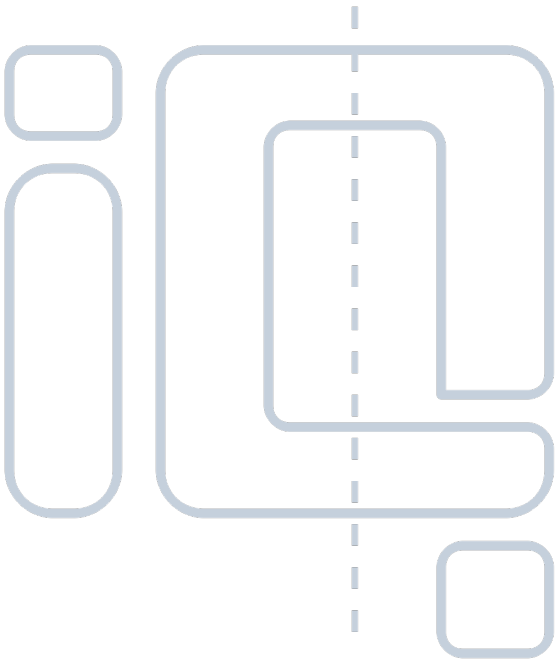
IQcad
User Manual

v2026



Contents

I Introduction 3

1 Welcome to IQcad 5
- 1.1 What IQcad Does . . . 5
- 1.2 Supported Platforms . . . 6
- 1.3 How This Manual Is Organized . . . 6

II Licensing 7

2 Licensing 11
- 2.1 License Manager . . . 11
- 2.2 Updates . . . 12
- 2.3 Get Support . . . 12
- 2.4 Modules . . . 13
- 2.5 IQcad Basic . . . 13
- 2.6 IQcad Pro . . . 18
- 2.7 IQcad Civil . . . 19
- 2.8 IQcad Point Cloud . . . 23
- 2.9 IQcad Drone . . . 24

III CAD Environment 25

3 Interface Overview 29
- 3.1 Main Areas . . . 30
- 3.2 Layout Persistence . . . 30
- 3.3 Ribbon Tabs . . . 31

4 Drawings 33
- 4.1 New Drawing . . . 33
- 4.2 Open a Drawing . . . 33
- 4.3 Saving . . . 33
- 4.4 Drawing Properties . . . 34
- 4.5 Multiple Drawings . . . 34

5 Navigation 35

5.1 Zoom . . . 35
5.2 Pan . . . 35
5.3 Grid . . . 35
5.4 Rulers . . . 35
5.5 3D View Commands . . . 36
5.6 UCS Gizmo . . . 36
5.7 Map Background . . . 38
5.8 Retained Geometry (experimental) . . . 40

6 Drawing Tools **41**
6.1 Object Snapping (Osnap) . . . 41
6.2 Lines and Curves . . . 42
6.3 Shapes and Fills . . . 44
6.4 Annotation . . . 45
6.5 Insert . . . 45
6.6 Dimensions . . . 46
6.7 Civil Drawing Tools . . . 47

7 Modifying Objects **49**
7.1 Modify Commands . . . 51
7.2 Mesh and Surface Conversion . . . 52
7.3 Clipboard . . . 52
7.4 OSNAP in Edit Commands . . . 53
7.5 Grips . . . 53
7.6 Undo and Redo . . . 54

8 Selection **55**
8.1 Selection Methods . . . 55
8.2 Selection Feedback . . . 55
8.3 Selection Filter . . . 56
8.4 Cursor Modes . . . 57
8.5 Right-Click Behaviour . . . 57

9 Layers, Linetypes and Text Styles **59**
9.1 Layer Manager . . . 59
9.2 Ribbon Layer Dropdown . . . 60
9.3 Linetype Manager . . . 60
9.4 Drawing Variables . . . 60
9.5 Text Styles . . . 61
9.6 Colour and Linetype Inheritance . . . 61

10 Object Inspector **63**
10.1 Architecture . . . 63
10.2 Selection Filter . . . 64
10.3 Multi-Selection . . . 64
10.4 Common Properties (All Objects) . . . 65
10.5 Canvas / Drawing Defaults . . . 66
10.6 Line Properties . . . 66
10.7 Polyline and Multiline Properties . . . 66
10.8 Hatch Properties . . . 66
10.9 3D Polyline Properties . . . 67
10.10 Text Properties . . . 67

10.11 Multileader Properties . . . 67
10.12 Arc / Circle Properties . . . 67
10.13 Dimension Properties . . . 68
10.14 AEC Survey Point Properties . . . 68
10.15 Surface Properties . . . 68
10.16 Alignment Properties . . . 69
10.17 Table Properties . . . 69
10.18 Viewport Properties . . . 69
10.19 Tunnel Section Line Properties . . . 70
10.20 Set Properties . . . 70

11 Blocks and Attributes **71**
11.1 Creating a Block . . . 71
11.2 Inserting a Block . . . 71
11.3 Editing a Block Definition . . . 72
11.4 Block Manager . . . 72
11.5 Exploding a Block . . . 72
11.6 Attributes . . . 73
11.7 Blocks, Layers and Colours . . . 73
11.8 Blocks in the File Formats . . . 73

12 Command Line **75**
12.1 Command Suggestions . . . 75
12.2 Command Reference . . . 76
12.3 Entering Coordinates . . . 81
12.4 Running Scripts . . . 82

13 Format Settings **85**
13.1 Layers . . . 85
13.2 Linetypes . . . 85
13.3 Drawing Variables . . . 85
13.4 Point Style (Ctrl+Shift+D) . . . 85
13.5 Text Style (Ctrl+Shift+T) . . . 86
13.6 Surface Styles . . . 86
13.7 Table Styles . . . 87
13.8 Multiline Styles (Ctrl+Shift+M) . . . 88
13.9 Dimension Styles . . . 89
13.10 Plot Style Tables (CTB) . . . 90
13.11 Tunnel Section Line Styles . . . 90
13.12 Parcel Styles . . . 92
13.13 Alignment Styles . . . 92
13.14 Alignment Label Styles . . . 93
13.15 Civil Label Styles . . . 93
13.16 Unified Style Managers . . . 94

14 Exporting and Importing **97**
14.1 Native File Formats . . . 97
14.2 Exporting . . . 97
14.3 Importing . . . 99

15 Layouts, Viewports and Printing **105**
15.1 Creating and Managing Layouts . . . 105

15.2 Page Setup . . . 105
15.3 Creating Viewports . . . 106
15.4 Scale, Lock and Freeze . . . 106
15.5 Viewport Commands . . . 107
15.6 Draw Order . . . 107
15.7 Printing . . . 107
15.8 Plotting to PDF . . . 108

16 DWG Compatibility **109**
16.1 Supported Versions . . . 109
16.2 What Round-Trips . . . 109
16.3 Opening and Re-saving Untouched . . . 110
16.4 Backends and the Fallback Chain . . . 110
16.5 Exporting Survey Objects . . . 111
16.6 Known Limitations . . . 111

17 Application Settings **113**
17.1 General . . . 113
17.2 Display . . . 113
17.3 Grid . . . 114
17.4 Ruler . . . 114
17.5 UCS Gizmo . . . 114
17.6 ViewCube . . . 114
17.7 Advanced . . . 115
17.8 DWG . . . 115
17.9 Where Settings Are Stored . . . 115

IV Super Tools **117**

18 Overview **121**
18.1 Access . . . 121
18.2 Ribbon and menu groups . . . 121
18.3 Current commands . . . 122

19 Cleanup **123**
19.1 FLATTEN . . . 123

20 Polylines **125**
20.1 PLSIMPLIFY . . . 125
20.2 PLGETZ . . . 126
20.3 BESTFITLINE . . . 127

V Civil Module **129**

21 Civil Toolspace and Settings **133**
21.1 Civil Toolspace . . . 133
21.2 Civil Settings . . . 135
21.3 Elevation Band Legend . . . 137

22 AEC Survey Points **139**
22.1 Properties . . . 139

22.2 Point Style Library . . . 139
22.3 Importing Survey Points from CSV . . . 140
22.4 Point Groups . . . 141
22.5 Point Manager . . . 145

23 TIN Surfaces **149**
23.1 Triangulation Algorithm . . . 149
23.2 Creating a Surface . . . 150
23.3 Adding Data to a Surface . . . 150
23.4 Surface Display & Styles . . . 151
23.5 Boundaries . . . 152
23.6 Surface Properties . . . 152
23.7 Surface Statistics . . . 155
23.8 Elevation Query . . . 155
23.9 Elevation Tracker . . . 156
23.10 Performance & Rendering . . . 156
23.11 IQML Streaming Save . . . 156
23.12 Decimating (Simplifying) a Surface . . . 157
23.13 Deleting a Surface . . . 158
23.14 Importing Large Surfaces (LandXML Fast Path) . . . 158
23.15 Exporting Surfaces . . . 158
23.16 Cross-Sections from a DEM . . . 158
23.17 Comparing Surfaces (Surface Differencing) . . . 159
23.18 Earthwork Volume Report . . . 160
23.19 Balancing Volumes . . . 161
23.20 Surface Editing Commands . . . 162
23.21 Hiding and Restoring Triangles . . . 162
23.22 Breaklines . . . 163
23.23 Spot Elevations . . . 163
23.24 Contour Label Placement . . . 164

24 Alignments and Profiles **165**
24.1 Horizontal Alignments . . . 165
24.2 Profile Views . . . 170
24.3 Alignment Editor . . . 172

25 Sections and Parcels **175**
25.1 Cross-Sections . . . 175
25.2 Association Highlighting . . . 176
25.3 Prospector Style Columns . . . 176
25.4 Parcels . . . 176

26 Tunnel Analysis and Coordinate Transformations **181**
26.1 Tunnel Section Line — Measured Profile . . . 181
26.2 Coordinate Transformations . . . 184

27 Assemblies and Corridors **193**
27.1 Overview . . . 193
27.2 Assembly elements . . . 194
27.3 Pavement structure — courses and materials . . . 201
27.4 Section styles . . . 204
27.5 Conditional rows, ditches and benching . . . 206

27.6 Standard sample library . . . 209
27.7 Editor display . . . 216
27.8 Building a corridor . . . 216

28 Grading, Culverts and the 3D Object Viewer 219
28.1 Grading . . . 219
28.2 Culverts . . . 222
28.3 3D Object Viewer . . . 224

29 Sites 227
29.1 Sites in the Prospector . . . 227
29.2 Creating a Site . . . 228
29.3 Assigning an Alignment to a Site . . . 228
29.4 Sites in the File Formats . . . 228

VI Point Cloud Module 229

30 Point Clouds 233
30.1 Importing a Point Cloud . . . 233
30.2 Attaching a Cache . . . 233
30.3 Exporting a Point Cloud . . . 233
30.4 Elevation Clipping . . . 234
30.5 Point Cloud Display . . . 234
30.6 Prospector Integration . . . 234

31 Point Cloud Management 235

32 Point Cloud Sections and Sheets 237
32.1 Point Cloud Sections . . . 237
32.2 Point Cloud Sections Along Alignment . . . 238
32.3 Point Cloud Section Sheet Styles . . . 242

VII Drone Module 245

33 Drone Survey 249
33.1 DEM and orthophoto . . . 249
33.2 Moving around . . . 249
33.3 Points, breaklines and boundaries . . . 250
33.4 Grid and low-pass . . . 250
33.5 Surface and contours . . . 250
33.6 Send to the drawing . . . 251
33.7 Sessions . . . 251
33.8 Keys . . . 251

VIII Command Reference 253

34 Command Reference 257
34.1 CAD Commands . . . 257
34.2 Civil Commands . . . 266
34.3 Point Cloud Commands . . . 273

34.4 Drone Commands . . . 275
34.5 Scripting & Automation . . . 275

35 Keyboard Shortcuts **277**
35.1 File . . . 277
35.2 Edit . . . 277
35.3 View . . . 278
35.4 Format . . . 278
35.5 Drawing Tools . . . 278
35.6 Modify Tools . . . 279
35.7 General . . . 279

36 Program Defaults Reference **281**
36.1 Application Settings . . . 281
36.2 Drawing Variables . . . 283
36.3 Layer Defaults . . . 284
36.4 Text Styles . . . 285
36.5 Dimension Styles . . . 285
36.6 Surface Styles . . . 286
36.7 Alignment Styles . . . 287
36.8 Parcel Styles . . . 287
36.9 Tunnel Section Line Styles . . . 287
36.10 Civil Object Layer Defaults . . . 288
36.11 Civil Label Styles . . . 289
36.12 Civil Settings . . . 290

IX SDK **293**

37 Scripting **297**
37.1 Overview . . . 297
37.2 Running scripts . . . 297
37.3 File format . . . 298
37.4 Examples by category . . . 298
37.5 Patterns . . . 301
37.6 Limitations . . . 301
37.7 Python scripting . . . 301
37.8 Future scripting languages . . . 307

38 Python Scripting Reference **309**
38.1 Intro . . . 309
38.2 Running Python . . . 309
38.3 The object model . . . 310
38.4 Class reference . . . 311
38.5 Decorators and events . . . 316
38.6 Dialogs . . . 316
38.7 Examples . . . 317
38.8 Dialogs and message boxes . . . 319
38.9 Live forms . . . 320
38.10 Ribbon & menu . . . 320
38.11 Limitations and tips . . . 321

39 Pascal Scripting **323**

39.1 Intro . . . 323
39.2 Running Pascal . . . 323
39.3 The object model . . . 324
39.4 Class and global reference . . . 325
39.5 Commands and events . . . 329
39.6 Dialogs . . . 330
39.7 Available RTL . . . 331
39.8 Examples . . . 331
39.9 Dialogs and message boxes . . . 334
39.10 Live forms . . . 335
39.11 Ribbon & menu . . . 335
39.12 Limitations and tips . . . 336

40 Plugins (Native SDK) 337
40.1 Plugins versus scripting . . . 339
40.2 Architecture and design . . . 340
40.3 The plugin lifecycle . . . 342
40.4 Trust and the Plugin Manager . . . 343
40.5 Installing a plugin . . . 344
40.6 The manifest (`plugin.json`) . . . 344
40.7 Tutorial 1 — your first plugin (Free Pascal) . . . 345
40.8 Tutorial 2 — drawing geometry . . . 348
40.9 Parameter dialogs and message boxes . . . 350
40.10 Live tool windows . . . 351
40.11 Ribbon & menu contributions . . . 354
40.12 Writing a plugin in C or C++ . . . 355
40.13 The bundled samples . . . 357
40.14 API reference . . . 359
40.15 Debugging plugins . . . 367
40.16 Building for other platforms . . . 368
40.17 Versioning and compatibility — summary . . . 368

41 The Blade Runner Script Editor 369
41.1 Buffers and Files . . . 369
41.2 Running a Script . . . 369
41.3 The Sidebar . . . 370
41.4 Find, Replace and Navigation . . . 370
41.5 The Command Palette . . . 370
41.6 Appearance . . . 370

X Tutorials 371

42 Tutorials 373
42.1 Tutorial 1: Working with Survey Points . . . 374
42.2 Tutorial 2: Working with TIN Surfaces . . . 378
42.3 Tutorial 3: Working with Alignments . . . 382
42.4 Tutorial 4: Working with Parcels . . . 387

XI Features and Changelog 391

43 Features and Changelog 393

43.1 Changelog 393
43.2 Drawing & Editing 426
43.3 Civil / AEC 426
43.4 Styles 427
43.5 Layout & Printing 427
43.6 File I/O 428
43.7 User Interface 428

Index **429**

IQcad is a 2D/3D CAD application for engineering and civil survey work. It provides standard drafting tools together with specialised civil objects for survey points, TIN surfaces, horizontal alignments, tunnel section line deviation analysis, and coordinate transformations, plus large-scale point cloud handling, drone survey, and a scripting/plugin SDK.

Part I

Introduction

1 Welcome to IQcad

IQcad is a 2D/3D CAD application for engineering and civil survey work. It combines standard drafting tools with specialised civil objects — survey points, TIN surfaces, horizontal alignments, profiles, corridors, gradings and culverts — plus large-scale point cloud handling, tunnel section analysis and coordinate transformations.

1.1 What IQcad Does

Drafting. The full set of everyday CAD tools: lines, polylines, arcs, circles, splines, hatches, text, tables and dimensions, organised in layers with linetypes, text styles and dimension styles. Editing follows the familiar select-then-modify model with grips, object snaps, and a command line that accepts AutoCAD-style command names. Spatial indexing and type-side document lists keep pick, OSNAP, civil tools and TRIM responsive on large drawings.

Civil engineering. Survey points with styles and groups, TIN surfaces with contours, breaklines, decimation and earthwork volume reports, horizontal alignments with spirals and a PI table editor, profile views, cross-section sheets, assemblies and corridors, site gradings, parametric box culverts, parcels, and a tunnel section line deviation analysis.

Point clouds. Import LAS/LAZ, E57, PLY, PTS/PTX and text formats with hundreds of millions of points, render them with adaptive level of detail, clip them by elevation, and cut vertical or horizontal sections that can be traced into plottable 2D drawings.

Drone survey. A GeoTIFF DEM and an orthophoto become survey points, breaklines and a surface in the drawing.

Interoperability. Native `.iqz` drawings, an in-process DWG reader and writer covering R2000 to R2018 with no external converter, DXF exchange, LandXML, GeoJSON, Esri JSON (Hellenic Cadastre), KML/KMZ, Shapefile, SVG, PDF and image export. Civil 3D COGO points, point groups, TIN surfaces, parcels and alignments are read from DWG and can be written back as genuine Civil 3D objects.

Automation. Command scripts, embedded Python and Pascal scripting, and a native plugin SDK for compiled extensions.

1.2 Supported Platforms

IQcad runs natively on **Windows** (x86-64), **Linux** (x86-64) and **macOS** (Apple Silicon).

1.3 How This Manual Is Organized

- **Licensing** — the license manager, the five modules, the commands each module unlocks, updates, and Get Support.
- **CAD Environment** — the interface, drawings, navigation, drawing and modification tools, selection, layers and styles, blocks, the Object Inspector, the command line, format settings, import/export, DWG compatibility, sheet layout and printing, and the application settings.
- **Super Tools** — Express/CadTools-style cleanup and polyline helpers (flatten, simplify, and more as the suite grows).
- **Civil Module** — the Civil Toolspace and every civil object type, from survey points to corridors, culverts and sites.
- **Point Cloud Module** — importing, displaying, managing and sectioning point clouds.
- **Drone Module** — Drone Survey from a DEM and an orthophoto.
- **Command Reference** — every command recognised by the command line, keyboard shortcuts, and the factory defaults.
- **SDK** — command scripting, the Python and Pascal scripting references, the native plugin SDK, and the Blade Runner script editor.
- **Tutorials** — step-by-step guides for common civil workflows.
- **Features and Changelog** — a feature summary and the release history.

Part II

Licensing

The license manager, the five product modules, and the commands each module unlocks.

2 Licensing

IQcad Basic is always on. Pro, Civil, Point Cloud and Drone turn on only when the license key grants them. A drawing still opens and saves every object when a module is off. The commands of that module do not run, and its menu and ribbon tab are hidden.

2.1 License Manager

Help > About, on the **License & Modules** tab, shows each module and whether it is on. **Help > License…** opens the License Manager, where you enter a key.

The manager shows the status, who the license is issued to, the version, how many machines the key allows, the expiry, and the machine id. Each module is marked on or not licensed. IQcad Basic stays marked on without a key.

The status is one of:

- **Basic** — no key is active. Only IQcad Basic runs.
- **Trial** — a trial is running. The line shows how many days remain.
- **Activated** — a key is active on this machine. A key with no end date is shown as a perpetual license.
- **Expired** — the key or the trial has ended. The paid modules turn off.
- **Grace period exceeded** — the key was not checked with the activation server inside its offline window. Only IQcad Basic stays on.
- **Cryptography unavailable** — the license libraries did not start. Reinstall IQcad, or contact support.

Paste the serial key and click **Activate**. The server address is already filled in. Leave it unless support gives you another one. **Update License** checks the key with the server again. **Deactivate** removes the license from this machine. You confirm that step before it runs.

IQcad warns at startup when a license or a trial ends within 30 days. A perpetual key, and a Basic session with no key, do not warn.

An activated key keeps working without a network connection. The offline window is the one stored in the key. When the key does not set one, IQcad uses 30 days from the last successful check. After that, connect to the internet and click **Update License**. Until the check succeeds, only IQcad Basic stays on.

Typing a command from a module that is off prints a line on the command line: the module is not licensed, and **Help > License** is where you enter a key. A menu item for that module shows the same message.

These parts of the window follow the license:

- Super Tools menu and ribbon tab, Extrude, Sweep, the 3D tools button, the Scripting pane, Blade Runner and Plugin Manager — IQcad Pro.
- Civil menu, Civil ribbon tab and the Civil Toolspace — IQcad Civil.
- Point Cloud menu and ribbon tab — IQcad Point Cloud.
- Drone menu and ribbon tab — IQcad Drone.
- Map background — IQcad Pro or IQcad Civil. Either module is enough.

2.2 Updates

Help > Check for Updates… looks for a newer IQcad. The same check runs once after startup. It stays quiet when this version is current, or when the update server cannot be reached. Use the menu to see that error.

IQcad asks before it downloads. The file is kept only when it matches the signed manifest. The download is one archive of the program folder. The program file is at the root of the archive. The same archive updates an installed copy and an unpacked folder.

On confirm, IQcad closes. A helper waits until the program has left, extracts the archive into the folder that contained the program, asks the system for permission when that folder is protected, and then starts IQcad again. Settings stay where they are.

The running version is the four-part version in **Help > About**.

2.3 Get Support

Help > Get Support… and the **Get Support** button on the **Help** ribbon send a report to IQSoft. The same form is on the web at https://iqcad.iqsoft.xyz/support/ (English) and https://iqcad.iqsoft.xyz/el/support/ (Greek). **Help > About** also shows that address.

Subject and **What happened** are required. **Email** is required, so the reply can reach you. **Name** is optional.

Add files… attaches screenshots, a log, or a drawing. **Remove** drops the selected files from the list. The limit is 8 files and 20 MB altogether. The accepted types are png, jpg, jpeg, webp, pdf, txt, log, csv, xml, dxf, dwg, zip and iqz. Attach the drawing when IQSoft needs the file.

Include version, platform and license starts ticked. It sends the IQcad version, the operating system, the modules that are on, and the license. The serial is masked: the first three groups stay, and the rest is replaced with asterisks. With the box cleared, those fields are left empty.

Send leaves the window open while the report goes out. When the site answers, a message shows the ticket number, for example `Sent. Your ticket number is IQ-2026-0001.` IQSoft replies by email. When the site cannot be reached, the form stays open with the text still in it.

2.4 Modules

Module	What it adds
IQcad Basic	Drafting: draw, modify, annotate, layers, blocks, dimensions, files, layouts and printing. Always on.
IQcad Pro	Extrude, sweep, flatten, the 3D object viewer, Super Tools, map background, Python, Pascal Script, Blade Runner and plugins.
IQcad Civil	Survey points, surfaces, alignments, profiles, sections, corridors, grading, parcels, culverts, tunnel section lines and coordinate transformations. Map background is included.
IQcad Point Cloud	Import, attach, edit, classify and compare point clouds, and cut sections from them.
IQcad Drone	The Drone Survey workspace.

How a command is used is in the *command reference* (page 257). The tables below are the license boundary: a command runs only when its module is on.

2.5 IQcad Basic

Basic is the CAD editor. Command scripts (`.scr`) are included. `SCRIPT` is the command.

The diagnostic commands at the end of this list also stay available. They add sample geometry or print renderer information. They are not part of a paid module.

2.5.1 File

Command	Alias	Description
`CLOSE`	—	Close current drawing
`EXPORTDXF`	—	Export drawing to DXF
`EXPORTIMAGE`	—	Export drawing to image
`EXPORTPDF`	—	Export drawing to PDF
`NEW`	—	Create new empty drawing
`OPEN`	`O`	Open drawing file
`SAVE`	—	Save drawing to file
`SAVEAS`	—	Save drawing with new name

2.5.2 Drawing

Command	Alias	Description
`-HATCH`	`BH`	Create hatch (command line)
`3DFACE`	`3F`	Draw a 3D face
`3DPOLY`	`3DP`	Draw a 3D polyline
`ARC`	`A`	Draw an arc
`ARC_CONT`	—	Arc: continue from last
`ARC_CSA`	—	Arc: center, start, angle
`ARC_CSE`	—	Arc: center, start, end

continues on next page

Table 2 – continued from previous page

Command	Alias	Description
ARC_CSL	—	Arc: center, start, length
ARC_SCA	—	Arc: start, center, angle
ARC_SCE	—	Arc: start, center, end
ARC_SCL	—	Arc: start, center, length
ARC_SEA	—	Arc: start, end, angle
ARC_SED	—	Arc: start, end, direction
ARC_SER	—	Arc: start, end, radius
ARROW	AR	Draw an arrow
BEZIER	BEZ	Draw a Bezier curve
CIRCLE	C	Draw a circle
CIRCLE_2P	—	Circle: 2-point
CIRCLE_3P	—	Circle: 3-point
CIRCLE_CD	—	Circle: center, diameter
ELLIPSE	EL	Draw an ellipse
HATCH	H	Create hatch pattern fill
LINE	L	Draw a line
MLEADER	MLD	Draw a multileader
MTEXT	MT	Place multiline text
MULTILINE	ML	Draw a multiline
POINT	P, PT	Place a point
POLYGON	POL	Draw a regular polygon
POLYLINE	PL, PLINE	Draw a polyline
RECTANGLE	REC, RECT	Draw a rectangle
SOLID	SO	Draw a 2D filled solid
SPLINE	SP, SPL	Draw a spline
TABLE	TBL	Place a table
TABLESTYLE	—	Manage table styles
TEXT	T	Place text
WIPEOUT	WO	Mask an area with a wipeout
XLINE	XL	Draw a construction line

2.5.3 Modify

Command	Alias	Description
ALIGN	AL	Align objects using source/destination point pairs
ARRAY	ARR	Rectangular, polar or path array of selected objects
BREAK	BR	Break an entity between two points
CHAMFER	CHA	Chamfer two lines
COPY	CO, CP	Copy selected objects
COPYCLIP	CC	Copy to clipboard
CUTCLIP	CX	Cut to clipboard
DIVIDE	DIV	Place points dividing an entity into equal segments
DUPLICATE	DU	Duplicate selected entities
ERASE	E	Delete selected objects
EXPLODE	—	Explode blocks/multilines
EXTEND	EX	Extend object to edge

continues on next page

Table 3 – continued from previous page

Command	Alias	Description
FILLET	F	Fillet two lines/arcs
JOIN	J	Join lines, arcs and polylines
MEASURE	ME	Place points along an entity at a fixed interval
MIRROR	MI	Mirror selected objects
MOVE	M	Move selected objects
OFFSET	O	Offset an object
PASTECLIP	CV	Paste from clipboard
PASTEORIG	—	Paste at original coordinates
PEDIT	PE	Edit polyline
REDO	—	Redo last undone action
ROTATE	RO	Rotate selected objects
SCALE	SC	Scale selected objects
SELECTALL	SA	Select all entities
TRIM	TR	Trim object to edge
UNDO	—	Undo last action

2.5.4 View

Command	Alias	Description
3DORBIT	ORBIT	3D orbit view
ALIGNVIEW	AV	Align view to two picked points
PAN	—	Pan the view (middle mouse)
PLAN	—	Return to plan view
PLANVIEW	—	Reset view twist to north-up
REDRAW	—	Refresh the display
REGEN	REA	Regenerate the display
VIEWTWIST	VT	Set 2D view rotation angle
VPOINT	VP	Set viewpoint preset
ZE	—	Zoom to extents
ZI	—	Zoom in
ZO	—	Zoom out
ZOOM	—	Zoom extents, window, or previous
ZOOMOBJECT	—	Zoom to a picked object
ZW	—	Zoom window

2.5.5 Measure

Command	Alias	Description
ANGLE	AN	Measure angle (3 points)
AREA	AA	Measure area (pick points)
DIST	DI	Measure distance between two points
HELP	?	Show command reference
ID	—	Report point coordinates
LIST	LI	List selected object properties

2.5.6 Format

Command	Alias	Description
CURRENTLAYER	CLAYER	Set the active layer: CURRENTLAYER name
LAYERS	LA	Open layers dialog
LINETYPES	LT	Open linetypes dialog
MAKELAYER	—	Create a layer: MAKELAYER name [color]
PLOTSTYLE	—	Open plot style dialog
POINTSTYLE	—	Open point style dialog
PROPERTIES	PR	Toggle properties panel
TEXTSTYLE	ST	Open text styles dialog

2.5.7 Dimensions

Command	Alias	Description
DIMALIGNED	DIMAL	Create aligned dimension
DIMANGULAR	DIMANG	Create angular dimension
DIMARC	—	Create arc length dimension
DIMBASELINE	DIMBASE	Create baseline dimension
DIMCONTINUE	DIMCONT	Create continue dimension
DIMDIAMETER	DIMDIA	Create diameter dimension
DIMJOGGED	DIMJOG	Create jogged dimension
DIMLINEAR	DIML	Create linear dimension
DIMORDINATE	DIMORD	Create ordinate dimension
DIMRADIUS	DIMRAD	Create radius dimension
DIMSTYLE	D	Open Dimension Style Manager

2.5.8 Blocks

Command	Alias	Description
BCLOSE	BC	Close block editor
BEDIT	BE	Edit block definition in-place
BLOCK	B	Create block from selected entities
BLOCKMANAGER	BM	Open Block Manager dialog
BLOCKS	BL	List block definitions
IMAGEATTACH	IA	Attach a raster image
INSERT	INS	Insert a block reference

2.5.9 Layout and printing

Command	Alias	Description
ABOVE	—	Move selected objects above a reference object

continues on next page

Table 9 – continued from previous page

Command	Alias	Description
BACK	—	Send selected objects to back
BELOW	—	Move selected objects below a reference object
DRAWORDER	DR	Display order (Front/Back/Above/Below)
FRONT	—	Bring selected objects to front
LAYOUT	LO	New / Delete / Rename / List / Set layout
MVIEW	MV	Create viewport (Fit or 2-point)
MVSETUP	MVS	Viewport setup (rotate/scale/title)
VPCLIP	—	Clip viewport boundary
VPFADE	VPF	Toggle viewport fade
VPSCALEBAR	VPSB	Toggle viewport scale bar
PAGESETUP	—	Open page setup for active layout
PLOT	PRINT	Open Print dialog (Ctrl+P)
PLOTALL	—	Export all layouts to PDF
PLOTPDF	PDF	Export active layout to PDF

2.5.10 Settings

Command	Alias	Description
GRID	—	Set grid spacing or ON/OFF (F7)
GRIDMAJOR	—	Set major grid interval
GRIDTEXT	—	Toggle grid coordinate labels
GRIDUNIT	—	Set grid unit spacing
GRIPSIZE	—	Set grip size (1-20)
LIMITS	—	Set drawing limits
ORTHO	—	Toggle orthogonal mode (F8)
OSMODE	—	Set OSNAP bitmask
RETAINEDGEOMETRY	RETAINED	Toggle retained batched-GPU geometry renderer (ON/OFF)
RETAINEDSTATS	RETSTATS	Print retained-geometry diagnostic counters (last paint + sample colour)
RETLOD	RETAINED-LOD	Print/tune retained-geometry LOD (RETLOD, RETLOD FACE|DENSITY|PTSIZE <n>)
SCALELISTEDIT	SLST	Edit scale list
SETVAR	SV	Set a system variable
SNAP	—	Toggle grid snap (F9)
UCSICON	—	UCS icon ON/OFF/Origin
UNITS	—	Units wizard

2.5.11 Utility

Command	Alias	Description
AUDIT	—	Check drawing integrity
DWGPROPS	DRAWING-PROPS	Show drawing statistics

continues on next page

Table 11 – continued from previous page

Command	Alias	Description
PURGE	—	Remove all unused items
PURGEBLOCKS	—	Remove unreferenced blocks
PURGELAYERS	—	Remove unused layers
PURGELINETYPES	—	Remove unused linetypes
PURGESTYLES	—	Remove unused styles
RECOVER	—	Fix integrity issues
RECOVERCREATELAY-ERS	—	Recover + recreate layers

2.5.12 Tools

Command	Alias	Description
SCRIPT	SCR	Run a script file of commands (.scr)

2.5.13 Diagnostic

Command	Alias	Description
GPUTEST	—	Test GPU/VBO capability and benchmark
ADDREALTUNNELSEC-TION	—	Add MEIS real tunnel section
ADDSAMPLEENTITIES	—	Add one of every entity type
ADDTUNNELSECTION	—	Add sample tunnel section

2.6 IQcad Pro

Pro adds the commands below. MAPBG and MAPSETTINGS also run when IQcad Civil is licensed, even if Pro is not. The other Pro commands need Pro.

Super Tools are described in *Super Tools* (page 119). Python, Pascal Script, Blade Runner and plugins are described in the *SDK* (page 295).

2.6.1 Drawing

Command	Alias	Description
EXTRUDE	EXT	Extrude a closed profile along Z
SWEEP	SW	Sweep profile along path
FLATTEN	—	Set Z=0 on selected objects
OBJVIEW	OB-JECTVIEWER	Open selected objects in the 3D Object Viewer

2.6.2 Map

Command	Alias	Description
MAPSETTINGS	—	Open map provider settings dialog
MAPBG	—	Toggle map background / set opacity / switch source / clear cache

2.6.3 Super Tools

Command	Alias	Description
BESTFITLINE	BFL	Best-fit 3D line through selected points/vertices (PCA)
DGRID	DG	Draw a coordinate grid in model space (lines or ticks)
PLGETZ	PGZ	Get Z from surface/point cloud on polylines and lines (multi-select)
PLSIMPLIFY	PLSIM	Simplify polyline vertices (Douglas–Peucker)

2.6.4 Scripting and plugins

Command	Alias	Description
BLADE	BLADERUNNER	Open the Blade Runner script editor
PASCAL	PAS	Run a Pascal Script file (.pas)
PASEVAL	—	Evaluate a Pascal Script expression
PASLOAD	—	Load a Pascal Script module (session-resident)
PASSCRIPTS	—	List registered Pascal Script commands
PLUGINS	PLUGINMANAGER	Open the Plugin Manager dialog
PYEVAL	—	Evaluate a Python expression
PYLOAD	—	Load a Python module (session-resident)
PYPIP	—	Run pip against the bundled Python (e.g. PYPIP install <pkg>)
PYSCRIPTS	—	List registered Python commands
PYTHON	PY	Run a Python script file (.py)

2.7 IQcad Civil

Civil adds survey and design commands. The Civil Toolspace, the Civil menu and the Civil ribbon tab are visible only while this module is on. The objects themselves still load and save with the drawing.

The civil tools are described in the *Civil module* (page 131).

2.7.1 Survey points

Command	Alias	Description
ADDTOGROUP	ATG	Add selected points to a group
CONVERTPOINTS	CPT	Convert AutoCAD points to civil points
CREATEGROUP	CRG	Create a new point group
POINTCHAINAGE	PCH	Point chainage/offset vs alignment
REMOVEFROMGROUP	RFG	Remove selected points from a group
SURVEYPOINTSTYLE	SPS	Edit survey point styles
SURVEYPOINT	SRVPT	Place a survey point

2.7.2 Surfaces

Command	Alias	Description
ADDCONTOURLABEL	ACL	Place contour labels on surface
ADDSURFACEBOUND-ARY	ASB	Add boundary to surface
ADDSURFACEENTI-TIES	ASE	Add selected entities to surface
ADDSURFACEPOINTS	ASP	Add survey points to surface
BUILDGRADING	BG	Rebuild a stale grading
CREATESURFACE	CS	Create surface
DECIMATESURFACE	DS	Decimate surface TIN
DELETESURFACE	—	Delete a surface
DEMTOTIN	D2T	Resample a DEM into a new editable TIN surface
ELEVTRACK	—	Toggle live surface elevation readout under the cursor
EXPLODECONTOURS	EXC	Explode a surface's contours into polyline entities
IMPORTDEM	IDEM	Import GeoTIFF DEM as surface
MERGESURFACES	MS	Merge two surfaces (paste or union) into a new or replaced surface
REBUILDSURFACE	RS	Rebuild surface TIN
SLOPEPATH	SLP	Trace a max-slope gradient line along a sketched polyline across a surface
SPOTELEV	SE	Add spot elevation to surface
SURFACEPROPERTIES	—	Show surface properties
SURFACESTYLE	SUS	Edit surface styles
SURFADDBREAKLINE	SAB	Add selected polyline as breakline
SURFADDLINE	SAL	Force edge between two points
SURFADDPOINT	SAP	Add a point to surface TIN
SURFBALANCE	BAL-ANCEVOL-UMES	Edit a design surface so cut/fill against existing ground hits a target volume
SURFBUILDSETTINGS	SBS	Edit per-surface build settings (algorithm + cleanup filters)
SURFCLEAREDITS	SCE	Clear surface edit log
SURFDELETELINE	SDL	Remove a TIN edge
SURFDELETE-LONGEDGES	SDLE	Delete TIN edges longer than a threshold

continues on next page

Table 19 – continued from previous page

Command	Alias	Description
SURFDELETEPOINT	SDP	Delete a point from surface
SURFDIFF	—	Compare two TIN surfaces and insert deviation map
SURFHIDETRI	SHT	Hide a TIN triangle
SURFLISTEDITS	SLE	List surface edit operations
SURFMODIFYELEV	SME	Modify point elevation
SURFMOVEPOINT	SMP	Move a surface TIN point
SURFRAISELOWER	SRL	Raise/lower entire surface
SURFREGRID	SRG	Resample surface onto a regular grid and rebuild as Gridded TIN
SURFSHOWTRI	SWT	Show hidden TIN triangles
SURFSWAPEDGE	SSE	Swap a TIN edge
SURFUNDO	—	Undo surface edits
VOLUMEREPORT	VOLREPORT	Earthwork volume report for a comparison (diff) surface

2.7.3 Alignments and profiles

Command	Alias	Description
ADDPI	API	Add a horizontal PI to an alignment (click)
ADDPVI	APV	Add a design-profile PVI in a profile view (click)
ALIGNFROMPOLYLINE	AFP	Create alignment from polyline
ALIGNMENT	ALG	Create alignment from PIs
ALIGNMENTREPORT	ALR	Alignment geometry report
ALIGNMENTSTYLE	ALS	Edit alignment styles
ALIGNPROPS	—	Show alignment properties
ALIGNREVERSE	—	Reverse alignment direction
CREATEPROFILEVIEW	CPV	Create profile view from alignment
DELPI	DPI	Delete a horizontal PI from an alignment (click)
DELPVI	DPV	Delete a design-profile PVI in a profile view (click)
DESIGNPROFILE	DP	Create design profile in view
EDITPI	EPI	Edit curve radius/spirals of an alignment PI (click)
PROFILEPROPS	—	Show profile view properties
QUICKPROFILE	QP	Quick profile along line/polyline

2.7.4 Sections and volumes

Command	Alias	Description
CREATESECTION-SHEET	CSS	Create cross-section sheet
CREATESECTION-SHEETDLG	CSSD	Create cross-section sheet (dialog)
EARTHWORKVOLUMES	EWV	Compute earthwork volumes

2.7.5 Assemblies and corridors

Command	Alias	Description
ASSEMBLIES	—	Open the Assembly Editor
BUILDCORRIDOR	BC	Build corridor for an alignment
CORRIDORAUTO	CA	Toggle corridor AutoRebuild (live update on alignment/surface edits)
SAMPLEASSEMBLIES	SAS	Seed library with 3 sample road assemblies

2.7.6 Grading

Command	Alias	Description
GRADING	GRD	Create/build a grading (pad) from a footprint polyline + target surface
GRADINGBALANCE	GRDB	Solve a grading's design elevation for a target net cut/fill volume
GRADINGEXTRACT	GRDX	Extract a grading's current TIN into a new standalone surface

2.7.7 Culverts

Command	Alias	Description
CULVERT	CULV	Create parametric box culvert from flow-line polyline
CULVERTEDIT	CULVEDIT	Edit a selected culvert in the parameter dialog
CULVERTQUANTITIES	CULVQTY	Report culvert volumes and place a quantities table
CULVERTREPORT	CULVRPT	Culvert PDF report with sketches, volumes and stake-out points
CULVERTSECTION	CULVSEC	Cut a culvert into a dimensioned 2D section

2.7.8 Parcels

Command	Alias	Description
CREATEPARCEL	PARCEL	Create parcel from polyline
PARCELCHECK	PCHK	Check and fix parcels (duplicates, vertex names)
PARCELREPORT	PREP	Parcel HTML report with sketch
PARCELTABLE	PTAB	Place parcel coordinate/length tables

2.7.9 Transformations

Command	Alias	Description
CSTRANSFORM	CST	Coordinate system transformation using common points
RUBBERSHEET	RBS	Rubber-sheet transformation using control points

2.7.10 Tunnel

Command	Alias	Description
TSOPTIMIZE	TSOPT	Optimize tunnel section line fit
TSLINE	TSL	Draw a tunnel section line
TSLADDPOINT	TSLADD	Add points to tunnel section line
TSLCLOSEINVERT	TSLINV	Add invert points near axis to close section line
TSLCLOSETOLEVEL	TSLCLVL	Extend section line ends to target elevation
TSLDELETEPOINT	TSLDEL	Delete points from tunnel section line
TSLSORT	—	Sort section line points radially CW from -Y

2.7.11 Convert

Command	Alias	Description
ADDDXFFACES	ADF	Add 3DFACEs from DXF to surface
FACE2SURFACE	F2S	Convert 3DFACEs to surface
MESH2SURFACE	M2S	Convert polyface mesh to surface
SURFACE2MESH	S2M	Convert surface to polyface mesh

2.8 IQcad Point Cloud

Point-cloud commands belong to this module, including the ones that build a surface or a section sheet from a cloud. **Point Cloud > Edit Point Cloud** (ribbon: **Edit Cloud**) opens the point-cloud cleaner. The ribbon tip calls it PCEDIT. It is not a name you type on the command line, and the menu is hidden when this module is off.

Import, display, sections and the cleaner are described in the *Point Cloud module* (page 231).

Command	Alias	Description
ATTACHPOINTCLOUD	APC	Attach existing .iqpc cache
CLOUDSURFACE	CLS	Build a grid surface (min-Z) from a point cloud for DEM comparison
EXPORTPOINTCLOUD	EPC	Export point cloud to file
IMPORTPOINTCLOUD	IPC	Import point cloud file
PCCLIP	—	Elevation clip for point cloud
PCDIFF	—	Compare TIN surface vs point cloud and insert deviation map
PCELEVTRACK	—	Toggle live point cloud elevation readout under the cursor

continues on next page

Table 29 – continued from previous page

Command	Alias	Description
PCGROUND	PCG	Slope-based ground classifier for point cloud
PCMERGE	PCM	Merge two or more point clouds into one new cloud
PCSECTIONS	PCSS	Point cloud sections along alignment
PCSECTIONSHEET-STYLES	PCSSS	Edit point-cloud section sheet styles
PCSURFACE	PCSURF	Build a surface from a point cloud (filter by classification, voxel decimate)
PCSECTION	PCS	Create a point-cloud section
PCSECTIONCOMMIT	PCSC	Commit traced section to drawing
PCSECTIONTRACE	PCST	Run line trace on a point-cloud section
PCSECTIONVIEW	PCSV	Open the floating viewer for a section

2.9 IQcad Drone

Drone Survey is its own module. The Drone menu, the Drone ribbon tab and DRONESURVEY need it. A Point Cloud license does not include it.

The workspace is described in *Drone Survey* (page 249).

Command	Alias	Description
DRONESURVEY	DSV	Open the Drone Survey workspace

Part III

CAD Environment

The core drafting environment: the interface, drawings and navigation, the drawing and modification tools, selection, layers and styles, blocks, the Object Inspector, the command line, format settings, file import/export, DWG compatibility, sheet layout and printing, and the application settings.

3 Interface Overview

IQcad uses a ribbon-based layout similar to modern CAD applications. All panels can be shown or hidden, and the drawing area resizes to fill the available space.

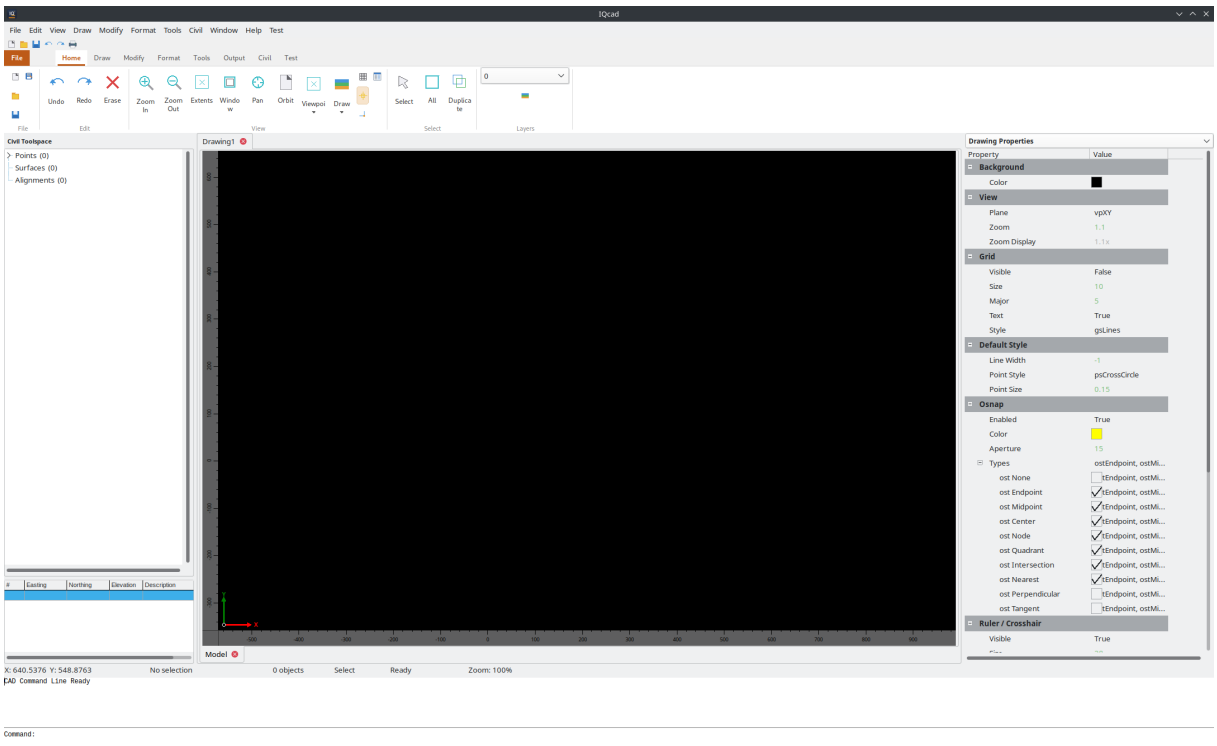


Fig. 1: IQcad main window layout.

3.1 Main Areas

Area	Description
Quick Access Toolbar	A small toolbar above the menu bar with frequently-used buttons: New, Open, Save, **Save As**, Undo, Redo, and a **Properties** toggle that shows or hides the Object Inspector panel.
Menu bar	Traditional pull-down menus: File, Edit, View, Draw, Modify, Format, Civil, Window, Help.
Ribbon	Tabbed toolbar built on the IQRibbon component with theme support. Click a tab — *Home*, *View*, *Draw*, *Modify*, *Annotate*, *Civil*, *Output*, *Help* — to access tools grouped by function.
Civil Toolspace	Left panel (toggle with **Civil › Civil Toolspace**). Displays Survey Points, Tunnel Section Lines, Surfaces, and Alignments in a tree view with a property grid below. Double-click a leaf node to select and zoom to that object in the canvas.
Drawing Canvas	The main work area. Each open drawing occupies a tab above the canvas. Rulers along the top and left edges show world coordinates.
Object Inspector	Right panel (toggle with the **Properties** button on the QAT). Displays and edits the properties of the selected object(s), the active layer, or drawing-level settings. Uses a tree-based property grid with collapsible categories. All fields are live — changes apply immediately. See *Object Inspector* (page 63) for full details.
Command line	Dark strip at the bottom. Shows the current operation, prompts for coordinate or text input, and accepts typed CAD commands (see *Command Line* (page 75)).
Status bar	Shows the active tool name and the cursor position in world coordinates.

3.2 Layout Persistence

Panel sizes are saved to `settings.json` under a `"layout"` section and restored automatically when IQcad starts:

- Prospector panel width
- Object Inspector panel width and column split position
- Command line height

3.3 Ribbon Tabs

Tab	Contents
Home	File (New, Open, Save, Save As, Close, Close All); Edit (Undo, Redo); Clipboard (Cut, Copy, Paste, Paste Orig).
View	Navigate (Zoom In/Out/Extents/Window, Pan, Orbit, Viewpoint, Draw Order); Display (Grid, Osnap, Ortho toggles, Properties panel); Select (Select, Select All, Duplicate).
Draw	All drawing tools: Line, Polyline, Multiline, Arc, Circle, Ellipse, Spline, Rectangle, Polygon, Hatch, Text, Multileader, Arrow, Point, XLine, 3D Face, 3D Polyline, Bézier, Table, Image, and more.
Modify	Erase, Move, Copy, Rotate, Scale, Offset, Trim, Extend, Chamfer, Fillet, Explode.
Annotate	Dimension tools (Linear, Aligned, Angular, Radius, Diameter, Ordinate, Arc Length, Baseline, Continue, Jogged), Dimension Styles.
Civil	Survey Point, Tunnel Section Line, Create Surface, Decimate, Spot Elevation, Alignment, Parcel, Civil Toolspace toggle, Point Import, Point Manager, Point Groups, LandXML import/export, Coordinate Transformations (CSTRANSFORM, RUBBERSHEET), Civil Style Manager.
Output	Export buttons (DXF, DWG, Image, SVG, LandXML, GeoJSON, KML, Shapefile, GPX), Import buttons (DWG, LandXML, GeoJSON, KML, Shapefile, GPX), Print, PDF, Page Setup.
Help	User manual, license, Get Support, and the About dialog.

4 Drawings

IQcad supports multiple drawings open simultaneously, each in its own tab.

4.1 New Drawing

Use **File › New Drawing** or press **Ctrl+N**. A new tab opens with an empty canvas ready for drawing.

4.2 Open a Drawing

Use **File › Open…** or press **Ctrl+O**. IQcad supports two native formats:

- `.iqz` — compressed format (gzip-compressed IQML). This is the **default format** for new drawings.
- `.iqml` — uncompressed XML-based format.

IQcad auto-detects both formats on open (gzip magic bytes are checked first). After loading, the view zooms to fit all objects.

> **Note**
>
> IQcad can also open DXF and DWG files (**File › Open…** — select the appropriate file-type filter) and import LandXML files (**File › Import LandXML…**).

4.3 Saving

Command	Description
File › Save (Ctrl+S)	Saves to the current file name in IQZ format (compressed) by default. The tab title loses the * modified indicator.
File › Save As… (Ctrl+Shift+S)	Prompts for a new file name and saves. The file-type filter lets you choose between `.iqz` (compressed) and `.iqml` (uncompressed). Also available from the **Save As** button on the Quick Access Toolbar.

The tab title shows an asterisk (`Drawing1*`) when the drawing has unsaved changes.

4.4 Drawing Properties

File › Drawing Utilities › Properties… opens a read-only summary of the current drawing as a collapsible tree:

- **Entities** — a count of every entity type in model space.
- **Layers** — each layer's entity count and its on/off, frozen and locked state, colour and linetype.
- **Blocks** — every block definition, its nested entity count and how many times it is inserted.
- **Styles & tables** — text, dimension, multiline and point styles, and linetypes.
- **Civil / 3D** — surfaces (with TIN point and triangle counts), DEM surfaces, alignments (length and element count), corridors, gradings and culverts.
- **Model summary** — total entity count, drawing extents, units and file information.

Use **Copy** to place the report on the clipboard, or **Save…** to write it to a text file. The same report can be printed to the command line with the `DWGPROPS` command.

4.5 Multiple Drawings

- Each drawing occupies a **tab** above the canvas.
- Click a tab to switch to that drawing. The Object Inspector, Layer Manager, and Civil Toolspace all update to reflect the newly active drawing.
- Close the active drawing with **Window › Close Drawing** (**Ctrl+W**) or by clicking the **×** button on the tab.
- If you close the **last open tab**, IQcad automatically opens a fresh empty drawing so there is always at least one drawing available.
- If a drawing has unsaved changes, a *Save / Discard / Cancel* prompt appears before the tab closes.

4.5.1 Drawing Tabs

```
[ Drawing1  × ]  [ Tunnel_A.iqml  × ]  [ Untitled*  × ]  [ + ]
```

The **+** button at the right end of the tab bar creates a new drawing (same as **Ctrl+N**).

5 Navigation

5.1 Zoom

Action	How
Zoom in / out	Scroll the **mouse wheel**, or press **+** / **−**.
Zoom extents	**Ctrl+Shift+E** — fits all objects in the window.
Zoom to window	**View › Zoom Window** — drag a rectangle; the view zooms to fill it.
Zoom to object	Type `ZOOMOBJECT` and pick an object; the view zooms to its extents.

5.2 Pan

Action	How
Real-time pan	Hold the **middle mouse button** and drag.
Pan from menu	**View › Pan** — click two points (displacement vector).

5.3 Grid

Toggle the background grid with **Ctrl+Shift+G** or **View › Grid**.

The grid is **off by default** for new drawings. Grid spacing is controlled by the *GridSpacing* field in the Drawing Variables tab of the Layer Manager.

5.4 Rulers

Rulers run along the **top** and **left** edges of the canvas. They always show world coordinates and update dynamically as you zoom or pan.

Note

IQcad uses a **right-hand CAD coordinate system**: X increases to the right, Y increases upward. Screen Y is inverted internally but all displayed coordinates are in world space.

5.5 3D View Commands

IQcad provides several commands for viewing the drawing from different angles:

Command	Description
3DORBIT (ORBIT)	Interactive 3D orbit mode — rotate the view around the model.
PLAN	Return to the standard top-down plan view.
VPOINT (VP)	Set the viewpoint to a preset direction.
VIEWTWIST (VT)	Set a 2D view rotation angle.
ALIGNVIEW (AV)	Align the view to two picked points on the canvas.
PLANVIEW	Reset the view twist back to north-up (0° rotation).

5.5.1 3D Behaviour

When the view is rotated away from the standard plan view (using 3DORBIT, VPOINT, or the UCS Gizmo), several canvas behaviours adapt automatically:

- **Zoom at mouse point** — the scroll wheel zooms toward the 3D point under the cursor, keeping the area of interest centred.
- **3D pan** — middle-mouse-button dragging pans along the current view plane rather than the XY plane.
- **3D crosshair** — the crosshair cursor draws three coloured axes (**red** = X, **green** = Y, **blue** = Z) to help orient the view.
- **Orbit around viewport center** — the camera orbits around the current viewport center rather than the world origin, so 3D navigation stays focused on the area you are working in regardless of the drawing's coordinate location.
- **Rulers auto-hide** — rulers along the canvas edges are hidden automatically in 3D mode, since world-axis labels are not meaningful for oblique views. They reappear when the view returns to plan.

5.6 UCS Gizmo

The **UCS Gizmo** is an interactive 3D orientation widget displayed in a corner of the canvas. It replaces the traditional flat UCS icon and provides both a visual reference for the current viewing direction and direct mouse interaction for navigating 3D views.

5.6.1 Using the Gizmo

Action	Result
Click an axis label (X, Y, or Z)	Snaps the view so that the selected axis points toward you. For example, clicking **Z** returns to the standard top-down plan view; clicking **X** shows a YZ elevation view.
Click a face label (TOP, BOTTOM, FRONT, BACK, LEFT, RIGHT)	Snaps the view to the corresponding orthographic view with a smooth 300 ms animation. Labels are rendered in Blender style on the visible faces of the gizmo cube.
Click a quadrant face	Snaps to the corresponding isometric view (e.g. top-front-right).
Drag a rotation ring	Constrains rotation to a single axis (see *Rotation Rings* below).
Drag on the gizmo (outside a ring)	Free-orbit the view — the model rotates interactively as you drag, the same as 3DORBIT but initiated from the gizmo area.

