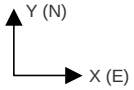
COGO Function Reference

Table of Contents

Coordinate Notation



All functions use XY coordinate notation. X and Y correspond to E and N, respectively in terms of North/East coordinates. Specify E, N (in this order) when entering North/East coordinates.

Referencing and Entering Points

Point coordinates and COGO functions that take them must be placed on the same worksheet.
In COGO functions, points are reffered to by either cell range reference to directly entered coordinates or cell reference to X/Y text string (e.g. [Pt->wp2]«200,50»). All point creating functions return X/Y text string.

Points with known coordinates may be entered directly in the adjacent cells on the same row (tag, X-crd, Y-crd) or entered using function *p_xy().*

Direct Input:

Pt.	X	Y
wp1	100	200

Function p_xy():

[Pt->wp2]«200,50» ⇦ X/Y text string returned by function p_xy() *=p_xy("wp2",200,50)*

Directly entered coordinates may be converted into X/Y text string using *c_xy()* function

Convert wp1 coordinates into X/Y string:

[Pt->wp1]«100,200» *=c_xy(A99:C99)*

To extract point tag and coordinates fron X/Y text string use *g_tag()* , *g_x()* and *g_y()* functions

Extract wp2 tag and coordinates:

Pt		X		Y	
wp2	*=g_tag(A102)*	200	*=g_x(A102)*	50	*=g_y(A102)*

Point tag (e.g. "wp2") is required part of point objects in COGO functions. Tags have no effect on calculations and included for point identification and tracking purposes only. In comparison, point references (e.g. A24:C24) are function arguments pointing to the cell(s) containing point coordinates that are used in the spreadsheet calculations. Because there are no imposed restrictions on tag naming, geometrically different points are allowed to have identical tags.

Optional Function Arguments

Some functions have optional arguments. Function use default value/setting when optional argument is omitted. When entering function, commas separating arguments must be placed up to the last specified optional argument. Comma for omitted optional argument must be included in case when at least one other optional argument that follows is explicitly specified.

For example, function *p_arca(new_pt_tag, arcp, ccp, ctr_ang, opt_ofs, opt_incl_descr)* has two optional parameters, *opt_ofs* and *opt_incl_descr.*

If implicit defaults are to be used for both optional arguments, then function is entered as follows:
p_arca(new_pt_tag, arcp, ccp, ctr_ang)

If implicit default value is to be used for *opt_ofs* only, then function is entered as follows:
p_arca(new_pt_tag, arcp, ccp, ctr_ang, , opt_incl_descr)

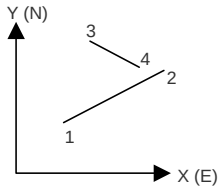
If implicid default value is to be used for *opt_incl_descr* only, then function is entered as follows:
p_arca(new_pt_tag, arcp, ccp, ctr_ang, opt_ofs)

Angle Types

Absolute angle measured relative to X-axis positive direction (positive angle: counterclockwise)
Azimuth clockwise angle measured relative to North direction (Y-axis positive direction)
Bearing directional angle in NE/NW/SE/SW notation
Relative angle between any two lines

Examples:

line 1-2 abs ang =	38°-23'-34"	*=ang_(38,23,34)*
line 1-2 az =	51°-36'-26"	*=dms(c_aang_az(deg(C149)))*
line 1-2 brg =	N51°-36'-26"E	*=c_aang_brg(deg(C149))*
line 3-4 brg =	S73°-12'-4"E	*=bang_("se",73,12,4)*
Lines 1-2/3-4 ang =	55°-11'-30"	*=dms(c_brg_az(C152)-deg(C150))*



For functions used in the above examples, see *Angle Conversion/Format* functions.

Entering/Editing/Inspecting Functions

Consider using build-in Excel Function Dialog Box for entering, editing and inspecting functions with multiple arguments.

To enter function using Excel Function Dialog Box:
Select target cell, click ***fx*** button on the Excel Menu Bar, choose 'User Defined' category, navigate to and choose target function.
Alternatively, enter function omitting arguments but include first round parenthesis bracket (e.g. =p_rang( ), press **Enter**, reselect target cell holding function, click ***fx*** button --> opens Excel Function Dialog Box.

To edit or inspect function using Excel Function Dialog Box:
Select target cell and click ***fx*** button on the Excel Menu Bar --> opens Excel Function Dialog Box.

Other function inspection options are as follows:

- Select target cell and press **F2** button --> opens pop-up window with extended formula information.
- Use *pr_frml()* function (see Cell Formula function in the Select Custom Function/Macro Reference).
- Set optional *opt_incl_descr* function argument (included in many COGO functions) to return extended COGO function information.

Note that above inspection methods may not be comprehensive when COGO functions are used inside other functions.

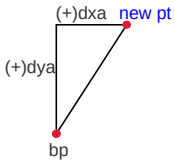
	A	B	C	D	E	F	G	H	I	J
179	**p_xy()**		Locate point by coordinates							
180	p_xy(*new_pt_tag, x, y*)									
181										
182	*new_pt_tag*		new point tag (enclose not-numeric tag by quotation marks							
183			when entering directly into function)							
184	*x, y*		new point coordinates							
185										
186	Function returns X/Y text string containing embedded tag and coordinates of new point									
187										
188	*Example:*									
189	[Pt->82]«100.25,-12.67»					*=p_xy(82,100.25,-12.67)*				
190										
191										

p_dxdy()

Locate point from another point by relative x and y offsets

p_dxdy(*new_pt_tag, bp, dx, dy, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
bp	[Ref] to bp
dx	X-offset from *bp to new pt; (+)dx-->* in (+)X-axis direction
dy	Y-offset from *bp to new pt; (+)dy-->* in (+)Y-axis direction
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



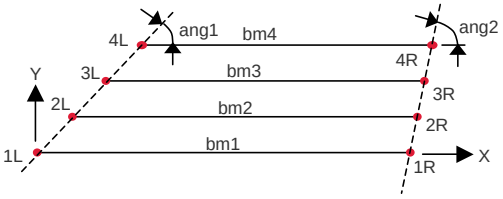
[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Objective:

Locate beam points @ CL support brgs



Given:

bm1 L =	50.00
ang1 =	50.00 deg
ang2 =	80.00 deg
bm spa =	5.50

Solution:

Assign bm1 coordinates:

[Pt->1L]«0,0»		*=p_xy("1L",0,0)*
[Pt->1R]«50,0»		*=p_xy("1R",B212,0)*

Calculate dx & dy:

dxL=	4.615	*=ROUND(5.5/tan_(50),4)*
dxR =	0.9698	*=ROUND(5.5/tan_(80),4)*
dyL = dyR =	5.50	

Create beam points:

2L	[Pt->2L]«4.615,5.5»	*=p_dxdy(A228,A219,$B$223,$B$225)*
3L	[Pt->3L]«9.23,11»	*=p_dxdy(A229,B228,$B$223,$B$225)*
4L	[Pt->4L]«13.845,16.5»	*=p_dxdy(A230,B229,$B$223,$B$225)*
2R	[Pt->2R]«50.9698,5.5»	*=p_dxdy(A231,A220,$B$224,$B$225)*
3R	[Pt->3R]«51.9396,11»	*=p_dxdy(A232,B231,$B$224,$B$225)*
4R	[Pt->4R]«52.9094,16.5»	*=p_dxdy(A233,B232,$B$224,$B$225)*

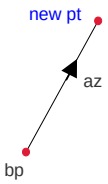
Pt 4R Extended output option:

[Pt->4R][@ <dx = 4.615, dy = 5.5> from Pt->3R]«56.5546,16.5»	*=p_dxdy(A233,B232,$B$223,$B$225,1)*

p_az() Locate point from another point by azimuth, distance

p_az(*new_pt_tag, bp, az, dist, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
bp	[Ref] to bp
az	azimuth angle (see *Angle Types*) in degrees from *bp* to *new pt*
dist	distance from *bp* to *new pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

			Line [*bp - new pt*]		
Pt	X	Y	bearing	dist	new pt
bp	10.5	-12.5	S48°-3'-4"W	34.7	np

Convert bearing to azimuth:

az = 228.0511111 *=c_brg_az(D256)*

Locate point *np* :

[Pt->np]«-15.3078277067365,-35.6958192150957» *=p_az(F256,A256:C256,B259,E256)*

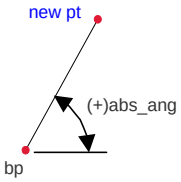
Extended output option:

[Pt->np][@ <az = 228.051111111111, dist = 34.7> from Pt->bp]«-15.3078277067365,-35.6958192150957»

=p_az(F256,A256:C256,B259,E256,1)

p_aang() Locate point from another point by absolute angle, distance

p_aang(*new_pt_tag, bp, abs_ang, dist, opt_incl_descr*)



new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
bp	[Ref] to bp
abs_ang	angle in degrees from (+)X-direction to direction of line [*bp -new pt*] (+)*abs_ang* --> counterclockwise
dist	distance from *bp* to *new pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

			Line [bp - new pt]		
Pt	X	Y	abs_ang	dist	new pt
bp	10.5	-12.5	221.9489	34.7	np

Locate point *np* :

[Pt->np]«-15.3078277067365,-35.6958192150957» *=p_aang(F286,A286:C286,D286,E286)*

Extended output option:

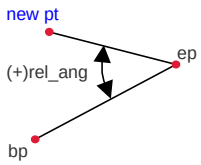
[Pt->np][@ <abs ang = 221.948888888889, dist = 34.7> from Pt->bp]«-15.3078277067365,-35.6958192150957»

=p_aang(F286,A286:C286,D286,E286,1)

p_rang() Locate point from end of line by relative angle, distance

p_rang(*new_pt_tag, bp, ep, rel_ang, dist, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
bp	[Ref] to bp
ep	[Ref] to ep
rel_ang	angle from line [*bp-ep*] to line [*ep-new pt*] in degrees
	(+)*rel_ang* --> Clockwise
dist	distance from *ep* to *new pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

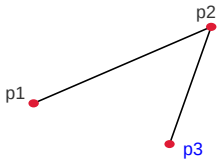
Example:

Objective:

Get point p3 coordinates

Given:

p1 coordinates:	[Pt->p1]«0,0»	*=p_xy("p1",0,0)*
line [p1-p2] Azimuth =	60.0	
line [p1-p2]/[p2-p3] Ang =	45.0	
p1-p2 dist =	10.0	
p2-p3 dist =	8.0	



Solution:

[Pt->p2]«8.66025403784439,5» *=p_az("p2",C318,C319,C321)*

[Pt->p3]«6.58970167702422,-2.72740661031255» *=p_rang("p3",C318,A326,-C320,C322)*

Extended output option:

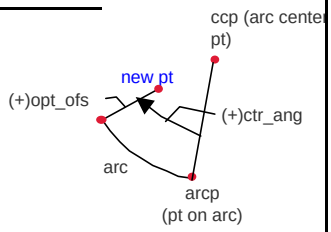
[Pt->p3][@ <rel ang = -45, dist = 8> from end pt of line <bp->p1, ep->p2>]«6.58970167702422,-2.72740661031255»

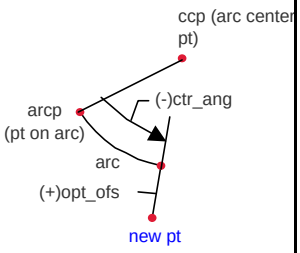
=p_rang("p3",C318,A326,-C320,C322,1)

p_arca() Locate point from point on arc by arc-angle, offset

p_arca(*new_pt_tag, arcp, ccp, ctr_ang, opt_ofs, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
arcp	[Ref] to arcp
ccp	[Ref] to ccp
ctr_ang	arc [*arcp-new pt*] central angle in degrees
	(+)*ctr_ang* --> Clockwise
opt_ofs	optional offset (default = 0)
	(+)*opt_ofs* --> on right side of arc looking from *arcp* towards *new pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)





[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Pt	X	Y			
arcp	34.6	-45.8	ctr ang	offset	new Pt
ccp	12.0	12.0	30.0	10.0	np

Locate point *np* :

[Pt->np]«4.17517710447816,-39.4698644401904» *=p_arca(F354,A353:C353,A354:C354,D354,E354)*

Extended output option:

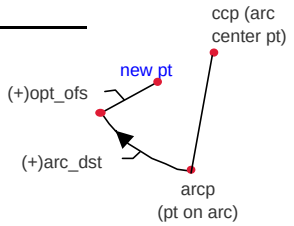
[Pt->np][@ <ctr ang = 30, ofs = 10> from Pt->arcp on arc <ccp->ccp]«4.17517710447816,-39.4698644401904»

