS") and forces have to be set with method `SetForcesId` (with appropriate ID, viz "SetForces").

Following method calculates coordinates for center of gravity of compressed reinforcement for reference point different from [0,0].

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedReinf(CharType CharType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CharType - Char type CONCRETE.CharType.CompressionCharReinf
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.CompressedReinf(Double CSIndex, Double ForcelIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Example:

```
//Activation of Concrete toolbox
double IdCss = 0;
CONCRETE.CreateCS(IdCSS, IO.CS.Component.Shape.Point, IO.CONCRETE.EC.Diagram.Point, IO.Beam.Reinforcement.Bar, IO.REINF.EC.Diagram.Point);
CONCRETE.SetForcesId(0, double N, double Vy, double Vz, double Mx, double My, double Mz); //CenterOfGravity
object e = CONCRETE.CenterOfGravity.CompressedReinf(CONCRETE.CharType.CompressionCharReinf);
object f = CONCRETE.CenterOfGravity.CompressedReinf(0, 0);
object g = CONCRETE.CenterOfGravity.CompressedReinf(CONCRETE.CharType.CompressionCharReinf, 1, 1, 0);
object h = CONCRETE.CenterOfGravity.CompressedReinf(0, 0, 1, 1, 0);
```


TensionConcrete

Method returns coordinates for center of gravity of tension concrete.

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionConcrete(CharType CharType);
```

Where:

- CharType - Char type CONCRETE.CharType.TensionCharConcrete

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionConcrete(Double CSIndex, Double ForcelIndex, CharType CharType);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation
- CharType - Char type CONCRETE.CharType.TensionCharConcrete

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Following method returns coordinates for center of gravity of tension concrete for reference point different from [0,0].

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionConcrete(CharType CharType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where :

- CharType - Char type CONCRETE.CharType.TensionCharConcrete
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionConcrete(Double CSIndex, Double ForcelIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Example:

```
//Activation of Concrete toolbox
double IdCss = 0;
CONCRETE.CreateCS(IdCSS, IO.CS.Component.Shape.Point, IO.CONCRETE.EC.Diagram.Point, IO.Beam.Reinforcement.Bar, IO.REINF.EC.Diagram.Point);
CONCRETE.SetForcesId(0, double N, double Vy, double Vz, double Mx, double My, double Mz); //CenterOfGravity
object i = CONCRETE.CenterOfGravity.TensionConcrete(CONCRETE.CharType.TensionCharConcrete);
object j = CONCRETE.CenterOfGravity.TensionConcrete(0, 0, CONCRETE.CharType.TensionCharConcrete);
object k = CONCRETE.CenterOfGravity.TensionConcrete(CONCRETE.CharType.TensionCharConcrete, 1, 1, 0);
object l = CONCRETE.CenterOfGravity.TensionConcrete(0, 0, 1, 1, 0);
```

TensionReinf

Method returns coordinates for center of gravity of tension reinforcement

Syntax:

```
object point =CONCRETE.CenterOfGravity.TensionReinf(CharType CharType);
```

Where:

- CharType - Char type CONCRETE.CharType.TensionCharReinf

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionReinf(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Following method returns coordinates for center of gravity of tension reinforcement for reference point different from [0,0].

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionReinf(CharType CharType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where :

- CharType - Char type CONCRETE.CharType.TensionCharReinf
- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.CenterOfGravity.TensionReinf(Double CSIndex, Double ForcelIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

- CSCH_dy_Center - is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Example:

```
//Activation of Concrete toolbox
double IdCss = 0;
CONCRETE.CreateCS(IdCSS, IO.CS.Component.Shape.Point, IO.CONCRETE.EC.Diagram.Point, IO.Beam.Reinforcement.Bar, IO.REINF.EC.Diagram.Point);
CONCRETE.SetForcesId(0, double N, double Vy, double Vz, double Mx, double My, double Mz); //CenterOfGravity
object m = CONCRETE.CenterOfGravity.TensionReinf(CONCRETE.CharType.TensionCharReinf);
object n = CONCRETE.CenterOfGravity.TensionReinf(0, 0);
object o = CONCRETE.CenterOfGravity.TensionReinf(CONCRETE.CharType.CompressionCharReinf, 1, 1, 0);
object p = CONCRETE.CenterOfGravity.TensionReinf(0, 0, 1, 1, 0);
```

Outline

Method returns center of gravity - coordinates [0,0]

Syntax:

```
object point = CONCRETE.CenterOfGravity.Outline();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

GetGravityCenterOfPolygon

Method returns coordinates of gravity center of polygon.

Syntax:

```
object Point = CONCRETE.CenterOfGravity.GetGravityCenterOfPolygon(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

[Load an example: CenterOfGravity](#)

Concrete toolbox - CreateCS

Help for methods under group "CreateCS"

CreateCS

Initialize cross-section in ConcreteToolbox library. Returns 1, if Csx is not created then returns -1

Syntax:

```
double Csx = CONCRETE.CreateCS(ICollection Polygon, ICollection ConcreteDiagram);
```

Where :

- Polygon - Shape of concrete cross section
- ConcreteDiagram - Material diagram of concrete

This syntax is obsolete, please, use following one.

Syntax:

```
double Csx = CONCRETE.CreateCS(Double CSIndex, ICollection Polygon, ICollection ConcreteDiagram);
```

Where:

- CSIndex - Id of cross section
- Polygon - Shape of concrete cross section
- ConcreteDiagram - Material diagram of concrete

Syntax:

```
double Csx = CONCRETE.CreateCS(ICollection Polygon, ICollection ConcreteDiagram, ICollection ReinfBars, ICollection ReinfDiagram);
```

Where:

- Polygon - Shape of concrete cross section
- ConcreteDiagram - Material diagram of concrete
- ReinfBars - Shape of cross section reinforcement bars
- ReinfDiagram - Material diagram of reinforcement

This syntax is obsolete, please, use following one.

Syntax:

```
double Css = CONCRETE.CreateCS(Double CSIndex, IList Polygon, IList ConcreteDiagram, IList ReinfBars, IList ReinfDiagram);
```

Where:

- CSIndex - Id of cross section
- Polygon - Shape of concrete cross section
- ConcreteDiagram - Material diagram of concrete
- ReinfBars- Shape of cross section reinforcement bars
- ReinfDiagram - Material diagram of reinforcement

Method creates reinforced cross-section or part of reinforced cross-section and returns ID of cross-section).

Syntax:

```
double Css = CONCRETE.CreateCS(Double CSIndex, IList ConCss, IList ConStressStrainDiagram, IList ArrayReinfBars, IList ReinfStressStrainDiagram, IList ArrayPointsRegion);
```

Where:

- CSIndex - Id of cross section
- ConCss - Array of concrete polygon with two columns (coordinate y[m] and z[m])
- ConStressStrainDiagram - Array of point of stress-strain diagram of concrete with two columns (coordinate strain[-] and stress[Pa])
- ArrayReinfBars - Array of bars of non-prestressed reinforcement with three columns (coordinate y[m] ,coordinate z[m], diameter of reinforcement [m])
- ReinfStressStrainDiagram - Array of point of stress-strain diagram of all non-prestressed rein. with two columns (coordinate strain[-] and stress[Pa])
- ArrayPointsRegion - Array of region (part of cross-section) for evaluation results with two columns (coordinate y[m] and z [m])

[Load an example: CreateCS](#)

CONCRETE Designer Library

InitialiseDesignAddData

Method initialise design add data which are needed for design. Default values are set for all data.

Syntax:

```
Concrete.Designer.InitialiseDesignAddData(Double desAddDataID, Double CssID);
```

Where:

- desAddDataID - ID for created design add data
- CcssID - ID of cross-section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

Syntax:

```
Concrete.Designer.InitialiseDesignAddData(Double desAddDataID)
```

Where:

- desAddDataID - ID for created design add data

Default values are set for all data, default Ccss with ID=0 is used, this function is obsolete.

Values are computed for default cross-section (CcssID = 0, viz "CreateCS").

SetDiameterAndCover

Method set diameter and cover into design add data.

Syntax:

```
Concrete.Designer.SetDiameterAndCover(Double desAddDataID, Double reinfDiam, Double reinfCover)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData viz "InitialiseDesignAddData"
- reinfDiam - diameter of reinforcement
- reinfCover - cover of reinforcement

SetOffsetOfLayersOfReinforcement

Method set offset between two layers of reinforcement

Syntax:

```
Concrete.Designer.SetOffsetOfLayersOfReinforcement(Double desAddDataID, Double reinfOffset)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- reinfOffset - offset of reinforcements

SetCoefficientForReinforcementAreaForLayer

Method set coefficient which is used for multiplication of reinforcements area in layer

Syntax:

```
Concrete.Designer.SetCoefficientForReinforcementAreaForLayer(Double desAddDataID, Double CoefAsmax)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- CoefAsmax - coefficient

SetIfCanBeUseUserReinforcement

Method set if can be take into account during design also reinforcement which user set.

Syntax:

```
Concrete.Designer.SetIfCanBeUseUserReinforcement(Double desAddDataID, Double Use)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- Use - parameter which marks in user reinforcement should be taken into account during design of reinforcement
 - use 0 if no,
 - use 1 if yes

SetMaximalCompressedPortion

Method set maximal compressed portion for interaction diagram which is used during design of reinforcement.

Syntax:

```
Concrete.Designer.SetMaximalCompressedPortion(Double desAddDataID, Double compPortion)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- compPortion - compressed portion coefficient

SetExtraRotationPoint

Method set rotation point for interaction diagram which is used during design of reinforcement

Syntax:

```
Concrete.Designer.SetExtraRotationPoint(Double desAddDataID ,Double heightRatio,Double strainRatio)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- heightRatio - height ratio
- strainRatio - strain ratio

SetDesignMaterialDiagram

Method set material diagram of reinforcement into design add data

Syntax:

```
Concrete.Designer.SetDesignMaterialDiagram(Double desAddDataID, Collections.IList DiagramList)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- DiagramList - of points of diagram - (strain, stress)

SetDesignUniaxialAddData

Method sets uniaxial edge add data - coordinates of two points in which reinforcement will be designed - for uniaxial design. Output is index of data - desUniaxialAddDataID .

Syntax:

```
double index = Concrete.Designer.SetDesignUniaxialAddData(Double desAddDataID, Collections.IList uniaxialAddEdgeData)
```

Where:

- desAddDataID - ID of design add data created with InitialiseDesignAddData (viz "InitialiseDesignAddData")
- uniaxialAddEdgeData - List of two points (index of edge, y coord., z coord)

SetEdgeAddDataGen

Method sets general edge add data - for selected edge have to be known index and ratio of reinforcement, additionally coordinates for centroid of reinforcement - for general and biaxial design. Output is index of data - desGeneralAddDataID.

Syntax:

```
double desGeneralAddDataID = Concrete.Designer.SetEdgeAddDataGen(Double desAddDataID, Collections.IList generalAddEdgeData)
```

Where:

- desAddDataID - ID of design add data
- generalAddEdgeData - List of points (index, ratio of reinforcement, y coord., z coord)

SetCornerAddData

Method sets general edge add data - for selected edge have to be known index, additionally ratio of reinforcement - for corner design. Output is index of data - desComerAddDataID.

Syntax:

```
double desComerAddDataID = Concrete.Designer.SetComerAddData(Double desAddDataID, Collections.IList generalAddComerData )
```

Where:

- desAddDataID - ID of design add data
- generalAddComerData - List of points (index, ratio of reinforcement)

GetInfoAboutCorner

Method returns info about corner of cross-section. Method returns comer ID, y coordinate, z coordinate.

Syntax:

```
object comer = Concrete.Designer.GetInfoAboutCorner(Double CssID, Double comerIndex)
```

Where:

- CssID - Id of cross-section, default CSS has CssId = 0
- comerIndex - index of comer

GetNumberOfEdgesWithDesignedReinforcement

Method returns number of edges with designed reinforcement.

Syntax:

```
double numEdges = Concrete.Designer.GetNumberOfEdgesWithDesignedReinforcement(Double HashKey)
```

Where:

- HashKey - HashKey obtained from design of reinforcement

GetDesignedReinforcementInfoForEdge

Method returns designed reinforcement for edge. Method returns y coordinate of centroid of reinforcement, z coordinate of centroid of reinforcement

area of reinforcement, if reinforcement is tensioned, diameter of reinforcement.

Syntax:

```
object reinfoEdge = Concrete.Designer.GetDesignedReinforcementInfoForEdge(Double HashKey, Double EdgeIndex)
```

Where:

- HashKey - HashKey obtained from design of reinforcement
- EdgeIndex - index of edge of cross-section

GetNumberOfLayersWithReinforcementForEdges

Method returns number of designed layers of reinforcement for edge of cross-section.

Syntax:

```
double numLayers = Concrete.Designer.GetNumberOfLayersWithReinforcementForEdges(Double HashKey, Double edgeIndex)
```

Where:

- HashKey - HashKey obtained from design of reinforcement
- edgeIndex - index of edge of cross-section

GetDesignedReinforcementInfoForLayerOfEdge

Method returns designed reinforcement for layer of reinforcement for edge. Method returns y coordinate of centroid of reinforcement, z coordinate of centroid of reinforcement area of reinforcement, if reinforcement is tensioned, diameter of reinforcement.

Syntax:

```
object layer = Concrete.Designer.GetDesignedReinforcementInfoForLayerOfEdge(Double, Double, Double)
```

Where:

- HashKey - HashKey obtained from design of reinforcement
- EdgeIndex - index of edge of cross-section
- layerIndex - index of layer

DesignReinforcementUniaxialAroundGeneralAxis

General uniaxial method for design reinforcement for general axis defined by angle. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementUniaxialAroundGeneralAxis(Double desAddDataID, Double N, Double My, Double Mz, Double Alpha, Double desUniaxEdgeAddData )
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- Alpha - Angle of axis
- desUniaxEdgeAddData - Id of uniaxial edge add data (viz "SetDesignUniaxialAddData"), if they are not defined use -1.0

Old function with default design add data which was set by obsolete SetAddData (viz "SetAddData") method.

Syntax obsolete:

```
object design = Concrete.Designer.DesignReinforcementUniaxialAroundGeneralAxis(Double N, Double My, Double Mz, Double Angle)
```

Where:

- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- Alpha - Angle of axis

DesignReinforcementUniaxialAroundYAxis

Uniaxial method for design reinforcement for moment action around y axis of cross-section. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementUniaxialAroundYAxis(Double desAddDataID, Double N, Double My, Double desUniaxEdgeAddData )
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force

- My - Value of bending moment acting around y axis
- desUniaxEdgeAddData - Id of uniaxial edge add data (viz "SetDesignUniaxialAddData"), if they are not defined use -1.0

DesignReinforcementUniaxialAroundZAxis

Uniaxial method for design reinforcement for moment action around z axis of cross-section. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementUniaxialAroundZAxis(Double desAddDataID, Double N, Double Mz, Double desUniaxEdgeAddData )
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- Mz - Value of bending moment acting around z axis
- desUniaxEdgeAddData - Id of uniaxial edge add data (viz "SetDesignUniaxialAddData"), if they are not defined use -1.0

DesignReinforcementGeneralBiaxial

Biaxial design of reinforcement - according to formula from EC $(M_{yrd}/M_{edy})^{biaxcoef} + (M_{zrd}/M_{edz})^{biaxcoef} < 1.0$. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneralBiaxial(Double desAddDataID , Double N, Double My, Double Mz, Double biaxCoef, Double desGeneralEdgeAddData)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- biaxCoef - Value of biaxial coefficient - default 1.4
- desGeneralEdgeAddData - Id of general edge add data (viz "SetEdgeAddDataGen"), if they are not defined use -1.0

Old function with default design add data which was set by obsolete SetAddData (viz "SetAddData") method.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneralBiaxial(Double N, Double My, Double Mz, Double biaxCoef)
```

Where:

- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- biaxCoef - Value of biaxial coefficient - default 1.4

DesignReinforcementGeneral

General design of reinforcement. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneral(Double desAddDataID, Double N, Double My, Double Mz, Double desGeneralEdgeAddData)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- desGeneralEdgeAddData - Id of general edge add data (viz "SetEdgeAddDataGen"), if they are not defined use -1.0

Old function with default design add data which was set by obsolete SetAddData (viz "SetAddData") method.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneral(Double N, Double My, Double Mz)
```

Where:

- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis

DesignReinforcementCorner

Corner design of reinforcement. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementCorner(Double desAddDataID, Double N, Double My, Double Mz, Double desGeneralCornerAddData)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- desGeneralCornerAddData - Id of uniaxial edge add data (viz "SetCornerAddData"), if they are not defined use -1.0

Old function with default design add data which was set by obsolete SetAddData (viz "SetAddData") method.

```
object design = Concrete.Designer.DesignReinforcementCorner(Double N, Double My, Double Mz)
```

Where:

- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis

DesignUniaxialGenLayers

General uniaxial method for design reinforcement in layers for general axis defined by angle. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignUniaxialGenLayers(Double desAddDataID, Double N, Double My, Double Mz, Double Alpha, Double numberOfLayers, Double desUniaxEdgeAddData)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- Alpha - Angle of axis
- numberOfLayers - Number of wanted layers
- desUniaxEdgeAddData - Id of uniaxial edge add data (viz "SetDesignUniaxialAddData"), if they are not defined use -1.0

DesignReinforcementGeneralBiaxialLayers

Biaxial design of reinforcement in layers - according to formula from EC $(M_{yrd}/M_{edy})^{biaxcoef} + (M_{zrd}/M_{edz})^{biaxcoef} < 1.0$. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneralBiaxialLayers(Double desAddDataID, Double N, Double My, Double Mz, Double, Double, Double)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- biaxCoef - Value of biaxial coefficient - default 1.4
- numberOfLayers - Number of wanted layers
- desGeneralCornerAddData - Id of uniaxial edge add data (viz "SetCornerAddData"), if they are not defined use -1.0

DesignReinforcementGeneralLayers

General layer design of reinforcement. Method returns HashKey - identifier of results of design, result - id of error, and msg - enum with error.

Syntax:

```
object design = Concrete.Designer.DesignReinforcementGeneralLayers(Double, Double, Double, Double, Double, Double)
```

Where:

- desAddDataID - ID of design add data
- N - Value of normal force
- My - Value of bending moment acting around y axis
- Mz - Value of bending moment acting around z axis
- numberOfLayers - Number of wanted layers
- desGeneralCornerAddData - Id of uniaxial edge add data (viz "SetCornerAddData"), if they are not defined use -1.0

GetNumberOfEdgesOfWholeCrossSection

Method returns number of edges of whole cross-section.

Syntax:

```
double num = Concrete.Designer.GetNumberOfEdgesOfWholeCrossSection(Double CsslDId)
```

Where:

- CsslD - Id of cross-section

Obsolete version - Method returns number of edges for default cross-section - id = 0

Syntax:

```
double num = Concrete.Designer.GetNumberOfEdgesOfWholeCrossSection
```


GetInfoAboutEdgeOfCrossSection

Method returns info about edge of cross-section. Method returns index of first point of edge, index of second point of edge, index of edge.

Syntax:

```
double edge = Concrete.Designer.GetInfoAboutEdgeOfCrossSection(Double CssID,Double EdgelIndex)
```

Where:

- CssID - Id of cross-section
- EdgelIndex - index of edge

Obsolete version - Method returns info about edge for default cross-section - id = 0

Syntax:

```
double edge = Concrete.Designer.GetInfoAboutEdgeOfCrossSection(Double EdgelIndex)
```

Where:

- EdgelIndex - index of edge

GetIndexesOfEdgesWithDesignedReinforcementFromUniaxialDesign

Method returns indexes of edges with designed reinforcement from uniaxial design. Method returns first edge index, second edge index.

Syntax:

```
object indexes = Concrete.Designer.GetIndexesOfEdgesWithDesignedReinforcementFromUniaxialDesign(Double HashKey)
```

Where:

- HashKey - HashKey obtained form design of reinforcement

CalculateReinfRatioForEdge

Method calculates ratio of reinforcement for edge for acting load.

Syntax:

```
double ratio = Concrete.Designer.CalculateReinfRatioForEdge(Double,Double,Double,Double,Double,Double)
```

Where:

-
- CssID - Id of cross-section
 - edgeIndex - index of edge
 - N - Value of normal force
 - My - Value of bending moment acting around y axis
 - Mz - Value of bending moment acting around z axis
 - fck - strength of concrete

CreatePracticalCalculator

Method creates practical calculator and returns Id of practical calculator - HashKey and error info.

Syntax:

```
object practCalc = Concrete.Designer.CreatePracticalCalculator(Double CssID, Double distance, Double diameter, Double minBarsSpacing, Double cover)
```

Where:

- CssID - Id of cross-section
- distance - perpendicular distance from edge
- diameter - diameter of reinforcement
- minBarsSpacing - minimal spacing between bars
- cover - cover of reinforcement

SetAreasOfReinforcementForEdgeForPracticalCalculator

Method sets amount of reinforcement for edges for practical calculator.

Syntax:

```
Concrete.Designer.SetAreasOfReinforcementForEdgeForPracticalCalculator(Double, Collections.IList)
```

Where:

- keyHash - Id of practical calculator
- reinfArea area of reinforcement

OptReinfDiamOnEdge

Method optimizes diameter of reinforcement and return its value.

Syntax:

```
double diam = Concrete.Designer.OptReinfDiamOnEdge(Double keyHashPractCalc, Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

OptReinfDiamOnEdge1

Method optimizes diameter of reinforcement and return its value.

Syntax:

```
double diam = Concrete.Designer.OptReinfDiamOnEdge1(Double keyHashPractCalc, Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

MaxNumberBarsOnEdge

Method calculates maximal number of bars.

Syntax:

```
double maxNum = Concrete.Designer.MaxNumberBarsOnEdge(Double keyHashPractCalc , Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

NumberBarsOnEdge

Method calculates number of bars for edge

Syntax:

```
double numBars = Concrete.Designer.NumberBarsOnEdge(Double keyHashPractCalc, Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

SpacingReqReinf

Method returns spacing of required reinforcement.

Syntax:

```
double spacinf = Concrete.Designer.SpacingReqReinf(Double keyHashPractCalc,Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

LengthOfLineForDesign

Method calculates length of layer of reinforcement and returns length, start point(y coord, z coord), end point(y coord, z coord) and error info.

Syntax:

```
object lenght = Concrete.Designer.LengthOfLineForDesign(Double keyHashPractCalc,Double edgeIndex)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge

AssignReinfToEdge

Method assign reinforcement bar to edge according to coordinates and returns index of edge and perpendicular distance from edge.

Syntax:

```
object edge = Concrete.Designer.AssignReinfToEdge(Double CssID,Double reinfCoordY ,Doubler einfCoordZ)
```

Where:

- CssID - Id of cross-section
- reinfCoordY - y coordinates of point
- reinfCoordZ - z coordinates of point

ArrayReqReinf

Method creates real reinforcement bars from required area via practical calculator. Method returns list of bars (barindex,y coord, z coord, As and diameter).

Syntax:

```
object listBars = Concrete.Designer.ArrayReqReinf(Double keyHashPractCalc, Double edgeIndex, Double typeArea, Double minNumber)
```

Where:

- keyHashPractCalc - Id of practical calculator which was created with method CreatePracticalCalculator (viz "CreatePracticalCalculator")
- edgeIndex - index of edge
- typeArea - type of area of reinforcement
 - 0 - round - area will be mathematically rounded
 - 1 - ceil - area will be rounded to the nearest integer value
- minNumber - minimal number of bars for edge

SetAddData

Old method for initialization of design add data - obsolete.

Syntax:

```
Concrete.Designer.SetAddData(Double Diameter, Double Cover)
```

Where:

- Diameter - diameter of reinforcement
- Cover - cover of reinforcement

SetDesignMaterial

Old method for set of material diagram of reinforcement - obsolete

Syntax:

```
Concrete.Designer.SetDesignMaterial(Collections.Generic.List Diagram)
```

Where:

- Diagram - List of material points

ClearReinfDiagram

Old method for clearing of material diagram

Syntax:

```
Concrete.Designer.ClearReinfDiagram()
```

Concrete toolbox - CrackingForce

Help for methods under group "CrackingForce"

CrackingForces1

Method returns the cracking forces (Ncr1, Mcry1, Mcrz1), when in the most tensile fibre of concrete cross section is reached inputted tensile strength of the concrete (value dFcteff)

Syntax:

```
object forces = CONCRETE.CrackingForce.CrackingForces1(Double CSIndex, Double ForcelIndex, Double dFcteff);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- dFcteff - Effective tensile strength of concrete, when the crack is created

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CrackingForces2

Method returns the cracking forces (Ncr2, Mcry2, Mcrz2), when in the most compressive fibre of concrete cross section is reached inputted tensile strength of the concrete (value dFcteff)

Syntax:

```
object forces =CONCRETE.CrackingForce.CrackingForces2(Double CSIndex, Double ForcelIndex, Double dFcteff);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- dFcteff - Effective tensile strength of concrete, when the crack is created

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: CrackingForces](#)

Concrete toolbox - General

Help for methods under group "General"

CentreOfPolygon

Function returns centre of gravity (yc[m] and zc[m] coordinate) of the polygon

Syntax:

```
object point = CONCRETE.General.CentreOfPolygon(List<Object> Polygon);
```

Where:

- Polygon - Array of concrete polygon with two columns (coordinate y[m] and z[m])

GetCountFibresConcrete

Function returns number of concrete fibres in actual cross-section

Syntax:

```
double fiberCount = CONCRETE.General.GetCountFibresConcrete();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double fiberCount = CONCRETE.General.GetCountFibresConcrete(Double CSIndex);
```

Function returns number of concrete fibres in cross-section

Where:

- CSIndex - Id of cross section

GetCountFibresReinforcement

Function returns number of non-prestressed bars of reinforcement in actual cross-section

Syntax:

```
double fiberCount = CONCRETE.General.GetCountFibresReinforcement();
```

Values are computed for default cross-section (CsslD = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double fiberCount = CONCRETE.General.GetCountFibresReinforcement(Double CSIndex);
```

Function returns number of non-prestressed bars of reinforcement in cross-section

Where :

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetCssHeight

Function returns dimension of Cssl in direction of y-axis

Syntax:

```
double height = CONCRETE.General.GetCssHeight(Double RotAngle);
```

Where :

- RotAngle - Angle for calculation dimension of cssl from y-axis of CSS in deg

Values are computed for default cross-section (CsslD = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double height = CONCRETE.General.GetCssHeight(Double CSIndex, Double RotAngle);
```


Where:

- CSIndex - Id of cross section
- RotAngle - Angle for calculation dimension of css from y-axis of CSS in deg

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetFibrePositionConcrete

Function returns concrete fibre position (y[m] and z[m] coordinate) for given index

Syntax:

```
object point = CONCRETE.General.GetFibrePositionConcrete(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.General.GetFibrePositionConcrete(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetFibrePositionReinf

Function returns position of non-prestressed bar of reinforcement (y[m] and z[m] coordinate) in cross-section

Syntax:

```
object point = CONCRETE.General.GetFibrePositionReinf(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.General.GetFibrePositionReinf(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetOutline

Function returns outline points of C_{ss} for actual cross-section as structured array with y and z positions

Syntax:

```
object points = CONCRETE.General.GetOutline();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object points = CONCRETE.General.GetOutline(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetReinfBars

Function returns non-prestressed bars of reinforcement in cross-section for actual cross-section as structured array with Area, diameter, y and z positions

Syntax:

```
object bars = CONCRETE.General.GetReinfBars();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object bars = CONCRETE.General.GetReinfBars(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

PointInPolygon

Function returns true if point is in polygon or false if point is out of polygon

Syntax:

```
bool bInside = CONCRETE.General.PointInPolygon(List<Object> Polygon, Double PointCoordY, Double PointCoordZ);
```

Where:

- Polygon - Array of concrete polygon with two columns (coordinate y[m] and z[m])
- PointCoordY - Y coordinate of point
- PointCoordZ - Z coordinate of point

StressFromStrain

Function returns value of stress calculated from defines strain-stress diagram

Syntax:

```
double stress = CONCRETE.General.StressFromStrain(List<Object> StressStrainDaigram, Double Strain);
```

Where:

- StressStrainDiagram- Array of points of stress-strain diagram of the material with two columns (coordinate strain[-] and stress[Pa])
- Strain - Value of strain

[Load an example: General](#)

Concrete toolbox - CharacteristicsCalculator

Help for methods under group "CharacteristicsCalculator"

Characteristics and CharacteristicsCalculator returns same values for same setting!

Following possibilities can be selected from CONCRETE.CharType - type of cross section characteristic

- CompressionCharConcrete - characteristics of compressed part of concrete cross section
- TensionCharConcrete - characteristics of tensioned part of concrete cross section
- WholeCharConcrete - characteristics of concrete cross section
- CompressionCharReinf - characteristics of compressed reinforcement
- TensionCharReinf - characteristics of tensioned reinforcement
- WholeCharReinf - characteristics of reinforcement
- CssCharacteristicTransfEquilibrium
- CssCharacteristicTransf
- AxialStiff - characteristics for axial stiffness
- BendingStiffY - characteristics for bending stiffness according to y axis
- BendingStiffZ - characteristics for bending stiffness according to z axis
- BendingStiff - characteristics for bending stiffness
- BendingStiffGen - characteristics for bending stiffness

Following possibilities can be selected from CONCRETE.StiffnessType - type of stiffness

- No - not defined
- Linear - linear stiffness
- Uncracked - stiffness of uncracked cross-section
- Cracked - stiffness of cracked cross-section

Characteristics

Function returns Css characteristics

Syntax:

```
object chars = CONCRETE.CharacteristicsCalculator.Characteristics(CharType CharType, StiffnessType StiffnessType);
```

Where:

- CharType - any CharType from CONCRETE.CharType
- StiffnessType - any StiffnessType from CONCRETE.StiffnessType

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object chars = CONCRETE.CharacteristicsCalculator.Characteristics(Double CSIndex, Double ForceIndex, CharType CharType, StiffnessType StiffnessType);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- CharType - any CharType from CONCRETE.CharType
- StiffnessType - any StiffnessType from CONCRETE.StiffnessType

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Syntax:

```
object chars = CONCRETE.CharacteristicsCalculator.Characteristics(CharType CharType, StiffnessType StiffnessType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CharType - Any CharType from CONCRETE.CharType
- StiffnessType - Any StiffnessType from CONCRETE.StiffnessType
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object chars = CONCRETE.CharacteristicsCalculator.Characteristics(Double CSIndex, Double ForceIndex, CharType CharType, StiffnessType StiffnessType, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- CharType - Any CharType from CONCRETE.CharType
- StiffnessType - Any StiffnessType from CONCRETE.StiffnessType
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: Characteristics](#)

Library_CONCRETE_IntDiagram

SetIntDiagramAddData

Method sets basic diagram add data - smoothing of diagram

Syntax:

```
Concrete.IntDiagram.SetIntDiagramAddData(Double IntDiagAddDataID, Double numberOfPoints, Double numberOfVerticalDiagram)
```

Where:

- IntDiagAddDataID - Id of diagram add data
- numberOfPoints - number of points in vertical cut
- numberOfVerticalDiagram - number of vertical cuts

SetExtraRotationPoint

Method set rotation point for interaction diagram which is used during design of reinforcement

Syntax:

```
Concrete.IntDiagram.SetExtraRotationPoint(Double IntDiagAddDataD, Double heighthRatio, Double strainRatio)
```

Where:

- IntDiagAddDataD - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData")
- heighthRatio - height ratio
- strainRatio - strain ratio

SetMaximalCompressedPortion

Method set maximal compressed portion for interaction diagram which is used during design of reinforcement

Syntax:

```
Concrete.IntDiagram.SetMaximalCompressedPortion(Double IntDiagAddDataD, Double compPortion)
```

Where:

- IntDiagAddDataD - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData")
- compPortion - compressed portion coefficient

InteractionDiagramNM

Method creates 2D plane interaction diagram and returns list of projected interaction diagram points (N,M).

Syntax:

```
object[] points = Concrete.IntDiagram.InteractionDiagramNM(Double, Double, Double)
```

Where:

- CsslD - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- IntDiagAddDataD - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

InteractionDiagramNMyMz

Method creates 3D interaction diagram surface and returns list of interaction diagram points (N, My, Mz).

Syntax:

```
object[] points = Concrete.IntDiagram.InteractionDiagramNMyMz(Double CssID, Double diagramAngle, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

CountPointsInteractionDiagram

Method returns number of points of plane interaction diagram points - InteractionDiagramNM.