5.6.2 Rotation Rings

Three coloured rings surround the gizmo sphere, one per axis:

- **Red ring** — X axis (drag to change elevation)
- **Green ring** — Y axis (drag to change azimuth)
- **Blue ring** — Z axis (drag to change azimuth)

Each ring is projected onto the screen as an ellipse. The front half of the ring draws fully opaque over the sphere; the back half draws behind it, giving a clear sense of 3D orientation. Hovering over a ring highlights it to confirm which axis will be constrained before you begin dragging.

While dragging a ring, an **arc feedback overlay** shows the rotation angle in degrees, so you can see exactly how far you have rotated.

When a ring is viewed nearly edge-on (i.e. the axis points almost directly at the camera), the ring becomes too thin to click reliably. In this case the gizmo falls back to **free orbit** so you are never stuck without a way to rotate.

> **Note**
>
> The gizmo and its rotation rings are only shown on **Model** tabs. On layout tabs the gizmo is hidden, since layouts represent fixed paper space.

5.6.3 Gizmo Settings

The gizmo appearance can be adjusted in **Settings > UCS Gizmo**:

Setting	Description
Corner	Which corner of the canvas the gizmo occupies (default: top-right).
Size	Diameter of the gizmo in pixels (default: 120).
Opacity	Transparency of the gizmo background (0 = fully transparent, 255 = fully opaque; default: 200).

5.7 Map Background

IQcad supports background map imagery from **XYZ tile servers** and **OGC Web Map Service (WMS)** endpoints. The map layer renders behind the drawing canvas, so you can overlay your design on street maps, satellite imagery, or cadastral orthophotos.

5.7.1 Prerequisites

The map layer needs to know where the drawing is located on Earth. Before enabling the map you must assign a **coordinate system** to the drawing:

1. Open **Civil Settings** (Civil tab on the ribbon, or command `CIVILSETTINGS`).
2. Go to the **Coordinate System** section.
3. Browse or search for the correct EPSG code (e.g. EPSG:2100 for GGRS87 / Greek Grid).
4. Click **OK**.

Without a valid coordinate system the map cannot project tiles to the correct position and the background will remain blank.

Note

See *Civil Toolspace and Settings* (page 133) for full details on the Civil Settings dialog and coordinate-system selection.

5.7.2 Built-in Providers

Three providers are included out of the box:

Provider	Type	Description
OpenStreetMap	XYZ	Standard OpenStreetMap street-map tile layer.
ESRI World Imagery	XYZ	High-resolution satellite imagery (JPEG tiles).
Greece Ktimatologio Orthophoto	WMS	Greek cadastral orthophoto from the Ktimatologio WMS service.

5.7.3 Map Settings Dialog

Open the Map Settings dialog using any of the following:

- Ribbon: **Civil** tab > **Configure Map Sources** button.
- Menu: **View > Map Background > Configure Sources**.
- Command line: `MAPSETTINGS`.

The dialog is divided into several sections:

Provider list — shows all available providers. The currently active provider is marked with a bullet (*). Use the **Set Active** button to choose which provider supplies the background tiles.

Display section — controls map visibility and transparency:

- **Show Map** checkbox — toggles the map layer on and off.

- **Opacity** slider (0–100 %) — sets the transparency of the map behind the drawing. Lower values make the map more transparent so entities stand out.

Provider details — shown for the selected provider in the list:

Field	Description
Name	Display name of the provider.
Type	`XYZ Tiles` or `WMS`. Determines how tile URLs are constructed.
URL Template (XYZ) / **Base URL** (WMS)	The server URL. For XYZ, includes `{z}`, `{x}`, `{y}` placeholders. For WMS, just the service endpoint (no query parameters).
Attribution	Copyright text drawn in the corner of the map overlay.
API Key	Optional key appended to requests for providers that require authentication.
Min Zoom / Max Zoom	Zoom-level range supported by the server (typically 0–19).
Tile Size	Pixel dimensions of each tile (usually 256).
Subdomains	Letters used with the `{s}` placeholder for load balancing (e.g. `abc`).
Projection	Tile grid projection (Web Mercator for most providers).
WMS Layers (WMS only)	Layer name from the server's `GetCapabilities` response.
WMS Format (WMS only)	Requested image format: `image/jpeg` or `image/png`.

5.7.4 Adding a Custom XYZ Provider

1. Click **Add** in the Map Settings dialog.
2. Set **Type** to `XYZ Tiles`.
3. Enter the **URL Template** with `{z}`, `{x}`, `{y}` placeholders — for example:

```
https://tile.example.com/{z}/{x}/{y}.png
```

4. If the server uses subdomain rotation, include `{s}` in the URL and fill in the **Subdomains** field (e.g. `abc` for servers like `https://{s}.tile.example.com/...`).
5. Set **Min Zoom** and **Max Zoom** to the range advertised by the server.
6. Set **Tile Size** (256 is the most common value).
7. Click **OK** to save.

5.7.5 Adding a Custom WMS Provider

1. Click **Add** in the Map Settings dialog.
2. Set **Type** to `WMS`.
3. Enter the **Base URL** — the WMS endpoint without any query parameters (e.g. `https://gis.example.com/wms`).
4. Set **WMS Layers** to the layer name from the server's `GetCapabilities` document.
5. Set **WMS Format** to `image/jpeg` or `image/png`.
6. Click **OK** to save.

IQcad automatically constructs the full `GetMap` URL by appending the required parameters (`SERVICE=WMS&REQUEST=GetMap&...`) with the BBOX computed from the tile coordinates.

5.7.6 MAPBG Command

The `MAPBG` command provides quick access to map layer controls from the command line:

Option	Description
`MAPBG`	Toggles map visibility on or off.
`MAPBG` > **Opacity**	Sets the opacity percentage (0–100).
`MAPBG` > **Source**	Switches to a different provider by name.
`MAPBG` > **ClearCache**	Clears the downloaded tile cache, forcing tiles to be re-fetched.

5.7.7 Provider Persistence

Provider definitions are saved in `settings.json` at the application level and are shared across all drawings. When an `.iqz` drawing that contains custom providers is opened, IQcad auto-merges any new providers into the application list so they are available in future sessions.

5.8 Retained Geometry (experimental)

For very large drawings — hundreds of thousands to millions of entities, such as an imported or exploded surface with 1M+ 3D faces — panning and zooming can become sluggish because the standard renderer re-projects every vertex on the CPU each frame. The **retained geometry** display mode makes these drawings responsive by batching the static wireframe into GPU buffers and projecting it on the GPU, so a change of view only updates a matrix instead of reprocessing every entity.

Two techniques keep it fast at any zoom:

- **Culling** — when you zoom in, only the buckets of geometry that overlap the view are drawn.
- **Level of detail** — geometry that is too small to show detail is drawn as a subsampled cloud of points (point-cloud style) rather than full wireframe; the wireframe returns as you zoom in.

Turn it on or off with the `RETAINEDGEOMETRY` command (`ON` / `OFF`; alias `RETAINED`). It is **off by default** and does not affect normal rendering. `RETLOD` tunes the level-of-detail appearance live (`RETLOD FACE <n>` / `DENSITY <n>` / `PTSIZE <n>`), and `RETAINEDSTATS` reports diagnostics.

> **Note**
>
> This mode is experimental and is aimed at reviewing large drawings. It currently accelerates the 2D wireframe view; text, dimensions, hatches and dashed linetypes continue to draw through the standard renderer.

6 Drawing Tools

Activate a tool from the **Draw menu**, the **Draw ribbon tab**, or by typing its command alias in the command line. Press **Escape** at any time to abandon the current operation and return to Select mode.

6.1 Object Snapping (Osnap)

While drawing, IQcad snaps the cursor to geometric points on nearby objects. A coloured marker and label appear near the cursor when a snap point is active.

Snap type	Description
Endpoint	Start or end of a line, arc, or polyline vertex.
Midpoint	Midpoint of any segment.
Centre	Centre of a circle or arc.
Node	A Point entity.
Quadrant	The 0°, 90°, 180°, and 270° points of a circle or arc.
Intersection	Where two objects cross.
Nearest	Closest point on any object.
Perpendicular	Perpendicular foot from the cursor onto a line.
Tangent	Point of tangency from the cursor to a circle or arc.

6.2 Lines and Curves

Tool	How to use
Line (L)	Click start point, click end point. Continue clicking to chain segments. Right-click or **Escape** to finish.
Polyline (PL)	Click successive vertices. Right-click or **Escape** to finish. Vertex *bulge* (arc segment) can be set afterwards in the Object Inspector.
Multiline (ML)	Click successive vertices like a polyline. Each stroke produces two or more parallel strands as defined by the current Multiline Style (see *Format Settings* (page 85)). Right-click or **Escape** to finish.
Spline (SP)	Click control points. The curve passes through each point using Catmull–Rom interpolation. Right-click or **Escape** to finish.
Arc (A)	Click three points: start, a point on the arc, end. All arc methods show a live phantom arc preview. Additional methods are available from the *Draw › Arc* submenu or the Arc dropdown button on the ribbon: **Start, Center, End** (ARC_SCE) — click start, center, endpoint. **Start, Center, Angle** (ARC_SCA) — click start, center; mouse sets endpoint. **Start, Center, Length** (ARC_SCL) — click start, center; mouse sets endpoint. **Start, End, Angle** (ARC_SEA) — click start, end; mouse sets arc shape. **Start, End, Direction** (ARC_SED) — click start, end; mouse sets arc shape. **Start, End, Radius** (ARC_SER) — click start, end; mouse sets arc shape. **Center, Start, End** (ARC_CSE) — click center, start, endpoint. **Center, Start, Angle** (ARC_CSA) — click center, start; mouse sets endpoint. **Center, Start, Length** (ARC_CSL) — click center, start; mouse sets endpoint. **Continue** (ARC_CONT) — continues from the last entity's endpoint.
Circle (C)	Click the centre, then click a second point or type the radius. Additional methods are available from the *Draw › Circle* submenu or the Circle dropdown button on the ribbon: **Center, Diameter** (CIRCLE_CD) — click center; mouse distance = diameter. **2 Points** (CIRCLE_2P) — click two diameter endpoints. **3 Points** (CIRCLE_3P) — click three points for a circumscribed circle.
Ellipse	Click the centre, drag to set the semi-major axis, then drag to set the semi-minor axis.
3D Polyline (3DP)	Click successive vertices in 3-D space (each vertex stores X, Y, Z). Right-click or **Escape** to finish. Enable **SplineFit** in the Object Inspector to render the polyline as a smooth Catmull–Rom spline through all vertices. Exports as `3DPOLYLINE` in DXF.
Bézier (BEZ)	Click control points to define a Bézier curve. Right-click or **Escape** to finish.
XLine	Click a point and drag to set the direction. Creates an infinite construction line.

6.3 Shapes and Fills

Tool	How to use
Rectangle (REC)	Click two opposite corners.
Polygon (POL)	Enter the number of sides, click the centre, then click a vertex to set the circumradius. Creates a regular polygon.
Hatch (H)	Click inside a closed area to flood-fill it. Pattern, angle, and scale are set in the Object Inspector after placement.
3D Face	Click four corner points. Renders as a filled flat face.
Wipeout (WO)	Masks an area of the drawing: a closed polygon filled with the drawing background colour, so everything beneath it is hidden. Use it to clear space for a label or a detail callout without erasing the geometry underneath. Type `WIPEOUT` (or `WO`), or use *Draw › Wipeout* on the menu, or the **Wipeout** button on the *Draw* ribbon tab under **Shapes**. The prompt offers `[Polyline/Frames]`: • **(default)** — click the polygon corners, then **right-click** to close and create the wipeout. At least three points are needed; press **Esc** to cancel. • **Polyline** (P) — converts an existing polyline into a wipeout over the same area. If a polyline is **already selected** it is used straight away; otherwise you are prompted to select one and press **Enter**. Selecting more than one polyline is refused — a wipeout has a single boundary. The polyline does **not** have to be closed: a mask is a closed polygon by definition, so an open one is closed automatically (the command says so when it does). A polyline with arc (bulged) segments **is** refused, because a wipeout boundary is straight-sided and the mask would not cover the same area. You are then asked whether to erase the source polyline (default **No**). • **Frames** (F) — turns the boundary outline of **every** wipeout in the drawing on or off. This is a drawing-wide setting, not a property of the individual entity; it is stored in the `WIPEOUTFRAME` system variable (`0` = off, the default; `1` = on) and can also be set with `SETVAR`. With frames off a wipeout is invisible against the background, which is usually what you want in a plot but makes the mask hard to find on screen — turn frames on to see and select it. Wipeouts read and write through both **DXF** and **DWG**. On import the clip boundary is transformed out of image space into world coordinates, so a wipeout authored in AutoCAD lands in the right place and at the right size.
Polyface Mesh	A collection of 3D faces grouped into a single mesh entity. Polyface meshes are typically created by importing DXF files that contain `POLYLINE`-based polyface meshes, or by using the `SURFACE2MESH` command to convert a TIN surface. Each face in the mesh stores three or four vertex indices. Select the mesh and use the Object Inspector to view vertex and face counts. Use `EXPLODE` to break the mesh into individual 3D Face entities, or `MESH2SURFACE` to convert it to a TIN surface.
Extrude (EXT)	Extrudes a closed 2D profile along the Z-axis to create a 3D polyface mesh. Select or pre-select a closed polyline, circle, ellipse, or rectangle, then specify the extrusion height. Positive height extrudes upward, negative downward. Optionally close the top and bottom with triangulated cap faces. The source profile is removed and replaced by the resulting mesh. The mesh is placed on the current layer with ByLayer color. Exports as a standard DXF polyface mesh (`POLYLINE` flag 64).
Sweep (SW)	Sweeps a closed 2D profile along a path curve to create a 3D polyface mesh. Select a profile (closed polyline, circle, ellipse, or rectangle), then select a path (open polyline, 3D polyline, arc, spline, or civil alignment). Before selecting the path, you can set options: • **Twist** (T): rotate the profile linearly from 0° to a specified angle along the path length. • **Scale** (S): taper the profile from 1.0 at the start to a specified factor at the end. • **Range** (R): for alignment paths, specify start and end chainage to sweep only

6.4 Annotation

Tool	How to use
Text (T)	Click an insertion point, type the text, press **Enter**. Font and size come from the current Text Style (see *Format Settings* (page 85)).
Multileader (MLD)	Type the label text in the prompt, then click the **arrowhead tip**, then click the **text anchor**. A horizontal dogleg connects the anchor to the label automatically.
Arrow	Click the arrowhead tip, then click the tail. Simple single-segment leader line.
Point (PO)	Click to place a marker. Appearance is controlled by the active Point Style (see *Format Settings* (page 85)).
Table (TBL)	Type `TABLE` (or `TBL`) in the command line, or use *Draw › Table* on the menu / ribbon. Click a single point to place the table; a default 3 × 3 table is created at the insertion point. The table entity supports named styles — **Standard**, **Schedule**, and **Legend** — which control title/header/data row formatting (see *Format Settings* (page 85)). After placement, select the table and use the Object Inspector to change style, row/column counts, and the title text. Double-click a cell to edit its content; cells support formulas and clipboard paste from Excel or LibreOffice Calc.

6.5 Insert

Tool	How to use
Image (IA)	Type `IMAGEATTACH` or `IA` in the command line. Enter the file path (or press **Enter** with an empty path to open a file dialog). Supported formats: **PNG**, **JPG/JPEG**, **BMP**, **TIFF**. After selecting the file, specify the **width** in world units (or press **Enter** to accept the pixel-based default) and the **rotation** angle (default 0). Click to place the image — the insertion point is the **bottom-left** corner. If an ESRI world file (`.jgw`, `.pgw`, `.bpw`, `.tfw`) exists alongside the image, the command reports the georeferencing parameters and asks: **"Use georeferenced placement? [Yes/No]"**. Press **Enter** or type **Yes** to place the image at the world-file origin automatically (no clicks needed). Type **No** to proceed with interactive placement using the world-file width and rotation as defaults.

6.5.1 Image Properties

After placing an image, select it and open the **Object Inspector** to adjust:

Property	Description
ImagePath	File path of the source image (referenced mode).
PixelSize	Pixel dimensions of the source image (read-only, "W × H").
Embedded	When True, image bytes are stored inside the drawing file. Toggle to embed or unembed the image.
WorldWidth	Width of the image in world units. Changing this automatically adjusts `WorldHeight` to preserve the pixel aspect ratio.
WorldHeight	Height in world units (read-only, computed from width + aspect ratio).
Rotation	Counter-clockwise rotation in degrees.
Opacity	0 (transparent) to 255 (fully opaque). Default 255.
ShowImage	When False, a dashed placeholder frame with the filename is displayed instead of the raster data.
ClipEnabled	Enable rectangular sub-region clipping.
ClipX, **ClipY**, **ClipW**, **ClipH**	Pixel coordinates and size of the clipping rectangle.

6.6 Dimensions

Dimension tools annotate distances, angles, and radii on the drawing. All dimension tools share the same workflow: click the definition points, then click to place the dimension line. Appearance is controlled by the current **Dimension Style** (see *Format Settings* (page 85)).

Tool	How to use
Aligned Dimension (DAL)	Click the first definition point, click the second definition point, then click to place the dimension line. The measurement shows the true distance between the two points regardless of angle.
Linear Dimension (DLI)	Click the first definition point, click the second, then click to place the dimension line. Measures the horizontal or vertical distance depending on cursor movement direction. For a rotated measurement, set `RotationAngle` in the Object Inspector.
Angular Dimension (DAN)	Click the vertex point, click the first arm endpoint, click the second arm endpoint. An arc dimension line shows the included angle in degrees.
Radius Dimension (DRA)	Click the centre of the arc or circle, then click a point on the curve. A leader line with "R" prefix annotates the radius.
Diameter Dimension (DDI)	Click the centre, then click a point on the curve. A leader line with the diameter symbol (Ø) annotates the full diameter.
Ordinate Dimension (DOR)	Click the feature point, then click the leader endpoint. Displays the absolute X or Y ordinate of the feature. Toggle X/Y with `OrdIsX` in the Object Inspector.
Arc Length Dimension	Click the arc centre, click the start point on the arc, click the end point. The measurement shows the arc length along the curve.
Baseline Dimension	Extends a series of dimensions from a shared baseline. Each measurement starts from the same first definition point.
Continue Dimension	Chains dimensions end-to-end. Each new dimension starts where the previous one ended.

6.6.1 Dimension Properties

After placing a dimension, select it and open the **Object Inspector** to adjust:

Property	Description
DimStyle	Dropdown to select a named dimension style from the style library. Controls all visual properties (colours, arrows, text size, etc.).
TextOverride	Custom text to display instead of the measured value. Use **<>** as a placeholder for the measurement (e.g. **<> mm**).
Measurement	The computed value (read-only). Updated automatically from the definition points.
Rotation	Rotation angle for linear rotated dimensions.

6.6.2 Dimension Styles

Dimension styles control the appearance of all dimension components. Open **Format › Units…** and switch to the **Dimensions** tab to edit the current style.

The Dimensions tab groups settings into five areas:

Lines — dimension line and extension line colours, weights, offsets, extension amounts, and suppression flags.

Arrows — arrowhead type (Closed Filled, Open, Dot, Arch Tick, None, and others) and size for each end of the dimension line.

Text — text colour, height, gap from the dimension line, vertical placement (centred or above), and font.

Fit — controls what happens when text and arrows do not fit between extension lines. Options: move text outside first, move arrows outside first, move both outside, or best fit.

Primary Units — number format (Decimal, Scientific, Engineering, Architectural, Fractional), decimal precision (0–8 places), angle format (Decimal Degrees, DMS, Gradians, Radians), leading/trailing zero suppression, prefix, suffix, scale factor, and rounding.

Use the style dropdown at the top of the Dimensions tab to switch between named styles. Click **Reset** to revert the current style to its defaults.

6.7 Civil Drawing Tools

These tools are also on the **Civil ribbon tab** and the **Civil menu**.

Tool	How to use
Survey Point	Click to place an AEC Survey Point. Fill in PointNumber, Northing, Easting, Elevation, and description in the Object Inspector.
Tunnel Section Line	Click successive survey points along the measured profile. Right-click to finish. Link to a theoretical polyline via *SectionLine* in the Object Inspector to enable deviation analysis.

7 Modifying Objects

Select one or more objects first, then apply a Modify command from the **Modify menu**, the **Modify ribbon tab**, or the keyboard.

7.1 Modify Commands

Command	Shortcut / Alias	Description
Erase	Del · E	Deletes all selected objects. If no objects are selected, prompts "Select objects:" — pick objects then right-click to confirm or press Escape to cancel.
Move	M	Enter a displacement as X,Y in the command line, or pick two reference points on the canvas.
Copy	CO	Creates copies of selected objects. Enter displacement the same way as Move.
Rotate	RO	Enter a rotation angle in degrees (positive = counter-clockwise).
Scale	SC	Enter a uniform scale factor. Values > 1 enlarge; values < 1 shrink.
Offset	O	Creates a parallel copy of an entity at a specified distance. Supports Line, Circle, Arc, XLine, Polyline, Ellipse, Spline, and Rectangle. The default distance is stored in OFFSETDIST. Options: **T** = through point, **E** = erase source, **U** = undo. A phantom preview shows the offset entity while picking the side.
Mirror	MI	Pick two points to define the mirror axis. Optionally delete the original.
Trim	TR	Two-phase: select cutting edges (or type ALL), then click objects to trim. Supports Line, Arc, Polyline, Ellipse, and Spline entities.
Extend	EX	Two-phase: select boundary edges, then click the end of an entity to extend it. Supports Line, Arc, Polyline, Ellipse, and Spline entities.
Chamfer	CHA	Pick two lines to chamfer. Use the D sub-option to set chamfer distances ($CHAMFERA / $CHAMFERB). The second distance defaults to the first. Zero distance trims to the intersection.
Fillet	F	Pick two lines to fillet with an arc. Use the R sub-option to set the radius ($FILLETRAD). Zero radius trims to the intersection.
Explode	(none)	Breaks compound objects into their component entities. Blocks are exploded into individual entities; multilines into separate lines; polyface meshes into individual 3D Face entities.
Break	BR	Breaks an entity between two points. Pick the object (the pick point is the first break point), then "Specify second break point or [First point]:" — type F to re-pick the first point, or @ to break at a single point with no gap. Lines, arcs and polylines split into two entities (or are shortened at an end); a circle becomes an arc; closed polylines become one open polyline. A single undo step restores everything.
Join	J	Joins lines, arcs and polylines. Collinear touching lines merge into one line; touching arcs on the same circle merge into one arc; other touching chains merge into a polyline, preserving arc segments as true bulge (curved) segments. Reports "N object(s) joined into M."
Align	AL	Aligns objects using one or two source/destination point pairs. The first pair translates; a second pair adds rotation (press Enter after the first pair to move only). Then prompts "Scale objects based on alignment points? [Yes/No] <No>". Objects picked as the destination reference are not affected.
Divide	DIV	Places point entities dividing an entity into a specified number of equal segments (2–32767). Open entities get N−1 interior points; closed entities (circles, closed polylines) get N points. Works on lines, polylines, arcs, circles, ellipses, and splines.

Note

FLATTEN (project to Z = 0) lives under **Super Tools → Cleanup**, not Modify. See *Cleanup* (page 123).

7.2 Mesh and Surface Conversion

Command	Alias	Description
Mesh2Surface	M2S	Converts a selected polyface mesh entity into a TIN surface. The mesh vertices become surface points and the mesh faces become TIN triangles. Prompts for a surface name (defaults to the mesh layer name). The original mesh entity is preserved.
Surface2Mesh	S2M	Converts a TIN surface into a polyface mesh entity on the canvas. Prompts for the surface name (auto-selects if only one surface exists). The resulting mesh can be exported to DXF as a standard polyface mesh, exploded into 3D Faces, or moved/copied like any other drawing entity.

7.3 Clipboard

IQcad supports clipboard operations for copying entities between drawings and between separate IQcad instances. Entities are serialised to the system clipboard as IQML (IQcad's XML format), so all entity types are preserved including civil objects, survey points, and their associated layers.

Command	Shortcut	Description
COPYCLIP	Ctrl+C	Copies the selected entities to the clipboard from whichever tab is active (model or a layout). Associated layers and survey points are included automatically, along with the source space and drawing scale so a later paste can auto-scale.
CUT	Ctrl+X	Copies the selected entities to the clipboard, then deletes them from the drawing, from whichever tab is active. Associated layers and survey points are included automatically, along with the source space and drawing scale so a later paste can auto-scale.
PASTECLIP	Ctrl+V	Pastes clipboard contents onto whichever tab is active (model or a layout). A ghost of the entities follows the cursor; click (or type `x,y`) to place them, Esc, a right-click, or pressing Enter on an empty command line cancels. Missing layers are created automatically. Civil point numbers are renumbered if they conflict with existing points in the target drawing. Pasting between model and a layout auto-scales by the drawing scale, so text and tables land at the size they plot at (e.g. a 1:500 drawing scales ×2 into paper mm). Viewports are never carried across the clipboard — copying leaves them out, so any other entities in the same selection paste normally and a viewport-only copy has nothing to paste.
PASTEORIG	Ctrl+Shift+V	Pastes clipboard contents at their original coordinates instead of the cursor position. Same auto-scaling as **PASTECLIP**, but non-interactive.

Tip

Because the clipboard uses the system clipboard, you can copy entities in one IQcad instance, switch to a completely separate IQcad window, and paste them there.

7.4 OSNAP in Edit Commands

MOVE, COPY, ROTATE, SCALE, and OFFSET use OSNAP when picking reference points and destinations. During the object-selection phase of these commands, OSNAP is temporarily disabled so that snapping does not interfere with object picking.

7.5 Grips

When objects are selected, **blue grip handles** appear at key points (endpoints, midpoints, centres). Drag a grip to move or reshape the object interactively without invoking a Modify command.

7.6 Undo and Redo

- **Ctrl+Z** — undo the last action.
- **Ctrl+Y** — redo.

The undo history is per-drawing and is preserved as long as the drawing tab remains open.

8 Selection

8.1 Selection Methods

Action	Result
Click an object	Selects it; deselects any previous selection.
Click empty space	Deselects all objects.
Drag **left → right**	**Window selection** — selects objects *fully inside* the rectangle.
Drag **right → left**	**Crossing selection** — selects objects that *touch or cross* the rectangle (dashed outline).
Ctrl+A	Select all objects in the drawing.
Click + **Shift**	Add to or remove from the current selection.
Click again on the **same spot**	Cycles to the next overlapping object (corridor linework on an alignment, stacked polylines, …). The command line names the next entity.
Alt+click	Cycles overlapping objects without starting a move or pan (**Ctrl+drag** still pans).
Overlap **badge + list**	When several objects sit under the pick box, a two-square badge appears at the cursor. Click opens a list (type, name, layer). Hover a row to preview; click the row to select. Toggle with **Ctrl+Shift+W** or the `SELECTIONCYCLING` sysvar (0=off, 1=badge, 2=badge+list). **Ctrl+W** still closes the drawing.
Escape	Cancels the active operation with priority: active command (OFFSET, ERASE, MOVE, etc.) → active drawing tool (polyline, circle, etc.) → deselect all. When nothing is active, deselects all objects.

8.2 Selection Feedback

- Selected objects highlight and show **blue grip handles** at key geometric points.
- The status bar shows the count of selected objects.
- The Object Inspector switches to show the properties of the selected object (or a summary if multiple objects with different types are selected).

8.3 Selection Filter

The Object Inspector includes a **Selection Filter** combo box at the top of the panel. When set to a specific entity type (e.g. *Line*, *Polyline*, *Circle*, *Survey Point*, etc.), only entities of that type can be selected on the canvas — all other entity types are ignored during selection.

Set the filter to **All** (default) to allow selection of any entity type.

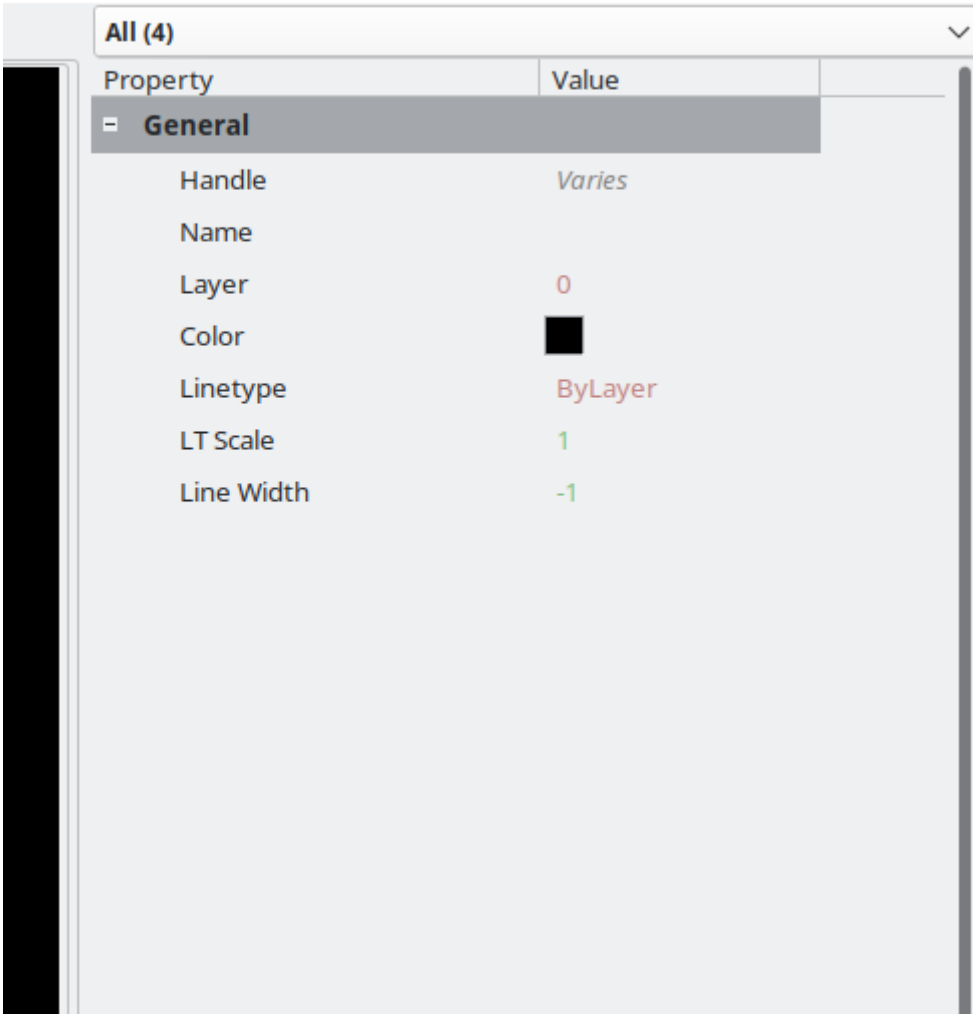


Fig. 1: The selection filter combo at the top of the Object Inspector.

Tip

To select objects on a specific layer only, first make that layer the only visible layer in the Layer Manager, then use **Ctrl+A**. Alternatively, use the selection filter to limit selection by entity type.

8.4 Cursor Modes

The cursor appearance changes to indicate the current interaction mode:

- **Full crosshair** — the default cursor, shown when idle, during drawing tools, and whenever a command prompts for a point (e.g. base point, destination, offset side). OSNAP is active in this mode.
- **Pickbox only** — shown when a command prompts for object selection (e.g. OFFSET "Select object to offset:", ERASE "Select objects:"). OSNAP is disabled in this mode so snapping does not interfere with object picking.

8.5 Right-Click Behaviour

Right-click context depends on the current state:

Context	Action
Drawing tool active (polyline with points)	Finishes the entity (closes/ends the polyline).
Drawing tool active (circle before first click)	Cancels the tool.
Command object-selection phase	Confirms the current selection (equivalent to Enter).
Civil points selected, no active tool	Opens the **point group context menu** (see below).
No active tool or command	Repeats the last command.

8.5.1 Point Group Context Menu

When one or more **civil survey points** are selected and you right-click with no active tool or command, a context menu appears instead of repeating the last command. The menu provides quick group assignment:

- **Header** — shows "Add *N* Point(s) to Group" (read-only).
- **Existing groups** — one entry per point group defined in the drawing. Click a group name to add all selected points to that group immediately.
- **New Group…** — prompts for a group name, creates the group, and adds the selected points to it.

This provides a fast alternative to typing group-management commands in the command line.

9 Layers, Linetypes and Text Styles

Open the **Layer Manager** from **Format › Layers…** (**Ctrl+Shift+L**) or the **Layers** button on the *Format* ribbon tab. Linetypes, drawing variables and text styles have their own dialogs under the same menu; they are described below and in *Format Settings* (page 85).

9.1 Layer Manager

Every object belongs to a layer. Layers control the default colour, linetype, linewidth and transparency of the objects assigned to them, and whether those objects are drawn, selectable and printed.

The Layer Manager is a modeless window: it stays open while you draw and follows the active drawing tab. The toolbar holds **New**, **Delete**, **Set Current** and a **Filter** box; typing in the box shows only the layers whose name contains the text. Layer `0` cannot be deleted.

The grid has twelve columns. The state columns toggle with a single click.

Column	Description
Cur	Marks the current layer, the one new objects are placed on. Click the cell (or press **Set Current**) to make a layer current. A frozen layer cannot be made current.
Layer Name	Layer name. Double-click the cell to rename.
On	On / Off. A layer that is off is not drawn and its objects cannot be selected, but they still take part in object snap.
Frz	Frozen / thawed. A frozen layer is excluded from display, object snap and selection altogether. The current layer cannot be frozen.
Lck	Locked / unlocked. Objects on a locked layer stay visible but cannot be selected or moved.
Color	Click the swatch to change the layer's default colour.
Linetype	Click to pick a linetype from the loaded set.
LW	Default linewidth for objects on the layer.
Transp	Transparency, 0 (opaque) to 90. Stored and written to DXF/DWG; on-screen blending is not applied yet.
Plot Style	Named plot style used when printing and exporting (`Normal` by default). See *Layouts, Viewports and Printing* (page 105).
Plot	Whether the layer is included in PDF and DXF/DWG plot output.
Description	Free text.

9.2 Ribbon Layer Dropdown

The **Layer** dropdown on the *Home* ribbon tab shows the current layer with its colour swatch; choosing another entry makes that layer current. Every row of the open list also carries three state icons in front of the name, in the style of the AutoCAD layer control:

- a **bulb** for On / Off,
- a **sun / snowflake** for thawed / frozen,
- a **padlock** for unlocked / locked.

Click an icon to toggle that state. The list stays open, so several layers can be switched in one visit. Freezing the current layer is refused and the reason is shown in the status bar. Changes take effect on the canvas at once and are mirrored in the Layer Manager if it is open.

9.3 Linetype Manager

Format › Linetypes… (or the **Linetypes** ribbon button) opens the modeless Linetype Manager. It lists every linetype loaded in the drawing in five columns: **Name**, **Description**, a **Preview** strip of the pattern, **Source** (built-in, library or file) and **Complex** (patterns that embed text or shapes).

Button	Action
Load from File	Opens a file browser to load a standard AutoCAD `.LIN` linetype definition file. All linetype entries in the file are added to the library.
Assign to Current Layer	Assigns the selected linetype to the currently active layer.
Library	Adds every linetype of the program's linetype library that the drawing does not have yet.

Custom linetype names can also be set per-object in the Object Inspector (*LineType* field).

Built-in linetypes: **Solid**, **Dashed**, **Dotted**, **DashDot**, **DashDotDot**, **LongDash**, **ShortDash**, **Divide**, **Center**.

9.4 Drawing Variables

Format › Variables Editor… (or the **Draw Vars** button on the *Format* ribbon tab) opens the Drawing Variables form, which shows and edits global settings that apply to the whole file. Units and precision also have their own dialog under **Format › Units…** (**Ctrl+Shift+U**).

Variable	Description
LtScale	Global linetype scale factor. Increase to stretch dash patterns; decrease to compress them. Command: `LTSCALE`.
LUnits	Linear unit type (Decimal, Engineering, Architectural, …).
LUprec	Linear precision (decimal places).
AUnits	Angular unit type (Degrees, Radians, Grads, …).
AUprec	Angular precision.
CurrentTextStyle-Name	The active text style used for new Text and Multileader objects.

9.5 Text Styles

Format › Text Style… (**Ctrl+Shift+T**) lists all named text styles with eight columns: **Name**, **Font**, **Height**, **Width Factor**, **Oblique**, **Bold**, **Italic**, **Description**.

Five built-in styles are always present:

Name	Font	Height	WFactor	Oblique	Bold	Italic	Notes
Standard	Arial	0	1.0	0°	No	No	Default for all new text
Title	Arial	0	1.0	0°	Yes	No	Headings
Subtitle	Arial	0	1.0	0°	No	Yes	Captions
Annotative	Arial	0	1.0	0°	No	No	Scale-aware annotation
Mono	Courier New	0	1.0	0°	No	No	Fixed-width labels

Height = 0 means the height is set individually per text object.

To edit or add styles use **Format › Text Style…** (see *Format Settings* (page 85)).

9.6 Colour and Linetype Inheritance

An object's colour and linetype can be set to **ByLayer** or to an explicit value:

Value	Meaning
ByLayer	The object inherits the colour / linetype of its layer. This is the recommended default.
Explicit value	The object always draws in that colour / linetype regardless of the layer setting.

10 Object Inspector

The Object Inspector is the right-side panel that displays and edits properties of the selected object(s), the active layer, or drawing-level settings. Toggle its visibility with the **Properties** button on the Quick Access Toolbar.

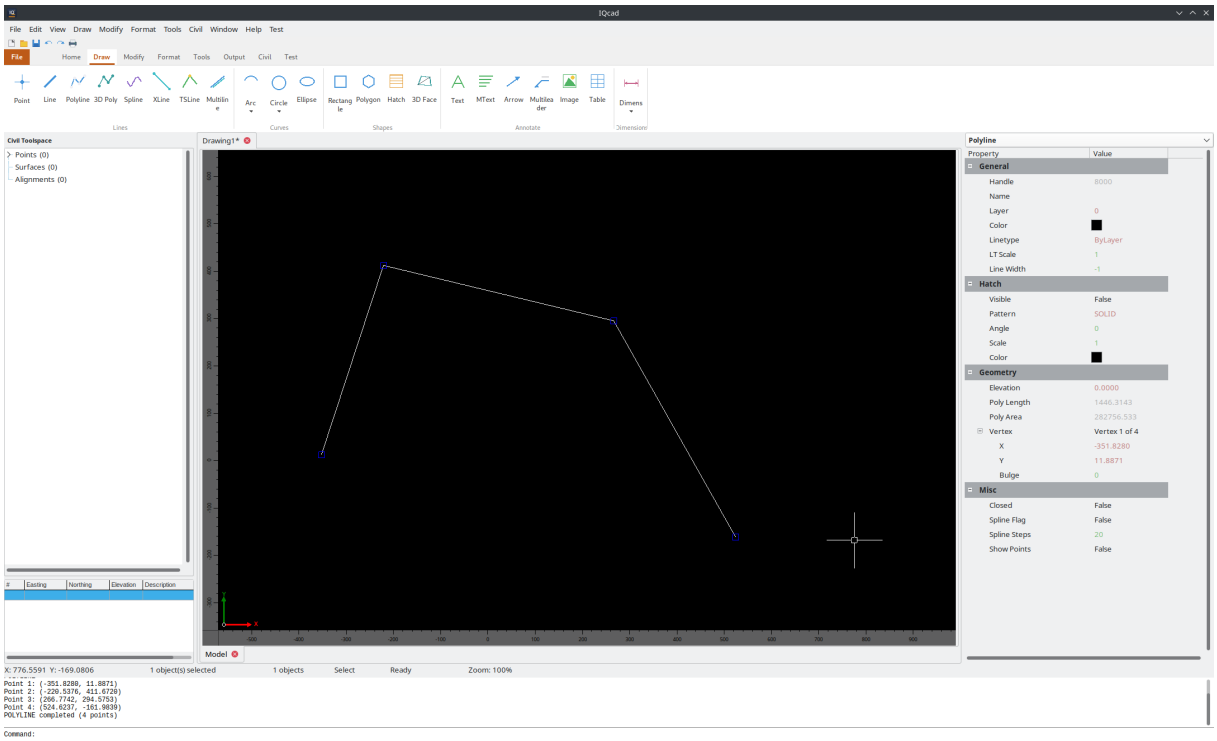


Fig. 1: The redesigned Object Inspector with collapsible categories.

10.1 Architecture

The inspector is built on **TVirtualStringTree** — a high-performance tree control that renders property rows as tree nodes organised into collapsible **category groups**. This replaces the earlier custom panel implementation.

Key design features:

- **Collapsible categories** — click a category header to expand or collapse its properties.

- **Owner-drawn combo editors** — Color properties show an inline colour swatch (no text). Layer combos show colour swatch + name. Linetype combos show a dash pattern preview. Lineweight combos show a thickness preview line.
- **Value type colour-coding** — numeric values are shown in green, strings in orange, read-only values in grey, and multi-select differing values in dimmed italic ("Varies").
- **Keyboard navigation** — **Tab / Shift+Tab** cycles through editable rows; **Enter** commits a value; **Escape** reverts.
- **Human-friendly labels** — PascalCase property names are automatically converted to spaced labels (e.g. `BackgroundColor` becomes "Background Color"). Proxies can override labels for context-aware naming (e.g. `ShowScaleBar` becomes "Show" under the Scale Bar group).

10.2 Selection Filter

At the top of the inspector panel is a **Selection Filter** combo box. Set it to a specific entity type (e.g. *Line*, *Polyline*, *Circle*, *Survey Point*) to restrict which entity types can be picked on the canvas. Set it to **All** (default) to allow selection of any entity type.

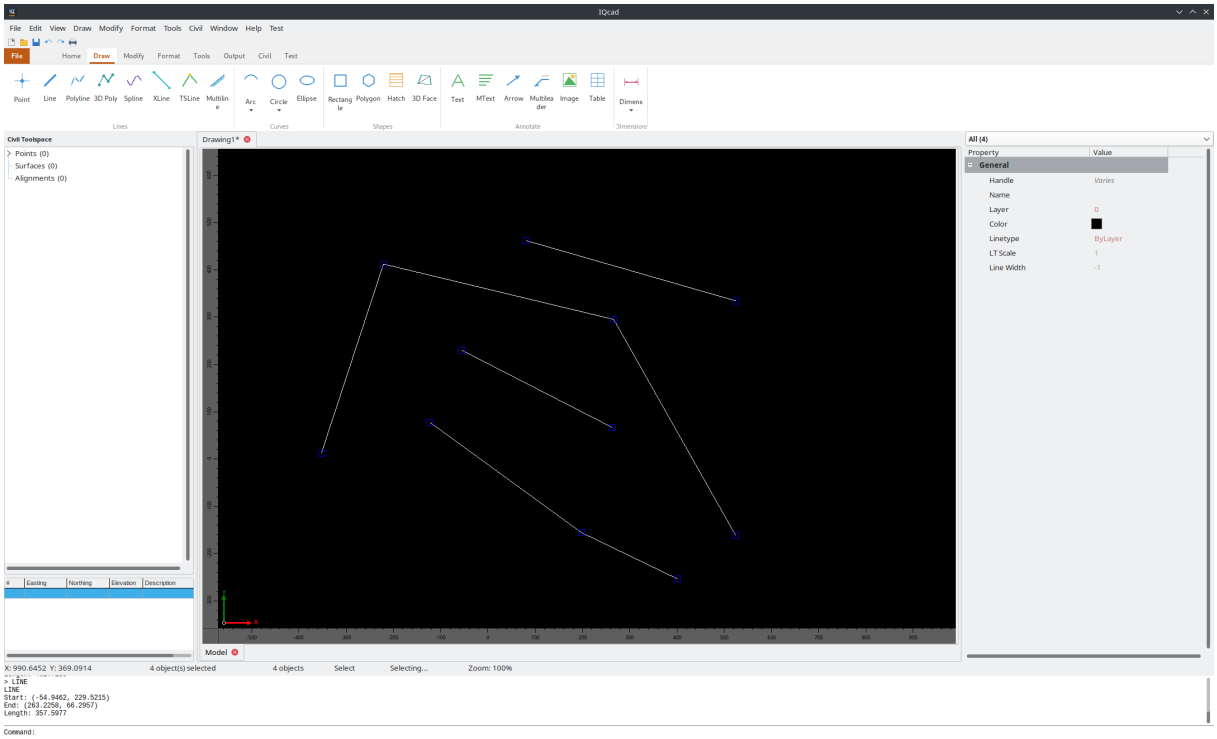


Fig. 2: Selection filter combo restricting picks to a specific entity type.

10.3 Multi-Selection

Select multiple objects (window, crossing, Ctrl+click, or Ctrl+A) to inspect them all at once. The inspector shows properties shared by all selected entities:

- Properties with the **same value** across all entities display that value normally.
- Properties with **different values** display as italic *Varies*.
- Editing a value applies the change to **all** selected entities simultaneously.

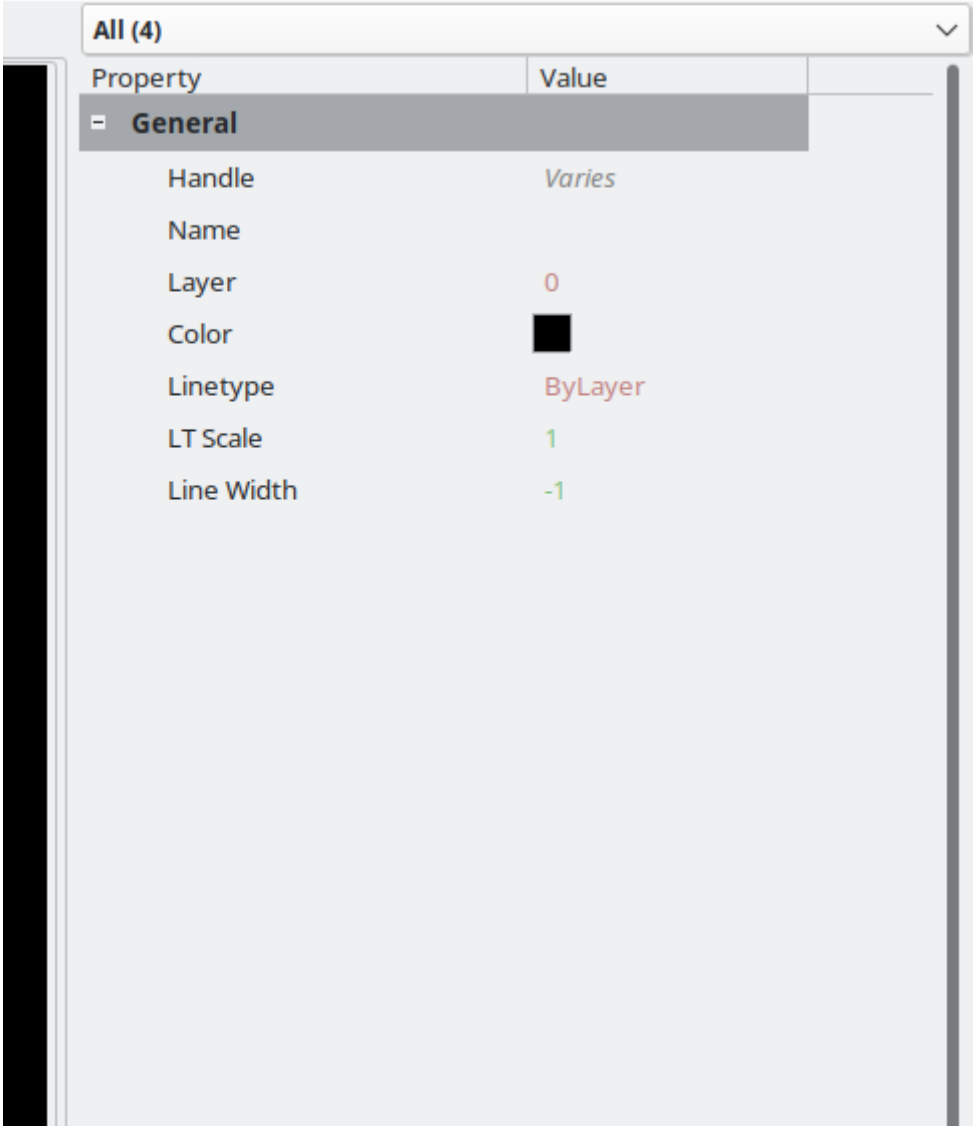


Fig. 3: Multi-selection showing “Varies” for differing property values.

10.4 Common Properties (All Objects)

Property	Description
Handle	Unique numeric identifier — read-only.
Name	Optional user-assigned label for the object.
Layer	The layer this object belongs to. Owner-drawn combo shows colour swatch + layer name.
Color	Object colour. `ByLayer` inherits from the layer. Owner-drawn combo shows colour swatch only.
Line Type	`ByLayer`, a built-in type, or a custom named type loaded from a `.LIN` file. Owner-drawn combo shows a dash pattern preview.
Line Width	Pen width. Owner-drawn combo shows a thickness preview line.

10.5 Canvas / Drawing Defaults

When **no object is selected**, the inspector shows drawing-level defaults used for new objects, organised into categories:

Property	Description
Current Color	Default colour for new objects (`ByLayer` recommended).
Current Linetype	Default linetype for new objects.
Current Line Width	Default pen width for new objects.
Grid Visible	Show or hide the background grid.
Background Color	Canvas background colour.

10.6 Line Properties

Property	Description
Start Point / End Point	World coordinates of the endpoints. Edit to reposition precisely.

10.7 Polyline and Multiline Properties

Property	Description
Closed	Whether the polyline is closed (last vertex connects to first).
Vertex	Vertex navigation row with **[<]** and **[>]** buttons and a label showing **Vertex N of M** between them. Click the arrows or change the index to navigate. Sub-properties: **X**, **Y**, **Bulge** (Z is also shown for 3D polylines).
End Caps (Multiline)	When checked, perpendicular lines are drawn across both ends of the multiline.

10.8 Hatch Properties

Hatch properties are shown on polylines, rectangles, and other entities that support hatching:

Property	Description
Hatch Pattern	Pattern name combo (ANSI31, ANSI32, NET, DOTS, SOLID, etc.).
Hatch Scale	Pattern scale factor.
Hatch Angle	Pattern rotation angle in degrees.
Hatch Color	Hatch line colour.
Hatch Background Color	Fill colour behind the hatch pattern.
Hatch Transparency	0 (opaque) to 255 (fully transparent).

10.9 3D Polyline Properties

Property	Description
Closed	Whether the 3D polyline is closed.
Spline Flag	Enable Catmull-Rom spline interpolation in full 3D (X, Y, Z).
Spline Steps	Number of interpolation steps per spline segment (default 20).
Point Count	Number of vertices — read-only.
Poly Length	Total polyline length — read-only.
Vertex	Navigation row with **[<] Vertex N of M [>]**. Sub-properties: **X**, **Y**, **Z**.

10.10 Text Properties

Property	Description
Text	The string content.
Height	Text height in world units.
Style Name	Named text style (font, width factor, oblique angle, bold/italic).
Bold / Italic	Per-object override of the style's weight/style.
H Align / V Align	Horizontal and vertical justification.
Rotation	Rotation angle in degrees.

10.11 Multileader Properties

Property	Description
Text	Label text content.
Arrow Style	*Filled*, *Open*, or *None*.
Arrow Size	Arrowhead size in world units.
Landing Gap	Length of the horizontal dogleg connecting the leader to the label.
Text Height	Label text height in world units.
Style Name	Named text style for the label.

10.12 Arc / Circle Properties

Property	Description
Centre	World coordinates of the arc/circle centre.
Radius	Radius in world units.
Start Angle / End Angle (Arc)	Arc extents in degrees. 0° = east; increases counter-clockwise.

10.13 Dimension Properties

Property	Description
Dim Style	Dropdown to select a named dimension style from the style library.
Text Override	Custom text to display instead of the measured value. Use <> as a placeholder for the measurement (e.g. <> mm).
Measurement	The computed value (read-only).
Rotation	Rotation angle for linear rotated dimensions.

10.14 AEC Survey Point Properties

Property	Description
Point Number	Numeric survey identifier.
Point Name	Descriptive name.
Northing / Easting	World coordinates — kept in sync with the object's drawn position.
Elevation	Z value (stored; not used for 2D drawing position).
Raw Description	Survey field code (e.g. EP, BM).
Full Description	Expanded description for reports.
Point Group	Group name for filtering in the Civil Toolspace and exports.
Lock Point	Prevents accidental moves when checked.
Point Style Name	Named style from the Point Style Library (see *Format Settings* (page 85)).

10.15 Surface Properties

Property	Description
Style Name	Named surface style (Standard, Triangles & Contours, Contours Only, or a user-defined style). See *Format Settings* (page 85).
Point Count	Number of TIN vertices — read-only.
Triangle Count	Number of Delaunay triangles — read-only.
Surface Area	Total 3D surface area — read-only.
Min Elevation / Max Elevation	Elevation range — read-only.
Show Contours	Toggle contour line display.

10.16 Alignment Properties

Property	Description
Style Name	Named alignment style controlling centerline, ticks, PI markers, and direction arrows.
Label Style Name	Named alignment label style controlling station labels, PI data, and curve data labels.
Start Station	Station value at the beginning of the alignment.
Total Length	Total alignment length — read-only.

10.17 Table Properties

Property	Description
Insert X / Insert Y *(Geometry)*	Coordinates of the table insertion point.
Rotation *(Geometry)*	Table rotation angle in degrees.
Rows *(Table)*	Number of rows (including title and header rows if present).
Cols *(Table)*	Number of columns.
Style Name *(Table)*	Named table style (Standard, Schedule, Legend, or a user-defined style). See *Format Settings* (page 85) for style details.
Title *(Table)*	Text displayed in the title row (visible only when the style has **HasTitle** enabled).

10.17.1 Image Properties

Property	Description
Image Path	File path of the source image (referenced mode).
Pixel Size	Pixel dimensions of the source image (read-only, "W x H").
Embedded	When True, image bytes are stored inside the drawing file.
World Width / World Height	Dimensions of the image in world units.
Rotation	Counter-clockwise rotation in degrees.
Opacity	0 (transparent) to 255 (fully opaque).
Show Image	When False, a dashed placeholder frame is displayed instead.
Clip Enabled	Enable rectangular sub-region clipping.

10.18 Viewport Properties

When a paper-space viewport is selected, the inspector shows categories for:

- **View** — scale, center position, locked flag.
- **Grid** — viewport grid visibility and spacing.
- **Border** — tick marks, ladder marks, crosses with coordinate labels, border interval.

- **North Arrow** — visibility, size, position.
- **Scale Bar** — visibility, style, position.

10.19 Tunnel Section Line Properties

See *Tunnel Analysis and Coordinate Transformations* (page 181) for the full list of Tunnel Section Line deviation-analysis properties.

10.20 Set Properties

Set-type properties (e.g. `OsnapTypes`) expand **in-place** as checkbox children instead of opening a popup dialog. Check or uncheck individual items to modify the set value.