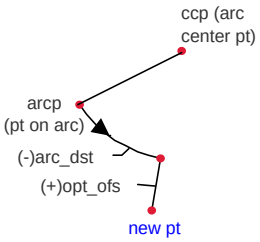
=p_arca(F354,A353:C353,A354:C354,D354,E354,1)

p_arcl() Locate point from point on arc by arc-distance, offset

p_arcl(*new_pt_tag, arcp, ccp, arc_dst, opt_ofs, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
arcp	[Ref] to arcp
ccp	[Ref] to ccp
arc_dst	arc distance from *arcp* to *new pt;* *(+)arc_dst* --> Clockwise
opt_ofs	optional offset (default = 0) (+)*opt_ofs* --> on right side of arc looking from *arcp* towards *new pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)





[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Pt	X	Y			
arcp	34.6	-45.8	arc_len	offset	new Pt
ccp	12.0	12.0	32.4952	10.0	np

Locate point *np* :

[Pt->np]«4.17517692078978,-39.4698644122647» *=p_arcl(F383,A382:C382,A383:C383,D383,E383)*

Extended output option:

[Pt->np][@ <arc dist = 32.4952, ofs = 10> from Pt->ccp on arc <ccp->ccp]«4.17517692078978,-39.4698644122647»

=p_arcl(F383,A382:C382,A383:C383,D383,E383,1)

p_bso() Locate point relative to baseline by station, offset

p_bso(*new_pt_tag, sta, ofs, gblock_rng, ref_sta, opt_incl_descr*)

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
sta	new point station
ofs	new point offset; (+)*ofs* --> on right side of baseline looking station ahead
gblock_rng	cell range reference to gblock defining baseline (see *Special Objects*)
ref_sta	station of first point in the baseline definition
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)

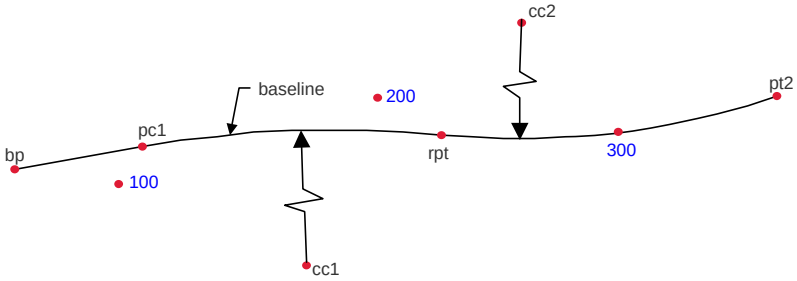
Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Objective:

Locate the following points by station and offset:

Pt	sta	ofs
100	10+40.00	15.00
200	11+50.00	-15.00
300	12+50.00	0.00



Given:

Points along baseline:

Point	E	N	Sta
bp	22.1073	434.8218	10+00.00
pc1	74.6152	445.4554	
cc1	154.0089	53.4137	
rpt	197.1418	451.0814	
cc2	229.4915	749.3321	
pt2	335.1841	468.567	

Solution:

Define baseline:

```
<gblock 1>
[Pt->bp]«22.1073,434.8218»                                  =c_xy(A418:C418)
[Pt->pc1]«74.6152,445.4554»                                 =c_xy(A419:C419)
<arc>
[Pt->cc1]«154.0089,53.4137»                                 =c_xy(A420:C420)
[Pt->rpt]«197.1418,451.0814»                                =c_xy(A421:C421)
<arc>(-clock)
[Pt->cc2]«229.4915,749.3321»                                =c_xy(A422:C422)
[Pt->pt2]«335.1841,468.567»                                 =c_xy(A423:C423)
<end gb>
```

Locate points:

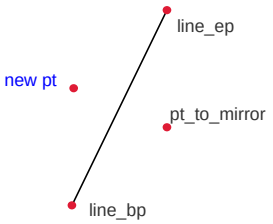
```
[Pt->100]«64.2887323443607,428.059643141561»                =p_bso(A411,B411,C411,$A$428:$A$437,$D$418)
[Pt->200]«171.124586593891,468.060669651529»                =p_bso(A412,B412,C412,$A$428:$A$437,$D$418)
[Pt->300]«270.237251696264,452.112035301658»                =p_bso(A413,B413,C413,$A$428:$A$437,$D$418)
```

Pt 300 extended output option:

```
[Pt->300][@ <sta = 1250, ofs = 0>; BL defined in <gblock 1> (ref sta = 1000)]«270.237251696264,452.112035301658»
=p_bso(A413,B413,C413,$A$428:$A$437,$D$418,1)
```

p_mir() Locate point by mirroring another point

p_mir(*new_pt_tag, pt_to_mirror, line_bp, line_ep, opt_incl_descr*)



new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
pt_to_mirror	[Ref] to pt_to_mirror
line_bp	[Ref] to line_bp
line_ep	[Ref] to line_ep
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)

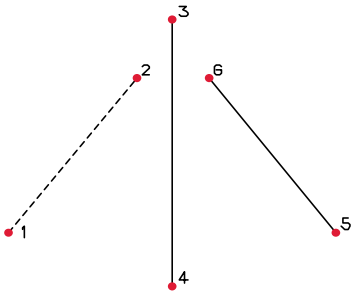
[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Objective:

Create line [5-6] by mirroring line [1-2] about vertical line [3-4]



Given:

[Pt->1]«10.3,1.45»	*=p_xy(1,10.3,1.45)*
[Pt->2]«20.7,15.6»	*=p_xy(2,20.7,15.6)*
[Pt->3]«23.7,21»	*=p_xy(3,23.7,21)*
[Pt->4]«23.7,-3.5»	*=p_xy(4,23.7,-3.5)*

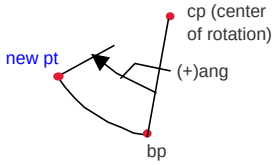
Solution:

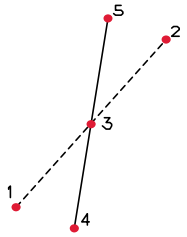
[Pt->5]«37.1,1.45»	*=p_mir(5,A470,A472,A473)*
[Pt->6]«26.7,15.6»	*=p_mir(6,A471,A472,A473)*

Pt 6 Extended output option:

[Pt->6][by mirror of Pt->2 about line <bp->3, ep->4>]«26.7,15.6»	*=p_mir(6,A471,A472,A473,1)*

	A	B	C	D	E	F	G	H	I	J
483	**p_rot()**		Locate point by rotating another point							
484	p_rot(*new_pt_tag, bp, cp, ang, opt_incl_descr*)									
485										
486	*new_pt_tag*		new point tag (enclose not-numeric tag by quotation marks							
487			when entering directly into function)							
488	*bp*		[Ref] to bp							
489	*cp*		[Ref] to cp							
490	*ang*		rotation angle in degrees; (+)*ang* --> Clockwise							
491	*opt_incl_descr*		optional argument --> enter 1 for extended info (default -> no info)							
492										
493	[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates									
494										
495	Function returns X/Y text string containing embedded tag and coordinates of new point									
496										
497	Example:									
498										
499	Objective:									
500	Create line [4-5] by rotating line[1-2] by -30 deg about its midpoint									
501										
502	Given:									
503	pt	X	Y							
504	1	39.3108	37.9696							
505	2	69.0648	76.3618							
506										
507	Solution:									
508	[1-2] L =	48.57223				*=g_dst(A504:C504,A505:C505)*				
509										
510	Locate line [1-2] midpoint:									
511	[Pt->3]«54.1878,57.1657»					*=p_rang(3,A504:C504,A505:C505,0,B508/2)*				
512										
513	Rotate line [1-2]:									
514	[Pt->4]«50.9019900678989,33.1028897464135»					*=p_rot(4,A504:C504,$A$511,-30)*				
515	[Pt->5]«57.4736099321011,81.2285102535865»					*=p_rot(5,A505:C505,$A$511,-30)*				
516										
517	Pt 5 extended output option:									
518	[Pt->5][by rotating Pt->2 about Pt->3 by <ang = -30>]«57.4736099321011,81.2285102535865»									
519	*=p_rot(5,A505:C505,$A$511,-30,1)*									
520										
521										

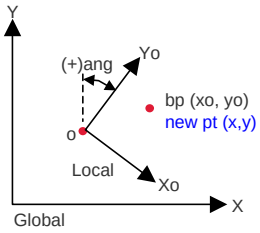




p_xytr()

Locate point by transforming its local coordinates

p_xytr(*new_pt_tag, bp, ang, dx, dy, opt_incl_descr*)



new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
bp	[Ref] to bp (local coordinates)
ang	angle in degrees from global Y-axis to local Yo-axis (+)*ang* --> Clockwise
dx, dy	global x & y coordinates of origin (o) of local system of crd's
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)

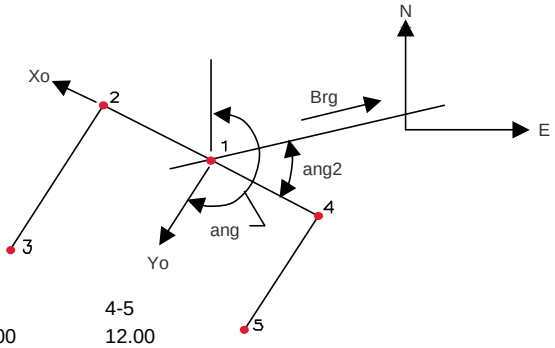
[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

Objective:

Get pt2÷pt5 E, N coordinates



Given:

origin pt	E (dx)	N (dy)
1	-30.00	-5.00
Brg =	N75°-0'-0"E	
ang2 =	45.00	

	1-2	1-4	2-3	4-5
L =	10.00	10.00	15.00	12.00

Solution:

pt2÷pt5 local coordinates:

pt	Xo	Yo
2	10.00	0.00
3	10.00	15.00
4	-10.00	0.00
5	-10.00	12.00

ang = 210.00 *=c_brg_az(N75°-0'-0"E)+45+90*

pt2÷pt5 E, N coordinates:

[Pt->2]«-38.6602540378444,1.11022302462516E-15»	*=p_xytr(A555,A555:C555,$B$560,$B$544,$C$544)*
[Pt->3]«-46.1602540378444,-12.9903810567666»	*=p_xytr(A556,A556:C556,$B$560,$B$544,$C$544)*
[Pt->4]«-21.3397459621556,-10»	*=p_xytr(A557,A557:C557,$B$560,$B$544,$C$544)*
[Pt->5]«-27.3397459621556,-20.3923048454133»	*=p_xytr(A558,A558:C558,$B$560,$B$544,$C$544)*

Pt 5 extended output option:

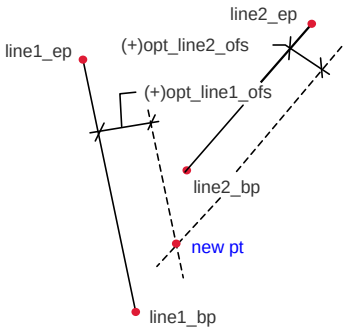
[Pt->5][by transforming Pt->5 crds by <ang = 210, dx = -30, dy = -5>]«-27.3397459621556,-20.3923048454133»

=p_xytr(A558,A558:C558,$B$560,$B$544,$C$544,1)

i_ll() Locate point at intersection of two lines

i_ll(*new_pt_tag, line1_bp, line1_ep, line2_bp, line2_ep, opt_line1_ofs, opt_line2_ofs, opt_incl_descr*

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
line1_bp	[Ref] to line1_bp
line1_ep	[Ref] to line1_ep
line2_bp	[Ref] to line2_bp
line2_ep	[Ref] to line2_ep
opt_line1_ofs	optional line 1 offset (default = 0)
	(+)opt_line1_ofs --> on right side of line 1 looking towards *line1_ep*
opt_line2_ofs	optional line 2 offset (default = 0)
	(+)opt_line2_ofs --> on right side of line 2 looking towards *line2_ep*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point

Example:

	Pt	X	Y	Line 1 ofs	Line 2 ofs
Line 1 bp	1	-23.3	12.0	-5.70	9.50
Line 1 ep	2	-5.7	34.4		
Line 2 bp	3	-30.5	20.7		
Line 2 ep	4	-30.0	28.9		

Locate pt 5 @ intersection of lines 1 and 2:

[Pt->5]«-20.7525923409725,24.4681201791404» *=i_ll(5,B594:D594,B595:D595,B596:D596,B597:D597,E594,F594)*

Pt 5 extended output option:

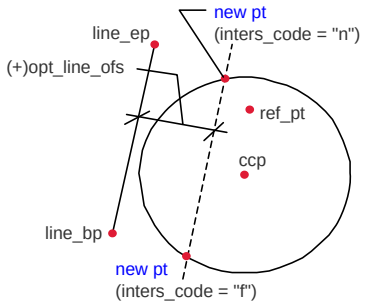
[Pt->5][@ inters of line1 @ -5.7 from <bp->1, ep->2> and line2 @ 9.5 from <bp->3, ep->4>]«-20.7525923409725,24.4681201791404»

=i_ll(5,B594:D594,B595:D595,B596:D596,B597:D597,E594,F594,1)

i_lc()

Locate point at intersection of line and circle

i_lc(*new_pt_tag, line_bp, line_ep, circ_ccp, circ_rad, ref_pt, inters_code, opt_line_ofs, opt_incl_descr*



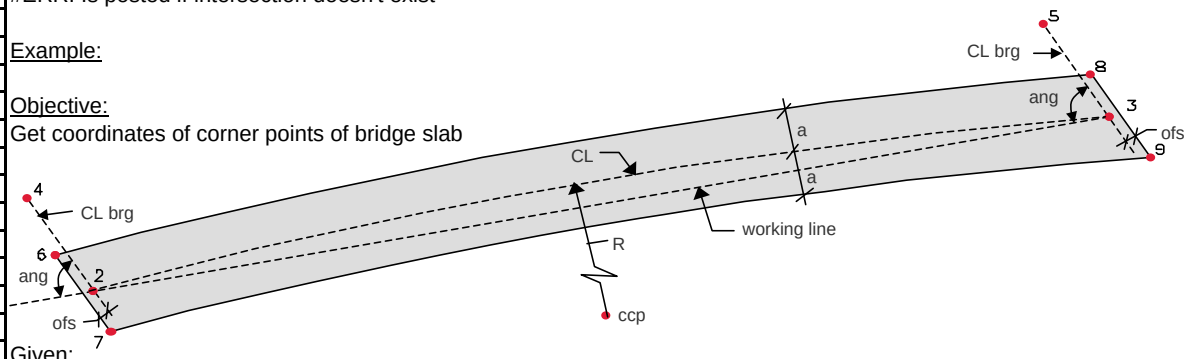
new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
line_bp	[Ref] to line_bp
line_ep	[Ref] to line_ep
circ_ccp	[Ref] to circ_ccp
circ_rad	circle radius
ref_pt	[Ref] to ref_pt *ref_pt* --> any known point except *ccp*
inters_code	"n" or "f" "n" --> *new pt* @ intersection point that is nearest to *ref_pt* "f" --> *new pt* @ intersection point that is further away from *ref_pt*
opt_line_ofs	optional line offset (default = 0) *(+)opt_line_ofs* --> on right side of line looking from *line_bp* towards *line_ep*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point
#ERR! Is posted if intersection doesn't exist

Example:

Objective:
Get coordinates of corner points of bridge slab



Given:

Pt	X	Y	R	a	ang	ofs
CCP	100.0000	-1000.0000	2000.00	20.00	70.00	4.00
2	-488.9584	911.3158				
3	-88.9584	991.0537				

Solution:

Locate CL brg working points:

[Pt->4]«-514.914838197759,954.05060216505» *=p_rang(4,A645:C645,A644:C644,-(180-F643),50)*
[Pt->5]«-114.914838197758,1033.78850216505» *=p_rang(5,A644:C644,A645:C645,F643,50)*

Locate slab corner points:

[Pt->6]«-503.596799411934,927.711312344167» *=i_lc(6,A644:C644,A649,A643:C643,$D$643+$E$643,A644:C644,"n",-$G$643)*
[Pt->7]«-482.193142593913,892.472230897094» *=i_lc(7,A644:C644,A649,A643:C643,$D$643-$E$643,A644:C644,"n",-$G$643)*
[Pt->8]«-96.0667242933865,1010.46209604281» *=i_lc(8,A645:C645,A650,A643:C643,$D$643+$E$643,A645:C645,"n",$G$643)*
[Pt->9]«-72.9669913326633,972.430586841861» *=i_lc(9,A645:C645,A650,A643:C643,$D$643-$E$643,A645:C645,"n",$G$643)*

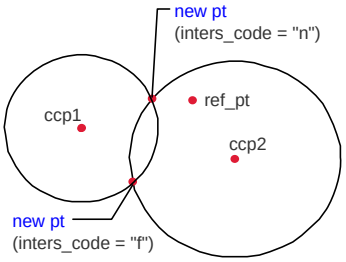
Pt 9 extended output option:

[Pt->9][@ inters of line @ 4 from <bp->3, ep->5> and circ <ccp->CCP, rad=1980> near Pt->3]«-72.9669913326633,972.430586841861»
=i_lc(9,A645:C645,A650,A643:C643,$D$643-$E$643,A645:C645,"n",$G$643,1)

i_cc() Locate point at intersection of two circles

i_cc(*new_pt_tag, circ1_ccp, circ1_rad, circ2_ccp, circ2_rad, ref_pt, inters_code, opt_incl_descr*

new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
circ1_ccp	[Ref] to circ1_ccp
circ1_rad	circle1 radius
circ2_ccp	[Ref] to circ2_ccp
circ2_rad	circle2 radius
ref_pt	[Ref] to ref_pt *ref_pt* --> any known point except *ccp1 and ccp2*
inters_code	"n" or "f" "n" --> *new pt* @ intersection that is nearest to *ref_pt* "f" --> *new pt* @ intersection that is further away from *ref_pt*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point
#ERR! Is posted if intersection doesn't exist

Example:

	Pt	X	Y	R
arc1	cc1	12.0	12.0	18.0
arc2	cc2	-10.0	5.0	10.0
ref point	rp	4.0	4.0	

Locate arc1/arc2 intersection points:

[Pt->np]«-1.369874128051,-5.18241689825771E-02» *=I_cc("np",B686:D686,E686,B687:D687,E687,B688:D688,"n")*
[Pt->fp]«-5.87590448358127,14.1099855198268» *=I_cc("fp",B686:D686,E686,B687:D687,E687,B688:D688,"f")*

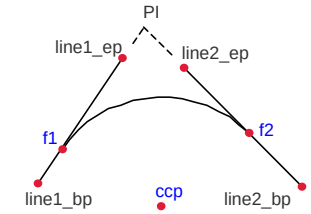
Point fp extended output option:

[Pt->fp][@ inters of circ1 <ccp->cc1, rad =18> and circ2 <ccp->cc2, rad=10> far from Pt->rp]«-5.87590448358127,14.1099855198268»
=I_cc("fp",B686:D686,E686,B687:D687,E687,B688:D688,"f",1)

i_flt() Locate circular fillet points at intersection of two lines

i_flt(*new_pt_tag, line1_bp, line1_ep, line2_bp, line2_ep, rad, output code, opt_incl_descr*

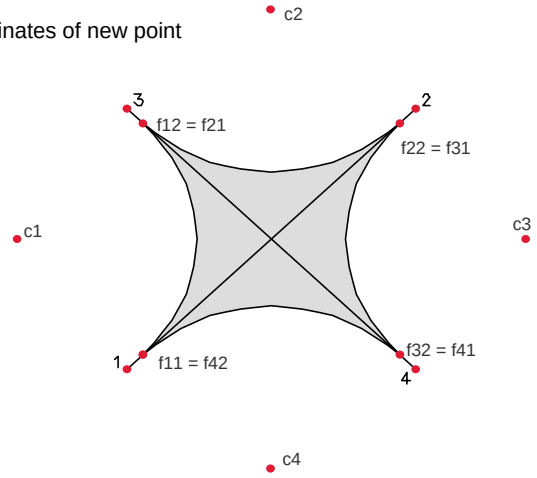
new_pt_tag	new point tag (enclose not-numeric tag by quotation marks when entering directly into function)
line1_bp	[Ref] to line1_bp
line1_ep	[Ref] to line1_ep
line2_bp	[Ref] to line2_bp
line2_ep	[Ref] to line2_ep
rad	fillet radius
output code	codes: "f1" or "f2" or "cc" "f1" --> function returns *f1* "f2" --> function returns *f2* "cc" --> function returns *ccp*
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)



Note:
Fillet is always placed inside angle [line1_bp - PI - line2_bp]

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns X/Y text string containing embedded tag and coordinates of new point
Error codes are posted if fillet doesn't fit



Example:

Objective:
Create shape bounded by fillets (R = 5.0) @ intersection of two lines

Given:

line 1	[Pt->1]«6,4»	*=p_xy(1,6,4)*
"	[Pt->2]«14,12»	*=p_xy(2,14,12)*
line 2	[Pt->3]«6,12»	*=p_xy(3,6,12)*
"	[Pt->4]«14,4»	*=p_xy(4,14,4)*

Fit fillet 1:

[Pt->f11]«6.46446609406726,4.46446609406726»	= f42 =	*=i_flt("f11",B728,B729,B730,B731,5,"f1")*
[Pt->f12]«6.46446609406726,11.5355339059327»	= f12 =	*=i_flt("f12",B728,B729,B730,B731,5,"f2")*
[Pt->c1]«2.92893218813452,8»		*=i_flt("c1",B728,B729,B730,B731,5,"cc")*

Fit fillet 2:

[Pt->f22]«13.5355339059327,11.5355339059327»	= f31 =	*=i_flt("f22",B730,B731,B729,B728,5,"f2")*
[Pt->c2]«10,15.0710678118655»		*=i_flt("c2",B730,B731,B729,B728,5,"cc")*

fit fillet 3:

[Pt->f32]«13.5355339059327,4.46446609406726»	= f41 =	*=i_flt("f32",B729,B728,B731,B730,5,"f2")*
[Pt->c3]«17.0710678118655,8»		*=i_flt("c3",B729,B728,B731,B730,5,"cc")*

Fit fillet 4:

[Pt->c4]«10,0.928932188134524»	*=i_flt("c4",B731,B730,B728,B729,5,"cc")*

Point c4 extended output option:

[Pt->c4][cc of fillet <R = 5> @ inters of line1 <bp->4, ep->3> with line2 <bp->1, ep->2>]«10,0.928932188134524»

=i_flt("c4",B731,B730,B728,B729,5,"cc",1)

g_dst() Get distance between two points

g_dst(*pt1, pt2, opt_incl_descr*)

pt1	[Ref] to pt1
pt2	[Ref] to pt2
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info) returned string with info can't be used in math calculations

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

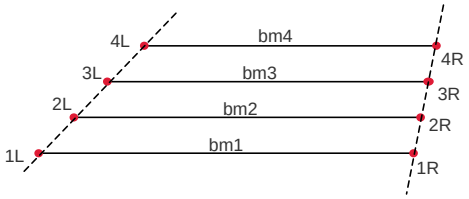
Function returns distance between points *p1* and *p2*

Example:

Tabulate beam lengths (ft-in) from example included with function *p_dxdy()*

Copy previously obtained points:

[Pt->1L]«0,0»	*=A219*
[Pt->1R]«50,0»	*=A220*
[Pt->2L]«4.615,5.5»	*=B228*
[Pt->2R]«50.9698,5.5»	*=B231*
[Pt->3L]«9.23,11»	*=B229*
[Pt->3R]«51.9396,11»	*=B232*
[Pt->4R]«52.9094,16.5»	*=B233*
[Pt->4L]«13.845,16.5»	*=B230*



bm	L	
1	50'-0"	*=fi(g_dst(A767,A768))*
2	46'-4 1/4"	*=fi(g_dst(A769,A770))*
3	42'-8 1/2"	*=fi(g_dst(A771,A772))*
4	39'-0 3/4"	*=fi(g_dst(A773,A774))*

Bm 4 extended output option:

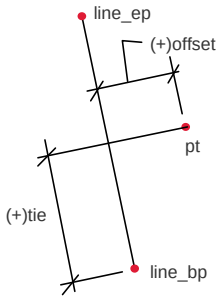
39.0644 [dist between Pt->4R and Pt->4L] *fi() omitted because it can't be used with text string*