Syntax:

```
double numPoints = Concrete.IntDiagram.CountPointsInteractionDiagram(Double, Double, Double)
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

CountPointsInteractionDiagramNMyMz

Method returns number of points of plane interaction diagram points - InteractionDiagramNMyMz.

Syntax:

```
double numPoints = Concrete.IntDiagram.CountPointsInteractionDiagramNMyMz(Double CssID, Double diagramAngle, Double IntDiagAddDataID)
```

Where:

CssID - Id of cross-section

diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz

IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

UltimateBorder3D

Method returns 3D ultimate border surface List of points of plane of deformations - eps0, epsY, epsZ.

Syntax:

```
object[] ultborder = Concrete.IntDiagram.UltimateBorder3D(Double CssID, Double diagramAngle, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

UltimateBorder

Method returns plane ultimate border - List of projected points of plane of deformations - eps0, kappa

Syntax:

```
object[] ultborder = Concrete.IntDiagram.UltimateBorder(Double CssID, Double diagramAngle, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

IntersectionsInteractionDiagramNM

Method returns intersection of plane interaction diagram and ray defined by angle and point. Output is number of intersection, first point intersection (N,M), second point intersection (N,M).

Syntax:

```
object intersections = Concrete.IntDiagram.IntersectionsInteractionDiagramNM(Double, CssID, Double diagramAngle, Double N, Double M, Double intersectionAngle, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- N - Value of normal forces
- M Value of projected bending moment
- intersectionAngle - angle of cut measured from M axis - Nu = 0°, Mu = 90°

- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

ResistanceInteractionDiagramNM

Method returns resistance point obtained from intersection of interaction diagram and ray defined by point (N,M) and angle. Output is resistance point - N,M.

Syntax:

```
object resistance = Concrete.IntDiagram.ResistanceInteractionDiagramNM(Double, CssID, Double diagramAngle, Double N, Double M, Double intersectionAngle, Double IntDiagAddDataID )
```

Where:

- CssID - Id of cross-section
- diagramAngle - angle of interaction diagram cut - 0 - My, 90 - Mz
- N - Value of normal forces
- M - Value of projected bending moment
- intersectionAngle - angle of cut measured from M axis - Nu = 0°, Mu = 90°
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

HorizontalSectionInteractionDiagram3D

Method returns plane horizontal cut obtained from intersection of 3D ID and cut plane defined by point (N,My,Mz). Output is list of points of cut (N, My, Mz).

Syntax:

```
object[] horCutPoints = Concrete.IntDiagram.HorizontalSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double IntDiagAddDataID )
```

Where:

CssID - Id of cross-section

N - Value of normal forces of definition point

My - Value of bending moment actioning around y axis of definition point

Mz - Value of bending moment actioning around z axis of definition point

IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

VerticalSectionInteractionDiagram3D

Method returns plane vertical cut obtained from intersection of 3D ID and cut plane defined by point (N,My,Mz) and angle. Output is list of points of cut (N, My, Mz).

Syntax:

```
object[] verCutPoints = Concrete.IntDiagram.VerticalSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N, My plane
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

GeneralSectionInteractionDiagram3D

Method returns plane general cut obtained from intersection of 3D ID and cut plane defined by point (N, My, Mz) and two angles. Output is list of points of cut (N, My, Mz).

Syntax:

```
object[] genCutPoints = Concrete.IntDiagram.GeneralSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor, Double sectionAngleVert, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N, My plane
- sectionAngleVert - angle between cut plane and My, Mz plane
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

CountPointsHorizontalSectionInteractionDiagram3D

Method returns number of points of horizontal cut obtained from intersection of 3D ID and cut plane defined by point (N, My, Mz)

Syntax:

```
double numPoints = Concrete.IntDiagram.CountPointsHorizontalSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double IntDiagAddDataID)
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

CountPointsVerticalSectionInteractionDiagram3D

Method returns number of points of plane vertical cut obtained from intersection of 3D ID and cut plane defined by point (N,My,Mz) and angle.

Syntax:

```
double numPoints = Concrete.IntDiagram.CountPointsVerticalSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor , Double IntDiagAddDataID )
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N,My plane
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

CountPointsGeneralSectionInteractionDiagram3D

Method returns number of points of plane general cut obtained from intersection of 3D ID and cut plane defined by point (N,My,Mz) and two angles

Syntax:

```
double numPoints = Concrete.IntDiagram.CountPointsGeneralSectionInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor , Double sectionAngleVert, Double IntDiagAddDataID )
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N,My plane
- sectionAngleVert - angle between cut plane and My,Mz plane

- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

IntersectionsInteractionDiagram3D

Method returns intersection points of 3D interaction diagram and ray defined by point and two angels - Output is number of inter-sections first intersection point (N,My,Mz), second intersection point (N,My,Mz).

Syntax:

```
object intersection = Concrete.IntDiagram.IntersectionsInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor, Double sectionAngleVert, Double IntDiagAddDataID )
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N,My plane
- sectionAngleVert - angle between cut plane and My,Mz plane
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

ResistanceInteractionDiagram3D

Method returns resistance points obtained from intersection of 3D interaction diagram and ray defined by point and two angels. Output is resistance point (N,My,Mz).

Syntax:

```
object resistpoint = Concrete.IntDiagram.ResistanceInteractionDiagram3D(Double CssID, Double N, Double My, Double Mz, Double sectionAngleHor, Double sectionAngleVert, Double IntDiagAddDataID )
```

Where:

- CssID - Id of cross-section
- N - Value of normal forces of definition point
- My - Value of bending moment acting around y axis of definition point
- Mz - Value of bending moment acting around z axis of definition point
- sectionAngleHor - angle between cut plane and N,My plane
- sectionAngleVert - angle between cut plane and My,Mz plane
- IntDiagAddDataID - Id of diagram add data which was created with function SetIntDiagramAddData (viz "SetIntDiagramAddData"), use -1 for default

Concrete toolbox - POD

Help for methods under group "POD"

AngleOfNeutralAxis

Function returns direction of neutral axis (angle between y axis and neutral axis of c s s in deg)

Syntax:

```
double angle = CONCRETE.POD.AngleOfNeutralAxis();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double angle = CONCRETE.POD.AngleOfNeutralAxis(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

BalanceHeighCompressionZone

Function returns limit value of depth of compression zone -value xbal [m], if limit strain of concrete in the most compressive fibre and strain of reinforcement on the beginning of plastic branch is reached

Syntax:

```
double height = CONCRETE.POD.BalanceHeighCompressionZone();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double height = CONCRETE.POD.BalanceHeighCompressionZone(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CompressivePartOfCss

Function returns polygon (array of point with two columns y[m] and z[m]) of compressive part of cross-section

Syntax:

```
object polygon = CONCRETE.POD.CompressivePartOfCss(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CurvatureRes

Returns the resultant value of curvature of Css

Syntax:

```
double cur = CONCRETE.POD.CurvatureRes(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CurvatureY

Returns the curvature of C_{ss} around y-axis of CSS

Syntax:

```
double cur = CONCRETE.POD.CurvatureY(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CurvatureZ

Syntax: CONCRETE.POD.CurvatureZ(Double CSIndex, Double ForceIndex);

Returns the curvature of C_{ss} around z-axis of CSS

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

EffectiveHeight

Function returns effective depth of cross-section (distance from the furthest compressed fibre of cross-section to center of gravity of tensile reinforcement)-value d [m]

Syntax:

```
double height = CONCRETE.POD.EffectiveHeight(Double EffectiveHeightParam);
```

Where:

- EffectiveHeightParam = 0.9 - default value

Values are computed for default cross-section (C_{ss}ID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double height = CONCRETE.POD.EffectiveHeight(Double CSIndex, Double ForceIndex, Double EffectiveHeightParam);
```

Function returns effective depth of cross-section (distance from the furthest compressed fibre of cross-section to center of gravity of tensile reinforcement)-value d [m]

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation
- EffectiveHeightParam = 0.9 - default value

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

EquilibriumPlane

Function returns plane of equilibrium (translation in direction x, rotation around y axis and z axis)

Syntax:

```
object POD = CONCRETE.POD.EquilibriumPlane();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object POD = CONCRETE.POD.EquilibriumPlane(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetIntersectionPoints

Function returns intersections (points with coordinates yi[m] and zi [m]) of neutral axis and concrete cross-section (polygon)

Syntax:

```
object points = CONCRETE.POD.GetIntersectionPoints();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object points = CONCRETE.POD.GetIntersectionPoints(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetIntersectionPointsCount

Function returns intersections points count of neutral axis and concrete cross-section (polygon)

Syntax:

```
double count = CONCRETE.POD.GetIntersectionPointsCount();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double count = CONCRETE.POD.GetIntersectionPointsCount(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

HeightOfCompressionZone

Function returns depth of compression zone (the distance from most compressive fibre to neutral axis) -value x [m]

Syntax:

```
double height = CONCRETE.POD.HeightOfCompressionZone();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double height = CONCRETE.POD.HeightOfCompressionZone(Double CSIndex, Double ForceIndex);
```

Where;

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLevelArm

Function returns inner level arm-value z [m]

Syntax:

```
double arm = CONCRETE.POD.InnerLevelArm();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double arm = CONCRETE.POD.InnerLevelArm(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLevelArm4Param

Function returns inner level arm-value z [m]

Syntax:

```
CONCRETE.POD.InnerLevelArm4Param(Double CSIndex, Double ForcelIndex, Double InnerleverParam, Double CalcType);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- InnerleverParam - Parameter for calculation when InnerLevelArm is calculated from formula. = 0
- CalcType - 0 = compression force concrete and tensile force reinforcement, 1 = compression force all tensile force all

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLeverArmPartNeg

Function returns part of inner level (distance to center of compressive to center of CSS force) -value z- [m]

Syntax:

```
double arm = CONCRETE.POD.InnerLeverArmPartNeg(Double CSIndex, Double ForcelIndex, Double InnerleverParam, Double CalcType);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation
- InnerleverParam - Parameter for calculation when InnerLevelArm is calculated from formula. = 0

- CalcType
 - 0 - compression force concrete and tensile force reinforcement
 - 1 - compression force and tensile force all

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLeverArmPartPos

Function returns part of inner level arm (distance to center of tensile force to center of CSS) -value z+ [m]

Syntax:

```
double armPart = CONCRETE.POD.InnerLeverArmPartPos(Double CSIndex, Double ForceIndex, Double Inner-leverParam, Double CalcType);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation
- InnerleverParam - Parameter for calculation when InnerLevelArm is calculated from formula. = 0
- CalcType
 - 1 - compression force and tensile force all
 - 0 - compression force concrete and tensile force reinforcement

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

LimitHeighCompressionZone

Function returns limit value of depth of compression zone -value xlim[m] , if limit strain of concrete in the most compressive fibre and limit strain of the most tensioned reinforcement is reached

Syntax:

```
double limit = CONCRETE.POD.LimitHeighCompressionZone();
```

Values are computed for default cross-section (CcssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double limit = CONCRETE.POD.LimitHeighCompressionZone(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TensilePartOfCss

Function returns polygon (array of point with two columns y[m] and z[m]) of tensile part of cross-section

Syntax:

```
double object = CONCRETE.POD.TensilePartOfCss(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TranslationX

Returns the translation of Css in direction of x -axis

Syntax:

```
double transX = CONCRETE.POD.TranslationX(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: POD](#)

Concrete toolbox - PODForce

Help for methods under group "PODForce"

ForcesFromEquilibriumOfPlane

Function returns internal forces (N, My, Mz) for inputting plane of equilibrium (translation in direction of x axis, rotation around y axis and z axis)

Syntax:

```
CONCRETE.PODForce.LimitHeighCompressionZone(Double CSIndex, Double eps0, Double epsY, Double epsZ);
```

Where:

- CSIndex - Id of cross section
- eps0 - translation in direction x of axis
- epsY - rotation around y axis
- epsZ - rotation around z axis

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example:PODForce](#)

Concrete toolbox - RegionData

Help for methods under group "RegionData"

ArrayOfReinforcement

Function returns array of reinforcement in selected cross-section with the basic properties - four columns (position y (.PosY) [m], position z (.PosZ) [m], diameter (.Diameter) [m], area of reinforcement (.Area) [m²])

Syntax:

```
object ar = CONCRETE.RegionData.ArrayOfReinforcement(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

CssCharacteristicCon

Function returns the basic cross-section characteristics ($t_y, t_z, A, I_y, I_z, S_y, S_z, D_{yz}, u$) of concrete cross-section

Syntax:

```
CONCRETE.RegionData.CssCharacteristicCon(Double CSIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

CssCharacteristicReinf

Function returns the basic cross-section characteristics ($t_y, t_z, A, I_y, I_z, S_y, S_z, D_{yz}, u$) of non-prestressed reinforcement

Syntax:

```
object charreinf = CONCRETE.RegionData.CssCharacteristicReinf(Double CSIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

RegionArrayOfReinforcement

Function returns array of reinforcement inside defined region of the cross-section with the basic properties - four columns (position y (.PosY) [m], position z (.PosZ) [m], diameter (.Diameter) [m], area of reinforcement (.Area) [m²])

Syntax:

```
object ar = CONCRETE.RegionData.RegionArrayOfReinforcement(Double CSIndex, Double AngleCutLine, Double
StripeWidthAboveCutLine, Double StripeWidthUnderCutLine, Double CoordYCutLine, Double CoordZCutLine);
```

Where:

- CSIndex - Id of cross section
- AngleCutLine - angle of cut line from y-axis of CSS [deg]
- StripeWidthAboveCutLine - Is width of the region above line for cutting of css
- StripeWidthUnderCutLine - Is width of the region under line for cutting of css
- CoordYCutLine - Is y-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CoordZCutLine - Is z-coordinate of reference point for definition of the line for cutting of css related to point [0,0]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

RegionCharacteristicCon

Function for calculation of basic css characteristic of part of concrete cross-section defined by cutting line with angle and width.

Syntax:

```
object cc = CONCRETE.RegionData.RegionCharacteristicCon(Double CSIndex, Double AngleCutLine, Double
StripeWidthAboveCutLine, Double StripeWidthUnderCutLine, Double CoordYCutLine, Double CoordZCutLine, Double
CSIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- AngleCutLine - angle of cut line from y-axis of CSS [deg]
- StripeWidthAboveCutLine - Is width of the region above line for cutting of css
- StripeWidthUnderCutLine - Is width of the region under line for cutting of css
- CoordYCutLine - Is y-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CoordZCutLine - Is z-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

RegionCharacteristicReinf

Function for calculation of basic css characteristic of reinforcement inside of part of cross-section

Syntax:

```
object cr = CONCRETE.RegionData.RegionCharacteristicReinf(Double CSIndex, Double AngleCutLine, Double StripeWidthAboveCutLine, Double StripeWidthUnderCutLine, Double CoordYCutLine, Double CoordZCutLine, Double CSIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- AngleCutLine - angle of cut line from y-axis of CSS [deg]
- StripeWidthAboveCutLine - Is width of the region above line for cutting of css
- StripeWidthUnderCutLine - Is width of the region under line for cutting of css
- CoordYCutLine - Is y-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CoordZCutLine - Is z-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

RegionCharacteristicTransf

Function for calculation of basic transformed C_{ss} characteristic of part of cross-section

Syntax:

```
object rct = CONCRETE.RegionData.RegionCharacteristicTransf(Double CSIndex, Double AngleCutLine, Double StripeWidthAboveCutLine, Double StripeWidthUnderCutLine, Double CoordYCutLine, Double CoordZCutLine, Double CSIndex, Double CSCH_dy_Center, Double CSCH_dz_Center, Double CSCH_alfa_rotation);
```

Where:

- CSIndex - Id of cross section
- AngleCutLine - angle of cut line from y-axis of CSS [deg]
- StripeWidthAboveCutLine - Is width of the region above line for cutting of css
- StripeWidthUnderCutLine - Is width of the region under line for cutting of css
- CoordYCutLine - Is y-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CoordZCutLine - Is z-coordinate of reference point for definition of the line for cutting of css related to point [0,0]
- CSCH_dy_Center - Is y-coordinate of reference point for calculation related to point [0,0]
- CSCH_dz_Center - Is z-coordinate of reference point for calculation related to point [0,0]
- CSCH_alfa_rotation - Angle from y axis of css to which value cs characteristics will be recalculated [deg]

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

[Load an example: RegionData](#)

Concrete toolbox - ReinfBars

Help for methods under group "ReinfBars"

ReinfBars

Function directly returns array of reinforcement in cross section in selected cross-section with the basic properties - four columns (position y (.PosY) [m], position z (.PosZ) [m], diameter (.Diameter) [m], area of reinforcement (.Area) [m²]). Returned values are same as used in CS library in dilaogue

Syntax:

```
object bars = CONCRETE.ReinfBars[double ReinfID].X or .Y or .D or .Area
```

Where:

- ReinfID- Id of reinforcement

[Load an example: ReinfBars](#)

Concrete toolbox - Response

Help for methods under group "Response"

CompressedConcrete

Function returns resultant forces in compression concrete

Syntax:

```
object forces = CONCRETE.Response.CompressedConcrete();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces = CONCRETE.Response.CompressedConcrete(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

CompressedReinf

Function returns resultant force in tension non-prestressed bars

Syntax:

```
object forces =CONCRETE.Response.CompressedReinf();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces = CONCRETE.Response.CompressedReinf(Double CSIndex, Double ForcelIndex);
```

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

Resultant

Function returns resultant force in whole Ccss

Syntax:

```
object forces =CONCRETE.Response.Resultant();
```

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces =CONCRETE.Response.Resultant(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TensionConcrete

Function returns resultant forces in tension concrete

Syntax:

```
object forces = CONCRETE.Response.TensionConcrete();
```

Values are computed for default cross-section (CcsID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces =CONCRETE.Response.TensionConcrete(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

TensionReinf

Function returns resultant force in tension non-prestressed bars

Syntax:

```
object forces =CONCRETE.Response.TensionReinf();
```

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces =CONCRETE.Response.TensionReinf(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

WholeConcrete

Function returns resultant forces in concrete

Syntax:

```
object forces =CONCRETE.Response.WholeConcrete();
```

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces =CONCRETE.Response.WholeConcrete(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

WholeReinf

Function returns resultant force in non-prestressed bars

Syntax:

```
object forces = CONCRETE.Response.WholeReinf();
```

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
object forces =CONCRETE.Response.WholeReinf(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex -Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: Responce](#)

Concrete toolbox - SetForces

Help for methods under group "SetForces"

SetForces

Set forces for ForcelIndex equal to 0. This force index is used in many others concrete commands for select appropriate set of forces.

Syntax:

```
CONCRETE.SetForces(Double N, Double Vy, Double Vz, Double Mx, Double My, Double Mz);
```

Where:

- N - Normal force
- Vy - Shear force in Y-axis direction
- Vz - Shear force in Z-axis direction
- Mx - Torsion moment
- My - Bending moment around Y-axis
- Mz - Bending moment around Z-axis

SetForcesId

Set forces for specified ForceIndex. This force index is used in many others concrete commands for select appropriate set of forces.

Syntax:

```
CONCRETE.SetForcesId(Double ForceIndex, Double N, Double Vy, Double Vz, Double Mx, Double My, Double Mz);
```

Where;

- ForceIndex - Id of forces for calculation
- N - Normal force
- Vy - Shear force in Y-axis direction
- Vz - Shear force in Z-axis direction
- Mx - Torsion moment
- My - Bending moment around Y-axis
- Mz - Bending moment around Z-axis

[Load an example: SetForces](#)

Concrete toolbox - Shear

Help for methods under group "Shear"

AngleShearForceResultant

Function returns angle of shear force resultant from y-axis of css

Syntax:

```
double angle = CONCRETE.Shear.AngleShearForceResultant(Double VdEy, Double VdEz);
```

Where:

- VdEy - Shear force in direction of y-axis of css
- VdEz - Shear force in direction of z-axis of css

CssDimension

Function returns dimension of cross-section in inputted direction crossing the inputted point

Syntax:

```
double dimension = CONCRETE.Shear.CssDimension(Double CSIndex, Double WidthPointY, Double WidthPointZ, Double RotAngle);
```

Where:

- CSIndex- Id of cross section
- WidthPointY - Is y-coordinate of reference point for definition of the line for calculation dimension of css related to point [0,0]
- WidthPointZ - Is z-coordinate of reference point for definition of the line for calculation dimension of css related to point [0,0]
- RotAngle - Angle for calculation dimension of css from y-axis of CSS in deg

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

EffectiveHeigthProjection

Function returns effective depth of Css recalculated to defined direction drec [m]

Syntax:

```
double height = CONCRETE.Shear.EffectiveHeigthProjection(Double CalcMode, Double RotAngle, Double EffectHeithParam);
```

Where:

- CalcMode - Unused param. Set 0
- RotAngle - Angle from y axis of css to which value d will be recalculated [deg]
- EffectHeithParam - Coefficient for calculation effective depth, if this value can not be calculated from plane of deformation, $d = x \cdot h$

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double height = CONCRETE.Shear.EffectiveHeighProjection(Double CSIndex, Double ForcelIndex, Double RotAngle, Double EffectHeithParam);
```

Where:

- CSIndex- Id of cross section
- ForcelIndex - Id of forces for calculation
- RotAngle - Angle from y axis of css to which value d will be recalculated [deg]
- EffectHeithParam - Coefficient for calculation effective depth, if this value can not be calculated from plane of deformation, $d = x \cdot h$

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLeverArmPartNegProjection

Function returns part of inner level arm (distance to center of compressive force to center of CSS) force recalculated to defined direction -value zrec+ [m]

Syntax:

```
double armPart = CONCRETE.Shear.InnerLeverArmPartNegProjection(Double CSIndex, Double ForcelIndex, Double RotAngle, Double InnerLeverParam, Double CalcType);
```

Where:

- CSIndex- Id of cross section
- ForcelIndex - Id of forces for calculation
- RotAngle - Angle from y axis of css to which value is recalculated [deg]
- InnerLeverParam - Coefficient for calculation inner lever arm, if this value can not be calculated from plane of deformation, $zcal = \text{InnerLeverParam} \cdot dcal$
- CalcType - Type of calculation inner lever arm
 - 0 - inner lever arm is calculated from eccentricity of forces in compressive concrete and in tensile reinforcement,
 - 1 - inner lever arm is calculated from eccentricity of all compressive forces and all tensile forces

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerLeverArmPartPosProjection

Function returns part of inner level arm (distance to center of tensile force to center of CSS) recalculated to defined direction -value zrec+ [m]

Syntax:

```
double armPart = CONCRETE.Shear.InnerLeverArmPartPosProjection(Double CSIndex, Double ForceIndex, Double RotAngle, Double InnerLeverParam, Double CalcType);
```

Where:

- CSIndex- Id of cross section
- ForceIndex - Id of forces for calculation
- RotAngle - Angle from y axis of css to which value is recalculated [deg]
- InnerLeverParam - Coefficient for calculation inner lever arm, if this value can not be calculated from plane of deformation, $zcal = \text{InnerLeverParam} * dcal$
- CalcType - Type of calculation inner lever arm
 - 0 - inner lever arm is calculated from eccentricity of forces in compressive concrete and in tensile reinforcement,
 - 1 - inner lever arm is calculated from eccentricity of all compressive forces and all tensile forces

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

InnerleverArmProjection

Function returns inner lever arm of C_{ss} recalculated to defined direction z_{rec} [m]

Syntax:

```
double armProj = CONCRETE.Shear.InnerleverArmProjection(Double CalcMode, Double RotAngle, Double InnerLeverParam, Double CalcType);
```

Where:

- CalcMode - Unused param. Set 0
- RotAngle - Angle from y axis of css to which value d will be recalculated [deg]
- InnerLeverParam - Coefficient for calculation inner lever arm, if this value can not be calculated from plane of deformation, $zcal = \text{InnerLeverParam} * dcal$
- CalcType - Type of calculation inner lever arm
 - 0 - inner lever arm is calculated from eccentricity of forces in compressive concrete and in tensile reinforcement,
 - 1 - inner lever arm is calculated from eccentricity of all compressive forces and all tensile forces

Values are computed for default cross-section (C_{ss}ID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double armProj = CONCRETE.Shear.InnerleverArmProjection(Double CSIndex, Double ForceIndex, Double RotAngle, Double InnerLeverParam, Double CalcType);
```

Where:

- CSIndex- Id of cross section
- ForceIndex - Id of forces for calculation
- RotAngle - Angle from y axis of css to which value is recalculated [deg]
- InnerLeverParam - Coefficient for calculation inner lever arm, if this value can not be calculated from plane of deformation, $z_{cal} = \text{InnerLeverParam} * d_{cal}$
- CalcType - Type of calculation inner lever arm
 - 0 - inner lever arm is calculated from eccentricity of forces in compressive concrete and in tensile reinforcement,
 - 1 - inner lever arm is calculated from eccentricity of all compressive forces and all tensile forces

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

PositionWidthCssForShear

Function returns distance from the center of cross-section, where value WidthCssForShear is calculated

Syntax:

```
object pos = CONCRETE.Shear.PositionWidthCssForShear(Double CSIndex, Double ForceIndex, Double TypeOfArea);
```

Where:

- CSIndex- Id of cross section
- ForceIndex - Id of forces for calculation
- TypeOfArea - Is type of area , when the minimum width is calculated
 - 0 - in tension area of the concrete,
 - 1 - area between resultant of force in compressive concrete and tensile reinforcement , it means within inner lever arm

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

ShearForceResultant

Function calculate and returns resultant of shear force $V_{Ed} = (V_{Edy}^2 + V_{Edz}^2)^{0.5}$

Syntax:

```
double forces = CONCRETE.Shear.ShearForceResultant(Double VdEy, Double VdEz);
```

Where:

- VdEy - Shear force in direction of y-axis of css
- VdEz - Shear force in direction of z-axis of css

WidthCssForShear

Function returns the minimum width of cross-section in selected area perpendicular to resultant of shear force -bw[m]

Syntax:

```
double width = CONCRETE.Shear.WidthCssForShear(Double CSIndex, Double ForceIndex, Double TypeOfArea);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- TypeOfArea - Is type of area , when the minimum width is calculated
 - 0 - in tension area of the concrete,
 - 1 - area between resultant of force in compressive concrete and tensile reinforcement , it means within inner lever arm

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

WidthCssGeneral

Function returns dimension of cross-section in inputted direction crossing the inputted point

Syntax:

```
double width = CONCRETE.Shear.WidthCssGeneral(Double WidthPointY, Double WidthPointZ, Double RotAngle);
```

Where:

- WidthPointY - Is y-coordinate of reference point for definition of the line for calculation dimension of css related to point [0,0]
- WidthPointZ - Is z-coordinate of reference point for definition of the line for calculation dimension of css related to point [0,0]
- RotAngle - Angle for calculation dimension of css from y-axis of CSS in deg

[Load an example: Shear](#)

Concrete toolbox - SpacingCover

Help for methods under group "SpacingCover"

CoverReinf

Function returns minimum distance of the all or selected reinforcement bar from nearest the edge of cross-section in defined direction. The direction is defined by the angle (for example straight line in direction of plane of deformation).

Syntax:

```
double dist = CONCRETE.SpacingCover.CoverReinf(Double CSIndex, Double DirAngle, Double BarIndex);
```

Where:

- CSIndex - ID of cross-section
- DirAngle - Angle for calculation cover from y-axis of CSS in deg
- BarIndex - Is index of bar for which the concrete cover will be calculated (if BarIndex=-1, the minimum cover from all bars of reinforcement will be calculated)

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

MaxSpacingReinf

Function returns maximum center to center spacing of non-prestressed bars for defined type of reinforcement and direction.

Syntax:

```
double max = CONCRETE.SpacingCover.MaxSpacingReinf(Double CSIndex, Double ForcIndex, Double ReinfType, Double DirType);
```

Where:

- CSIndex - ID of cross-section
- ForcIndex - Id of forces for calculation
- ReinfType - Type of reinforcement
 - 0 - all reinforcement,
 - 1 - only tensile reinforcement,
 - 2 - only compressive reinforcement
- DirType - Is direction of edges, for which the distance will be checked (0=bars at edges in all direction, 1=bars at edges in direction of y axis of css, bars at edges in direction of z axis of css)

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

MinSpacingReinf

Function returns minimum clear spacing (spacing between surfaces of bars) of non-prestressed bars for defined type of reinforcement and direction.

Syntax:

```
double min = CONCRETE.SpacingCover.MaxSpacingReinf(Double CSIndex, Double ForceIndex, Double ReinfType, Double DirType);
```

Where:

- CSIndex - ID of cross-section
- ForceIndex - Id of forces for calculation
- ReinfType - Type of reinforcement
 - 0 - all reinforcement,
 - 1 - only tensile reinforcement,
 - 2 - only compressive reinforcement
- DirType - Is direction of edges, for which the distance will be checked (0=bars at edges in all direction, 1=bars at edges in direction of y axis of css, bars at edges in direction of z axis of css)

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

[Load an example: SpacingCover](#)

Concrete toolbox - Stirrup

Help for methods under group "Stirrup"

CreateStirrup

Function creates stirrup and returns index of stirrup, if stirrup was created

Syntax:

```
CONCRETE.Stirrup.CreateStirrup(Double StirrupID, Double Diameter, Double AngleStirrup, Double DistanceOfStirrup, Double StirrupFyk, Object ArrayPointsStirrup);
```

Where:

-
- StirrupID - ID of stirrup, which will be created
 - Diameter - Diameter of stirrup dss
 - AngleStirrup - Angle of stirrup [deg] from horizontal axis -axis x of the member
 - DistanceOfStirrup - Is centre-to centre spacing of stirrup s [m] in direction of member axis - axis x
 - StirrupFyk - Characteristic yield strength of shear reinforcement [Pa]
 - ArrayPointsStirrup - Array of points of stirrup vertexes with two columns (coordinate y[m] and z[m])

NumberOfStirrupLinks

Function returns number of stirrup links (intersection of stirrups link with the cut line).

Syntax:

```
double num = CONCRETE.Stirrup.NumberOfStirrupLinks(Double StirrupID, Double PointCoordY, Double PointCoordZ, Double Angle);
```

Where:

- StirrupID - ID of stirrup, which will be created
- PointCoordY - Is y-coordinate of reference point for definition of the cut line of css related to point [0,0]
- PointCoordZ - Is z-coordinate of reference point for definition of the cut line of css related to point [0,0],
- Angle - Angle of cut line from y-axis of CSS in deg

[Load an example: Stirrup](#)

Concrete toolbox - StirrupAsw

Help for methods under group "StirrupAsw"

ClearStirrupZones

Clears all inputed zones.

Syntax:

```
CONCRETE.StirrupAsw.ClearStirrupZones();
```


CreateStirrupZone

Function creates stirrup zones and returns index of zone, if zone was created.

Syntax:

```
CONCRETE.StirrupAsw.CreateStirrupZone(Double ZoneID, Double Xbeg, Double Xend, Double NumberOfStirrupLinks, Array ShapeOfStirrup, Array PositionOfStirrup, Double Fywk);
```

Where:

- StirrupID - ID of stirrup, which will be created
- Xbeg - Is x coordinate of begin point of the zone on the member (in direction of member axis)
- Xend - Is x coordinate of the the end point of the zone on the member (in direction of member axis)
- NumberOfStirrupLinks - Number of stirrup links (if NumberOfStirrupLinks = 0, the number of stirrup links will be calculated automatically with using function NumberOfStirrupLinks, else defined value will be taken into account)
- ShapeOfStirrup - Array of points of stirrup vertexes with three columns (coordinate y[m], z[m], ID of stirrup shape)
- PositionOfStirrup - Array of with 4 column (position of stirrup in zone from Xbeg [m], diameter of stirrup [m], angle of stirrup from x axis of the member [rad], number of stirrup at defined position)
- Fywk - Is characteristic yield strength of shear reinforcement

[Load an example: StirrupAsw](#)

Concrete toolbox - StressStrain

Help for methods under group "StressStrain"

GetCountFibresConcrete

Function returns number of concrete fibers in cross-section.

Syntax:

```
double numFibConc = CONCRETE.StressStrain.GetCountFibresConcrete(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetCountFibresReinforcement

Function returns number of non-prestressed bars of reinforcement in cross-section.

Syntax:

```
double numFibReinf = CONCRETE.StressStrain.GetCountFibresReinforcement(Double CSIndex);
```

Where:

- CSIndex - Id of cross section

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetExtremeStrainOfConcreteMax

Function returns maximum value of strain from all concrete fibres.

Syntax:

```
double extremeMaxConc = CONCRETE.StressStrain.GetExtremeStrainOfConcreteMax(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStrainOfConcreteMin

Function returns minimum value of strain from all concrete fibres.

Syntax:

```
double extremeMinConc = CONCRETE.StressStrain.GetExtremeStrainOfConcreteMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStrainOfReinfMax

Function returns maximum value of strain from all non-prestressed bars of reinforcement.

Syntax:

```
double extremeMaxReinf = CONCRETE.StressStrain.GetExtremeStrainOfReinfMax(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStrainOfReinfMin

Function returns minimum value of strain from all non-prestressed bars of reinforcement.

Syntax:

```
double extremeMinReinf = CONCRETE.StressStrain.GetExtremeStrainOfReinfMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStressOfConcreteMax

Function returns maximum value of stress from all concrete fibres.

Syntax:

```
double extremeStressMaxConc = CONCRETE.StressStrain.GetExtremeStressOfConcreteMax(Double CSIndex,  
Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStressOfConcreteMin

Function returns minimum value of stress from all concrete fibres.

Syntax:

```
double extremeStressMinConc = CONCRETE.StressStrain.GetExtremeStressOfConcreteMin(Double CSIndex,  
Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStressOfReinfMax

Function returns maximum value of stress from all non-prestressed bars of reinforcement.

Syntax:

```
double extremeStressMaxReinf = CONCRETE.StressStrain.GetExtremeStressOfReinfMax(Double CSIndex, Double  
ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetExtremeStressOfReinfMin

Function returns minimum value of stress from all non-prestressed bars of reinforcement.