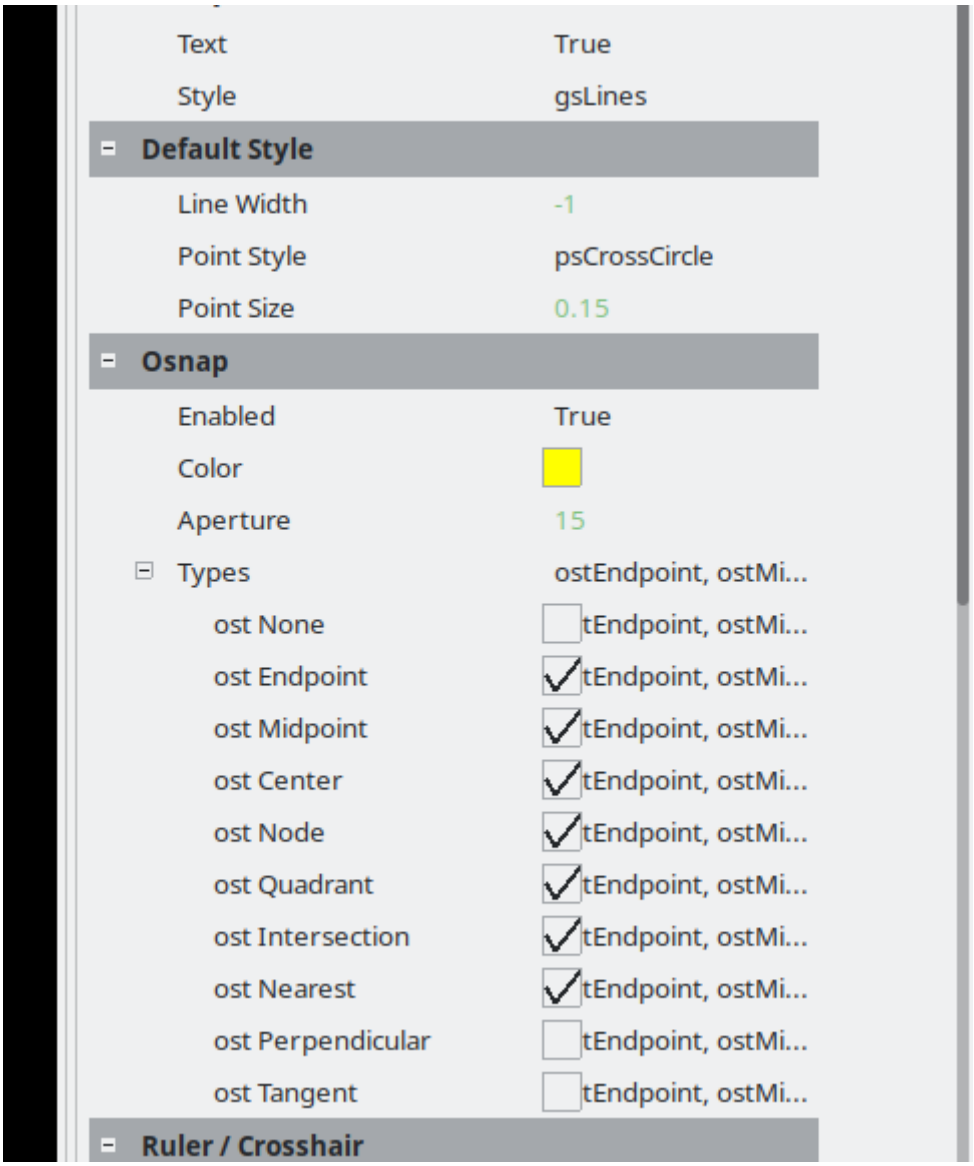


Fig. 4: A set property expanded as checkbox children.

11

Blocks and Attributes

A **block** is a named group of entities stored once in the drawing and placed any number of times as a **block reference** (an `INSERT`). Moving or editing the definition updates every reference at once, which makes blocks the right tool for repeated symbols — manholes, trees, north arrows, title blocks, standard details.

Blocks live in the drawing they are defined in. They survive a save to `.iqz`, `.dxf` and `.dwg`, and can be imported from another drawing through the Block Manager.

11.1 Creating a Block

1. Draw or select the entities that make up the symbol.
2. Type `BLOCK` (alias `B`) and press **Enter** to confirm the selection.
3. Click the **base point** — the handle the block is inserted by, and the point the reference is rotated and scaled about.
4. Type a name and press **Enter**.

The selected entities are copied into the definition (with their coordinates made relative to the base point), removed from model space, and replaced by a single block reference at the base point. Nothing is lost: exploding the reference brings the geometry back.

> **Note**
>
> Block names are unique per drawing and the check is case-insensitive. If the name is already taken, the command reports it and asks again rather than overwriting the existing definition.

11.2 Inserting a Block

- `INSERT <name>` (alias `INS`) starts a rubber-band insert of that block; click to place it.
- `INSERT` with no argument lists the defined blocks and asks for a name.
- `BLOCKS` (alias `BL`) prints the block table — name and entity count — without inserting anything.
- **Draw › Block Insert** on the menu does the same as `INSERT`.

A placed reference carries its own scale, rotation and array settings, all editable in the Object Inspector:

Property	Meaning
Block name	The definition this reference points at.
Scale X / Y / Z	Independent axis scales. Negative values mirror the reference.
Rotation	Degrees counter-clockwise about the insertion point.
Columns / Rows	Rectangular array count. A reference with more than one cell is the equivalent of AutoCAD's `MINSERT`; it draws the whole grid but stays one object.
Column / Row spacing	Centre-to-centre distance between array cells, in drawing units.

11.3 Editing a Block Definition

`BEDIT <name>` (alias `BE`) opens the **block editor**: the drawing area switches to the block's own contents, drawn about its base point at the origin. Draw, modify and erase there exactly as in model space.

`BEDIT` with no argument prompts for a name; typing `?` at that prompt lists every definition with its entity count.

`BCLOSE` (alias `BC`) saves the edited definition and returns to the drawing. **Every reference to that block redraws with the new contents.**

11.4 Block Manager

`BLOCKMANAGER` (alias `BM`) opens a dialog listing every definition in the drawing as a rendered thumbnail, with the selected block's details on the right: name, origin (base point), entity count, and the number of references placed.

Button	Action
Insert	Closes the dialog and starts an insert of the selected block.
New	Starts a `BLOCK` definition from the current selection.
Edit	Opens the selected block in the block editor (as `BEDIT`).
Rename	Renames the definition; existing references follow the new name.
Delete	Removes the definition. A block that still has references cannot be deleted — erase or explode them first.
Purge Unused	Deletes every definition with no references left in the drawing. The same clean-up the `PURGE` drawing utility performs.
Import	Reads block definitions out of another `.dxf` or `.iqz` file and adds them to this drawing. This is how a shared symbol library is moved between drawings.

11.5 Exploding a Block

`EXPLODE` replaces a selected block reference with independent copies of the definition's entities, transformed by the reference's own scale and rotation. An arrayed reference explodes to every cell. The definition itself is untouched, so other references are unaffected.

11.6 Attributes

Attributes are text fields carried by a block reference — a manhole number, a pipe diameter, the fields of a title block — so that two references of the same definition can show different text.

IQcad reads and preserves attributes on import:

- An **ATTDEF** in a block definition (the template that names the field) is staged as text carrying its attribute **tag**. A tag is a label, not drawable content, so ATTDEFs are not drawn when the block is inserted.
- An **ATTRIB** on a block reference (the filled-in value) is attached to that reference and drawn with it. Attribute values move, rotate and scale with the reference, and are included in its bounding box and in the result of `EXPLODE`.
- Attributes marked invisible are stored but not drawn.

Attribute text round-trips through DXF and DWG in both directions. IQcad has no attribute-definition command of its own: a block with attributes is authored in AutoCAD (or a compatible program) and then used here.

11.7 Blocks, Layers and Colours

Entities inside a definition keep the layer, colour and linetype they were drawn with. Entities drawn on layer `0` with colour and linetype set to **ByLayer** or **ByBlock** instead inherit from the reference — which is the standard way to build a symbol that takes on the layer it is inserted on. See *Layers, Linetypes and Text Styles* (page 59) for the inheritance rules.

11.8 Blocks in the File Formats

Format	Behaviour
IQML (`.iqz`)	Definitions and references are written in full, with attributes.
DXF	Written to the `BLOCKS` section as `BLOCK` / `ENDBLK` with `INSERT` references; read back the same way.
DWG	Read and written natively, including `MINSERT` arrays, `ATTRIB` and `ATTDEF`. See *DWG Compatibility* (page 109).
PDF / SVG / image	Flattened to their drawn geometry.

The **Drawing Utilities › Purge…** command (**File › Drawing Utilities**) removes unreferenced definitions from the drawing, along with unused layers and styles, before a save.

12 Command Line

The command line strip at the bottom of the window serves two purposes:

1. **Output** — IQcad writes operation prompts, results, and messages here.
2. **Input** — type a command alias and press **Enter** to activate a tool or run an operation.

While a tool is active, the command line shows the current prompt (e.g. *"Specify start point:"*). Type coordinates as `X,Y` and press **Enter** to supply a precise point.

12.1 Command Suggestions

As soon as you start typing at an idle `Command:` prompt, a suggestion popup opens below the input box (AutoCAD-style), listing matching commands as **name (aliases) — description**. Matches are ordered exact name/alias hits first, then name/alias prefixes, then plain substring matches, each group alphabetical and case-insensitive; up to 8 rows show at once with the rest reachable by scrolling.

- **Up / Down** — move the highlight between suggestions.
- **Enter** — run the highlighted command, or submit the typed text as-is if nothing is highlighted.
- **Tab** — fill the input with the highlighted (or first) suggestion without running it, so you can keep editing or add options.
- **Esc** — dismiss the popup first; press again to clear the input.
- **Click** a row to run that command immediately.

The popup only appears while the command line is idle — it stays out of the way during an active tool's sub-prompts (e.g. `PEDIT`'s option letters). It has no on/off setting; it is always available.

12.2 Command Reference

12.2.1 Drawing Commands

Command	Alias	Action
LINE	L	Start the Line tool.
POLYLINE	PL, PLINE	Start the Polyline tool.
MULTILINE	ML	Start the Multiline tool.
ARC	A	Start the Arc tool.
CIRCLE	C	Start the Circle tool.
ELLIPSE	EL	Start the Ellipse tool.
SPLINE	SP	Start the Spline tool.
RECTANGLE	REC	Start the Rectangle tool.
HATCH	H	Start the Hatch tool.
TEXT	T	Start the Text tool.
MLEADER	MLD	Prompt for label text, then start the Multileader tool.
POINT	P, PT	Start the Point tool.
XLINE	XL	Start the XLine (infinite construction line) tool.

12.2.2 Modify Commands

Command	Alias	Action
ERASE	E	Erase selected objects. If nothing selected, prompts "Select objects:" — pick then right-click to confirm.
MOVE	M	Move selected objects.
COPY	CO	Copy selected objects.
ROTATE	RO	Rotate selected objects.
SCALE	SC	Scale selected objects.
OFFSET	O	Offset an entity by distance or through point. Supports Line, Circle, Arc, XLine, Polyline, Ellipse, Spline, Rectangle. Options: T = through, E = erase source, U = undo. Phantom preview shown.
MIRROR	MI	Mirror selected objects.
TRIM	TR	Two-phase: select cutting edges (or ALL), then click objects to trim.
EXTEND	EX	Two-phase: select boundary edges, then click the end of a line to extend.
CHAMFER	CHA	Pick two lines to chamfer. D sub-option sets distances ($CHAMFERA / $CHAMFERB).
FILLET	F	Pick two lines to fillet. R sub-option sets radius ($FILLETRAD).
BREAK	BR	Break an entity between two points. F = re-pick first point, @ = break at a single point with no gap.
JOIN	J	Join lines, arcs and polylines that touch into a single entity.
ALIGN	AL	Align objects using 1–2 source/destination point pairs, with optional scaling.
DIVIDE	DIV	Place point entities dividing an entity into N equal segments.
MEASURE	ME	Place point entities along an entity at a fixed interval.
ARRAY	ARR	Rectangular, polar or path array of selected objects.
PEDIT	PE	Edit a polyline. Sub-options: Close/Open, Join, Width, Edit vertex (Next/Prev/Move/Insert/Delete/Straighten), Spline, Decurve, Reverse. Converts Lines, Arcs and Circles to polylines (a circle becomes a closed three-vertex polyline, start point repeated as the end point, with bulge 1 on both halves).

12.2.3 Super Tools Commands

See *Super Tools* (page 119) for full descriptions.

Command	Alias	Action
FLATTEN	—	Set Z = 0 on selected objects (or the whole drawing).
PLSIMPLIFY	PLSIM	Simplify polyline vertices (Douglas–Peucker); prompts for tolerance.
PLGETZ	PGZ	Get Z from surface/PC on polylines and lines (multi-select).
BESTFITLINE	BFL	Best-fit 3D line through selected points/vertices.

12.2.4 View Commands

Command	Alias	Action
ZOOM	Z	Sub-options: E = extents, W = window, + / - = in/out.
PAN	—	Activate real-time pan (middle-mouse drag).
REGEN	—	Force a full redraw of the canvas.

12.2.5 Settings Commands

Command	Alias	Action
UNITS	—	Opens a multi-step wizard to set linear units, precision, and angular units.
SETVAR LTSCALE	SV	Set the global linetype scale factor ($LTSCALE).
LAYERS	LA	Opens the Layer Manager dialog.
MAKELAYER	—	Create a layer from the command line. Usage: MAKELAYER name [color]. Colour is either an AutoCAD Color Index (1–255) or a named shortcut (RED / YELLOW / GREEN / CYAN / BLUE / MAGENTA / WHITE). Defaults to white when omitted. Use this for scripted layer setup; the GUI LAYERS dialog covers the rest of the management workflow.
CURRENT-LAYER	CLAYER	Set the active drawing layer. Usage: CURRENTLAYER name. Frozen layers are refused.

12.2.6 Inquiry Commands

Command	Alias	Action
LIST	LI	List selected object properties.
ID	—	Click a point — report its world X, Y, Z.
DIST	DI	Measure distance between two points.
AREA	AA	Measure area (pick points or select object).
ANGLE	AN	Measure angle (vertex + two points).

12.2.7 Surface Commands

Command	Alias	Action
CREATESURFACE	CS	Create a TIN surface. If survey points exist they are triangulated automatically; otherwise an empty surface is created for later population.
ADDSURFACEPOINTS	ASP	Add all survey points in the drawing to a named surface and build/rebuild the TIN.
ADDSURFACEENTITIES	ASE	Add selected drawing objects (3DFace, Line, Polyline, Point) to a surface.
REBUILDSURFACE	RS	Rebuild an existing surface TIN from current survey points.
ADDSURFACEBOUNDARY	ASB	Add a boundary to a surface (select a closed polyline).
DELETESURFACE	—	Delete a surface by name.
DECIMATESURFACE	DS	Decimate (simplify) a TIN surface.
SURFACEPROPERTIES	—	Display surface statistics.
SPOTELEV	SE	Place a spot elevation label on a surface.
ADDCONTOURLABEL	ACL	Place contour labels along surface contour lines.

12.2.8 Alignment Commands

Command	Alias	Action
ALIGNMENT	ALG	Create a horizontal alignment by picking PI points.
ALIGNREVERSE	—	Reverse the direction of an alignment.
ALIGNPROPS	—	Display alignment properties in the command line.
ALIGNFROMPOLYLINE	AFP	Create an alignment from a polyline.

12.2.9 Profile Commands

Command	Alias	Action
QUICKPROFILE	QP	Quick profile along a line or polyline.
CREATEPROFILEVIEW	CPV	Create profile view from alignment + surface.
DESIGNPROFILE	DP	Create design profile (vertical alignment) in profile view.
PROFILEPROPS	—	Display profile view properties.

12.2.10 Cross-Section Commands

Command	Alias	Action
CREATESECTIONSHEET	CSS	Create a cross-section sheet from alignment + surfaces (command-line wizard; retained for scripting and power users).
CREATESECTIONSHEETDLG	CSSD	Open the **Create Section Sheet** dialog — a two-tab modal GUI that gathers all parameters at once: alignment, one or more surfaces each with a per-surface line color, station interval, left/right width, optional station range (From / To), columns and rows per sheet, optional cut/fill base + comparison surfaces with cut/fill colors, and advanced display settings (vertical exaggeration, grid major/minor intervals and colors, auto-fit elevation band, area labels, title). After clicking OK the user picks an insertion point on the canvas and the sheet(s) are created. Per-surface colors, cut/fill colors, grid settings, title, and station range are saved in the `.iqz` file and survive alignment edits and resample. The `CSS` command-line wizard is retained unchanged. This command is also reached via the **Civil → Section Sheet** ribbon button.
EARTHWORKVOLUMES	EWV	Compute earthwork volume table (average end area method).

12.2.11 Coordinate Transformation Commands

Command	Alias	Action
CSTRANSFORM	CST	Global coordinate transformation (Helmert, Affine, Polynomial).
RUBBERSHEET	RBS	Non-uniform rubber-sheet transformation (Linear or Smooth TPS).

12.2.12 Layout Commands

Command	Alias	Action
LAYOUT	LO	Sub-options: N = new, D = delete, L = list, S = set active.
MVIEW	MV	Create a viewport in paper space (must be on a Layout tab).
PAGESETUP	—	Open page setup dialog for the active layout.
PLOTPDF	PDF	Export the active layout to PDF (EXPORTPDF is the older equivalent).
PLOT	PRINT	Open the Print dialog (Ctrl+P).
PLOTALL	—	Export all layouts to a multi-page PDF.

12.2.13 Scripting Commands

Command	Alias	Action
SCRIPT	SCR	Run a `.scr` text file of commands. SCRIPT with no arguments opens a file picker; SCRIPT <path> runs the file directly. See *Scripting* (page 297) for the file format, examples for each entity type, and limitations.

12.3 Entering Coordinates

When a tool is active and waiting for a point you can type a coordinate at the command line instead of clicking on the canvas. The same five forms work for every drawing tool — LINE, CIRCLE, ARC, POLYLINE, RECTANGLE, ELLIPSE, POINT, SPLINE, HATCH, and the rest:

- **Absolute 2D:** X,Y — e.g. `100,50`. Z inherits from the previously-entered point (or 0 if this is the first input).
- **Absolute 3D:** X,Y,Z — e.g. `100,50,12.5`.
- **Relative:** @dX,dY[,dZ] — e.g. `@10,0` (10 units to the right of the last point).
- **Polar relative:** @distance<angle — e.g. `@25<45` (25 units at 45° from the last point, angle in degrees).
- **Survey point:** #N — e.g. `#42`, looks up survey point number 42 in the active drawing's survey database and uses its (Easting, Northing, Elevation).

Press **Enter** or **Space** to confirm. Press **Escape** to cancel the current prompt.

The "last point" used by @-relative and @dist<angle forms is whichever world point you most recently entered into the active tool — whether by clicking on the canvas or by typing a previous coordinate. That lets you mix mouse picks with typed input freely inside a single drawing operation.

Note

IQcad's drawing tools require two world points for "circle by centre + radius point" — there is no AutoCAD-style shortcut where a bare scalar (e.g. `10`) is interpreted as a radius. To draw a circle of radius 10 centred at the origin, follow CIRCLE with `0,0` and then `10,0` (a rim point at distance 10), or `@10,0` after the centre.

12.4 Running Scripts

The SCRIPT command (alias SCR) plays a text file of commands. This is useful for repetitive setup work — applying view defaults, running a sequence of operations, or replaying a recorded series of inputs.

Invoke it in one of two ways:

- SCRIPT — opens a file picker filtered to *.scr / *.txt.
- SCRIPT <path> — loads the named file directly. Relative paths are resolved against the working directory. Quote the path if it contains spaces, e.g. SCRIPT "C:\Project Setup\boilerplate.scr".

You can also reach the same workflow from **Tools → Run Script…**.

12.4.1 File format

Plain UTF-8 text:

- Lines starting with ; are comments (leading whitespace allowed).
- **Spaces inside a line act like Enter** — the line is split into whitespace-separated tokens and each token is fed to the dispatcher in turn. Matches AutoCAD's script convention.
- **A blank line acts like a right-click** — it finishes the active drawing tool (POLYLINE / LINE chain / SPLINE / etc.). At the empty Command: prompt with no tool running, a blank line is a no-op.
- A double-quoted string ("…") is one token even if it contains spaces — useful for file paths. The surrounding quotes are stripped.
- Anything else is fed to the dispatcher exactly as if you had typed it at the prompt.

Example script (samples\scripts\draw_demo.scr):

```
; Three lines in three commands, two of them on one line each
LINE 0,0 10,0
LINE 0,5 10,5
LINE 0,10 @10,0

; Circle of radius 5 centred at (20,0)
CIRCLE 20,0 25,0

; Same circle using polar
CIRCLE 30,0 @5<0

; Polyline closed by the blank line below
PLINE 0,20 5,25 10,20 15,25 20,20

ZE
```

While a script is running, each executed line is echoed back to the command line as [script] <line> in grey so you can follow what's happening. A successful run ends with Script complete. in green.

12.4.2 Stopping a script

- **Esc** cancels both the currently-running CAD command and the script itself — the player exits and the command line returns to the normal prompt.

- The script stops automatically on the first dispatch failure (unknown command name) or unhandled exception, reporting the line number so you can find the bad line: `Script stopped at line 9 (dispatch failed).`

12.4.3 Limitations

- **Bare scalar radius / distance** — AutoCAD's `CIRCLE` / `0,0` / `10` (single number for radius) is not supported. IQcad's CIRCLE wants two world points, so use `CIRCLE 0,0 10,0` (rim point at distance 10) instead.
- **Modal dialogs** opened by a scripted command (e.g. `LAYERS`, `MAPSETTINGS`) block until you close them manually — the script pauses, then continues with the next line.
- **Layer creation** — use `MAKELAYER name [color]` to create a layer and `CURRENTLAYER name` to make it active; the `LAYERS` dialog itself is modal and blocks playback until closed manually.

Three sample scripts ship under `samples\scripts\`:

- `hello.scr` — single-shot view commands, smoke test for the SCRIPT command itself.
- `setup_view.scr` — reset the view to north-up zoomed extents.
- `draw_demo.scr` — exercises typed coordinates with LINE and CIRCLE.

13 Format Settings

All format dialogs are under the **Format** menu.

13.1 Layers

Format › Layers… opens the standalone **Layers Manager** form. See *Layers, Linetypes and Text Styles* (page 59) for full details.

13.2 Linetypes

Format › Linetypes… opens the standalone **Linetypes Manager** form. Displays all loaded linetypes with name, description, pattern preview, source, and complex flag. Toolbar buttons: *Load from File…* (import `.LIN` files), *Assign to Current Layer* and *Library* (add the program's library linetypes that the drawing lacks).

13.3 Drawing Variables

Format › Variables Editor… (or the **Draw Vars** ribbon button) opens the **Drawing Variables** form showing all per-drawing settings in a two-column grid. Units and precision have their own dialog under *Format › Units…*.

13.4 Point Style (Ctrl+Shift+D)

Opens the **Point Style Editor** — a 4 × 5 grid of 19 marker shapes. Click a cell to select the active global marker shape, and set the display size in the *Size* field below the grid.

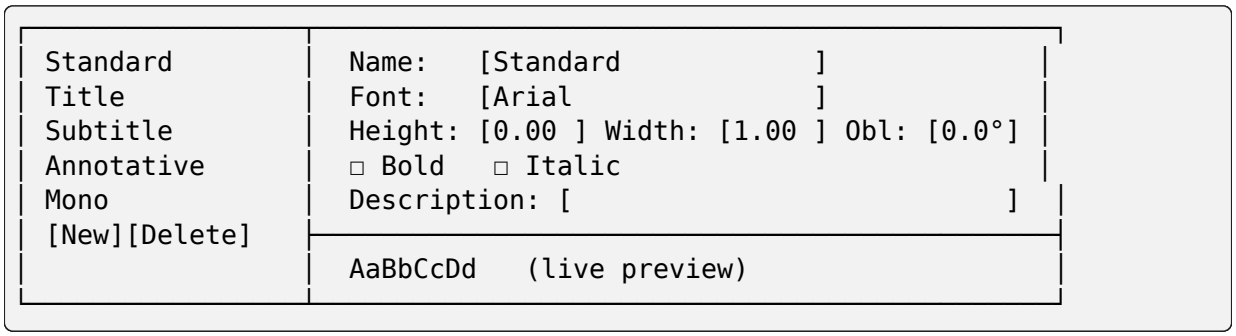
The named **Point Style Library** stores per-style settings that can be assigned to individual AEC Survey Points via the `PointStyleName` property.

Built-in style	Default marker
Standard	Cross (+)
Benchmark	Triangle with dot
Control Point	Circle with cross
GPS Rover	Square with dot
Iron Rod	Circle with vertical bar

Additional styles can be created in the editor and saved with the drawing.

13.5 Text Style (Ctrl+Shift+T)

Opens the **Text Style Editor** — a list on the left and an edit panel on the right with a live preview strip at the bottom.



Fields:

Field	Description
Name	Unique style name.
Font	Font family name (any font installed on the system).
Height	Fixed height in world units. `0` = variable (set per-object).
Width Factor	Horizontal scale. `1.0` = normal width.
Oblique Angle	Slant in degrees. Positive = forward lean.
Bold / Italic	Font weight and style.
Description	Free-text note for your reference.

The style selected in the **Drawing Variables** (*CurrentTextStyleName*) is used automatically for new Text and Multileader objects.

13.6 Surface Styles

Format > *Surface Styles…* opens the **Surface Style Editor**. A surface style bundles all visual settings for a TIN surface into one named preset: border, triangle, contour, contour label, and point display properties.

Five built-in presets are provided:

Preset	Description
Standard	Triangles + contours + boundaries + labels. All features visible.
Triangles & Contours	Triangles and contours visible, boundary hidden.
Contours Only	Only contour lines and labels displayed.
Elevation Banding	Contours and elevation colour bands visible.
Empty	All components hidden. Use to suppress a surface from display without deleting it.

The editor has a style list on the left and tabs on the right:

- **Border & Triangles** — visibility, colour, and weight for boundary polygons and TIN triangle edges; point marker settings.
- **Contours** — major/minor intervals, colours, weights, smoothing passes, base elevation.
- **Contour Labels** — visibility, text height, precision, prefix/suffix, colour, font, uphill-readable rotation, background mask, label interval.
- **Elevation Bands** — band count, color ramp, **Generate Bands from Surface** (opens a dialog to pick which surface to sample and edit the exact min/max elevation range before generating), and the **ShowLegend** toggle (displays an on-canvas legend with stacked color boxes and elevation ranges).
- **Draw Order** — intra-surface sub-element rendering priority (e.g. triangles before contours, contours before labels).

Styles are saved with the drawing. Assign a style to a surface via the **StyleName** property in the Object Inspector, or right-click a surface in the Civil Toolspace and choose **Surface Style…**.

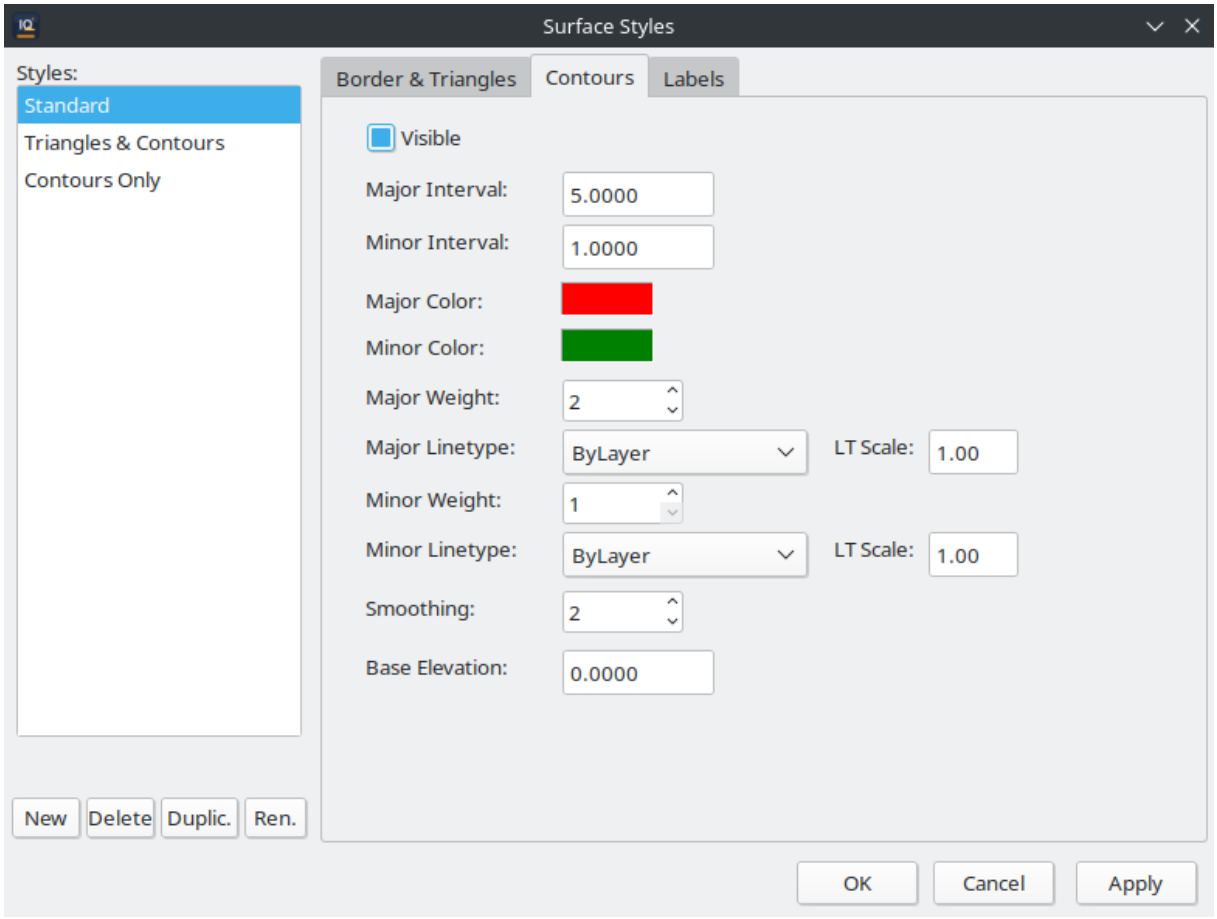


Fig. 1: The Surface Style Editor dialog.

13.7 Table Styles

Format › Table Styles… opens the **Table Style Manager**. Select a style and click **Edit** to open the **Table Style Editor**, or click **New** to create a style from scratch.

Three built-in presets are provided:

Preset	Description
Standard	General-purpose table with a bold title row, a header row, and plain data rows.
Schedule	Optimised for quantity schedules — compact data rows with alternating fill colours for readability.
Legend	Suitable for map legends — no title row, header row uses a coloured background.

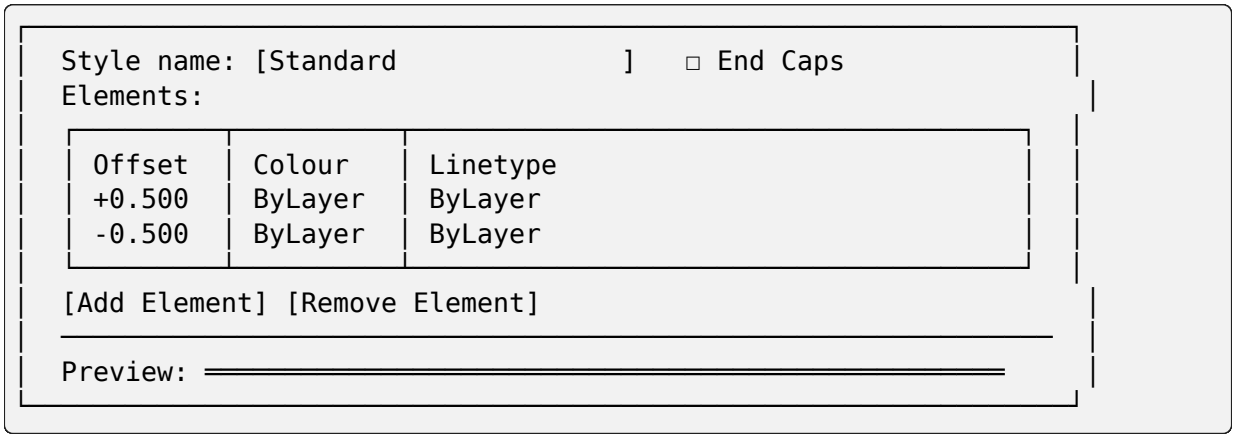
The editor organises settings into three row categories — **Title**, **Header**, and **Data** — each with its own tab. Per-category properties:

Property	Description
Text Height	Height of cell text in world units.
Alignment	Horizontal alignment: Left, Centre, or Right.
Bold / Italic	Font weight and style for the row category.
Fill Color	Background colour of the cells.
Borders	Border visibility and weight for horizontal / vertical / outline edges.
Margins	Horizontal and vertical padding inside each cell.

The **HasTitle** and **HasHeader** checkboxes at the top of the editor control which row categories appear in new tables that use the style. Unchecking **HasTitle**, for example, omits the title row entirely.

13.8 Multiline Styles (Ctrl+Shift+M)

Opens the **Multiline Style Editor**. A Multiline Style defines how many parallel strands a Multiline entity has, their spacing, colours, and linetypes.



Field	Description
Style name	Name saved with the drawing.
End Caps	When checked, perpendicular lines close the open ends of the multiline.
Offset	Distance from the centreline. Use positive and negative values to place strands symmetrically.
Colour	Per-strand colour override (`ByLayer` to inherit).
Linetype	Per-strand linetype override.
Preview strip	Live preview of the style at the bottom of the dialog.

13.9 Dimension Styles

Format › Dimension Styles… (or type `DIMSTYLE` in the command line) opens the **Dimension Style Manager**. Select a style and click **Edit** to open the **Dimension Style Editor**, or click **New** to create a style from scratch.

The editor groups settings into five areas:

- **Lines** — dimension line and extension line colours, weights, offsets, extension amounts, and suppression flags.
- **Arrows** — arrowhead type (Closed Filled, Open, Dot, Arch Tick, None, and others) and size for each end of the dimension line.
- **Text** — text colour, height, gap from the dimension line, vertical placement (centred or above), and font.
- **Fit** — controls what happens when text and arrows do not fit between extension lines.
- **Primary Units** — number format (Decimal, Scientific, Engineering, Architectural, Fractional), decimal precision, angle format, zero suppression, prefix, suffix, scale factor, and rounding.

Use the style dropdown at the top to switch between named styles. Click **Reset** to revert the current style to its defaults.

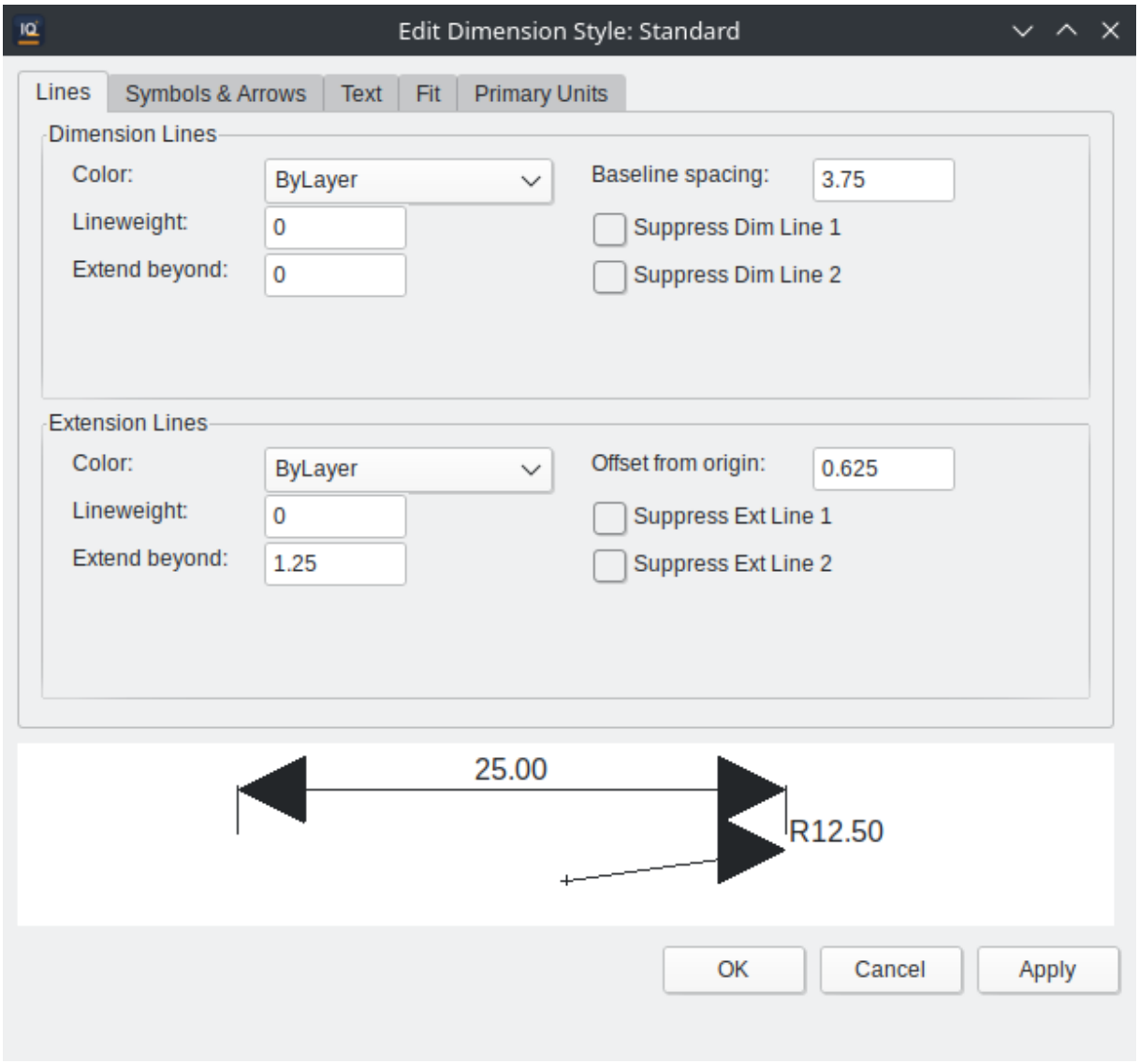


Fig. 2: The Dimension Style Editor dialog.

13.10 Plot Style Tables (CTB)

Format › Plot Styles… opens the **Plot Style Editor**. Plot style tables (CTB files) map AutoCAD colour indices to pen settings (colour, weight, screening) for printing and PDF export.

13.11 Tunnel Section Line Styles

Format › Tunnel Section Line Styles… opens the **Tunnel Section Line Style Editor**. A style bundles all visual settings for a tunnel section line entity into one named preset.

Three built-in presets are provided:

Preset	Description
Standard	Measured line with deviation annotations visible, no area hatch.
Hatched	Measured line with coloured area hatch (green = overbreak, red = underbreak) between measured and theoretical profiles.
Simple	Measured line only, no annotations or hatch.

The editor organises settings into groups:

- **Measured Line** — colour, weight, linetype, spline interpolation settings.
- **Deviation Lines** — visibility, weight, linetype, overbreak/underbreak colours.
- **Area Hatch** — visibility, pattern, scale, angle, alpha, separate overbreak/underbreak hatch colours.
- **Center Marker** — visibility, colour, size.
- **DR Display** — orientation (Auto/Inside/Outside), alignment (Horizontal/Vertical/Aligned), display mode (All, None, Extremes, Zone Extremes, Threshold, Interval, Smart), threshold and interval values.
- **DR Label Style** — reference to a named civil label style for the deviation labels.

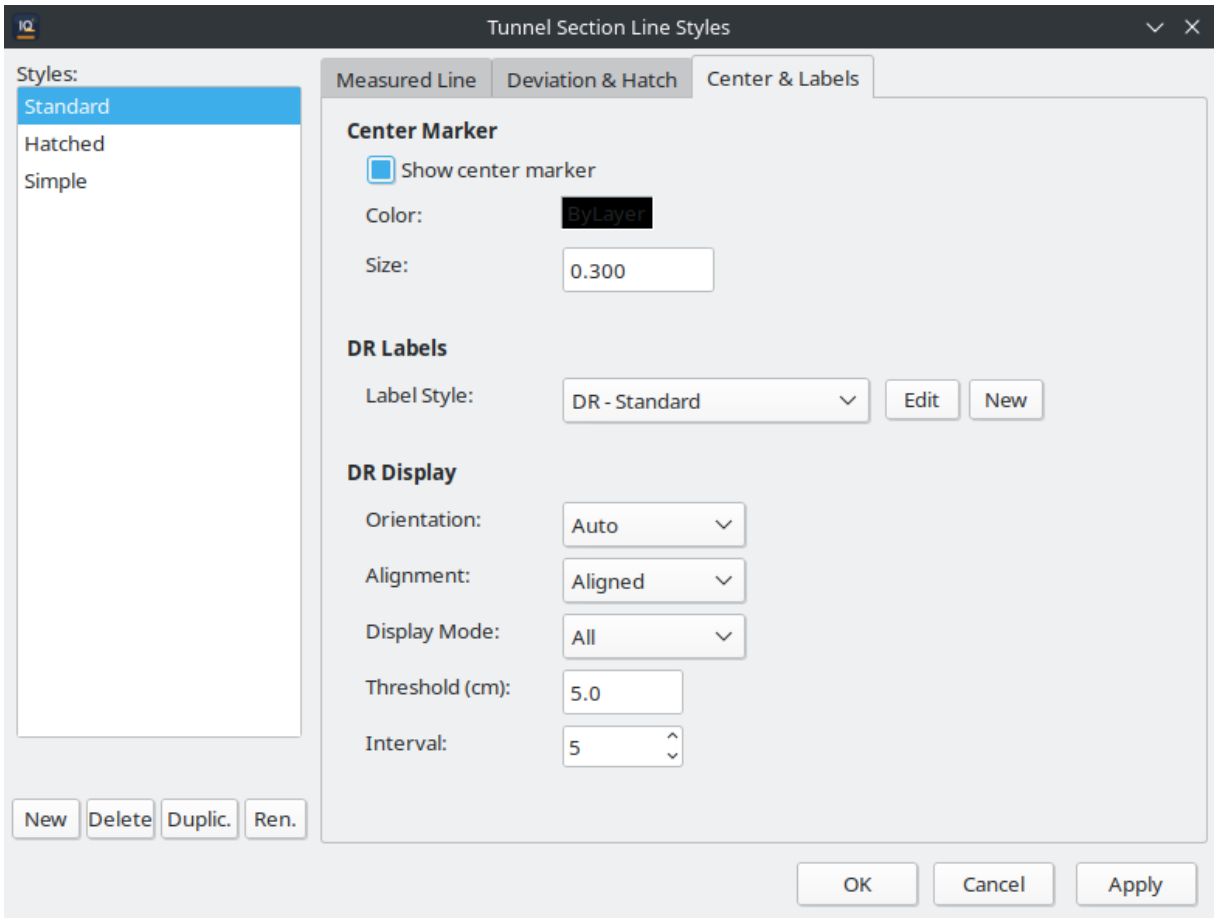


Fig. 3: The Tunnel Section Line Style Editor.

13.12 Parcel Styles

Format › Parcel Styles… opens the **Parcel Style Editor**. A parcel style controls the visual display of parcel boundary polygons, fills, and segment labels.

Five built-in presets are provided:

Preset	Description
Standard	Boundary line only, no fill or segment labels.
Property	Boundary with semi-transparent fill and bearing/distance labels on each boundary segment.
Easement	Dashed boundary with a light fill pattern for restricted zones.
ROW	Right-of-way style with a heavy boundary and cross-hatch fill.
Empty	All components hidden (`BoundaryVisible=False`). Use to suppress a parcel from display without deleting it.

The **BoundaryVisible** flag in the Boundary settings group controls whether the parcel outline is drawn. Setting it to False (as the Empty style does) hides the boundary entirely while preserving the parcel data.

The editor organises settings into groups:

- **Boundary** — visibility, colour, weight, linetype.
- **Fill** — visibility, fill colour, transparency.
- **Labels** — visibility, text height, colour, bearing format (Bearing, South Azimuth, North Azimuth, Angle), distance display, segment label text properties; a **Vertex Labels** group adds a **Vertex Names Visible** checkbox plus a label-style dropdown that picks which civil label style renders the corner names (see *Parcel Vertex Names* (page 177)). The built-in **Property** style has Vertex Names Visible checked by default — pair it with the built-in **Parcel Vertex** civil label style to display the names.

13.13 Alignment Styles

Format › Alignment Styles… opens the **Alignment Style Editor**. An alignment style controls the display of horizontal alignment entities: centerline, station ticks, PI markers, and direction arrows.

Four built-in presets are provided:

Preset	Description
Standard	Centreline + station ticks + PI markers + direction arrows.
Centerline Only	Centreline visible; ticks, PI markers, and arrows hidden.
Design Phase	Centreline as CENTER linetype, PI details and PI lines visible, full labels.
Empty	All components hidden. Use to suppress an alignment from display without deleting it.

Property Group	Settings
Centerline	Visibility, colour, weight, linetype, linetype scale.
Station Ticks	Visibility, major/minor tick intervals, tick lengths, colour, weight, linetype.
PI Markers	Visibility, colour, marker size.
Direction Arrows	Visibility, interval, colour, arrow size.

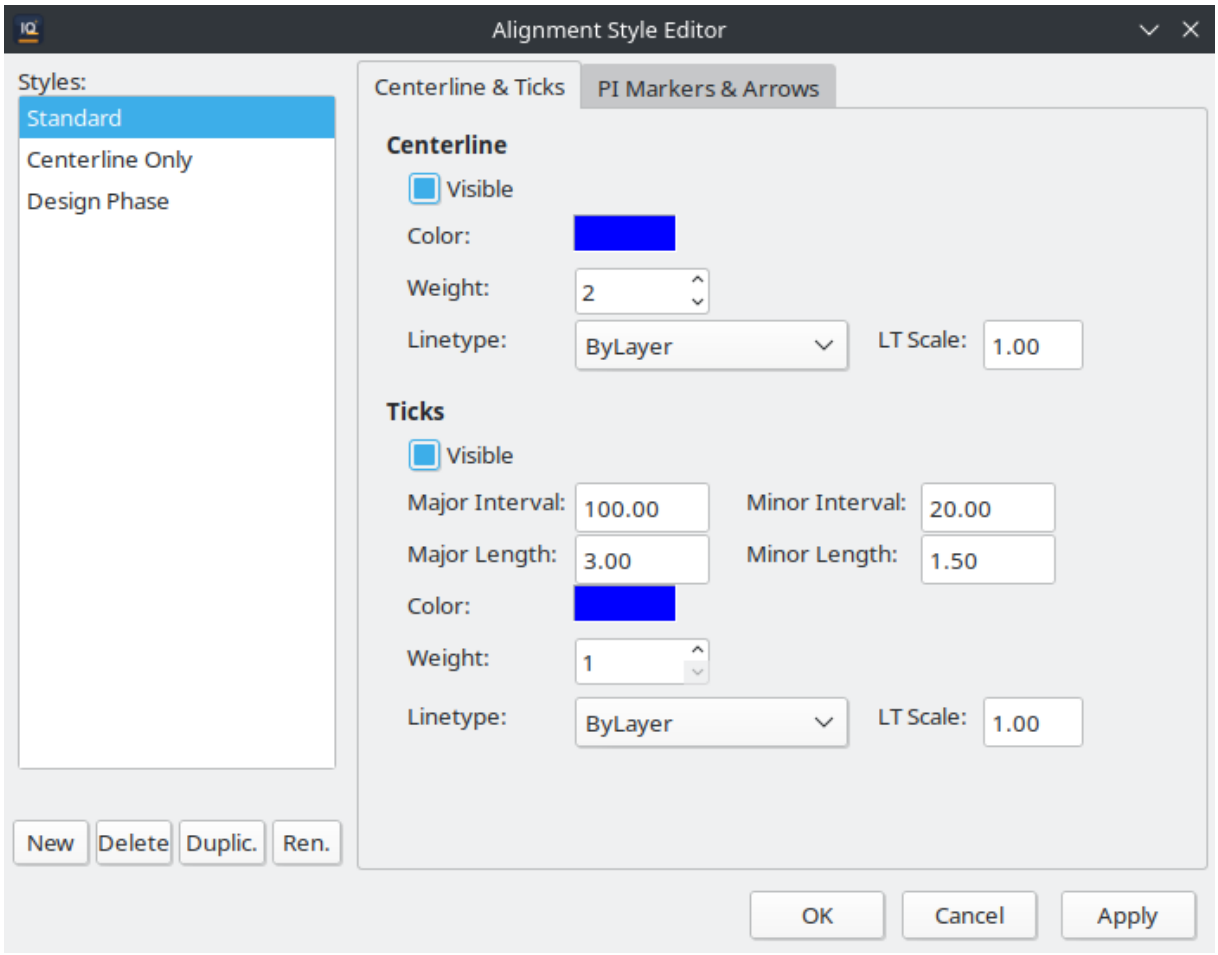


Fig. 4: The Alignment Style Editor.

13.14 Alignment Label Styles

Format › Alignment Label Styles… opens the **Alignment Label Style Editor**. Label styles control the text annotations shown along an alignment:

Property Group	Settings
Text	Text style name, text height, text colour.
Station Labels	Major/minor station label visibility, label offset from centerline.
PI Labels	Visibility, which data to show (radius, spiral length, tangent length, deflection angle), label offset.
Curve Data Labels	Visibility, which data to show (arc length, chord length, degree of curve), label offset.
Other	Alignment name label visibility, station equation labels.

13.15 Civil Label Styles

Format › Civil Label Styles… opens the **Civil Label Style Editor**. This is a unified label style system shared by all civil entities: survey points, surfaces (contour labels), alignments, tunnel deviations, and parcels.

A civil label style defines:

- **Components** — a list of label rows, each specifying a content type (e.g. point number, elevation, station, contour elevation, deviation), prefix, suffix, and number format.
- **Font** — text style, height, colour, bold/italic.
- **Placement** — anchor mode (point, linear, or area), offset, rotation, background mask settings.

Built-in presets cover common use cases for points, contours, alignments, tunnel deviations, and parcels. The label style editor groups styles by category: Point, Contour, Spot Elevation, Station, PI, Tunnel (DR), and Parcel.

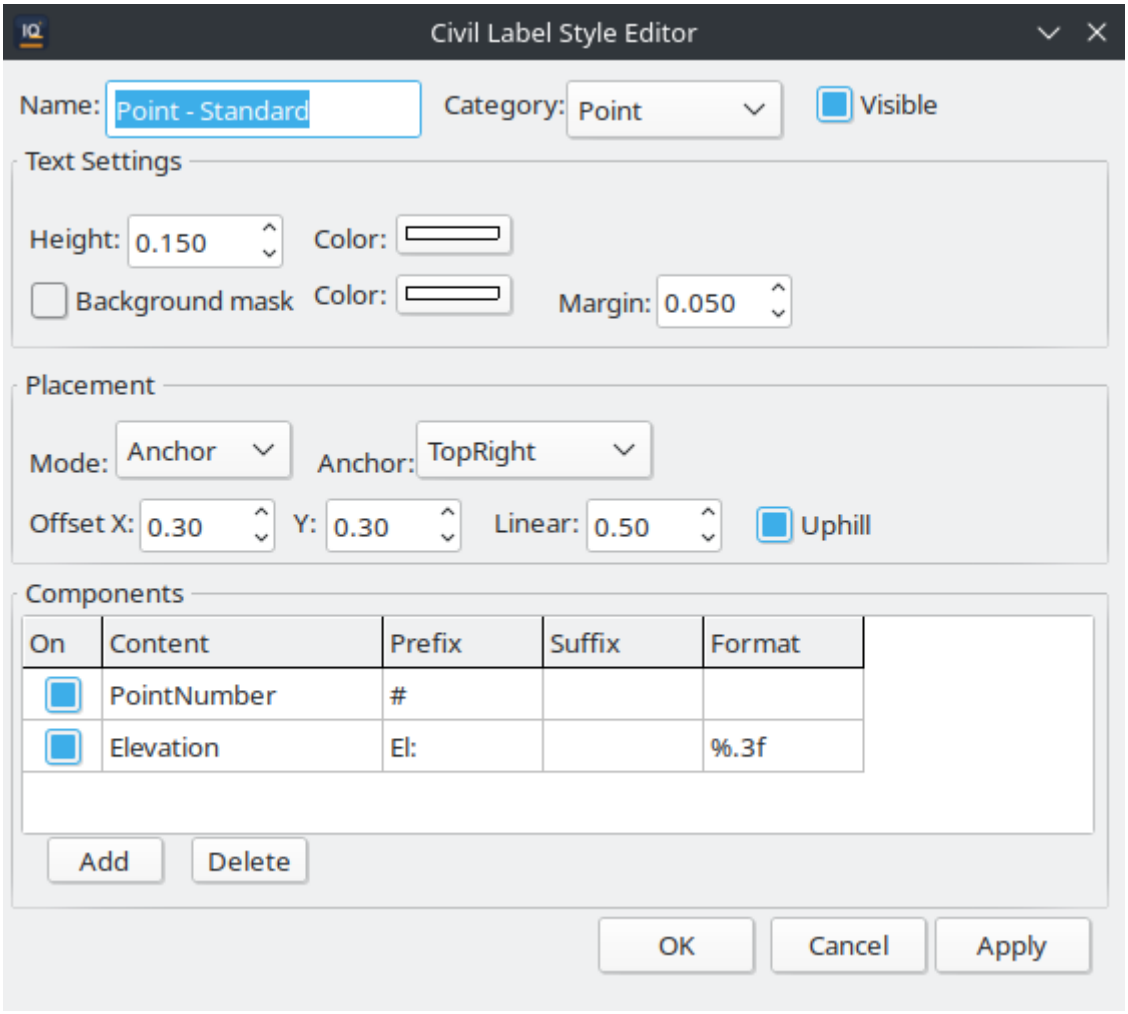


Fig. 5: The Civil Label Style Editor.

13.16 Unified Style Managers

IQcad provides two unified style manager windows that consolidate all style editors into a single dialog, eliminating the need to open individual style editors from separate ribbon buttons.

CAD Style Manager (*Format > CAD Style Manager…*) — a single window with a category tree on the left and the corresponding embedded editor on the right. Categories include:

- Text Styles

- Dimension Styles
- Table Styles
- Multiline Styles
- Point Styles (CAD point marker shape)
- Plot Style Tables (CTB)

Civil Style Manager (*Civil* > *Styles* or *Format* > *Civil Style Manager…*) — a single window for all civil-specific styles:

- Point Styles (survey point marker + label)
- Surface Styles
- Tunnel Section Line Styles
- Alignment Styles
- Parcel Styles
- Civil Label Styles

Both managers support creating, editing, duplicating, and deleting styles. Select a category in the tree, then use the style list and editor panel on the right. Styles are saved with the drawing.

The individual *Format* menu entries (e.g., *Format* > *Text Style…*, *Format* > *Surface Styles…*) remain available as shortcuts that open the corresponding section directly.

14 Exporting and Importing

14.1 Native File Formats

IQcad uses two native file formats:

Format	Description
.iqz	**Compressed** (gzip-compressed IQML). This is the default format for new drawings. Files are typically 5-10x smaller than uncompressed IQML.
.iqml	**Uncompressed** XML-based format. Human-readable and suitable for version control or debugging.

Both formats are auto-detected on open. Use **File > Save As…** to choose between them.

Large drawings with 1 000 000+ entities use a **streaming writer** that writes directly to the file instead of building an in-memory DOM tree, keeping memory usage constant.

14.2 Exporting

14.2.1 DXF Export

File › Export › DXF… exports the current drawing to an AutoCAD- compatible DXF file.

- All layers, colours, linetypes, and linewidths are preserved.
- Supports ASCII (R12, R2000, R2018) and Binary (R2000, R2018) formats, selectable in the export dialog.
- AEC Survey Points are exported as standard POINT entities.
- Tunnel Section Line deviation annotations are exported as LINE and TEXT entities.
- Polyface mesh entities are exported as a DXF `MESH` (AcDbSubDMesh) on R2000 and R2018, so the mesh stays one object. R12 export does not write the mesh. A DXF `POLYLINE` with the polyface-mesh flag is still imported.
- ACIS-based entities (3DSOLID, REGION, SURFACE) are passed through on export — their SAT/SAB data is preserved unchanged so they can be opened in AutoCAD or other ACIS-capable software.

14.2.2 Image Export

File › Export › Image… saves a raster snapshot of the current view.

- Supported formats: PNG, BMP.
- The exported image represents the current zoom and pan position.

14.2.3 SVG Export

File › Export › SVG… creates a Scalable Vector Graphics file.

- Suitable for embedding in reports, web pages, or vector editing tools.
- All geometry is preserved at full precision.

14.2.4 DWG Export

File › Export › DWG…, or **File › Save As…** with the DWG filter, writes the current drawing to AutoCAD's native DWG binary format.