=g_dst(A773,A774,1)

g_lo() Get point offset and tie to line

g_lo(*pt, line_bp, line_ep, opt_output_code*)

pt	[Ref] to pt
line_bp	[Ref] to line_bp
line_ep	[Ref] to line_ep
opt_output_code	optional codes: "o" or "t" or "e" (Default --> "e")
	"o" --> function returns *offset*
	"t" --> function returns *tie*
	"e" --> function returns text string with extended info (string can't be used in math calculations)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns point *pt offset & tie* relative to line [*line_bp - line-ep*]
(+)*offset* --> on right side of line looking from *line_bp* towards *line_ep*
(+)*tie* --> running from *line_bp* towards *line_ep*

Example:

Get offset and tie of point 100 relative to tangent [bp-pc1] (points are from example included with function *p_bso())*

Copy previously obtained points:

[Pt->100]«64.2887323443607,428.059643141561»	*=A440*	
[Pt->bp]«22.1073,434.8218»	*=A429*	
[Pt->pc1]«74.6152,445.4554»	*=A430*	

Point 100 offset and tie dimensions:

offset =	15.00	*=g_lo(A809,A810,A811,"o")*
tie =	40.00	*=g_lo(A809,A810,A811,"t")*

Extended output option:

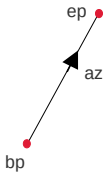
[ofs = 15][tie = 40][ofs is from Pt->100 to line <bp->bp, ep->pc1>][tie is from <Pt->bp> to ofs pt on line]

=g_lo(A809,A810,A811)

g_az() Get line azimuth

g_az(*bp, ep, opt_incl_descr*)

bp	[Ref] to bp
ep	[Ref] to ep
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info) returned string with info can't be used in math calculations



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns azimuth angle (see *Angle Types*) in degrees of direction of line [*bp-ep]*

Example:

Get azimuth from point *bp* to point *np* (points are from example included with function *p_az()*

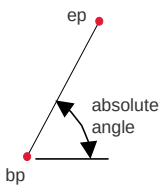
Copy previously obtained points:

[Pt->bp]«10.5,-12.5»		*=c_xy(A256:C256)*
[Pt->np]«-15.3078277067365,-35.6958192150957»		*=A262*
bp-np Az =	228.0511	*=g_az(A838,A839)*

Extended output option:

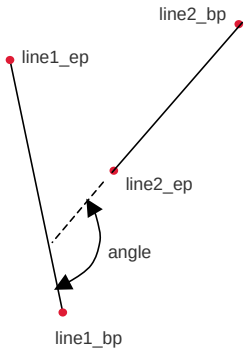
228.05111[Asimuth for line <bp->bp, ep->np>]	*=g_az(A838,A839,1)*

	A	B	C	D	E	F	G	H	I	J
848	**g_aang()**			Get absolute angle of line						
849	g_aang(*bp, ep, opt_incl_descr*)									
850										
851	*bp*		[Ref] to bp							
852	*ep*		[Ref] to ep							
853	*opt_incl_descr*		optional argument --> enter 1 for extended info (default -> no info)							
854			returned string with info can't be used in math calculations							
855										
856	[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates									
857										
858	Function returns counterclockwise angle in degrees from (+)X-direction to direction of line [bp-ep]									
859										
860	Example:									
861	Get absolute angles of lines [bp-np] and [np-bp] (see example included with function *p_aang()*									
862										
863	Copy previously obtained points:									
864	[Pt->bp]«10.5,-12.5»					*=c_xy(A286:C286)*				
865	[Pt->np]«-15.3078277067365,-35.6958192150957»					*=A289*				
866										
867	Get angles:									
868	bp-np abs ang =		221.9489			*=g_aang(A864,A865)*				
869	np-bp abs ang =		41.9489			*=g_aang(A865,A864)*				
870										
871	np-bp extended output option:									
872	41.94889[Abs angle for line <bp->np, ep->bp>]					*=g_aang(A865,A864,1)*				
873										
874										



g_rang() Get relative angle between two lines

g_rang(*line1_bp, line1_ep, line2_bp, line2_ep, opt_incl_descr*



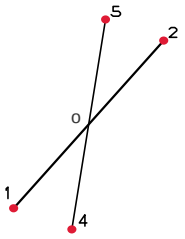
line1_bp	[Ref] to line1_bp
line1_ep	[Ref] to line1_ep
line2_bp	[Ref] to line2_bp
line2_ep	[Ref] to line2_ep
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)
	returned string with info can't be used in math calculations

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns angle between lines [*line1_bp - line1_ep*] and [l*ine2_bp - line2_ep*] in degrees
Returned angle is angle [line1_bp - line1&2_ep - line2_bp] formed by joining lines at their end points. 0 ≤ angle ≤ 180

Example:

Get relative angles between lines`from example included with function *p_rot()*



Copy previously obtained points:

[Pt->1]«39.3108,37.9696»		*=c_xy(A504:C504)*
[Pt->2]«69.0648,76.3618»		*=c_xy(A505:C505)*
[Pt->4]«50.9019900678989,33.1028897464135»		*=A514*
[Pt->5]«57.4736099321011,81.22	52.2243	*=A515*

Get angles:

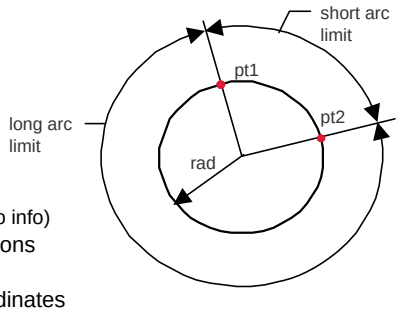
ang [1-o-4] =	30.0	*=g_rang(A895,A896,A897,A898)*
ang [1-o-5] =	150.0	*=g_rang(A895,A896,A898,A897)*

ang [1-o-5] extended output option:

150 [Angle between line1 <bp->1, ep->2> and line2 <bp->5, ep->4>] *=g_rang(A895,A896,A898,A897,1)*

g_arcl() Get arc length

g_arcl(*pt1, pt2, rad, opt_arc_type, opt_incl_descr*



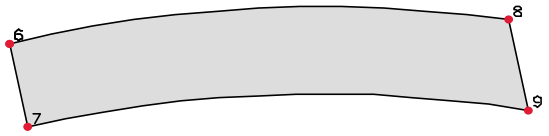
pt1	[Ref] to pt1
pt2	[Ref] to pt2
rad	arc radius
opt_arc_type	optional types: "s" or "l" (Default --> "s")
	"s" --> short arc (central angle ≤ 180)
	"l" --> long arc (central angle > 180)
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)
	returned string with info can't be used in math calculations

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns length of arc bounded by *pt1* and *pt2*

Example:

Get length along outside edges of bridge slab from example included with function *i_lc()*



Copy previously obtained points:

[Pt->6]«-503.596799411934,927.711312344167»	*=A653*
[Pt->7]«-482.193142593913,892.472230897094»	*=A654*
[Pt->8]«-96.0667242933865,1010.46209604281»	*=A655*
[Pt->9]«-72.9669913326633,972.430586841861»	*=A656*
[Pt->CCP]«100,-1000»	*=c_xy(A643:C643)*

Edge [6-8]

R =	2020.00	*=g_dst(A928,A932)*
L =	416.5845	*=g_arcl(A928,A930,C935)*

Edge [7-9]

R =	1980.00	*=g_dst(A929,A932)*
L =	417.7388	*=g_arcl(A929,A931,C939)*

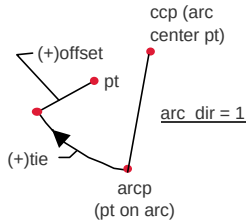
Edge [7-9] extended output option:

417.73882 [short arc <R = 1980> length bounded by Pt->7 and Pt->9]

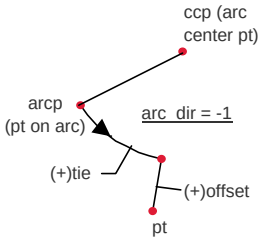
=g_arcl(A929,A931,C939,"s",1)

g_arco() Get point offset and tie to arc

g_arco(*pt, arcp, ccp, arc_dir, opt_output_code)*



pt	[Ref] to pt
arcp	[Ref] to arcp
ccp	[Ref] to ccp
arc_dir	1 or -1
	1 --> clockwise
	-1 --> counterclockwise
opt_output_code	optional codes: "o" or "t" or "e" (Default --> "e")
	"o" --> function returns *offset*
	"t" --> function returns *tie (arc* dist. from *arcp* to offset point on arc)
	"e" --> function returns text string with extended info (string can't be used in math calculations)



[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

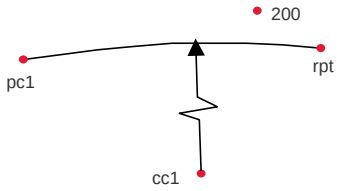
Function returns point *pt offset & tie* relative to arc defined by *arcp, ccp and arc_dir*
(+)*offset* --> on right side of arc looking from *arcp* in the direction specified by *arc_dir*
(+)*tie* --> runnung from *arcp* in the direction specified by *arc_dir*

Example:

Get offset and tie of point 200 relative to arc [pc1-rpt] (points are from example included with function *p_bso())*

Copy previously obtained points:

	E	N	
pc1	74.6152	445.4554	*=C419*
rpt	197.1418	451.0814	*=C421*
cc1	154.0089	53.4137	*=C420*
[Pt->200]«171.124586593891,468.060669651529»			*=A441*



pt 200 offset and tie relative pc1:

offset =	-15.00	*=g_arco(A976,A973:C973,A975:C975,1,"o")*
tie =	96.43	*=g_arco(A976,A973:C973,A975:C975,1,"t")*

pt 200 offset and tie relative rpt:

offset =	15.00	*=g_arco(A976,A974:C974,A975:C975,-1,"o")*
tie =	26.72	*=g_arco(A976,A974:C974,A975:C975,-1,"t")*

Extended output option:

[ofs = 15.00001][tie = 26.7152][ofs is from Pt->200 to arc <arcp->rpt, ccp->cc1>][tie is from <Pt->rpt> to ofs pt on arc]

=g_arco(A976,A974:C974,A975:C975,-1)

g_bso() Get point station and offset relative to baseline

g_bso(*pt, gblock_rng, ref_sta, opt_output_code)*

pt	[Ref] to pt
gblock_rng	cell range reference to gblock defining baseline (see *Special Objects*)
ref_sta	station of first point in the baseline definition
opt_output_code	optional codes: "s" or "o" or "e" (Default --> "e")
	"s" --> function returns *station*
	"o" --> function returns *offset*
	"e" --> function returns text string with extended info (string can't be used in math calculations)

[Ref] --> Cell reference to cell range or X/Y string, whichever holds point tag and coordinates

Function returns point *pt* station/offset relative to baseline defined in *gblock_rng*
(+)*offset* --> on right side of baseline looking station ahead

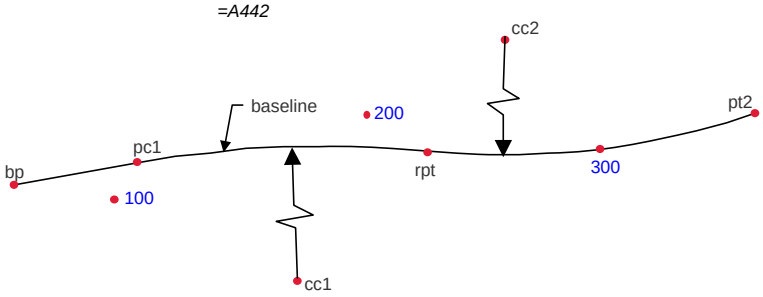
Example:
Get stations and offsets of points from example included with function *p_bso())*

Copy previously obtained points:

[Pt->100]«64.2887323443607,428.059643141561»	*=A440*
[Pt->200]«171.124586593891,468.060669651529»	*=A441*
[Pt->300]«270.237251696264,452.112035301658»	*=A442*

Copy Gblock defining baseline:

<gblock 1>	*=A428*
[Pt->bp]«22.1073,434.8218»	*=A429*
[Pt->pc1]«74.6152,445.4554»	*=A430*
<arc>	*=A431*
[Pt->cc1]«154.0089,53.4137»	*=A432*
[Pt->rpt]«197.1418,451.0814»	*=A433*
<arc>(-clock)	*=A434*
[Pt->cc2]«229.4915,749.3321»	*=A435*
[Pt->pt2]«335.1841,468.567»	*=A436*
<end gb>	*=A437*



pt bp sta = 10+00.00 *=D418*

Stations and offsets:

Pt	sta		offset	
100	10+40.00	*=g_bso(A1012,$A$1017:$A$1026,$B$1028,"s")*	15.00	*=g_bso(A1012,$A$1017:$A$1026,$B$1028,"o")*
200	11+50.00	*=g_bso(A1013,$A$1017:$A$1026,$B$1028,"s")*	-15.00	*=g_bso(A1013,$A$1017:$A$1026,$B$1028,"o")*
300	12+50.00	*=g_bso(A1014,$A$1017:$A$1026,$B$1028,"s")*	0.00	*=g_bso(A1014,$A$1017:$A$1026,$B$1028,"o")*
bp	10+00.00	*=g_bso(A1018,$A$1017:$A$1026,$B$1028,"s")*	0.00	*=g_bso(A1018,$A$1017:$A$1026,$B$1028,"o")*
pc1	10+53.57	*=g_bso(A1019,$A$1017:$A$1026,$B$1028,"s")*	0.00	*=g_bso(A1019,$A$1017:$A$1026,$B$1028,"o")*
cc1	10+53.57	*=g_bso(A1021,$A$1017:$A$1026,$B$1028,"s")*	400.00	*=g_bso(A1021,$A$1017:$A$1026,$B$1028,"o")*
rpt	11+76.72	*=g_bso(A1022,$A$1017:$A$1026,$B$1028,"s")*	0.00	*=g_bso(A1022,$A$1017:$A$1026,$B$1028,"o")*
cc2	11+76.72	*=g_bso(A1024,$A$1017:$A$1026,$B$1028,"s")*	-300.00	*=g_bso(A1024,$A$1017:$A$1026,$B$1028,"o")*
pt2	13+17.14	*=g_bso(A1025,$A$1017:$A$1026,$B$1028,"s")*	0.00	*=g_bso(A1025,$A$1017:$A$1026,$B$1028,"o")*

Point 300 extended output option:

[ofs = 0][sta = 1250][Pt->300, BL defined in <gblock 1> (ref sta = 1000)] *=g_bso(A1014,$A$1017:$A$1026,$B$1028)*

g_ba() Get area of shape bounded by lines/arcs

g_ba(*gblock_rng, opt_incl_descr)*

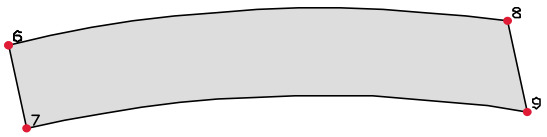
gblock_rng cell range ref to gblock defining closed shape (see *Special Objects*)
Gblock restrictions:
1) points must be specified in a clockwise order
2) shape sides can't cross over each other

opt_incl_descr optional argument --> enter 1 for extended info (default -> no info)
returned string with info can't be used in math calculations

Function returns area of shape defined in *gblock_rng*

Note:
If first and last points aren't the same, they are connected with straight line to form closed figure.

Example:
Get area of bridge slab from example included with function *i_lc()*



Copy previously obtained points into Gblock defining slab:

<gblock 1>	
[Pt->6]«-503.596799411934,927.711312344167»	*=A653*
<arc>	
[Pt->CCP]«100,-1000»	*=c_xy(A643:C643)*
[Pt->8]«-96.0667242933865,1010.46209604281»	*=A655*
[Pt->9]«-72.9669913326633,972.430586841861»	*=A656*
<arc>(-clock)	
[Pt->CCP]«100,-1000»	*=A1069*
[Pt->7]«-482.193142593913,892.472230897094»	*=A654*
[Pt->6]«-503.596799411934,927.711312344167»	*=A1067*
<end gb>	

Calculate area of slab:

area = 16686.19975 *=g_ba(A1066:A1076)*

Extended output option:

16686.19975 [area of shape defined in <gblock 1>] *=g_ba(A1066:A1076,1)*

g_bl() Get cumulative length of consecutive lines/arcs

g_bl(*gblock_rng, opt_bp_tag, opt_ep_tag, opt_incl_descr)*

gblock_rng	cell range ref to gblock defining figure (see *Special Objects*)
opt_bp_tag	optional tag of beginning point of length measurement
	Default --> tag of first point in gblock
opt_ep_tag	optional tag of end point of length measurement
	end point must be in lower position within gblock than beginning point
	Default --> tag of last point in gblock
opt_incl_descr	optional argument --> enter 1 for extended info (default -> no info)
	returned string with info can't be used in math calculations

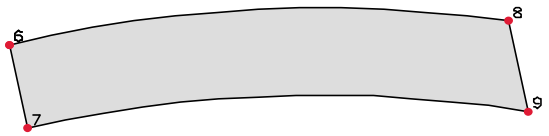
Function returns cumulative length of consecutive lines/arcs between points *opt_bp_tag* and *opt_ep_tag* .
In closed figures, length accumulation follows point placement order inside gblock.

Example:

Get length of edges and sides of bridge slab from example included with function *i_lc()*

Copy gblock defining slab:

<gblock 1>	
[Pt->6]«-503.596799411934,927.711312344167»	*=A1067*
<arc>	*=A1068*
[Pt->CCP]«100,-1000»	*=A1069*
[Pt->8]«-96.0667242933865,1010.46209604281»	*=A1070*
[Pt->9]«-72.9669913326633,972.430586841861»	*=A1071*
<arc>(-clock)	*=A1072*
[Pt->CCP]«100,-1000»	*=A1073*
[Pt->7]«-482.193142593913,892.472230897094»	*=A1074*
[Pt->6]«-503.596799411934,927.711312344167»	*=A1075*
<end gb>	*=A1076*



Calculate length:

[6-8] L =	416.58	*=g_bl(A1104:A1114,,8)*
[7-9] L =	417.74	*=g_bl(A1104:A1114,9,7)*
[6-7] L =	41.23	*=g_bl(A1104:A1114,7)*
[8-9] L =	44.50	*=g_bl(A1104:A1114,8,9)*

[8-9] extended output option:

44.49712 [length from Pt->8 to Pt->9 along edge of figure defined in <gblock 1>]

=g_bl(A1104:A1114,8,9,1)

	A	B	C	D	E	F	G	H	I	J
1127	**c_aang_az()**				Convert absolute angle to azimuth (See Angle Types)					
1128	c_aang_az(*aang*)									
1129										
1130	*aang*		absolute angle in degrees							
1131										
1132	Function returns angle converted from absolute angle *aang* to azimuth in degrees									
1133										
1134	Example:									
1135	abs ang =		-30.5 deg							
1136	az =		120.5 deg			*=c_aang_az(C1135)*				
1137										
1138										

	A	B	C	D	E	F	G	H	I	J
1139	**c_aang_brg()**				Convert absolute angle to bearing (See Angle Types)					
1140	c_aang_brg(*aang, opt_sec_rnd*)									
1141										
1142	*aang*		absolute angle in degrees							
1143	*opt_sec_rnd*		optional number of digits after seconds (") decimal point in returned angle							
1144			Default --> 0 (no digits)							
1145										
1146	Function returns angle converted from absolute angle *aang to* bearing									
1147	Returned angle is in text format and can't be used in math calculations									
1148										
1149	*Example:*									
1150	abs ang =		-30.5 deg							
1151	brg =		S59°-30'-0.00"E			*=c_aang_brg(C1150,2)*				
1152										
1153										

	A	B	C	D	E	F	G	H	I	J
1154	**c_az_aang()**			Convert azimuth to absolute angle (See Angle Types)						
1155	c_az_aang(*az*)									
1156										
1157	*az*		azimuth in degrees							
1158										
1159	Function returns angle converted from azimuth *az* to absolute angle in degrees									
1160										
1161	Example:									
1162	az =		120.5 deg							
1163	abs ang =		329.5 deg			*=c_az_aang(C1162)*				
1164										
1165										

	A	B	C	D	E	F	G	H	I	J
1166	**c_az_brg()**				Convert azimuth to bearing (See Angle Types)					
1167	c_az_brg(*az, opt_sec_rnd*)									
1168										
1169	*az*		azimuth in degrees							
1170	opt_sec_rnd		optional number of digits after seconds (") decimal point in returned angle							
1171			Default --> 0 (no digits)							
1172										
1173	Function returns angle converted from azimuth *az* to bearing									
1174	Returned angle is in text format and can't be used in math calculations									
1175										
1176	Example:									
1177	az =		120.5 deg							
1178	brg =		S59°-30'-0.00"E							
1179										
1180										

	A	B	C	D	E	F	G	H	I	J
1181	**c_brg_aang()**				Convert bearing to absolute angle (See Angle Types)					
1182	c_brg_aang(*brg*)									
1183										
1184	*brg*		bearing (must be bearing returned by other angle functions)							
1185										
1186	Function returns angle converted from bearing *brg* to absolute angle in degrees									
1187										
1188	Example:									
1189	brg =		S59°-30'-0.00"E			*=C1178*				
1190	abs ang =		329.5 deg			*=c_brg_aang(C1189)*				
1191										
1192										

	A	B	C	D	E	F	G	H	I	J
1193	**c_brg_az()**				Convert bearing to azimuth (See Angle Types)					
1194	c_brg_az(*brg*)									
1195										
1196	*brg*		bearing (must be bearing returned by other angle functions)							
1197										
1198	Function returns angle converted from bearing *brg* to azimuth in degrees									
1199										
1200	Example:									
1201	brg =		S59°-30'-0.00"E			*=C1189*				
1202	az =		120.5 deg			*=c_brg_az(C1201)*				
1203										
1204										

	A	B	C	D	E	F	G	H	I	J
1205	**dms()**				Convert angle in degrees to d-m-s format					
1206	dms(*angle, opt_sec_rnd*)									
1207										
1208	*angle*		angle in degrees							
1209	*opt_sec_rnd*		optional number of digits after seconds (") decimal point in returned angle							
1210			Default --> 0 (no digits)							
1211										
1212	Function returns angle converted from degrees to d-m-s format									
1213										
1214	*Example:*									
1215	ang =		120.5 deg	=	120°-30'-0.0"	*=dms(C1215,1)*				
1216										
1217										

	A	B	C	D	E	F	G	H	I	J
1218	**deg()**				Convert d-m-s angle to degrees					
1219	deg(*angle*)									
1220										
1221	*angle*		angle in d-m-s format (must be angle returned by *dms()* or *ang_()* functions)							
1222										
1223	Function returns angle converted from d-m-s format to degrees									
1224										
1225	Example:									
1226	ang =		120°-30'-0.0"	=	120.5 deg					
1227			*=E1215*		*=deg(C1226)*					
1228										
1229										

	A	B	C	D	E	F	G	H	I	J
1230	**ang_()**				Enter angle in d-m-s format					
1231	ang_(*dg, opt_min, opt_sec*)									
1232										
1233	*dg*		degree portion of angle							
1234	*opt_min*		optional minutes portion of angle (is req'd if *opt_sec* isn't omitted)							
1235	*opt_sec*		optional seconds portion of angle							
1236										
1237	Function returns angle in d-m-s format									
1238										
1239	Example:									
1240	ang =		34°-30'-20"		*=ang_(34,30,20)*					
1241										
1242										

	A	B	C	D	E	F	G	H	I	J
1243	**bang_()**				Enter angle in bearing notation					
1244	bang_(*dir, dg, opt_min, opt_sec*)									
1245										
1246	*dir*		direction codes --> "*ne*" or "*nw*" or "*se*" or "*sw*"							
1247	*dg*		degree portion of angle							
1248	*opt_min*		optional minutes portion of angle (is req'd if *opt_sec* isn't omitted)							
1249	*opt_sec*		optional seconds portion of angle							
1250										
1251	Function returns angle in bearing format									
1252										
1253	Example:									
1254	bearing =		N34°-30'-20"W		*=bang_("nw",34,30,20)*					
1255										
1256										

	A	B	C	D	E	F	G	H	I	J
1257	**c_xy()**				Convert coordinates into X/Y text string					
1258	c_xy(*pt*)									
1259										
1260	*pt*		cell range reference to cells holding *pt* tag and coordinates							
1261										
1262	Function returns X/Y text string holding embedded *pt* tag and coordinates									
1263										
1264	Example:									
1265	Pt	X	Y							
1266	wp3	10.25	-56.7							
1267										
1268	Convert wp3 coordinates into X/Y text string:									
1269	[Pt->wp3]«10.25,-56.7»				*=c_xy(A1266:C1266)*					
1270										
1271										

	A	B	C	D	E	F	G	H	I	J
1272	**g_tag()**	g_tag(*pt*)		Get point tag from X/Y text string						
1273	**g_x()**	g_x(*pt*)		Get point X-crd from X/Y text string						
1274	**g_y()**	g_y(*pt*)		Get point Y-crd from X/Y text string						
1275										
1276	*pt*		cell reference to X/Y string holding *pt* tag and coordinates							
1277										
1278	*g_tag* returns *pt* tag									
1279	*g_x* returns *pt* X-coordinate									
1280	*g_y* returns *pt* Y-coordinate									
1281										
1282	Example:									
1283	Tabulate coordinates of new points from example included with function *p_xytr*									
1284										
1285	Copy previously obtained points:									
1286	[Pt->2]«-38.6602540378444,1.11022302462516E-15»					*=A563*				
1287	[Pt->3]«-46.1602540378444,-12.9903810567666»					*=A564*				
1288	[Pt->4]«-21.3397459621556,-10»					*=A565*				
1289										
1290	Coordinates:									
1291	Pt			N		E				
1292	2	*=g_tag(A1286)*		0.0000	*=g_y(A1286)*	-38.6603	*=g_x(A1286)*			
1293	3	*=g_tag(A1287)*		-12.9904	*=g_y(A1287)*	-46.1603	*=g_x(A1287)*			
1294	4	*=g_tag(A1288)*		-10.0000	*=g_y(A1288)*	-21.3397	*=g_x(A1288)*			
1295										
1296										

Gblocks

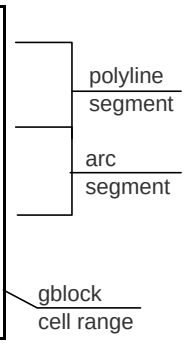
Overview

Gblock defines geometric figures consisting of polyline and arc segments. Gblocks are used in some COGO functions, viewports and DXF-blocks.