Syntax:

```
double extremeStressMaxReinf = CONCRETE.StressStrain.GetExtremeStressOfReinfMin(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetFiberReinfDiameter

Function returns bar diameter for non-prestressed bar index in actual cross section.

Syntax:

```
double diam = CONCRETE.StressStrain.GetFiberReinfDiameter(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CcsID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double diam = CONCRETE.StressStrain.GetFiberReinfDiameter(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetFibreExtremeStrainMax

Function returns index of concrete fibre, where is maximum value of strain.

Syntax:

```
double fiberIdExStrMax = CONCRETE.StressStrain.GetFibreExtremeStrainMax(Double CSIndex, Double For-  
celIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

GetFibreExtremeStrainMin

Function returns index of concrete fibre, where is minimum value of strain.

Syntax:

```
double fiberIdExStrMin = CONCRETE.StressStrain.GetFibreExtremeStrainMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetFibreExtremeStressMax

Function returns index of concrete fibre , where is maximum value of stress.

Syntax:

```
double fiberIdExStressMax = CONCRETE.StressStrain.GetFibreExtremeStressMax(Double CSIndex, Double For-  
celIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetFibreExtremeStressMin

Function returns index of concrete fibre , where is minimum value of stress.

Syntax:

```
double fiberIdExStressMin = CONCRETE.StressStrain.GetFibreExtremeStressMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

GetFibrePositionConcrete

Function returns index of concrete fibre , where is minimum value of stress in actual cross section.

Syntax:

```
object point = CONCRETE.StressStrain.GetFibrePositionConcrete(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.StressStrain.GetFibrePositionConcrete(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetFibrePositionReinf

Function returns non-prestressed bar position for given index in actual cross section.

Syntax:

```
object point = CONCRETE.StressStrain.GetFibrePositionReinf(Double Index);
```

Where;

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
object point = CONCRETE.StressStrain.GetFibrePositionReinf(Double CSIndex, Double Index);
```

Where

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetLimitStrainOfConcrete

Function returns limit value of strain in selected concrete fibre in actual cross section.

Syntax:

```
double limStrainConc = CONCRETE.StressStrain.GetLimitStrainOfConcrete(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double limStrainConc = CONCRETE.StressStrain.GetLimitStrainOfConcrete(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetLimitStrainOfReinf

Function returns limit value of stain in selected non-prestressed bar of reinforcement in actual cross section.

Syntax:

```
double limStrReinf = CONCRETE.StressStrain.GetLimitStrainOfReinf(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CcsID = 0, viz "CreateCS").

This syntax is obsolete, please, use following one.

Syntax:

```
double limStrReinf = CONCRETE.StressStrain.GetLimitStrainOfReinf(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

GetLimitStressOfConcrete

Function returns limit value of stress in selected concrete fibre in actual cross section.

Syntax:

```
double limStressConc = CONCRETE.StressStrain.GetLimitStressOfConcrete(Double Index);
```

Where:

- Index - Fibre index

This syntax is obsolete, please, use following one.

Syntax:

```
double limStressConc = CONCRETE.StressStrain.GetLimitStressOfConcrete(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

GetLimitStressOfReinf

Function returns limit value of stress in selected non-prestressed bar of reinforcement in actual cross section.

Syntax:

```
double limStressReinf = CONCRETE.StressStrain.GetLimitStressOfReinf(Double Index);
```

Where:

- Index - Fibre index

Syntax:

```
double limStressReinf = CONCRETE.StressStrain.GetLimitStressOfReinf(Double CSIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- Index - Fibre index

GetReinfExtremeStrainMax

Function returns index of non-prestressed bar of reinforcement, where is maximum value of strain.

Syntax:

```
double index = CONCRETE.StressStrain.GetReinfExtremeStrainMax(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

GetReinfExtremeStrainMin

Function returns index of non-prestressed bar of reinforcement, where is minimum value of strain.

Syntax:

```
double index = CONCRETE.StressStrain.GetReinfExtremeStrainMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetReinfExtremeStressMax

Function returns index of non-prestressed bar of reinforcement, where is maximum value of stress.

Syntax:

```
double index = CONCRETE.StressStrain.GetReinfExtremeStressMax(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetReinfExtremeStressMin

Function returns index of non-prestressed bar of reinforcement, where is minimum value of stress.

Syntax:

```
double index = CONCRETE.StressStrain.GetReinfExtremeStressMin(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetStrainOfConcrete

Function returns value of strain in selected concrete fibre in actual cross section.

Syntax:

```
double strain = CONCRETE.StressStrain.GetStrainOfConcrete(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CsslD = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double strain = CONCRETE.StressStrain.GetStrainOfConcrete(Double CSIndex, Double, ForcelIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetStrainOfReinf

Function returns value of stain in selected non-prestressed bar of reinforcement in actual cross section.

Syntax:

```
double strain = CONCRETE.StressStrain.GetStrainOfReinf(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double strain = CONCRETE.StressStrain.GetStrainOfReinf(Double CSIndex, Double, ForceIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- Index - Fibre index

GetStressOfConcrete

Function returns value of stress in selected concrete fibre in actual cross section.

Syntax:

```
double stress = CONCRETE.StressStrain.GetStressOfConcrete(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double stress = CONCRETE.StressStrain.GetStressOfConcrete(Double CSIndex, Double, ForceIndex, Double Index);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

GetStressOfReinf

Function returns value of stress in selected non-prestressed bar of reinforcement in actual cross section.

Syntax:

```
double stress = CONCRETE.StressStrain.GetStressOfReinf(Double Index);
```

Where:

- Index - Fibre index

Values are computed for default cross-section (CssID = 0, viz "CreateCS") and for default forces (ForcesID = 0, viz "SetForces").

This syntax is obsolete, please, use following one.

Syntax:

```
double stress = CONCRETE.StressStrain.GetStressOfReinf(Double CSIndex, Double, ForceIndex, Double Index);
```

Function returns value of stress in selected non-prestressed bar of reinforcement.

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation
- Index - Fibre index

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

[Load an example: StressStrain](#)

Concrete toolbox - Torsion

Help for methods under group "Torsion"

AdditionalStrainReinf

Returns max. strain in reinforcement for C_{ss} with linear diagram, and only inputed axial force

Syntax:

```
double maxStrain = CONCRETETorsion.AdditionalStrainReinf(Double CSIndex, Double AddTensionForce);
```

Where:

- CSIndex - Id of cross section
- AddTensionForce - Tension forces added on the CSS

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

CreateEffRectCssForTorsion

Function creates rectangular concrete cross-section to perimeter and area of target and source css were same and returns ID of cross-section.

Syntax:

```
CONCRETE.Torsion.CreateEffRectCssForTorsion(Double CssIDTarget, Double CssIDSource);
```

Where:

- C_{ss}IDTarget - Id of target C_{ss}
- C_{ss}IDSource - Id of source C_{ss}

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

CreateGeneralCssForTorsion

Function returns polygon (array of point - y[m] and z[m] coordinate) of thin-walled cross-section for torsion according to selecting type (0-outer polygon, 1-center line polygon, 2-inner polygon).

Syntax:

```
CONCRETE.Torsion.CreateGeneralCssForTorsion(Double CssIDTarget, Double CssIDSource, Double EffectWallThickness, Double StirrupID);
```

Where:

- CssIDTarget - Id of target Css
- CssIDSource - Id of source Css
- EffectWallThickness - Effective wall thickness
- StirrupID - Id of stirrup

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

LongReinAreaForTorsion

Function returns area of longitudinal reinforcement enclosed by stirrup and inside of defined strip from the edge of the stirrup for torsion.

Syntax:

```
double area = CONCRETE.StressStrain.LongReinAreaForTorsion(Double CSIndex, Double StirrupID, Double StripeWidth);
```

Where:

- CSIndex - Id of cross section
- StirrupID - Id of stirrup
- StripeWidth - Width of stripe

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

MinDistReinInsideStirrup

Function returns minimum distance between edge of the cross-section and center of bar of longitudinal reinforcement enclosed by selected stirrup and inside define stripe from the edge of stirrup.

Syntax:

```
double dist = CONCRETE.Torsion.MinDistReinInsideStirrup(Double CSIndex, Double StirrupID, Double StripeWidth);
```

Where:

- CSIndex - Id of cross section
- StirrupID - Id of stirrup

- StripeWidth - Width of stripe

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

ParameterAkFromEffRectCSS

Function returns value A_k [m²] calculated for effective rectangular cross-section.

Syntax:

```
double Ak = CONCRETE.Torsion.ParameterAkFromEffRectCSS(Double CSIndex, Double EffectWallThickness);
```

Where:

- CSIndex - Id of cross section
- EffectWallThickness - Effective wall thickenss

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

ParameterAkFromEffRectCSS

Function returns value A_k [m²] (area enclosed by defined polygon) for check torsion.

Syntax:

```
double Ak = CONCRETE.Torsion.ParameterAkFromEffRectCSS(Array PolygonPoints);
```

Where :

- PolygonPoints - Name of array with points of polygon.

ParameterAkFromTorGenCss

Function returns value A_k [m²] (area enclosed by center-line of general cross-section for torsion, created by function CreateGeneralCssForTorsion) for check torsion.

Syntax:

```
CONCRETE.Torsion.ParameterAkFromTorGenCss(Double CSIndex, Double ForceIndex);
```

Where:

- CSIndex - Id of cross section
- ForceIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

ParameterUkFromEffRectCss

Function returns value u_k [m] calculated for effective rectangular cross-section.

Syntax:

```
double uk = CONCRETE.Torsion.ParameterUkFromEffRectCss(Double CSIndex, Double EffectWallThickness);
```

Where:

- CSIndex - Id of cross section
- EffectWallThickness - Effective wall thickenss

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

ParameterUkFromPolygon

Function returns value u_k [m] (perimeter of defined polygon) for check torsion.

Syntax:

```
double uk = CONCRETE.StressStrain.ParameterUkFromPolygon(Array PolygonPoints);
```

Where:

- PolygonPoints - Name of array with points of polygon.

ParameterUkFromTorGenCss

Function returns value u_k [m] (perimeter of center-line of general cross-section for torsion, created by function CreateGeneralCssForTorsion).

Syntax:

```
double uk = CONCRETE.Torsion.ParameterUkFromTorGenCss(Double CSIndex, Double ForcelIndex);
```

Where:

- CSIndex - Id of cross section
- ForcelIndex - Id of forces for calculation

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS") and forces have to be set with method SetForcesId (with appropriate ID, viz "SetForces").

PolygonCssForTorsion

Function returns polygon (array of point - y[m] and z[m] coordinate) of thin-walled cross-section for torsion according to selecting type (0-outer polygon, 1-center line polygon, 2-inner polygon).

Syntax:

```
object polygon = CONCRETE.Torsion.PolygonCssForTorsion(Double CSIndex, Double EffectWallThickness, Double EffectWallThickness, Double StirrupID);
```

Where:

- CSIndex - Id of cross section
- EffectWallThickness - Effective wall thickenss
- PolygonType -
 - 0 - outer polygon,
 - 1 - center line polygon,
 - 2 - inner polygon
- StirrupID - Id of stirrup

Cross-section have to be created with method CreateCS (with appropriate ID, viz "CreateCS").

[Load an example: Torsion](#)

Libraries

Please, find the chapter about adding and modifying libraries [here](#).

IDs in standard libraries consist of two parts:

1. **blue part** - target IO node - determines from which library the value should be read;
2. **red part** - indicates which property/parameter from the library should be used (see chapter about IDs in standard libraries).

The screenshot shows a software interface with a table of material properties and a detailed view of a specific property. The table has columns for Description, Symbol, Value, and Unit. The detailed view shows a table with columns for ID, Description, Symbol, Value, Unit, and Precision. A blue box highlights the 'CONCRETE.EC' target ID node in the top right. A red box highlights the 'f_{ck}' symbol in the main table. A blue arrow points from the 'CONCRETE.EC' box to the 'CONCRETE.EC.f_{ck}' ID in the detailed view. A red arrow points from the 'f_{ck}' symbol in the main table to the 'f_{ck}' symbol in the detailed view.

Description	Symbol	Value	Unit
Characteristic compressi...	f _{ck}	2,00	MPa
Characteristic compressi...	f _{ck,cube}	5,00	MPa
Average cylinder compre...			
Average axial tensile str...			
5% fractile of cylinder te...			
95% fractile of cylinder t...			
Elastic modulus			
Concrete strain at chara...			

ID	Description	Symbol	Value	Unit	Preci
CONCRETE.EC.f _{ck}		f _{ck}	12e6		2
CONCRETE.EC.f _{ck,cube}		f _{ck,cube}	15e6		2
CONCRETE.EC.f _{cm}		f _{cm}	20e6		2

Available cross-sections in the Steel section library

The steel cross-section library contains common rolled sections:

- I, IPE, IPN, HEA, HEB, HEM profiles;
- U, UPE, UPN, 2xU boxes;
- Circular tubes, square (box), rectangular sections (cold-formed, hot-rolled);
- L-sections - equal-leg angles;
- L-sections - unequal-leg angles;
- T-sections (rolled or cut from I-sections);
- Circular bars.

List of ID codes

Steel cross sections:

CS.SectionName	Cross section name
CS.SectionImage	Cross section type image
CS.Geometry.FormCode	Cross section type: 0 = NotDefined 1 = I, IPE, HEA, HEB, HEM 2 = Tube 3 = U, UPE 4 = square/ rectangular tube 5 = L 6 = T 7 = bar
CS.Geometry.H	Height
CS.Geometry.B	Width
CS.Geometry.tw	Web width
CS.Geometry.tf	Flange width
CS.Geometry.r1	Diameter of the connection between web and flange
CS.Geometry.r2	Diameter of the flange ends
CS.Geometry.d	Effective height of the web
CS.Geometry.A	Area
CS.Geometry.Avz	Effective shear area
CS.Geometry.Iy	Moment of inertia - y axis
CS.Geometry.Wy	Elastic cross section modulus y
CS.Geometry.Wply	Plastic cross section modulus y
CS.Geometry.iy	Radius of gyration - y axis
CS.Geometry.Iz	Moment of inertia - z axis
CS.Geometry.Wz	Elastic cross section modulus z
CS.Geometry.Wplz	Plastic cross section modulus z
CS.Geometry.iz	Radius of gyration - z axis
CS.Geometry.IT	Torsion moment of inertia
CS.Geometry.lw	Warping constant
CS.Geometry.ss	Length of the rigid part of a flange
CS.Geometry.ay	Buckling coefficient - y axis
CS.Geometry.az	Buckling coefficient - z axis

Concrete cross sections:

CS.Geometry.H	Height
CS.Geometry.B	Width/Effective width
CS.Geometry.th	Web width
CS.Geometry.sh	Slab thickness
CS.Geometry.FormCode	Cross section type - according to Scia Engineer
CS.Reinf.n1	Number of bars on top
CS.Reinf.Cover1	Top bars concrete cover
CS.Reinf.φ1	Top bars diameter
CS.Reinf.n2	Number of bars on bottom
CS.Reinf.Cover2	Bottom bars concrete cover
CS.Reinf.φ2	Bottom bars diameter

New concrete cross sections:

The new concrete cross section library adds some functionality to the old one. It allows to use various cross section shapes, reinforcement templates or stirrup shapes.

[Load the example: ConcreteSectionLibrary.cls](#)

IO values:

CS.Component.Shape.Point	Array of outline points = cross section shape
CS.Geometry	Set of values which defines the cross section shape (dimensions)
Beam.Reinforcement.Bar	Array of bars of longitudinal reinforcement
Beam.Reinforcement.c_min	Min cover of longitudinal reinforcement
Beam.Reinforcement.c_max	Max cover of longitudinal reinforcement
Beam.Reinforcement.StirrupChars	Stirrups characteristics

Stirrup characteristics:

Beam.Reinforcement.StirrupChars.Asw	Area of all stirrups in one layer
Beam.Reinforcement.StirrupChars.ss	Average distance between layers
Beam.Reinforcement.StirrupChars.Asw_ss	Average area of stirrups per meter [m ² /m]
Beam.Reinforcement.StirrupChars.D	Average diameter of stirrup bar
Beam.Reinforcement.StirrupChars.Ns	Number of stirrup bars in one layer
Beam.Reinforcement.StirrupChars.Angle	Average angle of the stirrups (0°= horizontal, 90°= vertical)
Beam.Reinforcement.StirrupChars.Branch	Array of arrays of points which defines the stirrup shape. Example: Beam.Reinforcement.StirrupChars.Branch[0] contains an array of points which defines the shape of the first stirrup (index 0).

The list of ID codes

STEEL.EC.fy	Yield stress
STEEL.EC.fu	Ultimate strength
STEEL.EC.Es	Modulus of elasticity (compression, tension)
STEEL.EC.G	Shear modulus
STEEL.EC. ν	Factor of transverse deformation
STEEL.EC. γ	Density
STEEL.EC. α	Coefficient of linear thermal expansion

The list of codes ID

CONCRETE.EC.fck	Characteristic compressive cylindrical strength
CONCRETE.EC.fckcube	Characteristic compressive cubical strength
CONCRETE.EC.fcm	Mean value of compressive cylindrical strength
CONCRETE.EC.fctm	Mean value of axial tensile strength of concrete
CONCRETE.EC.fctk005	5 % fractile of the tensile strength
CONCRETE.EC.fctk095	95 % fractile of the tensile strength
CONCRETE.EC.Ecm	Modulus of elasticity
CONCRETE.EC.eps_c1	Compressive strain in concrete at limit stress f_c
CONCRETE.EC.eps_cu1	Ultimate compressive strain in concrete at stress f_c
CONCRETE.EC.eps_c2	Compressive strain in concrete at limit strength
CONCRETE.EC.eps_cu2	Ultimate compressive strain in concrete at maximum strength
CONCRETE.EC.eps_c3	Compressive strain in concrete at the limit strengths - according to bilinear diagram
CONCRETE.EC.eps_cu3	Ultimate compressive strain in concrete at maximum strengths - according to bilinear diagram
CONCRETE.EC.n	Exponent according to table 3.1 ČSN EN 1992-1-1

The list of codes ID

TIMBER.EC.fmk	Bending strength
TIMBER.EC.f0k	Tensile strength parallel to grain
TIMBER.EC.f90k	Tensile strength perpendicular to grain
TIMBER.EC.f0k	Compression strength parallel to grain
TIMBER.EC.f90	Compression strength perpendicular to grain
TIMBER.EC.fvk	Shear strength
TIMBER.EC.E0	Modulus of elasticity parallel to grain
TIMBER.EC.E005	5 percent fractile of the modulus of elasticity
TIMBER.EC.E90	Modulus of elasticity perpendicular to grain
TIMBER.EC.G	Shear modulus
TIMBER.EC. ρ	Density

The list of ID codes

Bolt_D	Bolt diameter
Bolt_A	Bolt area
Bolt_Anet	Bolt area on threads
Bolt_fyb	Yield strength of the bolt material
Bolt_fub	Ultimate strength of the bolt material
Bolt_Material	Bolt material (double)

Code editor directives

#region ... #endregion

The pair directives #region and #endregion allow for the code contained within these commands to be collapsed and expanded by using icons (see picture below).

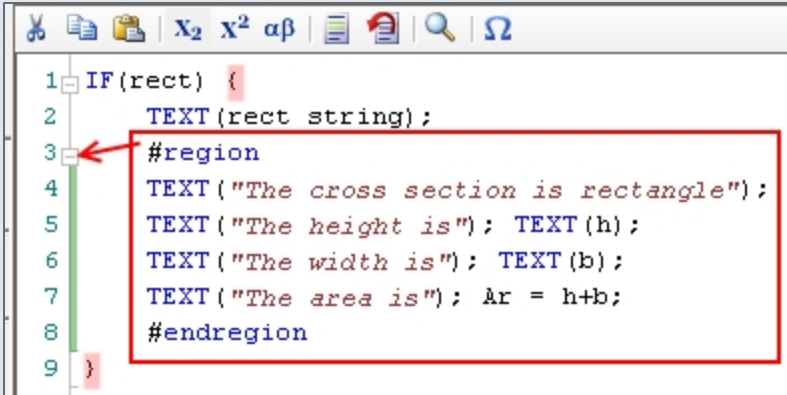
Syntax:

```
#region
```

```
#endregion
```

- the first command is a begin of the region, the second one is end

Example



```
1 IF(rect) {  
2     TEXT(rect string);  
3     #region  
4     TEXT("The cross section is rectangle");  
5     TEXT("The height is"); TEXT(h);  
6     TEXT("The width is"); TEXT(b);  
7     TEXT("The area is"); &r = h+b;  
8     #endregion  
9 }
```

Automatic tests

The BUILDER application enables the user to create sets of automatic tests and use these to check a calculation form in common and extreme cases.

About automatic tests

Automatic tests allows the user to check and correct results in his forms.

- Create a new calculation form;
- Define which variables are the results;
- Define input values by the manager;
- Test the form using the tool.

The automatic tests check if the results of the tested forms remain the same for the same input. The check is done by a simple tool which calculates the form and compares the new results with the original ones.

Create automatic tests

What can be tested

An automatic test is defined by two files. The calculation *.CLC file, and the file with input values and results - format TestIO.

*.TestIO is a file which contains input and output values for the calculations. It is created in the Builder application. If the file is open, it can be edited and viewed outside the SDF applications. It contains a name for the calculation, input values and results for the marked variables.

One test may contain more sets for one calculation. One Test IO can be used for common and extreme situations at the same time.

```

<Root>
<Test Description="C14_200x140_6m,F-2m,3m">
<Input Name="L" Type="Numeric" ESA_ID="" Value="6.000000" />
<Input Name="E" Type="Numeric" ESA_ID="TIMBER.EC.E005" Value="4700000000.000000" />
<Input Name="MIRd-" Type="Numeric" ESA_ID="" Value="9046.153846" />
<Input Name="Caption" Type="String" ESA_ID="" Value="Bracket - timber - triangular increasing force" />
<Input Name="PrintCaption" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintSection" Type="Bool" ESA_ID="" Value="True" />
<Input Name="MEd-" Type="Numeric" ESA_ID="" Value="9000.000000" />
<Input Name="s" Type="Numeric" ESA_ID="" Value="0.994898" />
<Input Name="PrintSchema" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintULS" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintSLS" Type="Bool" ESA_ID="" Value="True" />
<Input Name="H" Type="Numeric" ESA_ID="" Value="0.200000" />
<Input Name="fjm,d-" Type="Numeric" ESA_ID="" Value="9692307.692308" />
<Input Name="kmod-" Type="Numeric" ESA_ID="" Value="0.900000" />
<Input Name="fjm,k-" Type="Numeric" ESA_ID="TIMBER.EC.fmk" Value="14000000.000000" />
<Input Name="yIm-" Type="Numeric" ESA_ID="" Value="1.300000" />
<Input Name="B" Type="Numeric" ESA_ID="" Value="0.140000" />
<Input Name="Ily-" Type="Numeric" ESA_ID="" Value="0.000093" />
<Input Name="wly-" Type="Numeric" ESA_ID="" Value="0.000933" />
<Input Name="a" Type="Numeric" ESA_ID="" Value="2.000000" />
<Input Name="wlb-" Type="Numeric" ESA_ID="" Value="0.101273" />
<Input Name="wlL-" Type="Numeric" ESA_ID="" Value="0.129483" />
<Input Name="ib-" Type="Numeric" ESA_ID="" Value="0.028210" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="1.000000" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="3.000000" />
<Input Name="qd-" Type="Numeric" ESA_ID="" Value="1500.000000" />
<Input Name="qk-" Type="Numeric" ESA_ID="" Value="1000.000000" />
<Input Name="kih-" Type="Numeric" ESA_ID="" Value="1.000000" />
</Test>
<Test Description="C18_400x100_6m,F-3m,3m">
<Input Name="L" Type="Numeric" ESA_ID="" Value="6.000000" />
<Input Name="E" Type="Numeric" ESA_ID="TIMBER.EC.E005" Value="6000000000.000000" />
<Input Name="MIRd-" Type="Numeric" ESA_ID="" Value="33230.769231" />
<Input Name="Caption" Type="String" ESA_ID="" Value="Bracket - timber - triangular increasing force" />
<Input Name="PrintCaption" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintSection" Type="Bool" ESA_ID="" Value="True" />
<Input Name="MEd-" Type="Numeric" ESA_ID="" Value="11250.000000" />
<Input Name="s" Type="Numeric" ESA_ID="" Value="0.338542" />
<Input Name="PrintSchema" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintULS" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintSLS" Type="Bool" ESA_ID="" Value="True" />
<Input Name="H" Type="Numeric" ESA_ID="" Value="0.400000" />
<Input Name="fjm,d-" Type="Numeric" ESA_ID="" Value="12461538.461539" />
<Input Name="kmod-" Type="Numeric" ESA_ID="" Value="0.900000" />
<Input Name="fjm,k-" Type="Numeric" ESA_ID="TIMBER.EC.fmk" Value="18000000.000000" />
<Input Name="yIm-" Type="Numeric" ESA_ID="" Value="1.300000" />
<Input Name="B" Type="Numeric" ESA_ID="" Value="0.100000" />
<Input Name="Ily-" Type="Numeric" ESA_ID="" Value="0.000533" />
<Input Name="wly-" Type="Numeric" ESA_ID="" Value="0.002667" />
<Input Name="a" Type="Numeric" ESA_ID="" Value="3.000000" />
<Input Name="wlb-" Type="Numeric" ESA_ID="" Value="0.025523" />
<Input Name="wlL-" Type="Numeric" ESA_ID="" Value="0.025523" />
<Input Name="ib-" Type="Numeric" ESA_ID="" Value="0.005977" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="0.000000" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="3.000000" />
<Input Name="qd-" Type="Numeric" ESA_ID="" Value="1500.000000" />
<Input Name="qk-" Type="Numeric" ESA_ID="" Value="1000.000000" />
<Input Name="kih-" Type="Numeric" ESA_ID="" Value="1.000000" />
</Test>
<Test Description="C18_50x50_6m,F-3m,3m">
<Input Name="L" Type="Numeric" ESA_ID="" Value="6.000000" />
<Input Name="E" Type="Numeric" ESA_ID="TIMBER.EC.E005" Value="6000000000.000000" />
<Input Name="MIRd-" Type="Numeric" ESA_ID="" Value="323.410917" />
<Input Name="Caption" Type="String" ESA_ID="" Value="Bracket - timber - triangular increasing force" />
<Input Name="PrintCaption" Type="Bool" ESA_ID="" Value="True" />
<Input Name="PrintSection" Type="Bool" ESA_ID="" Value="True" />
<Input Name="MEd-" Type="Numeric" ESA_ID="" Value="11250.000000" />
<Input Name="s" Type="Numeric" ESA_ID="" Value="34.785468" />

```

1. Orange part - first input data
2. Blue part - second input data for the same calculation

Creating the test

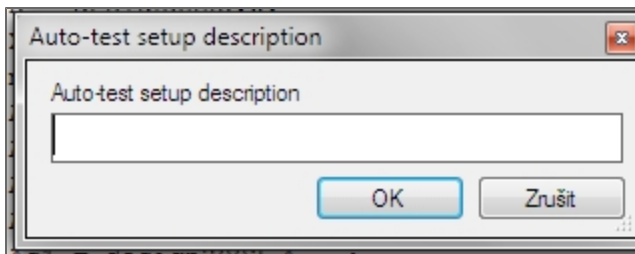
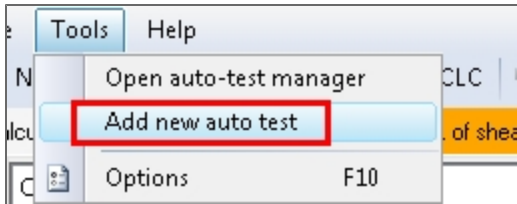
TestIO files are generated in the BUILDER. The default values are used as input and output values.

Results which should be checked must be marked by ID in the table of variables as Result.1, Result.2... The table of variables may contain a random number of results.

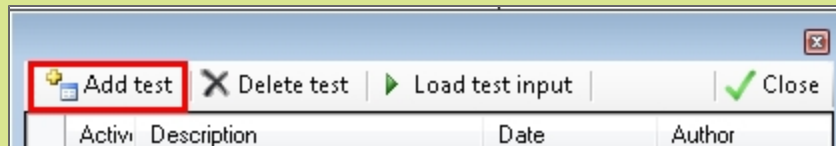
Input values and results can be edited afterwards in the *.TestIO file. The file is not editable from within the BUILDER.

Use Main Menu > Tools > Add new auto test. Set the name for the first set of input values (e.g. Concrete_press_rectangle_check).

The default TestIO file is created automatically when the command is used for the first time. The name of the TestIO is the same as the name of the CLS file.

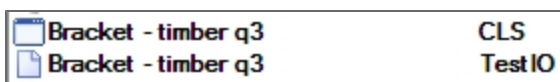


The user can save a new set from the Auto test manager. Use the button "Add test" and the input and output values from table of variables will be saved to the *.TestIO file. The user needs to define a name for the set.



The set name is saved to the *.TestIO file as "Description".

```
Root>
<Test Description='C14,200x140,6m,F-2m,3m'>
  <Input Name="L" type="Numeric" ESA_ID="value"
  <Input Name="E" Type="Numeric" ESA ID="TIMBER
```



The next run of the command will save the next set of input and output values to the same file.

If the file TestIO is deleted, the file is generated automatically again when the command is used. The values from the deleted file are lost.

Auto test manager

The manager loads saved sets from the TestIO file with the same name as the current form.

One row refers to one set of input and output values. A test is defined by its name, data at the moment it was added and author.

Add test

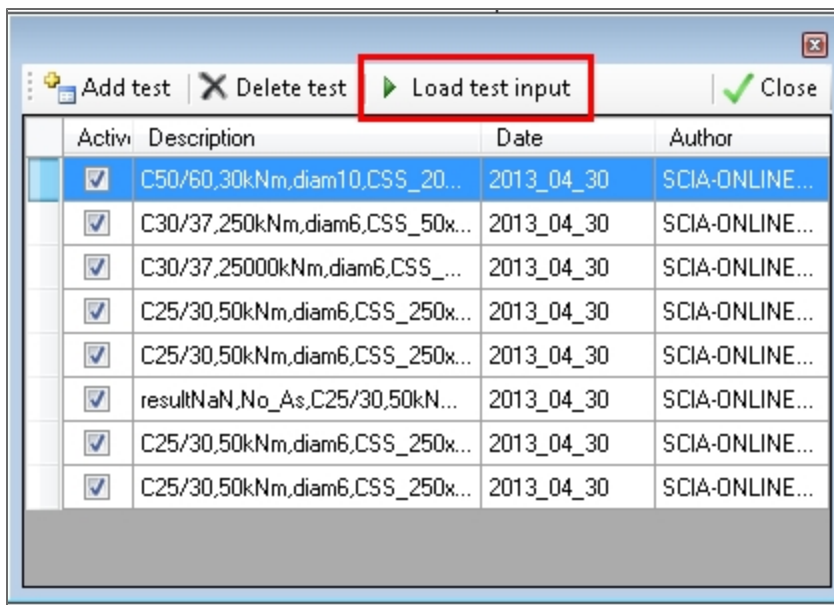
The button "Add test" works in the same way as Tools> Add new auto test set. It adds a new set of input and output values from table of variables to the file *.TestIO.

Edit test

If the test set is added to the TestIO file, then it can be edited directly outside the SDF applications or it can be created again, with changes.

Inserting the inputs and outputs to the table of variables is possible through the Auto test Manager:

1. Open the Auto test Manager;
2. Select the set which should be loaded back to the BUILDER;
3. Use the button "Load test input;"
4. Close the Manager.



Deactivate test set

A test set can temporarily be deactivated. Use the check-box on the corresponding row. The tests which are checked are activated and will run.