- Written **in process** by IQcad's own DWG writer. No external converter is needed, and nothing is routed through a temporary DXF.
- Six target versions — R2000, R2004, R2007, R2010, R2013 and R2018 — chosen once in **File › Settings › DWG › Save Version**. R2018 is the default; R2000 is the most widely readable.
- Survey points and TIN surfaces are exported according to the **Survey objects** setting: as genuine Civil 3D objects, as plain geometry with or without labels, or not at all.
- A drawing opened from DWG and saved again without being edited is re-emitted byte for byte.

DWG Compatibility (page 109) covers version support, the entity list, the survey-export modes and the fallback converters in full.

14.2.5 PDF Export

File > Export > Export PDF or the `EXPORTPDF` command exports the active layout to a PDF file. Use `PLOTALL` to export all layouts to a single multi-page PDF.

- Supported from model space and paper space (layout) tabs.
- Page size and orientation come from the **Page Setup** dialog.
- Plot style tables (CTB) are applied if configured.

14.2.6 LandXML Export

File › Export › LandXML… writes survey data to LandXML 1.2 format.

- **Survey Points** → `<CgPoints>` elements.
- **Polylines** → `<Alignments>` elements.
- **Surfaces** → `<Surfaces>` with full `<Pnts>`, `<Faces>`, `<Boundaries>` (outer + island, inline NEZ point lists), `<BreakLines>` (each polyline as `<Breakline brkType="standard">`) and an IQcad `<Feature code="IQcadEdits">` block carrying the surface edit log.
- **Point Groups** → top-level `<Feature code="IQcadPointGroups">` with each group's name, description, visibility flag and point-number membership. Each surface also emits `<Feature code="IQcadSourceGroups">` listing the names of the point groups it was built from.

Polylines that are already attached to a surface as breaklines are not re-exported as alignments — they round-trip as breaklines only.

The IQcad-namespaced `<Feature>` blocks are vendor-private and are ignored by other LandXML consumers.

Coordinate mapping: World X → Easting, World Y → Northing.

14.2.7 GeoJSON Export

File › Export › GeoJSON… exports drawing entities to a GeoJSON file.

- Points, lines, polylines, arcs, and circles are exported.
- Arc and circle entities are tessellated into line segments.
- Bulge arcs in polylines are tessellated.
- A coordinate system dialog allows selecting the target EPSG code (default WGS 84). Coordinates are transformed from drawing units to the selected CRS.

14.2.8 KML / KMZ Export

File › Export › KML… exports drawing entities to KML format; choose the `.kmz` filter in the save dialog to write a compressed KMZ archive.

- Features are grouped into KML Folders by layer name.
- Coordinates are transformed to WGS 84 (longitude/latitude).

14.3 Importing

14.3.1 Open DXF

Use **File › Open…** and select a `.dxf` file. IQcad supports DXF R12 and R2000. Both ASCII and binary DXF files are supported and auto-detected. Layers, colours, linetypes, lines, polylines, arcs, circles, texts, and hatches are imported.

Additional 3D entity support on import:

- **Polyface meshes** — DXF `MESH` entities, and DXF `POLYLINE` entities with the polyface-mesh flag, are imported as native polyface mesh entities. Use `EXPLODE` to break them into individual 3D Faces or `MESH2SURFACE` to convert to a TIN surface.
- **ACIS entities** — 3DSOLID, REGION, BODY and SURFACE entities are imported as pass-through objects. Their ACIS SAT/SAB geometry data is preserved so that re-exporting to DXF **or DWG** produces a valid file, but IQcad does not render or edit the ACIS geometry itself. These entities appear in the drawing as bounding-box placeholders on their original layer. See *DWG Compatibility* (page 109) for how the solid is carried at each DWG version.

14.3.2 Open / Import DWG

Use **File › Open…** and select a `.dwg` file, or **File › Import › DWG…**

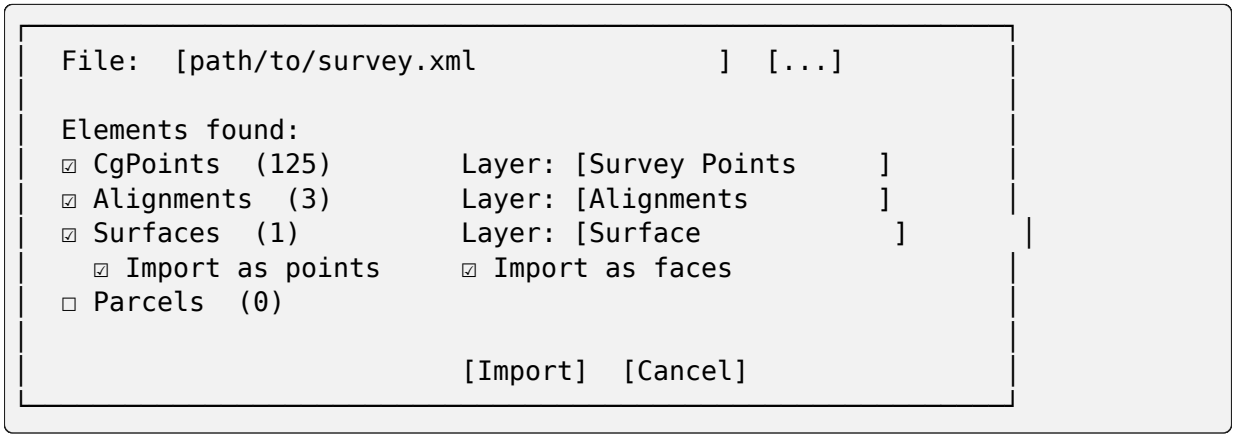
- Read **in process** by IQcad's own DWG reader — no external converter, no temporary DXF.
- R2000 through R2018 are read natively, including blocks, attributes, hatches, dimensions, tables, meshes, wipeouts, ACIS solids, layouts and viewports.

- Civil 3D objects in the file — COGO points, point groups, TIN surfaces, parcels and alignments — are decoded into IQcad's own civil objects and appear in the Prospector.
- Versions the native reader does not cover (R13/R14, R2025) fall through to **LibreDWG** or the **ODA File Converter** if either is installed.

See *DWG Compatibility* (page 109) for the full support matrix, the backend preference and how to install the fallback converters.

14.3.3 LandXML Import

File › Import LandXML… opens the LandXML import dialog.



LandXML element	Imported as	Notes
CgPoints	AEC Survey Points	PointNumber, Name, Northing/Easting/Elevation, Description preserved.
Alignments	Polylines	Arc elements are tessellated to line segments.
Surfaces	Points and/or 3D Face tiles, **or** a native AEC surface	Toggle *Import as points* and *Import as faces* independently. When imported as a native surface, `<Boundaries>`, `<BreakLines>` and the `IQcadEdits / IQcadSourceGroups` feature blocks are restored automatically.
Parcels	Closed polylines	Boundary curves tessellated.

If the file contains an `IQcadPointGroups` feature, the survey database's point groups are recreated and members re-assigned after the `<CgPoints>` are loaded.

Coordinate mapping: LandXML Northing → Y, LandXML Easting → X.

IQcad auto-detects LandXML versions 1.1 and 1.2 from the `xmlns` attribute.

14.3.4 GeoJSON and Esri JSON Import

File › Import › GeoJSON… imports a GeoJSON file or an Esri feature-set JSON file. The menu item stays **GeoJSON**. The open dialog is titled **Import GeoJSON or Esri JSON** and lists `.geojson` and `.json`. IQcad tells the two formats apart from the file contents. A document whose root `type` is `FeatureCollection`, `Feature`, or a geometry name is read as GeoJSON.

File › Export › GeoJSON… writes GeoJSON only.

The CRS dialog that follows asks for the drawing coordinate system. The default is EPSG:2100 (GGRS87 / Greek Grid). Its prompt still describes the incoming data as WGS 84. That description matches a GeoJSON file. An Esri file names its own source system, and the dialog choice is the coordinate system the geometry lands in.

The command requires the **Civil** module.

GeoJSON

- FeatureCollection, Feature, and a bare geometry object.
- Point, LineString, Polygon, the Multi* variants, and GeometryCollection.
- Coordinates are WGS 84 degrees (EPSG:4326) and are projected into the drawing CRS chosen in the dialog.
- Each polygon ring, including a hole, becomes its own closed polyline. A line becomes a polyline. A point becomes a CAD point.
- The CAD layer is the feature `name` property, or `GIS_Import` when that property is empty.

Esri feature-set JSON

This is the JSON from an ArcGIS feature service and from the Hellenic Cadastre map, maps.ktimatologio.gr[1]. There is no GeoJSON `type`. Attributes sit under `attributes`. A polygon is stored as `rings` and a polyline as `paths`.

- `spatialReference.wkid` is the source EPSG. `latestWkid` is used when `wkid` is absent. A Κτηματολόγιο extract uses **2100**, so the coordinates are already Greek Grid metres. Leave the drawing CRS at **2100** and the metres are copied as they stand. Another drawing CRS projects them into that system.
- A file that names no wkid is placed in the drawing CRS with its coordinates unchanged.
- Each ring of a polygon, including holes, is one closed polyline. Each path is a polyline. `points` and an `x` / `y` pair become CAD points. A feature with no geometry is skipped.
- The CAD layer is the `LAYER` attribute. On a Κτηματολόγιο extract that is the layer name written in the file, for example `ΛΕΙΤΟΥΡΓΟΥΝ ΚΤΗΜΑΤΟΛΟΓΙΟ`.
- `KAEK` is the entity **Name**. `AREA` is the entity **Description** (the official area, shown in the Object Inspector).
- Polygons come in as closed polylines, one per ring, so a hole keeps its own outline. Parcel objects are created with `CREATEPARCEL`.

14.3.5 KML / KMZ Import

File › Import › KML/KMZ… imports geographic data.

- KMZ archives are decompressed transparently.
- Placemarks with Point, LineString, and Polygon geometry are imported.
- Folder hierarchy is traversed recursively.
- A CRS dialog lets you specify the target drawing coordinate system.

[1] https://maps.ktimatologio.gr/

14.3.6 Shapefile

ESRI Shapefile is the format used by QGIS, ArcGIS and GDAL for GIS data exchange. IQCAD can both import and export shapefiles, including their sidecar files (.shx record index, .dbf attribute table, .prj projection WKT, .cpg text codepage).

Import

- Menu: `Files → Import → Shapefile...`
- Ribbon: `Output → Import → Shapefile`

The CRS picker dialog opens after file selection so you can choose the target EPSG (the .prj sidecar is auto-matched against 9 known presets; unknown WKT falls through to the dialog).

Export

- Menu: `Files → Export → Shapefile...`
- Ribbon: `Output → Export → Shapefile`

The export dialog asks for a basename without extension. Because the shapefile format requires one geometry type per file, IQCAD splits the output into up to three groups:

- `<basename>_points.shp` – all Point + MultiPoint features
- `<basename>_lines.shp` – all PolyLine features
- `<basename>_polygons.shp` – all Polygon features

Empty groups produce no files. After export, a message lists exactly which files were written.

For full reference (geometry coverage, DBF schema, CRS support, permissive vs. strict modes) see `cad/_documentation/io/shapefile.rst` in the source tree.

14.3.7 GPX

GPX 1.1 (GPS Exchange Format[2]) is the de-facto interchange for GPS units, hiking apps (Garmin, OsmAnd, Strava, Komoot), and OpenStreetMap editors. IQCAD reads and writes GPX 1.1 only; GPX 1.0 files are rejected with a clear error — re-save them in 1.1 first.

Import

- Menu: `Files → Import → GPX...`
- Ribbon: `Output → Import → GPX`

Export

- Menu: `Files → Export → GPX...`
- Ribbon: `Output → Export → GPX`

What round-trips

- Waypoints (`<wpt>`) ↔ point entities
- Routes (`<rte>`) → polyline entities (read only — see below)
- Tracks (`<trk>` with one or more `<trkseg>`) ↔ polyline entities
- Per-vertex elevations (`<ele>`)

[2] https://www.topografix.com/gpx.asp

- Track-level start/end timestamps stored as `time_start` / `time_end` attributes (per-vertex `<time>` is **not** preserved)
- Names and descriptions

Limitations

- All polylines export as `<trk>` (never `<rte>`). Major GPX consumers handle `<trk>` better and CAD geometry has no semantic difference between recorded vs. planned paths.
- Polygons, multipolygons, and multipoints have no GPX equivalent and are skipped on export with a warning per skipped feature.
- The GPX `<extensions>` namespace (vendor-specific data such as Garmin heart-rate samples) is read but ignored, and never written.

Licensing

GPX import and export require the **Civil** module.

> **Note**
>
> **Behaviour change as of 2026-05-01:** all GIS-format import and export (LandXML, GeoJSON, KML, Shapefile, GPX) now require the Civil module. Customers on Basic who relied on these features will need a Civil license.

14.3.8 Point Cloud Import

Point Cloud › Import Point Cloud imports point cloud data from LAS, LAZ, XYZ/CSV/TXT, PTS/PTX, PLY, and E57 files. See *Point Clouds* (page 233) in the Civil Tools chapter for full details. E57 (ASTM E2807) requires the `libE57CWrapper` runtime library next to the executable; without it the format is silently disabled and not offered in the import dialog.

14.3.9 Point Cloud Export

Point Cloud › Export Point Cloud exports the active point cloud to LAS, LAZ, XYZ, PTS, PLY, or E57 format. The export includes all points from the octree and runs in a background thread with progress reporting. E57 export emits a single scan with XYZ plus RGB / intensity when the source cloud carries them, and likewise needs `libE57CWrapper` at runtime.

15

Layouts, Viewports and Printing

A drawing is built at full size in **model space** and presented at a chosen scale on paper through a **layout**. A layout is a sheet of a definite size — A4, A3, A2, A1, A0 or a custom width and height in millimetres — carrying **viewports**: rectangular windows onto model space, each with its own centre, scale and rotation.

The tab strip along the bottom of the drawing area holds the **Model** tab and one tab per layout. Switching tabs switches the canvas: on **Model** you draw, on a layout you compose the sheet. When a layout tab is active the ribbon grows a **Layout** tab with the commands below.

15.1 Creating and Managing Layouts

Right-click the tab strip, use the **Layout** ribbon tab (**New**, **Delete**, **Rename**, **Page Setup**), or run `LAYOUT` (alias `LO`):

Option	Action
New	Adds a layout and its tab. Layer definitions are copied from model space, and stay in step with it: a layer added, renamed or recoloured in model space is pushed to every layout on the next tab switch.
Delete	Removes the current layout and everything drawn on it.
Rename	Renames the current layout. The name is what appears on the tab and in the DWG/DXF layout table.
List	Prints the layouts in the drawing to the command line.
Set	Switches to a layout by name.

Layouts are saved in `.iqz`, and read from and written to DXF and DWG, so a sheet composed here opens as a layout in AutoCAD and vice versa.

15.2 Page Setup

File › Print › Page Setup…, the **Page Setup** button on the *Layout* ribbon tab, the tab's right-click menu, or `PAGESETUP` opens the page setup for the active layout.

Setting	Meaning
Paper size	A0 (841 × 1189 mm), A1 (594 × 841), A2 (420 × 594), A3 (297 × 420), A4 (210 × 297), or **Custom** with an explicit width and height in millimetres.
Orientation	Portrait or landscape. Landscape swaps the effective width and height of the sheet.
Margins	Non-printable border, in millimetres, on each of the four edges.
Plot style table	The CTB table that maps drawing colours to printed pen colours, widths and screening. See *Format Settings* (page 85).
PSLTSCALE	`1` (the default) normalises linetype dash lengths to paper units, so a dashed line looks the same in every viewport regardless of its scale. `0` scales dashes with model space.

A layout can hold several **named page setups** — for example one for A3 draft prints and one for A1 final plots. Save the current settings under a name with **Save…** in the Print dialog and pick it again later; **Delete** removes one.

15.3 Creating Viewports

On a layout tab, `MVIEW` (alias `MV`) or the **Viewport** button on the *Layout* ribbon tab creates a window onto model space:

- **Fit** (the default) — one viewport filling the printable area of the sheet.
- **2Points** — click two opposite corners to place a viewport of your own size.

The new viewport shows the whole model-space extent. To set what it shows:

1. **Double-click inside the viewport** to activate it. The border highlights and the mouse now pans and zooms *model space through the window* rather than moving the sheet.
2. Pan and zoom to the view you want.
3. **Double-click outside** the viewport to go back to paper space.

Select a viewport (a single click on its border in paper space) to edit it in the Object Inspector: centre, scale, rotation, size, lock and border options are all properties there — see *Object Inspector* (page 63), *Viewport Properties*.

15.4 Scale, Lock and Freeze

- **Scale** is the ratio of paper units to model units. Setting *ViewScale* to `0.001` on a metre-based drawing gives a 1:1000 plot.
- **Lock** freezes the view. A locked viewport ignores zoom and pan while it is active, which is what stops an accidental scroll from silently changing the plot scale of a finished sheet.
- **Frozen layers** are per viewport: a layer listed there is hidden in that window only, and stays visible everywhere else. This is how one sheet shows contours and the next one shows only the road centreline, from a single model.

15.5 Viewport Commands

Command	Alias	Action
VPCLIP	—	Clips the viewport to a shape other than its rectangle: **Rectangle**, **Circle**, or **Polygon** (pick the corners). Only the model geometry inside the clip is drawn.
VPFADE	VPF	Toggles fading of everything outside the active viewport, so the window under edit stands out. Requires an active viewport — double-click into one first.
VPSCALEBAR	VPSB	Toggles a scale bar drawn along the selected viewport.
MVSETUP	MVS	Reserved for viewport rotate/scale/title setup. **Not implemented in this release** — it reports so on the command line. Use the Object Inspector for viewport scale and rotation in the meantime.

15.6 Draw Order

Sheet composition often needs one object drawn over another — a hatch behind its linework, a wipeout over a detail, a title block on top. DRAWORDER (alias DR) sets the display order of the selection, with four shorthand commands for the common cases:

Command	Action
FRONT	Brings the selected objects to the front of everything.
BACK	Sends them behind everything.
ABOVE	Places them immediately above a reference object you then pick.
BELOW	Places them immediately below a picked reference object.

Draw order is a drawing property: it is saved and restored, and honoured by the screen, the PDF export and the printer alike.

15.7 Printing

File › Print › Print…, **Ctrl+P**, the **Print** button on the *Layout* ribbon tab, or PLOT (alias PRINT) opens the Print dialog.

Group	Contents
Layout	Which layout to plot. Model space can be plotted directly too.
Printer/Plotter	The system printer to send the job to.
Paper size	Paper size and Portrait / Landscape.
Plot area	**Layout** (the sheet as composed), **Extents** (everything drawn), **Window** (click **Pick Window** and drag a box), or **Display** (what is on screen now).
Plot scale	**Fit to paper**, or an explicit ratio typed as two numbers (for example `1 : 1000`). **Center on page** centres the result.
Plot offset	X and Y shift of the plotted area on the paper, in millimetres.
Plot options	**Plot lineweights** (honour per-object and per-layer widths), **Plot with plot styles** (apply the CTB table), and **Plot stamp** (print the file name, date and scale in the margin).
Plot style table	Selects the CTB table, with **Edit…** opening the plot style editor.
Page setup	**Save…** stores the current dialog settings on the layout as a named page setup; **Delete** removes one.
Preview	A live thumbnail of the sheet as it will print.

Print to PDF at the bottom of the dialog writes the same output to a PDF file instead of a printer.

15.8 Plotting to PDF

- `PLOTPDF` (alias `PDF`), **File › Print › Export PDF…**, or the **PDF** button on the *Layout* ribbon exports the **active layout** to a PDF file at its page setup's paper size and scale.
- `PLOTALL`, **File › Print › PDF All…**, or the **PDF All** button writes **every layout in the drawing** to one PDF, one page per layout.

The PDF is written by IQcad's own vector engine, not by a print driver: text stays text, lines stay lines, and the file opens at any zoom without pixellation. Surfaces, survey points, parcels, gradings, culverts, alignments, profile views, point-cloud sections and tunnel sections have dedicated renderers so that styled output matches the screen: an alignment, for example, plots with the centerline, ticks, station labels, arrows, PI markers and PI detail tables its alignment style enables, on the alignment's component layers, and a profile view plots its grid, profiles, labels, strips and title exactly as the viewport shows them. See *Exporting and Importing* (page 97) for the plain **PDF Export** of model space.

16

DWG Compatibility

DWG is AutoCAD's native binary format, and IQcad reads and writes it **in process** — there is no temporary DXF, and no external converter to install. Opening a `.dwg` file is the same operation as opening an `.iqz`: **File › Open…** and pick it. Saving to DWG is **File › Save As…** with the DWG filter, or **File › Export › DWG…**.

Because the format is binary and undocumented in places, this chapter states plainly what is supported, what is approximated, and where the fallbacks are.

16.1 Supported Versions

Version	Magic	Read	Write	Notes
R13 / R14	`AC1012` / `AC1014`	—	—	Recognised, then handed to the fallback converter.
R2000	`AC1015`	✓	✓	The most widely readable output. Choose this when the file must open in old or third-party software.
R2004	`AC1018`	✓	✓	
R2007	`AC1021`	✓	✓	Compressed three-stream objects.
R2010	`AC1024`	✓	✓	
R2013	`AC1027`	✓	✓	
R2018	`AC1032`	✓	✓	**The default output version.**
R2025	`AC1036`	—	—	Not yet supported.

Every written version is validated offline against the checksums a conforming reader verifies, and files at all six versions are accepted by the ODA File Converter and open in AutoCAD.

The output version is set once, in **File › Settings › DWG › Save Version** — see *Application Settings* (page 113).

16.2 What Round-Trips

Read and written natively:

Group	Objects
Geometry	LINE, CIRCLE, ARC, POINT, ELLIPSE, LWPOLYLINE, 2D and 3D POLYLINE with their vertices, SPLINE, XLINE, RAY, 3DFACE, SOLID, TRACE.
Annotation	TEXT, MTEXT, ATTRIB, ATTDEF, all seven DIMENSION sub-types plus arc-length and jogged-radius dimensions, MULTILEADER, legacy LEADER, TOLERANCE, SHAPE (a single glyph from an `.shx` shape-file).
Fills and masks	HATCH with its pattern and boundary, WIPEOUT, IMAGE (raster references).
Structure	BLOCK definitions, INSERT references, MINSERT arrays, LAYER, LTYPE, STYLE, DIMSTYLE, TABLESTYLE, LAYOUT records and VIEWPORT windows.
Meshes	`AcDbSubDMesh`, polyface meshes and M × N polygon meshes.
Tables	`ACAD_TABLE`, with its cell contents and a Standard table style.
Solids	REGION, 3DSOLID and BODY. The ACIS geometry is carried through unchanged — encrypted SAT on R2000–R2010, and the SAB prototype section on R2013 and R2018 — so a solid authored in AutoCAD comes back out of IQcad as the same solid. IQcad displays its wireframe extent; it does not model or edit ACIS geometry.
Civil 3D	COGO points and the point-group tree, TIN surfaces (including the surface's own stored triangulation, hidden faces and deleted faces), parcels and alignments. See *Exporting Survey Objects* (page 111) below.

Objects IQcad has no equivalent for are counted and reported rather than silently dropped, and proxy entities are recorded under the class they stand in for.

16.3 Opening and Re-saving Untouched

If a drawing is opened and saved again **without being edited**, and the target version matches the source, IQcad re-emits the original bytes exactly. Nothing is re-encoded, so nothing can be lost. The moment an object is added, removed or changed the drawing goes through the normal encoder instead.

This makes "open it here, save it, send it on" safe even for drawings using features IQcad does not model.

16.4 Backends and the Fallback Chain

IQcad tries backends in order and takes the first that succeeds:

1. **Native (in-process)** — the built-in reader/writer. The default, and the fastest by a wide margin.
2. **LibreDWG CLI** — the external `dwg2dxf` / `dxf2dwg` tools.
3. **ODA File Converter** — the Open Design Alliance's free converter.

Both CLI backends work by converting through a temporary DXF, so they are slower and lose anything DXF cannot carry. They exist to cover file versions the native engine does not read (R13/R14, R2025).

The chain stops immediately on success and on a **corrupt** file — a damaged file surfaces an error rather than being quietly retried, because corruption is not something a different backend will fix. It moves on to the next backend only for an unsupported version, an unsupported object, or a backend that is not installed.

Set the preference in **File › Settings › DWG › Reader Backend**. Pin a CLI converter there if you need to reproduce its behaviour or suspect a problem in the native path.

Installing a CLI converter (only needed for the fallbacks):

- **Linux** — `sudo apt install libredwg-tools`, or build from https://github.com/LibreDWG/libredwg.
- **Windows** — the LibreDWG release binaries, or the free ODA File Converter from https://www.opendesign.com/guestfiles/oda_file_converter.

16.5 Exporting Survey Objects

Survey points and TIN surfaces have no counterpart in plain DWG: they are Civil 3D custom objects, which ordinary CAD programs cannot read. IQcad therefore lets you choose how they leave the drawing, in **File › Settings › DWG › Survey objects**.

Mode	What is written
Civil 3D objects	Genuine Civil 3D objects: an `AeccDbPointsNode` carrying the COGO points with the group and query tree above it, a `Root` entry in the named-object dictionary, `AeccDbSurfaceTin` for surfaces, and `AeccDbFace` / `AeccDbAlignment` for parcels and alignments. **Civil 3D lists these in the Prospector and edits them as its own.** Plain AutoCAD and other CAD programs show nothing for them.
Points + surface, with labels	The default. Geometry any CAD can read: a `POINT` per COGO point, a `TEXT` label for its number and description, and a `3DFACE` per TIN triangle. Civil 3D rebuilds a surface from the 3DFACEs in a single command.
Points + surface, geometry only	The same, without the text labels.
Do not export	Neither points nor surfaces are written. Use this when the recipient only wants the drafting content.

Tip

Send **Civil 3D objects** to a Civil 3D user and **with labels** to anyone else. If you do not know which, the labelled geometry mode is the safe choice — it is readable everywhere and still rebuilds into a surface.

Reading survey objects needs no setting: Civil 3D points, point groups, surfaces, parcels and alignments in an incoming file are decoded into IQcad's own civil objects automatically, and appear in the Prospector.

16.6 Known Limitations

- **Multiline styles** — the writer emits one `Standard` MLINESTYLE. A multiline's geometry and its element pair survive; a bespoke named style does not.
- **ACIS solids** — carried through, not modelled. IQcad cannot edit, boolean or section a 3DSOLID.
- **Point descriptions** — Civil 3D COGO points written in *Civil 3D objects* mode carry number and position, not the description field.

- **R2000 is still the safest target** for third-party software, even though all six versions are written and validated. The Save Version list carries a stale “not yet supported” note beside R2007; the version is in fact written and accepted.

If a file will not open, switch the reader backend to **ODA File Converter** in the settings and try again; that tells you quickly whether the problem is the file or the native reader.

17 Application Settings

File › Settings opens the *Application Settings* dialog. These are **program** preferences — how IQcad looks and behaves — as distinct from **drawing** settings such as layers, styles and drawing variables, which live with the drawing and are covered in *Format Settings* (page 85).

The dialog has eight tabs. **OK** applies and closes, **Apply** applies without closing, **Cancel** discards.

17.1 General

Setting	Default	Meaning
Theme	Metro Dark	Ribbon and window colour scheme. Also on the **File › Theme** menu.
Show menu bar	On	Shows the classic menu bar above the ribbon. **File › Toggle Menu Bar** does the same thing.

17.2 Display

Setting	Default	Meaning
Background	Black	Drawing-area background colour. Entities drawn in the "white/black" colour follow it so they stay visible.
Snap marker	Yellow	Colour of the object-snap marker and its tooltip.
Grip size	5 px	Half-side of the square grip handles on a selected object.
Crosshair size	50 px	Arm length of the cursor crosshair.
Pick size	5 px	Half-side of the pick box. A larger box makes objects easier to click and overlapping ones easier to catch.
Snap aperture	15 px	Radius within which an object snap engages.

17.3 Grid

Setting	Default	Meaning
Show grid	On	Draws the reference grid. `GRID` toggles it per drawing.
Grid spacing	10	Minor grid interval, in drawing units.
Major every N	5	Draws a heavier line every N minor intervals.
Show coordinate labels	Off	Prints the coordinate value along the major grid lines.
Grid style	Lines	**Lines**, **Dashed** or **Dots**.

17.4 Ruler

Setting	Default	Meaning
Show rulers	On	Rulers along the top and left edges of the drawing area.
Ruler width	28 px	
Ruler background	System button face	
Ruler text	Black	
Opacity	200	`0`–`255`. Lower values let the drawing show through the ruler.

17.5 UCS Gizmo

Setting	Default	Meaning
Show UCS gizmo	On	The axis indicator described in *Navigation* (page 35).
Display at origin	Off	Draws the gizmo at the world origin instead of in a screen corner.
Icon size	80 px	
Corner	Bottom-left	Which corner the gizmo sits in when it is not at the origin.
Background opacity	80	Opacity (`0`–`255`) of the gizmo's backing plate.

17.6 ViewCube

Setting	Default	Meaning
Visible	On	The 3D orientation cube. Click a face, edge or corner to snap the view to it.
Corner	Top-right	
Size	100 px	Cube size on screen.

17.7 Advanced

Setting	Default	Meaning
Enable debug logging	Off	Writes `iqcad_debug.log` next to the executable. Useful when reporting a problem. **Requires a restart.**
GPU rendering	Auto	**Auto** uses the GPU when a usable OpenGL context is available and falls back to software rendering when it is not; **On** forces GPU rendering; **Off** forces software. GPU rendering is what makes large surfaces and point clouds usable — see *Point Cloud Module* (page 231). `GPUTEST` reports what the machine actually supports.

17.8 DWG

Setting	Default	Meaning
Reader Backend	Native (in-process)	Which engine opens DWG files: **Native**, **LibreDWG CLI** or **ODA File Converter**. Native is fastest and needs no external install; the CLI converters remain in the fallback chain.
Save Version	R2018 (AC1032)	Target version for the native writer: R2000, R2004, R2007, R2010, R2013 or R2018. Older versions are more widely compatible.
Survey objects	Points + surface, with labels	How COGO points and TIN surfaces are exported — as genuine Civil 3D objects, as plain geometry with or without labels, or not at all.

DWG Compatibility (page 109) covers all three in full, including which mode to send to whom.

17.9 Where Settings Are Stored

Settings are written to `settings.json` as soon as they are applied, and reloaded at startup:

Platform	Path
Windows	`%APPDATA%\IQcad\settings.json`
Linux	`~/.config/IQcad/settings.json`

The same file carries state the dialog does not show: window geometry, panel widths, the recent-files list, custom map providers, plugin enable/disable state, and the Blade Runner editor's session. Deleting the file resets IQcad to factory defaults — the full list of which is in *Program Defaults Reference* (page 281).

Part IV

Super Tools

Express- and CadTools-style drawing helpers for cleanup and polyline work. Commands live on the **Super Tools** ribbon tab and the **Super Tools** menu, and are fully available on the command line.

18 Overview

Super Tools is the home for drafting utilities that sit alongside standard Modify commands: geometry cleanup, polyline simplification, and (as the suite grows) best-fit lines, stationing, and similar helpers inspired by Autodesk Express Tools and third-party toolkits such as CadTools.

18.1 Access

Where	How
Ribbon	Tab **Super Tools** (after **Tools**, before **Output**)
Menu bar	**Super Tools** (after **Tools**; Civil / Point Cloud insert between Tools and Super Tools when those modules are present)
Command line	Type the command name or alias (see each command page and the *Command Reference* (page 257))

18.2 Ribbon and menu groups

Commands are organised so related tools share a group:

Group	Ribbon control	Menu path
Cleanup	**Cleanup** split button (click = default command; arrow = full list)	**Super Tools → Cleanup → …**
Polylines	**Polylines** split button (click = default command; arrow = full list)	**Super Tools → Polylines → …**

New Super Tools commands are added to these groups (or new groups as needed). After each command lands, this chapter and the changelog are updated.

18.3 Current commands

Command	Alias	Group	Summary
FLATTEN	—	Cleanup	Project selected objects (or the whole drawing) to Z = 0
PLSIMPLIFY	PLSIM	Polylines	Douglas–Peucker vertex reduction on 2D/3D polylines
PLGETZ	PGZ	Polylines	Sample Z from surface/point cloud on polylines and lines (multi-select)
BESTFITLINE	BFL	Polylines	Best-fit 3D line through selected points/vertices (PCA)

See *Cleanup* (page 123) and *Polylines* (page 125) for full behaviour, prompts, and notes.

19 Cleanup

Cleanup tools remove unwanted elevation or duplicate geometry so drawings stay flat and tidy for 2D delivery.

19.1 FLATTEN

Sets **Z = 0** on the selected objects. For polylines, the **Elevation** property is zeroed as well so DXF/IQZ export stays consistent.

Access

- Ribbon: **Super Tools → Cleanup** (default click) or the Cleanup dropdown → **Flatten**
- Menu: **Super Tools → Cleanup → Flatten**
- Command line: `FLATTEN`

Workflow

1. Optionally pre-select the objects to flatten.
2. Run `FLATTEN`.
3. If nothing is selected, pick objects and press **Enter**, or press **Enter** immediately to flatten **all** objects in the drawing.
4. The command line reports how many objects changed.

Notes

- Undoable (full vertex Z and polyline Elevation restore).
- Non-zero Z only: objects already at Z = 0 are left alone in the change count.
- Formerly available under **Modify**; it now lives only under **Super Tools** so cleanup stays with other Super Tools utilities. The command name `FLATTEN` is unchanged.

20 Polylines

Polyline helpers for dense paths from contours, GPS traces, survey imports, and traced sections. (Reverse a polyline with **PEDIT → Reverse** — that remains under Modify / PEDIT, not Super Tools.)

20.1 PLSIMPLIFY

Reduces the number of vertices on **2D polylines** (`TCADPolyline`) and **3D polylines** (`TCAD3DPolyline`) with the **Douglas–Peucker** algorithm, keeping the shape within a maximum deviation **tolerance** (drawing units).

Access

- Ribbon: **Super Tools → Polylines** (default click) or the Polylines dropdown → **Simplify…**
- Menu: **Super Tools → Polylines → Simplify…**
- Command line: `PLSIMPLIFY` or `PLSIM`
- With pre-selection and a tolerance: `PLSIMPLIFY 0.05`

Workflow

1. Select one or more polylines (or run the command first, then select and press **Enter**).
2. Enter a **tolerance** (default `0.01`). Press **Enter** alone to accept the last value used in the session.
3. Each polyline is rewritten in place with fewer vertices.
4. The command line reports how many polylines changed and the total vertex count before → after.

Tolerance geometry

Entity	Deviation measure
2D polyline	Perpendicular distance in the **XY** plane (plan)
3D polyline	Perpendicular distance in **3D**

Special cases

Case	Behaviour
Closed polylines	Remain closed; anchors keep a continuous ring
Arc bulges	Densified to straight chords, then simplified; result has **straight** segments only (bulges cleared)
Spline-fit	Control / densified vertices are simplified; **SplineFlag** is cleared
Widths	Reset with the new vertex array (uniform width drawings stay visually fine)
Non-polylines	Skipped with a count in the command-line summary
Minimum vertices	Open: at least 2; closed: at least 3

Notes

- In-place edit (not a new copy); fully **undoable**.
- Tolerance `0` or a value smaller than residual noise leaves the polyline unchanged (reported as "no vertices removed").
- Reverse direction is intentionally **not** duplicated here — use `PEDIT` → **R**everse.

20.2 PLGETZ

Samples elevations from a **surface** or **point cloud** along the selected path(s) and writes **Z** onto the path.

Access

- Ribbon: **Super Tools → Polylines** dropdown → **Get Z…**
- Menu: **Super Tools → Polylines → Get Z…**
- Command line: `PLGETZ` or `PGZ`

Supported objects (multi-select)

2D polylines, 3D polylines, and **lines**. Window/crossing or pick-additive multi-select is honoured.

Workflow

1. Select one or more paths (or run the command then select and press **Enter**).
2. Choose elevation source: **Surface** or **Pointcloud** (`S` / `P`). If a point cloud is already selected, or the drawing has only clouds (no surfaces), **Pointcloud** is the default.
3. If more than one source exists, enter its **name** or **index** (`1`, `2`, …) or **click** it. A single surface/cloud is auto-selected; a cloud already in the selection is preferred. Point-cloud sampling uses the nearest point in a vertical column of the given search radius (default `2`).
4. Sample mode: **Vertices** (default), **Interval**, or **Both**.
5. If **Interval** or **Both**: enter **Vertex interval** (default `1` drawing unit, session-remembered).
6. If Pointcloud: enter search radius (default `2`).
7. The command drapes Z and reports how many samples hit vs missed.

Result

Input	Output
2D polyline	**Replaced** by a new **3D polyline** (same layer/color/closed flag); the 2D entity is removed
3D polyline	Z updated **in place** (XY preserved according to sample mode)
Line (vertices mode)	Endpoints updated **in place** (Z sampled at each end)
Line (interval densify)	**Replaced** by a 3D polyline with intermediate samples

Sampling

Mode	Behaviour
Vertices (default)	Z only at existing vertices (after densifying arc bulges)
Interval	Samples every *interval* along plan length (default interval `1`)
Both	Original vertices plus interval fills between them

Missing samples

If the surface has no triangle at (X,Y) or the point cloud has no point within the radius, **Z = 0** is used (default policy). The command-line summary reports hit and miss counts.

Notes

- Surface sampling uses `GetElevationAt` (TIN / DEM).
- Point cloud sampling takes the **nearest XY** point inside a vertical search column of the given radius.
- Arc bulges on 2D polylines are densified to chords before sampling; the result is a straight-segment 3D polyline.
- Fully **undoable**.

20.3 BESTFITLINE

Fits a **3D straight line** through the vertices of the selection using principal-component analysis (largest covariance axis). Creates a new `LINE` entity spanning the cloud of points.

Access

- Ribbon: **Super Tools → Polylines** dropdown → **Best Fit Line…**
- Menu: **Super Tools → Polylines → Best Fit Line…**
- Command line: `BESTFITLINE` or `BFL`

Workflow

1. Multi-select points, lines, and/or polylines (all their vertices are collected).
2. Run `BESTFITLINE` (or select after starting the command, then Enter).
3. A new line is added on the current layer from the min to max projection along the best-fit axis.

Notes

- Needs at least **2** distinct sample points.
- Works in full **3D** (not plan-only).

- Does **not** delete or modify the source objects.
- Undoable via the standard add-object undo.

Part V

Civil Module

Civil tools extend the standard CAD objects with survey-specific capabilities: survey points, TIN surfaces, alignments and profiles, section sheets, parcels, tunnel analysis, coordinate transformations, corridors, gradings, culverts and sites.

21

Civil Toolspace and Settings

Civil tools extend the standard CAD objects with survey-specific capabilities. They are accessible from the **Civil menu**, the **Civil ribbon tab**, and the **Civil Toolspace** panel.

21.1 Civil Toolspace

Toggle the panel with **Civil › Civil Toolspace**. The panel shows a tree with four top-level branches:

```
▼ Survey Points  (N)
    ▼ _All Points  (N)
        └ #1   N=538345.123   E=4872341.456   Z=412.5   BM-001
        └ #2   …
    ▼ GroupA  (3)
        └ #1   …
▼ Tunnel Section Lines  (N)
    └ Measured-1  (43 pts)
▼ Surfaces  (1)
    ▼ Terrain
        ▼ Definition
            └ Point Groups
            └ Boundaries
        ▼ Contours
            └ Major @ 5.0
            └ Minor @ 1.0
▼ Alignments  (1)
    └ Alignment_1  (L=350.00  PIs=5)
```

- Click the **Refresh** button (top-right of the panel header) to rebuild the tree after bulk operations.
- **Double-click** a leaf node to select that object in the canvas and zoom to it.



Fig. 1: The Civil Toolspace panel with survey points, surfaces, and alignments.

21.2 Civil Settings

Open with **Civil › Civil Settings…** to configure per-drawing civil engineering parameters. All settings are saved inside the IQML file and restored when the drawing is reopened.

The dialog has six tabs.

21.2.1 Units & Zone

Setting	Description
Linear Unit	Choose **Meters**, **Feet**, or **US Survey Feet**.
Drawing Scale	Numeric scale denominator — e.g. enter `500` for a 1:500 drawing.
Coordinate System	Browse by category or search by name/code from ~182 built-in EPSG systems. The detail panel shows the EPSG code, datum, projection method, and linear unit.

21.2.2 Transformation

Setting	Description
Enable Transformation	Check to activate coordinate transformation for the drawing.
Reference Easting (X)	X coordinate of the reference (base) point.
Reference Northing (Y)	Y coordinate of the reference (base) point.
Rotation	Rotation angle in degrees.
Scale Factor	Multiplicative scale factor (default `1.0`).

21.2.3 Ambient Settings

Setting	Description
Precision	Number of decimal places (0–8) for distance, elevation, coordinate, angle, grade, and slope values.
Angle Format	Choose **Degrees**, **DMS** (degrees-minutes-seconds), **Grads**, or **Radians**.
Direction Format	Choose **Bearing**, **South Azimuth**, or **North Azimuth**.

21.2.4 Abbreviations

Standard abbreviations used in labels and reports:

Abbreviation	Default value
Station	`STA`
Right	`RT`
Left	`LT`
North / South / East / West	`N` / `S` / `E` / `W`
Elevation	`EL`
Grade Break	`GB`

Click **Reset to Defaults** to restore the factory abbreviation values.

21.2.5 Layers

The **Layers** tab controls the Civil 3D-style layers that civil objects create for their components (see *Civil Object Layer Defaults* (page 288) for the full list).

- **Enable civil object layers** turns the automatic layers on or off.
- **Layer prefix** is the leading group code, `C` by default.
- **Include object name in layer** inserts the object's name between the group and the suffix, e.g. `C-ROAD-Alignment_1-PI`.

The grid lists every object component with its **Group**, **Suffix**, the resulting **Layer** pattern and the layer **Colour**. Type in the Group and Suffix cells to rename a layer; click the colour swatch to choose a new colour. The Layer column updates as you type. **Reset Layers** restores the built-in names and colours for the whole grid.

The changes apply to layers created from then on. A layer that already exists in the drawing keeps its name and colour; rename or recolour it in the Layer Manager.

21.2.6 Draw Order

The Civil Settings dialog includes a **Draw Order** tab that controls the rendering priority of civil objects.

Inter-object type priority determines the order in which different civil object categories are drawn. The default order (back to front) is:

1. Surfaces
2. Parcels
3. Alignments
4. Points

Drag items in the list to reorder them. Objects drawn later appear on top.

Intra-surface sub-element ordering controls the draw order of components within a single surface (e.g. triangles before contours, contours before labels). This is configured per-surface in the **Surface Style Editor** on the Draw Order tab.

Dialog buttons:

- **OK** — applies all changes.
- **Cancel** — discards all changes.

- **Reset All** — restores every setting across all tabs to its default value.

21.3 Elevation Band Legend

When a surface uses the **Elevation Banding** style (or any style with elevation bands enabled), an on-canvas legend can be displayed showing stacked color boxes with the corresponding elevation ranges.

Enable or disable the legend per-style using the **ShowLegend** toggle in the **Elevation Bands** tab of the Surface Style Editor. The legend is rendered as a GL overlay and updates automatically when the surface elevation range or band configuration changes.

22 AEC Survey Points

An AEC Survey Point is a full civil survey marker storing a complete set of survey attributes alongside its drawing position.

Place with **Civil › Points › Civil Point** or the ribbon button, then fill in the properties in the Object Inspector.

22.1 Properties

Property	Description
PointNumber	Numeric survey ID. Used as the primary key in the database and displayed as the point label.
PointName	Descriptive name.
Northing / Easting	World coordinates. Moving the marker on the canvas automatically updates Northing and Easting, and vice versa.
Elevation	Z value stored with the point.
RawDescription	Survey field code (e.g. `EP`, `BM`, `CL`).
FullDescription	Expanded description used in reports and exports.
PointGroup	Assigns the point to a named group.
LockPoint	Prevents the point from being moved accidentally.
PointStyleName	Selects a named point style from the library.

22.2 Point Style Library

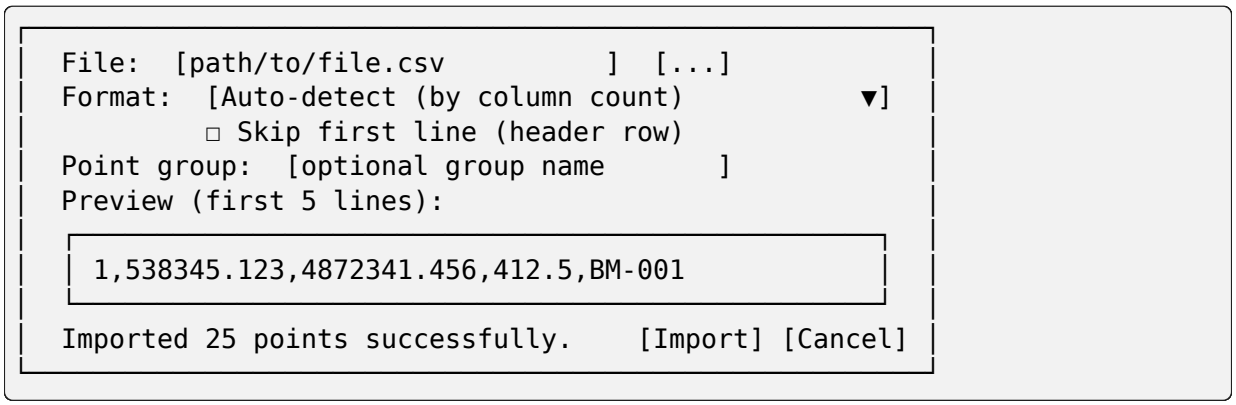
Named point styles control the marker shape and label appearance. Nine built-in presets are always available:

Style	Default marker	Label
Standard	Dot	Name and elevation
Benchmark	Circle with cross, yellow	Name, elevation and description
Control Point	Triangle with dot, red	Name, elevation and number
GPS Rover	Square with cross, green	Name, elevation, northing and easting
Iron Rod	Cross, cyan	Name and elevation
Detailed Points	Dot, cyan	Elevation with `El:` prefix
Spot Elevation Cross	Small cross	From the surface's spot elevation label style
Spot Elevation Dot	Small dot	Elevation value only
Empty	Hidden	Hidden

Edit or add styles via **Format › Point Style…** (**Ctrl+Shift+D**).

22.3 Importing Survey Points from CSV

Open the **Point Manager** (**Civil › Points › Point Manager…**) and click the **Import** button on its toolbar to open the CSV import dialog.



Supported formats:

Format	Column order
Auto-detect (by column count)	Detects automatically from the number of columns.
P, N, E, Z, Desc	PointNumber, Northing, Easting, Elevation, Description
N, E, Z, Desc	Northing, Easting, Elevation, Description
N, E, Z	Northing, Easting, Elevation
P, E, N, Z, Desc	PointNumber, Easting, Northing, Elevation, Description
E, N, Z, Desc	Easting, Northing, Elevation, Description
E, N, Z	Easting, Northing, Elevation

Accepted file types: `.csv`, `.txt`, `.pts`. Lines starting with `#` or `;` are treated as comments. Decimal separator is always `.`.

22.4 Point Groups

A point group is a named set of survey points. The points stay in the drawing; deleting a group does not delete them. `_All Points` is every point in the database, not a group, and it has no properties window.

Open the properties. In the Civil Toolspace, right-click a group under **Survey Points** and choose **Properties…**. The same item is on a group listed under a surface (**Definition › Point Groups**). On a surface the delete item is **Remove from Surface**; elsewhere it is **Delete Point Group**. A locked group cannot be deleted, and neither can a group that a surface still uses — remove it from the surface first.

Create a group from the tree. Right-click **Survey Points** and choose **Add Point Group…**. The properties window opens on a new empty group named `Point Group - (1)`, `Point Group - (2)`, and so on. **Cancel** before **Apply** leaves no group behind.

The window title is **Point Group Properties -** followed by the group name. **OK** writes the changes and closes. **Apply** writes them and stays open, so the Point List can be checked. **Cancel** discards edits that have not been applied. A change that was already applied is kept.

22.4.1 Point Group Properties — Information

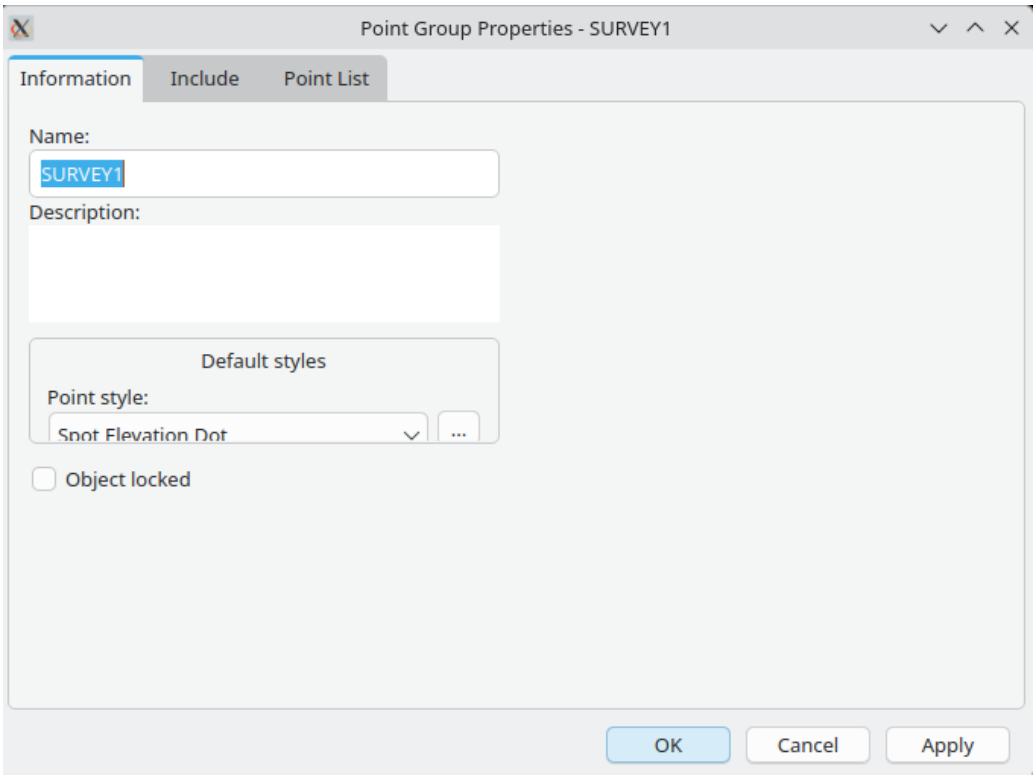


Fig. 1: Information tab: name, description, point style and lock.

Field	What it does
Name	Required, and unique. `_All Points` is reserved. Renaming the group also updates that name on the points and on every surface that uses the group.
Description	Free text stored with the group.
Point style	A style from the drawing's point-style library, under **Default styles**. Points in the group whose own style is **Standard** or **ByGroup** are drawn with it. When a point belongs to several groups, the first group that has a style other than **Standard** supplies the marker. `<none>` stores no style.
…	Opens the Point Style Editor on the style currently chosen. Styles added or renamed there appear in the list when the editor closes.
Object locked	While ticked, the name, description, style and Include rules cannot be edited. Clear the box to change them, then **Apply** or **OK**. A locked group also cannot be deleted.

22.4.2 Point Group Properties — Include

The Include tab decides which points belong to the group. A point is a member when **any** ticked rule matches it. The rules are not combined with "and".

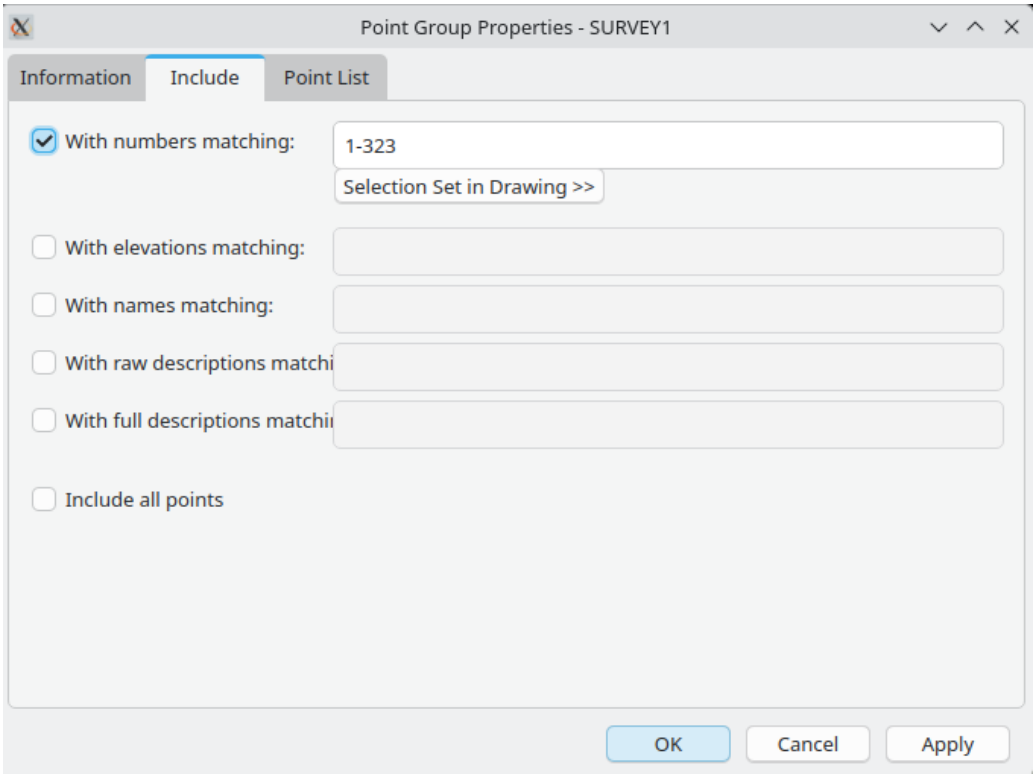


Fig. 2: Include tab: points 1 through 323, by number.

Tick a rule to use it. Its box stays disabled until it is ticked. **Include all points** selects every point in the database and disables the other rules. An empty pattern on a ticked rule matches nothing.

Rule	Pattern
With numbers matching	Point numbers and ranges, separated by commas. `1-323` is every number from 1 to 323. `1-323,400,500-510` adds 400 and the range 500 to 510. A number cannot be negative, and the end of a range cannot be lower than the start.
Selection Set in Drawing >>	Adds the survey points that are **already selected** in the drawing to the number box, compressed into ranges (`1,2,3,7` becomes `1-3,7`). The dialog is modal, so select the points before you open it. If nothing is selected, the button says so and changes nothing. It is available only while the number rule is ticked.
With elevations matching	Tests separated by commas. `>100`, `<200`, `>=100` and `<=200` are comparisons. `100-200` is a closed range, and `150` is that elevation exactly (within 0.000001). The decimal mark is `.`. A leading minus is a negative elevation, so `>-5` and `-2-10` are valid.
With names matching	The point's name, not its number. Comma-separated, ignoring case. `*` matches any run of characters and `?` matches one. `SP-*` matches `SP-001`.
With raw descriptions matching	Same wildcards, against the raw description.
With full descriptions matching	Same wildcards, against the full description.
Include all points	Every point in the database. The other rules are ignored.

OK and **Apply** refuse a pattern they cannot read, switch back to this tab, and leave the group unchanged. A half-typed pattern does not blank the Point List: until it is valid, the list keeps the members the group already has.

A group read from LandXML or DWG often has members and no rules. The dialog then ticks **With numbers matching** and fills the box with those numbers, so opening it does not drop the points.