Gblock consists of vertically stacked cells holding keywords and formulas that result in a X/Y text string (e.g. [Pt->wp2]«200,50»). Formula types that produce X/Y text strings are as follows:

- COGO function that create new point
- Function c_xy() (see *X/Y Strings* functions)
- Cell reference to another cell containing X/Y text string (e.g. =c45).
- Conditional formula resulting in X/Y text string

Gblock Definition



- Gblock starts with keyword <gblock id> (id - positive integer number, unique for each gblock) and ends with keyword <end gb>. P-points are placed between keywords.
- P-point ----> X/Y text string representing point.
- Polyline segment consists of consecutively connected P-points.
- Arc segment is defined in four consecutive cells as follow:
  1. arc_bp --> beginning arc point
  2. Keyword: <arc> or <arc>(+clock) or <arc>(-clock)
     +clock --> clockwise arc
     -clock --> counterclockwise arc
     if "clock" is omitted then clockwise arc is used
  3. CC --> center of arc point
  4. arc_ep --> end arc point
- There is no restrictions on a number and position (within gblock) of polyline and arc segments.

Gblock Keyword Functions

In addition to typing, keywords may be entered using following functions:

gb_start(id)	returns <gblock id>	*id* --> positive integer number, unique for each gblock
gb_end()	returns <end gb>	function has no arguments (parenthesis are required)
gb_arc(dir)	returns <arc>(+clock)	if *dir* = 1
	or <arc>(-clock)	if *dir* = -1

Gblock Validation

The functions that take gblock as an argument return an error if any of the following is found:

1. Misspelled or missing gblock keywords
2. Gblock entrees other than X/Y strings and gblock keywords
3. Empty cells inside gblock (doesn't apply to viewports and DXF-blocks)
4. Incorrect gblock cell range
5. Beginning (arc_bp) and end (arc_ep) arc point radiuses (rounded to 4th decimal point) are not equal
6. Non-compliance with function-specific restrictions if any (see individual functions)

Notes for Gblocks used in Viewports and DXF-blocks

- Use blank cells to separate individual figures and points. Stand-alone points should have blank cells immediately above and bellow them. There is no limit on number of blank cells separating adjacent figures and points.
- Arc points are treated as non-arc points if there are empty cells inside arc segment definition.
- Same points may be used multiple times inside gblocks.
- Closed figures must have the same first and last points.
- The figures are displayed on top of other figures and points that precede them in gblock definition order; 1st figure in the gblock is displayed on the very bottom and last figure is displayed on the very top.
- Closed figures are displayed filled or unfilled depending on Excel autoshape default setting. Filled figures will cover other figures and points that are under. Change Excel autoshape default setting or break closed figure if fill is undesirable. For example, closed figure 1-2-3-1 may be broken in two opened figures: 1-2-[blank cell]-2-3-1.

Viewports

For Viewport examples, see *Example 1* and *Example 2*

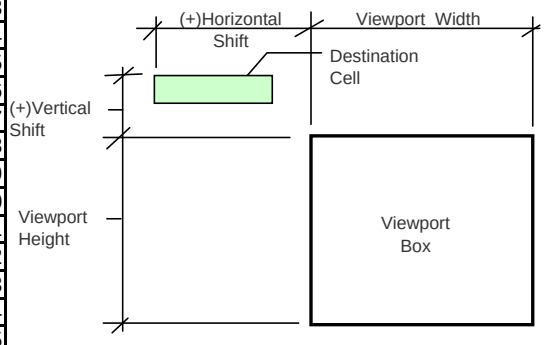
Overview

Viewport object provides a visual feedback of geometry created with COGO functions. Graphics are displayed in the designated area on the spreadsheet. Display is dynamically updated reflecting spreadsheet changes. The graphics can be displayed in different ways, depending on specified scale, rotation and other viewport settings. Viewport supports layers of graphics that can be turned on and off.

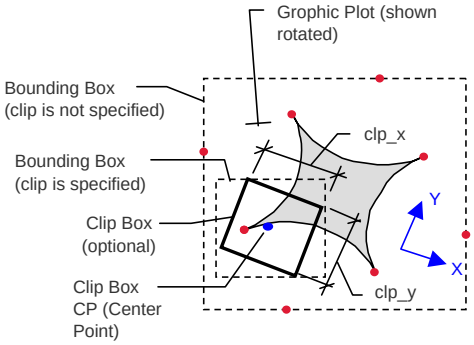
Viewports: Introductory Guide (included in spreadsheet zip folder) describes purpose and creation timing of viewport components and related spreadsheet elements. It also provides guidelines on working with viewports.

Viewports are set up in the *Viewport Parameter Blocks* and activated by function v*_plot()* .

Viewport Schematics



Graphics Placement Components
(Units of measurement: Points)



Graphics Boundary Components
(Units of measurement: graphic units)

Notes:

1. The Bounding Box is a boundary that contains graphics. It passes through the most extreme outer points of the graphic plot.
2. The Bounding Box bounds graphics inside Clip Box if Clip Box data is specified.
3. The graphic plot is positioned inside viewport by superimposing top left corner of the Bounding Box over top left corner of the Viewport Box.
4. When Viewport and/or Clip Box data is specified, arc segment CC-points (see Gblocks) are excluded from the points defining Bounding Box and are not plotted if they are outside Bounding Box. This doesn't apply to CC-points specified outside arc segments in the gblocks.
5. Clip Box is aligned with X/Y axes. It's size is not limited by overall graphic extends.
6. Points are default units that Excel uses for screen measurements (according to Microsoft documentation, 1 point = 1/72 inches = 0.3528 mm in Excel). Graphic units are processed as if they were Points. For plotting in graphic units, refer to *Plotting to Scale* .

Viewport Parameter Block

parameter 1
parameter 2
parameter 3
…
parameter n

Viewport Parameter Block consists of vertically stacked cells holding viewport parameters listed in the table below

block cell range

Vewport Parameter Table

Parameter	Description	Required/ Optional	Default	Comment
1	Keyword: <viewport id>	req'd	n/a	id - positive integer number, unique for ea. viewport. See optional function *v_start()*
2	Source Gblocks	req'd	n/a	Use function *v_gblk_rng()* (single Gblock) or *v_gblist_rng()* (multiple Gblocks)
3	Destination Cell	req'd	n/a	Cell address (e.g. f56) See optional function *v_cell_adr()*
4	Horizontal Shift (in points)	optional	0	(+) shift --> to right from top/lt corner of cell
5	Vertical Shift (in points)		0	(+) shift --> down from top/lt corner of cell
6	Viewport Box Width (in points)		see comment	Both width and height use default setting if either one is left blank. Box size is specified Plot is scaled to produce largest Bounding Box that fits inside VP Box. Relative scale is retained If par's 8 & 9 are set. Default Setting Viewport Box size is set to size of Bounding Box.
7	Viewport Box Height (in points)		see comment	
8	X_scale		1.0	Only relative scale is used if Viewport Box dimensions (par's 6 & 7) are specified.
9	Y_scale		1.0	
10	Rotation (degrees)		0	(+) rotation --> clockwise
11	Clip Box --> CP (X/Y text string)		no clip	Clip Box isn't used if any one parameter is left blank. Dimensions are in graphic units.
12	Clip Box --> Dimension *clp_x*		no clip	
13	Clip Box --> Dimension *clp_y*		no clip	
14	Include Point Symbology (y/n)		y	y --> yes n --> no Regardless, border is not displayed If Viewport Box dimensions (par's 6 & 7) aren't specified.
15	Display Viewport Border (y/n)		n	
16	Suspend Plot Update (y/n)		n	
17	Hide Viewport (y/n)		n	
18	Direction Symbol (n/x)	optional	no symbol	n --> North arrow x --> X/Y axis

Notes:

1. Function *v_gblk_rng()* or *v_gblist_rng()* must be used for parameter 2. Other parameters may be literal values or optional functions (where noted) or Excel formulas resulting in a parameter-appropriate values.
2. When encountered, erroneous values are replaced with default values, whenever possible.

Viewport Functions

v_start(id)
Returns viewport parameter 1

id --> positive integer number, unique for each viewport

v_gblk_rng(gblk_rng, opt_on_off)
Returns address and id of source gblock.
This function is used for parameter 2 (single gblock) or inside gblist block (multiple (gblocks).

gblk_rng --> cell range reference to source gblock
opt_on_off --> optional argument used only when function is placed inside gblist block
1 --> display is on (default setting)
0 --> display is off

v_gblist_rng(gblist_rng)
Returns address and id of source gblist block.
This function is used for parameter 2 (multiple gblocks).

gblist_rng --> cell range reference to source gblist block

v_cell_adr(cell_adr)
Returns viewport parameter 3

cell_adr --> cell reference to viewport destination cell

v_plot(vport_rng)
Returns text string with viewport information. This function is required to activate viewport.

vport_adr --> cell range reference to Vewport Parameter Block
Bottom cell in the range should be last cell holding either last non-default parameter or last default parameter that may be altered in the future, whichever placement is closer to the bottom of block. At minimum, parameters 1, 2 & 3 must be included.

v_gblist_start(id)
Returns keyword <gblist id> used in gblist block

id --> positive integer number, unique for each gblist block

v_gblist_end()
Returns keyword *<end list>* used in gblist block

function has no arguments (parenthesis are required)

Gblist Block

Gblist block lists multiple gblocks (layers), subject for display in the viewport. Viewport function *v_gblist_rng()* takes gblist block address as an argument.

Gblist block consists of vertically stacked *gblk_rng()* functions placed between keywords *<gblist id>* and *<end list>,* where id --> positive integer number, unique for each gblist block.
Keywords may be typed or entered using *v_gblist_start()* and *v_gblist_end()* functions.
Function *gblk_rng()* optional argument controls on/off status of the gblock display.

Gblocks are placed on top of other gblocks that precede them in gblist order; 1st gblock is displayed on the very bottom and last gblock is displayed on the very top.

Example:

<gblist 13>	*=v_gblist_start(13)*
[$A$1864:$A$1878] <gblock 4>	*=v_gblk_rng(A1864:A1878)*
[$A$1848:$A$1859] <gblock 3> (off)	*=v_gblk_rng(A1848:A1859,0)*
<end list>	*=v_gblist_end()*

Plotting to Scale

In order to plot to absolute scale, Viewport Box size parameters 6 & 7 should be left blank. Use following examples as a guide to set desired scale (parameters 8 & 9):

Graphic Units	Plot Scale	Par's 8 & 9	
points	1:1	1	
inches	1:1	72	*=1/(1/72)*
feet	1" = 1'-0"	72	*=1/(1/72)/12*12*
feet	1/8" = 1'-0"	9	*=1/(1/72)/12*12*(1/8)*
feet	1" = 10'	7.2	*=1/(1/72)/12*12/10*
mm	1:1	2.8345	*=1/0.3528*
m	1:1000 (1mm = 1m)	2.8345	*=1/0.3528/1000*1000*

Above examples are based on equality: 1 point = 1/72 inches. In reality, actual Excel screen and print plots deviate from the stated equality. In addition, V/H aspect ratio of the printed plots is distorted. The degree of inaccuracy is device (monitor/printer) depended.
To account for inaccuracy and distortion, parameter 8 & 9 values above should be calibrated for specific plotting tasks.