The condition of the check-box is visible in the *.TestIO file as value 0 for the item "Active."

```
Calculation_of_crack_width_TestIO - Poznámkový blok
Soubor Úpravy Formát Zobrazení nápověda

<input Name="H1" Type="Bool" ESA_ID="" value="true" />
<input Name="ε11" Type="Numeric" ESA_ID="" value="0" />
<input Name="ε12" Type="Numeric" ESA_ID="" value="797206.2" />
</Test>
<Test Description="C30/37,250kNm,diam6,CSS_50x500mm" Author="SCIA-ONLINE\ztruxova" Active="1" date="2013_04_30">
<Result Name="Δε1sc" Type="Numeric" ESA_ID="Result.0" value="13.41869" />
<input Name="Δ1p" Type="Numeric" ESA_ID="" value="6e-3" />
<input Name="Δ1p1" Type="Numeric" ESA_ID="" value="400e-3" />
<Result Name="k1t" Type="Numeric" ESA_ID="Result.7" value="600e-3" />
<input Name="f1cteff" Type="Numeric" ESA_ID="" value="1.666" />
<input Name="ρ1peff" Type="Numeric" ESA_ID="" value="91.4289" />
<input Name="κ1e" Type="Numeric" ESA_ID="" value="7.77778" />
<Result Name="wk" Type="Numeric" ESA_ID="Result.6" value="1.14074" />
<Result Name="s1mmax" Type="Numeric" ESA_ID="Result.5" value="85.01129e-3" />
<input Name="E1cm" Type="Numeric" ESA_ID="Concrete.EC.Ecm" value="27e9" />
<input Name="E1s" Type="Numeric" ESA_ID="STEEL.EC.E" value="210e9" />
<input Name="ε11" Type="Numeric" ESA_ID="" value="894.4272e-3" />
<Result Name="Δ1ceff" Type="Numeric" ESA_ID="Result.0" value="3.5e-3" />
<input Name="k11" Type="Numeric" ESA_ID="" value="800e-3" />
<input Name="k12" Type="Numeric" ESA_ID="" value="500e-3" />
<input Name="k13" Type="Numeric" ESA_ID="" value="3.4" />
<input Name="k14" Type="Numeric" ESA_ID="" value="430e-3" />
<input Name="b" Type="Numeric" ESA_ID="" value="50e-3" />
<input Name="h" Type="Numeric" ESA_ID="" value="500e-3" />
<Result Name="d1" Type="Numeric" ESA_ID="Result.3" value="472e-3" />
<Result Name="d12" Type="Numeric" ESA_ID="Result.4" value="33e-3" />
<input Name="x" Type="Numeric" ESA_ID="" value="21.9978e-3" />
<input Name="ε" Type="Numeric" ESA_ID="" value="800e-3" />
<input Name="ρ1ceff" Type="Numeric" ESA_ID="" value="70e-3" />
<input Name="r" Type="Numeric" ESA_ID="" value="3.14159" />
<input Name="fctm" Type="Numeric" ESA_ID="Concrete.EC.fctm" value="1.6e6" />
<input Name="n11" Type="Numeric" ESA_ID="" value="0" />
<input Name="Δ1s1" Type="Numeric" ESA_ID="" value="6e-3" />
<input Name="c12" Type="Numeric" ESA_ID="" value="30e-3" />
<input Name="n12" Type="Numeric" ESA_ID="" value="5" />
<Result Name="Δ1s2" Type="Numeric" ESA_ID="" value="6e-3" />
<Result Name="Δ1s2" Type="Numeric" ESA_ID="Result.2" value="141.3716e-6" />
<input Name="E1c" Type="Numeric" ESA_ID="" value="655.957e-3" />
<input Name="M1Ed" Type="Numeric" ESA_ID="" value="250000" />
<Result Name="σ1s1" Type="Numeric" ESA_ID="Result.8" value="2.817931e12" />
<input Name="Caption" Type="String" ESA_ID="" value="uzivatelsky popis" />
<input Name="PrintCaption" Type="Bool" ESA_ID="" value="False" />
<input Name="H1" Type="Bool" ESA_ID="" value="true" />
<input Name="ε11" Type="Numeric" ESA_ID="" value="0" />
<input Name="ε12" Type="Numeric" ESA_ID="" value="163366.5" />
</Test>
<Test Description="C30/37,25000kNm,diam6,CSS_50x500mm" Author="SCIA-ONLINE\ztruxova" Active="0" date="2013_04_30">
<Result Name="Δε1sc" Type="Numeric" ESA_ID="Result.0" value="1341.872" />
<input Name="Δ1p" Type="Numeric" ESA_ID="" value="6e-3" />
<input Name="Δ1p1" Type="Numeric" ESA_ID="" value="400e-3" />
<Result Name="k1t" Type="Numeric" ESA_ID="Result.7" value="600e-3" />
<input Name="f1cteff" Type="Numeric" ESA_ID="" value="1.666" />
<input Name="ρ1peff" Type="Numeric" ESA_ID="" value="91.4289" />
<input Name="κ1e" Type="Numeric" ESA_ID="" value="7.77778" />
<Result Name="wk" Type="Numeric" ESA_ID="Result.6" value="114.074" />
<Result Name="s1mmax" Type="Numeric" ESA_ID="Result.5" value="85.01129e-3" />
<input Name="E1cm" Type="Numeric" ESA_ID="Concrete.EC.Ecm" value="27e9" />
<input Name="E1s" Type="Numeric" ESA_ID="STEEL.EC.E" value="210e9" />
<input Name="ε11" Type="Numeric" ESA_ID="" value="894.4272e-3" />
<Result Name="Δ1ceff" Type="Numeric" ESA_ID="Result.0" value="3.5e-3" />
<input Name="k11" Type="Numeric" ESA_ID="" value="800e-3" />
<input Name="k12" Type="Numeric" ESA_ID="" value="500e-3" />
```

Delete test set

A test set can be deleted in the Manager:

1. Select the test set which should be deleted; it is indicated in blue.
2. Use the button "Delete test";
3. Confirm with 'OK.'

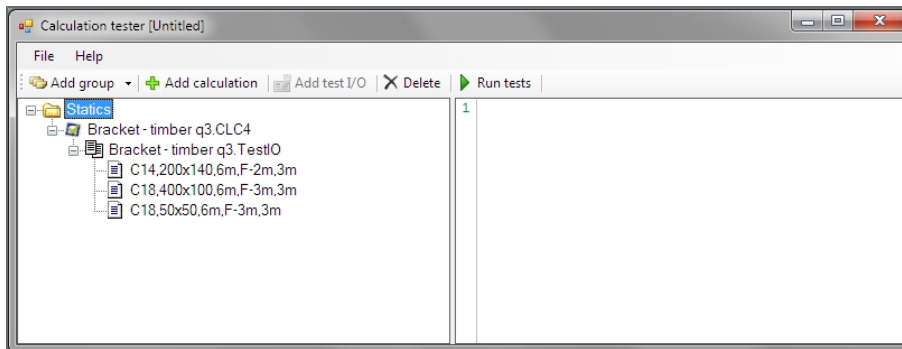
Run and evaluation of the test

The test runs in the separate application Calculation Tester. This application runs the tests and display their results in the left part of the dialogue.

The list of automatic tests can be saved in the XML file.

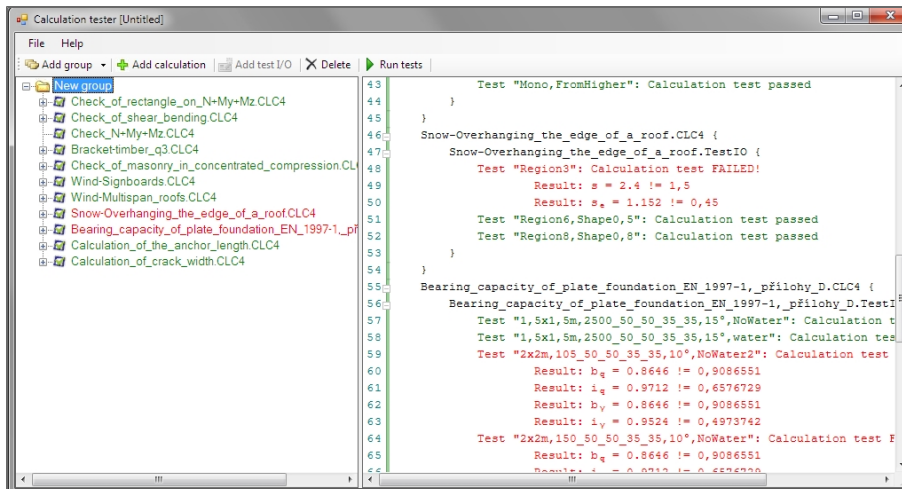
The tree in the right part displays the list of prepared tests. Use buttons on the toolbar to Add group and Add subgroup to the tree. Use Add calculation to add a calculation and its TestIO to the tree. It will be added to the selected group. The application loads CLS (it automatically creates CLC) or CLC. TestIO is automatically loaded to the selected calculation. Use button Add test I/O to add additional TestIO file to already added calculation.

All CLC files and names of calculation sets are displayed in the tree.



Use button Run to run tests. The left part display results of tests. Each set is evaluated separately.

If the test runs OK, it is displayed by the green colour. If the test fails, it is displayed by the red colour. The left part display the wrong result and expected value behind symbol !=.



Important rules when creating a new form

This chapter summarizes all known issues and FAQ on how to create a good calculation form. What should be included, who should handle translations, what about formatting, etc.?

The dialogue

Fill the dialogue in the Builder application. This is one of the things that will be visible in the User application.

Add ALL variables that should be manually defined. Add all libraries which are used.

ID	Description	Symbol	Value
	The compensatory...	ϕ_p	8 mm
	Cross-sectional ...	A_{ps}	0... m ²
		$\sigma_{pe,eff}$	111 MPa
		σ_{cm}	6... MPa
		M_{Ed}	0... kNm
		E_s	210 GPa
		E_{cm}	33 GPa
	Secant modulus of elast...	E_{cm}	33 GPa
	Modulus of elast...	E_s	210 GPa
	Effective area o...	$A_{s,eff}$	0... m ²
	Coefficient of b...	k_1	0.8
	Coefficient of s...	k_2	0.5
	Coefficient	k_3	3.4
	Coefficient	k_4	0...
	Cross section width	b	200 mm
	Cross section he...	h	300 mm
		d_1	271 mm
		d_2	34 mm
		x	4... mm
	Ratio of bond st...	ξ	0.8
		$\sigma_{s,eff}$	7... MPa
		σ	3... MPa
		f_{ctm}	2.9 MPa
	Reinforcement cover	c_1	25 mm

Long calculation

A long calculation is usually messy. Try to split it to more separate forms or use lots of conditions.

Example of conditions:

- IF (material) {...code which displays material characteristics...} - the end-user can decide if the material characteristics are visible by using a check-box in the dialogue;
- IF (pictures) {...code which displays image...} - the end-user can decide whether a picture should be visible (this condition can be applied in many places in the code);
- IF (remark) {...code which displays additional information about the calculation...} - the user can decide if the additional notes and remarks should be visible.

Example of splitting:

- `object load = LoadExternCLC("Load_calculation.clc");` - this code loads a separate form named "Load_calculation.clc" to the current form;
- `load.Calculate(false);` - the variables from the external *.CLC are taken as defined there; these will not be overwritten by

the values in the current calculation;

- or load.Draw(true); - the external calculation is displayed in the current form.

ID

The column 'ID' in the table of variables was originally meant for material ID.

Write CS.Geometry.H in the column 'ID' and this variable will be connected to the cross-section height in the library Steel steel-sections. The end-user should only define the library item by selecting with the arrows in the library component in the dialogue; not all parameters which are connected to the library by ID.

ID	Description	Symbol	Value	Unit
STEEL.EC.E	Modulus of elasticity of reinforcement	E_s	210	GPa
CONCRETE.EC.fctm		f_{ctm}	2.9	MPa
CONCRETE.EC.Ecm	Secant modulus of elasticity of concrete	E_{cm}	33	GPa

ID	Description	Symbol	Value	Unit
Bolt_fyb		f_{yb}	640	MPa
Bolt_fub	Strength of bolts	f_{ub}	800	MPa
Bolt_D	Bolt diameter	d	16	mm
Bolt_Anet		A_{net}	157	mm ²

Translation

Split texts

The parts that should be translated should be separated from parts that do not need to be translated.

Example:

Replace this:

```
TEXT("N↓Ed←->N↓c,bal← => " & VAL(N↓Ed←-/1000, 2) & " kN > " & VAL(N↓c,bal←-/1000, 2) & " kN => The case with prevailing compression");
```

With this:

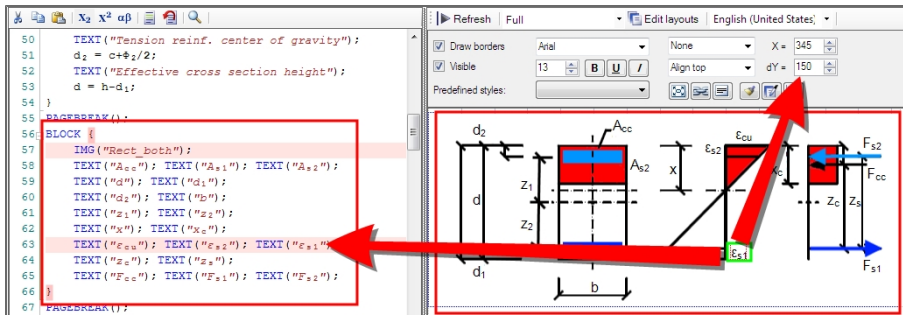
```
TEXT("N↓Ed←->N↓c,bal← => " & VAL(N↓Ed←-/1000, 2) & " kN > " & VAL(N↓c,bal←-/1000, 2) & " kN");
TEXT("=> The case with prevailing compression");
```

Pictures

Text is created by the TEXT command. If it is included as a picture, it will not be possible to translate it afterwards.

Text can be moved on a picture by formatting in the layout; it is possible to numbers pictures consecutively or add captions under pictures.

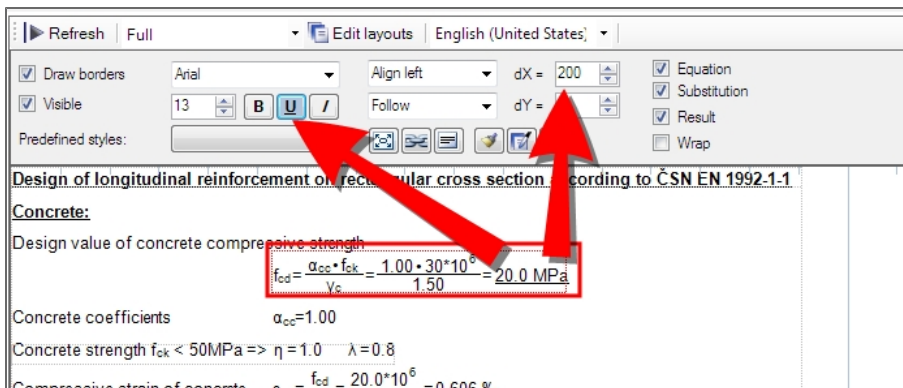
If text is placed on a picture, define the command TEXT under the command IMG; this prevents texts from overlapping. Group the picture and text in a BLOCK command. This ensures that the text moves together with the picture.



Formatting

It is never necessary to use an empty text box (TEXT (" "); TEXT ("
");) for formatting reasons.

Each generated component can be formatted by its properties - offsets, alignment, etc.

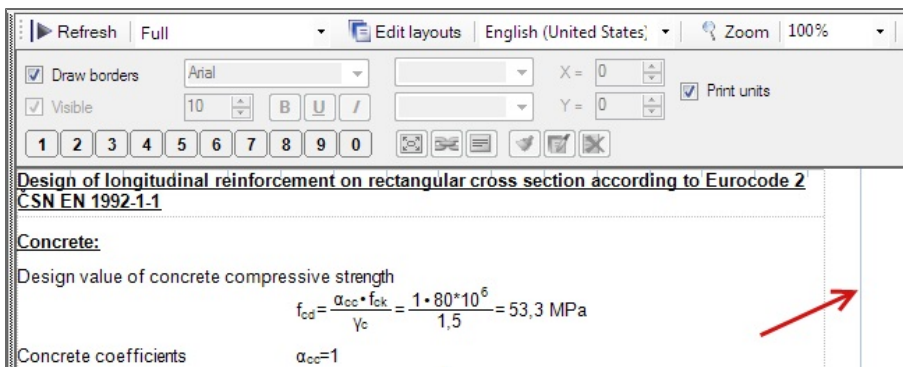


Shortkeys can be used for font type settings (CTRL+B, CTRL+I, CTRL+U).

Page width

The vertical blue line in the layout is the page edge; if something is behind it, it won't be visible correctly on e.g. A4.

The setting for this line is located in Main Menu > Tools.



NaN

If a value is displayed in red (as NaN (Not Any Number)), it means that there is some mathematical error (dividing by zero, etc.).

Predefined styles

Formatting is quicker with predefined styles. There are 8 styles which are used in the predefined forms certified by Nemetschek Scia.

Style	Horizontal align	Vertical align	Font				Equation			
			Font	Bolt	Underline	Italic	Equation	Substitution	Result	Wr- ap
Standard 1	Align left 0	Follow 5	Arial 13				X	X	X	
Standard 2	Align left 10	Follow 5	Arial 13				X	X	X	
Headline 1	None 0	Follow 0	Arial 13	X	X		X	X	X	
Headline 2	None 0	Follow 10	Arial 13	X	X		X	X	X	
Formula 1	None 200	On line 0	Arial 13				X	X	X	
Formula 2	Follow 30	On line 0	Arial 13						X	
Remark 1	None 50	Follow 5	Arial 13			X	X	X	X	
Remark 2	Follow 5	On line 0	Arial 13	X		X	X	X	X	

Some additional information about the USER application

One or more forms in the USER application can be bundled in a PROJECT. The project is created automatically when the USER application is started. This project is saved as a *.CLP file (the CLP contains calculation forms and values).

When the *.CLP file is open and some used forms are missing (or e.g. a *.DEFAULT file is missing), a warning message appears. It is recommended to check the project for missing forms and correctly loaded values in the dialogue.

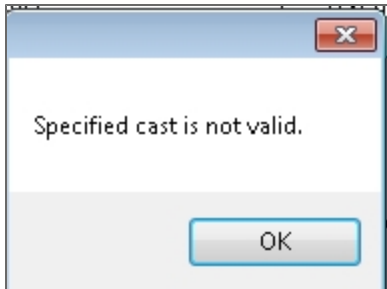
The button 'Open project' cannot load *.CLC files, only *.CLP files.

FAQ for the Builder application

Error messages displayed in Layout editor (error in the source code or a logic error in calculation) [are here](#).

Messages

1) The syntax of a condition is wrong. For example, there is only one AND - IF ($N < 0 \ \& \ ABS(M_y) < 10$)



2) Syntax error - when some keyword is used as variable name (concrete, min, power, ...)

FAQ

Q: How to define variable types?

A: Each variable of the type Double, String and Boolean is created automatically or manually; it stays in CLS until it is deleted by the user.

Variables of the structured type are created every time the CLS is opened; these do not stay in CLS and for this reason the definition of structured variables has to remain in the source code.

Q: What is the difference between the TRUE and FALSE parameters. (commands Calculate() and Draw())

A: True synchronizes values of variables with the same symbol. False will not do that.

Q: How to make some variable invisible in the Dialogue

A: Use the command `Dialog.GetComponentByName("<name of component>").Visible = false;`

The syntax may look like this:

```
IF(select == 0) { Dialog.GetComponentByName("T1").Visible = true;
Dialog.GetComponentByName("T2").Visible = false;
Dialog.GetComponentByName("T3").Visible = false;
}
```

- when the condition is fulfilled then only variable T1 is visible in the dialogue, T2 and T3 are invisible.

Q: How to write the math constant without using manually defined variable?

A: Use [math class](#).

Q: How to convert the number to string?

A: Use command `VAL` or `.ToString` (<http://msdn.microsoft.com/cs-cz/library/3hfd35ad.aspx>).

Q: How to find a problem in the code?

A: Use [Trace listener](#).

Q: How to create a coloured text?

A: Use the graphic command [DrawText](#).

Q: How to work with values for external XML file?

A: Use command [Custom data tables](#).

Q: How to create two lines in the Table to layout?

A: Use command
 to create a new paragraph in the text line. Be very careful with using it and always check the translation string with this command.

Example: T[0][1].Value = "c↓user←
[mm]"; - unit is displayed on the second row in one cell.

Q: How to display the long equation when it doesn't fit to page?

A: Substitute some part of the equation by one more equation which will be displayed above or write the equation to text.

<pre> TEXT("Foundation base bearing capacity"); R24 = c1*Nq*Bz*sz*ie+q1*Nq*Bz*sz*ie+0.5*Y1*Bz*Ny*sz*iy; TEXT(" = " & c1*Nq*Bz*sz*ie & " + "); TEXT(" = " & q1*Nq*Bz*sz*ie+0.5*Y1*Bz*Ny*sz*iy & " = "); TEXT (VAL(R24*10⁻³), 3) & " kPa"); </pre>	<pre> Foundation base bearing capacity R24 = c1*Nq*Bz*sz*ie+q1*Nq*Bz*sz*ie+0.5*Y1*Bz*Ny*sz*iy = 0*8.34*0.898*1.12+0.31* +150000*2.47*0.939*1.07*0.589+0.5*20000*0.8*0.519*0.939*0.871*0.42 = 221.9 kPa </pre>
---	---

Error messages

Missing semicolon

Error message: Syntax error, unexpected ...

Underlined code: The command without semicolon and the next command

Cause: Missing semicolon behind the command

Solution: Add the semicolon

Missing end curly bracket

Error message: Syntax error, unexpected EOF (= End Of Code)

Underlined code: The part of the code after the error

Cause: One or more curly bracket are missing

Solution: Add the curly end bracket to the block end or where it is missing.

Tip: Each block is displayed with left offset.

Wrong variable type

Error message: Compiler error: Variable "... " is not a valid type!

Underlined code: Wrong command

Cause: Variable is not defined or it has the wrong type

Solution: Change the variable type

or

Change the right part of the equation, so the original type will be suitable

Common error in syntax

Error message: Syntax error, unexpected ... , expecting ...

Underlined code: Wrong command

Cause: Defined symbol cannot be used as it is - syntax error

Solution: Fill the missing symbol, replace the wrong symbol

The list of potential symbols is on the end of the error message: "expecting...".

Common error in syntax

Error message: Syntax error, unexpected error

Underlined code: Wrong command

Cause: Common syntax error

Solution: Fill the missing symbol, e.g. end bracket

Error displayed in the calculation

Numeric variable with value NaN

Error message: NaN

Cause: A mathematical formula is not calculated due to an error argument of the function

Example: Dividing by zero, Square root of a negative number etc.

Solution: Check the input values of the mathematical formula

Unknown variable

Error message: The given key was not present in the dictionary.

Cause: SDF doesn't recognize the variable type from the code, the variable was not automatically generated in the table of variables

Solution: Define the variable type manually in the code

or

Add the variable manually to the table of variables

External reference was not found

Error message: Could not find file "...".

Cause: Searched file is not in the defined location.

Solution: Check if the file has been saved.

Check if the external file is saved as CLC

Check if external file is in the same folder as the current one, or if it is in the defined location

Remark: If no path is defined, SDF searches in the folder where the current CLS / CLC is saved. If the current CLS is not saved, the empty path is used!

Uninitialized object

Error message: Object reference not set to an instance of an object.

Cause: Initialization of the object is not done, or it failed.

Solution: Check if a command enables initialization of the object.

Follow the error messages about the initialization.

Unknown method

Error message: No method '...' accepts parameters (...)!

Cause: the object does not support the method which accepts the defined parameters.

Solution: Check the name of the method (including upper / lowercase)

Check the method parameter types.

Unknown property

Error message: ... - not such a field or property

Cause: The object doesn't support the defined property.

Solution: Check the name of the property (including upper / lowercase)

Unknown property

Error message: Exception has been thrown by the target of an invocation

Cause: Calling of the object method caused error.

Solution: Check the method parameters.

Additional packages and applications

The Scia Design Forms functionality is enlarged by the set of additional forms packages and applications.

Each set is protected by the dynamic licence and may be bought and installed separately.

Forms packages:

- the package has a special licence number, it will run with special dynamic protection
- it may be opened by standard SDF User application
- each package is determined to a specific group of calculations (piles, steel connections, etc.)

Separate application:

- the application requires a standard installation
- it is protected by a special licence number, it will run with special dynamic protection
- the separate application is determined to a design or check of one particular use case (e.g. [the design and check of the concrete cross section](#))

Installation/Un-installation of additional package

Installation

The additional package contains only CLC files. These are forms which may be opened by the standard SDF User application.

Installation process:

1. Run the setup.
2. Set the setup language.
3. Go through the wizard and install the forms to the computer.

The forms are installed to the folder c:\Users\Public\Documents\DesignForms_*\Forms\.....

Un-installation

Each package may be un-installed or just deleted from the folder.

Installation/Un-installation of additional application





Installation

The additional application is based on its platform - SDF User application.

There are three possibilities of installation procedure:






1. User doesn't have SDF User application installed yet
 - a. Run the setup.
 - b. The platform SDF User is installed to the computer to c:\Program Files (x86)\SCIA\DesignForms*\.
 - c. The additional application is installed to the root Scia folder: c:\Program Files (x86)\SCIA\, next to the platform folder.
 - d. Start the additional application by icon from the desktop or by *.exe from the installed folder.

Example with Scia Concrete Section:

 DesignForms5	5.12.2013 19:11	File folder
 Engineer2013.0	17.10.2013 13:18	File folder
 Engineer2013.1	4.12.2013 13:46	File folder
 SciaConcreteSection1.0	4.12.2013 14:23	File folder

2. User has SDF User application installed yet. The additional application requires the same version of the User application.
 - a. Run the setup.
 - b. The additional application is installed to the root Scia folder: c:\Program Files (x86)\SCIA\, next to the platform folder.
3. User has SDF User application installed yet, but not the required version.
 - a. Run the setup.
 - b. The required version of SDF platform is installed to the computer to c:\Program Files (x86)\SCIA\DesignForms*.
 - c. The additional application is installed to the root Scia folder: c:\Program Files (x86)\SCIA\, next to the platform folder.

Example with Scia Concrete Section and old version SDF 4.0:

 DesignForms4	10.10.2013 10:30	File folder
 DesignForms5	5.12.2013 19:11	File folder
 Engineer2013.0	17.10.2013 13:18	File folder
 Engineer2013.1	4.12.2013 13:46	File folder
 SciaConcreteSection1.0	4.12.2013 14:23	File folder

Un-installation

The additional application may be un-installed in two different ways:

1. Un-install just the additional application:
 - a. Go to un-installation dialogue.
 - b. Click to un-install the additional application.
 - c. The additional application is un-installed, the platform stayed.
2. Un-install Scia platform:
 - a. Go to un-installation dialogue.
 - b. Click to un-install the SDF platform.
 - c. The SDF platform with additional application is un-installed.

Installation Scia Concrete Section

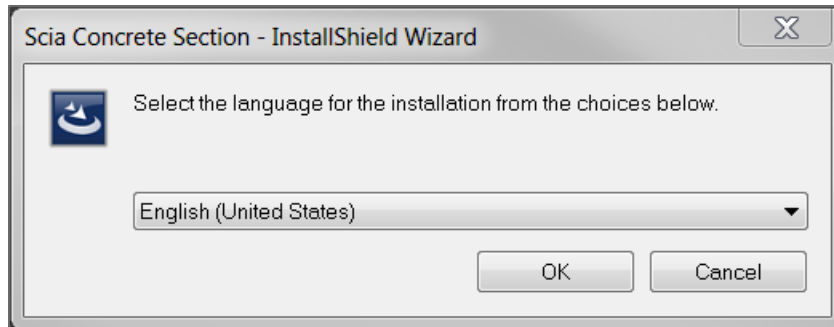
The installation procedure includes:

- Installation of Scia Concrete Section;
- Installation of Scia Design Forms;

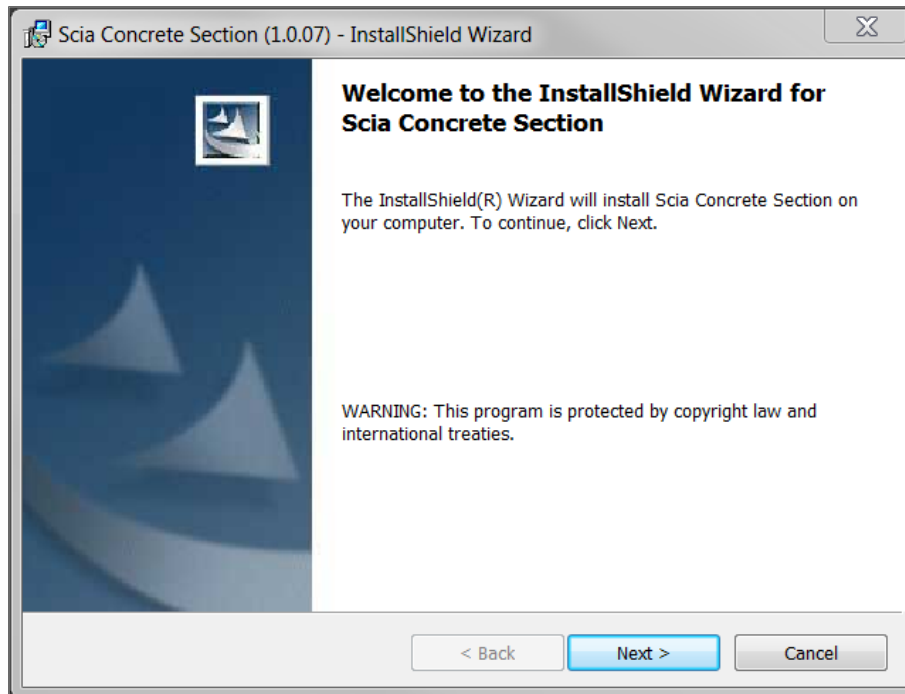
- Installation of FlexNET Scia Licence Server or Sentinel Protection Installer (if required);
- Licence activation.

Software installation

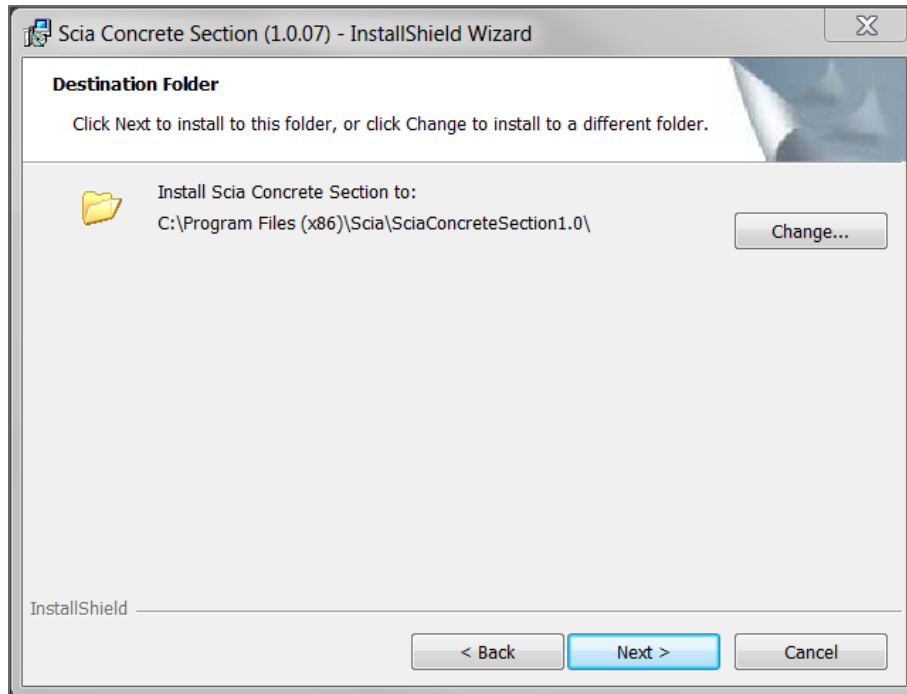
1. Download and run **SciaConcreteSection_setup.exe**.
2. Select the installation language from the dialog:



3. The 'Welcome' screen appears, click **[Next]** to continue.



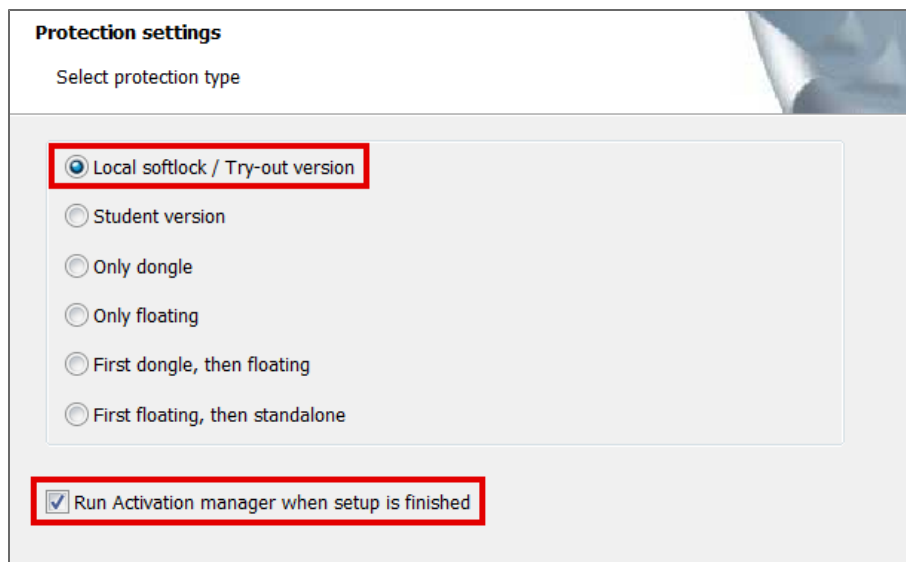
4. Select the Destination Folder for installation of Scia Concrete Section files. It is recommended to keep the default installation path X:\Program Files\.



Remark: In case you already have a version of Scia Design Forms installed on your pc, the path cannot be changed.

Click **[Next]** to continue.

5. Select the default values for the protection in the **Protection settings** dialog. These can always be adapted after installation, if required.
 - a. You received a commercial **softlock** licence or a **tryout** licence:
 - Select 'Local softlock / Tryout version'.
 - Check the option 'Run Activation manager when setup is finished'.



- a. You received a commercial **hardlock** licence (dongle):
 - Select 'Only dongle'.