22.4.3 Point Group Properties — Point List

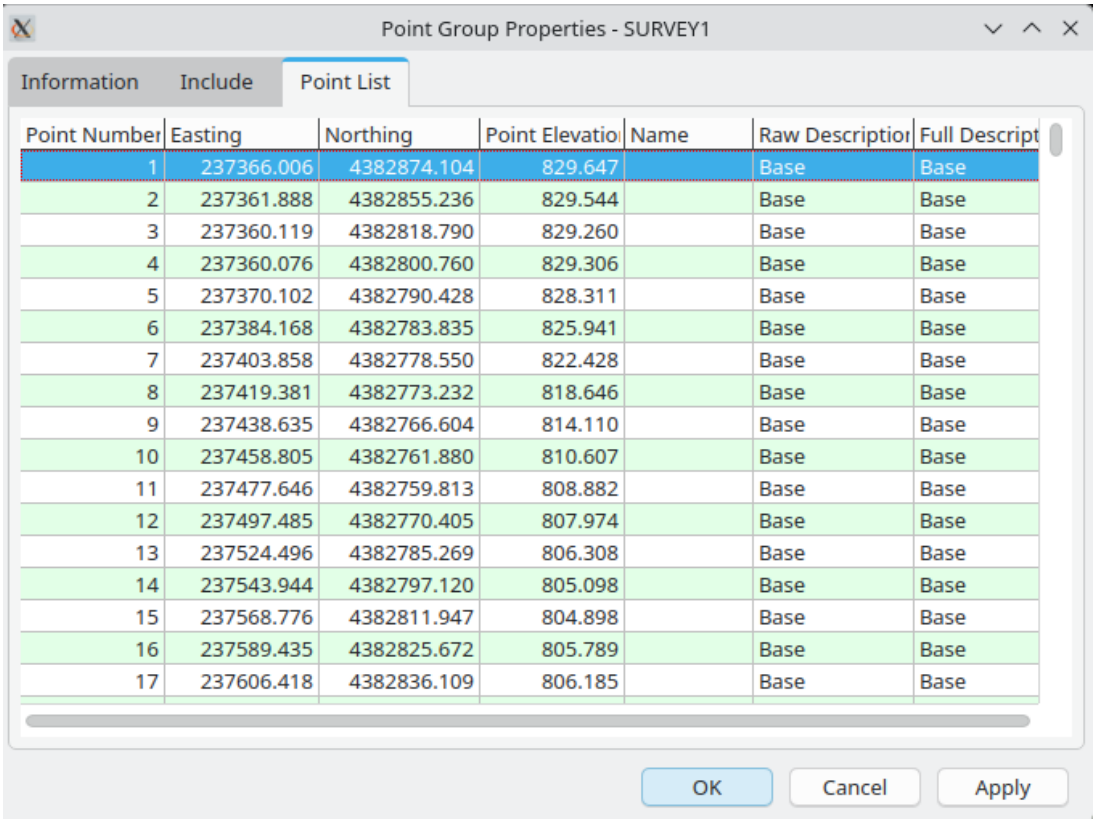


Fig. 3: Point List: the points the Include rules currently select.

The list is a preview, not an editor. It is rebuilt when you open the tab, from the rules as they stand, before you press **Apply**. Rows alternate shade and are sorted by point number. The columns are **Point Number**, **Easting**, **Northing**, **Point Elevation**, **Name**, **Raw Description** and **Full Description**. Coordinates are shown to three decimal places.

From the canvas, without this window:

- **ADDTOGROUP** (`ATG`) adds the selected points to a group you name.
- **Right-click** the selection — the menu lists the existing groups and **New Group…**.
- **REMOVEFROMGROUP** (`RFG`) removes the selected points from a group.
- **CREATEGROUP** (`CRG`) creates a group and lets you click points to add. **Civil › Points › Create Group…** and **Add to Group…** do the same. The Point Manager's group tabs are another way to see one group's points.

22.4.4 Converting AutoCAD Points

The **CONVERTPOINTS** command (alias `CPT`) converts standard AutoCAD point markers into AEC civil survey points. This is useful when importing DXF files that contain point entities with X/Y/Z coordinates but no civil metadata.

Workflow:

1. Select one or more AutoCAD point entities on the canvas.
2. Type `CONVERTPOINTS` (or `CPT`) in the command line.

3. Optionally enter a **group name** to assign all converted points to a point group. Press Enter to skip.
4. Choose whether to **delete the original point entities** after conversion (`Y` / `N`).

Each converted point receives the next available point number and inherits its X/Y/Z coordinates from the original point entity.

22.5 Point Manager

The **Point Manager** is a spreadsheet-style window for viewing, editing, importing, and transforming survey points. Open it with **Civil › Points › Point Manager** or the ribbon button.

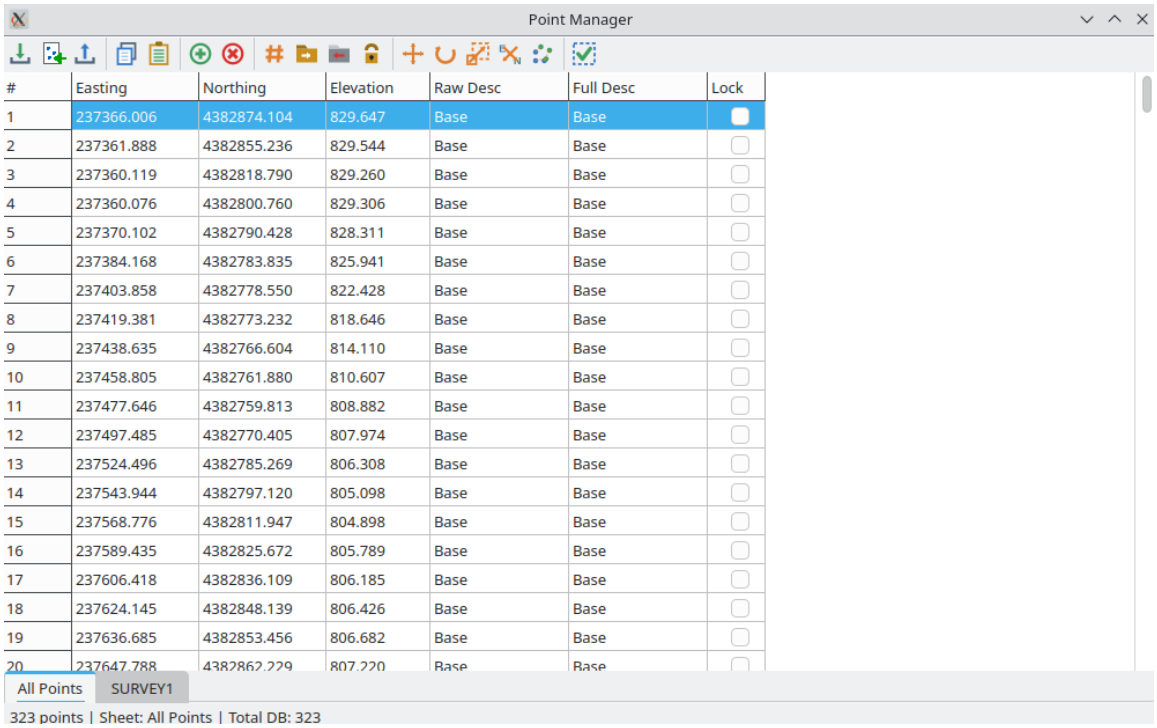


Fig. 4: Point Manager. **All Points** lists the database; each group has its own tab. The status bar shows the sheet count and the database total.

Layout:

- **Toolbar** at the top with all operations.
- **Grid** in the centre — 7 columns (#, Easting, Northing, Elevation, Raw Desc, Full Desc, Lock). All cells are editable; changes are applied immediately to the database and the canvas updates.
- **Sheet tabs** at the bottom — “All Points” shows every point in the database. Each point group appears as its own tab, showing only the points in that group.
- **Status bar** — current point count, active sheet name, total database count.

22.5.1 Toolbar Operations

Button	Description
Import	Opens the CSV/TXT/PTS import dialog. Imported points are added to the database and appear on the canvas.
Import from Drawing	Reads the point groups of another IQcad drawing (`.iqz` or `.ixml`) and lets you tick the groups to bring in. The first entry, **_All Points**, imports every point of the file whether grouped or not. Groups are created or merged by name. A point whose number is already used in this drawing gets the next free number above both drawings; the summary reports how many.
Export	Opens the export dialog. If a group tab is active, only those points are exported; otherwise all points. Supports CSV and Excel (XLSX/ODS) formats.
Copy	Copies the selected rows to the system clipboard as tab-separated text. Can be pasted into Excel, LibreOffice, or a text editor.
Paste	Parses tab-separated text from the clipboard and creates new points in the database. **Only works on the "All Points" tab.** Expected columns: Number, Easting, Northing, Elevation, Description (optional). Header rows and duplicate point numbers are skipped automatically. Ideal for pasting data from Excel or LibreOffice Calc.
Add	Creates a new point at (0, 0, 0) with the next available point number. Edit the coordinates in the grid afterwards.
Delete	Deletes the selected points from the database and the canvas (with confirmation).
Renumber	Opens the renumber dialog to reassign point numbers.
To Group	Assigns selected points to a group (new or existing). A dialog asks for the group name.
Ungroup	Removes selected points from the current group (only available on group tabs, not "All Points").
Lock	Toggles the Lock flag on selected points. Locked points cannot be moved accidentally on the canvas.
Translate	Opens the Translate dialog — shifts selected points by dE, dN, dZ.
Rotate	Opens the Rotate dialog — rotates selected points around a base point.
Scale	Opens the Scale dialog — scales selected points from a base point.
Swap E/N	Swaps the Easting and Northing values of selected points.
Helmert	Opens the Helmert transformation dialog — 2D conformal transformation (translate + rotate + scale) fitted to control point pairs.

22.5.2 Sheet Tabs & Group Management

The tab bar at the bottom of the Point Manager shows one tab per group plus the "All Points" tab which always appears first.

Right-click a tab to open the group context menu:

- **New Group…** — create a named group.
- **Rename Group…** — change the name (not available for "All Points").
- **Delete Group** — remove the group; points are not deleted from the database, only the group membership is removed.

Typical workflows:

1. **Create a group from selected points:** select rows on "All Points", click **To Group**, type a name.

2. **Remove points from a group:** switch to the group tab, select rows, click **Ungroup**.
3. **Paste points from Excel:** copy rows in Excel (columns: Number, Easting, Northing, Elevation, Description), switch to "All Points" tab in the Point Manager, click **Paste**.
4. **Export a group:** switch to the group tab, click **Export**.
5. **Apply a coordinate transform:** select rows, click one of the transform buttons (Translate, Rotate, Scale, Swap E/N, Helmert).

22.5.3 Inline Editing

Click any cell in the grid to edit it directly. Changes are committed when you press Enter or move to another cell. The following columns are editable:

- **Easting, Northing, Elevation** — numeric coordinates.
- **Raw Desc, Full Desc** — text descriptions.
- **Lock** — type `Yes` or `No` (case-insensitive; also accepts `True` / `False` / `1` / `0`).

After editing, the canvas entity is updated immediately via `UpdateSurveyPointsFromDB`. The point number column (#) is not editable from the grid — use the **Renumber** button instead.

23 TIN Surfaces

A TIN (Triangulated Irregular Network) surface models terrain or any 3D surface from a set of survey points. The surface is built using Delaunay triangulation and can display elevation contour lines.

23.1 Triangulation Algorithm

The triangulation engine uses the **Bowyer-Watson incremental Delaunay** algorithm (`UTINTriangulation.pas`), accessed through the `ITINTriangulator` interface for future algorithm swappability.

How Bowyer-Watson works:

1. Create a super-triangle that encloses all input points.
2. Insert points one at a time in randomised order (Fisher-Yates shuffle).
3. For each inserted point, find all existing triangles whose circumcircle contains the new point (these are the "bad" triangles).
4. Remove the bad triangles, leaving a star-shaped polygonal hole.
5. Re-triangulate the hole by connecting the new point to all edges of the hole boundary.
6. After all points are inserted, remove any triangles that share vertices with the original super-triangle.

Performance optimisations for large point sets (100K+):

- **Randomised insertion** — prevents worst-case $O(n^2)$ behaviour on ordered point sets (e.g. grid scans, sorted CSV files).
- **Flat-grid spatial index** — triangles are assigned to grid cells based on their circumcircle AABB. Point insertion only checks triangles in nearby cells instead of scanning all triangles.
- **Auto-scaling grid** — resolution scales as `sqrt(N)` cells per axis, clamped to [16..1024], balancing cell occupancy and lookup cost.
- **Overall complexity** — $O(n\sqrt{n})$ average, compared to $O(n^2)$ without the spatial index.

23.2 Creating a Surface

Use any of the following methods:

- **Civil › Surfaces › Create Surface** or ribbon button **Create Srf**
- Type `CREATESURFACE` (alias `CS`) in the command line
- Right-click **Surfaces** in the Civil Toolspace and choose **Create Surface…**

A surface with an auto-generated name (`Surface_1`, `Surface_2`, …) is created. If survey points already exist in the drawing they are triangulated automatically; otherwise an empty surface definition is created.

23.3 Adding Data to a Surface

An empty surface can be populated in several ways, matching the AutoCAD Civil 3D workflow:

From survey points:

- Type `ADDSURFACEPOINTS` (alias `ASP`) and enter the surface name. All survey points in the drawing are collected and the TIN is built.
- Or right-click the surface name in the Civil Toolspace and choose **Add Survey Points**.

From drawing objects (3DFaces, Lines, Polylines, etc.):

1. Select the source entities on the canvas.
2. Type `ADDSURFACEENTITIES` (alias `ASE`). If only one surface exists it is used automatically; otherwise you are prompted for the name.

Supported entity types:

Entity	How it contributes
3DFace	3 or 4 corner vertices loaded as TIN triangles directly (no re-triangulation). Quads are split into two triangles.
Line	Both endpoints contribute as TIN data points.
Polyline / Spline	All vertices contribute as TIN data points.
Point / Survey Point	Position contributes as a TIN data point.

When the selection contains **only 3DFaces**, triangles are loaded verbatim (preserving the original mesh topology). When non-face entities are present, all collected vertices are triangulated using Delaunay.

You can also right-click a surface name in the Civil Toolspace and choose **Add Drawing Objects…** to add the currently selected entities.

Rebuilding:

After adding or moving survey points, type `REBUILDSURFACE` (alias `RS`) and enter the surface name to re-triangulate from the current point positions.

23.4 Surface Display & Styles

Surface display is controlled through **Surface Styles** — named presets that bundle all visual settings into one reusable object.

Each surface references a style by name via the **StyleName** property in the Object Inspector. Three built-in presets are provided:

Style	Description
Standard	Shows triangles, contours, boundaries, and contour labels.
Triangles & Contours	Triangles and contours visible, boundary hidden.
Contours Only	Only contour lines and labels.
Elevation Banding	Contours and elevation bands visible.
Empty	All components hidden. Use to suppress a surface from display without deleting it.

Open the **Surface Style Editor** via:

- **Format > Surface Styles…**
- Right-click a surface in the Civil Toolspace > **Surface Style…**

The editor has 3 property tabs (Border & Triangles, Contours, Contour Labels) plus a style list for creating/editing/deleting styles.

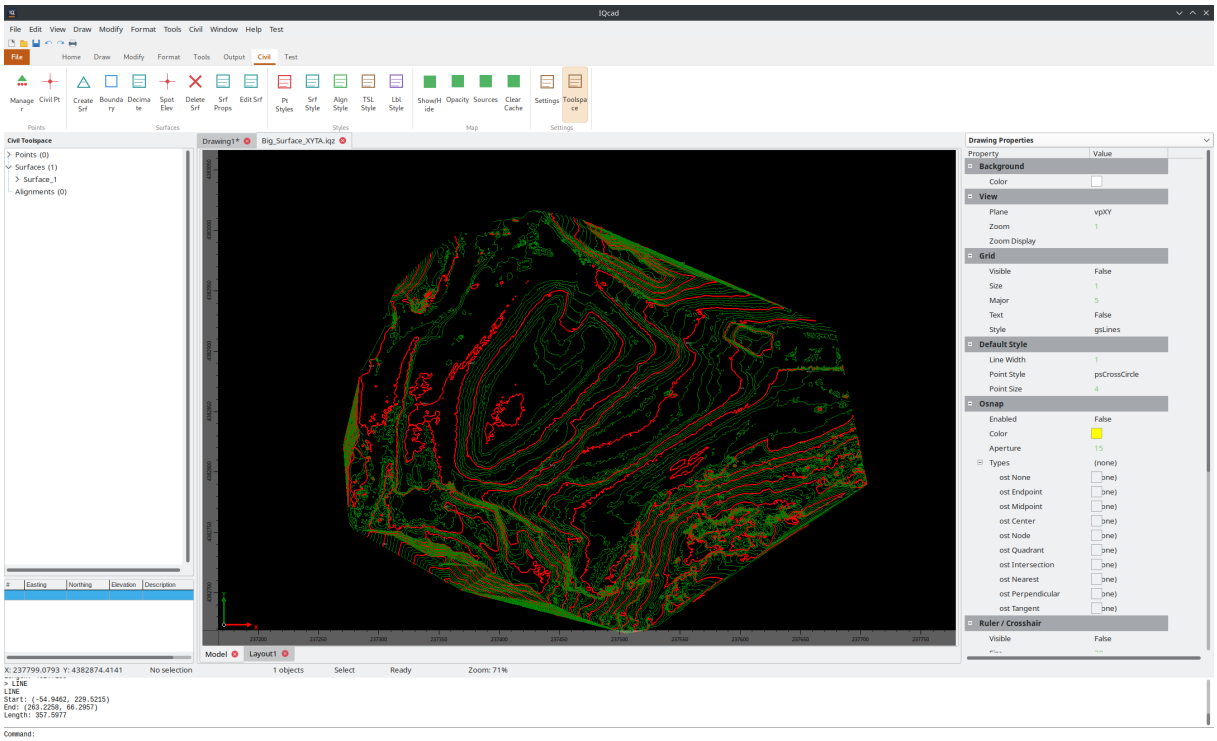


Fig. 1: A TIN surface with contour lines and elevation labels.

Style properties cover 5 groups (~35 settings): border (visibility, colour, weight), triangles (visibility, colour, weight), contours (visibility, major/ minor interval, colours, weights, smoothing, base elevation), contour labels (visibility, interval, text height, precision, prefix/suffix, colour, font, background mask, uphill-readable rotation), and points (visibility, colour, size).

Contour labels are placed automatically along contour polylines with:

- Interval-based placement with collision avoidance
- **Uphill-readable** rotation — label text top faces uphill
- Optional background mask for readability
- Manual label placement via user-placed points

Changing the style immediately regenerates contours if the interval or smoothing parameters differ from the cached values.

23.4.1 Contour Generation Algorithm

1. **Edge marching** — for each triangle, for each contour elevation within its Z range, find the two edge crossings by linear interpolation. Each crossing produces a 2-point segment.
2. **Segment joining** — chain the 2-point segments into continuous polylines by matching endpoints. Uses a **spatial hash table** with linked-list chaining for O(1) endpoint lookup (Fibonacci hashing, QUANT = 1e-6, table size = next power of 2 ≥ 2×N segments). This replaces the naive $O(s^2)$ greedy scan and makes joining nearly instant even for millions of segments.
3. **Chaikin smoothing** — apply corner-cutting subdivision (`ContourSmoothing` passes) to produce smooth curves.

Lazy generation for large surfaces. When `LoadTINData` receives more than 100 000 triangles, contour generation is deferred: `ShowContours` is set to False and contours are only computed when the user first enables `ShowContours` via the Object Inspector. This prevents the UI from blocking during large LandXML imports.

23.5 Boundaries

Boundaries clip the TIN to a specific area. Use **Civil › Surfaces › Add Boundary** or the command `ADDSURFACEBOUNDARY` (alias `ASB`) to add an outer boundary or hole boundary to a surface.

- **Outer boundary** — triangles whose centroids fall outside the polygon are removed.
- **Hole boundary** — triangles whose centroids fall inside the polygon are removed.

23.6 Surface Properties

Right-click a surface name in the Civil Toolspace and choose **Surface Properties…**, or type `SURFACEPROPERTIES`. A selected surface is used at once; otherwise the command asks for the name. `SURFACEPROPERTIES TEXT` prints the short command-line report instead of opening the window (point count, triangles, breaklines, elevation range, area, style, source type and the edit-log count).

The title is **Surface Properties -** followed by the surface name. **OK** writes the Information tab and closes. **Apply** writes it and stays open. **Cancel** drops edits that have not been applied. The Statistics tab is read-only.

23.6.1 Information

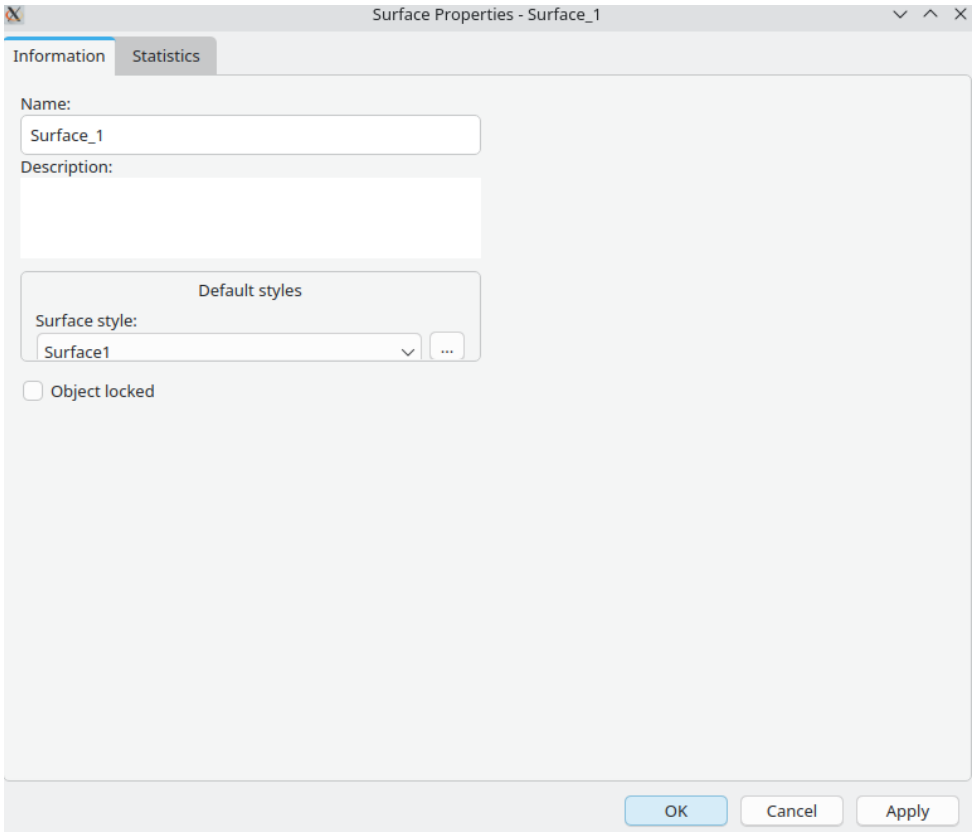


Fig. 2: Information tab: name, description, surface style and lock.

Field	What it does
Name	Required, and unique among the surfaces in the drawing. The tree updates to the new name.
Description	Free text. It is saved with the surface in IQZ and in DWG.
Surface style	The named style that draws this surface, under **Default styles**. The list is the drawing's surface-style library. Changing it and pressing **Apply** or **OK** redraws the surface with that style.
…	Opens the Surface Style Editor on the style currently chosen. If the editor adds, renames or deletes styles, the list is refreshed. The previous choice is kept when it still exists; otherwise the surface's own style, or the first style in the library.
Object locked	A locked surface refuses TIN edits (add, delete or move a point, swap an edge, raise or lower, and the other **Edit Surface** commands). The command line reports *Surface is locked — unlock in Properties to edit*. A locked surface is also left out of a coordinate transformation. Clear the box here and press **Apply** or **OK** to unlock it. Unlike a point group, the other fields on this tab stay editable while the box is ticked.

23.6.2 Statistics

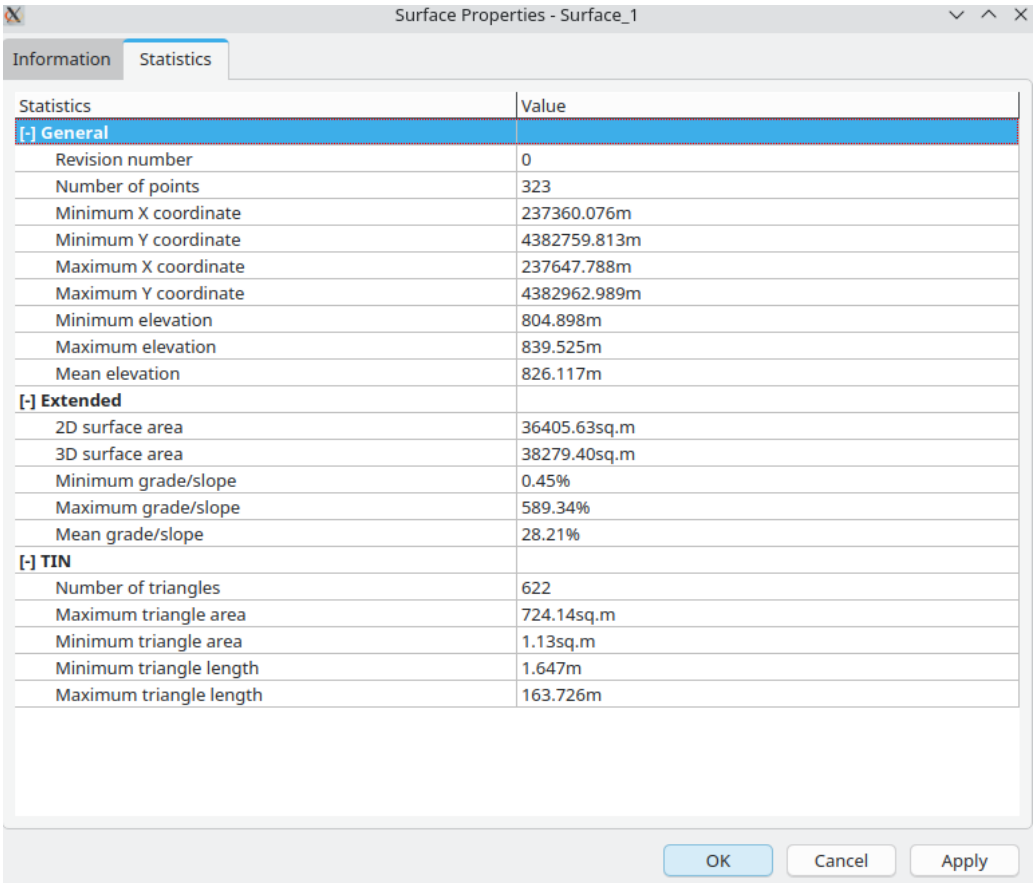


Fig. 3: Statistics tab. Click a bold heading to fold or unfold that group.

The figures follow Civil 3D's Statistics tab for the same surface. Lengths use the drawing's linear unit from Civil Settings: `m`, `ft` or `US ft`. Areas use `sq.m`, `sq.ft` or `sq.US ft`. Lengths and elevations are shown to three decimal places, areas and grades to two. Click a bold **[-]** heading to collapse that group to **[+]**, and click again to open it.

Only triangles that are still in the TIN are counted. A deleted triangle is skipped. Points that no remaining triangle uses are skipped too, which leaves out the four corner points a Civil 3D DWG stores for the extents box.

Statistic	Meaning
Revision number	How many edits are in the surface edit log.
Number of points	Vertices used by the visible TIN.
Minimum / maximum X coordinate	Easting extent of those points.
Minimum / maximum Y coordinate	Northing extent of those points.
Minimum / maximum elevation	Lowest and highest of those points.
Mean elevation	Average of the triangle centroid elevations, weighted by each triangle's plan area. A flat average of the vertices is not used.
2D surface area	Plan area: the sum of the triangle areas in the XY plane.
3D surface area	Sum of the true areas of the triangles, slope included.
Minimum / maximum grade/slope	Steepest and gentlest triangle, as a percent (rise over run × 100). A vertical triangle has no plan area, so it has no slope here.
Mean grade/slope	The same percent, weighted by each triangle's 3D area.
Number of triangles	Triangles in the TIN.
Minimum / maximum triangle area	Plan area of the smallest and largest single triangle.
Minimum / maximum triangle length	Shortest and longest triangle edge, measured in plan, not along the slope.

A surface that has points but no triangles yet reports only the general box and a plain average elevation. The extended and TIN figures stay at zero until the surface is built.

23.7 Surface Statistics

The Object Inspector shows a shorter read-only set. The full list is the **Statistics** tab above.

- **PointCount** — number of TIN vertices.
- **TriangleCount** — number of Delaunay triangles.
- **SurfaceArea** — total 3D surface area. For surfaces exceeding 500 000 triangles the area is estimated by sampling every *N*-th triangle and extrapolating, so the calculation remains instant.
- **MinElevation / MaxElevation** — elevation range.

These statistics are also displayed in the Civil Toolspace grid when the surface node is selected.

23.8 Elevation Query

The surface provides a `GetElevationAt(X, Y)` method that returns the interpolated Z elevation at any (X, Y) position on the TIN. Interpolation uses **barycentric coordinates** inside the triangle that contains the query point — no re-triangulation is needed. This method is used internally by the greedy decimation algorithm and can be called from scripts.

23.9 Elevation Tracker

The **Elevation Tracker** is a live readout of terrain elevation under the mouse cursor, as you move it over the canvas — no picking or clicking required. Toggle it from the **Civil** ribbon or menu (the button stays pressed while active), or type `ELEVTRACK` in the command line. While on, moving the mouse shows a small overlay box near the cursor listing the current elevation of **every visible surface** under that point (a surface hidden by its layer or style is skipped); if several surfaces overlap, all their elevations are listed together so you can compare them at a glance. When more surfaces are under the cursor than the overlay can comfortably show, the extra rows collapse into a "+*N* more…" line.

Point clouds. A second, independent toggle — `PCELEVTRACK` (also on the Point Cloud ribbon tab and menu) — extends the same readout to loaded point clouds: for each visible cloud, it reports the elevation of the single nearest point to the cursor within a small search radius (no averaging or interpolation — a real measured point), sorted with the surface hits in one combined overlay. A cloud with no point close enough to the cursor contributes no row. Turning either tracker off only clears its own rows; the other keeps working independently. See *Point Clouds* (page 233) for more on point cloud display.

23.10 Performance & Rendering

Large surfaces (millions of triangles) use several automatic optimisations to keep the editor responsive:

- **Triangle sub-sampling** — when the TIN exceeds 500 000 triangles, only every *N*-th triangle is drawn (`MAX_DRAW_TRIANGLES = 500 000`). This caps GDI draw calls and keeps panning/zooming smooth.
- **Viewport culling** — triangles whose three vertices all fall outside the same viewport edge (left, right, top, or bottom) are skipped entirely.
- **Sub-pixel culling** — triangles that project to a single screen pixel (all three screen coordinates identical) are not drawn.
- **Lazy contour generation** — when `LoadTINData` receives more than 100 000 triangles (e.g. during a LandXML import), `ShowContours` is set to False and contours are only computed when the user first enables them via the Object Inspector. This prevents the UI from blocking.
- **Surface area sampling** — for surfaces exceeding 500 000 triangles the 3D surface area statistic is computed by sampling rather than iterating every triangle.

No user configuration is required — all thresholds are applied automatically.

23.11 IQML Streaming Save

Surface entities (and all other entity types) are serialised to IQML using a **streaming writer** that writes directly to a text file instead of building an in-memory DOM tree. This eliminates millions of XML node allocations and keeps memory usage constant regardless of drawing size.

- Drawings with 1 000 000+ entities save in seconds instead of minutes.
- The IQML file format is identical — streaming is transparent.

23.12 Decimating (Simplifying) a Surface

Dense TIN meshes — for example from drone photogrammetry, LiDAR, or large LandXML imports — can contain millions of triangles. The **Decimate Surface** command reduces the TIN to a much smaller mesh while preserving the surface shape within a user-specified vertical tolerance.

Use any of the following methods:

- **Civil › Surfaces › Decimate Surface…**
- Ribbon button **Decimate** in the Surfaces pane of the Civil tab
- Type `DECIMATESURFACE` (alias `DS`) in the command line

You will be prompted for:

1. **Surface name** — auto-selected if only one surface exists.
2. **Max elevation tolerance** — the maximum allowed vertical error (in drawing units, e.g. `0.05` for 5 cm).

23.12.1 Decimation Algorithms

The system automatically selects the best algorithm based on the number of input points (threshold: 100 000 points).

Algorithm A — Grid-Based Decimation (> 100K points)

O(n) single-pass algorithm designed for very large surfaces (millions of points). Completes in seconds even for 4M+ points.

1. **Bounding box pass** — compute the XY extent and simultaneously track the 4 boundary extreme points (min-X, max-X, min-Y, max-Y). These replace an expensive O(n log n) convex hull computation.
2. **Grid setup** — divide the XY extent into an aspect-ratio-aware grid. Grid resolution: `N / 50` cells (roughly 2% of input), capped at **50 000** cells to keep the subsequent triangulation fast.
3. **Cell statistics pass** — single pass over all N points. For each grid cell, track the point with the minimum Z and the maximum Z.
4. **Point selection** — for each occupied cell:
   - Always keep the **max-Z** point (preserves terrain highs).
   - Also keep the **min-Z** point if the cell's Z range exceeds the tolerance (preserves valleys, ridges, and terrain features).
5. **Triangulation** — Bowyer-Watson on the reduced point set (typically 30K–80K points).

Algorithm B — Greedy-Insertion Refinement (≤ 100K points)

Higher-quality iterative algorithm where each inserted point is chosen to maximise error reduction.

1. **Seed** — start with a triangulation of the convex hull corners only.
2. **Error computation** — for every omitted point, compute its vertical error against the current (coarse) TIN surface using a grid-accelerated triangle lookup for Z interpolation.
3. **Batch insertion** — pop a batch of the highest-error points from a binary max-heap and insert them all at once, then re-triangulate. Adaptive batch sizes (20–500 points) scale with remaining count.

4. **Error update** — recompute errors for all remaining omitted points against the updated TIN. O(n) bottom-up heap rebuild.
5. **Stop** — repeat until max error ≤ tolerance, or `MaxPoints` reached.

Typical results:

Input	Algorithm	Tolerance	Time	Reduction
4 235 712 pts	Grid	0.10 m	< 5 sec	~99%
500 000 pts	Grid	0.05 m	< 2 sec	~97%
50 000 pts	Greedy	0.02 m	~10 sec	~70%

After decimation the contours and boundaries are automatically regenerated.

23.13 Deleting a Surface

Use **Civil › Surfaces › Delete Surface** or type `DELETESURFACE` in the command line and enter the surface name.

23.14 Importing Large Surfaces (LandXML Fast Path)

LandXML files ≥ 10 MB are imported using a **streaming fast path** that bypasses DOM parsing entirely:

- **Line-by-line reader** (`ULandXMLFastReader`) — reads the file sequentially without building an XML DOM tree in memory.
- **Direct array ID→index mapping** — O(1) point ID lookup instead of `TFPGMap` sorted-list binary search.
- **Inline number parsing** — float and integer values are parsed in-place without allocating `TStringList` per XML element.
- **Memory** — O(points + triangles) only; doubling array growth for amortised O(1) insertions.

The fast path is activated automatically and transparent to the user — the import dialog and workflow are identical. Small files continue to use the standard DOM-based path.

23.15 Exporting Surfaces

- **DXF**: triangles export as `3DFACE` entities; contour lines export as 3D polylines on dedicated layers (`<Name>-Contour-Major` and `<Name>-Contour-Minor`).
- **LandXML**: surfaces export as `<Surface>/<Definition surfType="TIN">` with `<Pnts>` and `<Faces>` elements.
- **IQML**: full round-trip including TIN data, boundaries, contour settings, and source group names.

23.16 Cross-Sections from a DEM

Existing-ground cross-sections can be taken directly from a raster DEM — no triangulation or survey points required. The workflow combines three commands and reuses the standard section-sheet pipeline.

1. Import the DEM. `IMPORTDEM` (alias `IDEM`) reads a GeoTIFF elevation raster (`.tif`) and adds it to the drawing as a *grid-backed surface*. The reader is native — it handles uncompressed, Deflate, and LZW

compression; single-band Float32 or Int16 samples; and both strip and tiled layouts — and georeferences the DEM from the GeoTIFF's own tags (model tiepoint, pixel scale, and CRS), so no world file or external conversion is needed. `NoData` values, voids, and non-finite cells are treated as gaps: the surface returns no elevation there rather than fabricating ground.

A grid-backed DEM surface answers elevation queries by bilinear interpolation directly on the raster grid; it never builds a TIN. Because it still behaves as a surface, it is sampled by section sheets, reported in cut/fill, and picked from the canvas exactly like a triangulated surface. It is drawn as its extent rectangle and is persisted in the drawing by storing its source path; on reopen the `.tif` is re-decoded automatically (a warning is shown if the file has been moved).

> **Note**
>
> The DEM's CRS is reported on import but **not** reprojected. The DEM, the alignment, and any point cloud must already share one projected coordinate system, or the sections will be sampled at the wrong location.

2. Build a comparison surface from the point cloud. To validate the DEM against survey data, **`CLOUDSURFACE`** (alias `CLS`) turns a loaded point cloud into a second grid surface. It bins the cloud's ground-classified points (LAS class 2) into a regular grid, keeping the lowest Z in each cell to estimate bare earth, and names the result `Cloud_<name>`. Cell size defaults to roughly twice the average point spacing. Empty cells stay void, so the comparison surface never invents ground from a single stray return. If the cloud has no ground classification, `CLOUDSURFACE` falls back to all points and suggests running **`PCGROUND`** first to classify ground for a cleaner bare-earth result.

3. Cut the sections. Run **`CREATESECTIONSHEET`**, pick the alignment, then select **both** the DEM surface and the `Cloud_*` surface (the cut/fill prompts appear only when two or more surfaces are chosen). Each station then shows a DEM ground line and a cloud ground line, with cut/fill reported between them — close agreement confirms the DEM, while large gaps flag DEM voids, interpolation artefacts, or vegetation left in the cloud.

Orthophoto backdrop (optional). A georeferenced orthophoto can be draped under the plan for context with **`IMAGEATTACH`**, which reads the image's world file (`.jgw`/`.tfw`) for placement, followed by **`DRAWORDER`** → *Back* to send it beneath the vector geometry. Large JPEG orthophotos are decoded at reduced scale, so a gigapixel image displays as a backdrop instead of a white placeholder.

23.17 Comparing Surfaces (Surface Differencing)

Two surfaces — or a surface and an as-built point cloud — can be compared to produce a **difference (diff) surface** whose elevation at every point is the vertical deviation `dz` between them. This is the basis for earthwork quality control: comparing a theoretical (design) surface against a built or surveyed one shows exactly where material is missing (fill) or in excess (cut), colour-banded by how far each area strays from tolerance.

Two commands open the same **Compare Surfaces** dialog:

- **`SURFDIFF`** — compare two TIN surfaces. **Civil › Surfaces › Compare Surfaces…**, or the **Compare Two Surfaces** ribbon button on the Civil tab.
- **`PCDIFF`** — compare a surface against a loaded point cloud. The **Compare** ribbon button in the Point Cloud pane.

In the dialog:

- **Surface A (theoretical)** and **Source B (built)** — pick the two inputs. A is always a TIN surface; B is a surface (SURFDIFF) or a point cloud (PCDIFF).
- **Cell size (m)** — the diff is computed on a regular grid; smaller cells give finer resolution but more triangles. A live preview reports the overlap extent, the resulting cell count and an estimated build time, and refuses to run if the two inputs do not overlap in plan or the cell count exceeds the safety cap.
- **Tolerance (m)** — deviations within ±tolerance are treated as *on target* (the neutral band). Default 0.20 m.
- **Max deviation (m)** — the outer limit of the colour ramp; areas beyond it saturate to the end colours. Leave 0 to auto-set (5 × tolerance).
- **Sign convention** — $A - B$ (positive = under-build, fill needed) or $B - A$ (positive = over-build). The note under the combo and the colour swatch update live.
- **PC reducer** (point-cloud compare only) — how multiple returns in one cell collapse to a single ground value: *Median* (recommended for noisy scans), *Minimum* or *Mean*.
- **Insert as named entity** — add the result to the drawing as `DIFF_<A>-vs-<B>` (clear it for a statistics-only run).
- **Show statistics after compare** — pop up a summary of valid samples, min/max/mean/RMS dz and cut/fill volumes when the compare finishes.

The result surface carries a **deviation-banded style** (blue → red ramp by default): a neutral band around zero and symmetric cut/fill bands out to the max deviation. It behaves like any surface — pick it, restyle it, sample it — and is saved in the drawing (IQML) together with its comparison provenance (both source names, sign, tolerance, cell size) so the volume report below still works even after the source surfaces are gone.

23.18 Earthwork Volume Report

The **Volume Report** turns a comparison (diff) surface into a printable HTML earthwork report — cut, fill and net volumes, the areas they cover, deviation statistics, and a per-band breakdown. Every figure is integrated directly from the diff surface's own triangulation (each triangle contributes its 2D-projected area × its mean deviation), so the report is self-contained and works on any saved diff surface without the original inputs.

Open it via:

- **Civil › Surfaces › Volume Report…**
- The **Earthwork Volume Report** ribbon button on the Civil tab
- The **Volume Report** action on a selected diff surface in the Object Inspector
- The **View the full volume report?** prompt offered right after a compare (when *Show statistics after compare* is on and the diff was inserted)
- Command **`VOLUMEREPORT`** (alias `VOLREPORT`)

When no surface is passed, the command uses the single selected diff surface, or — when several diff surfaces exist — prompts you to choose one. A surface that is not a comparison surface is rejected with a message. The report opens in the system browser, from where it can be printed or saved to PDF.

The report has four sections:

- **Summary** — the diff surface name, both source names, sign convention, compare timestamp, tolerance, cell size, and grid coverage (valid vs total samples).
- **Volumes** — cut volume, fill volume, net volume (fill − cut), and the cut / fill / total areas, in m^3 and m^2.
- **Deviation statistics** — minimum, maximum, mean and RMS `dz`. These use the compare-time grid samples when available, otherwise the TIN nodes.
- **Deviation bands** — one row per colour band, each with its dz range, a colour swatch, its area, its signed volume, and its share of the total area. The band ranges and colours are read from the surface's on-screen style, so the table matches the map exactly.

Note

Cut and fill are both reported as positive magnitudes; **net = fill − cut** (a negative net means net cut). The band *volumes* are signed (negative for cut bands), so they sum to the net volume, while the band *areas* sum to the total area.

23.19 Balancing Volumes

Balance Volumes edits a design surface so that its cut/fill against existing ground hits a target — most often *net zero* for a pad, or a given fill volume for a landfill or stockpile. The design surface itself is not changed: the edited copy is inserted as a new surface named `BAL_<design>`, so you can run the tool again with another target.

Open it via **Civil › Surfaces › Balance Volumes…**, the Civil ribbon **Surfaces** menu, or the command **`SURFBALANCE`** (alias `BALANCEVOLUMES`). The drawing needs at least two surfaces.

- **Existing ground / Design** — the two surfaces. Only the design moves.
- **Region of the edit** — *Whole overlap*, or *Inside polyline* to confine the edit to a closed polyline (a pad outline, a landfill plateau).
- **Transition to untouched design** — how the edit fades out beyond the inner polyline: *None* (a step, built as a 5 cm taper), *To polyline* (linear fade out to a second, outer polyline — for a pad use the daylight line) or *Width* (linear fade over the given distance).
- **Method** — *Shift design up/down* raises or lowers the region by one value `dZ` (pads). *Scale thickness, toe fixed* multiplies the thickness between design and existing ground by a factor `K`, so the design stays pinned wherever it already meets the ground (landfills, stockpiles, ponds). Scaling cannot balance a surface that has both cut and fill — use Shift there.
- **Target** — the measure and its value in m^3: *Net* (fill − cut; 0 = balanced), *Cut* or *Fill*.
- **Fill factor** — multiplies the fill before it is compared with the target or the cut: above 1 for shrinkage (more bank material per placed m^3), below 1 for swell. The cut is never factored.
- **Cell** — the sampling grid, as for **`SURFDIFF`**.

Press **Compute**: the table shows the current cut, fill, factored fill and net, the solved `dZ` or `K`, and the predicted volumes after the edit. All volumes are totals over the overlap of both surfaces — the region only confines the edit. **Apply** inserts the new surface (with the design's style), recomputes the volumes on it and shows them next to the prediction. Report the recomputed figures: the prediction is made on the sampling grid, the new surface is a TIN, and on a coarse design surface the two can differ by a little along the region edge. Run **Compare Surfaces** on the new surface against existing ground for the full volume report.

Changing any input after a compute clears the prediction and disables Apply until you compute again. When the target cannot be reached the dialog says so and shows the closest result; Apply then asks for confirmation.

23.20 Surface Editing Commands

IQcad provides interactive TIN editing commands for modifying a surface after it has been created. All surface edit operations are logged and can be undone.

Command	Alias	Description
SURFADDPOINT	SAP	Add a single point to the surface TIN interactively.
SURFDELETE-POINT	SDP	Delete a point from the surface TIN.
SURFMOVEPOINT	SMP	Move a surface TIN point to a new position.
SURFMODI-FYELEV	SME	Modify the elevation of a surface point.
SURFSWAPEDGE	SSE	Swap a TIN edge between two adjacent triangles.
SURFADDLINE	SAL	Force an edge between two points in the TIN (breakline).
SURFDELETE-LINE	SDL	Remove a TIN edge.
SURFDELETE-LONGEDGES	SDLE	Delete every TIN edge longer than a threshold you type — the standard way to trim the long slivers a convex hull throws around a concave survey.
SURFHIDETRI	SHT	Hide triangles by clicking inside them; right-click to finish. See *Hiding and Restoring Triangles* (page 162) below.
SURFSHOWTRI	SWT	Restore every hidden triangle on the picked surface.
SURFRAISELOWER	SRL	Raise or lower the entire surface by a specified elevation offset.
SURFUNDO	(none)	Undo the last surface edit operation.
SURFCLEARED-ITS	SCE	Clear the surface edit log.
SURFLISTEDITS	SLE	List all surface edit operations performed since the last clear.
SURFADDBREAK-LINE	SAB	Convert the selected polyline(s) into surface breaklines (forced edges that the TIN must respect). If nothing is selected when the command runs, you are prompted to pick polylines on the canvas.

23.21 Hiding and Restoring Triangles

Triangulation is convex: a TIN built from a survey that is not convex — a road corridor, a river bank, a site with a re-entrant boundary — always spans ground that was never surveyed. A **boundary** clips that away (see *Boundaries* above); hiding individual triangles handles the cases a boundary cannot express, such as a single bad face over a building or a gap between two survey blocks.

SURFHIDETRI (alias SHT) runs as a loop:

1. Pick the surface if more than one is in the drawing.
2. Click **inside** a triangle to hide it. The face disappears from the surface display, contours regenerate around it, and the command line reports the count hidden in this run and in total.
3. Right-click to finish.

A hidden triangle is **kept, not deleted**. It stays in the surface's edit log, is excluded from the displayed and contoured surface, and can be brought back at any time:

- `SURFSHOWTRI` (alias `SWT`) restores every hidden triangle on the picked surface at once and reports how many came back.
- `SURFUNDO` steps back through hide operations like any other surface edit.

Note

Hiding faces changes what the surface **displays**, not the points it was built from. Elevation queries and volume calculations use the surface as displayed, so hiding a face over a building removes it from the volume too — which is normally what you want, and is why the operation is logged rather than destructive.

Civil 3D stores hidden faces the same way, in the triangulation itself. Surfaces read from a Civil 3D DWG come in with their hidden faces already flagged, and faces deleted there are dropped on read.

23.22 Breaklines

A **breakline** is a polyline whose segments are forced into the TIN as constrained edges. Use them to model break-of-slope features — top of bank, bottom of bank, pavement edge, ridge, swale — that the triangulation must follow. A constrained edge is never flipped by a later swap-edge or delete-line operation.

Adding a breakline:

1. Draw a polyline along the feature.
2. Select the polyline.
3. Run `SURFADDBREAKLINE` (alias `SAB`), or right-click the surface's **Definition** node in the Civil Prospector and choose `Add Breakline...`.

The surface stores its own copy of the polyline geometry, so editing or deleting the source polyline afterwards leaves the breakline intact. Each vertex is snapped to an existing TIN point within 1 cm or inserted into the triangulation if no nearby point exists; the segments between consecutive vertices are then forced as constrained edges.

The Civil Prospector tree shows a **Breaklines (N)** node under each surface's *Definition*. Right-click an entry to **Delete** it. Removing a breakline drops its constraints — points it inserted remain in the TIN, but the triangulation is free to flip those edges again.

Visual style is controlled by the *Breaklines* tab in the Surface Style editor: visible toggle, color, weight, linetype, linetype scale. The default colour is magenta so breaklines stand out from triangle wireframe and boundary edges. When the Civil object-layer system is active, the breakline geometry is placed on the dedicated `Surface/Breakline` component layer (default suffix `BRKL` → `C-TOPO-BRKL`).

23.23 Spot Elevations

The **Spot Elevation** tool (`SPOTELEV`, alias `SE`) places an elevation label at a clicked point on a surface. The tool queries the TIN to interpolate the Z value at the click position and displays it as a text annotation on the canvas.

Access via:

- **Civil > Surfaces > Spot Elevation**

- Ribbon button **Spot Elev** on the Civil tab
- Command SPOTELEV (alias SE)

23.24 Contour Label Placement

The **Add Contour Labels** command (ADDCONTOURLABEL, alias ACL) places elevation labels along contour lines of a surface. Labels are positioned automatically with collision avoidance and uphill-readable rotation.

24

Alignments and Profiles

24.1 Horizontal Alignments

A horizontal alignment defines the centreline geometry of a road, railway, pipeline, or other linear infrastructure project. The alignment represents the designed path in plan view (the horizontal plane) and serves as the reference axis for stationing, profile views, and cross-sections.

24.1.1 Geometry Elements

Every alignment is built from a sequence of geometric elements placed end-to-end. The alignment engine computes the precise geometry from **PI (Point of Intersection)** data that you provide.

Element	Description
Tangent	A straight segment connecting two consecutive PIs, or connecting a curve end to a PI. Tangents have constant bearing along their length.
Circular Curve	A constant-radius arc that rounds the angle between two tangents. Defined by the curve **radius** and the **deflection angle** at the PI. The curve produces a smooth transition from one tangent direction to the next.
Spiral (Clothoid)	An Euler spiral where curvature increases linearly with arc length. Clothoid spirals provide a gradual transition between a tangent (zero curvature) and a circular curve (constant curvature), or between two curves of different radii. Specified by either the spiral **length (L)** or the spiral **parameter (A)**.
Spiral (Cubic Parabola)	A transition curve defined by the equation y = x^3 / (6RL). Used in some European road design standards. Like the clothoid, it can be specified by **length (L)** or **parameter (A)**.

A typical alignment follows the pattern: *Tangent - Spiral In - Circular Curve - Spiral Out - Tangent - …* but simpler configurations (tangent-curve-tangent without spirals, or tangent-only) are also supported.

24.1.2 Creating an Alignment

There are two ways to create an alignment: interactively by picking PI points, or by converting an existing polyline.

Method 1 — ALIGNMENT command (interactive PI picking)

1. Type `ALIGNMENT` (alias `ALG`) in the command line, or choose **Civil > Alignments > Create Alignment** from the menu.
2. Click points on the canvas to place each PI (Point of Intersection). The first click places PI #1, the second places PI #2, and so on. A rubber-band preview line follows the cursor.
3. Right-click or press **Enter** to finish. The alignment is created and appears on the canvas with its default style.

The initial alignment has straight tangent segments only (no curves). To add curves and spirals, open the PI Table Editor (see below).

Method 2 — ALIGNFROMPOLYLINE command

1. Type `ALIGNFROMPOLYLINE` (alias `AFP`) in the command line.
2. Click on an existing polyline or line in the drawing.
3. The alignment is created using the polyline vertices as PI points. The original polyline remains unchanged.
4. If the polyline's vertices carry differing elevations (a 3D polyline with real elevations, or a 2D polyline with per-vertex Z), IQcad asks `Create vertical profile from vertex elevations? [Yes/No] <Yes>`. Accepting it also asks for a `Vertical curve length <0>` applied to every interior PVI (0 keeps an angular profile).

Note

When the profile is built from vertex elevations, each PVI is placed at the station where the vertex projects onto the finished alignment, so elevations line up with station queries even where corner curves cut inside the sketched polyline. The profile can be refined afterwards with `DESIGNPROFILE`.

This is useful when you have already drawn the approximate route as a polyline and want to convert it into a civil alignment for further design.

24.1.3 PI Table Editor

The PI Table Editor is the primary tool for fine-tuning alignment geometry. Open it from the **Civil ribbon > Edit PIs** button, or by double-clicking an alignment in the Civil Toolspace.

The editor has two panels:

Left panel — Alignment list:

- Shows all alignments in the drawing with their names.
- **New** — creates a new empty alignment.
- **Delete** — removes the selected alignment.
- **Rename** — changes the alignment name.
- **Import** — imports alignment data from ALM or PSIMO files.

Right panel — PI data grid:

Each row represents one PI. The following columns are available:

Column	Description
PI #	Sequential PI number (read-only).
Easting	X coordinate of the PI point. Editable.
Northing	Y coordinate of the PI point. Editable.
Radius	Radius of the circular curve at this PI (0 = no curve). Editable.
Spiral In	Length or parameter of the entry spiral. Editable.
Type (In)	Spiral type for the entry spiral. Choose from the dropdown: Clothoid L, Clothoid A, Cubic L, Cubic A.
Spiral Out	Length or parameter of the exit spiral. Editable.
Type (Out)	Spiral type for the exit spiral.
Deflection	Deflection angle at this PI (computed, read-only).
Tangent	Tangent distance from PI to curve start/end (computed, read-only).
Curve Length	Arc length of the curve at this PI (computed, read-only).
Station	Station value at this PI (computed, read-only).

After editing values, click **Apply** to rebuild the alignment geometry. The canvas updates immediately to show the new alignment shape.

Spiral types explained:

- **Clothoid L** — specify the spiral by its arc length (L).
- **Clothoid A** — specify the spiral by its parameter (A), where A^2 = R * L.
- **Cubic L** — cubic parabola spiral specified by arc length.
- **Cubic A** — cubic parabola spiral specified by parameter.

24.1.4 Editing vertexes graphically

Five commands edit alignment vertexes directly on the canvas. `ADDPI` (`API`), `DELPI` (`DPI`) and `EDITPI` (`EPI`) work on the alignment itself: pick the alignment (or pre-select it), then click to insert a PI into the nearest tangent segment, click an existing PI to remove it, or click an interior PI to edit its curve radius and spiral (current values shown as the defaults; end PIs have no curve to edit). `ADDPI` first prompts once for a default curve radius — and, if greater than 0, a spiral length and type — and applies those defaults to every PI clicked in the run; press Enter through the prompts for sharp radius-0 PIs. `ADDPVI` (`APV`) and `DELPVI` (`DPV`) work inside a profile view: each click adds a design-profile PVI at the clicked station and elevation, or removes the nearest one. All five repeat until **Enter**, and every click (or edit) is a single undo step. New PVIs are inserted with vertical curve length 0 — refine them afterwards with `DESIGNPROFILE`; PIs added with a default of radius 0 (or curves added later) can be refined afterwards with the PI Table Editor or `EDITPI`.

24.1.5 Alignment Properties

When an alignment is selected, the Object Inspector displays the following properties:

Property	Description
Name	The alignment name, displayed in the Civil Toolspace and used to link profile views and cross-sections to the alignment.
Style Name	The named alignment style that controls all visual aspects of the alignment on the canvas (see Alignment Styles below).
Start Station	Station value at the beginning of the alignment. Change this to shift the stationing (e.g., set to `1000` if the alignment starts at station 1+000).
Total Length	Total alignment length in drawing units (read-only).