Copying Graphics

Conventional Copy and Paste won't work with viewport graphics (see *Graphic Naming Effects*) unless copy is placed on another worksheet. To make a viewport copy on the same worksheet, use any one of the following methods:

Method 1
Select cell holding *v_plot(*) function referencing target viewport --> Press **CTRL**+**F4** (default key combination assigned to *Copy_Viewport* macro)

Method 2 (Excel 2000 and 2003 only)
Select viewport graphic group --> Change it's name in the Excel **Name Box** --> Press **Enter**
Renamed group may be ungrouped for editing but should be regrouped following editing.
Name Box it's where selected cell addresses/range names are displayed. It's located near top/left corner of the Excel window.

Method 3 (Excel 2000 and 2003 only)
Select viewport graphics --> Ungroup selection --> Edit graphics if req'd --> Regroup graphics

Method 4
Select viewport graphics --> Ungroup selection --> Rename each shape included in the group --> Edit graphics as required --> Regroup graphics

Zooming and Panning

Use Clip Box to zoom and pan as follows:

To zoom in --> change parameters 12 and 13 (clp_x and clp_y) to smaller values
To zoom out --> change parameters 12 and 13 (clp_x and clp_y) to larger values
To pan view --> change parameter 11 (clip center point coordinates)

Graphic Naming Effects

Viewport and associated gblock and layer list id-numbers are part of the shape names that spreadsheet assigns to viewport graphic group and group shape components. Spreadsheet uses these names to identify and delete older graphics each time viewport update event takes place. The way in which spreadsheet processes graphic elements effects end results of some User actions. Impacted actions, end results and follow-up comments are as follows:

- Changing viewport or associated gblock/layer list Id -number --> updated plot is placed on top of older graphics that was not deleted.
  Relocate or delete residual graphics to clear viewport.
- Copy and Paste viewport graphics --> copy is lost, following viewport update event.
  Use one of the copying methods described under Copying Graphics.
- Modifying (relocate, ungroup, edit, etc.) viewport graphics --> modification is lost, following viewport update event.
  Only copy of viewport graphics can be modified.
- Changing name of the viewport graphic group in Excel 2007 or later version --> causes Excel error (due to duplicate group shape component names) resulting in incomplete, partially or completely filled plots, following viewport update event.
  Delete renamed graphic group to eliminate cause of the problem.

Troubleshooting

Problem	Probable Cause	Resolution
There is no visible graphics from the outset Plot has abruptly disappeared Incomplete plot	1. Parameter 17 (Hide Viewport) is set to "n" 2. Duplicate gblock and viewport id-numbers 3. Errors due to erroneous/missing input data 4. Deletion of worksheet rows effecting vport and gblock ranges	Identify and address the cause of the problem. Note: Manual viewport update (see note 2) may resolve problems associated with cause 4.
Incomplete, partially or completely filled plot	Viewport graphic group name has been changed (see *Graphic Naming Effects*)	Delete renamed graphic group
Graphic modification is lost following viewport graphics update event	Spreadsheet graphic naming algorithm (see *Graphic Naming Effects*)	Modify copy of viewport (see Copying Graphics)
Updated plot is placed on top of older graphics	Id -number of viewport or source gblock has been changed (see *Graphic Naming Effects*)	Relocate or delete older graphics
Viewport does't reflect spreadsheet changes	Parameter 16 (Suspend Plot Update) is set to "y"	Set parameter 16 to "n"
Point markers and labels are not visible	Parameter 14 (Include Point Symbology) is set to "n"	Set parameter 14 to "y"
Point markers are on but there are no labels	Excel internal event handling	Manually update viewport graphics (see note 2)

Notes:

1. Manual viewport update (see note 2) may be required as a last step in problem resolution or if viewport graphics were incidentally deleted, moved or distorted.
2. To manually update viewport graphics, reenter *v_plot()* function referencing target viewport. To reenter function, place cursor on the formula in the Excel *Formula Bar* and press **ENTER.** In certain cases, reentering *v_plot()* function may be required more then once.
3. Viewports are not compatible with worksheet protection.

DXF-Blocks

DXF-blocks are used to specify content and properties of AutoCAD DXF file. To create AutoCAD DXF file, do the following:

1. Setup DXF-block
2. Select keyword <dfx> and press **CTRL**+**F5** (default key combination assigned to *create_dxf* macro).

DXF-block consists of vertically stacked par-blocks placed between keywords <dxf> and <end>. Each par-block consists of optional parameters and cell range reference to source gblock.

DXF-block

```
<dxf>
par_block 1
par_block 2
…….
par_block n
<end>
```

par-block

```
opt_par1
opt_par2
opt_par3
opt_par4
opt_par5
opt_par6
gblock_ref
```

- Each optional parameter occupies single cell and consists of keyword and value (see table below).
- If optional parameter is not included in the par-block 1 then default value is used. If optional parameter is not included in another par-block then value of corresponding parameter (specified or default) in the preceding par-block is used.

Optional Parameter Table

Parameter	Keyword	Value	DefaultValue	Comment
opt_par1	<layer>	Layer name	layer999	
0pt_par2	<color>	DXF color number (1-255)	3	example of colors: 1 - red 2 - yellow 3 - green 5 - blue 7 - white 255 - black
opt_par3	<ln_style>	Line style number (1-4) 1 - continuous 2 - dashed 3 - center 4 - phantom	1	
opt_par4	<plot_opt>	Plot option (1-3) 1 - plot all 2 - exclude points from plot 3 - plot points only	1	
opt_par5	<pt_size>	Diameter of circle representing point	0.50	ignored if opt_par4 = 2
opt_par6	<txt_size>	Text character width and height	1.00	ignored if opt_par4 = 2

gblock_ref

Use Viewport function *v_gblk_rng()* (required) to enter gblock_ref. Refer to *Viewport Functions* for function reference.

Example:

Setup DXF-block for slab from example included with function *g_bl().* Put slab outline and points on different layers.

```
<dxf>
<layer> slab
<plot_opt> 2
[$A$1085:$A$1095] <gblock 1>          =v_gblk_rng(A1085:A1095)
<layer> points
<color> 1
<plot_opt> 3
[$A$1085:$A$1095] <gblock 1>          =A1643
<End>
```

Note:

CADD Zooming to drawing extends may be required to see opened DXF-file contents.

Example 1: Stakeout

Objectives:

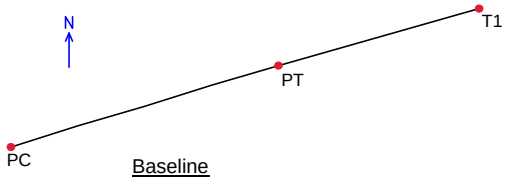
1. Obtain and tabulate coordinates of abutment working points (control points @ front face of abutment wall)
2. Get control stations and offsets
3. Setup stakeout DXF-block

Baseline Data:

	E	N	Sta		
PC	1132044.6831	225848.6924	345+23.00		
CC	1136916.3574	211661.8397	-	R=15000.0000	(Rt curve)
PT	1132469.5517	225987.5480	349+70.00		
T1	1132787.5824	226086.2671			

Define baseline:

<gblock 1>	*=gb_start(1)*
[Pt->PC]«1132044.6831,225848.6924»	*=c_xy(A1662:C1662)*
<arc>(+clock)	*=gb_arc(1)*
[Pt->CC]«1136916.3574,211661.8397»	*=c_xy(A1663:C1663)*
[Pt->PT]«1132469.5517,225987.548»	*=c_xy(A1664:C1664)*
[Pt->T1]«1132787.5824,226086.2671»	*=c_xy(A1665:C1665)*
<end gb>	*=gb_end()*



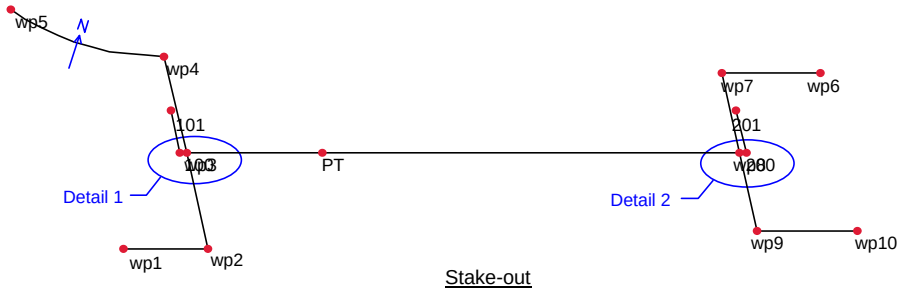
Baseline

<VP 1> [Par's 1-18 @ $C$1795] *=v_plot(C1795:C1812)*

Note:
Bridge and wing walls (except as noted) are aligned with tangent portion of baseline (PT-T1).

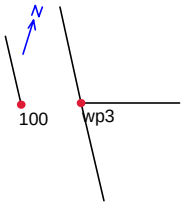
Baseline tangent absolute angle and bearing:

abs ang =	17.2447	*=g_aang(Pt-»PT,Pt-»T1)*
brg =	N72°-45'-19"E	*=c_aang_brg(17.2447323548023)*



Stake-out

<VP 2> [Par's 1-18 @ $D$1795]



Detail 1

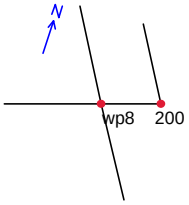
<VP 3> [Par's 1-18 @ $E$1795]

Locate Abutment 1 points:

Given Data:

	Pt	Sta	dist to FF	[wp4-wp3-PT] ang
CL brg	100	349+40.00	1.25	102.0000

	Length			
wp1/wp2	18.0000			
wp2/wp3	22.5000			
wp3/wp4	22.5000			
wp4/wp5	35.0000	Rad =	55.00	curved wing



Detail 2

<VP 4> [Par's 1-18 @ $F$1795]

	A	B	C	D	E	F	G	H	I	J
1707	[Pt->100]«1132440.90919751,225978.62577103»				*=p_bso(NP->100,34940,0,<gblock1>,34523)*					
1708	[Pt->101]«1132436.02378727,225987.351180291»				*=p_aang(NP->101,Pt-»100,17.2447323548023+102,10)*				*CL brg working pt*	
1709	[Pt->wp3]«1132442.11470727,225979.031352931»				*=i_ll(NP->"wp3",Pt-»PT,Pt-»T1,Pt-»100,Pt-»101,0,1.25)*					
1710	[Pt->wp2]«1132453.10688031,225959.399182093»				*=p_aang(NP->"wp2",Pt-»wp3,17.2447323548023+102+180,22.5)*					
1711	[Pt->wp1]«1132435.91603064,225954.063014209»				*=p_aang(NP->"wp1",Pt-»wp2,17.2447323548023+180,18)*					
1712	[Pt->wp4]«1132431.12253423,225998.663523769»				*=p_aang(NP->"wp4",Pt-»wp3,17.244732354 wing CC*					
1713	[Pt->102]«1132414.81757681,226051.191119977»				*=p_aang(NP->102,Pt-»wp4,17.2447323548023+90,55)*					
1714	[Pt->wp5]«1132396.71520953,225999.255541624»				*=p_arcl(NP->"wp5",Pt-»wp4,Pt-»102,35)*					
1715										
1716	*Locate Abutment 2 points:*									
1717	*Given Data:*									
1718		Pt	Sta	dist to FF	[PT-wp8-wp9] ang					
1719	CL brg	200	350+60.00	1.25	102.0000					
1720										
1721		Length								
1722	wp6/wp7	21.0000								
1723	wp7/wp8	18.5000								
1724	wp8/wp9	18.5000								
1725	wp9/wp10	21.0000								
1726										
1727	[Pt->200]«1132555.50595794,226014.228842401»				*=p_bso(NP->200,35060,0,<gblock1>,34523)*					
1728	[Pt->201]«1132550.6205477,226022.954251662»				*=p_rang(NP->201,Pt-»wp3,Pt-»200,180-102,10)*				*CL brg working pt*	
1729	[Pt->wp8]«1132554.28547853,226013.849996494»				*=i_ll(NP->"wp8",Pt-»PT,Pt-»T1,Pt-»200,Pt-»201,0,-1.25)*					
1730	[Pt->wp7]«1132545.24746958,226029.992003627»				*=p_rang(NP->"wp7",Pt-»wp3,Pt-»wp8,180-102,18.5)*					
1731	[Pt->wp6]«1132565.30346086,226036.217532825»				*=p_aang(NP->"wp6",Pt-»wp7,17.2447323548023,21)*					
1732	[Pt->wp9]«1132563.32348748,225997.707989361»				*=p_rang(NP->"wp9",Pt-»wp3,Pt-»wp8,-102,18.5)*					
1733	[Pt->wp10]«1132583.37947876,226003.933518559»				*=p_aang(NP->"wp10",Pt-»wp9,17.2447323548023,21)*					
1734										
1735	*Get wp3 & wp8 stations:*									
1736		Sta								
1737	wp3	349+41.27			*=g_bso(Pt-»wp3,<gblock1>,34523,"s")*					
1738	wp8	350+58.72			*=g_bso(Pt-»wp8,<gblock1>,34523,"s")*					
1739										
1740	*Get wp3 offset to baseline:*									
1741		Offset								
1742	wp3	-0.0275			*=g_bso(Pt-»wp3,<gblock1>,34523,"o")*					
1743										
1744	*Tabulate working point coordinates:*									
1745		N	E							
1746	wp1	225954.0630	1132435.9160							
1747	wp2	225959.3992	1132453.1069							
1748	wp3	225979.0314	1132442.1147							
1749	wp4	225998.6635	1132431.1225							
1750	wp5	225999.2555	1132396.7152							
1751	wp6	226036.2175	1132565.3035							
1752	wp7	226029.9920	1132545.2475							
1753	wp8	226013.8500	1132554.2855							
1754	wp9	225997.7080	1132563.3235							
1755	wp10	226003.9335	1132583.3795							
1756										