Protection settings

Select protection type

Local softlock / Try-out version

Student version

Only dongle

Only floating

First dongle, then floating

First floating, then standalone

- a. You received a commercial **network** licence (floating):
Select 'Only floating'.
Fill in the path to your licence server: port number@server name.

Protection settings

Select protection type

Local softlock / Try-out version

Student version

Only dongle

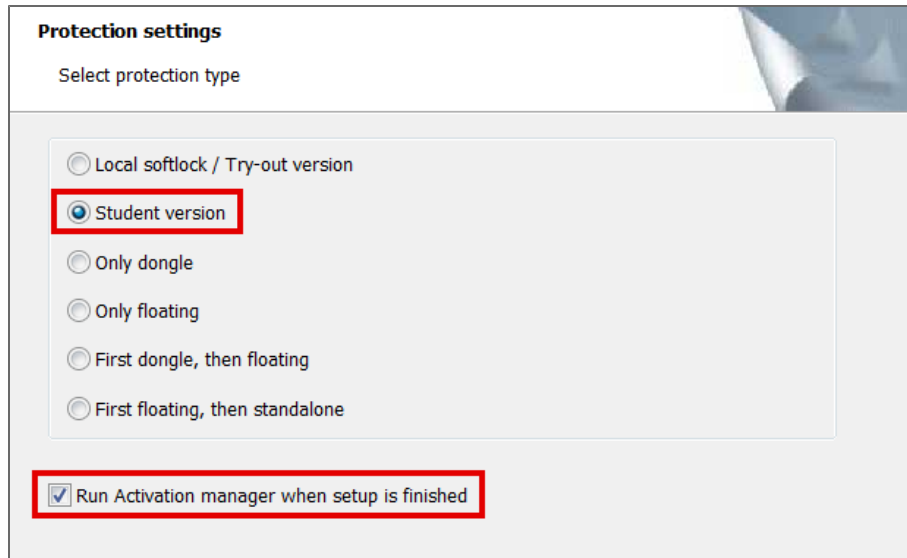
Only floating

First dongle, then floating

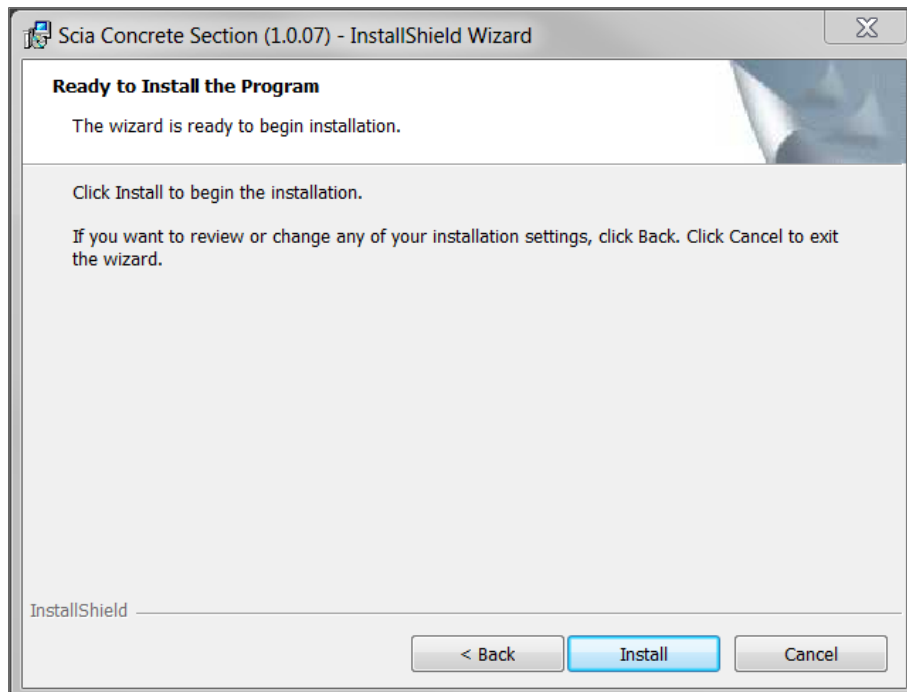
First floating, then standalone

Licence server:

- a. You received a **student** licence:
Select 'Student version'.
Check the option 'Run Activation manager when setup is finished'.



6. Click **[Next]** to continue.
7. Click **[Install]** to start the installation process.

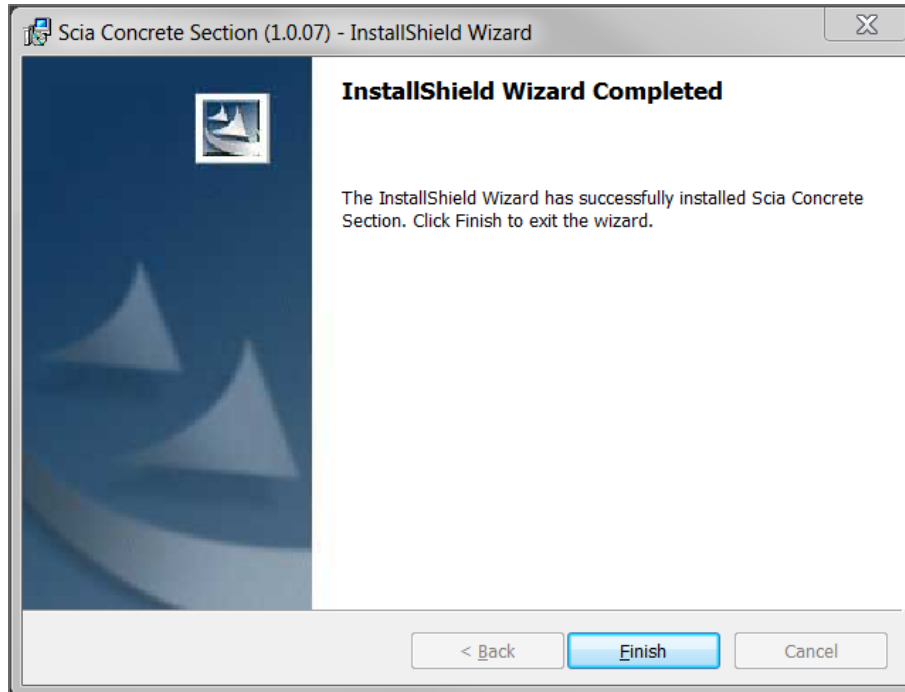


Next to Scia Concrete Section, also Scia Design Forms is installed from scratch, or upgraded in case you have this software already installed on your pc.

FlexNET Scia Licence Server is installed, in case softlock, tryout, or student licence has been selected, and you don't have this software yet installed on your pc.

Sentinel Protection Installer is installed, in case hardlock licence (dongle) has been selected, and you don't have this software yet installed on your pc.

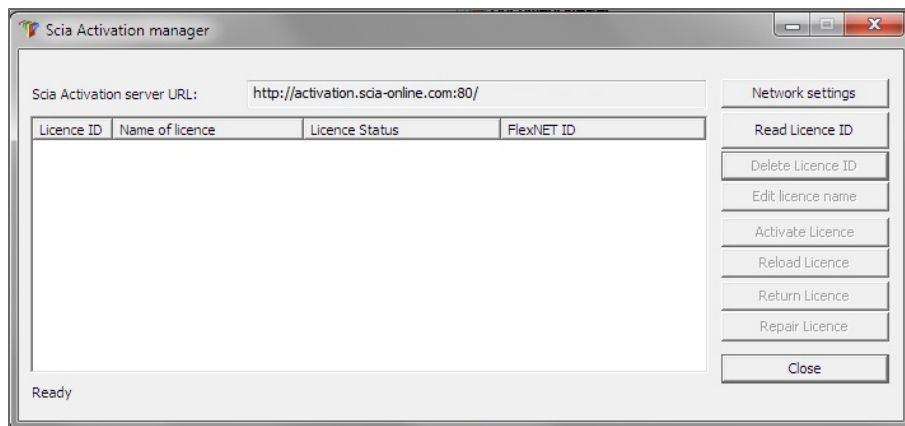
8. The next screen appears to confirm a successful installation. Click **[Finish]** to end the installation procedure.



9. Continue with the licence activation as described in the next chapters; the procedure differs per licence type.

Licence activation - softlock, tryout, student licence

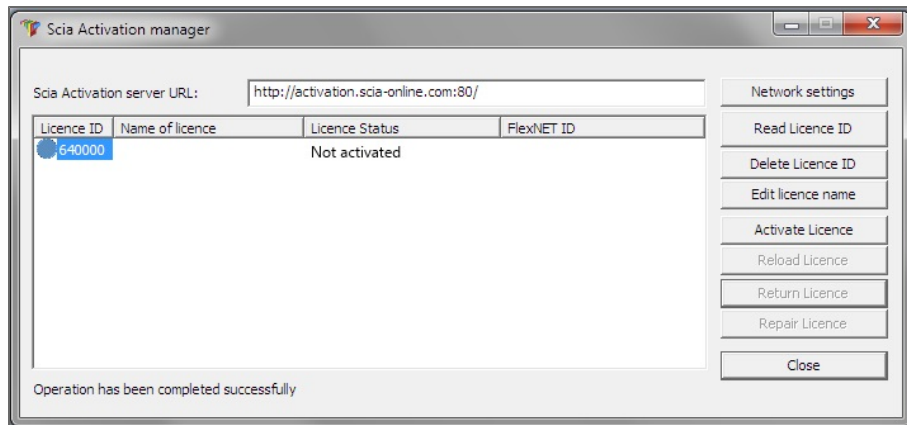
1. The **Scia Activation manager** dialog appears on your screen, at the end of the installation procedure.



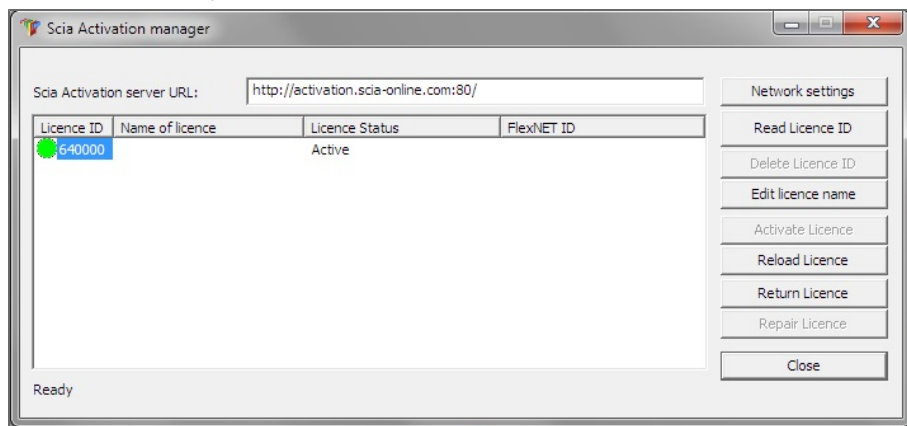
If this would not be the case, go to Windows Start menu > All Programs > Scia Licence Server > Activation Manager, and start the application. A shortcut with icon should be available on your desktop as well.

2. You received a *.LID file by e-mail, containing your authorization code. Download this file to your hard disk.

- Click **[Read Licence ID]** in the Scia Activation manager, and select the SCIAxxxxxx.LID file (xxxxxx representing your licence number). Click **[Open]**. Your licence number appears in the list.



- Select the licence number and click **[Activate licence]**. The circle in front of your licence number turns green when the licence has been correctly activated.



Click **[Close]**.

If you want to use the licence file on another computer, it is necessary to first deactivate it on the current one:

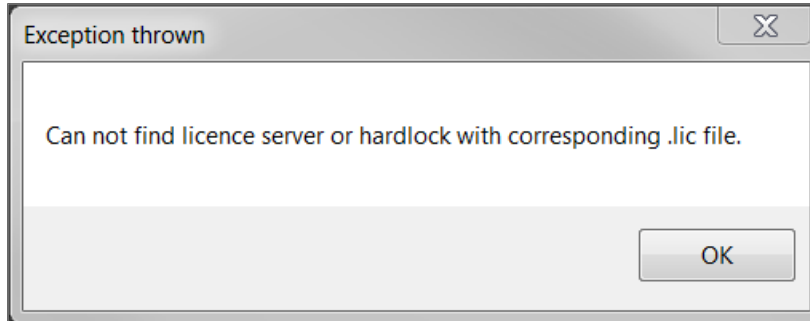
- Run the **Activation manager**.
- Select the licence number and click **[Return licence]**.
- Click **[Close]**.

- Now Scia Concrete Section should run correctly without further intervention.

Licence activation - hardlock (dongle), network (floating) licence

Scia Concrete Section should run correctly without further intervention.

However, in case of a protection problem, you will receive next message when starting the application:



Go to Windows Start menu > All Programs > Scia Design Forms x > Protection Setup SDFx (x representing the current SDF version). Open the **Protection Setup**, check the selected protection type and change it if required. Click **[Apply / Refresh]** and close with **[OK]**.

First issue handling - hardlock (dongle) licence

1. Connect the hardware key (dongle) to your pc, and activate your internet connection.
2. In the **Protection setup**, click **[Import licence file]**. The licence file will be loaded from the Nemetschek Scia server.
3. Click **[OK]** and start the application.
4. In case the protection problem persists, please consult the [Protection page](#) of the Scia webhelp.

First issue handling - network (floating) licence

On the server pc:

New installation

1. Install the FlexNET Scia Licence Server.
2. Activate your *.LID file in the **Scia Activation manager**.

Update

1. Reload your *.LID file in the **Scia Activation manager**.
2. Restart the FlexNET Licensing Service.

On the client pc:

1. In the **Protection setup**, check the licence server path (port number@server name).
2. Click **[OK]** and start the application.
3. In case the protection problem persists, please consult the [Protection page](#) of the Scia webhelp.

Scia Concrete Section

Scia Concrete Section is a stand-alone application for the verification of reinforced concrete cross-sections, developed within the Scia Design Forms environment. It offers a wide range of checks at the ultimate and serviceability limit states, in compliance with EN 1992-1-1. All checks are supported for predefined as well as arbitrary cross-section shapes, subject to uniaxial or biaxial bending in combination with normal and shear forces. Results are displayed in a numerical as well as a graphical way, including dynamic images of interaction diagrams and stress / strain distributions.

Scia Concrete Section is a fast and intuitive calculation tool that is bound to make your daily work more efficient and pleasant. Quick check or detailed analysis? Your choice!

For more information, have a look at this [Factsheet](#).

CLS examples

Here is the list of source code (CLS files) examples which are used in the SDF setup:

[Statics / Calculation of single force on the steel bracket](#)

[Concrete / Calculation of the concrete cover](#)

[Geotechnics / Calculation of the geostatic stress](#)

[Masonry / Check of masonry in compression](#)

[Steel / Check of tension on the steel cross section](#)

[Timber / Check of tension on the timber cross section](#)

[Wind / Calculation of the wind pressure on canopies](#)

Examples of forms (CLS) which are more complicated

The special group of forms may be downloaded from this page. Examples of source code which solves some more complicated issues or problems, special workarounds etc.

Selection from CustomDataTable

Load the example of CustomDataTable (XML file for Custom library). It is common usage of CustomDataTable and user is able to create a data selection.

Usage of Custom data table on table 4.1 from the NBR code:

Type of joint		Chords	Verticals	Diagonals
Gap joints	K type	1.5	1.0	1.3
	N type / KT type	1.5	1.8	1.4
Overlap joints	K type	1.5	1.0	1.2
	N type / KT type	1.5	1.65	1.25

Source code in CLS and compiled version CLC:

[2D 3D Table selection as function.CLC](#)

[2D 3D Table selection as function.CLS](#)

And the libraries:

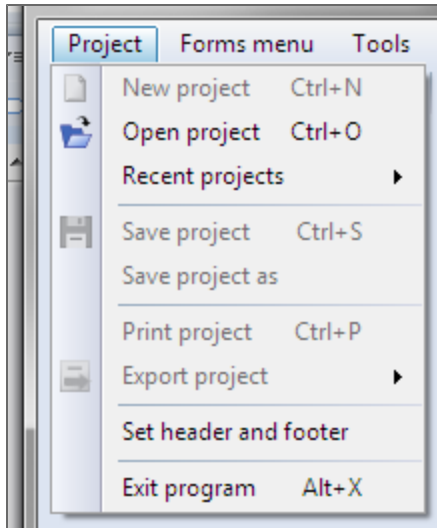
[Table4.1_k1factors.xml](#)

[2DTableSelection.xml](#)

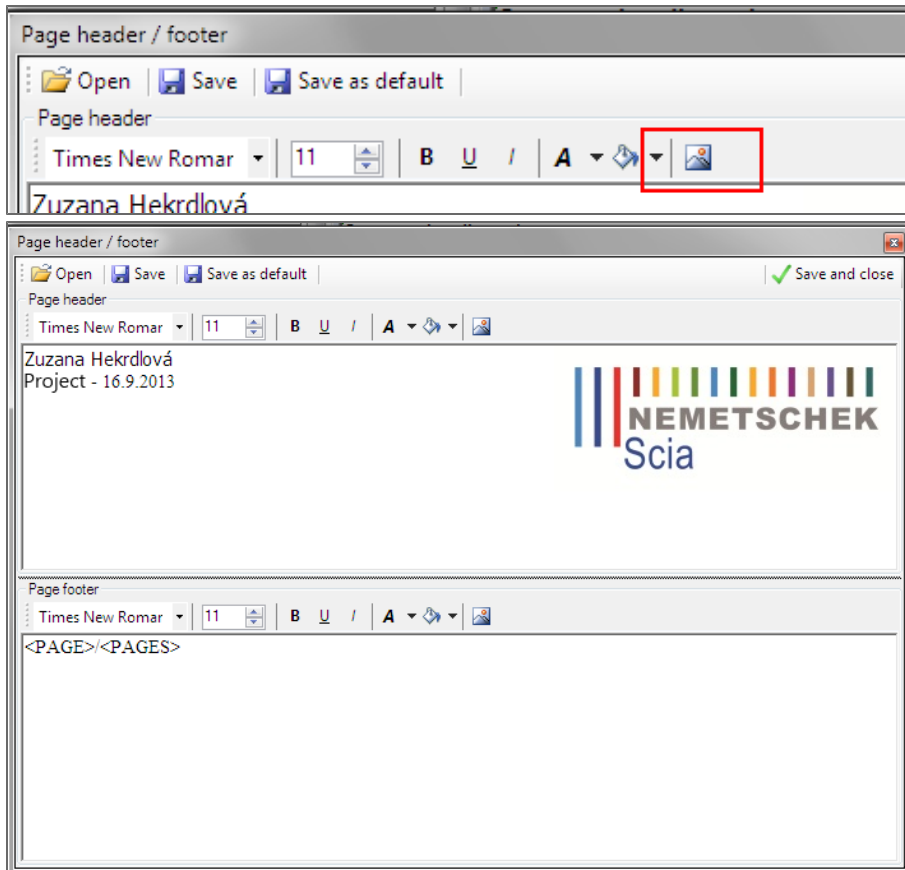
[3DTableSelection.xml](#)

How to create header and footer in User application

1. Open User application.
2. Go to main toolbar / Project / Set header and footer:



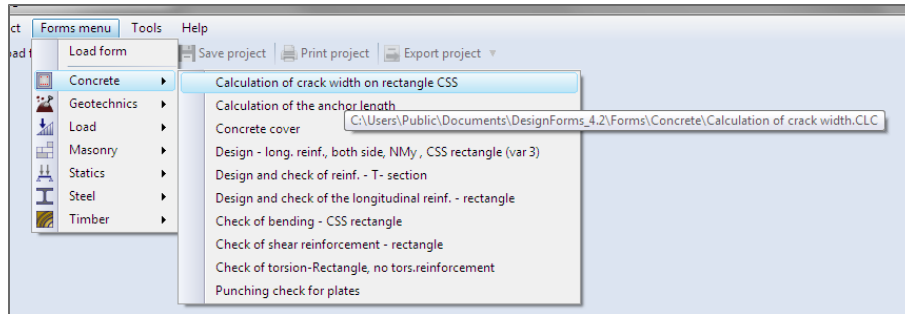
3. Add required name, description or logo to the header. And active items <PAGE> and <PAGES> to the footer.
Add picture by clipboard (copy and paste) or using a button.



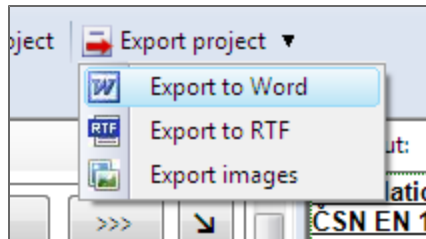
The active item is filled automatically by the application. See more info [here](#).

4. User button Save and close.

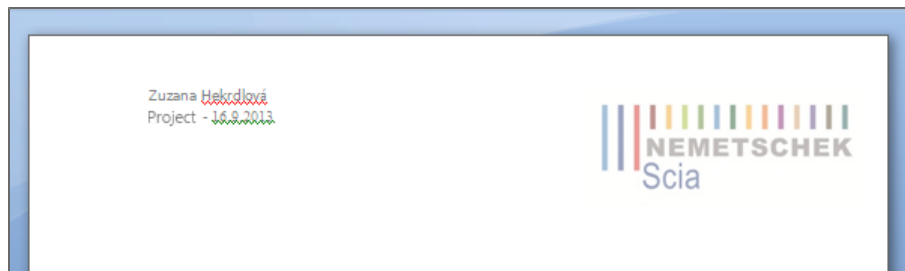
5. Load one form from Forms menu to the User application.



6. Start export to DOCX format.



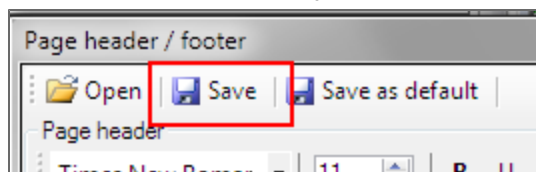
7. Check the header in the generated document.



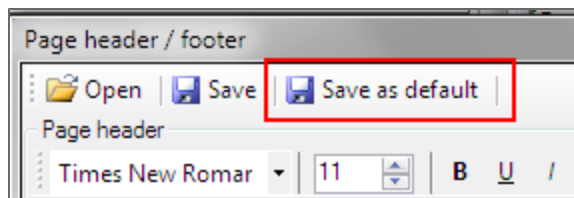
8. Check the footer in the generated document.



9. Open the Set header and footer tool again. Save the header and footer to the file on the driver. The format is .HF.



10. Adapt the header and footer.
11. Save it as a different file.
12. Use button Open and select the previous HF file.
13. Save it as default.



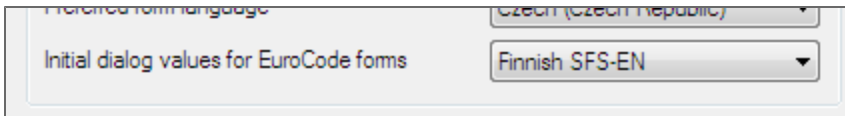
14. Check that exported file will display the correct header and footer.

15. Close the application and reopen it again. The default header and footer should be used again.

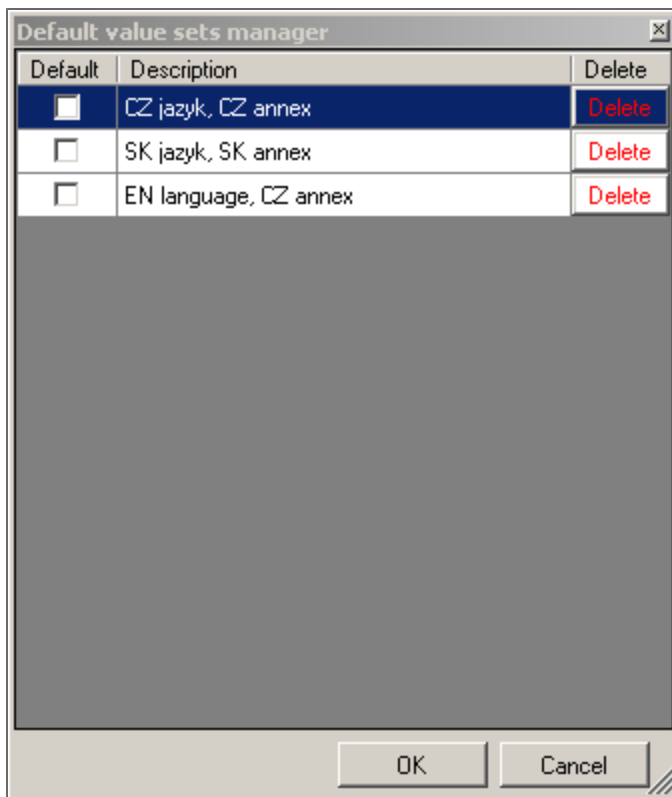
How to use Dialog default in User application

This tutorial describes what is defined by the [.DEFAULT file](#).

1. Open the User application. Go to main toolbar / Tools / Options. Set the default NA on Finnish.

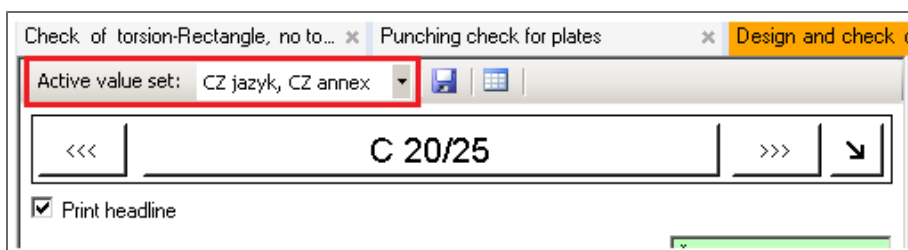


2. Go to main toolbar / Forms menu and select some form from the list.
3. The Default value set manager appears, because the Finnish annex is not defined in the .DEFAULT.

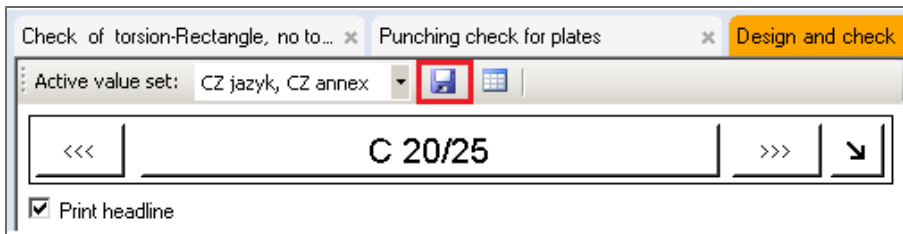


Select one default set by the checkbox.

4. The default set (the original or another) may be loaded by the combo-box on the Dialogue toolbar.



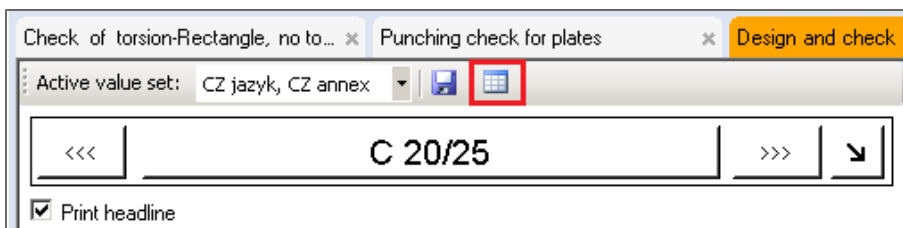
- Change some values in the Dialogue. The name of the default set disappears, because it is not longer valid. Click on the icon "save default values set".



- New dialogue appears. Define the name to the new default value set and confirm.
- The current dialogue set is now saved into .DEFAULT file. Change some values in the dialogue again.
- The name disappears, go to the combo-box and select the last saved set. The saved values are loaded back to the dialogue.

The dialogue set contains library, numeric, text and boolean values of variables - all what is visible in the dialogue.

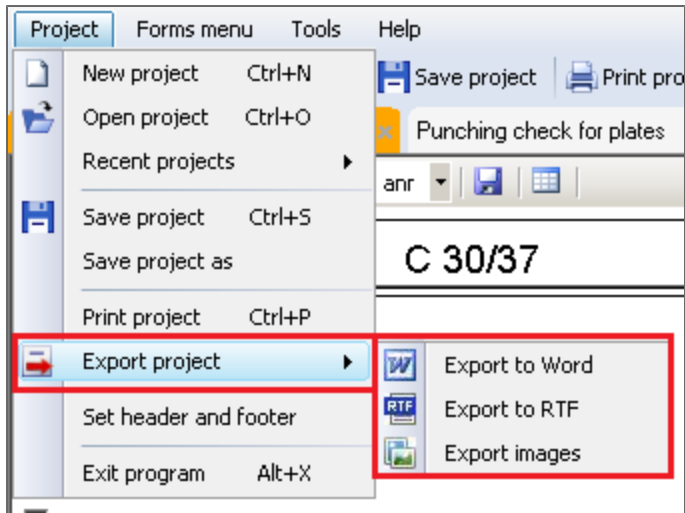
- Each set may be selected, renamed or deleted by the Default value set manager. Start is by icon next to the Save default value set.



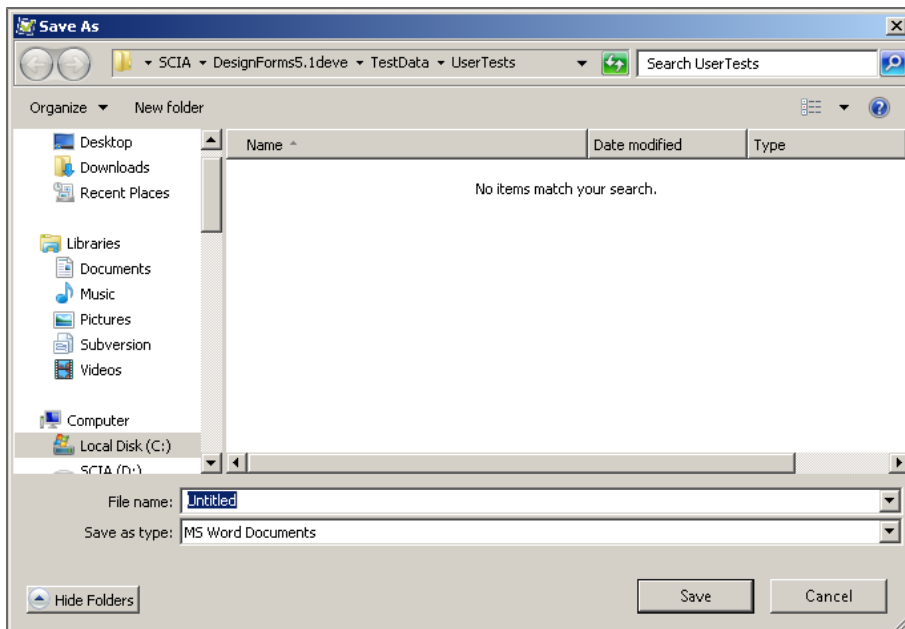
All default value sets are stored in `c:\Users\<user_name>\Documents\DesignForms_*\Forms\` as `*.DEFAULT` files. The name corresponds with the name of the CLC file.

How to export project CLP in User application

- Open the User application.
- Open some forms.
- Go to Project / Export project or use icon.



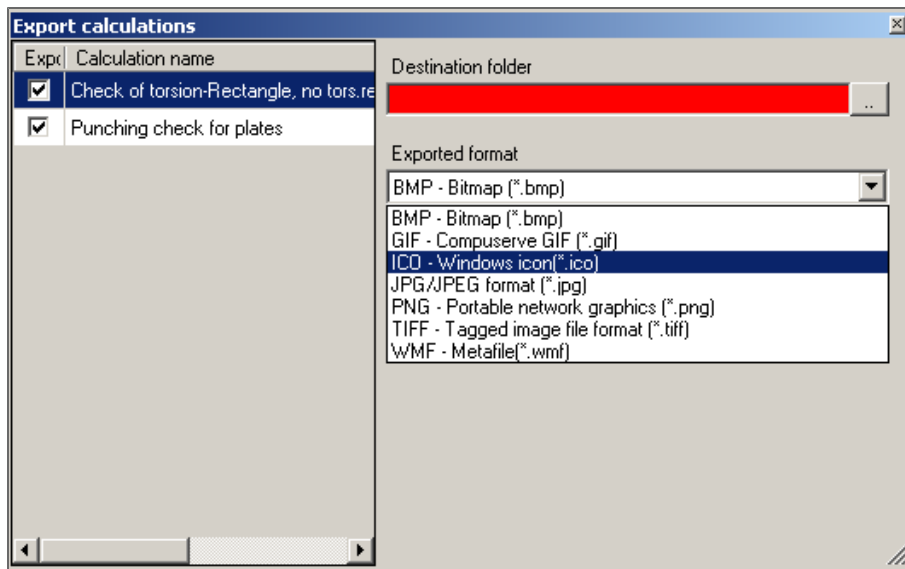
4. "Export to Word" exports the project into MS Word in format *.DOCX. All forms from project are exported. File is automatically opened after export.
5. "Export to RTF" exports the project into Rich Text format *.RTF. All forms from project are exported. File is automatically opened after export.



6. "Export images" exports each form into one image.