24.1.6 Alignment Commands

Command	Alias	Description
`ALIGNMENT`	`ALG`	Create a new alignment by picking PI points interactively on the canvas. Click to place each PI, right-click or Enter to finish.
`ALIGN-FROMPOLYLINE`	`AFP`	Create an alignment from an existing polyline. Click on a polyline to convert its vertices into alignment PIs; optionally builds the vertical profile from a 3D polyline's vertex elevations.
`ALIGNREVERSE`	(none)	Reverse the direction of an alignment. The start becomes the end and vice versa; stationing is recalculated accordingly.
`ALIGNPROPS`	(none)	Display alignment properties (name, length, PI count, element list) in the command line output.

24.1.7 Alignment Styles

Alignment styles control every visual aspect of how an alignment appears on the canvas. Each alignment references a style by name via the **Style Name** property. Edit styles via **Format > Alignment Styles…** or right-click an alignment in the Civil Toolspace and choose **Alignment Style…**.

The style editor has three tabs:

Tab 1 — Centerline & Ticks:

Setting	Description
Centerline visible	Show or hide the alignment centerline.
Centerline color / weight / linetype	Visual appearance of the centerline.
Major tick interval / length	Spacing and length of major station ticks (e.g., every 100 m).
Minor tick interval / length	Spacing and length of minor station ticks (e.g., every 20 m).
Tick color / weight	Visual appearance of station ticks.

Tab 2 — PI & Arrows:

Setting	Description
PI markers visible / color / size	Show PI points as markers on the alignment.
Show PI Details	When enabled, geometry tables are drawn at each PI showing the computed curve parameters (see PI Detail Tables below).
PI lines visible / color / weight / linetype	Show the tangent extension lines through each PI.
Direction arrows visible / interval / color / size	Show arrows along the centerline indicating the alignment direction.

Tab 3 — Labels:

Setting	Description
Text height / color	Base text properties for all labels.
Station labels (major / minor + offset)	Station values at major and minor tick marks.
PI labels (radius, tangent, deflection, spiral length + offset)	Curve parameter labels placed near each PI.
Curve labels (length, chord, degree + offset)	Arc geometry labels placed along curves.
Name label	The alignment name displayed at the start of the alignment.
Station equation labels	Labels at station equation points (if any).

Four built-in styles are provided: **Standard** (centerline, ticks, PI markers and PI lines), **Centerline Only** (ticks, PI markers, and arrows hidden), **Design Phase** (includes PI detail tables, PI lines, and full labels), and **Empty** (all components hidden — use to suppress an alignment from display without deleting it).

24.1.8 Alignment Label Styles

Labels are part of the alignment style (Tab 3 in the style editor). The following label categories can be independently enabled or disabled:

- **Station labels** — placed at major and minor tick positions along the centerline. Major labels show the full station value (e.g., `0+100.000`); minor labels show an abbreviated value. The label offset controls how far the text sits from the centerline.
- **PI data labels** — placed near each PI marker. They can show radius, tangent distance, deflection angle, and spiral lengths. Useful during design to see curve parameters directly on the plan.
- **Curve data labels** — placed along circular curve arcs. They can show curve length, chord length, and degree of curvature.

All label text respects the text height and color settings from the style.

24.1.9 PI Detail Tables

When **Show PI Details** is enabled in the alignment style, geometry tables are drawn at each PI showing the computed parameters:

- **Start / End PIs** — PI number and coordinates (X, Y).

- **Symmetric curves** (equal entry and exit spirals) — single column showing: deflection angle, radius, arc length, spiral parameter (A), spiral length (L), shift (epsilon), and tangent distance (T).
- **Asymmetric curves** (different entry and exit spirals) — two columns showing A1/A2, L1/L2, epsilon1/epsilon2, and T1/T2.

Tables are placed on **alternating sides** of the alignment with leader lines connecting the table to the PI point. Tables **auto-hide** when you zoom out and PIs are too close together on screen, keeping the display uncluttered.

The tables take the colour of the alignment's PI layer, or the style's PI marker colour when that layer is left at its default, so they read on the black model background and on the white sheet of a layout alike. They are part of the plot: a layout exported to PDF draws the same tables, with the auto-hide rule measured on paper instead of on screen.

24.1.10 Import Formats

Alignments can be imported from several external formats:

Format	Description
LandXML	International standard for civil engineering data exchange. Imports horizontal geometry (lines, curves, spirals) and vertical profile. Use **File > Import LandXML**.
ALM	Alignment Solver format. Reads both geometry elements and PI data including spiral parameters. Use the **Import** button in the PI Table Editor.
PSIMO	Greek road design format (`.S01` for horizontal, `.S02` for vertical alignment). PI-based with spiral parameters. Use the **Import** button in the PI Table Editor.

24.1.11 Station Format

Station display format is configurable in Civil Settings (**Civil ribbon > Settings**):

- **Station Index Increment** — `1000` for metric notation (e.g., `0+300.000`) or `100` for imperial notation (e.g., `3+00.000`).
- **Station Precision** — 0 to 4 decimal places.

24.2 Profile Views

A profile view is a graph that shows elevation along the length of a reference line. The horizontal axis represents station (distance along the alignment or polyline), and the vertical axis represents elevation. Profile views are used to visualise existing ground levels, design vertical alignments, and compare surfaces.

Profile views can contain two types of profiles:

- **Sampled profiles** — elevation data sampled from a TIN surface along the alignment at regular intervals. These represent existing conditions (e.g., natural ground, rock surface).
- **Design profiles** — manually defined vertical alignments consisting of PVI (Point of Vertical Intersection) points connected by straight grades and optional vertical curves.

24.2.1 Creating a Profile View

Method 1 — CREATEPROFILEVIEW command (from alignment)

Use this when you have an alignment and want to see how it crosses one or more surfaces.

1. Type `CREATEPROFILEVIEW` (alias `CPV`) in the command line, or choose **Civil > Profile > Create Profile View**.
2. If multiple alignments exist, type the alignment name or click on it. If only one alignment exists, it is selected automatically.
3. Optionally select a surface to sample. Press **Enter** to skip if you only want to show a design profile.
4. Click a point on the canvas to place the profile view.

The command creates a profile view entity at the insertion point. If a surface was selected, the existing ground profile is sampled automatically and displayed as a line in the view. If the alignment has a vertical profile (e.g., imported from LandXML), a design profile is also added.

Method 2 — QUICKPROFILE command (from any line or polyline)

Use this for a quick elevation check along any line or polyline, without needing a formal alignment.

1. Type `QUICKPROFILE` (alias `QP`) in the command line.
2. If multiple surfaces exist, type the surface name or click on it. If only one surface exists, it is selected automatically.
3. Click on a line or polyline in the drawing.

The command samples the selected surface along the line/polyline and creates a profile view near the selected entity. This is the fastest way to see a terrain profile.

24.2.2 Creating a Design Profile

A design profile represents the proposed vertical alignment (the road grade line). To create one inside an existing profile view:

1. Type `DESIGNPROFILE` (alias `DP`) in the command line.
2. Click on a profile view in the drawing to select it.
3. Click points inside the profile view to place PVI (Point of Vertical Intersection) points. Each click defines a station and elevation.
4. Right-click or press **Enter** to finish placing PVIs.
5. For each interior PVI (not the first or last), you are prompted for a **vertical curve length**. Enter `0` for a sharp grade break, or a positive value for a smooth parabolic vertical curve.

The design profile is added to the profile view and drawn as a line.

24.2.3 Profile View Properties

When a profile view is selected, the Object Inspector displays the following properties:

Property	Description
Name	Display name of the profile view.
AlignmentName	Name of the source alignment (read-only for alignment-based views).
Title	Title text displayed above the profile view grid.
StartStation	Start station of the horizontal axis.
EndStation	End station of the horizontal axis.
DatumElevation	Bottom elevation of the vertical axis.
TopElevation	Top elevation of the vertical axis.
ViewWidth	Width of the profile view on the canvas (in drawing units).
ViewHeight	Height of the profile view on the canvas (in drawing units).
VerticalExaggeration	Ratio that stretches the vertical axis relative to the horizontal. A value of 5 means 1 m of elevation is drawn as 5 m on screen. Useful for seeing subtle grade changes on long alignments.
MajorGridX / MinorGridX	Major and minor grid line spacing on the horizontal (station) axis.
MajorGridY / MinorGridY	Major and minor grid line spacing on the vertical (elevation) axis.
GridColor / MinorGridColor	Colors for major and minor grid lines.
ShowAxisLabels	Show or hide station and elevation labels along the axes.

Profile views update automatically when the source alignment PIs are edited or when a source polyline is modified (for quick profiles).

24.2.4 Profile View Commands

Command	Alias	Description
`CREATEPROFILEVIEW`	`CPV`	Create a profile view from an alignment and optionally sample a surface along it.
`QUICKPROFILE`	`QP`	Create a quick profile view by sampling a surface along any line or polyline in the drawing.
`DESIGNPROFILE`	`DP`	Add a design profile (vertical alignment) to an existing profile view by clicking PVI points.

24.3 Alignment Editor

Open with **Civil › Alignment › Edit Alignment PIs…** or the ribbon's **Alignment** dropdown.

The editor has two tabs:

- **Horizontal** — PI table with Easting, Northing, Radius, Spiral In/Out, computed Deflection, Tangent, Curve Length, and Station. Start Station field at the top sets the initial chainage.
- **Vertical Profile** — VPI table with Station, Elevation, CurveLength, computed Grade (%) and K-Value. Add/Delete VPI buttons.

Import — the Import button is the single entry point for alignment files:

- **ALM format** — reads horizontal PIs, vertical VPIs, start station, superelevation, offsets, widenings, cant, and rail data.

- **PSIMO format (.S01)** — reads horizontal and vertical alignment data.

After import, if a vertical profile is found, the editor offers to create a Profile View entity in the drawing.

25

Sections and Parcels

25.1 Cross-Sections

Cross-sections are perpendicular slices through terrain at stations along an alignment. Each section can show **multiple surface profiles** (e.g., existing ground, rock layer, design surface), with cut/fill computed between any two selected surfaces.

25.1.1 Creating a Section Sheet

Use **Civil > Cross-Sections > Create Section Sheet** or type `CREATESECTIONSHEET` (alias `CSS`) in the command line.

Workflow:

1. Select alignment
2. Enter surface names to sample (comma-separated)
3. Set station interval (default 20m)
4. Set left/right widths (default 20m each)
5. Set columns and rows per sheet
6. Optionally select two surfaces for cut/fill comparison
7. Click insertion point

The command generates sample lines along the alignment, samples all specified surfaces at each station, computes cut/fill areas, and creates one or more section sheet entities on the canvas.

25.1.2 Earthwork Volumes

Type `EARTHWORKVOLUMES` (alias `EWV`) to compute and display an earthwork volume table using the **average end area method**.

The table shows for each station: cut area, fill area, cut volume, fill volume, cumulative cut, cumulative fill, and net volume. Totals are printed at the bottom.

25.1.3 Cross-Section Commands

Command	Alias	Description
CREATESECTIONSHEET	CSS	Create a cross-section sheet interactively.
EARTHWORKVOLUMES	EWV	Compute and display earthwork volume table.

25.2 Association Highlighting

When you select a civil entity that references other objects, IQcad highlights the associated entities in light cyan to show their relationship:

- **Selecting a Profile View** highlights the parent alignment, all surfaces sampled by the view, and any path entities used as the profile source. The surface boundary is force-drawn during highlighting even if the surface style normally hides it.
- **Selecting a Section View or Section Sheet** highlights the parent alignment and all surfaces sampled by the sections.

This makes it easy to visually confirm which alignment and surfaces a profile or section view is based on, especially in drawings with multiple alignments and overlapping surfaces.

25.3 Prospector Style Columns

In the **Civil Toolspace** (prospector panel), clicking the **Surfaces** or **Alignments** top-level node displays a summary grid. The grid includes a **Style** column with a dropdown that lets you change the style of any surface or alignment directly from the prospector, without opening the Object Inspector.

25.4 Parcels

A parcel is a closed boundary polygon representing a land area such as a property lot, easement, or right-of-way. Each parcel stores its boundary geometry together with area, perimeter, and user-defined data fields. Parcels participate in the civil entity hierarchy alongside survey points, surfaces, and alignments, and are saved/loaded in IQML and LandXML formats.

25.4.1 Creating a Parcel

Use the CREATEPARCEL command (alias PARCEL) or choose **Civil > Parcels > Create Parcel** from the menu.

Workflow:

1. Type CREATEPARCEL in the command line (or use the menu/ribbon).
2. Enter a **parcel name** when prompted (e.g., Lot-1).
3. Click on a **closed polyline** in the drawing to use as the parcel boundary. The polyline must be closed (first and last vertex coincide, or the Closed flag is set).
4. The parcel is created and appears on the canvas with the active parcel style. The source polyline remains in the drawing.

The parcel entity computes its **area** and **perimeter** automatically from the boundary geometry. These values update if the boundary is edited. Every corner of the new boundary is also assigned a survey-style **vertex name** automatically — see *Parcel Vertex Names* (page 177) below.

25.4.2 Parcel Properties

When a parcel is selected, the Object Inspector displays the following properties:

Property	Description
ParcelName	Display name of the parcel (e.g., `Lot-1`, `Parcel A`).
ParcelNumber	Numeric identifier for the parcel.
StyleName	Named parcel style controlling boundary, fill, and label display.
Area	Computed area of the parcel boundary polygon (read-only).
Perimeter	Computed perimeter of the parcel boundary (read-only).
UserData	Free-form key-value data attached to the parcel. Common fields include Owner, Description, Zoning, LandUse, and TaxID.
Geometry › Vertex	Vertex navigation row (**[<] Vertex N of M [>]**) under a **Geometry** group alongside Area/Perimeter/PointCount. Sub-properties: **Index**, **Name**, **X**, **Y**, **Bulge**. See *Parcel Vertex Names* (page 177).

25.4.3 Parcel Styles

Four built-in parcel styles are provided:

Style	Description
Standard	Boundary line only, no fill or segment labels.
Property	Boundary with semi-transparent fill and bearing/distance labels on each boundary segment.
Easement	Dashed boundary with a light fill pattern to indicate a restricted zone.
ROW	Right-of-way style with a heavy boundary and cross-hatch fill pattern.
Empty	All components hidden (`BoundaryVisible=False`). Use to suppress a parcel from display without deleting it.

Edit parcel styles via the **Civil Style Manager** (Civil > Styles) or **Format > Parcel Styles…**. Style settings include boundary colour, weight, and linetype; fill colour and transparency; label visibility, bearing format (bearing, azimuth, or angle), and segment label text properties.

25.4.4 Parcel Vertex Names

Every parcel corner carries a survey-style **vertex name** — a drawing-wide prefix plus a running number (`A1`, `A2`, `A3`, …). The sequence continues across every parcel in the drawing rather than restarting at each boundary, and a corner shared by two adjacent parcels always carries the **same** name on both sides. Vertex names are meant to double as setout identifiers: they appear on the drawing as labels and in the parcel table and parcel report instead of a bare running index.

Automatic assignment. `CREATEPARCEL` names every unnamed corner of the new boundary as soon as it is created: a corner that coincides with an already-named corner of another parcel adopts that name, and the rest continue the drawing's numbering sequence. Parcels created before this feature shipped keep their corners unnamed until you run `PARCELCHECK` (below) — opening an old drawing never renames anything on its own.

Editing a name. Select a parcel and open **Geometry › Vertex** in the Object Inspector: the vertex navigation row shows **Index**, **Name**, **X**, **Y**, and **Bulge** for the current corner (step through corners with the **[<]/[>]** buttons). Typing a new value in **Name** and committing it:

- Rejects an empty name — the field reverts to its previous value.
- Propagates live to every other parcel that shares that exact corner, so a renamed shared corner stays in sync on both sides.
- Warns on the command line (rather than blocking with a dialog) if the name you typed is already used somewhere else in the drawing — run PARCELCHECK to find and resolve the clash.

Grip-dragging a shared corner. Dragging a corner that two parcels share pulls only the picked parcel's copy — the other parcel's corner stays put. Once the two corners are no longer coincident, the one left behind keeps the original name and the moved corner is issued a fresh number. If you instead drag a corner so it lands exactly on another parcel's named corner, the moved corner adopts that name.

Labels on the drawing. The parcel style's **Labels** tab has a **Vertex Labels** group: a **Vertex Names Visible** toggle and a label-style dropdown that selects which civil label style draws the names (see *Civil Label Styles* in *Format Settings* (page 85)). Vertex name labels are anchored at the corner, offset outward along the angle bisector of the two adjacent boundary segments, matching conventional topographic-diagram placement. An unnamed corner draws no label. Vertex names render both on screen and in PDF plots. The built-in **Property** parcel style has Vertex Names Visible enabled and pairs with the built-in **Parcel Vertex** civil label style (category *Parcels*) to display them.

Coordinate tables and reports. PARCELTABLE and the PARCELREPORT HTML report both show the vertex name in the corner column instead of a bare `1..N` index, falling back to the index for any corner that has no name.

Checking and fixing (``PARCELCHECK``). PARCELCHECK (alias PCHK) is a PEDIT-style check-and-fix hub for parcel vertex issues, with four sub-options:

Option	Action
Report (default)	Read-only. Lists every duplicate boundary vertex, every unnamed vertex, every shared corner whose two sides disagree on the name, and every name that is used at two different locations. Nothing is changed; prints "No issues found." on a clean drawing.
Fix	Repairs everything the report finds: removes duplicate vertices (the original PARCELCHECK behaviour, name-aware so a named vertex always survives the weld over an unnamed one), assigns names to unnamed corners (reusing a coincident neighbour's name first, otherwise the next number), unifies mismatched shared corners, and renames the newer of two vertices that ended up with the same name in different places. Locked parcels are skipped and reported. On first use it also offers to set the drawing's vertex prefix.
reNumber	Full re-sequence, guarded by a confirmation prompt (default **No**) since it is destructive: clears every vertex name in the drawing and reassigns them all from scratch in parcel order, resolving shared corners as it goes. Use it if numbering has drifted and you want a clean, fully continuous sequence.
Prefix	Changes the drawing's vertex-name prefix directly (default A). Prompts for the new prefix with the current one as the default; an empty answer keeps it. Only newly assigned names use the new prefix — existing names are left untouched; run **reNumber** afterwards to re-assign everything under it.

Persistence. Vertex names and the drawing's vertex prefix (default A; any text, including Greek letters, is allowed) are saved with the drawing and round-trip through `.iqz` and through DXF export/import (as

extended entity data on the parcel). Older files with no saved names simply load unnamed.

26

Tunnel Analysis and Coordinate Transformations

26.1 Tunnel Section Line — Measured Profile

A Tunnel Section Line (`TAECC_TunnelSectionLine`) is a measured polyline representing the surveyed cross-section of a tunnel or excavation. It inherits from `TIQCustomObject` (the AEC base class) rather than `TCADPolyline`, placing it in the civil entity hierarchy alongside survey points and surfaces. When linked to a theoretical profile it automatically computes and displays radial deviations.

The entity lives in `UAECC_TunnelSectionLine.pas` under the `cad/Civil/` directory and is serialised as `<TunnelSectionLine>` in IQML.

26.1.1 Creating a Tunnel Section Line

1. Draw the **theoretical profile** as a normal closed polyline on a dedicated layer (e.g. `Theoretical`).
2. Activate **Draw › TSLine** (or type `TSLINE`) and click the measured survey points. Right-click to finish.
3. In the Object Inspector, set **SectionLine** to the theoretical polyline.
4. Enable **ShowDR** to display deviation annotations.

26.1.2 Tunnel Section Line Properties

Property	Description
SectionLine	The theoretical polyline to compare against.
SplineFlag	Renders the measured profile as a smooth spline curve.
SplineSteps	Interpolation steps per segment (higher = smoother curve).
PointSize	Size of the survey point markers along the profile.
ShowDR	Show deviation (DR) annotation lines and values.
DRDecimals	Number of decimal places in DR labels.
DRFontSize	Label font size in world units.
DROffset	Radial offset of the label from the profile line.
DRUnit	Unit for DR values: `mm`, `cm`, or `m`.
DROrientation	`Auto` (follows deviation direction), `Inside`, or `Outside`.
DRAlignment	`Horizontal`, `Vertical`, or `Aligned` (label follows the deviation line direction).
ShowAreaHatch	Fills the area between the measured and theoretical profiles with a hatch pattern.
AreaBetween / AreaInner / AreaOuter	Computed areas (read-only) between the two profiles.

26.1.3 DR Display Modes

The **DRDisplayMode** property controls how deviation labels are rendered:

Mode	Description
dmAll (default)	Shows a deviation label at every measured point.
dmZoneExtremes	Shows only the highest (most extreme) deviation per overbreak and underbreak zone. This reduces label clutter while highlighting the worst-case deviations in each zone.

26.1.4 Deviation Colour Coding

DR annotation lines are colour-coded automatically:

- **Green** — measured profile is *outside* the theoretical (overbreak).
- **Red** — measured profile is *inside* the theoretical (underbreak / intrusion).

Note

Radial sorting is performed clockwise from the bottom (−Y direction), which corresponds to the invert of a tunnel cross-section.

26.1.5 Area Calculation Algorithm

The area computation between the measured and theoretical profiles uses the **Greiner-Hormann polygon intersection** algorithm (`UPolygonClip.pas`).

Step-by-step process:

1. **Polygon densification** — both the measured Tunnel Section Line and the theoretical section are sampled into dense point arrays using `DensifyPoly` with a maximum step of `SampleMaxStep` (default 0.02 m). This ensures that spline curves and bulge arcs are accurately represented as polygons. For the measured profile, `GetWorldPoly` first produces the polyline's own sample points (including Catmull-Rom spline interpolation if `SplineFlag` is enabled); then `DensifyPoly` subdivides any segment longer than `SampleMaxStep`. For the theoretical section, `SampleSectionLine` does the same two-pass process.
2. **Zone clipping** (optional) — if `AreaLevels` is non-zero, both polygons are clipped to the selected quadrant using **Sutherland-Hodgman** rectangle clipping (four successive half-plane clips: Left, Right, Bottom, Top). This allows reporting overbreak/underbreak per tunnel quadrant independently (e.g. crown only, or left wall only).
3. **Area computation** — three polygon areas are calculated:
   - **AM** = `PolygonArea2D(MeasuredPoly)` — measured polygon area (shoelace formula).
   - **AS** = `PolygonArea2D(TheoreticalPoly)` — theoretical polygon area.
   - **AI** = `PolygonIntersectionArea(MeasuredPoly, TheoreticalPoly)` — exact intersection area via Greiner-Hormann.
4. **Result derivation:**
   - **AreaOuter** (overbreak) = AM − AI — area of the measured profile that lies outside the theoretical section.
   - **AreaInner** (underbreak) = AS − AI — area of the theoretical section not covered by the measured profile.
   - **AreaBetween** = AreaOuter + AreaInner — total deviation area.
5. **Numerical guards** — AI is clamped to min(AM, AS) to prevent floating-point over-subtraction; negative results are clamped to zero.

Greiner-Hormann algorithm (six phases):

1. **Build** — normalise both polygons to CCW orientation; build circular vertex linked lists.
2. **Intersect** — find all edge-edge intersections and insert intersection vertices with parametric *t* values along each edge.
3. **Mark** — classify each intersection vertex as Entry or Exit relative to the subject polygon.
4. **Containment check** — if no intersections exist, use `PointInPolygon` (winding number) to detect full containment.
5. **Trace** — walk the intersection-marked lists to extract the output polygon(s); sum their areas.
6. **Free** — release all linked-list memory.

Numeric constants: `GH_EPS = 1e-9` (parallel guard), `GH_CORNER_EPS = 5e-3` (near-vertex rejection), `MIN_REGION_AREA = 1e-4 m²` (sliver filter).

Caching — all area values are cached and only recomputed when the section geometry changes (`FAreaCacheDirty` flag).

Hatch rendering — the `PolygonDifferenceRegions` function traces the same Greiner-Hormann linked lists to extract the actual overbreak (outer) and underbreak (inner) polygon regions, which are then filled with colour-coded hatch patterns (green = overbreak, red = underbreak).

26.2 Coordinate Transformations

IQcad provides two coordinate transformation commands for aligning drawings, survey data, and imported plans to a target coordinate system. Both commands use the IQCalc computation engine and share a common point-picking workflow.

26.2.1 CSTRANSFORM — Global Transformation

Command: `CSTRANSFORM` (alias `CST`)

Computes a single global transformation from common (control) point pairs and applies it to all selected entities. The transformation is solved by **least-squares adjustment**, producing residuals that allow you to identify and remove blunder points.

Supported transformation types:

Type	Min Pts	Description
Helmert 2D	2	4-parameter similarity (conformal): translation, uniform scale, rotation. Preserves angles and shape. The standard choice for local survey alignment.
Affine 2D	3	6-parameter: translation, non-uniform scale (independent X/Y), shear, rotation. Does not preserve angles. Use for scanned plans with differential paper shrinkage.
Polynomial Deg 1	3	6-parameter 1st-degree polynomial (equivalent to affine).
Polynomial Deg 2	6	12-parameter 2nd-degree polynomial for modelling non-linear distortions across large areas.
Polynomial Deg 3	10	20-parameter 3rd-degree polynomial for complex systematic distortions.

Workflow:

1. Select entities to transform (or type `All` for all entities in the drawing).
2. Pick common point pairs:
   - **Click** source point on canvas, then click target point
   - **Type coordinates**: `1000.5,2000.3` or `1000.5,2000.3,100.0`
   - **Type point number**: `#123` (looks up point 123 in the survey database)
   - **Load from file**: type `F` to open a `.cpt` or `.csv` file
3. Type `D` (Done) when all pairs are entered.
4. Choose the transformation type (`H` = Helmert, `A` = Affine, `P1`/`P2`/`P3` = Polynomial).
5. Review the residual table printed in the command line.

6. Save the transformation report (`.trpt`) and common points (`.cpt`) when prompted.
7. Confirm `Y` to apply the transformation.

Residuals and quality control:

When more than the minimum number of points are provided, the system is over-determined and the least-squares solution produces **residuals** at each control point — the difference between the transformed source and the known target. Large residuals indicate:

- A blunder in the control point pair (wrong target entered)
- A point in a locally distorted area that the global model cannot fit

Review the residuals, disable suspect points (re-enter the workflow), and re-solve until the RMS is acceptable. The report file records all residuals, statistics, and the computed parameters for archival.

Mathematical detail — Helmert 2D:

```
Model:   X' = tx + a*X - b*Y
         Y' = ty + b*X + a*Y

where a = scale * cos(rotation),  b = scale * sin(rotation)

Parameters: tx, ty (translations), scale, rotation (4 unknowns)

Each control point provides 2 equations (X' and Y'), so n points give
2n equations.  With n >= 2 the system is solved by least squares:

    A * p = b + v     (design matrix A, parameter vector p, residuals v)
    p = (A'A)^(-1) * A'b       (normal equations)
    RMSE = sqrt( v'v / n )
```

Mathematical detail — Affine 2D:

```
Model:   X' = tx + a11*X + a12*Y
         Y' = ty + a21*X + a22*Y

Parameters: tx, ty, a11, a12, a21, a22 (6 unknowns)
Minimum points: 3  (6 equations for 6 unknowns)

Decomposition:
    Scale X  = sqrt(a11^2 + a21^2)
    Scale Y  = sqrt(a12^2 + a22^2)
    Rotation = atan2(a21, a11)
    Shear    = (a11*a12 + a21*a22) / (Scale_X * Scale_Y)
```

Mathematical detail — Polynomial 2D:

```
Model (degree d):
    X' = SUM( c_ij * x'^i * y'^j )   for i+j <= d
    Y' = SUM( d_ij * x'^i * y'^j )   for i+j <= d

where x' = x - cx, y' = y - cy  (centroid-reduced for numerical stability)

Terms per coordinate:  (d+1)*(d+2)/2
    Degree 1:  3 terms  (1, x, y)
    Degree 2:  6 terms  (1, x, y, x^2, xy, y^2)
    Degree 3: 10 terms  (1, x, y, x^2, xy, y^2, x^3, x^2*y, xy^2, y^3)
```

Warning

Polynomial transformations must not be used for extrapolation beyond the convex hull of the control points. The polynomial may oscillate wildly outside the fitted area (Runge's phenomenon).

26.2.2 RUBBERSHEET — Local Non-Uniform Transformation

Command: RUBBERSHEET (alias RBS)

Applies a spatially-varying correction where the displacement at each point depends on its proximity to nearby control points. Unlike CSTRANSFORM, there is **no single global model** — the transformation varies continuously across the drawing.

Two modes:

Linear (Delaunay Piecewise Affine)

The source control points are triangulated using the Bowyer-Watson Delaunay algorithm. For each triangle, a unique affine transformation is computed that maps its three source vertices exactly to their three target positions (6 equations, 6 unknowns — exact solution). For any query point, the algorithm locates the containing triangle and applies that triangle's affine.

```
For each triangle T_k with source vertices (S0, S1, S2)
mapping to target vertices (T0, T1, T2):

    X' = a11*X + a12*Y + tx
    Y' = a21*X + a22*Y + ty

Solved by Cramer's rule:
    D = x0*(y1-y2) + x1*(y2-y0) + x2*(y0-y1)
    a11 = (X'0*(y1-y2) + X'1*(y2-y0) + X'2*(y0-y1)) / D
    ... (similarly for a12, tx, a21, a22, ty)

Point location uses barycentric coordinates:
    Q is inside triangle ABC if u >= 0, v >= 0, (1-u-v) >= 0
```

Properties: C0 continuous (slope breaks at triangle edges), exact at control points, O(n log n) triangulation, brute-force O(m) point location.

Smooth (Thin Plate Spline)

The TPS minimizes the bending energy of a thin metal plate passing through prescribed displacements. Two independent TPS functions are solved — one for X displacement, one for Y displacement.

```
f(x,y) = a0 + a1*x + a2*y + SUM_i( w_i * U(r_i) )

where U(r) = r^2 * ln(r)  (radial basis function)
      r_i  = distance from (x,y) to control point i

Linear system (n+3 x n+3):

    | K   P | | w |   | v |      K[i,j] = U(|Pi - Pj|)
    | P'  0 | | a | = | 0 |      P[i]   = [1, xi, yi]

The constraint P'*w = 0 ensures affine far-field behavior:
far from all control points, f reduces to a0 + a1*x + a2*y.
```

Properties: C2 continuous everywhere, exact at control points, O(n^3) solve, O(n) per-point evaluation. Natural affine extrapolation in far field.

Workflow:

1. Select entities (or `All`).
2. Pick common point pairs (same methods as CSTRANSFORM).
3. Type `D` (Done).
4. Choose mode: `L` = Linear (default), `S` = Smooth.
5. Review displacement table and save report/points when prompted.
6. Confirm `Y` to apply.

Key differences from CSTRANSFORM:

- **No residuals** — rubber sheeting interpolates exactly through all control points, so there is no over-determined system and no residual analysis. Blunder points cannot be detected from the math — verify independently.
- **Spatially varying** — different parts of the drawing are corrected differently based on proximity to control points.
- **Outside convex hull** — points outside the control network are extrapolated (linear: nearest triangle, smooth: affine). A warning count is reported.

Elevation (Z) handling:

Both CSTRANSFORM and RUBBERSHEET preserve the original Z coordinate by default — only Easting (X) and Northing (Y) are transformed.

26.2.3 Common Points File Format (.cpt)

Common point pairs can be saved and loaded in JSON format:

```
{
  "version": 1,
  "description": "Site A to Grid transform",
  "coordinateOrder": "ENZ",
  "points": [
    {
      "name": "CP1",
      "source": { "e": 1000.000, "n": 2000.000, "z": 100.000 },
      "target": { "e": 1000.152, "n": 2000.318, "z": 100.023 },
      "enabled": true
    }
  ]
}
```

Alternatively, CSV format is supported:

```
Name,SrcE,SrcN,SrcZ,TgtE,TgtN,TgtZ,Enabled
CP1,1000.000,2000.000,100.000,1000.152,2000.318,100.023,True
```

26.2.4 Transformation Report (.trpt)

The `Save Report` dialog writes a detailed text report including:

- Transformation type, date and the model's equations
- Common point table with source/target coordinates
- Computed parameters with standard deviations: translations, scale and rotation (Helmert), the matrix with derived scales, rotations and shear (Affine), the centroid and coefficients (Polynomial)
- The inverse transformation (Helmert and Affine)
- Computed coordinates and residuals at each control point (computed minus target)
- Disabled points as check points, measured against the solution
- RMS statistics, standard error of unit weight, redundancy and degrees of freedom
- For rubber sheet: the triangles and their parameters (Linear) or the spline's parameters and weights (Smooth), and the displacement table with how far each control point ends up from its target
- Number of entities selected and locked points skipped
- IQCalc engine version

26.2.5 Practical Guide: Which Transformation to Use

Choosing the right transformation depends on the nature and magnitude of the coordinate differences, the number and distribution of control points, and whether the distortion is uniform or spatially varying.

Decision Flowchart

```
Do you have common points in both systems?
 │
 ├─ No → You cannot compute a transformation.
 │        Obtain at least 2 common points first.
 │
 └─ Yes
     │
     Is the distortion uniform across the area?
     (i.e., a simple shift/rotate/scale explains the differences)
      │
      ├─ Yes → Use CSTRANSFORM
      │    │
      │    How many common points?
      │    ├─ 2     → Helmert 2D (exact fit, no residuals)
      │    ├─ 3-5   → Helmert 2D (least-squares, with residuals)
      │    ├─ 3+    → Affine 2D (if X/Y scales differ)
      │    └─ 6+    → Polynomial deg 2 (if residuals show spatial pattern)
      │
      └─ No → The distortion varies across the area
           │
           Use RUBBERSHEET
           ├─ Many control points (>20) → Linear mode
           └─ Few control points (<20)  → Smooth (TPS) mode
```

Common Survey Scenarios

Scenario 1: Aligning a total station survey to control points

You have a local total station survey (arbitrary setup) and 3-5 points with known coordinates in the national grid.

Recommended: **CSTRANSFORM → Helmert 2D**

This is the classic case: the local survey differs from the grid by a translation, rotation, and uniform scale. Helmert preserves angles (conformal) which is essential for survey data.

```
Workflow:
1. Import your total station points into IQcad
2. Run CSTRANSFORM
3. For each control point: click the local position, then enter the
   known grid coordinate (e.g. #101 or 538345.12,4872341.56)
4. Type D (Done), select H (Helmert)
5. Check residuals — all should be < 0.01 m for a good survey
6. If one point has a much larger residual, investigate (blunder?)
7. Apply
```

Typical RMS: 3-10 mm for a well-executed survey.

Scenario 2: Georeferencing a scanned cadastral plan

You digitised an old cadastral plan and need to bring it into the national grid. The paper has shrunk non-uniformly (different scale in X and Y).

Recommended: **CSTRANSFORM → Affine 2D** (minimum 3 points)

Affine allows independent X and Y scales and a shear component, which models paper shrinkage. Use at least 4 points for redundancy.

```
Workflow:
1. Identify 4+ points on the plan that you can also locate in the
   national grid (road intersections, property corners, benchmarks)
2. Run CSTRANSFORM, pick the digitised positions and enter grid coords
3. Select A (Affine)
4. Check residuals — if they show a spatial pattern (e.g., all positive
   in one corner), the plan may have local folds → use RUBBERSHEET
```

Typical RMS: 0.1-0.5 m depending on plan age and condition.

Scenario 3: Aligning an imported DXF from a different coordinate system

A colleague sent you a DXF in a local grid, and you need to merge it with your project in the national grid. You have 2-3 matching points.

Recommended: **CSTRANSFORM → Helmert 2D** (2 points for exact fit, 3 for QC)

With exactly 2 points, the Helmert gives an exact solution (no residuals). If you have 3 points, use all three — the third point's residual tells you whether the two coordinate systems are related by a simple similarity transform. If the residual is large, the systems may have different scales in X and Y (→ try Affine) or local distortions (→ try RUBBERSHEET).

Scenario 4: Correcting a drawing with known local distortions

Your drawing has localized errors: some areas are shifted by 0.2 m, others by 0.5 m in different directions. You have 10+ control points scattered across the project.

Recommended: **RUBBERSHEET → Linear mode**

The Delaunay triangulation adapts the correction locally — each triangle gets its own affine transform, so areas near a shifted control point are corrected by that point's displacement. With 10+ points the triangles are small enough that the C0 discontinuities at edges are negligible.

```
Workflow:
1. Select All entities
2. Pick all 10+ control point pairs
3. Choose L (Linear)
4. Review the displacement table — are the magnitudes consistent?
5. Check the "outside hull" count — if high, add control points at
   the project boundary
6. Apply
```

Scenario 5: Deformation monitoring — comparing two epochs

You have two sets of measurements of the same points at different times and want to visualise the displacement field smoothly.

Recommended: **RUBBERSHEET → Smooth (TPS) mode**

TPS produces a C2-continuous displacement field ideal for deformation visualisation. The smooth gradient avoids artificial slope breaks that would appear with linear rubber sheeting.

Scenario 6: Large-area datum transformation (hundreds of km)

You have survey data in a local datum and need to convert to a national or global reference frame. The area is large enough that a simple 4-parameter transform leaves systematic residual patterns (bowl-shaped or tilted).

Recommended: **CSTRANSFORM → Polynomial degree 2** (minimum 6 points)

The 2nd-degree polynomial captures regional curvature that Helmert/Affine cannot model. Place control points at the edges and centre of the area.

> **Warning**
>
> Never extrapolate polynomial transforms beyond the control network. Outside the convex hull, the polynomial may diverge rapidly.

Scenario 7: Quick approximate shift (no rotation/scale)

You just need to shift your drawing by a known offset (e.g., "everything is 100 m too far East").

Recommended: **CSTRANSFORM → Helmert 2D** with 1 point pair.

Although Helmert needs 2 points, with 1 point it effectively computes a pure translation (scale = 1, rotation = 0). Alternatively, use the MOVE command with a known delta.

Quality Control Checklist

After any transformation, verify the result:

1. **Check the RMS** (CSTRANSFORM only). Compare with your expected survey accuracy. RMS >> expected accuracy suggests blunders or wrong model.
2. **Examine individual residuals**. A single large residual (> 3× RMS) suggests a blunder in that control point. Disable it and re-solve.

3. **Look for spatial patterns** in the residuals. If positive residuals cluster in one area and negative in another, the model may be too simple → try a higher-order transform.
4. **Check the outside-hull count** (RUBBERSHEET). A large number means many entities are being extrapolated — add boundary control points.
5. **Spot-check transformed points** against known coordinates not used in the transformation (independent check points).
6. **Save the report and common points** for audit trail and repeatability.
7. **Verify locked points** were correctly skipped (check the "locked skipped" count in the output).

Control Point Distribution Tips

Guideline	Reason
Surround the project area	Minimises extrapolation; rubber sheet needs enclosing hull
Distribute points evenly	Avoids poorly conditioned least-squares solutions
Avoid collinear arrangements	Collinear points cannot define 2D transforms (singular matrix)
Use more than the minimum	Extra points give redundancy for residual-based QC
Include points at different elevations	Important for 3D transforms (Bursa-Wolf)
Remove one point at a time	When testing outliers, remove one, re-solve, then evaluate
Keep point names consistent	Helps match pairs when loading from files

27

Assemblies and Corridors

The corridor system turns an alignment plus a parametric cross-section (“assembly”) plus a ground surface into a 3D road model with cut/fill volumes, top and datum surfaces, and feature lines.

27.1 Overview

An **assembly** is a named cross-section template containing left and right rows of pre-built road elements: lanes, shoulders, curbs, sidewalks, medians, slopes. Each drawing has its own *Assembly Library*, accessed via **Civil › Assemblies…**.

A **corridor** is a binding between one alignment, one target surface (an existing-ground TIN), and one assembly. The **Build Corridor** command evaluates the assembly along the alignment and produces 3D outputs in the drawing.

You need three things to build a corridor:

1. An **alignment** with horizontal geometry and a vertical profile.
2. A **target surface** (TIN) representing the existing ground.
3. An **assembly** describing the cross-section.

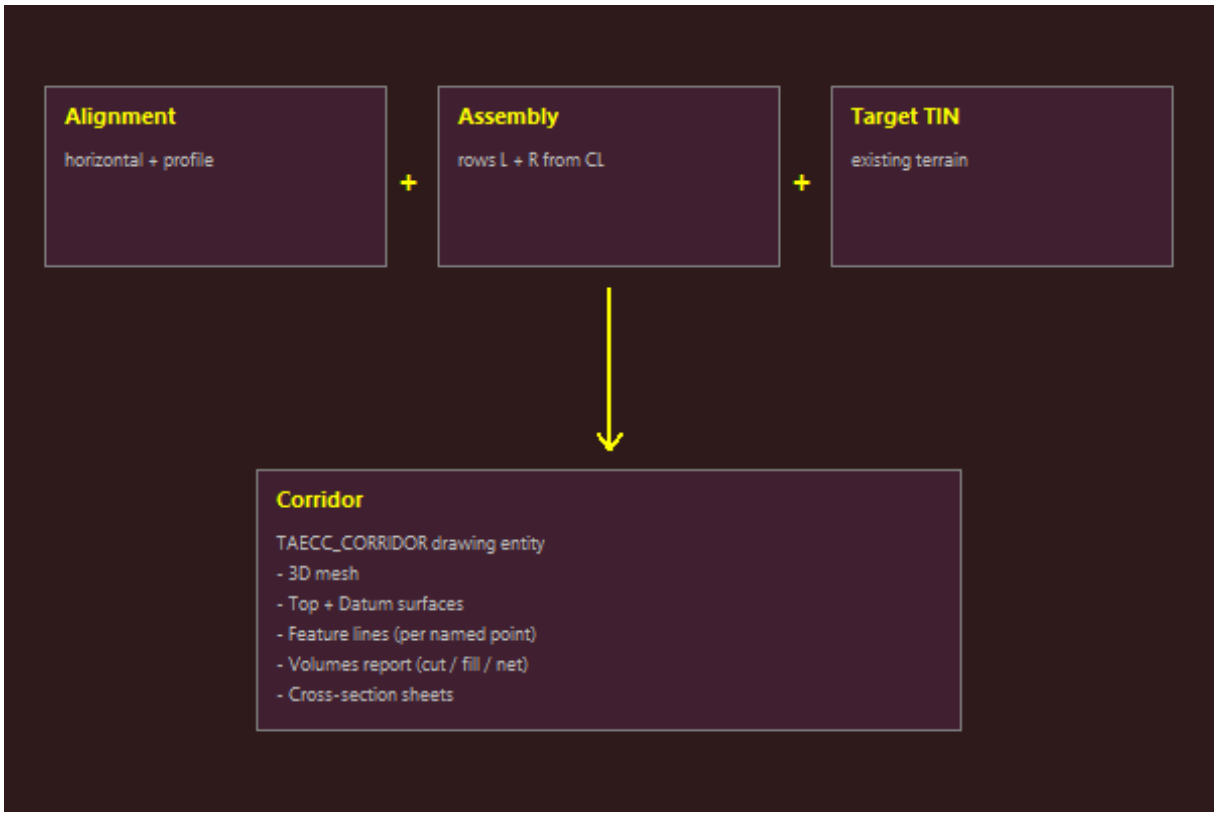


Fig. 1: Three inputs (alignment, assembly, target TIN) combine into a single corridor entity that produces six output artifacts.

27.2 Assembly elements

Seven row types cover the building blocks of a typical road cross-section. Rows are added in order from the centerline outward on each side via the row-palette buttons in the Assembly Editor.

27.2.1 Lane

A drivable carriageway segment. Parameterised by width, cross-slope (superelevation-aware), and pavement depth. The pavement depth drives the datum surface — what sits under the road.

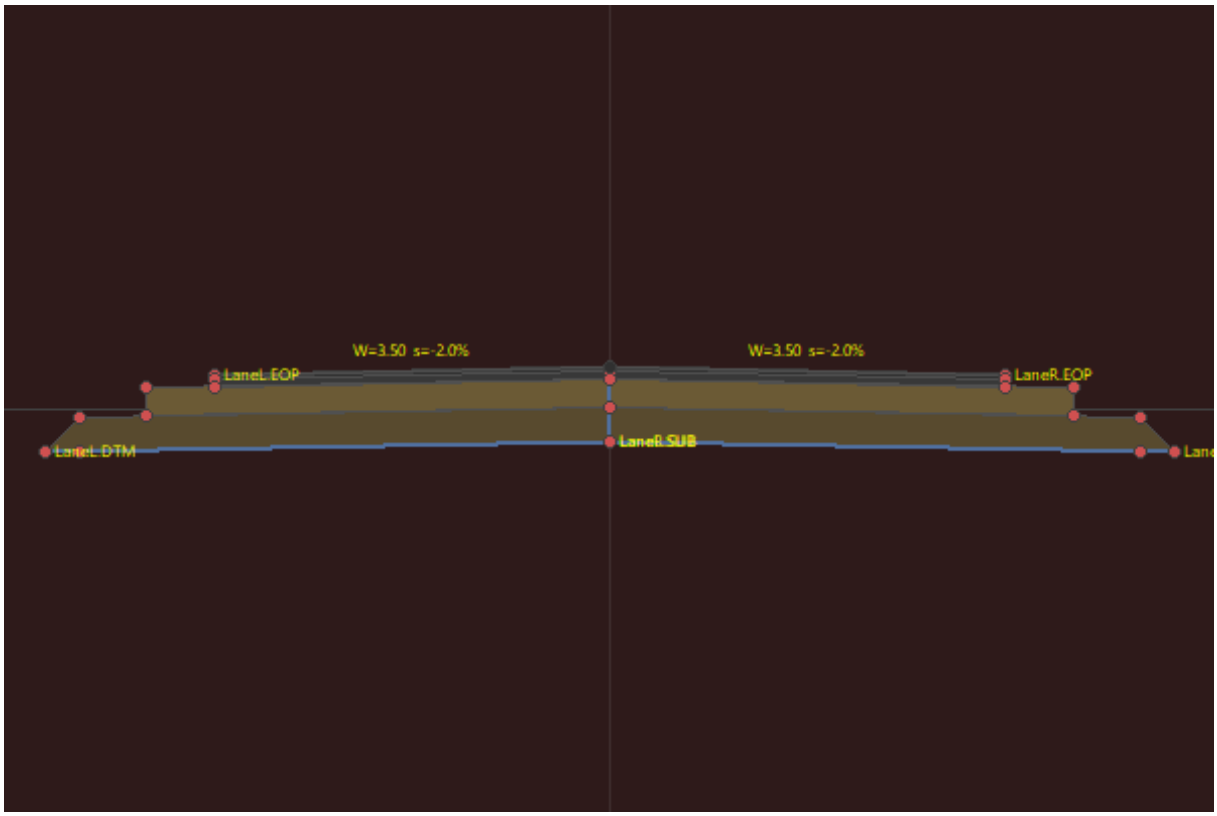


Fig. 2: Lane element schematic.

Parameter	Description
Width	Lane width in metres, measured from the parent point outward.
CrossSlope	Signed cross-slope as a decimal fraction. Negative means falling away from CL. Receives the alignment's superelevation when `Peraltable=True`.
PaveDepth	(v1 legacy — migrated to a single course on load; use **Courses…**) Vertical pavement thickness in metres. In the current model the pavement is a **course stack** — see "Pavement structure — courses and materials" below.
Peraltable	When True, the cross-slope is overridden by the alignment's SuperelevationLeft / SuperelevationRight value at each station.

When to use it. Every road has at least one lane per direction. For divided roads, combine with a Median element between the opposing directions.

27.2.2 Shoulder

A narrower paved strip outboard of the outermost lane, typically sloped more steeply away from CL to shed water. Mechanically a narrower Lane with its own layer and label, named differently so quantities and feature lines distinguish it from the carriageway.

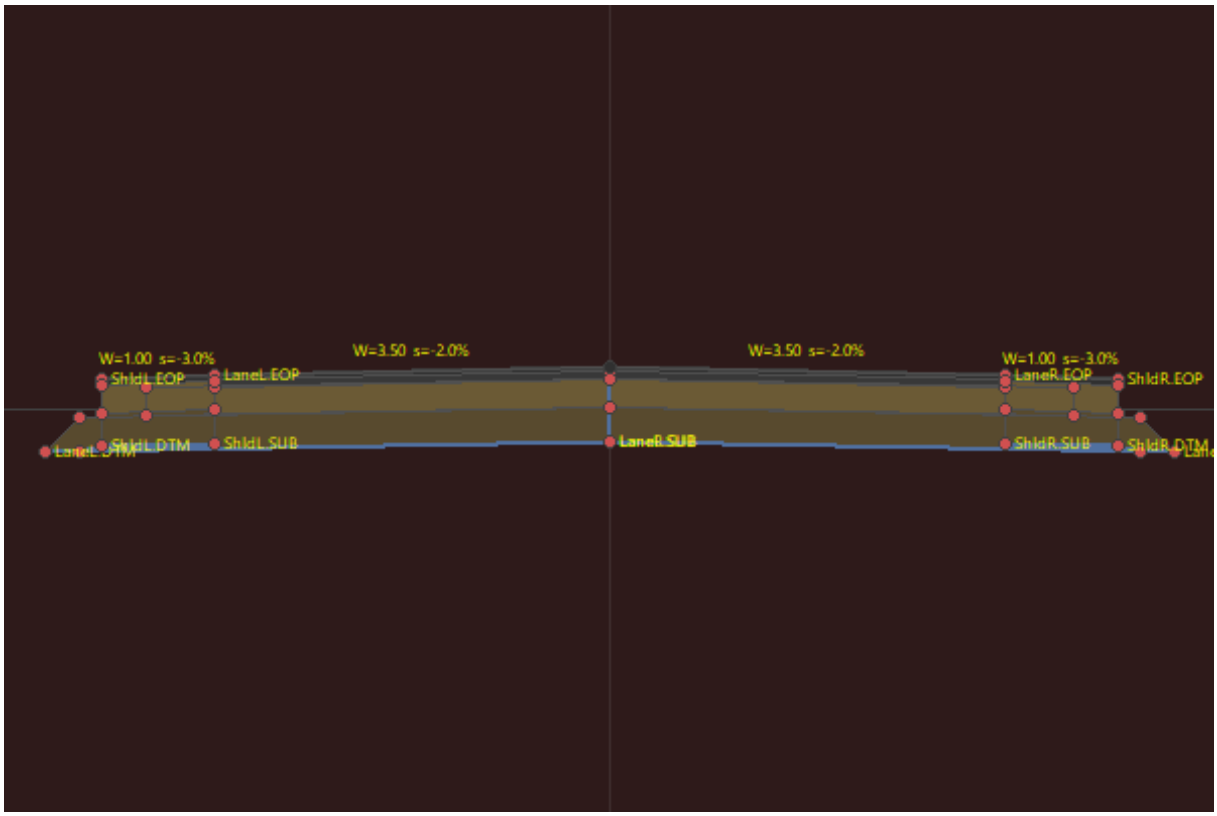


Fig. 3: Shoulder element schematic (with parent Lane).

Parameter	Description
Width	Shoulder width in metres. Typical 0.5–2.5 m depending on road class.
CrossSlope	Signed cross-slope, typically steeper than the lane (e.g. −3%).
PaveDepth	(v1 legacy — migrated to a single course on load; use **Courses…**) Pavement thickness, usually thinner than the lane (e.g. 0.05 m).
Peraltable	When True, follows the alignment's superelevation.

When to use it. Rural and highway designs almost always have shoulders. Urban designs typically use a Curb + Sidewalk instead.

27.2.3 Median

A divider strip at the centerline of a divided road, typically depressed below the running surface for drainage. Sits at the pivot.

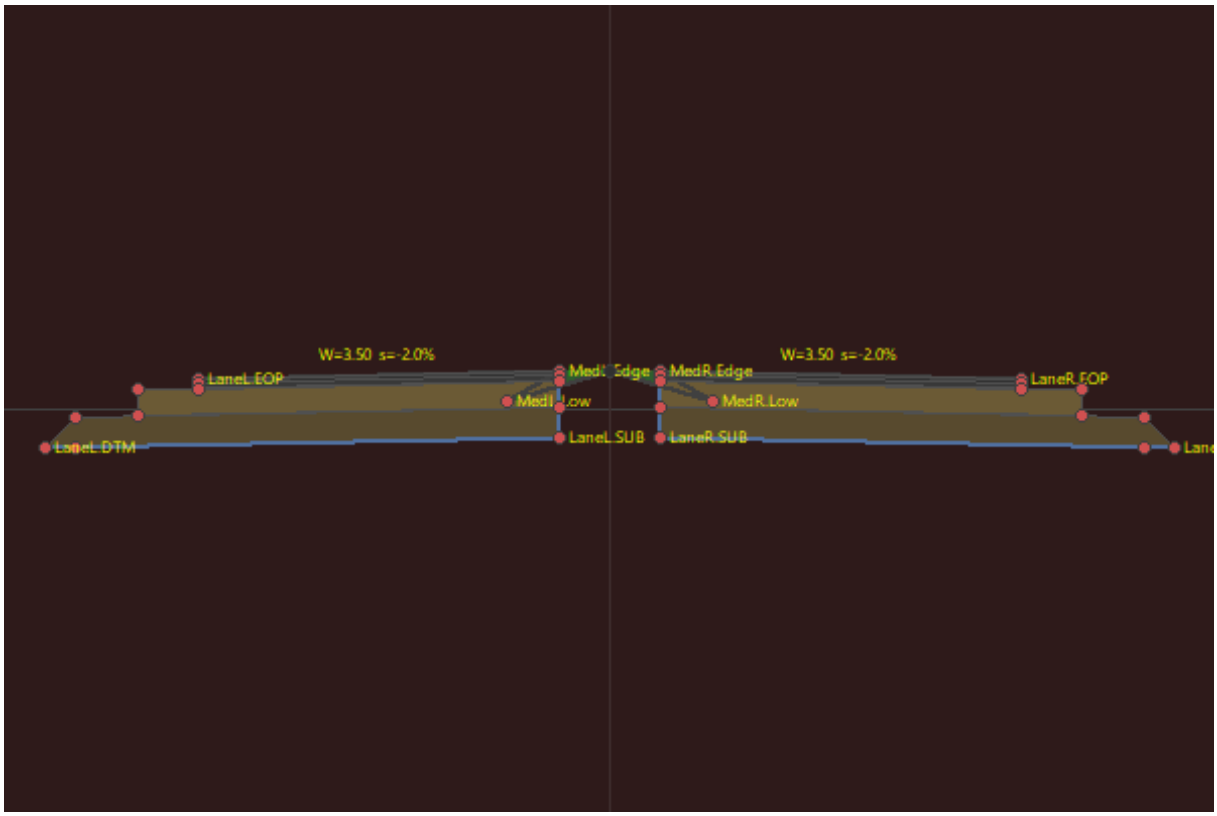


Fig. 4: Median element schematic.

Parameter	Description
HalfWidth	Half the total median width in metres (the full median is 2 × HalfWidth across both sides).
Depth	Vertical depression below the lane level in metres.
InnerSlope	Slope of the inner face of the median (V/H).

When to use it. Divided highways and urban boulevards. Place the Median as the first row on each side so the lanes that follow sit on its outer edge.

27.2.4 Curb

An urban kerb — a stepped concrete profile separating the carriageway from the sidewalk. Three points define the profile: foot (at lane edge), top (raised), back (where the sidewalk begins).