	A	B	C	D	E	F	G	H	I	J
1757	*Define stakeout gblock:*									
1758	<gblock 2>									
1759	[Pt->101]«1132436.02378727,225987.351180291»									
1760	[Pt->100]«1132440.90919751,225978.62577103»									
1761										
1762	[Pt->wp3]«1132442.11470727,225979.031352931»									
1763	[Pt->PT]«1132469.5517,225987.548»									
1764	[Pt->200]«1132555.50595794,226014.228842401»									
1765	[Pt->201]«1132550.6205477,226022.954251662»									
1766										
1767	[Pt->wp1]«1132435.91603064,225954.063014209»									
1768	[Pt->wp2]«1132453.10688031,225959.399182093»									
1769	[Pt->wp3]«1132442.11470727,225979.031352931»									
1770	[Pt->wp4]«1132431.12253423,225998.663523769»									
1771	<arc>(+clock)									
1772	[Pt->102]«1132414.81757681,226051.191119977»									
1773	[Pt->wp5]«1132396.71520953,225999.255541624»									
1774										
1775	[Pt->wp6]«1132565.30346086,226036.217532825»									
1776	[Pt->wp7]«1132545.24746958,226029.992003627»									
1777	[Pt->wp8]«1132554.28547853,226013.849996494»									
1778	[Pt->wp9]«1132563.32348748,225997.707989361»									
1779	[Pt->wp10]«1132583.37947876,226003.933518559»									
1780	<end gb>									
1781										
1782	*Setup stakeout DXF-block:*									
1783	<dxf>									
1784	<layer> stakeout									
1785	[$A$1758:$A$1780] <gblock 2>									
1786	<color> 5									
1787	<ln_style> 3									
1788	<layer> baseline									
1789	[$A$1668:$A$1674] <gblock 1>									
1790	<End>									
1791										
1792	*Viewport Reference:*									
1793										
1794			Baseline	Stakeout	abut 1 detail	abut 2 detail				
1795	1. viewport		<viewport 1>	<viewport 2>	<viewport 3>	<viewport 4>				
1796	2. Source Gblock		[$A$1668:$A$16	[$A$1758:$A$17	[$A$1758:$A$17	[$A$1758:$A$1780] <gblock 2>				
1797	3. Destination Cell		$F$1668	$A$1684	$H$1685	$H$1696				
1798	4. Horizontal Shift (in points)		5	5	5	5				
1799	5. Vertical Shift (in points)				5					
1800	6. VP Box Width (in points)		200	380	100	100				
1801	7. VP Height (in points)		100	110	110	110				
1802	8. X_scale									
1803	9. Y_scale									
1804	10. Rotation (degrees)			17.2447	17.2447	17.2447				
1805	11. Clip Box --> CP (X/Y string)				[Pt->wp3]«11324	[Pt->wp8]«1132554.28547853,226013.849996494»				
1806	12. Clip Box --> clp_x				4	4				
1807	13. Clip Box --> clp_y				4	4				
1808	14. Include Point Symb (y/n)									
1809	15. Display VP Border (y/n)									
1810	16. Suspend Plot Update (y/n)									
1811	17. Hide Viewport (y/n)									
1812	18. Direction Symbol (n/x)		n	n	n	n				
1813										
1814										

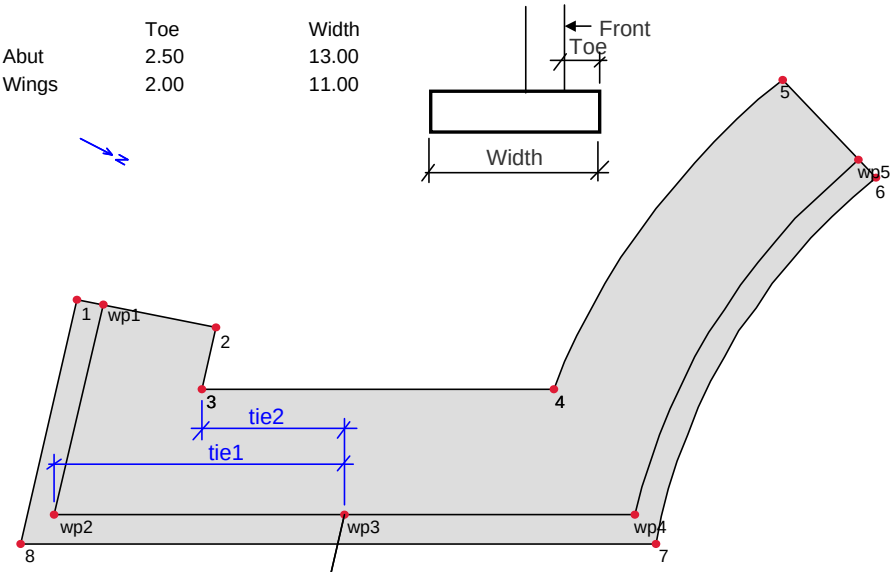
Example 2: Footing Geometry

Objectives:

1. Obtain dimensions and area of footing of abutment 1 from Example 2 bridge.
2. Setup footing DXF-block

Design Footing Dimensions:

	Toe	Width
Abut	2.50	13.00
Wings	2.00	11.00



Abut 1 Footing Plan

<VP 5> [Par's 1-18 @ $C$1907]

Copy points from Example 1 and define front face gblock:

```
<gblock 3>
[Pt->wp1]«1132435.91603064,225954.063014209»
[Pt->wp2]«1132453.10688031,225959.399182093»
[Pt->wp3]«1132442.11470727,225979.031352931»
[Pt->wp4]«1132431.12253423,225998.663523769»
<arc>(+clock)
[Pt->102]«1132414.81757681,226051.191119977»
[Pt->wp5]«1132396.71520953,225999.255541624»

[Pt->wp3]«1132442.11470727,225979.031352931»
[Pt->PT]«1132469.5517,225987.548»
<end gb>
```

	A	B	C	D	E	F	G	H	I	J
1861	*Locate footing corner points and define footing outline gblock:*									
1862	Note: List points in clockwise order in order to use gblock in *g_ba()* function									
1863										
1864	<gblock 4>									
1865	[Pt->1]«1132436.50893818,225952.152919801»				*=p_rang(NP->1,Pt-»wp2,Pt-»wp1,90,2)*					
1866	[Pt->2]«1132433.2479467,225962.658439043»				*=p_rang(NP->2,Pt-»wp2,Pt-»wp1,-90,11-2)*					
1867	[Pt->3]«1132438.35975539,225964.245182567»				*=i_ll(NP->3,Pt-»wp1,Pt-»wp2,Pt-»wp2,Pt-»wp3,-(11-2),-(13-2.5))*					
1868	[Pt->4]«1132425.05112313,225988.014581141»				*=i_lc(NP->4,Pt-»wp2,Pt-»wp4,Pt-»102,55-2+11,Pt-»wp4,"n",-(13-2.5))*					
1869	<arc>(+clock)									
1870	[Pt->102]«1132414.81757681,226051.191119977»									
1871	[Pt->5]«1132393.75300398,225990.756992439»				*=p_rang(NP->5,Pt-»102,Pt-»wp5,180,11-2)*					
1872	[Pt->6]«1132397.37347743,226001.14410811»				*=p_rang(NP->6,Pt-»5,Pt-»wp5,180,2)*					
1873	<arc>(-clock)									
1874	[Pt->102]«1132414.81757681,226051.191119977»									
1875	[Pt->7]«1132432.54337107,226001.243167323»				*=i_lc(NP->7,Pt-»wp2,Pt-»wp4,Pt-»102,55-2,Pt-»wp4,"n",2.5)*					
1876	[Pt->8]«1132456.54674977,225958.372805886»				*=i_ll(NP->8,Pt-»wp1,Pt-»wp2,Pt-»wp2,Pt-»wp3,2,2.5)*					
1877	[Pt->1]«1132436.50893818,225952.152919801»									
1878	<end gb>									
1879										
1880	*Determine footing outline dimensions:*									
1881	2-3	5.3524		*=g_bl(<gblock4>,LEFT(2-3,1),RIGHT(2-3,1))*						
1882	3-4	27.2416		*=g_bl(<gblock4>,LEFT(3-4,1),RIGHT(3-4,1))*						
1883	4-5	31.7424		*=g_bl(<gblock4>,LEFT(4-5,1),RIGHT(4-5,1))*						
1884	6-7	35.8496		*=g_bl(<gblock4>,LEFT(6-7,1),RIGHT(6-7,1))*						
1885	7-8	49.1328		*=g_bl(<gblock4>,LEFT(7-8,1),RIGHT(7-8,1))*						
1886	8-1	20.9810		*=g_bl(<gblock4>,LEFT(8-1,1),RIGHT(8-1,1))*						
1887										
1888	*Determine footing tie dimensions to wp3:*									
1889	tie1 =	25.0761		*=g_lo(Pt-»wp3,Pt-»8,Pt-»7,"t")*						
1890	tie2 =	11.0671		*=g_lo(Pt-»wp3,Pt-»3,Pt-»4,"t")*						
1891										
1892	*Determine footing area:*									
1893	Area =	1011.36		*=g_ba(<gblock4>)*						
1894										
1895	*Setup footing DXF-block:*									
1896	<dxf>									
1897	<layer> ff_line									
1898	[$A$1848:$A$1859] <gblock 3>									
1899	<layer> ftg_outline									
1900	<color> 5									
1901	<plot_opt> 2			*exclude ftg points from plot*						
1902	[$A$1864:$A$1878] <gblock 4>									
1903	<End>									
1904										

	A	B	C	D	E	F	G	H	I	J
1905	*Viewport Reference:*									
1906			Abut 1 Footing							
1907	1. viewport		<viewport 5>							
1908	2. Source Gblock		[$F$1908:$F$1911] <gblist 1>			<gblist 1>			0n(1)/off(0)	
1909	3. Destination Cell		$A$1825			[$A$1864:$A$1878] <gblock 4>			1	
1910	4. Horizontal Shift (in points)		10			[$A$1848:$A$1859] <gblock 3>			1	
1911	5. Vertical Shift (in points)		5			<end list>				
1912	6. VP Box Width (in points)									
1913	7. VP Height (in points)									
1914	8. X_scale		5.5							
1915	9. Y_scale		5.5							
1916	10. Rotation (degrees)		119.2447	*=17.2447323548023+102*						
1917	11. Clip Box --> CP (X/Y string)									
1918	12. Clip Box --> clp_x									
1919	13. Clip Box --> clp_y									
1920	14. Include Point Symb (y/n)									
1921	15. Display VP Border (y/n)									
1922	16. Suspend Plot Update (y/n)									
1923	17. Hide Viewport (y/n)									
1924	18. Direction Symbol (n/x)		n							