Possibly formats are:

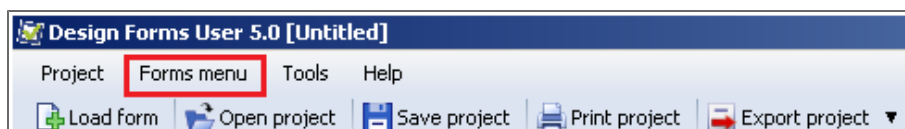
- BMP (Bitmap)
- GIF (Compuserve GIF)
- ICO (Windows icon)
- JPG/JPEG format
- PNG (Portable network graphics)
- TIFF (Tagged image file format)
- WMF (Windows metafile).



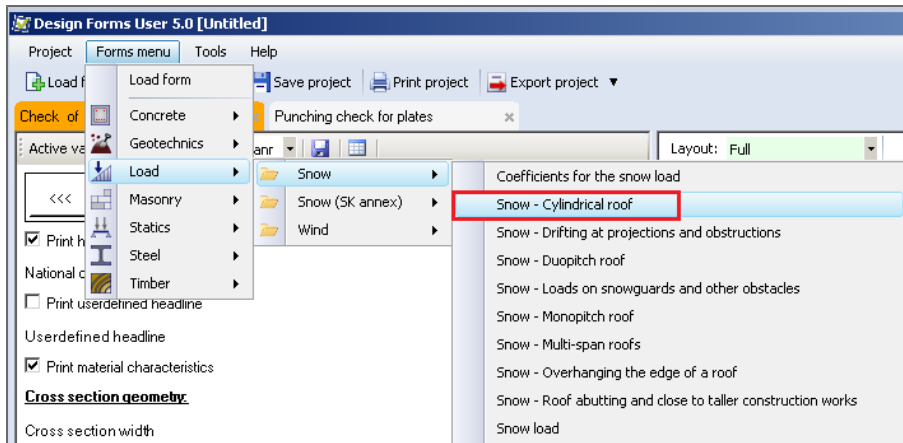
8. Only selected forms are exported, select some forms by checkboxes in the left part of the dialogue.
9. Select the destination folder for exported images.

How to use Forms menu in User application

1. Open The User application.
2. Go to main toolbar / Forms menu. This menu contains all forms which are stored in User directory (c:\User-s\Public\Documents\DesignForms_*(Forms)). There is also a possibility to load any form saved on the hard drive by "Load form" command.



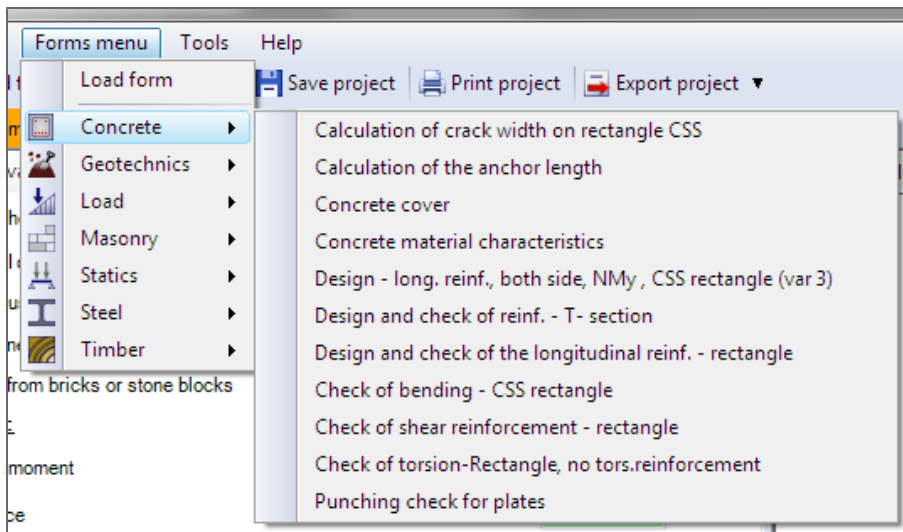
3. Click on any form from the list and it is automatically loaded to the application. Other selected form is opened next the current one.



All forms opened in User application in one moment defines project *.CLP , and it may be saved together in this format.

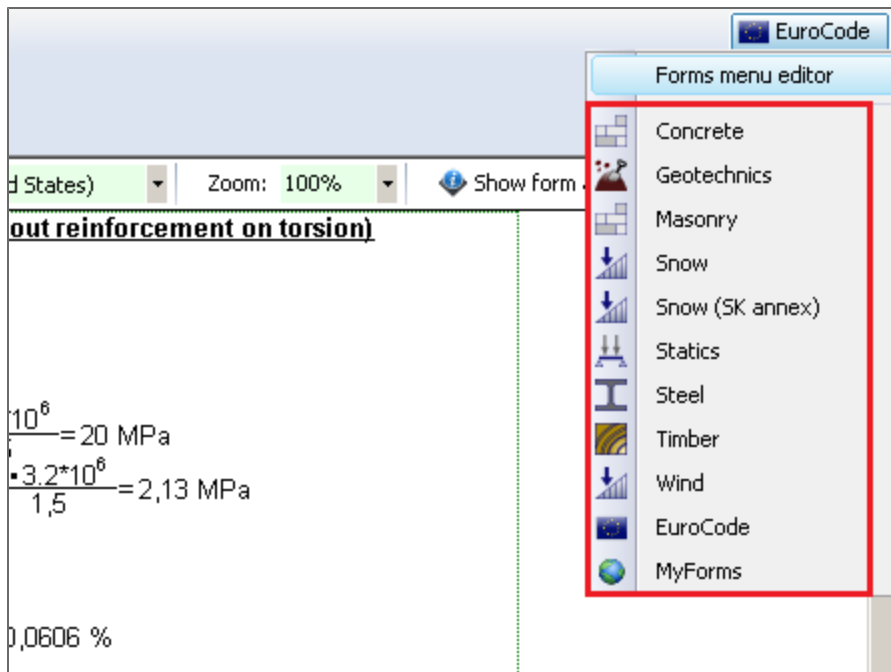
How to use Forms menu editor in User application

1. Open the User application. Check the Forms menu on the main toolbar.
The predefined tree with forms is displayed.

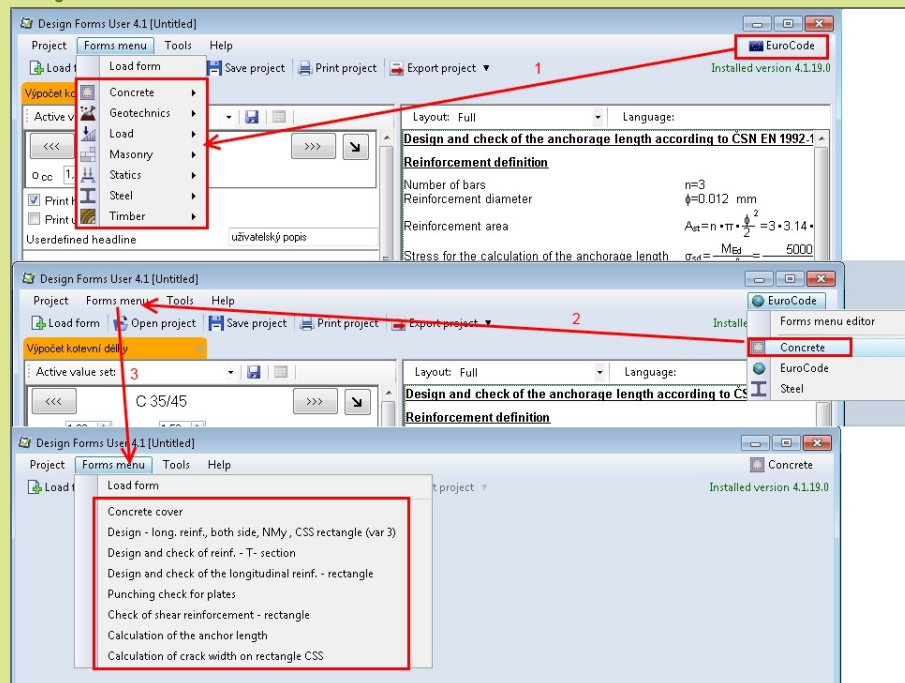


Forms menu editor is a tool which defines the Forms menu (the tree) on the main toolbar.

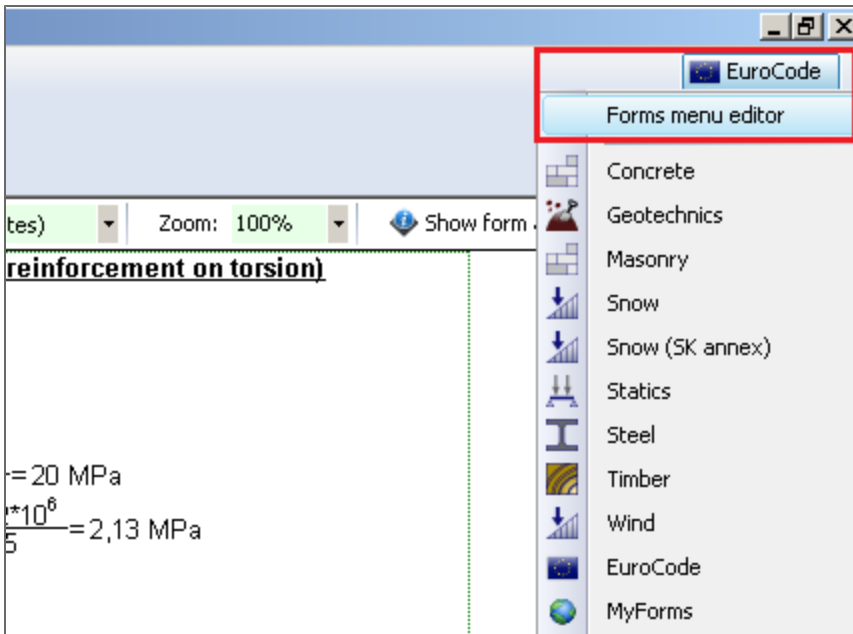
2. Click on the button in the top right corner and select one of the items. The Forms menu on the main toolbar is changed.



Change from Eurocode to Concrete:

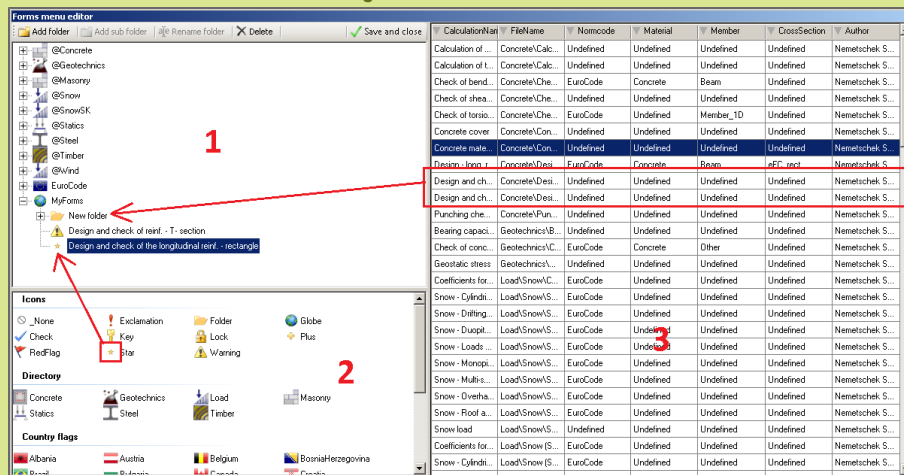


3. Click on the button in the top right corner and select the first option - Forms menu editor or go to Tools / Form menu editor.

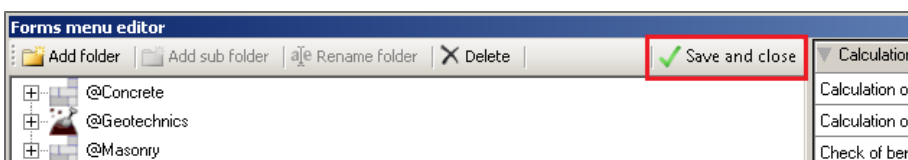


Forms menu editor is divided into three parts.

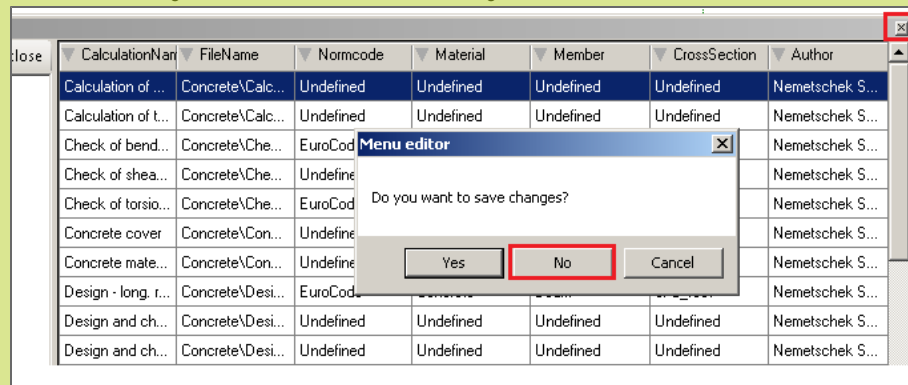
- 1) The tree of forms which is shown in Forms menu. User can add folder, sub-folder, rename folder or delete folder or form.
- 2) There is a list of icons which can be used for folders or forms.
- 3) Forms which may be added to the tree and paths to them. Use "drag and drop" the form to the folder in the tree. It is located in c:\Users\Public\Documents\DesignForms_*(Forms\).



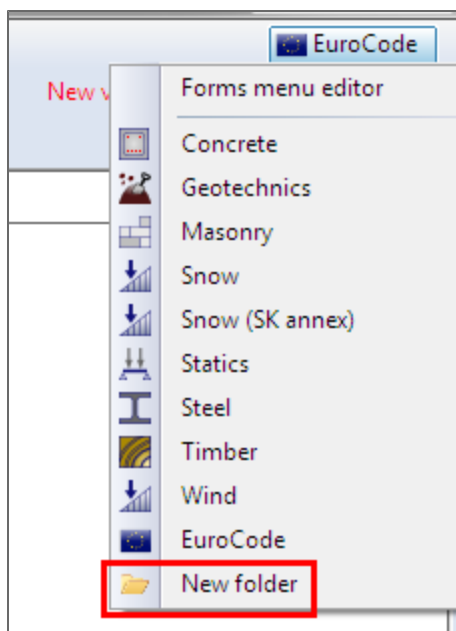
4. Create a new folder and add some forms from the right part to it.
5. Select one form in the tree and click on icon. The icon is now displayed in the tree.
6. Select the folder, hold CTRL + click on the icon. The icon is used for all sub-folders or forms in the selected one.
7. Save the changes by button "Save and close". Changes are saved in XML file in c:\Users\\Documents\DesignForms_*(Forms\).



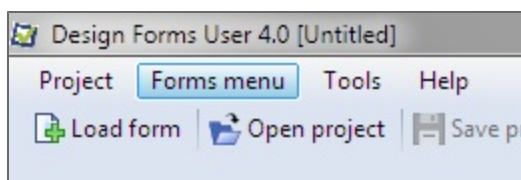
Close without saving - use the cross. Select No in the dialogue.



8. Select the newly created folder in the top right button list.



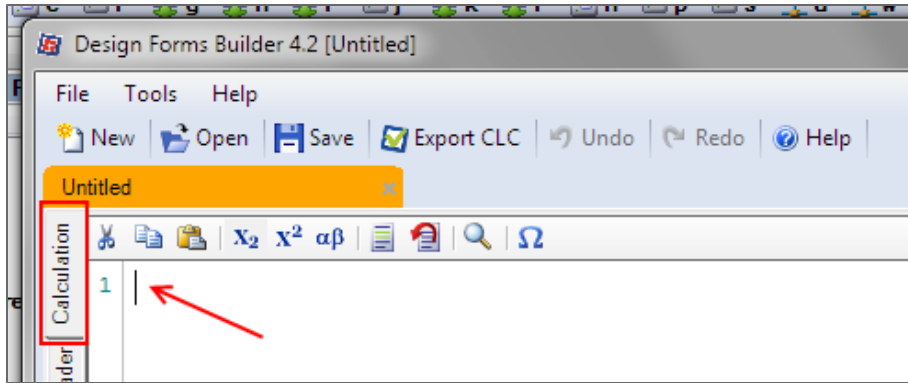
9. Check the Forms menu on the main toolbar.



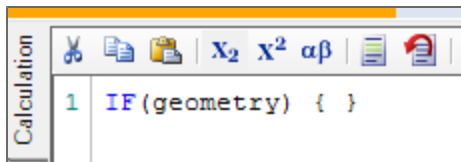
How to create a condition in Builder application

1. Open the Builder application.
2. Create a new form - button New.

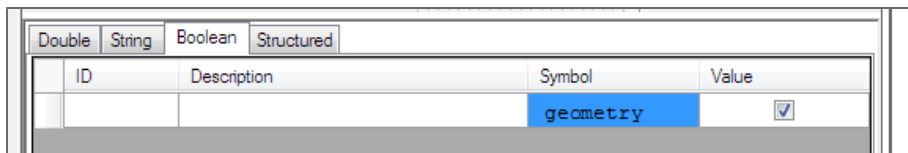
- Click to the editor on the Calculation tab.



- Write command IF and the condition text to brackets and two curly brackets.



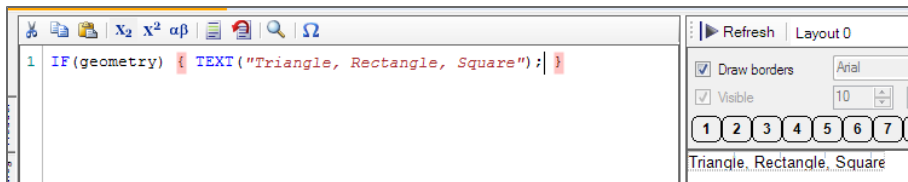
- Press Refresh button. The right part of the application is still empty, but the boolean variable with symbol "geometry" is added to the table of variables (bottom part of the code editor).



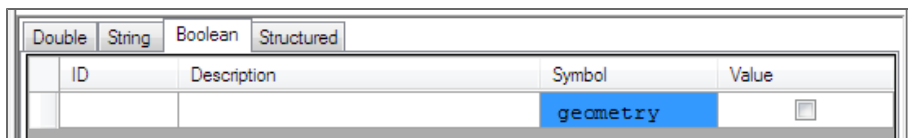
- Write commands which are linked to this boolean variable.



- Press Refresh button. Now the result is visible in the right part.



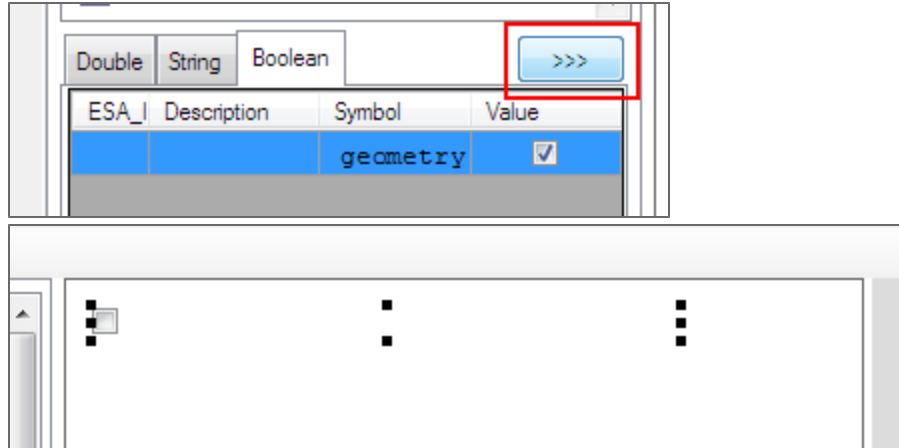
- Go to table of variables and uncheck the geometry variable. Confirm by Enter.



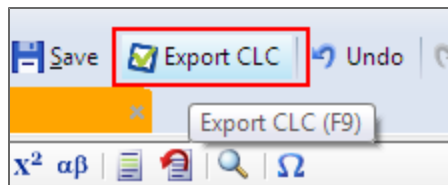
- Press Refresh button. Now the right part is empty again.

- Go to the tab Dialogue.

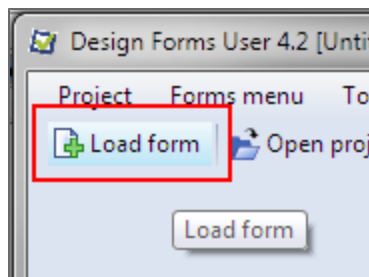
11. Add the variable geometry from the table of variables to the Dialogue.



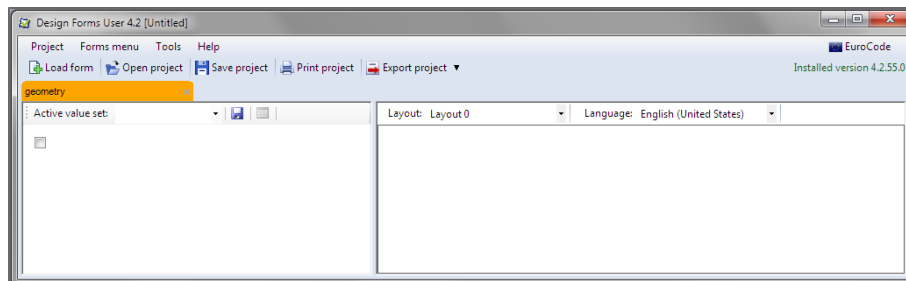
12. Save the file, and export it to CLC.



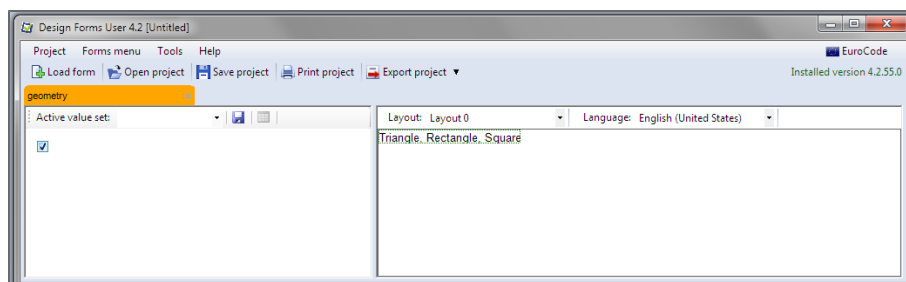
13. Run the User application.
14. Open the CLC file in the User application - button Load form.



15. The Dialogue contains the unchecked checkbox.



16. Check the checkbox. The text is visible now.

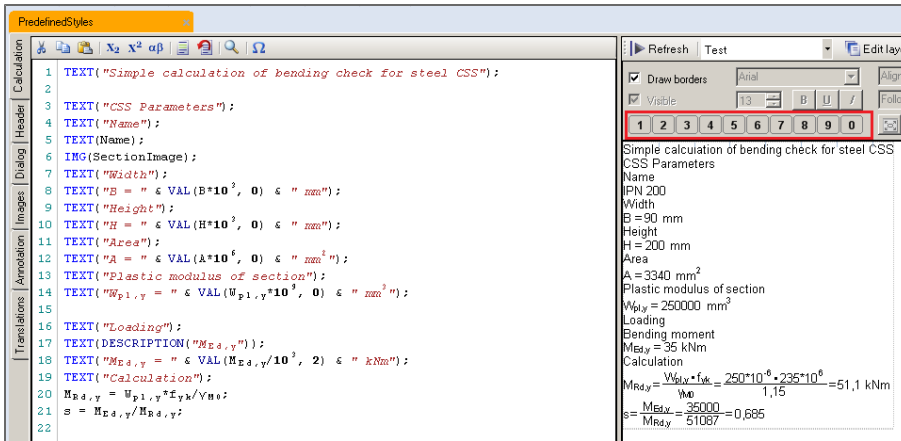


- Download the final files [geometry.CLS](#) and [geometry.CLC](#).

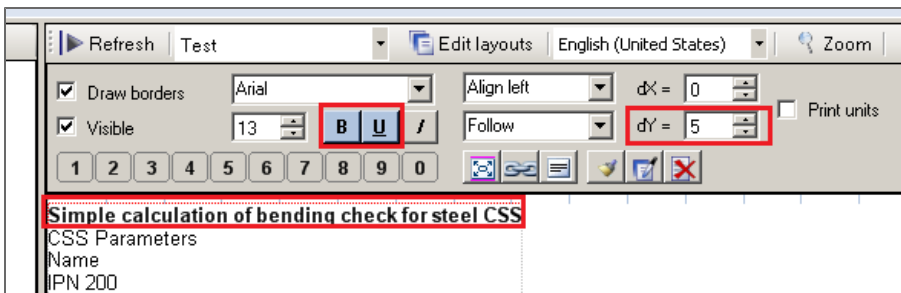
How to create and use Predefined styles in Builder application

This tutorial describes how to create a Predefined style and how to use it in the layout.

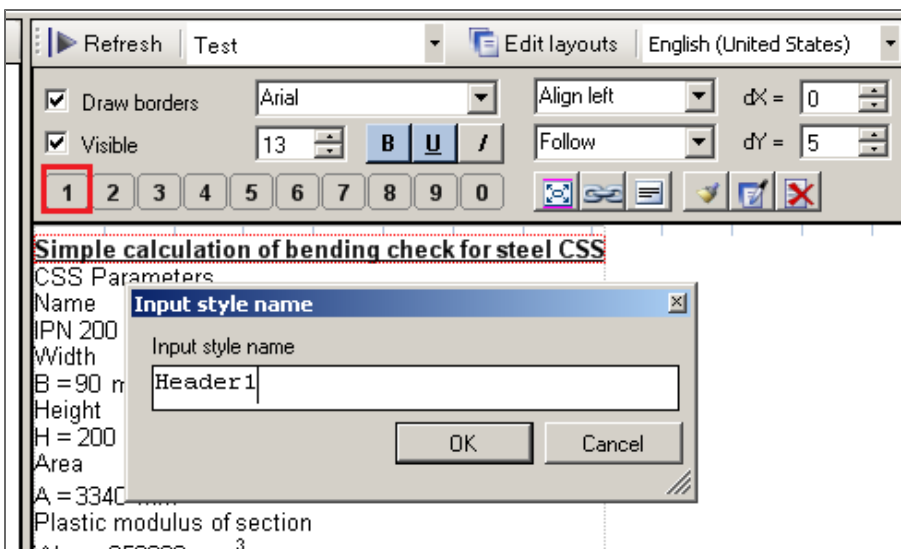
- Download [PredefinedStyles.cls](#) and open it in Builder application. It is a simple example without the layout definition. The predefined styles are displayed by numbers on the layout toolbar.



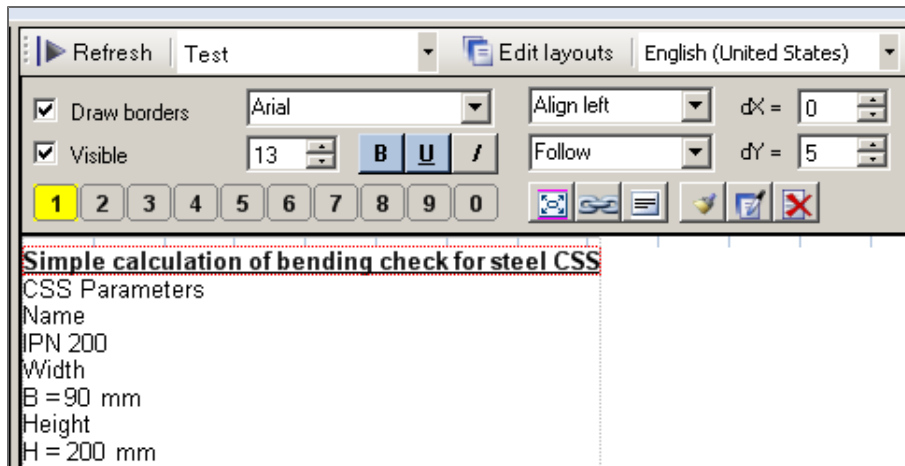
- Select the first row in the Layout define a style - Bold, Underline font and Follow dY=5 offset.



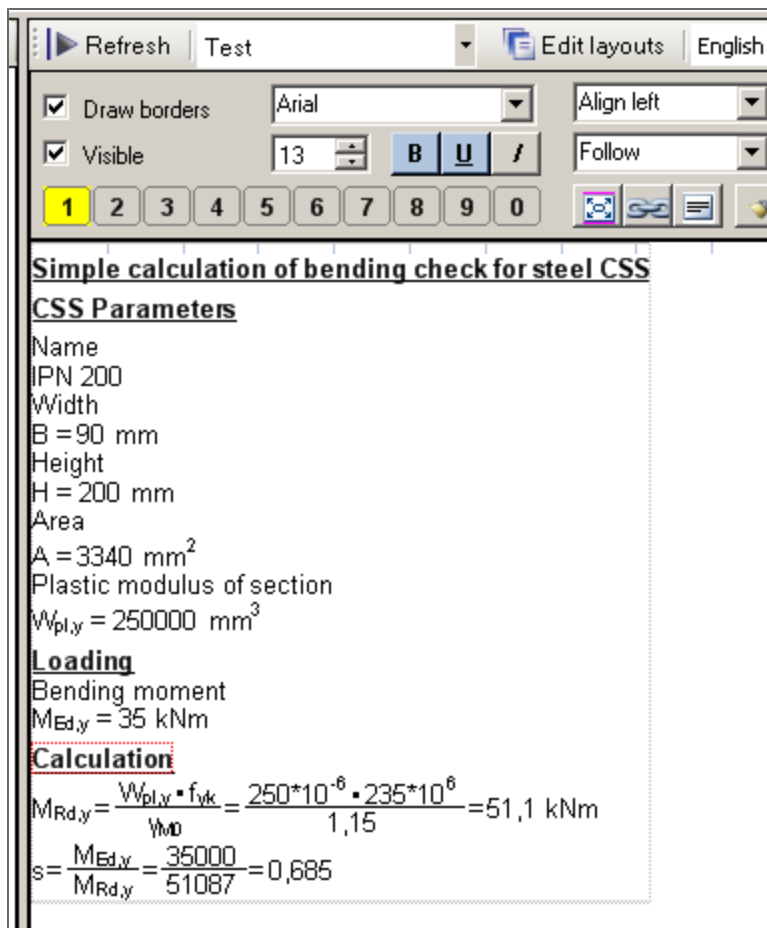
- Hold the CTRL key and click on the first number - 1. Define the style name Header1 in a new dialogue.



4. The style Header1 is now stored under the button 1. The button has a yellow colour when the selected visual component has its style.



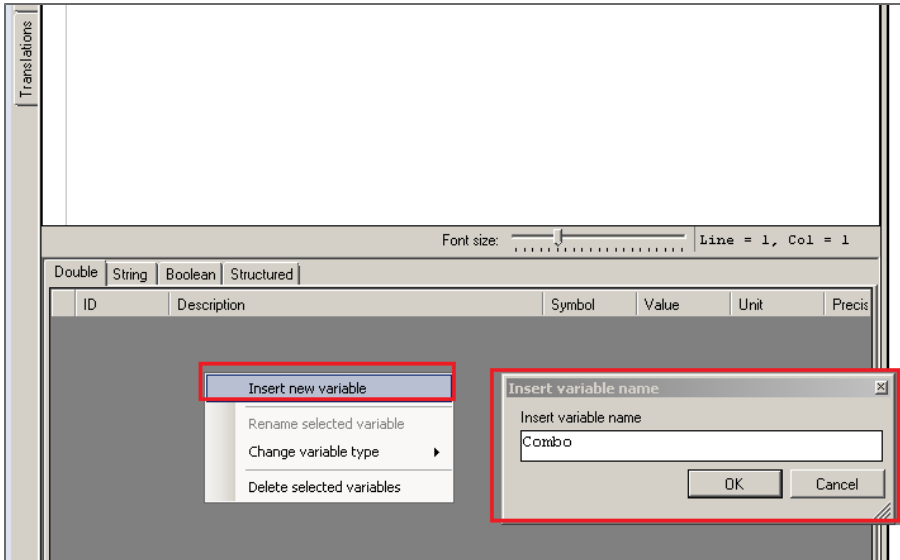
5. Select visual components "CSS Parameters", "Loading" and "Calculation". Click on button 1. These components are now formatted according to the style Header1.



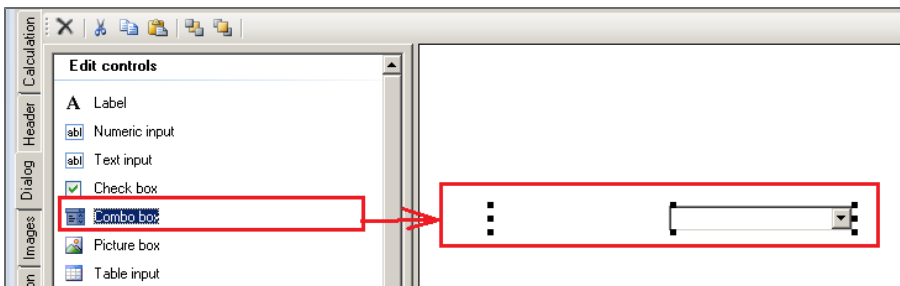
It is possible to define up to 10 different predefined styles.