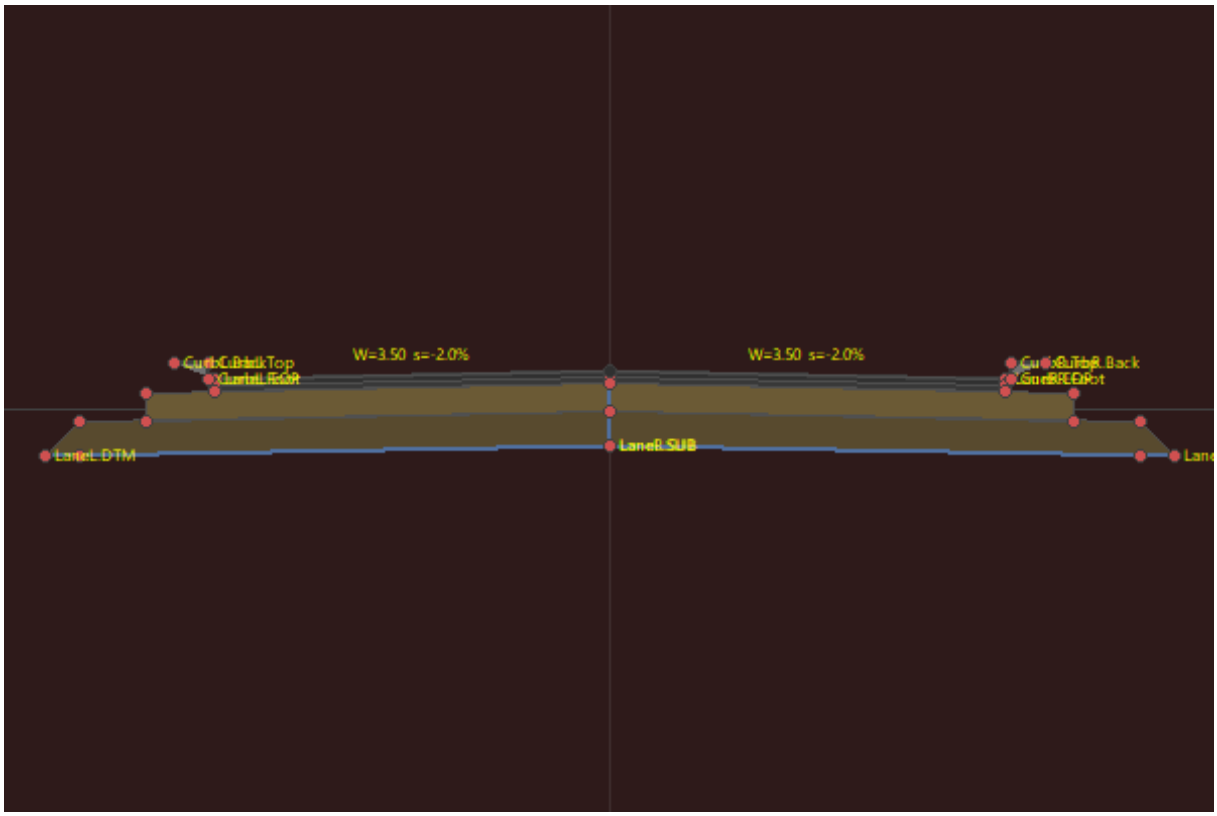


Fig. 5: Curb element schematic (with parent Lane).

Parameter	Description
FaceW	Horizontal width of the curb face in metres (typically 0.02–0.10 m).
FaceH	Vertical face height in metres (typically 0.10–0.20 m).
TopW	Top width in metres (the flat top of the kerb stone).

When to use it. Urban streets with sidewalks. Combine with a Sidewalk row immediately after the Curb.

27.2.5 Sidewalk

A pedestrian walking surface outboard of the curb. Mechanically similar to a Lane (width + cross-slope + depth) but typically with gentler slopes and shallower pavement.

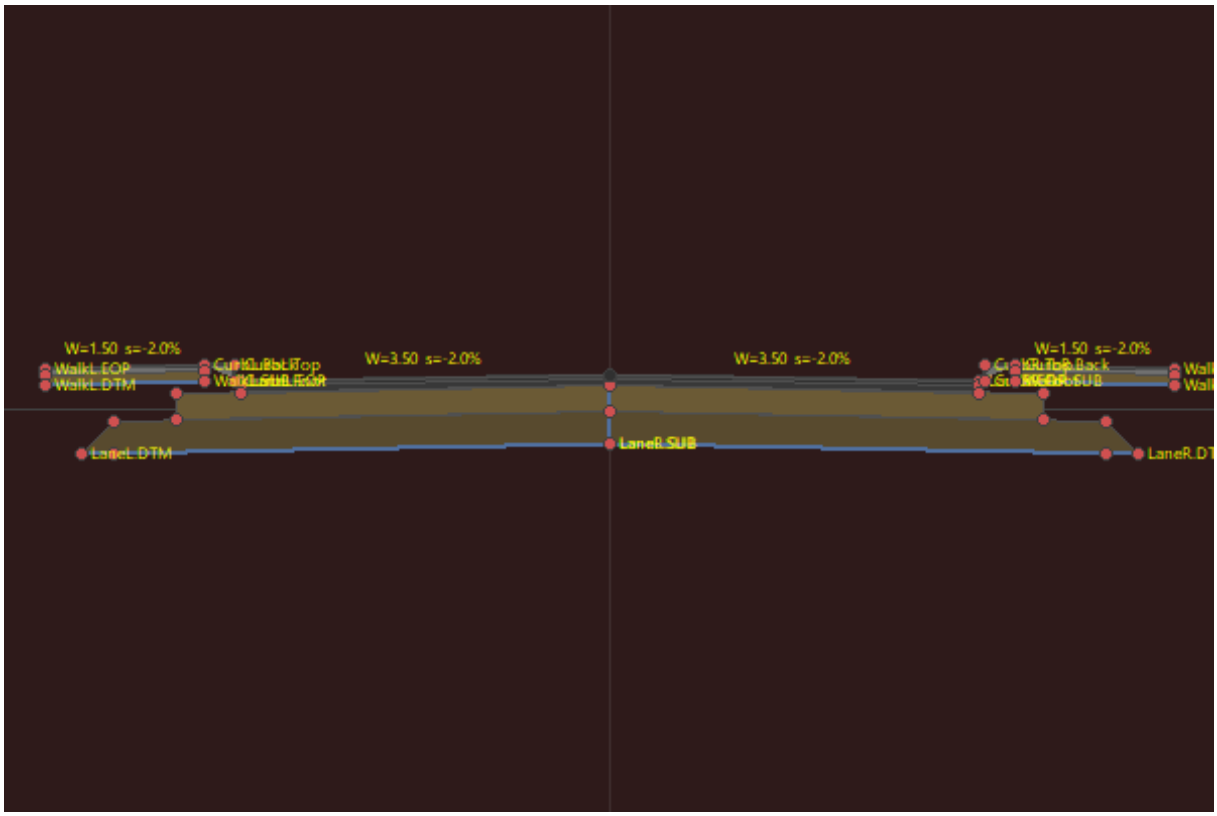


Fig. 6: Sidewalk element schematic (with parent Lane + Curb).

Parameter	Description
Width	Sidewalk width in metres. Typical 1.0–3.0 m.
CrossSlope	Drainage slope, typically gentle (e.g. −2%).
PaveDepth	(v1 legacy — migrated to a single course on load; use **Courses…**) Pavement thickness, typically shallow (e.g. 0.05 m).

When to use it. Urban roads with pedestrian provision. Always follows a Curb.

27.2.6 Slope

The daylight chain — connects the outermost designed point to the existing-ground TIN. Walks at the cut slope until the TIN is reached if the design is above ground; walks at the fill slope if below ground.

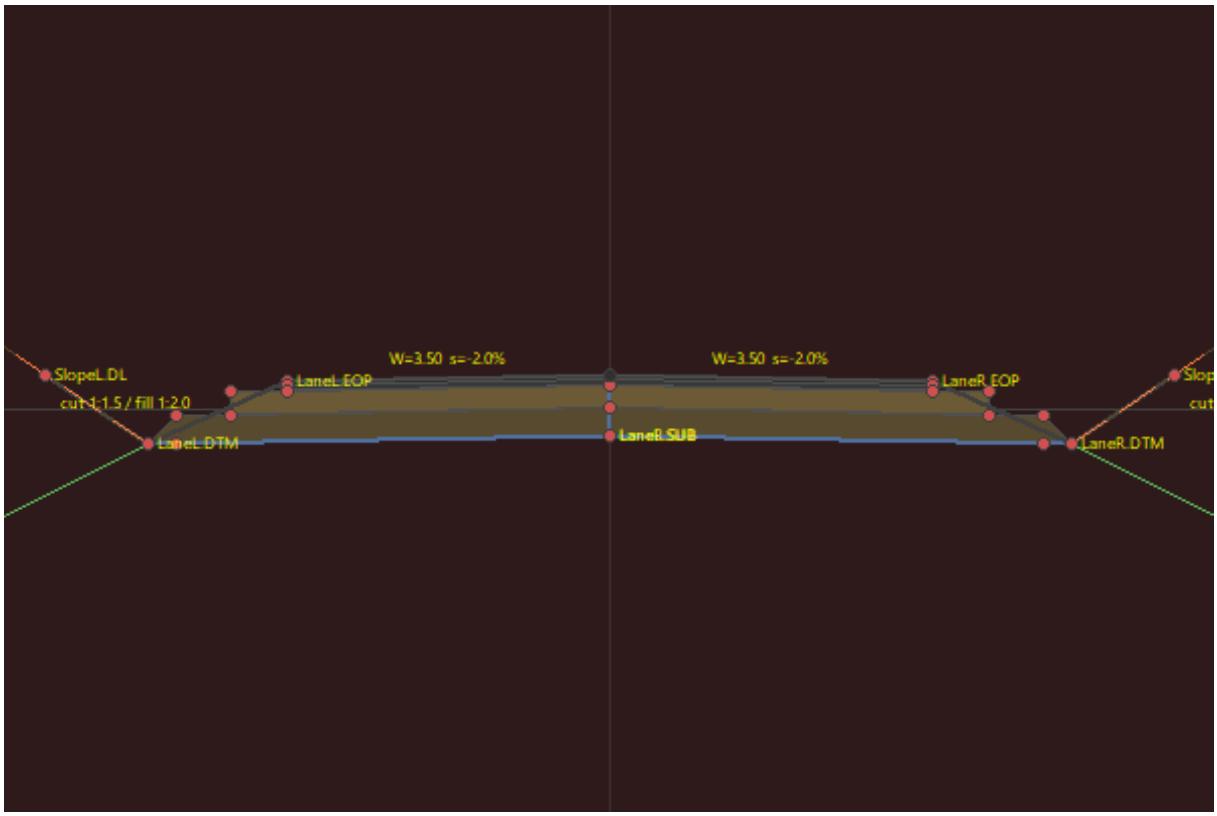


Fig. 7: Slope element schematic showing cut and fill daylighting.

Parameter	Description
CutSlope	Vertical-to-horizontal ratio for cut sections (design above ground). E.g. `0.6667` = 1V:1.5H, the typical default. Steeper for rock, gentler for soft soils.
FillSlope	Vertical-to-horizontal ratio for fill sections (design below ground). E.g. `0.5` = 1V:2H. Typically gentler than cut slopes.

When to use it. The last element on each side of every assembly. Without it the corridor has no defined edge to the existing-ground.

27.2.7 Generic

An escape hatch for cross-section elements not covered by the catalogue. Specifies a raw (DX, DY) offset from the parent point. No automatic relationship to lanes, layers, or superelevation — exactly what you type is exactly what you get.

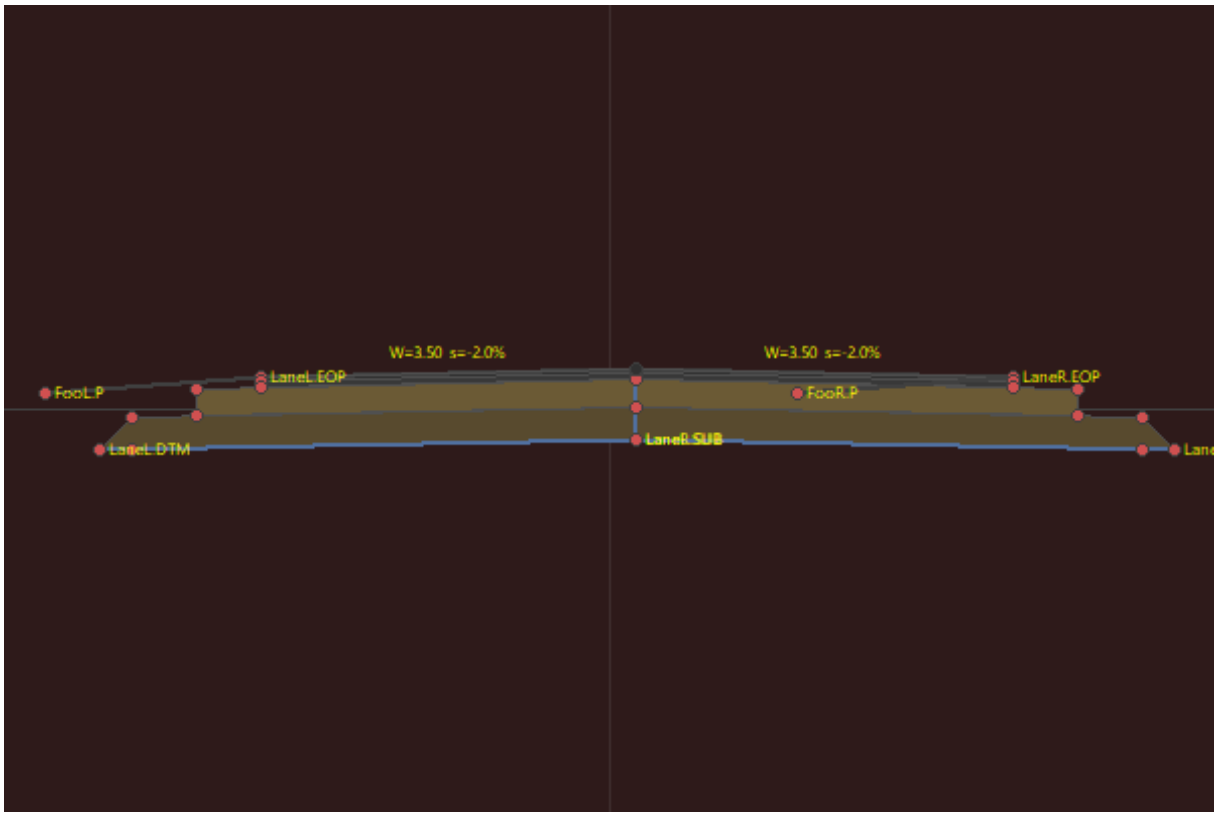


Fig. 8: Generic element schematic — raw DX/DY offset.

Parameter	Description
DX	Horizontal offset in metres from the parent point. Sign is absolute (not flipped by side), so author Left- and Right- side Generic rows with opposing signs if you want symmetry.
DY	Vertical offset in metres from the parent point.

When to use it. Crash barriers, retaining walls, custom guardrails, ditches that don't fit the Slope template, or any geometry the Lane/Shoulder/Curb/Median/Sidewalk vocabulary doesn't capture.

27.3 Pavement structure — courses and materials

Lane, Shoulder, and Sidewalk rows no longer carry a single pavement thickness. Each carries a **course stack**: an ordered list of courses, where each course is a parametric shape (material + width + thickness + edge treatment) anchored to an earlier point in the assembly.

27.3.1 What a course is

A course has:

- **Material** — a name from the drawing's material library (below).
- **Width / Thickness** — in metres; a Width of 0 inherits the parent row's width.
- **Edge treatment** (left and right independently) — **Vertical** (a plain rectangular edge) or **Taper** with a slope (dh/dv), which widens the course's bottom edge outward, producing a benched or stepped

course boundary.

- **Anchor** — a 9-point attachment: **SelfPt** (the course's own corner) glued to **TargetPt** (a named point of an earlier structure), plus a **DX/DY** offset. The nine anchor points are the corners and edge midpoints of the course's rectangle: `TL / TC / TR / ML / MC / MR / BL / BC / BR` (top/middle/bottom × left/centre/right).

27.3.2 Chain default

Leave **Target** blank and a course anchors to the chain default: the first course (index 0) hangs from the row's top link (`TL`); course *k* hangs under course *k* − 1's bottom-left corner (`TL → BL`). This is exactly the historical single-PaveDepth behaviour — templates only need explicit anchors where they deviate from a straight-down stack.

27.3.3 Wider / centered courses

A base or subbase course usually needs to be wider than the layer above it. To centre the extra width on both sides, set **SelfPt = TC** and **TargetPt = BC** (top-centre onto bottom-centre) together with a **Width** greater than the row width: the course centres itself under the course above and steps out evenly on both sides. The shipped Base course instead widens *outward only* (see below): it keeps the default anchor, so its inner edge stays pinned directly under the course above (and ultimately under the row's own inner anchor) while the full extra width goes on the outer edge (`width = W + 0.6`).

27.3.4 Taper feet

Set an edge to **Taper** with a slope to get a widening foot instead of a vertical step — typical for a subbase course that needs to bear on undisturbed ground beyond the layers above it. The shipped Subbase course uses **Taper 1:1** on its outer edge only — the inner edge stays vertical, pinned at the row's own inner anchor — with `width = W + 1.2`.

27.3.5 Cross-row anchoring

A course's **Target** field can name another row's structure, not just "the course above" — e.g. a drainage course anchored to `LaneL.Subbase.BR` ties its self point to the bottom-right corner of the left lane's subbase course, letting one row's stack key off another row's. Anchoring to a structure that hasn't been built yet (a forward reference, or a cycle) fails the build with a console error naming the unresolved target instead of crashing — fix the name and rebuild.

27.3.6 Drawing-level material library

Every drawing has its own material library (independent of the assembly library), opened with the **Materials…** button in the Assembly Editor header. It comes seeded with seven defaults:

- Asphalt Wearing
- Asphalt Binder
- Asphalt Base
- Base (crushed stone)
- Subbase
- Concrete
- Drainage gravel

Each material is a name plus a display colour; add, delete, or recolour entries freely, and reference any of them from any course's **Material** field.

27.3.7 The datum: bottom of the last course

The corridor's datum surface follows the **outer bottom corners of a row's LAST course** — not whichever course happens to be geometrically deepest. Because custom anchors (cross-row targets, DX/DY offsets) mean the true deepest bottom isn't knowable until the whole stack resolves, the rule is simply "the deepest course goes last"; every shipped template already follows it (subbase is always the final course). Curb, Median, and Generic rows still advance the datum chain the way they always did (to their Back / Edge / offset point). A row with **no courses** (a "zero-course" row — see below) advances the datum from its own top link only.

27.3.8 Quantities: per-material volumes + excavation-to-formation

`BUILDCORRIDOR` now prints, after the usual cut/fill/net line:

- **Excavation (to formation)** — the volume between existing ground and the datum surface where ground sits above datum, i.e. what actually has to be dug to reach subgrade. This is distinct from the ground-vs-top cut/fill figure, which ignores everything below the finished road surface.
- A **material table**, one row per material referenced anywhere in the assembly, each the integrated volume of every course tagged with that material along the whole corridor.

27.3.9 Known v1 simplifications

Two shortcuts are accepted in this phase and may be refined later. (The 2026-07-22 final review corrected both this section and the shipped templates themselves — see below; the previously-published "roughly 0.075 m^2" figure was wrong on top of describing the wrong joint, by about 8x.)

- **CL/median joint — none.** Base and Subbase widen *outward only* from each row's own inner anchor (never back past CL or a median edge), so on a single-lane side (Rural, Rural-Narrow, Urban) the two lanes' courses meet exactly at the shared edge with **zero** overlap.
- **Adjacent-lane stack overlap (multi-lane sides only).** On a multi-lane side (Urban-Boulevard, Highway, Freeway) each lane's Base/Subbase still widens outward past its *own* outer edge into the footprint of the *next* lane, whose own courses start fresh from its own inner edge and never reach back — so the same kind of double count still happens, just at the internal lane-to-lane joint instead of at CL. For the shipped stacks: Base overlaps 0.6 m × 0.25 m = 0.15 m^2, Subbase overlaps 1.5 m (1.2 m rectangle + 0.3 m taper foot) × 0.30 m = 0.45 m^2, roughly **0.6 m^2 of cross-section per internal lane-to-lane joint** (every shipped multi-lane template's lane width exceeds the 1.5 m Subbase reach, so this figure is exact, not capped by a narrower neighbour).
- **Shoulder interface.** The same outward reach also overlaps a Lane's own following Shoulder — capped by the shoulder's width where that is narrower than the 0.6 m/1.5 m reach (e.g. Rural-2x3.5's 1.0 m-wide shoulder caps the Subbase overlap at 1.0 m × 0.30 m = 0.30 m^2 rather than the full 0.45 m^2) — on top of the pre-existing small thickness step where the shoulder's own Wearing course meets the lane's Binder course rather than tying in continuously. A fully continuous tie-in is possible today via cross-row anchors but the shipped templates don't yet author it — planned template work for a later phase.

27.3.10 Zero-course rows

A pavement row (lane/shoulder/sidewalk) with an empty course array is a legal "zero-course" row: for datum purposes it behaves like a bare top-only row — only its top link is emitted, no separate datum point is paired to it, and the next row's stack continues from that top link exactly as before courses existed. This is also the shape a lane/shoulder/sidewalk row takes if you delete every row in its **Courses…** dialog.

27.4 Section styles

Corridor cross-sections — both model-space section views (`TAECC_SECTIONVIEW`) and the section sheets generated for plotting (`TAECC_SECTIONSHEET`) — draw their appearance from a named **corridor section style**. A style is a single named record that controls material fills (solid colour or hatch), per-element outline colours, the corridor silhouette pen, the background/grid, cut/fill shading, and four independent annotation toggles. Styles live in a library on the drawing, exactly like the alignment, surface, and profile-view style libraries; a view or sheet picks one by name, and a blank name — or a name that no longer exists in the library — falls back to **Standard**.

Geometry (extents, datum/top elevation, vertical exaggeration, title) stays on the view/sheet itself; the style only ever controls how that geometry is *drawn*.

27.4.1 Built-in styles

Every drawing's corridor section style library ships with two built-in styles that cannot be renamed or deleted:

Style	Description
Standard	Solid material fills using each material's colour from the drawing's material library (the same colours seen in the Assembly Editor preview), grey course outlines, and a yellow dash-dot corridor silhouette.
Monochrome	Plot-ready: white background, black pens throughout (grid, ground, outlines, silhouette), cut/fill shading turned off, and all seven default materials filled with a black **ANSI31** hatch (alternating 0°/90° angle material-to-material) instead of a solid colour.

Both built-ins, and any custom style you add, are available to every section view and sheet in the drawing.

27.4.2 Opening the Style Editor

Civil › Style Manager, then select **Section View Styles** in the left-hand category tree. The editor embeds in the manager's right pane: a left list strip of every style in the library — **New**, **Duplicate**, **Rename**, **Delete** buttons below it (**Standard** is protected from rename and delete) — and a tabbed panel on the right with five tabs: **View**, **Ground**, **Corridor**, **Annotations**, **Preview**.

27.4.3 Editable fields

Table 1: View tab

Field	Meaning
Show grid	Toggles the background grid.
Background / grid / minor-grid colour	Colours for the view background and the major/minor grid lines.
Major/minor grid X/Y	Grid line spacing, horizontal and vertical, in drawing units.
Axis text height	Pixel height of the offset/elevation axis labels.

Table 2: Ground tab

Field	Meaning
Ground pen colour / width	Pen for the existing-ground surface polyline(s).
Show cut/fill shading	Toggles the tinted region between existing ground and the corridor's top surface.
Cut colour / Fill colour	Shading colours for the cut region (ground above design) and the fill region (ground below design).

Table 3: Corridor tab

Field	Meaning
Show material fills	Master toggle for every course-shape fill (solid or hatch).
Overrides grid	Per-material fill overrides — see "Material fills and overrides" below.
Outline colours (7)	One colour per row kind — **Lane, Shoulder, Median, Curb, Sidewalk, Slope, Generic** — for the course-shape outlines of that kind.
Silhouette colour / width / style	Pen for the corridor's outer silhouette (Top left-to-right, Datum right-to-left) — the single yellow dash-dot line sections drew before this feature shipped.
Course-line colour	Reserved colour field for course-boundary lines; the shipped renderer currently draws course boundaries through the outline pens above.

Table 4: Annotations tab

Field	Meaning
Show slope/width labels	`W=<width> s=<slope>%` labels on each segment of the corridor's top surface.
Show point marks / Show point names	Marker + name for every named corridor point (`CL`, `EPS`, …) carried on the section.
Show area block	Per-station cut/fill/excavation-to-formation figures plus a per-material area table, drawn beside the section.
Show legend / Legend per sheet	Material-color legend box; per sheet draws one legend for the whole sheet instead of one per section cell.
Annotation text height	Pixel height shared by every annotation label.

The **Preview** tab paints a small synthetic section (two course shapes plus a ground polyline) through the same renderer used for real section views and sheets, so every field change is visible immediately without

building a corridor.

27.4.4 Material fills and overrides

Every course shape resolves its fill in this order:

1. **Override** — if the style has a row for the shape's material in the Corridor tab's overrides grid, that row wins: **Mode** (typed as `solid` or `hatch`), and for `hatch` a **Pattern** name (e.g. `ANSI31`, any pattern from the drawing's hatch-pattern library), **Scale**, **Angle**, and **Colour**.
2. **Material colour** — otherwise, the shape fills solid with its material's colour from the drawing's material library (the same library the Assembly Editor's **Materials…** button opens).
3. **Row-kind fallback** — untagged shapes (curbs, medians — see "Pavement structure" above) fall back to a colour keyed by row kind.

This is the same resolution the Assembly Editor's schematic preview uses (see "Editor display" below), so a material recoloured via **Materials…** updates consistently everywhere: assembly preview, thumbnails, section views, and section sheets.

Add or remove override rows with the grid's **Add/Delete** buttons; each row is independent, so a style can hatch-fill some materials while leaving others solid.

27.4.5 Persistence

The style library is saved into the drawing (IQML/IQZ) under a `CorridorSectionStyles` element, alongside the other civil style libraries. Section views and sheets store only the style **name**; resolution to a concrete style happens at draw time, so a missing or deleted style name falls back to **Standard** without invalidating the view or sheet. Because assemblies and corridor sections were not previously released, there is no legacy-drawing migration concern — every existing view/sheet simply resolves as **Standard**.

> **Note**
>
> The **Create Section Sheet…** dialog's Settings tab now includes a **Section style** combo (defaulting to **Standard**) so a new sheet can be assigned any style, including **Monochrome**, at creation time. There is still no per-entity control to switch an *existing* view or sheet to a different style after the fact — use the **Preview** tab above to check how any style will render before that control ships in a later phase.

27.5 Conditional rows, ditches and benching

(Corridor v1.3 Phase 5.) Three additions let an assembly's daylight side adapt to the ground it actually meets, instead of always resolving the same geometry: a per-row **Cut/Fill condition**, a trapezoidal **Ditch** row for the cut side, and **benching** — tall daylights broken into repeated lift-and-bench steps rather than one long slope face. A related geometry fix ships alongside them: the daylight hinge now springs from the true edge of excavation on both branches (see "The excavation-edge hinge fix" below).

27.5.1 Row condition: Always / Cut / Fill

Every row in the Assembly Editor grid now carries a **Cond** column (inserted right after **Name**), a three-way choice: blank/**Always** (the row is unconditional — today's behaviour, and the default for every existing row), **Cut**, or **Fill**. A row set to Cut or Fill only contributes its geometry, area, and section-view shapes at stations where the corridor resolver decides that side of the alignment is actually in cut (respectively fill) — at every other station the row emits nothing at all, as if it had never been authored.

The decision is made once per side per station, at the **daylight hinge** (the point described in the hinge fix below): the resolver probes the target surface directly above/below the hinge — ground higher than the hinge means the station is in cut on that side, otherwise it's in fill — then activates every Cut-conditioned row and deactivates every Fill-conditioned row, or vice-versa. The losing branch is not merely hidden: it contributes zero to cut/fill/ excavation quantities, and its shapes never reach the section views or section sheets, exactly as if it were pruned from the assembly for that station.

In the Assembly Editor's live preview, conditional geometry is drawn schematically on **both** branches at once so you can review each side's design without building a corridor: the cut branch dashed and tinted orange, the fill branch dashed and tinted green. Only one branch will actually resolve at any given corridor station, but the preview always shows both so a conditional ditch or bench setup can be checked without needing test ground.

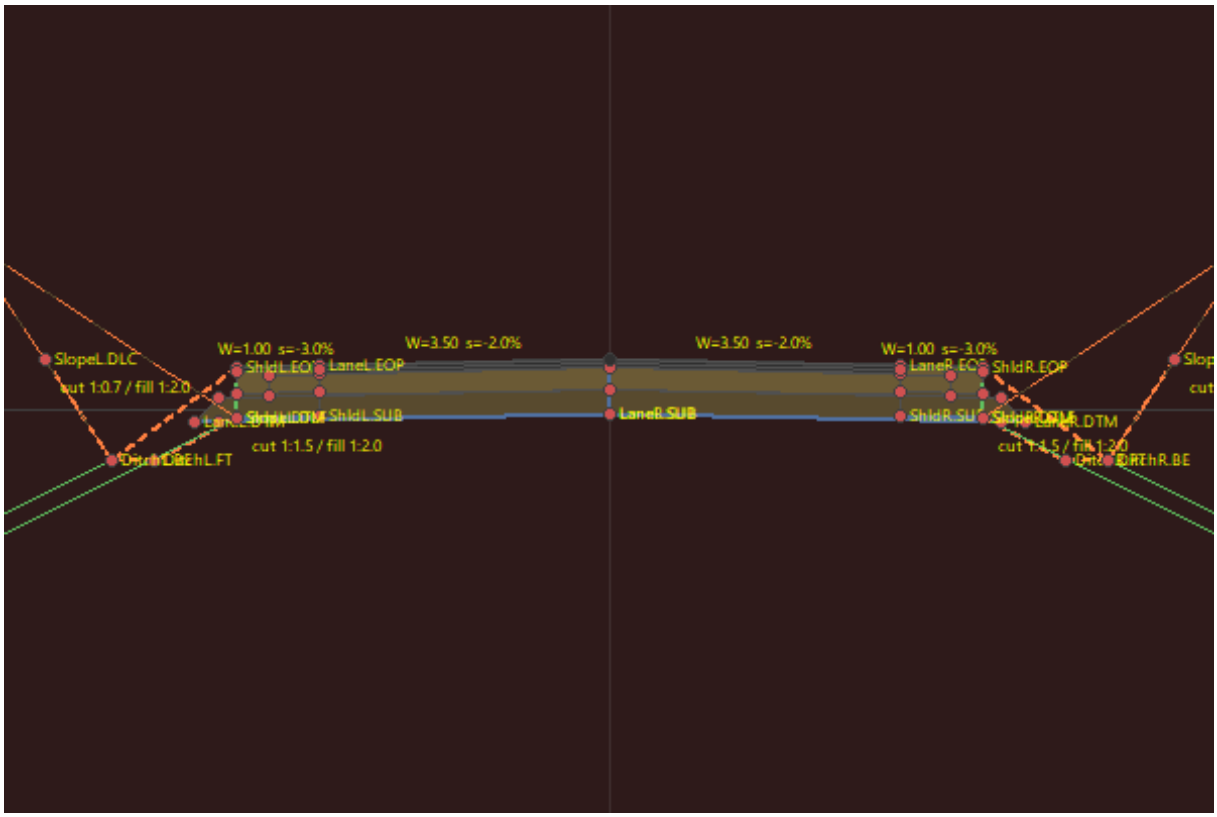


Fig. 9: A cut-only Ditch row (orange, dashed) ahead of the daylight.

27.5.2 Ditch

Ditch is a new row kind for a trapezoidal drainage channel, added with the **+Ditch** button under each side's row grid (next to the other row-palette buttons). A new ditch row defaults to name `Ditch`, layer `C-DTCH`, and condition **Cut** — it is meant for the cut branch and is authored that way by default, though any condition is legal if you have an unusual use for it.

Parameter	Description
Depth	Vertical depth of the ditch invert below the point it springs from, in metres.
BottomWidth	Width of the ditch's flat invert in metres. **0 produces a V-ditch** (no flat bottom, the two slopes meet at a point).
ForeSlope	Slope of the near (inboard) face, as 1:n (rise:run — e.g. `2.0` = 1:2).
BackSlope	Slope of the far (outboard) face, as 1:n. Edited in the row params dialog (double-click any P column of a Ditch row) rather than the grid, alongside Depth/BottomWidth/ForeSlope.

A ditch springs from the excavation edge exactly like an unconditional daylight would, then walks down at **ForeSlope** to the invert, across **BottomWidth**, and back up at **BackSlope** — and it is that rising backslope which becomes the starting slope of the cut daylight chain that follows it, so the ditch and the cut face read as one continuous excavated profile rather than two unrelated elements. The ditch's own bottom also extends the excavation envelope: quantities' **Excavation (to formation)** figure counts everything down to the ditch invert on an active ditch, not just to the pavement datum, since a ditch is simultaneously a finished-surface feature (it's what you see) and a formation feature (it's what has to be dug). A ditch row draws as an open outline (no fill) using its own outline-colour slot in the section style's Corridor tab, alongside Lane/Shoulder/Median/ Curb/Sidewalk/Slope/Generic.

27.5.3 Benching

Tall daylights no longer walk a single uninterrupted slope face all the way to the ground: a Slope row can now specify **benching** separately for its cut branch and its fill branch, each a triple of **MaxHeight / BenchWidth / BenchSlope** edited in the row params dialog (double-click a Slope row's P column). Leaving **MaxHeight** at its default of **0** disables benching for that branch — the daylight behaves exactly as before.

With benching enabled, the resolver walks the active branch's slope for at most **MaxHeight** of vertical rise/fall; if the ground crossing is found inside that lift, the ordinary daylight point closes the chain there (to the usual 1 mm tolerance) and benching never engages for that station. If the ground is not yet reached, the daylight instead breaks outward: a bench point at the lift's top, walking **BenchWidth** outward at **BenchSlope** to a second bench point, then the next lift starts from there. Named points appear along the way — `DL.B1`, `DL.B1E`, `DL.B2`, `DL.B2E`, … — one pair per completed lift, so section views and feature lines carry a distinct point per bench. Lifts repeat until the ground is met; a hard cap of **20 lifts** stops the walk and flags that station **Ungraded** on the command line rather than looping forever on pathological ground.

The Assembly Editor's preview draws one schematic bench break — a single jog partway along the branch's daylight stub — whenever a branch's MaxHeight is non-zero, as a reminder that benching is active; the preview's flat test ground can't show a real multi-lift walk, so the schematic jog is deliberately just an indicator, not a solved bench chain.

27.5.4 The excavation-edge hinge fix

Both the cut and the fill daylight now spring from the **end of excavation** — the outer bottom corner of a side's outermost course stack, exactly where the excavation-to-formation envelope actually ends — rather than from the shoulder's finished top edge as before. Previously the daylight point anchored to the row chain's top link, which sits above and inboard of a widened subbase or a taper foot; because those courses are authored wider than the row itself (see "Pavement structure" above), the true excavated face reached further out than the daylight line implied, undercutting it. Anchoring to the excavation edge instead means the daylight — and any ditch feeding into it — starts exactly where the dig actually stops.

A short connector link closes the gap down the outermost course stack's outer face (from the row's top point to the excavation-edge point) so the Top chain stays continuous for area and silhouette purposes; on a side

with no course stacks at all (e.g. a bare Generic or Median row feeding straight into a Slope), there is no excavation edge to speak of and the daylight falls back to anchoring at the row's own top point, exactly as before this fix.

This is a **bug fix**, not a new option — every assembly with course stacks on its outer rows sees its daylight (and, if you add one, its ditch) start slightly further out and down than in earlier releases; unconditional daylight-only assemblies otherwise resolve bit-identical geometry.

27.5.5 Upgraded standard templates

All six templates in `standard-roads.iqal` (Rural-Narrow-2x3.0, Rural-2x3.5, Urban-2x3.5+walks, Urban-Boulevard-2x2x3.25, Highway-2x2x3.5+median, Freeway-2x3x3.75) now carry, on each side, immediately before the closing Slope row:

- A **Cut-only ditch** — Depth 0.5 m, BottomWidth 0.5 m, ForeSlope 1:2, BackSlope 1:1.5.
- **Benching on both branches** of the Slope row — 5 m lifts, 1.5 m benches, 4% bench slope.

> **Note**
>
> On **Urban-2x3.5+walks** and **Urban-Boulevard-2x2x3.25** the ditch ends up sitting *behind the sidewalk* (Lane → Curb → Sidewalk → **Ditch** → Slope) — a cut-drainage ditch with no shoulder buffer ahead of it, which is unusual civil practice for an urban street. This is a literal application of "the ditch goes between the last row and the slope on each side" to templates that have no shoulder; the other four templates place the ditch after a shoulder, which reads normally. Review this per project: edit the ditch's parameters, move it, or delete the row entirely on a copy of these two templates if a project doesn't want cut drainage immediately behind its pedestrian walk.

27.5.6 Persistence

Row condition, ditch parameters, and both branches' bench parameters round-trip through `.iqal` (and the equivalent IQML) exactly like every other row field: an unconditional row omits the `condition` attribute entirely (so pre-Phase-5 files still load with every row defaulting to Always), a Ditch row always writes its four ditch attributes, and a Slope row writes each of its six bench attributes only when non-zero. This is an additive extension of the existing v2 schema — no version bump was needed.

27.6 Standard sample library

IQcad ships six pre-authored road templates in `standard-roads.iqal` (under `testdata/library/` in the distribution). An empty drawing's assembly library auto-loads them the first time you open **Civil › Assemblies…** or run `BUILDCORRIDOR` — no manual import step is needed. You can also seed them on demand with `SAMPLEASSEMBLIES` (alias `SAS`), which is idempotent: assemblies that already exist by name are skipped. The **Import** button in the Assembly Editor remains available for loading additional/custom `.iqal` files from elsewhere.

27.6.1 Rural-Narrow-2x3.0

Low-traffic backroad. Narrow lanes and shoulders with relatively steep slopes — appropriate for mountain roads, forest tracks, and single-family rural access.

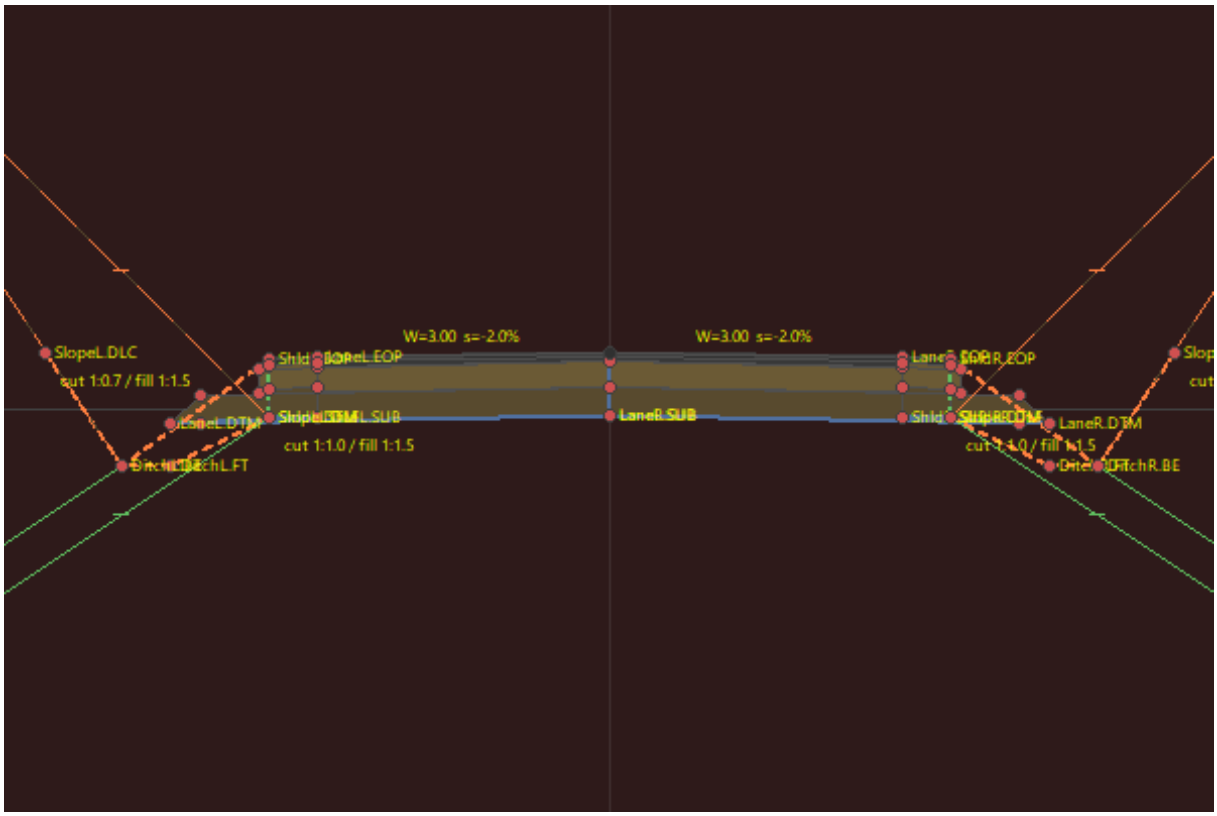


Fig. 10: Rural-Narrow-2x3.0 cross-section.

When to use it. Low-volume rural backroads where right-of-way is constrained and design speed is low (under 50 km/h).

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Lane	Width 3.0 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
2	Shoulder	Width 0.5 m, CrossSlope −3%, peraltable. Courses: Wearing 5 cm AC + Base 25 cm + Subbase 30 cm.
3	Slope	Cut 1:1, Fill 1:1.5

27.6.2 Rural-2x3.5

Standard symmetric two-lane rural road. 3.5 m lanes with 1.0 m shoulders and 1:1.5 cut / 1:2.0 fill slopes daylighting to the TIN.

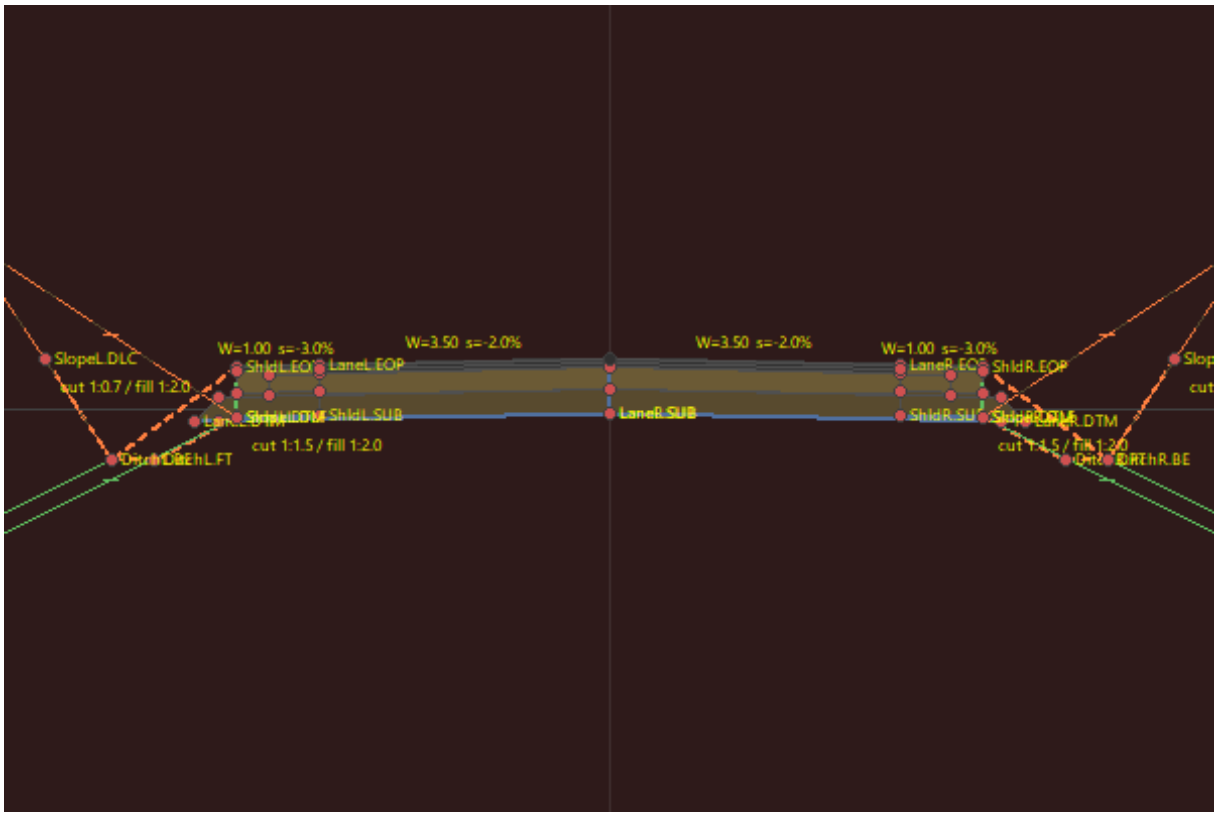


Fig. 11: Rural-2x3.5 cross-section.

When to use it. Most rural and inter-urban roads outside settlements. The slopes assume typical engineering soils; for rock cuts use a steeper Slope row, for soft soils a gentler one.

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Lane	Width 3.5 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
2	Shoulder	Width 1.0 m, CrossSlope −3%, peraltable. Courses: Wearing 5 cm AC + Base 25 cm + Subbase 30 cm.
3	Slope	Cut 1:1.5, Fill 1:2.0

27.6.3 Urban-2x3.5+walks

Urban street with kerb and sidewalk on each side. 3.5 m lanes, 0.15 m face-height kerb, 1.5 m sidewalk. No shoulder.

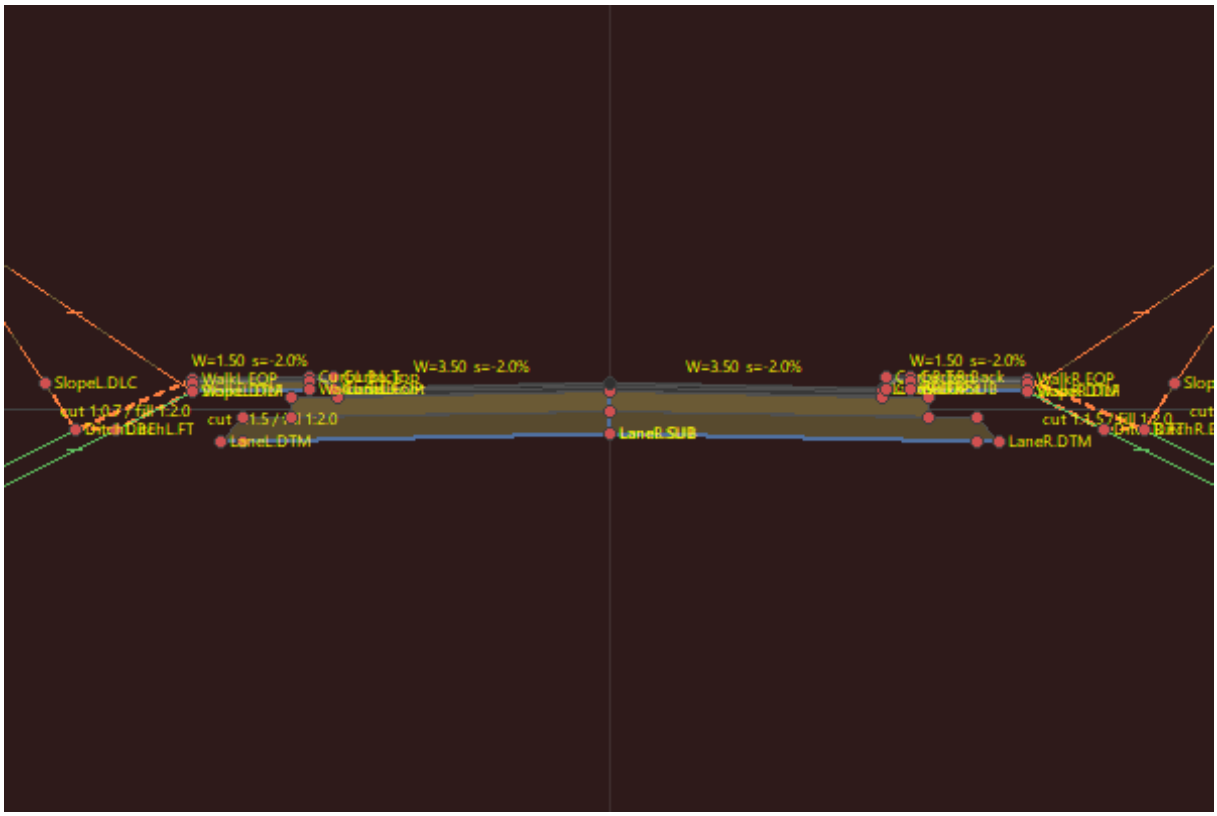


Fig. 12: Urban-2x3.5+walks cross-section.

When to use it. Urban single-carriageway streets with pedestrian provision — the typical "residential collector" cross-section.

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Lane	Width 3.5 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
2	Curb	FaceW 0.05 m, FaceH 0.15 m, TopW 0.30 m
3	Sidewalk	Width 1.5 m, CrossSlope −2%. Courses: Slab 5 cm concrete + Base 10 cm.
4	Slope	Cut 1:1.5, Fill 1:2.0

27.6.4 Urban-Boulevard-2x2x3.25

Divided urban with central median and sidewalks. Two 3.25 m lanes per direction, narrow planted median, kerb and 2.0 m sidewalk.

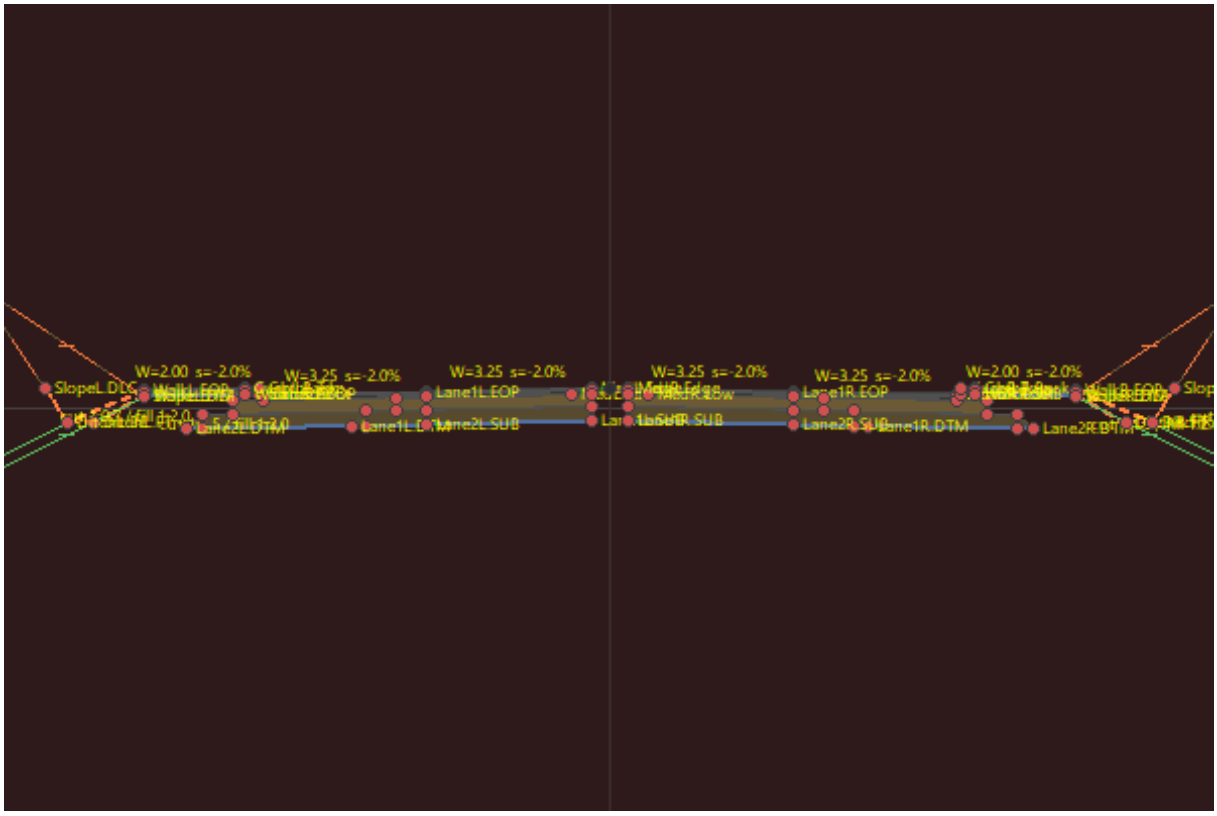


Fig. 13: Urban-Boulevard-2x2x3.25 cross-section.

When to use it. Urban arterials with pedestrian provision and traffic separation — typical “city boulevard” cross-section.

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Median	HalfWidth 0.75 m, Depth 0.10 m, InnerSlope 1:10
2	Lane (Lane1)	Width 3.25 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
3	Lane (Lane2)	Width 3.25 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
4	Curb	FaceW 0.05 m, FaceH 0.15 m, TopW 0.30 m
5	Sidewalk	Width 2.0 m, CrossSlope −2%. Courses: Slab 5 cm concrete + Base 10 cm.
6	Slope	Cut 1:1.5, Fill 1:2.0

27.6.5 Highway-2x2x3.5+median

Divided rural highway. Two 3.5 m lanes per direction, 2.5 m shoulders, 1.0 m central median.

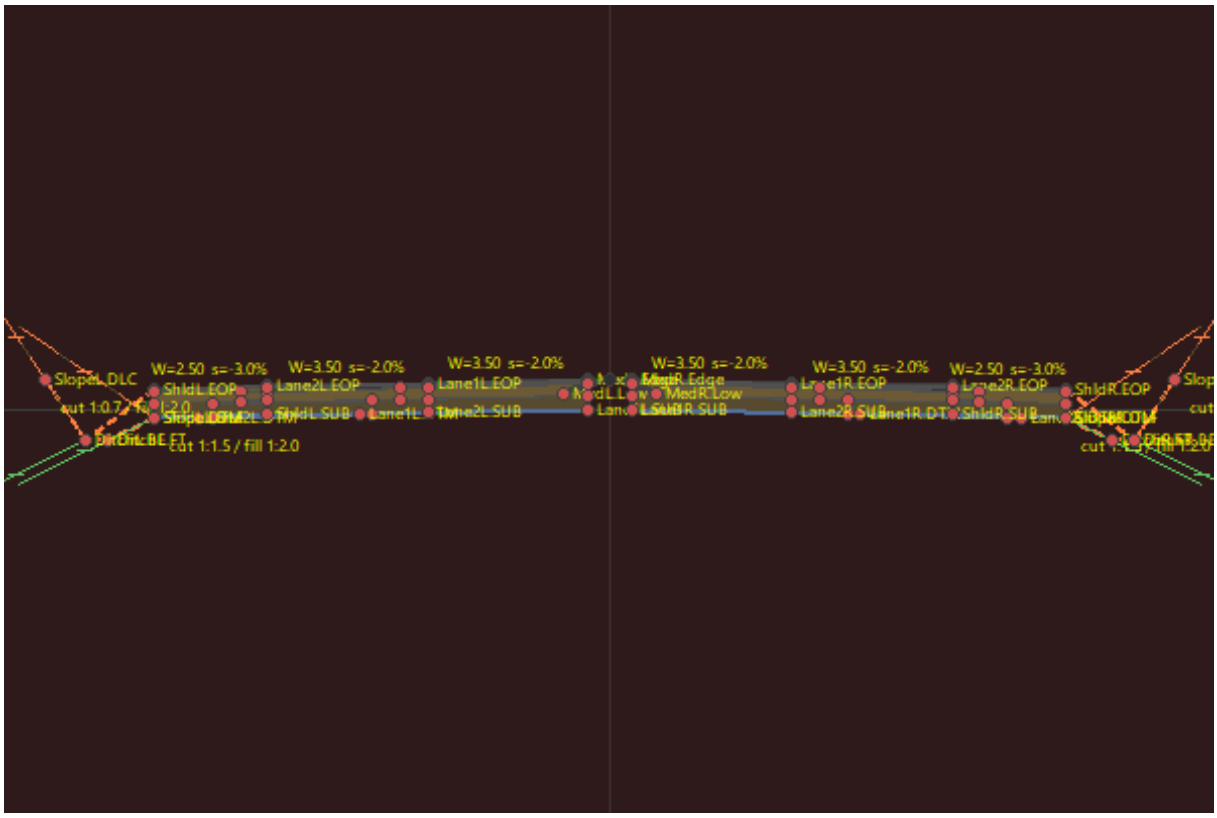


Fig. 14: Highway-2x2x3.5+median cross-section.

When to use it. Inter-city divided highways with moderate traffic. Wider median templates can be authored by adjusting HalfWidth on the Median row.