How to create and use combo-box in Builder application

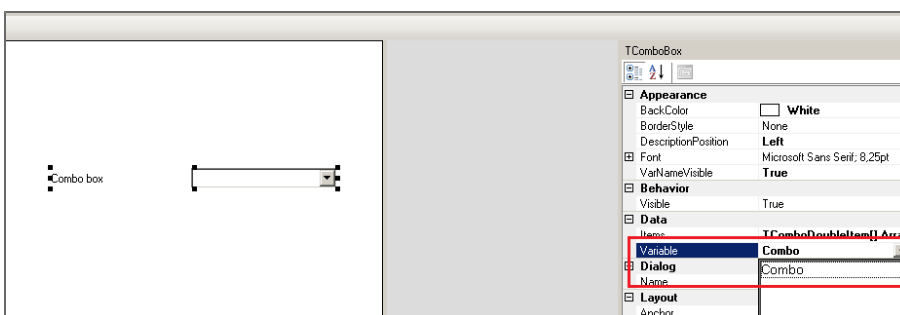
1. Open Builder application and start a new form.
2. Use the right click in the table of variables and display a context menu. Add a new numeric variable and name it "Combo".
When the new variable is added, define some description e.g. "Combo box" and click on the "Refresh" button (or use short-cut F5).



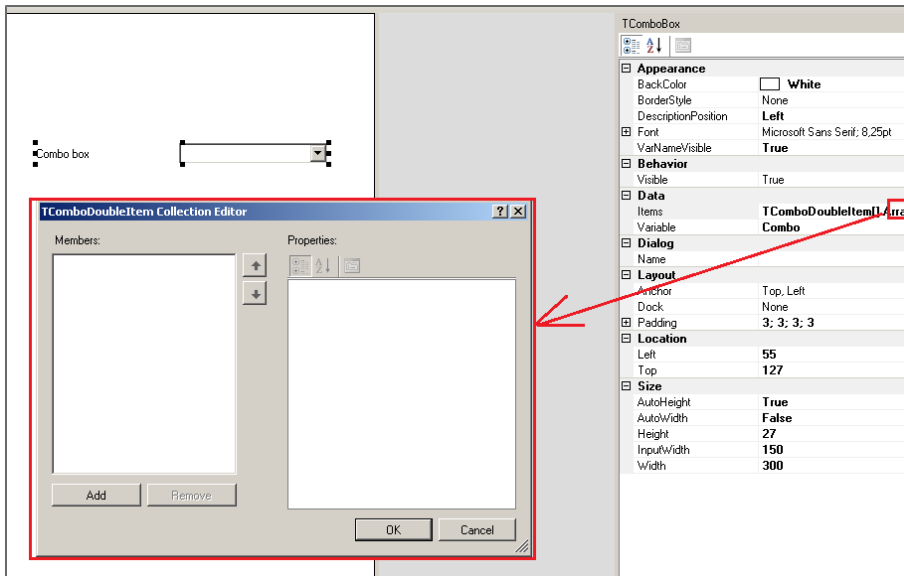
3. Go to the Dialogue tab and drag&drop [Combo box edit control](#) on the blank panel.



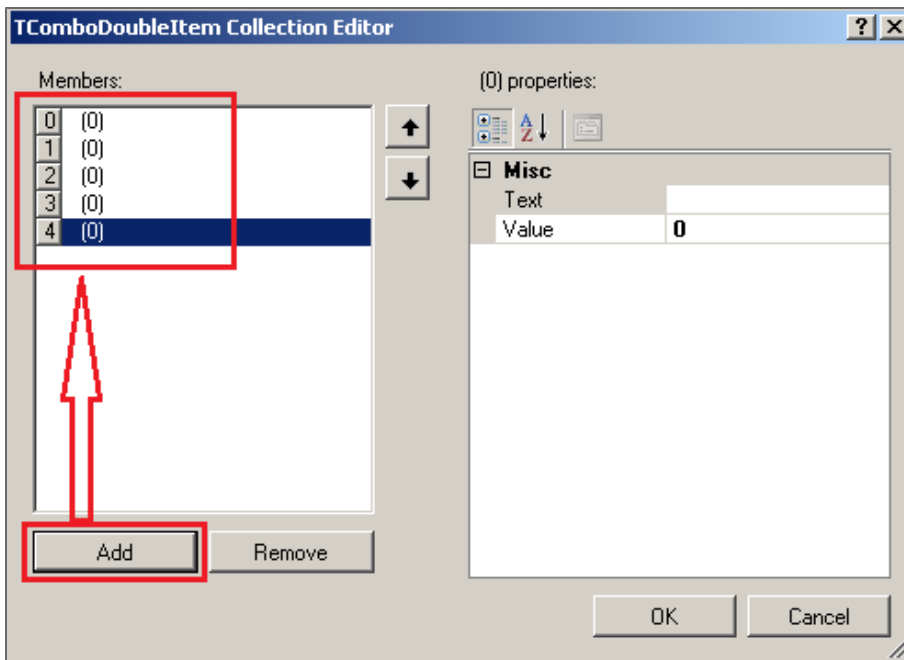
4. Set the Variable in the properties to the new variable "Combo".



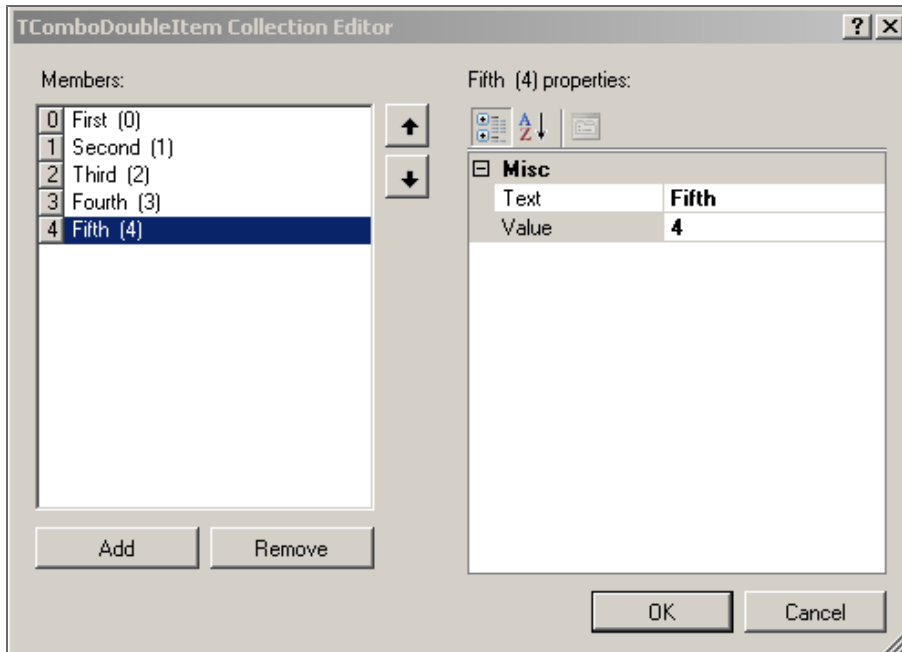
5. Go to property Items and click on button [...]. The [combo box definition](#) appears in a special dialogue.



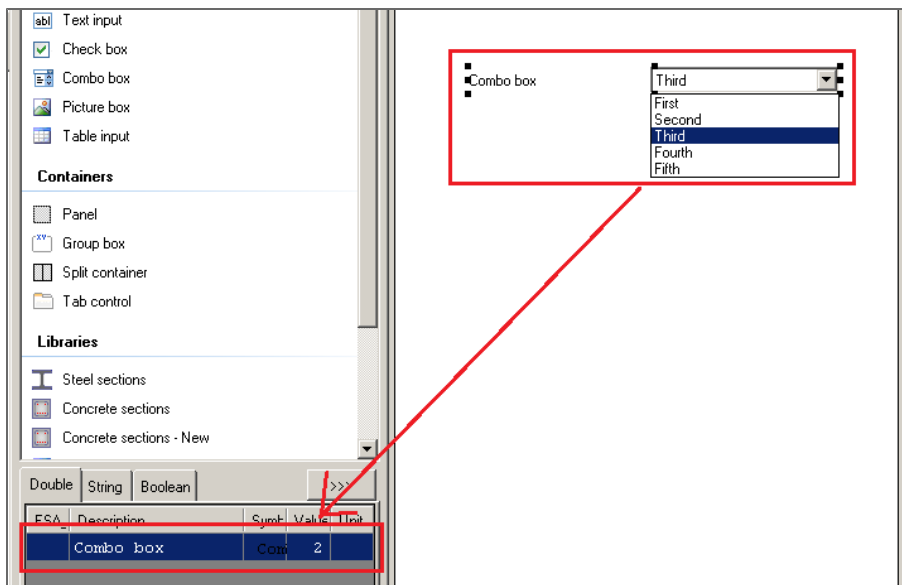
6. Add 5 items (rows 0 - 4):



7. Add some description and value to each item in the list. The value have to be the integer number - it means 0, 1, 2 ..., it is used as an index of the item. There is 5 rows in the example so the indexes are from 0 to 4. Confirm it by OK.



8. The combo-box is added and it is assigned with variable "Combo". It returns value number according to the selected item.



9. Combo-box can be than used in [SWITCH - CASE command](#):

Copy this example into the form:

```
SWITCH(Combo) {
CASE 0: {
TEXT("Combobox first position");
}
CASE 1: {
TEXT("Combobox second position");
```

```
}  
CASE 2: {  
  TEXT("Combobox third position");  
}  
CASE 3: {  
  TEXT("Combobox fourth position");  
}  
CASE 4: {  
  TEXT(".....");  
}  
}
```

10. Refresh the calculation and go back to the Dialogue tab. Change the selected item in the combobox. Check the changed value "Combo" in the table of variables. This way the combobox selection affect the calculation.

Combobox is a greate tool for switching the way of the calculation.

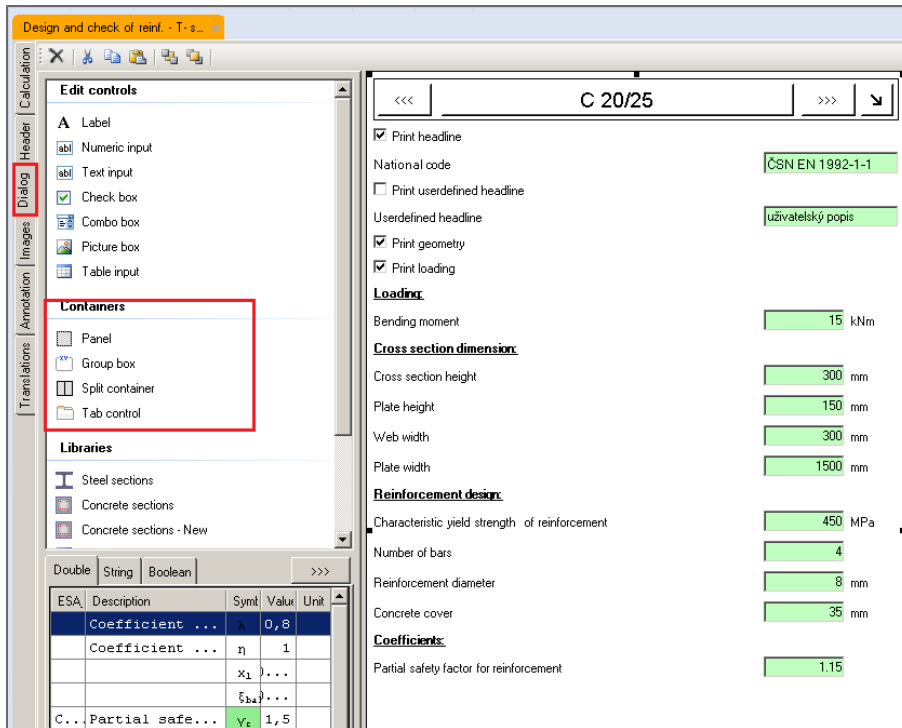
For example:

1. Calculate Rigid, Sliding or Custom support
2. Snow load on Monopitch, Duopitch, Flat roof
3. Display ULS, SLS or ULS+SLS results

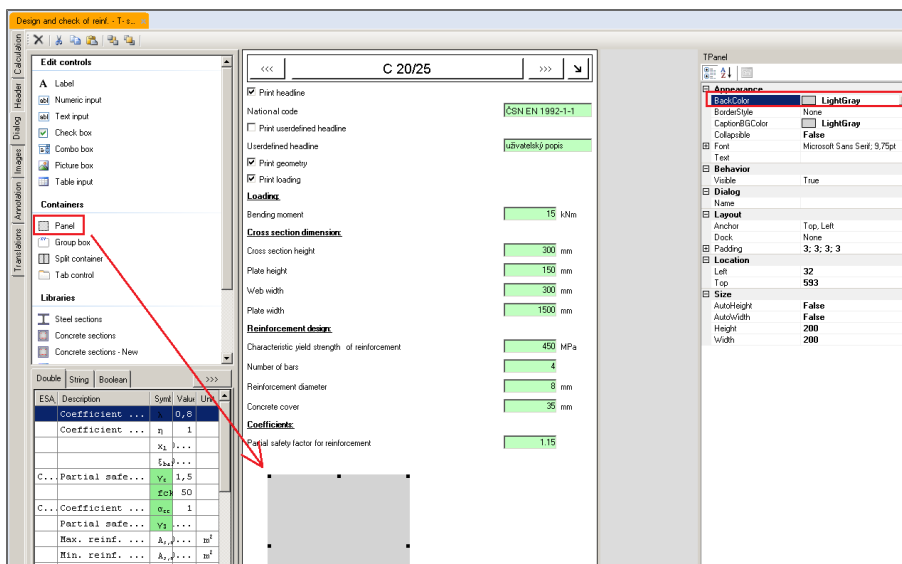
How to use dialogue containers in Builder application

Dialogue container is a special type of control which may help to organize the Dialogue layout. User may use just panel with different colour, panel with frame and name, panel with tow tabs and so.

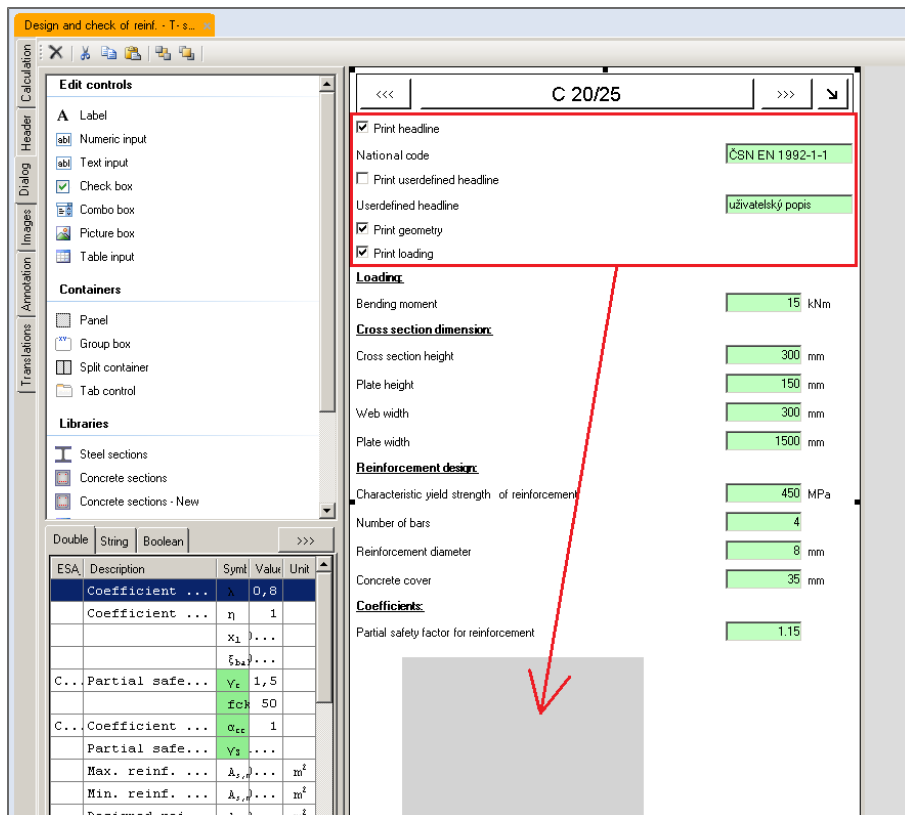
1. Open the Builder application.
2. Open form [Design and check of reinf. - T-section.CLS](#)
3. Click on the Dialogue tab. There is a dialogue which was created without dialogue containers.



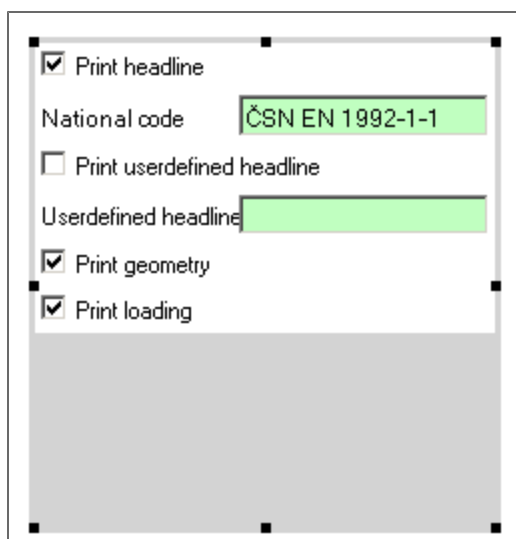
4. Add a Panel container by Drag&Drop it below the last variable and than change its BackColor to LightGray.



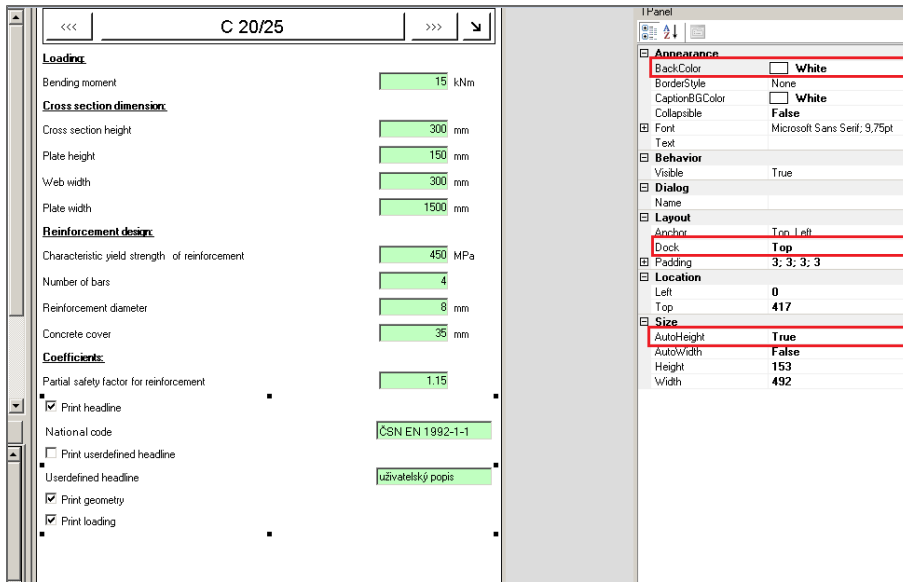
5. Drag&Drop first six variables (from Print Headline to Print Load) into the panel (the variables are docked, so they wont be displayed next to the cursor). Change the size of panel using black square in its corner, if it is necessary.



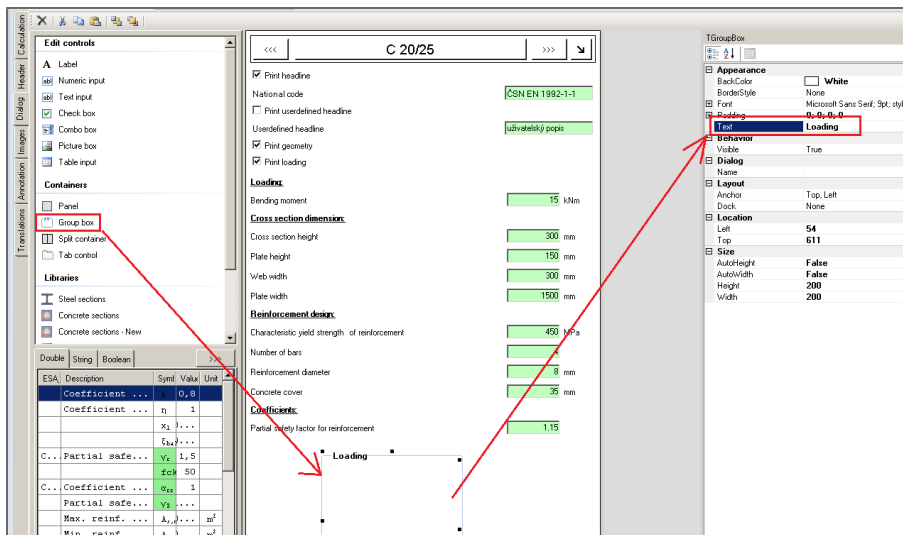
6. Change the order of variables on the panel. Select the variable and use the keyboard:
 - Page Up on the keyboard for moving variable up
 - Page Down for move it down
 - Home for move it at the first position in the panel
 - End for move it at the last position.
7. Now reorder variables into original order.



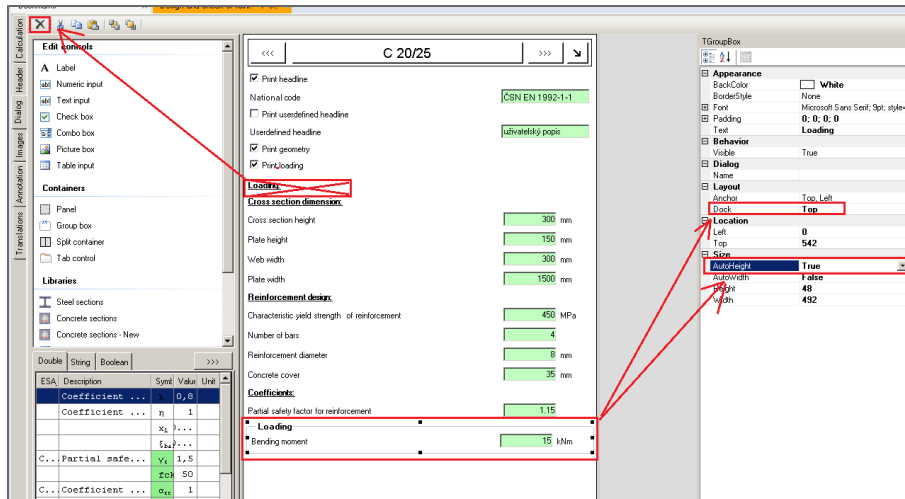
8. Select panel and change its properties - BackColor on White, AutoHeight on True and Dock on Top. Panel is than coloured, docked and resized similar to the other dialogue components.



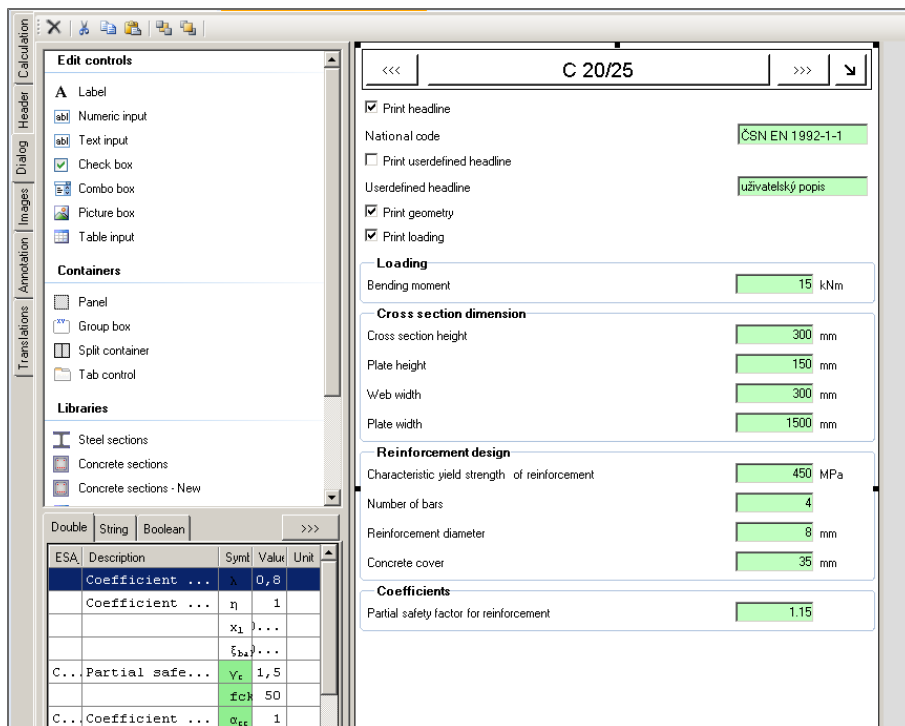
9. Select panel and move it by Page Up at the top of the variables, just below the Concrete library.
10. Drag&Drop the Group box container into the dialogue. Select it and rename it to "Loading".



11. Drag&Drop "Bending moment" variable to this Group box. Dock the Group box to the Top and set AutoHeight on True. Select text row Loading: and delete it.

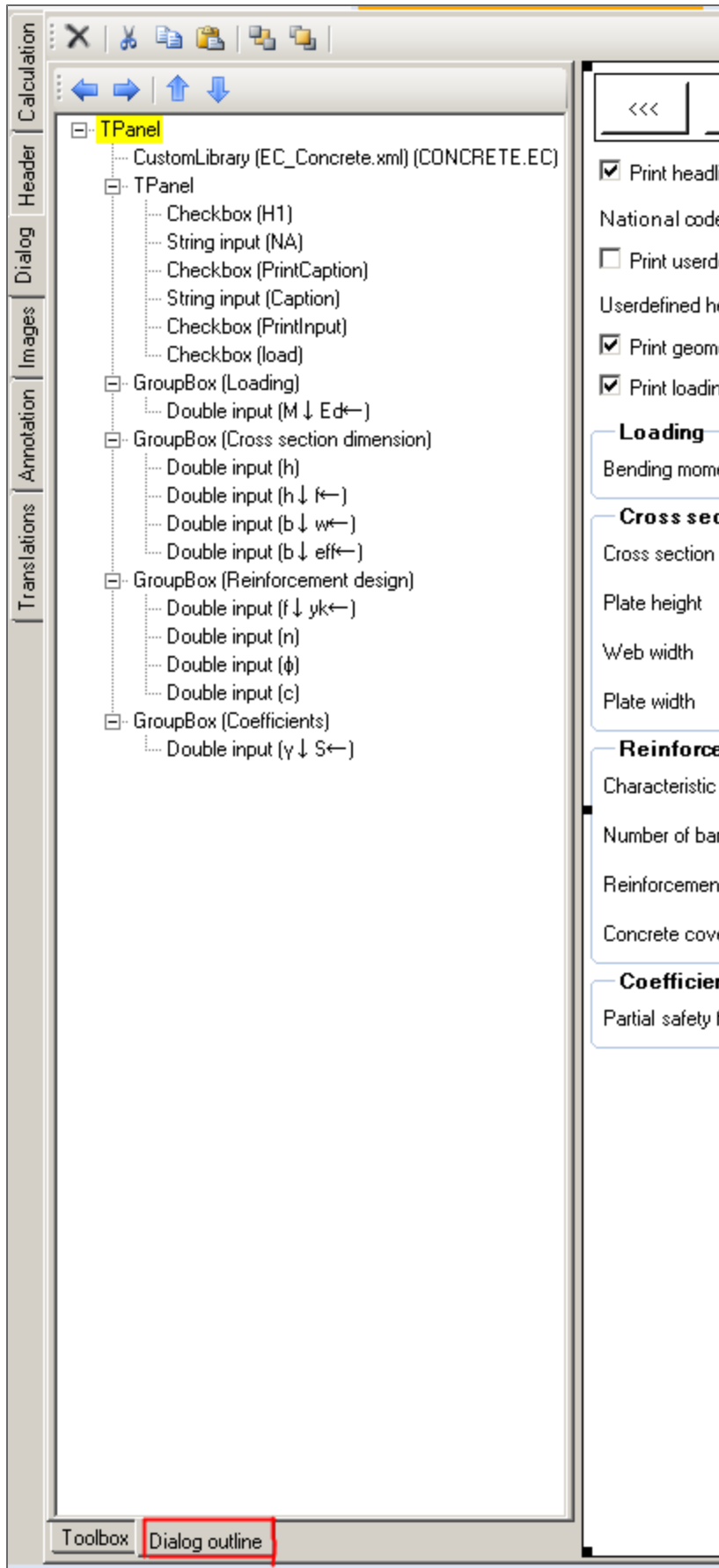


12. Repeat steps 9 and 10 for the Cross section dimension, Reinforcement design and coefficients.



13. The dialogue is now correctly adapted on version 5.0 with containers.

14. You can check the dependences in the Dialogue outline.

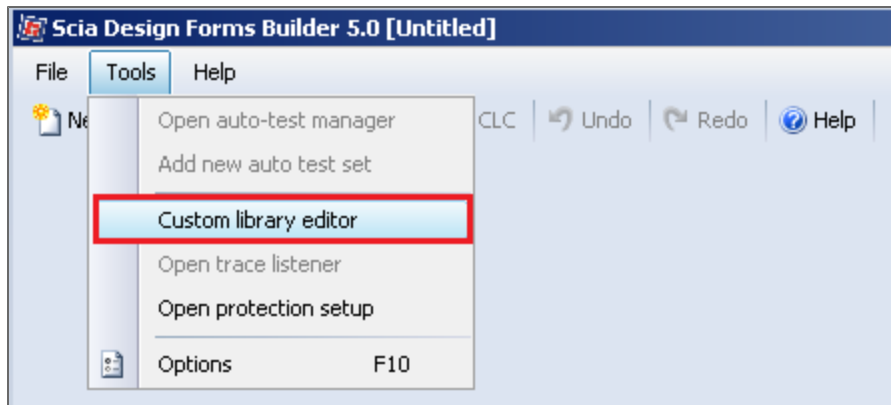


15. Other containers (Split container and Tab control) may be used in the same way.

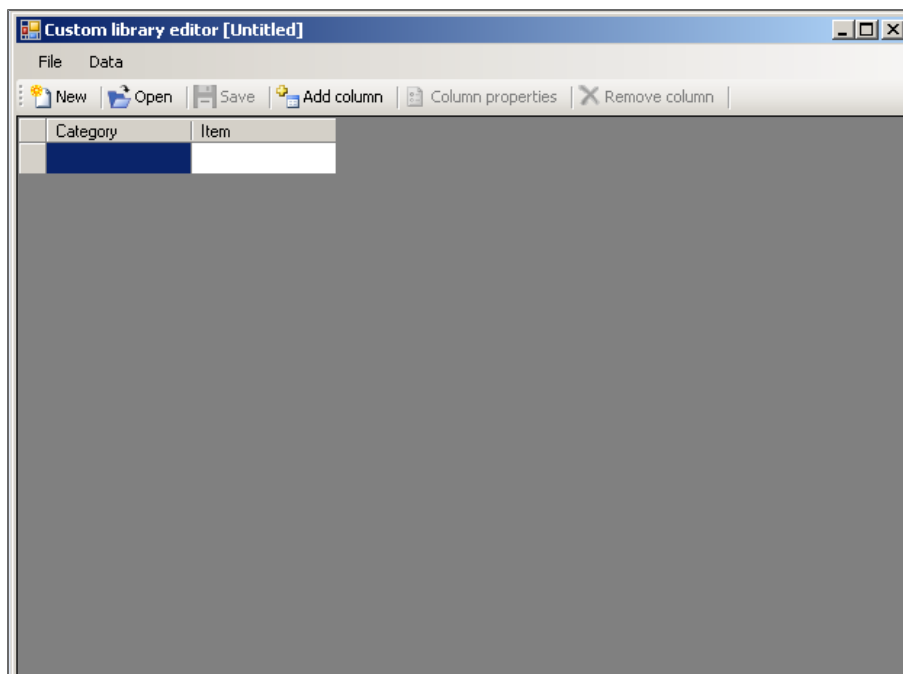
How to create and use Custom Library in Builder application

This tutorial describes how to create a Custom Library file (XML format) and how it can be used in the form.

1. Open Builder application.
2. Go to Main toolbar -> Tools -> Custom library editor.



3. A new dialogue appears, it displays a new empty custom library. It is possible to create new library or open and edit an existing one.



4. Download an example custom library [Cars.xml](#) and save it to c:\Users\Public\Documents\DesignForms_*(CustomLibrary\) (symbol * represents the version of Scia Design Forms). Open it in the custom library editor. There is a table with cars types and their properties.

Custom library editor [C:\Users\Public\Documents\DesignForms_5.0\CustomLibrary\cars.xml]

File Data

New Open Save Add column Column properties Remove column

Category	Item	Engine power	Engine displacement	Color of body
Sport	A	500	3000	Red
Sport	B	300	2000	Black
SUV	SUV1	200	2500	Black

The columns Category and Item used for sorting the library. They are required for all items.

Category - Items can be placed in many different categories. If one category is used in the whole library it is not visible in the dialogue (it would be the same for all library items) - see the step 9.

Items - The column "Items" contains an unique name for each item in the library. The name must be unique within one category. It is possible to use the same name in more different categories.

Other columns - Columns are created by "Add column" button. The column properties may be edited by "Column properties" button. The column may be deleted by "Remove column" button. Last two buttons are active, when some column is selected. Other columns display the item properties (e.g. colour, size, strength, yield strength, coordinates, ...).

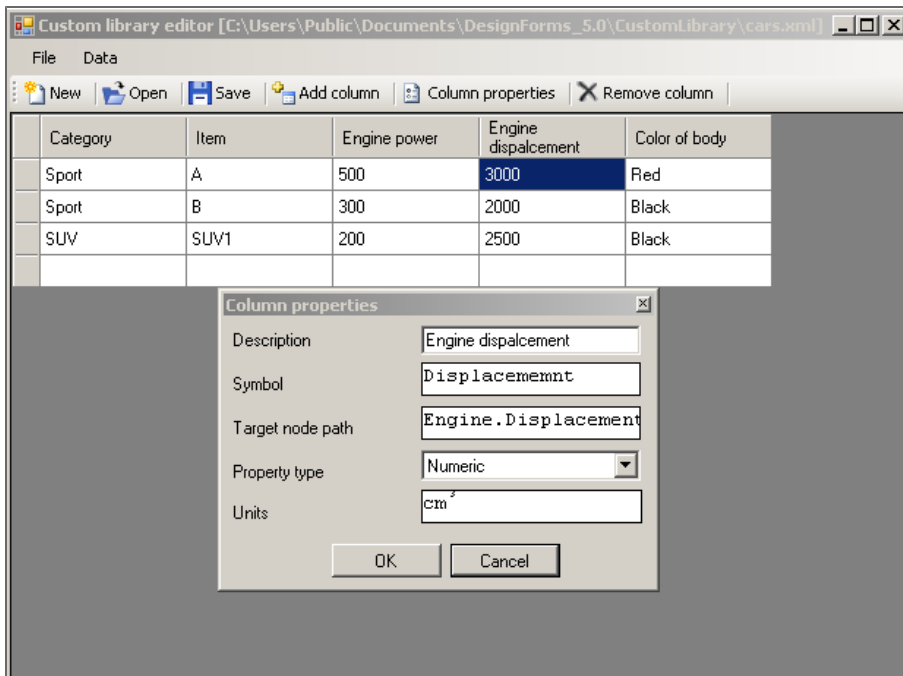
Custom library editor [C:\Users\Public\Documents\DesignForms_5.0\CustomLibrary\cars.xml]

File Data

New Open Save Add column Column properties Remove column

Category	Item	Engine power	Engine displacement	Color of body
Sport	A	500	3000	Red
Sport	B	300	2000	Black
SUV	SUV1	200	2500	Black

5. Check the Cars library - select the column Engine displacement and display the column properties. A new dialogue is appears.



Description - It displays the description of the variable, this is visible afterwards in the header of the library and in the Dialogue.

Symbol - It displays a symbol of variable which is visible in the Dialogue.

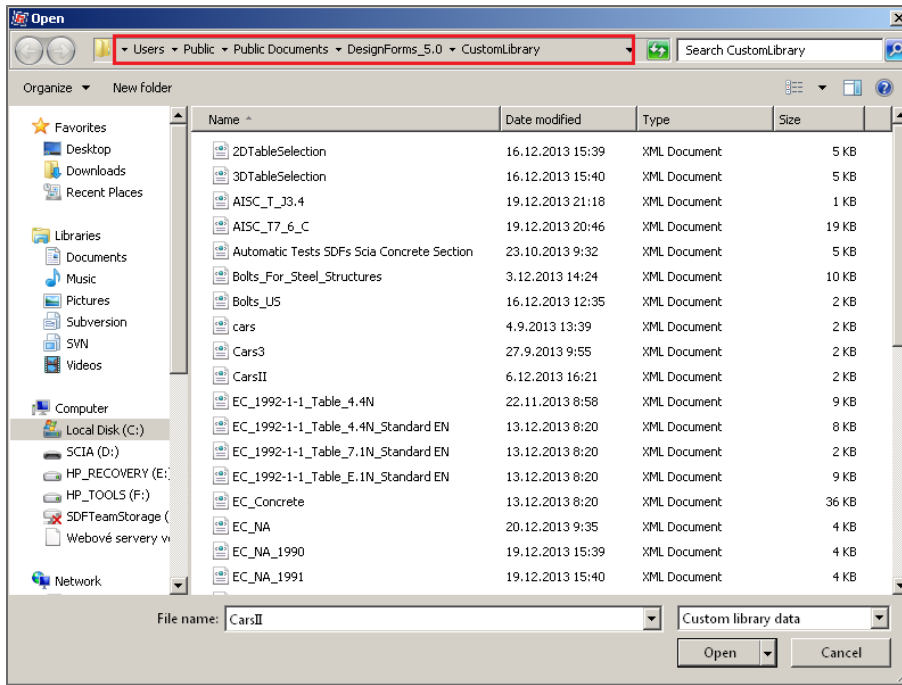
Target node path - This is very important property. It is used for mapping the value of the variable into the form. It supports the dot convention so the tree dependency can be visible (e.g. Car.Visual.Colour.Hood or Car.Interior.Material.Dashboard).

Property type - Type of the variable.

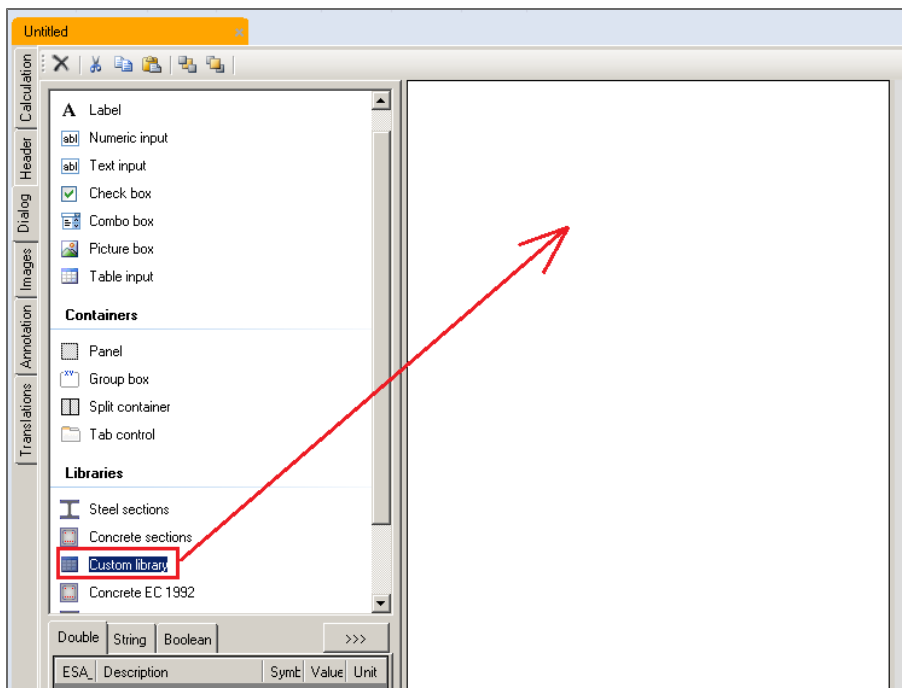
- **Numeric** - Numeric variable type with the double precision. Units can be defined.
- **String** - String variable type.
- **Bool** - Boolean variable type, the possible value is TRUE or FALSE.
- **Struct** - Structural variable type, it may contain Numeric, String and Bool variables.

All item properties are defined by columns. Add one column for each property and then define all items in the library by those properties. The content may be copied and pasted from another table (e.g. MS Excel, Open office).

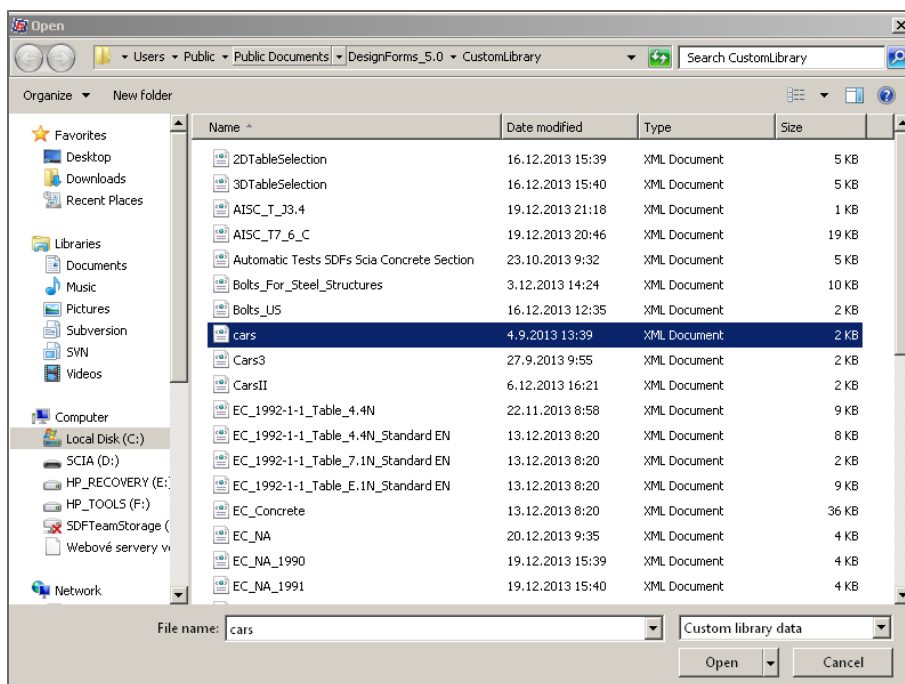
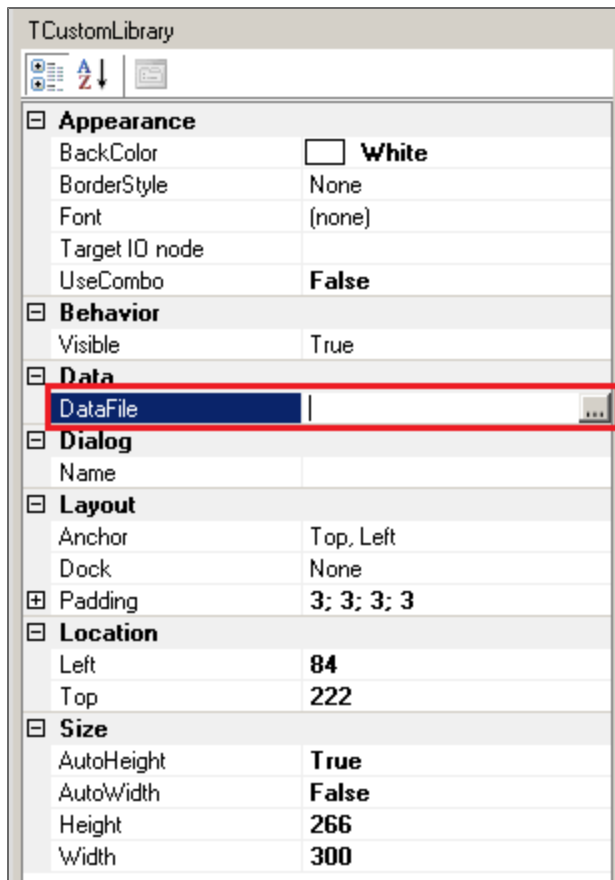
6. The Custom library have to be stored in the Public documents in the CustomLibrary folder. It must be saved under the appropriate version of Scia Design Forms.



7. Close Custom Library editor and open a new Form.
8. Go to the Dialogue and add a new Custom library from the Libraries part.



9. Go to property DataFile, click on [...] button and select cars.xml.



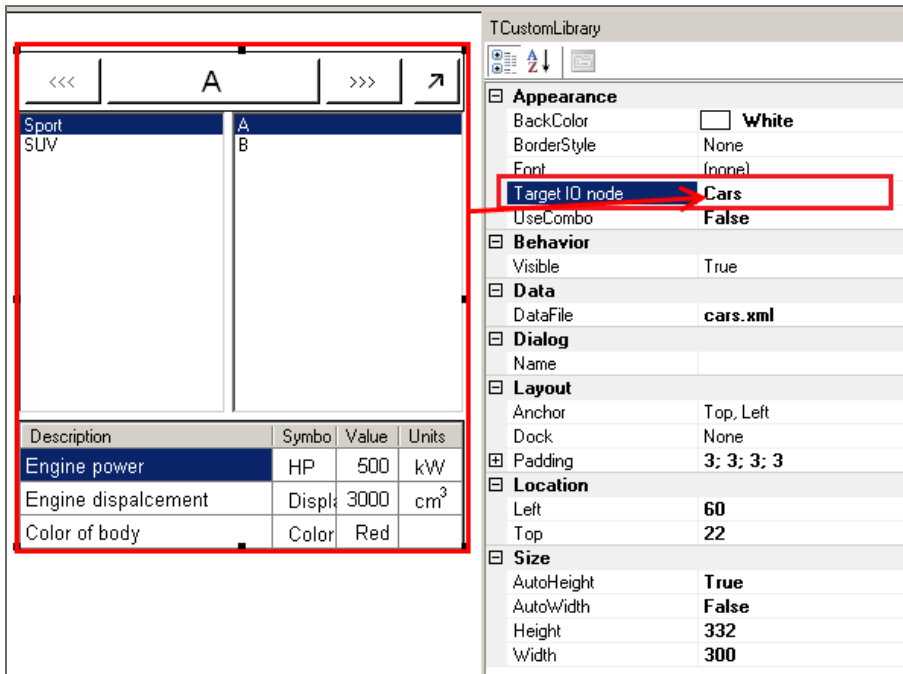
10. The Cars custom library is added to the dialogue. It displays all defined categories, items and variables. The pictures show examples with one and two categories in the custom library.

Description	Symbo	Value	Units
Engine power	Hp	500	kW
Engine displacement	Disp	2500	cm ³
Color	Color	Gr...	

Description	Symbo	Value	Units
Engine power	HP	500	kW
Engine displacement	Displ	3000	cm ³
Color of body	Color	Red	

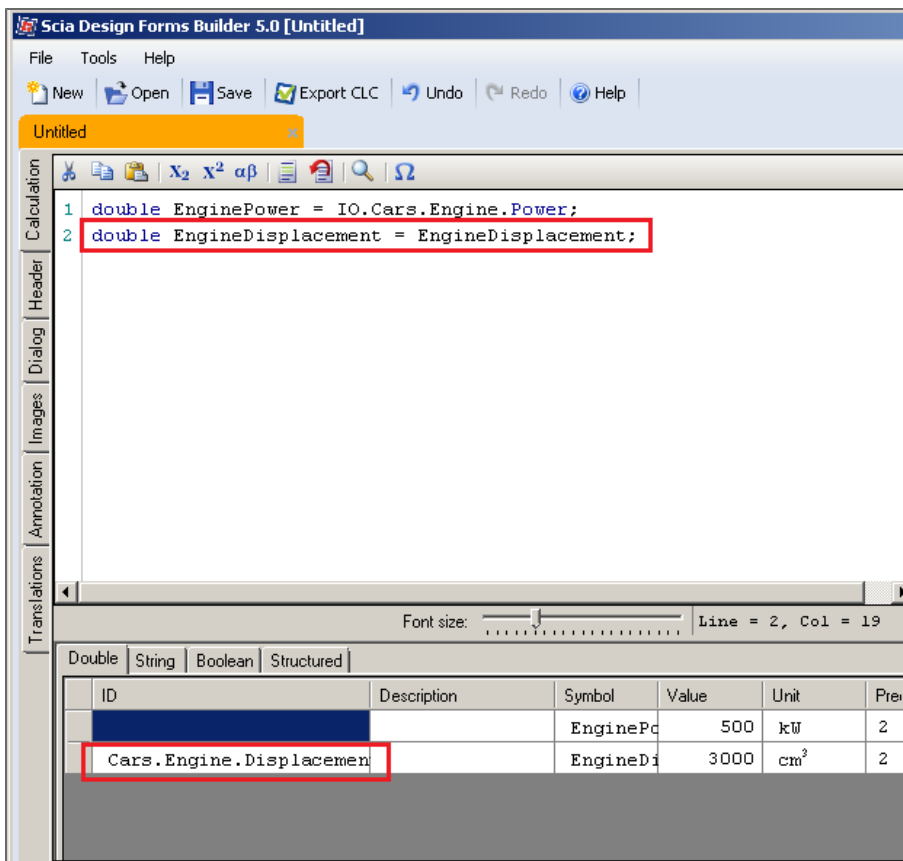
11. Fill the correct Target IO node property to add the custom library correctly to the form. Use its name "Cars".

This name will be used in the source code to find the required property. See more about [Target IO node and dot convention](#) and [using it in the Dialogue](#) in the separate chapters.



12. Use F5 to refresh the calculation. All changes the dialogue are now correctly loaded and active.
13. Go to the Calculation tab.
14. Write this to the source code editor (see the picture):

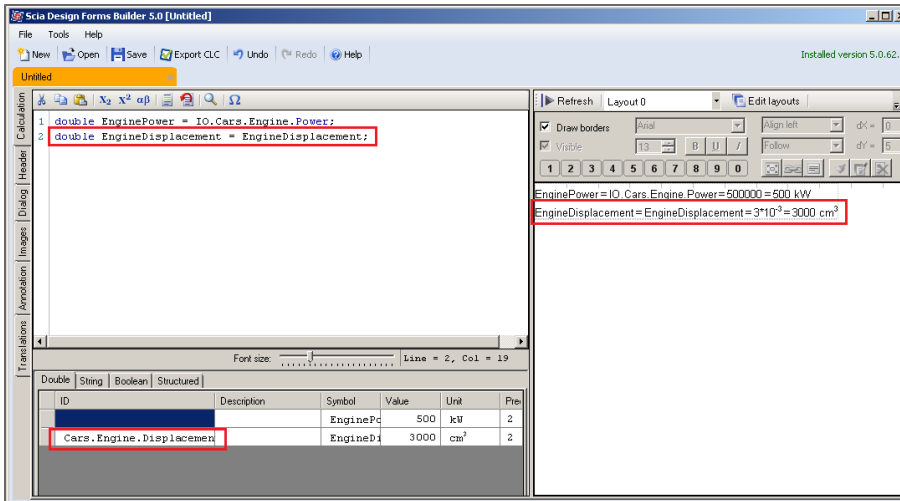
```
EnginePower = IO.Cars.Engine.Power;  
double EngineDisplacement = EngineDisplacement;
```



If the variables are not added automatically to the table of variables, add them manually by the context menu.

There are two ways of using the Custom library:

- 1st way based on the Target IO node property - 1st command
 - 2nd way based on the Target node path property - 2nd command
15. Refresh the calculation (or use F5) to see the result in the layout.



How to create and use Auto test in Builder application

This tutorial describes how to create [TestIO](#) file and how to run autotest for the SDF form.

The autotest consists from source (CLS, CLC) and input/output values (TestIO). The input/output is XML file with values which are used in the calculation (input) to get results (output).

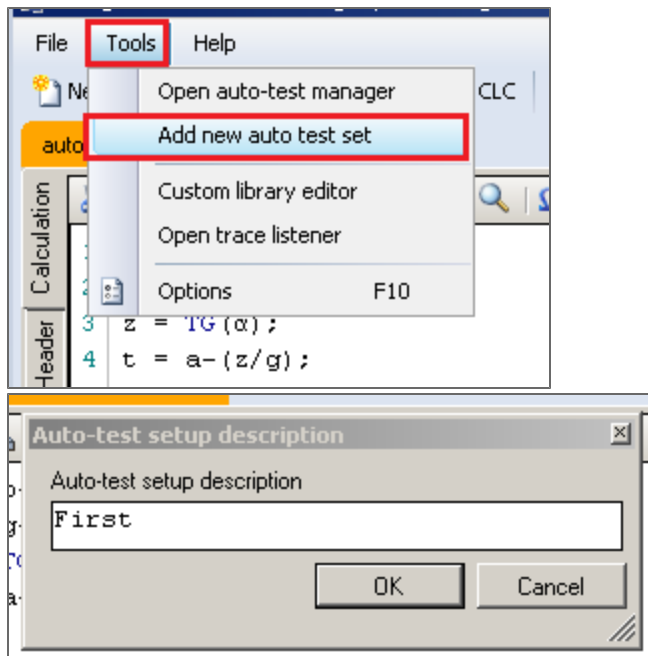
1. Open the Builder application.
2. Open the file [autotest.cls](#)

When the variable is used as result, user has to use a special ID. Write the ID Result.0, Result.1 into ID column in the table of variables. See the chapter about [auto tests](#).

1. Write Result.0, 1, 2... into all four variables which represent results.

ID	Description	Symbol	Value	Unit	Precisio
		b	2.0		2
		c	4		2
		d	5		2
Result.0		a	2.9		2
		g	7.9		2
		f	555		2
Result.1		h	-547		2
Result.2		z	0.933		2
Result.3		t	28.9		2
		α	43		2

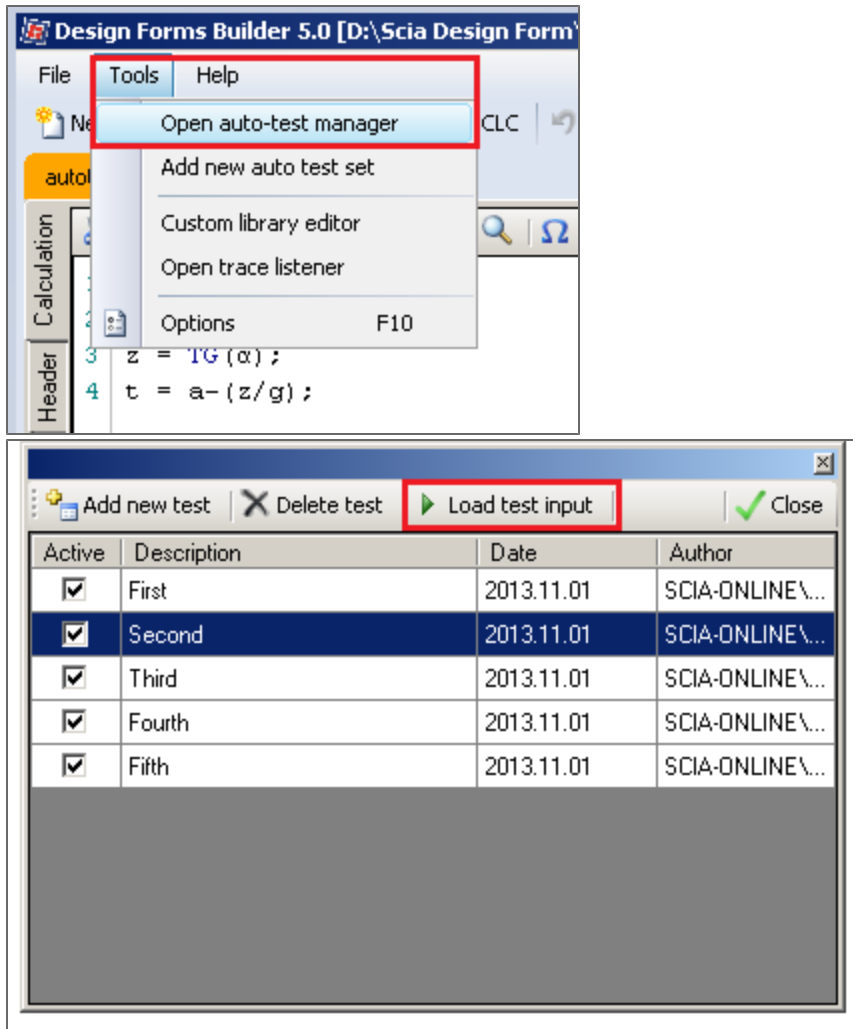
2. Go to Tools / Add new auto test set and write description for the test (e.g. its name - First).



3. Change some input variables (variables with green value cell), Refresh calculation and add next auto-test (repeat step 2). Repeat this four times and create approx 4-5 tests

ID	Description	Symbol	Value	Unit	Preciso
		b	20		2
		c	4		2
		d	5		2
Result.0		a	29		2
		g	7.9		2
		f	555		2
Result.1		h	-547		2
Result.2		z	0.933		2
Result.3		t	28.9		2
		α	43		2

4. Prepare once more the input values. Go to Tools / Open auto test manager. Add another test here by Add test set. Delete some set. Load test data stored in particular test (check that values are changed in the table of variables).



Everything for the automatic test is prepared.

Test data file is stored in same directory as CLS file.

5. Open TestIO file. There are test name, author, date and all your variables marked as Input or Result. File is in the XML format.

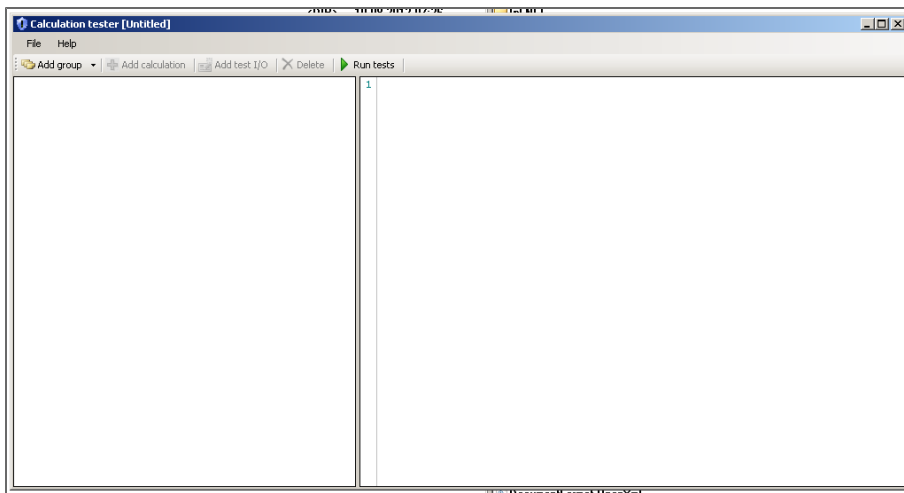
```

<Root>
<Test Description="FIRST" Author="SCIA-ONLINE\jkrstik" Active="1"
Date="2013.11.01">
<Dialog />
<Result Name="a" Type="Numeric" ESA_ID="Result.0" Value="29" />
<Input Name="b" Type="Numeric" ESA_ID="" Value="20" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="4" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="5" />
<Input Name="f" Type="Numeric" ESA_ID="" Value="555" />
<Input Name="g" Type="Numeric" ESA_ID="" Value="7.9" />
<Result Name="h" Type="Numeric" ESA_ID="Result.1" Value="-547.1" />
<Result Name="t" Type="Numeric" ESA_ID="Result.3" Value="28.88196" />
<Result Name="z" Type="Numeric" ESA_ID="Result.2" Value="932.5151e-3" />
<Input Name="Iz" Type="Numeric" ESA_ID="" Value="43" />
</Test>
<Test Description="Second" Author="SCIA-ONLINE\jkrstik" Active="1"
Date="2013.11.01">
<Dialog />
<Result Name="a" Type="Numeric" ESA_ID="Result.0" Value="45" />
<Input Name="b" Type="Numeric" ESA_ID="" Value="20" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="20" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="5" />
<Input Name="f" Type="Numeric" ESA_ID="" Value="450" />
<Input Name="g" Type="Numeric" ESA_ID="" Value="7.9" />
<Result Name="h" Type="Numeric" ESA_ID="Result.1" Value="-442.1" />
<Result Name="t" Type="Numeric" ESA_ID="Result.3" Value="44.95887" />
<Result Name="z" Type="Numeric" ESA_ID="Result.2" Value="324.9197e-3" />
<Input Name="Iz" Type="Numeric" ESA_ID="" Value="48" />
</Test>
<Test Description="Third" Author="SCIA-ONLINE\jkrstik" Active="1"
Date="2013.11.01">
<Dialog />
<Result Name="a" Type="Numeric" ESA_ID="Result.0" Value="87.5" />
<Input Name="b" Type="Numeric" ESA_ID="" Value="45" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="10" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="32.5" />
<Input Name="f" Type="Numeric" ESA_ID="" Value="155" />
<Input Name="g" Type="Numeric" ESA_ID="" Value="4.2" />
<Result Name="h" Type="Numeric" ESA_ID="Result.1" Value="-150.8" />
<Result Name="t" Type="Numeric" ESA_ID="Result.3" Value="89.19414" />
<Result Name="z" Type="Numeric" ESA_ID="Result.2" Value="-7.11537" />
<Input Name="Iz" Type="Numeric" ESA_ID="" Value="98" />
</Test>
<Test Description="Fourth" Author="SCIA-ONLINE\jkrstik" Active="1"
Date="2013.11.01">
<Dialog />
<Result Name="a" Type="Numeric" ESA_ID="Result.0" Value="432" />
<Input Name="b" Type="Numeric" ESA_ID="" Value="24" />
<Input Name="c" Type="Numeric" ESA_ID="" Value="384" />
<Input Name="d" Type="Numeric" ESA_ID="" Value="24" />
<Input Name="f" Type="Numeric" ESA_ID="" Value="24" />
<Input Name="g" Type="Numeric" ESA_ID="" Value="1235" />
<Result Name="h" Type="Numeric" ESA_ID="Result.1" Value="1211" />
<Result Name="t" Type="Numeric" ESA_ID="Result.3" Value="494.007" />

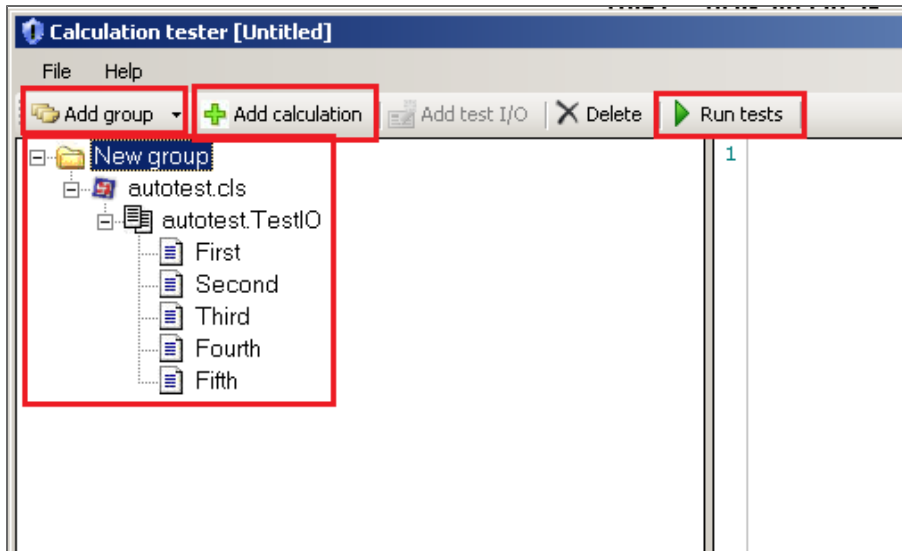
```

Auto tests runs in another application named CalculationTester.exe. This application is in the SDF installation directory of SDF (standard path is c:\Program Files (x86)\SCIA\DesignForms5).

6. Start CalculationTester.exe. This is a tool for running and evaluating automatic tests.



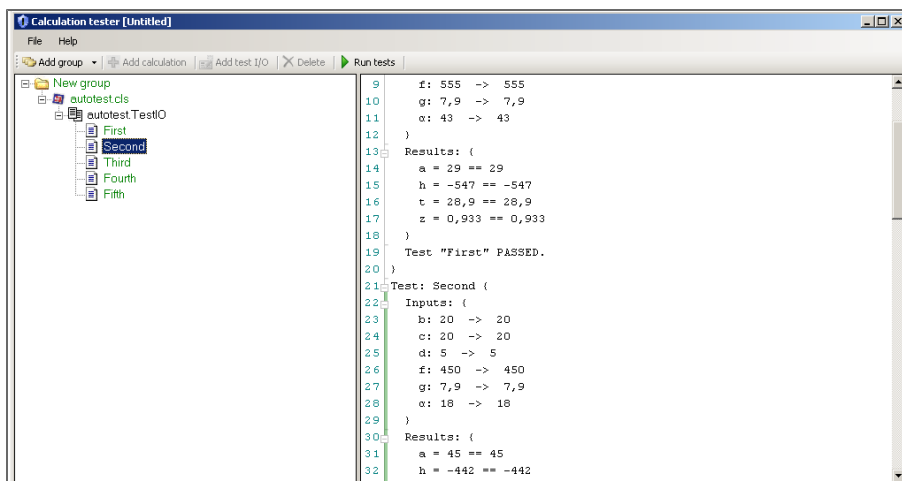
7. Click on a New group button, select this group in the list and click on Add calculation. Find autotest.cls and open it. TestIO is automatically added with CLS file. The list of all test sets is shown in the left part. Click on button Run tests.



How it works:

- 1) CLS is used as source: Compiler is automatically started, CLS is compiled to CLC and it is calculated. The results are immediately displayed. If test passed, it is green (on the left and on the right part). If test failed, it is red with mark of what is wrong.
- 2) CLC is used as source (already compiled): Compiler didn't run and the calculation starts. This process is faster.

8. Check the results in the right part of the window. If the test runs without problems, each item on the left and on the right part is green.



Only variable types Double, String and Boolean can be used in auto test. Not arrays! If you have calculation in arrays, it is necessary to create Result double variable which will represent just one number from the array.