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Median	HalfWidth 1.0 m, Depth 0.30 m, InnerSlope 1:10
2	Lane (Lane1)	Width 3.5 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
3	Lane (Lane2)	Width 3.5 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
4	Shoulder	Width 2.5 m, CrossSlope −3%, peraltable. Courses: Wearing 5 cm AC + Base 25 cm + Subbase 30 cm.
5	Slope	Cut 1:1.5, Fill 1:2.0

27.6.6 Freeway-2x3x3.75

Three-lane freeway per direction. 3.75 m lanes, 3.0 m shoulders, wider 2.0 m median.

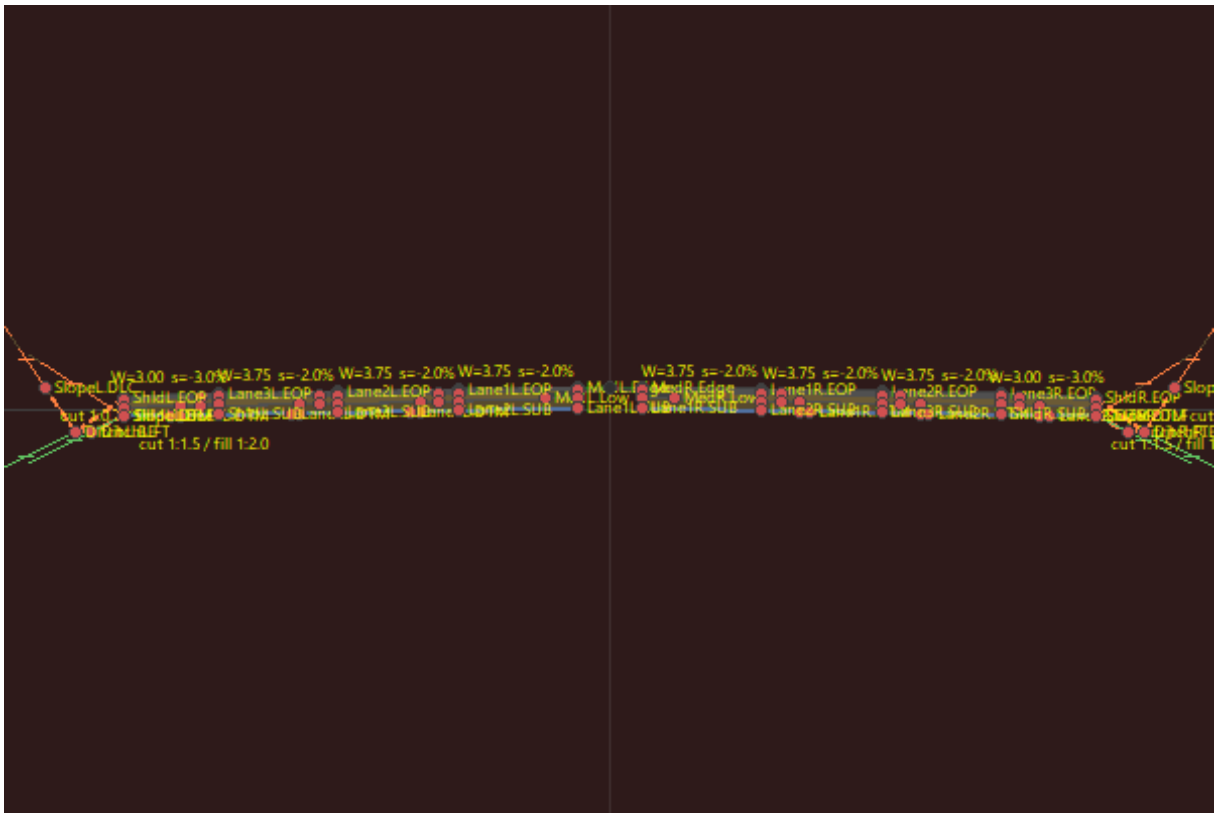


Fig. 15: Freeway-2x3x3.75 cross-section.

When to use it. High-volume motorways and freeways. For four+ lanes per direction, copy and add additional Lane rows.

Composition (per side; right side is mirrored):

Order	Element	Parameters
1	Median	HalfWidth 2.0 m, Depth 0.30 m, InnerSlope 1:10
2	Lane (Lane1)	Width 3.75 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
3	Lane (Lane2)	Width 3.75 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
4	Lane (Lane3)	Width 3.75 m, CrossSlope −2%, peraltable. Courses: Wearing 4 cm AC + Binder 6 cm AC + Base 25 cm (W+0.6 m, stepped) + Subbase 30 cm (W+1.2 m, taper 1:1).
5	Shoulder	Width 3.0 m, CrossSlope −3%, peraltable. Courses: Wearing 5 cm AC + Base 25 cm + Subbase 30 cm.
6	Slope	Cut 1:1.5, Fill 1:2.0

27.7 Editor display

The Assembly Editor's live preview and assembly picker read like a real cross-section rather than a wireframe — the shipped fills use the same colors seen in the Rural-2x3.5 figure above.

27.7.1 Filled parts

Every course shape fills with its material's color from the drawing's material library: recolor a material via **Materials…** and the preview and every thumbnail follow immediately. Curbs and medians carry no material reference yet, so they fill with fixed fallback colors (concrete-grey and green respectively). Course outlines draw as thin grey lines over the fills; the Top and Datum lines keep their existing (thicker, colored) styling.

27.7.2 Annotations

Lane, shoulder, and sidewalk rows show `W=<width> s=<slope>%` above their top link; daylight points show their `cut 1:n / fill 1:n` ratios. Labels declutter automatically — a link drawn narrower than 40 px on screen hides its label; zoom in to bring it back.

27.7.3 Both daylight branches

At the daylight hinge the preview always draws both a schematic cut branch (dashed, rising) and a schematic fill branch (solid, falling), each 3 m long, in addition to the branch actually resolved against the flat preview ground, which keeps its normal link styling.

27.7.4 Thumbnail list

The assembly picker is a list of thumbnail + name rows to the left of the preview, replacing the old name combo. Selecting a row loads that assembly into the left/right grids. Thumbnails refresh automatically on **Apply**, **Import**, **Rename**, and **Delete**, and after material color edits.

27.7.5 Course highlighting

While the **Courses…** dialog is open, selecting a row outlines the corresponding course in cyan in the preview behind the dialog. On some systems the outline also updates live while the dialog is open; on every system the highlighted course remains visible after you close the dialog, until the next edit action (**Apply**, switching the selected assembly, or adding, deleting, moving, or mirroring a row) clears it.

27.8 Building a corridor

Once you have an alignment, a vertical profile, and an existing-ground surface in the drawing, building a corridor takes four steps.

1. **Open the Assembly Editor.**
   `Civil → Assemblies…` opens the editor. In an empty drawing the six standard road templates auto-load — the combo already lists them, no import needed. Use **Import** only to bring in additional or custom `.iqal` files from elsewhere.

2. **(Optional) Author your own assembly.**
   Click **New**, give it a name, then click the row-palette buttons (**+Lane**, **+Shoulder**, etc.) under the left grid to add rows in order from CL outward. Click **Mirror** to copy the left side to the right. The live preview updates as you type. Click **OK** to save into the drawing.

For lane/shoulder/sidewalk rows, the **Courses…** button (one per side, next to the row grid) opens a modal grid listing every course in the row's stack — columns #, Name, Material, Thickness, Width, SelfPt, Target, TargetPt, EdgeL, SlopeL, EdgeR, SlopeR, DX, DY — with **Add / Delete / Up / Down** buttons to manage rows and a **Material** combo populated from the drawing's material library. Click **OK** in the courses dialog to write the stack back, then **OK/Apply** in the assembly editor to commit it.

The editor header's **Materials…** button opens the drawing-level material library: a name + colour grid with **Add / Delete** and a colour-picker button that recolours the selected row. Changes apply immediately to every course referencing that material's name.

3. Bind the corridor to an alignment.

Open the **Alignment PI Editor**, select your alignment, switch to the **Corridor** tab. Pick the assembly + target surface, set the sample frequency (default 5 m).

4. Build.

Click **Build Corridor** (or run BUILDCORRIDOR from the command line, alias BC). Within a few seconds the drawing gains:

- A **3D mesh** of the corridor body.
- A **Top surface** (AECC_SURFACE) draping the road's running surface.
- A **Datum surface** (AECC_SURFACE) under the pavement.
- One **3D feature line** per named point along the alignment (edge of pavement, daylight line, etc.).
- A **volumes report** on the command line: cut, fill, net, excavation-to-formation, and a per-material quantity table (see "Pavement structure — courses and materials" above).

If the status label reads **Out of date**, you have edited a source (alignment, target surface, or library); click Build again to regenerate the outputs.

28

Grading, Culverts and the 3D Object Viewer

28.1 Grading

Grading designs a graded pad — a building platform, parking lot, pond, or terrace — by projecting cut and fill slopes outward from a footprint at a design elevation until they *daylight* (meet) an existing-ground surface. The result is a graded terrain surface plus the earthwork cut/fill volumes.

A grading is a **dynamic, self-contained object** (`AECC_GRADING`): it owns and draws its own geometry (the graded TIN, the daylight line and any bench lines) — no separate polylines or surfaces clutter the drawing — and it **recomputes automatically** when you change its inputs. Grip-drag a footprint vertex and the graded surface follows live; edit a slope in the inspector and it recomputes at once.

28.1.1 The grading object

A grading binds three inputs and produces its outputs internally:

- **Footprint** — a closed (or open) polyline outlining the pad.
- **Target surface** — the existing-ground TIN (or DEM) the slopes daylight onto.
- **Criteria** — the design elevation, the cut/fill slopes, and benching.

From these it builds, and draws itself:

- the **graded TIN** (pad top + the sloped skirt out to daylight), styled by a surface style;
- the **daylight line** — the outer perimeter where the slopes meet ground;
- **bench lines** — the concentric terrace edges when benching is enabled;
- the **cut, fill and net volumes** against the target surface.

28.1.2 Creating a grading

1. Draw a **closed polyline** for the pad outline and have an existing-ground **surface** in the drawing.
2. Run **`GRADING`** (alias `GRD`), or **Civil › Grading**. The **Grading** dialog opens.
3. Fill in the criteria (below) and click **Create / Build**.

The **Grading** dialog groups the settings:

- **Footprint** — pick the pad polyline (all polylines in the drawing are listed; unnamed ones are auto-named so they can be referenced).

- **Target surface** — the existing-ground surface, listed by its civil name.
- **Elevation** — how the pad-top elevation is set:
  - **Flat** — a single **design elevation** for the whole pad.
  - **Footprint vertex Z** — the pad edge follows the polyline's own per-vertex elevations (use a 3D polyline).
  - **Drainage** — the design elevation plus a constant cross-pad slope in a given direction (a drainage grade to a low edge).
- **Cut / Fill** — the outslope for each, entered as a **slope** (H:V ratio, e.g. `3` = 3:1) or a **grade** (%); toggle the **Format** to switch. Each perimeter point automatically uses the **cut** slope where the pad is below ground (excavation) and the **fill** slope where it is above (embankment).
- **Benching** — terraces on tall slopes, set separately for cut and fill:
  - **Enable** — turn benching on for that side.
  - **Width** — the horizontal width of each flat bench shelf (m).
  - **Interval** — the vertical rise between benches (m); a bench is inserted each time the slope has climbed/dropped this far.
  - **Cross-slope (%)** — an optional tilt across each shelf (a back-drain). A positive value tilts the shelf down toward the outer, slope-face edge so water collects at the toe of the upper batter; `0` = flat.
- **Sample interval** — how finely the perimeter is discretised (smaller = smoother daylight, more triangles).
- **Max search distance** — how far a slope may run before it is reported as *ungraded* (it never reaches ground within this distance).

Note

Cut vs fill sign. Cut and fill volumes are both reported as positive magnitudes; **net = fill − cut** (a negative net means net cut / export). A pad below existing ground yields cut; a pad above yields fill.

28.1.3 Dynamic editing and rebuild

A grading keeps itself up to date. With **Auto rebuild** on (the default):

- **Grip-drag a footprint vertex** → the grading recomputes and redraws live, every mouse-move.
- **Move the whole footprint** → it updates when you release.
- **Edit a criterion** (design elevation, a cut/fill ratio, benching, the surface style) in the **Object Inspector** → it recomputes immediately.
- **Edit the target surface** → the grading follows.

Turn **Auto rebuild** off (in the inspector) for very large pads; the grading then marks itself `[stale]` on an edit and you rebuild on demand with **`BUILDGRADING`** (alias `BG`) or the prospector **Rebuild** item.

28.1.4 Volume balance

Balance solves the pad's design elevation so the earthwork hits a target net volume — most often **0**, a balanced cut/fill site with no import or export.

Run **GRADINGBALANCE** (alias GRDB), or right-click the grading in the prospector and choose **Balance…**. Enter the **target net volume** (default 0) and a **tolerance** (default 1 m^3). Because net volume rises monotonically with the pad elevation, the tool root-finds the elevation (bracket + bisection, re-running the grading at each trial), sets it, and rebuilds. It reports the solved elevation and achieved net, or that the target is **not achievable** for this pad within the search range (e.g. asking for more fill than the pad can physically produce).

Balance adjusts the **design elevation**, so it applies to **Flat** and **Drainage** pads; on a **vertex-Z** footprint (whose Z comes from the polyline) it reports that it needs a Flat or Drainage pad instead.

28.1.5 Extract, show/hide, lock

The grading's graded surface is internal, but you can get a real one on demand:

- **Extract Surface** — **GRADINGEXTRACT** (alias GRDX) or the prospector **Extract Surface** item clones the graded TIN into a standalone AECC_SURFACE named <Grading>_Extracted, which you can section, contour, difference or feed to a corridor. It is a snapshot — re-extract to refresh.
- **Show / Hide** — the prospector **Show/Hide** item (or the **Visible** inspector property) hides the grading from the drawing; hidden gradings are not drawn and cannot be clicked.
- **Lock / Unlock** — the prospector **Lock/Unlock** item (or the **Locked** inspector property) prevents the grading from being selected or moved. A locked civil object is unlocked from the **prospector** node (not by clicking it on the canvas).

28.1.6 Selecting a grading

Click anywhere on the graded surface to select the grading and edit its values in the Object Inspector. A grading appears in the **Civil Toolspace** under a **Gradings** node, tagged [stale], [hidden] or [locked] as applicable; right-click it for **Rebuild**, **Balance…**, **Extract Surface**, **Show/Hide**, **Lock/Unlock** and **Delete**.

28.1.7 Grading commands

Command	Alias	Description
GRADING	GRD	Open the Grading dialog to create/build a grading from a footprint, target surface and criteria.
BUILDGRADING	BG	Rebuild a stale grading (when Auto rebuild is off).
GRADINGBALANCE	GRDB	Solve the design elevation for a target net cut/fill volume.
GRADINGEXTRACT	GRDX	Clone the graded surface into a standalone editable surface.

28.1.8 Limitations and notes

- **Convex pads are fully supported.** For a strongly **concave / notched** (e.g. L-shaped) footprint the graded TIN may bridge the notch — check the result on such shapes.
- Per-segment cut/fill overrides are supported by the engine and files but are not yet editable from the dialog (the global cut/fill applies to every side).
- Balance targets the design elevation (Flat/Drainage pads); vertex-Z pads are warned.
- Corner rounding is applied automatically (convex corners fan, concave corners trim), so the skirt closes cleanly at every vertex.

28.2 Culverts

Culvert creates a parametric **box culvert** — a reinforced-concrete drainage structure — as a dynamic 3D civil object (`AECC_CULVERT`) swept along a **flow-line** polyline. From a compact parameter set it builds the whole assembly: the box barrel (in segments), bottom and top slabs and haunches, the subbase and protection layers, inlet and outlet headwalls with splayed wing walls, footings and aprons, and an optional fall (drop) chamber — each as a solid part on its own layer, with a computed concrete volume.

Like a grading, a culvert is **self-drawing and dynamic**: it owns its part meshes and rebuilds whenever you edit its parameters or move the flow-line.

28.2.1 Creating a culvert

1. Draw the **flow-line** as a 2D or 3D polyline along the culvert axis (its vertices set the plan route; a 3D polyline also sets the invert levels). At least two vertices are required.
2. Run **`CULVERT`** (alias `CULV`), or **Civil › Culverts › Culvert**, and click the flow-line polyline when prompted (or pre-select it first).
3. The culvert is built with default box parameters and reported on the command line (*"Culvert created: N parts, total volume = X m³"*). Double-click it to open the editor and tune the design.

28.2.2 Editing — the Culvert editor

Double-click a culvert (or run **`CULVERTEDIT`**, alias `CULVEDIT`) to open the tabbed **Culvert editor**. Each tab draws a live cross- and long-section schematic that highlights the dimension you are editing:

- **Box** — the barrel: clear width **B** and height **H**, wall thickness **T**, top haunches, invert chamfer, bottom/top slab thickness, subbase and top-protection thickness, footing depth, and the segment length **L1** (split the barrel into *n* equal pieces, or into fixed-length pieces).
- **Left Wing / Right Wing** — the inlet and outlet wing walls: splay angle, wing length, footing thickness and top slope for each wing panel.
- **Fall Chamber** — an optional drop structure at the inlet or outlet (**None / Inlet / Outlet**) with its inner size, wall thickness, floor drop and a stepped ditch profile.
- **Materials** — the material name assigned to each component group (box, subbase, protection, wings, fall chamber); used to group the quantities.
- **Quantities** — a live table of concrete volume per component and material.

Use **Apply** to rebuild the culvert without closing the dialog, or **OK** to apply and close. **Load / Save** store a parameter set as a `.cvt` template; **Import** reads a legacy GCulvert `.txt` parameter file.

You can also edit the box parameters and flow-line vertices, and read the total volume, directly in the **Object Inspector** when a culvert is selected.

28.2.3 Moving a culvert

The grips are the flow-line vertices. Grip-drag a vertex to move it in plan (its level is kept) and the culvert rebuilds along the new route.

28.2.4 Sections, quantities and report

- **CULVERTSECTION** (alias CULVSEC) cuts the culvert into a dimensioned 2D **section** — a **cross-section** at a station you pick along the flow-line, or a **longitudinal** section (run it with the LONG option). It draws the outlines, material hatches and dimensions at a point you pick.
- **CULVERTQUANTITIES** (alias CULVQTY) reports the concrete volumes — grouped by component and by material, with a grand total — and places a **quantities table** on the drawing.
- **CULVERTREPORT** (alias CULVRPT) writes a **PDF report** of the selected culvert: identity, a plan sketch with numbered stake-out points, cross and long sections, a volumes table, and an X/Y/Z co-ordinates table. **Civil › Culverts › Culvert Report** on the menu and ribbon.

28.2.5 Viewing a culvert in 3D

Select a culvert and open the **3D Object Viewer** (**OBJVIEW**, below) to orbit the assembled solid, isolate individual parts and read part names and coordinates.

28.2.6 Culvert commands

Command	Alias	Description
CULVERT	CULV	Create a parametric box culvert from a picked flow-line polyline (built with default parameters; reports the part count and total volume).
CULVERTEDIT	CULVEDIT	Open the tabbed Culvert editor for a selected culvert (same as double-clicking it) to edit box, wing, fall-chamber and material parameters with a live schematic and quantities.
CULVERTSECTION	CULVSEC	Cut the culvert into a dimensioned 2D cross-section (or a longitudinal section with the LONG option) with outlines, material hatches and dimensions.
CULVERTQUANTITIES	CULVQTY	Report concrete volumes grouped by component and material and place a quantities table on the drawing.
CULVERTREPORT	CULVRPT	Write a PDF report of a selected culvert (identity, plan with numbered stake-out points, cross and long sections, volumes, X/Y/Z table).

28.2.7 Notes

- A culvert does **not** appear in the Civil Toolspace / Prospector. Reach it by clicking or double-clicking it on the drawing, or from the ribbon / menu / command line.
- The flow-line polyline drives the route — keep it in the drawing; the culvert references it.
- The angle and batter fields on the wing tabs are stored with the design but a few are reserved for a future solver revision; wing geometry currently follows the splay, length, footing and slope values.

28.3 3D Object Viewer

The **3D Object Viewer** is a separate, modeless window that isolates the objects you have selected and shows them in **true 3D** — orbit, shade, hide individual objects and pick to inspect. It is **read-only**: it never changes the drawing. It is ideal for checking a **surface**, a **grading** pad or a **culvert** (or several together) in three dimensions without leaving your 2D layout.

28.3.1 Opening the viewer

1. Select one or more objects that have 3D geometry (surfaces, gradings, culverts, 3D polylines).
2. Open the viewer by any of:
   - **`OBJVIEW`** (alias `OBJECTVIEWER`);
   - **View** ribbon → **3D** pane → **Obj Viewer**;
   - right-click the canvas → **Object Viewer…**.

The selected objects are isolated into the viewer. Objects with no 3D geometry are skipped (a note appears in the status bar).

28.3.2 Navigating

- **Left-drag** — pan.
- **Right-drag** — orbit (rotate).
- **Middle-drag** — pan.
- **Wheel** — zoom.
- **Preset views** — a dropdown of Top / Bottom / Front / Back / Left / Right and the four isometric corners (SW / SE / NE / NW).
- **Zoom Extents** — frame everything.
- **Ortho / Persp** — toggle parallel vs perspective projection.

28.3.3 Display modes

A dropdown selects how solids are drawn: **Wireframe**, **Hidden Line**, **Flat**, **Gouraud**, **Shaded + Edges** (default) and **X-ray**. Surfaces are rendered **double-sided and smooth-shaded**, so a TIN never drops triangles when you rotate it.

28.3.4 Object list — show / hide / isolate

The left panel lists one row per object (a multi-part culvert is a single row):

- **Checkbox** — show or hide that object.
- **Double-click a row** — *isolate* it (hide the others) and refit the view.
- **Show All** — show every object again.

Below the list, a summary panel reports the object, triangle and vertex counts and the XYZ extents.

28.3.5 Inspecting

Hover the cursor over the geometry and the status bar reads out the point under the cursor:

- for a **TIN / graded surface** triangle: XYZ, elevation, **slope %** and ratio, **aspect** (compass bearing) and triangle **area**;
- for a **culvert** part: the **part name** and XYZ.

28.3.6 Colours

Surface triangles are coloured by the surface's **elevation-band** style (the low-to-high gradient set in the Surface Style Editor); gradings and culverts use their entity colour. Lighting shades the result.

28.3.7 Live update

While the viewer is open it tracks the objects you loaded and **updates automatically** when you edit them in model space — grip-drag a surface point or a grading footprint, edit a culvert parameter — and the 3D view follows within a moment, keeping your camera angle and any hidden/shown state. A **Live** toggle (on by default) turns this off for a frozen view. The viewer follows the objects by handle, so they no longer need to stay selected.

Reload re-syncs to the current model-space selection if you have one, otherwise it refreshes the objects already shown.

28.3.8 Object Viewer commands

Command	Alias	Description
`OBJVIEW`	`OBJECTVIEWER`	Open the 3D Object Viewer on the current selection: orbit, shade, show/hide and isolate objects, pick to inspect, with live update while you edit in model space. Read-only.

29 Sites

A **site** is a named container for civil objects that share a topology — Civil 3D's own organising concept, and compatible with it. A site is not drawn: it has no geometry, no style and no layer. It exists so that parcels, grading groups and alignments can be grouped by the piece of ground they belong to, and listed that way in the Prospector.

Membership rules:

Object	Site membership
Parcels	Always belong to a site. A parcel added without one is stamped with the default site, `Site 1`.
Grading groups	Always belong to a site, with the same default.
Alignments	**Optional.** An alignment with no site stays in the top-level *Alignments* collection, which is where a road centreline normally belongs. Put an alignment in a site when it is part of that site's layout — a subdivision access road that defines parcel frontages, for instance.
Surfaces, points, corridors	Not site members. They live in their own top-level collections.

29.1 Sites in the Prospector

The Prospector's **Sites** node lists every site in the drawing, with the count in brackets. Expanding a site shows four folders, each with its own count:

- **Alignments** — the alignments assigned to this site, each with its element count and length.
- **Feature Lines** — reserved; feature lines are not yet site members.
- **Grading Groups** — the gradings in this site, with cut, fill and net volumes, or `[stale]` when a rebuild is pending.
- **Parcels** — the parcels in this site, with their areas.

Sites with nothing in them are still listed, so a site created ahead of the work it will hold does not disappear.

29.2 Creating a Site

Right-click the **Sites** node (or any site under it) and choose **Create Site…**, then type a name. Names are compared case-insensitively, so `Phase 1` and `phase 1` are the same site; creating a name that already exists simply selects it.

Sites are also created implicitly:

- Adding a parcel or grading with no site creates `Site 1` if it is not there already.
- Opening a drawing that names sites — an IQcad `.iqz` or a Civil 3D DWG — creates each one as it is read.

29.3 Assigning an Alignment to a Site

Right-click an alignment in the Prospector and choose **Move to Site…**. The dialog lists `<none>` plus every site in the drawing; `<none>` returns the alignment to the top-level *Alignments* collection. Typing a name that does not exist yet creates that site.

The tree refreshes immediately and the drawing is marked modified.

> **Note**
>
> Site membership is **membership only**. Unlike Civil 3D, assigning an alignment to a site does not subdivide the parcels it crosses, and moving an object between sites never changes its geometry.

29.4 Sites in the File Formats

Format	Behaviour
IQML (`.iqz`)	The site list is written with its names and descriptions, and each parcel, grading and alignment carries its `siteName`.
DWG	Civil 3D sites are read back with the objects that belong to them. Writing sites out as Civil 3D objects requires the **Civil 3D objects** survey-export mode — see *DWG Compatibility* (page 109).
LandXML	Sites are not part of the LandXML schema; parcels and alignments export without their membership.

Part VI

Point Cloud Module

Import, display, manage and section large point cloud datasets — up to hundreds of millions of points.

30 Point Clouds

IQcad can import, display, and export large point cloud datasets (hundreds of millions of points). Point cloud data is stored in an octree spatial index for efficient LOD (Level of Detail) rendering.

30.1 Importing a Point Cloud

Use **Point Cloud › Import Point Cloud** or the `IMPORTPOINTCLOUD` command.

Supported import formats:

Format	Description
LAS	ASPRS LAS 1.0–1.4. XYZ, intensity, classification, RGB, return number.
LAZ	LASzip-compressed LAS. Requires `laszip.dll` (Windows) or `liblaszip.so` (Linux) next to the executable.
XYZ / CSV / TXT	Delimited text: X Y Z [R G B] [Intensity] [Classification].
PTS / PTX	Leica interchange formats. Header with point count, then X Y Z I R G B.
PLY	Stanford PLY (ASCII and binary little-endian). Vertex x y z [red green blue].
E57	ASTM E2807 / libE57Format. Multi-scan files with XYZ, RGB and intensity. Requires the `libE57CWrapper` dynamic library next to the executable; if absent the format is silently disabled and does not appear in the import dialog.

After import, an `.iqpc` octree cache file is created next to the source file. Subsequent opens load from cache in seconds instead of re-processing.

30.2 Attaching a Cache

Use **Point Cloud › Attach Point Cloud Cache** or `ATTACHPOINTCLOUD` to load an existing `.iqpc` file directly, without the original source file.

30.3 Exporting a Point Cloud

Use **Point Cloud › Export Point Cloud** or `EXPORTPOINTCLOUD`.

Supported export formats: LAS, LAZ, XYZ, PTS, PLY, E57. The export runs in a background thread with a progress bar and cancel support.

E57 export emits a single scan with XYZ plus RGB and intensity (when the source cloud carries them). Intensity is normalised to the 0..1 Double range required by the E57 spec. As with import, E57 export needs the `libE57CWrapper` library at runtime.

30.4 Elevation Clipping

Use **Point Cloud › Elevation Clip** or the `PCCLIP` command to filter the visible points by elevation (Z) range.

- `PCCLIP 80 160` — show only points with Z between 80 and 160.
- `PCCLIP OFF` — disable clipping, show all points.
- `PCCLIP` — display current clip state and cloud Z range.

This is useful for viewing specific floor levels in building scans, isolating terrain from vegetation, or inspecting horizontal cross-sections.

The clip range is saved with the drawing (IQZ file).

30.5 Point Cloud Display

Six colour modes are available (set via Object Inspector):

- **True Colour** — uses RGB stored in each point.
- **By Elevation** — maps Z to a blue→red colour ramp.
- **By Intensity** — maps sensor return intensity to greyscale.
- **By Classification** — LAS classification codes with standard colours.
- **Single Colour** — flat colour for all points.
- **By Return Number** — first/second/third returns in red/green/blue.

Point size and attenuation are configurable. The adaptive LOD system automatically reduces detail at zoom-out to maintain interactive frame rates (default 5 million point budget).

For a live Z readout of the nearest point under the cursor as you move the mouse — the point cloud counterpart of the surface Elevation Tracker — see the **Elevation Tracker** section (`PCELEVTRACK` command) in *TIN Surfaces* (page 149).

30.6 Prospector Integration

The Civil Toolspace shows a **Point Clouds** node. Right-click on an individual point cloud for:

- **Show / Hide** — toggle visibility.
- **Zoom To** — fit the viewport to the point cloud bounds.
- **Delete** — remove the point cloud from the drawing (with confirmation, undoable via Ctrl+Z).

31

Point Cloud Management

Right-click a point cloud in the Civil Toolspace to access:

- **Show / Hide** — toggle visibility.
- **Zoom To** — zoom the viewport to the point cloud extents.
- **Export…** — export to LAS, LAZ, XYZ, PTS, PLY, or E57 format.
- **Rename…** — change the display name of the point cloud.
- **Delete** — remove the point cloud (undoable).

Point counts are displayed with thousand separators for readability.

32

Point Cloud Sections and Sheets

32.1 Point Cloud Sections

The point cloud section editor cuts arbitrary 2D sections out of a loaded point cloud — vertical (façade / wall elevation cuts) or horizontal (floor plan slices). Sections are persistent CAD entities with grips; each can be opened in its own floating viewer, line-traced into polylines, and committed back to the drawing as flat 2D geometry ready for plotting.

32.1.1 Workflow

1. **Create** the section — **`PCSECTION`** (alias **`PCS`**). Pick *Vertical* or *Horizontal*, then the geometry:
   - **Vertical**: two plan points define the cut line and view direction; slab thickness controls the depth of points to include either side of the plane. Z range can be auto-derived from the cloud bounds or set manually.
   - **Horizontal**: an elevation point (or typed Z value) plus two plan corners give a horizontal slab at that elevation.

   The section appears in the drawing as a translucent rectangle with grips at its corners and a thickness arrow. Drag the grips in plan to move or resize the cut; drag the thickness arrow to thicken or thin the slab.
2. **View** the section — **`PCSECTIONVIEW`** (alias **`PCSV`**). A floating viewer opens with the projected points in the cutting plane's local coordinates. Pan with the middle mouse button, zoom with the wheel. The viewer auto-refreshes whenever the underlying section's geometry or thickness changes — drag a grip in the main canvas and the viewer updates immediately.
3. **Trace** straight lines through the points — **`PCSECTIONTRACE`** (alias **`PCST`**), or click **Trace** in the viewer toolbar. Two modes:
   - **Simple** — clusters connected points and fits each cluster with a polyline (PCA + Douglas-Peucker simplification). Good for clean wall scans regardless of orientation.
   - **Manhattan** — the Simple pipeline plus a final pass that snaps near-orthogonal segments to the dominant axis. Use this when the building is mostly orthogonal in plan (most are); curved walls and deliberately diagonal walls are left untouched. Toggle in the viewer by clicking the **Snap** button before pressing **Trace**.

   Tolerance defaults to 2 cm (the polyline simplification threshold) — tighten for finer detail, loosen for cleaner output. The trace result overlays in the viewer in blue.

4. **Commit** the section to the drawing — **PCSECTIONCOMMIT** (alias **PCSC**). Pick a section entity, then an insertion point in model space, type a rotation angle (default 0°), and choose whether to add a section-name label. The traced polylines drop in flattened to the XY plane at the insertion point, ready to be dimensioned, hatched, or plotted.

   Re-running **PCSECTIONCOMMIT** produces an independent copy each time — committed geometry does **not** auto-update if the source section changes. Re-commit if you want refreshed output.

32.1.2 Tips

- Multiple sections can coexist in one drawing. Each remembers its own trace mode, tolerance, and last-computed polylines.
- Sections persist across save/reload via IQML. The trace cache is recomputed on demand and is not saved to file.
- Use the section's **Properties** panel (Object Inspector) to fine-tune slab thickness, trace tolerance, Manhattan-snap angle, or the maximum point cap for the viewer (default 500 000 — uniform-stride sample is used if the slab exceeds this).
- For multi-story buildings, copy a horizontal section and edit its elevation property to produce floor plans at every level. A dedicated batch command is planned for a future release.

32.2 Point Cloud Sections Along Alignment

A modeless viewer that generates a series of point-cloud cross-sections along an alignment at regular chainages, then drops them onto a grid **Section Sheet** ready for plotting. This is the alignment-driven counterpart to the single-section **PCSECTION** workflow above. The standalone *Tunnel Section Viewer* has been retired; all its capabilities — angular-bin filtering, plane-mode toggle, and tunnel section placement — are now native to this viewer under the **Tunnel** radio button.

Launched from **Civil → Point Cloud Sections Along Alignment** or by typing **PCSECTIONS** (alias **PCSS**) at the command line.

Workflow:

1. **Pick alignment** from the top combo box.
2. **Pick a point cloud** from the cloud combo box.
3. **Choose a mode:**
   - **Ground** — planimetric slab filter (lateral width, elevation band, percentile / mode pick rule, gap fill). Use for road, rail and surface scans.
   - **Tunnel** — angular-bin filter around the alignment center. Converts each point to polar coordinates (angle, radius) from the alignment axis, divides them into angular bins, and selects one representative point per bin. Use for tunnels and other tubular geometries. See the Tunnel Filter Panel subsection below for the eleven filter controls.
4. **Generate stations** — set start / end / step (and optionally *Include PIs*) and press **Generate** to populate the station list, or add stations manually with the **+/-** buttons.
5. **Preview** any station in the centre paint box; **Prev / Next** walk through the list.
6. **Place sections** in the drawing:
   - In **Ground** mode each station becomes a `TAECC_PointCloudSection` entity.

- In **Tunnel** mode each station becomes a `TAECC_TunnelSectionLine` entity (the same entity type produced by the retired standalone viewer).

7. **Generate sheet…** — see below.

32.2.1 Toolbar Controls

Two shared controls appear on the viewer toolbar regardless of mode:

- **Plane** combo (**Vertical / Perpendicular**) — selects how the cutting plane is oriented at each station. *Vertical* uses a plane perpendicular to the horizontal alignment projection (standard civil cross-section). *Perpendicular* tilts the plane to be truly perpendicular to the 3D alignment tangent, accounting for vertical grade — most useful for steep tunnel grades (>5 %). In Ground mode the Perpendicular option is not yet fully implemented: the viewer shows a warning and falls back to the vertical slice.
- **Absolute Z** checkbox — when unchecked (default) the vertical axis shows ΔZ relative to the alignment centre line. Checking it switches to absolute world Z. Toggling this control does not recompute the point data; it repaints instantly and resets the pan/zoom so the new extents are fully visible.

32.2.2 Axis Info Strip

Above the station slider a six-cell strip shows alignment data at the active station: **X**, **Y**, **Z** (world coordinates of the alignment at that chainage), **Dir** (azimuth in degrees), **Grade** (%), and **R** (radius of curvature, metres; blank on straight sections). The strip updates every time the station changes.

32.2.3 Slab Thickness (shared)

Above both filter panels sits a single **Slab thickness** spin edit (default 0.5 m, range 0.05–10 m). It controls the depth of the slab of points the extractor considers for the section: points within ± half-thickness of the cutting plane are kept. Thicker slabs include more raw points — the trace is more robust against sparse scans — at the cost of smearing longitudinal detail. The control is shared between Ground and Tunnel modes.

32.2.4 Ground Filter Panel

When **Ground** mode is selected the filter panel exposes seven controls. Ground extraction projects raw slab points onto the section plane (lateral offset U, elevation V), then sweeps lateral bins across the width and picks one V value per bin via a configurable percentile. Use this for road, rail, ground-surface or façade scans where a top-of-points profile is wanted.

Table 1: Ground Filter Parameters

Parameter	Default	Description
Lateral width	30 m	Total extent of the bin sweep across the section, centred on the alignment centre line (e.g. 30 m = ±15 m). Points beyond this lateral range are discarded before binning.
Bin size	0.5 m	Width of each lateral bin in metres. Smaller = more polyline vertices and tighter following of the surface; larger = a smoother, simpler polyline.
Elev below	3 m	Maximum metres below the alignment centre line to consider. Points lower than this are discarded before binning — keeps the slab clean of below-grade clutter.
Elev above	5 m	Maximum metres above the alignment centre line to consider. Clips out vegetation, overhead structures, or scan returns from above the surface.
Pick	Top point	Quick preset combo that drives the **Percentile** value: *Top points* (95 %), *Middle points* (50 %), *Bottom points* (5 %). Adjust **Percentile** manually for any other value.
Percentile	95	The percentile of the per-bin Y values that becomes the output vertex (0–100). 95 is the canonical *top of points* value; 50 picks the median; 5 picks the floor.
Gap fill	2 m	Bridges short empty runs in the bin sweep. When the next non-empty bin sits within this many metres of the last emitted vertex, intermediate empty bins are linearly interpolated; longer gaps leave a discontinuity in the polyline so missing-data regions remain visible.

The algorithm is a single pass: (1) clip points to the lateral and elevation windows, (2) bin them along the lateral axis, (3) sort each bin's Y values ascending and take the configured percentile, (4) emit one vertex per non-empty bin at *(bin centre X, percentile Y)*, (5) bridge short empty runs via linear interpolation.

The filtered ground polyline is rendered with the same 3-pass styling as the tunnel trace: black outline (width 5), white line (width 2), green vertex dots (radius 3).

32.2.5 Tunnel Filter Panel

When **Tunnel** mode is selected the filter panel exposes eleven controls:

Table 2: Tunnel Filter Parameters

Parameter	Default	Description
Mode	Outermost	Selection rule per bin: **Outermost** (excavation surface), **Innermost** (lining), **Median** (robust), **Percentile** (configurable)
Bin	5 deg	Angular bin width (2–15 deg). Smaller = more polyline vertices.
MinR / MaxR	0 / 15 m	Radius cutoff from alignment center. Excludes noise and over-range points.
MinPts	3	Minimum points per bin; bins with fewer points are skipped.
Pct	90 %	Percentile value (active only in Percentile mode).
Prox	100 %	Plane proximity — percentage of closest-to-chainage points to keep per bin. At 100 % all slab points are used; lower values keep only points nearest the exact cutting plane.
Inv.Skip	0 deg	Invert skip — excludes an angular zone at the bottom (tunnel floor). A shaded wedge labelled "SKIP" is drawn in the viewer.
MinElev / MaxElev	-999 / +999 m	Elevation cutoffs relative to the alignment center. A dashed line is drawn in the viewer at each active cutoff level.
Gap	0 m	Radial gap threshold per bin. When the gap between consecutive points exceeds this value, inner points (ducts, cable trays, equipment) are removed, keeping the outer cluster (rock surface). Set to 0 to disable.

The filtered result is rendered with 3-pass styling: black outline (width 5), white line (width 2), green vertex dots (radius 3).

32.2.6 Generating a Section Sheet

Pressing **Generate sheet…** hides the viewer and switches the main canvas into a one-shot pick mode (status bar prompt *"Pick insertion point (Esc to cancel):"*). Left-click anywhere in the drawing — object snap is honoured, so endpoint / midpoint / intersection snaps work for placement. The viewer reappears and a `TAECC_PointCloudSectionSheet` entity is added at the picked point, laying the placed sections out in a 4-column grid (rows are computed as `ceil(count / 4)`; cell width and height are derived from the filter's lateral-width setting).

Press **Esc** or **right-click** on the canvas to cancel — the viewer reappears with a *"Sheet generation cancelled."* message and no sheet is added. Closing the viewer mid-pick (e.g. from the taskbar) is also safe; any pending pick is dropped.

The sheet entity carries a **SheetKind** discriminator (Ground or Tunnel) that controls how cells are rendered and updated. Ground sheets store the `TAECC_PointCloudSection` handles; tunnel sheets store the `TAECC_TunnelSectionLine` handles plus the eleven tunnel filter parameters. Tunnel cell thumbnails are rendered using a PCA-derived local frame so the cross-section profile fills the cell correctly. Both kinds round-trip through IQML and the compact streaming format; legacy IQZ files load as Ground.

The Object Inspector shows a read-only **SheetKind** field under the **Source** category. The **Re-sample** action (Properties pane) re-fits every section in the sheet from scratch, using the stored filter parameters for tunnel sheets.

The sheet entity exposes its grid (Columns, Rows, Cell Width / Height, H/V Spacing), insertion point and

label format in the Object Inspector under the **Layout**, **Position** and **Display** categories, and the linked alignment and section handles under **Source**.

When a sheet is generated, it is placed on the civil-default layer `C-PCSC-{Name}` (yellow by default, see *Program Defaults Reference* (page 281)). All visual elements painted by the sheet — frame, axis guides, trace polylines and labels — default to **ByLayer** in the sheet style, so the entire sheet inherits that layer's colour until you override individual elements via the style editor below.

32.3 Point Cloud Section Sheet Styles

Every `TAECC_PointCloudSectionSheet` resolves its visual appearance through a named *style entity* (`TAECCPCSectionSheetStyle`) stored in a per-drawing library on the canvas. Each drawing ships with a built-in **Standard** style that cannot be renamed or deleted; you may add, duplicate, rename or delete any other style. The sheet picks its style via the **Style** dropdown in the Object Inspector (`Identity` category). Sheets whose `Style` is blank or names a missing style fall back to **Standard**.

32.3.1 Opening the Style Editor

Three entry points:

- **Civil › Styles › PC Section Sheet Styles…** menu — launches the modal editor.
- The `PCSECTIONSHEETSTYLES` command (alias `PCSSS`).
- **Civil › Civil Style Manager** — select the *Point Cloud Sections* node in the left tree to embed the same editor in the manager's right pane.

The editor has a left strip listing every style in the canvas's library (with *New*, *Duplicate*, *Rename*, *Delete* buttons — *Standard* is protected from rename and delete) and a tabbed right panel with five tabs: **Cell**, **Axis Guides**, **Label**, **Trace** and **Preview**.

32.3.2 Editable Fields

Table 3: Cell tab

Field	Meaning
Frame visible	Toggles the rectangular cell border.
Frame colour / weight / line style	Pen for the cell border.
Padding top / bottom / left / right (%)	Per-side inset of the trace inside each cell, as a percentage of cell height/width. Only used when Centering is *Bounding Box*.
Padding horizontal (%)	Symmetric horizontal inset used when Centering is *Center Line*.
Centering	*Bounding Box* fits the trace's data extents into the cell minus per-side padding; *Center Line* centres station U=0 on the cell's vertical midline and scales symmetrically about it.
Fill visible / Fill colour	Optional solid background colour for each cell.

Table 4: Axis Guides tab

Field	Meaning
Vertical axis visible	Draws a dashed line at station U=0 (the alignment centreline).
Vertical colour / weight / line style	Pen for the vertical guide.
Horizontal axis visible	Draws a dashed line at the section's elevation centroid (tunnel cells only).
Horizontal colour / weight / line style	Pen for the horizontal guide.

Table 5: Label tab

Field	Meaning
Label visible	Toggles the per-cell chainage label.
Label colour / font size (%)	Text colour and font height as a percentage of cell height (minimum 8 pixels).
Label format	Format string with a single `%s` placeholder for the formatted chainage (e.g. `STA %s` → `STA 0+020.000`).
Label position	One of *Top Left / Top Center / Top Right / Bottom Left / Bottom Center / Bottom Right*.

Table 6: Trace tab

Field	Meaning
Trace colour / weight / line style	Pen for the section profile polyline.
Vertex dots	Draws a filled dot at every vertex of the trace polyline.
Dot colour / radius (px)	Colour and pixel radius of those dots.

The **Preview** tab paints a synthetic cell (Ground or Tunnel, set by the radio group) using the current style values, so you can see the effect of every change before committing.

32.3.3 ByLayer Colour Picker

Every colour swatch in the editor opens a three-button dialog when clicked: **ByLayer**, **Pick Color…**, **Cancel**. Choosing *ByLayer* sets the field to the special `clByLayer` sentinel — the painter resolves it at draw time to the sheet entity's effective colour (layer colour or per-entity override). This is the default for every colour field in a freshly created style and matches the convention used by the other civil style editors (Tunnel Section Line, Surface, Alignment, Profile View).

32.3.4 Persistence

The library is serialised into the IQML / IQZ drawing file under the `<PCSectionSheetStyles>` element, alongside the tunnel-section-line styles and civil label styles. Sheets store only the style **name**; the resolution to a concrete style happens at draw time, so deleting a style automatically falls back to *Standard* without invalidating sheets.

Part VII

Drone Module

Drone Survey turns a GeoTIFF DEM and an orthophoto into survey points, breaklines, boundaries, a grid and a TIN, then sends that work into the drawing.

33 Drone Survey

Drone Survey turns a GeoTIFF DEM and an orthophoto into survey points, breaklines, boundaries, a grid and a TIN, then sends that work into the current drawing. It belongs to the Drone module.

Open it from **Drone › Drone Survey**, the **Drone Survey** button on the Drone ribbon, or the command **DRONESURVEY** (alias DSV). The window stays open while you work in the drawing.

33.1 DEM and orthophoto

Load DEM reads a GeoTIFF elevation grid, or builds one from a LAS or LAZ point cloud. The grid stays in this window. It is not inserted into the drawing as a DEM.

For a point cloud, set the cell size and which returns to keep. Ground uses LAS class 2. All points uses every class. Class keeps one classification from 0 to 255. Lowest stores the lowest Z in each cell, which is the bare-earth value when the cloud is classified. Average stores the mean Z of the points that landed in the cell. The suggested cell is about twice the mean point spacing. A cell that would build more than 16 million cells is refused. The LAS or LAZ file stays external, and the cell size, the class and the lowest-or-average choice are stored with the session.

A progress bar tracks the read, the terrain mesh, and the orthophoto. The window is shown before a saved session starts that load, so you can see the work coming in.

Load orthophoto drapes a JPEG, PNG, BMP or TIFF across the DEM extent. The image is stretched to that extent. It is not reprojected. Loading a new DEM clears the photo, and you load it again afterwards.

The view exaggerates height so flat sites still read as terrain. Elevations you pick and send are the DEM values, not the exaggerated display height. The picture on screen is a lighter mesh (the long side is capped at 400 cells). The full grid is still used for every elevation.

33.2 Moving around

Right-drag orbits, middle-drag pans, the wheel zooms, and left-drag pans.

Walk / Fly (the move icon after Zoom extents) stands you on the surface where you were looking.

- **W A S D** walk on the terrain. **Shift** runs.
- Right-drag or left-drag looks around. The wheel steps forward. Middle-drag slides.
- **F**, **Space** or **E** starts fly. **W A S D** then follow the view, **Q** descends, and **E** or **Space** climbs. Fly is faster than walk, and **Shift** is faster still.

- **F** returns to a walk. Flying into the surface lands you.
- **Esc** leaves Walk / Fly. Zoom extents, a new DEM, or the Point, Breakline or Boundary tool also leaves it. Orbit, pan and zoom are there again when the button is up.

33.3 Points, breaklines and boundaries

Pick a tool, then click the terrain. A short drag still pans. A click stores a sample.

- **Point** (P) stores a survey point. The number, the code from the **Code** box, and the DEM elevation are kept.
- **Breakline** (L) draws a line the TIN must follow. Double-click or Enter finishes it. Esc drops the open line. A breakline is a forced edge. It does not clip the surface.
- **Boundary** (B) draws a closed limit. Click near the first vertex (about two DEM cells, and never under 5 cm), double-click, or press Enter to close it. You need three vertices. Esc drops the open boundary.

The side list shows how many points, breaklines, boundaries and surfaces the window holds.

33.4 Grid and low-pass

Grid (G) samples the DEM on the **Spacing** in the toolbar. A dot or a comma is accepted. The spacing is at least 0.001 in the DEM units. Nodes land on multiples of that spacing.

Finished boundaries clip the grid. With no finished boundary, an open boundary must be closed first. With no boundary at all, the grid covers the DEM. More than 25 000 cells is refused. NoData cells are skipped, and a node within 1 mm of a point you already kept is skipped. Running the grid again replaces the grid points that have not been sent.

Low-pass (Shift+G) uses the same spacing and keeps the lowest DEM sample in each cell, so vegetation on a DSM is less likely to win. A cell with no sample falls back to the elevation at the node. Regular and low-pass points share one pool, so either run replaces the other. Both are ordinary survey points and use the **Code** box.

33.5 Surface and contours

Surface (S) builds one TIN from the survey points, including grid points, and from the finished breaklines. You need three points, or the vertices of a breakline.

One finished boundary clips the TIN to that ring. Several finished boundaries keep triangles whose centres fall inside any ring. An open boundary, and no finished one, must be closed first. Open lines are not part of the surface.

The first surface is named `Drone Surface`. Building again before you send replaces that surface and clears contours. Building again after a send creates `Drone Surface 2` and leaves the sent surface in the drawing.

The TIN is drawn on the terrain as a light-blue wireframe.

Contour (C) draws that TIN at the **Interval**. Major lines are five times the interval. More than 400 levels is refused. Contours are a picture in this window. They are not sent as their own objects, and a new surface or a new DEM clears them.

33.6 Send to the drawing

Send to Drawing writes what has not been sent yet:

- Survey points as civil points on layer `DRONE-POINTS`.
- Breaklines as 3D polylines on `DRONE-BREAKLINES`.
- Boundaries as closed 3D polylines on `DRONE-BOUNDARIES`.
- The TIN as a civil surface on `DRONE-SURFACE`.

The points go into a survey point group named for the surface, or `Drone Survey` when there is no surface yet. That group is listed on the surface definition, so the surface can be rebuilt from it later. Points you sent earlier in this window are added to the same group. The sent objects are one undo step. Anything already sent is skipped until you open Drone Survey on another drawing.

33.7 Sessions

A session is the survey on screen: the DEM and orthophoto paths, the points, the lines, the boundaries, and the spacing, interval and code. The TIN itself is not stored. A saved surface is rebuilt from the points and lines when you open the session.

Type a name in **Session** and press **Save**. Several names can sit in one drawing. **Open** loads the name in the box. **New** asks whether to save the survey on screen, then starts blank. **Delete** removes the named session after you confirm, and clears the screen.

Saving the drawing writes the sessions into the IQZ. The DEM and the orthophoto stay external files. Each path is stored relative to the drawing, and the absolute path is kept as well. Opening tries the relative path first. A missing file is reported, and the surface is not rebuilt when the DEM is missing. The live window is also written when you save the drawing and when you close the form. An empty window does not wipe a session already in the drawing. Save the drawing itself to keep the sessions in the file.

33.8 Keys

These keys work while the code, spacing, interval and session boxes are not focused. Ctrl and Alt are left to the system.

Key	Action
`P` / `L` / `B`	Point, breakline, boundary. Press again to leave the tool.
`G` / `Shift+G`	Regular grid, or low-pass grid.
`S` / `C`	Build the surface, or draw contours.
Enter	Finish the open breakline or boundary.
Esc	Leave Walk / Fly. Otherwise cancel the open chain, then leave the tool.
`W A S D`	Move, while Walk / Fly is on. `S` does not build the surface in that mode.
`F` / `E` / Space / `Q`	Fly, climb, or descend. See *Moving around*.

Note

A point-cloud DEM is a grid in this window. **`PCSURFACE`** still builds a TIN in the drawing from a point cloud that is already loaded.

Part VIII

Command Reference

Lookup material: every command recognised by the command line, the keyboard shortcuts, and the complete list of factory defaults.

34

Command Reference

Complete list of every command recognised by the IQcad command line. Type a command name or its alias and press **Enter**.

34.1 CAD Commands

34.1.1 File

Command	Alias	Description
NEW	—	Create a new empty drawing.
OPEN	O	Open a drawing file.
SAVE	—	Save the drawing to its current file. An untitled drawing prompts for a name and is then added to the recent-files list.
SAVEAS	—	Save the drawing under a new name. The chosen file joins the recent-files list.
CLOSE	—	Close the current drawing.
EXPORTDXF	—	Export the drawing to DXF.
EXPORTIMAGE	—	Export the drawing to an image file.
EXPORTPDF	—	Export the drawing to PDF. For sheet output use PLOTPDF, which honours the layout's page setup.

34.1.2 Drawing

Command	Alias	Description
LINE	L	Draw a line (click start and end points, or type coordinates).
POLYLINE	PL, PLINE	Draw a polyline (multi-segment, optional arc segments via A).
MULTILINE	ML	Draw a multiline (parallel strands using the current multiline style).
ARC	A	Draw a three-point arc (start, pass-through, end).
ARC_SCE		Arc: Start, Center, End (3 mouse clicks).
ARC_SCA		Arc: Start, Center, then mouse defines endpoint (angle).
ARC_SCL		Arc: Start, Center, then mouse defines endpoint (chord length).

continues on next page

Table 2 – continued from previous page

Command	Alias	Description
ARC_SEA		Arc: Start, End, then mouse defines arc midpoint (angle).
ARC_SED		Arc: Start, End, then mouse defines arc midpoint (direction).
ARC_SER		Arc: Start, End, then mouse defines arc midpoint (radius).
ARC_CSE		Arc: Center, Start, End (3 mouse clicks).
ARC_CSA		Arc: Center, Start, then mouse defines endpoint (angle).
ARC_CSL		Arc: Center, Start, then mouse defines endpoint (chord length).
ARC_CONT		Arc: Continue from last entity's endpoint.
CIRCLE	C	Draw a circle (centre + radius).
CIRCLE_CD		Circle: Centre, then mouse defines diameter.
CIRCLE_2P		Circle: Two points define the diameter endpoints.
CIRCLE_3P		Circle: Three points define the circumscribed circle.
ELLIPSE	EL	Draw an ellipse (centre, then X-radius, then Y-radius).
SPLINE	SP	Draw a spline through a series of control points.
RECTANGLE	REC	Draw a rectangle (two opposite corners).
POLYGON	POL	Draw a regular polygon (number of sides, centre, radius).
HATCH	H	Create a hatch fill area.
TEXT	T	Place single-line text.
MLEADER	MLD	Draw a multileader (arrowhead, landing, and text label).
ARROW	AR	Draw an arrow entity.
POINT	P, PT	Place a point marker.
XLINE	XL	Draw a construction line (infinite in both directions).
3DFACE	3F	Draw a 3-D face (3 or 4 corner points).
3DPOLY	3DP	Draw a 3-D polyline (vertices with X, Y, Z). Supports spline-fit (Catmull-Rom interpolation in full 3-D). Right-click to finish.
EXTRUDE	EXT	Extrude a closed 2D profile along the Z-axis to create a 3D polyface mesh. Accepts closed polylines, circles, ellipses, and rectangles. Prompts for extrusion height (positive = up, negative = down) and whether to close top and bottom with triangulated caps. The source profile is removed after extrusion. Supports pre-selection. Output: TCADPolyfaceMesh (DXF-compatible polyface mesh).
SWEEP	SW	Sweep a closed 2D profile along a path curve to create a 3D polyface mesh. Select a profile, then a path. Valid paths: open polyline, 3D polyline, arc, spline, or civil alignment. Options available before path selection: **T** (Twist) — rotate profile along path (degrees); **S** (Scale) — taper from 1.0 to end factor; **R** (Range) — specify start/end chainage for alignment paths; **I** (Interval) — fixed path sampling distance; **N** (Segments) — profile vertex count for circles/ellipses, or Douglas-Peucker simplification for polylines. Supports pre-selection (one profile + one path). Output: TCADPolyfaceMesh.
IMAGEATTACH	IA	Attach a raster image (PNG, JPG, BMP, TIFF). Enter file path or press Enter to browse. If an ESRI world file (`.jgw`, `.pgw`, `.bpw`, `.tfw`) exists, prompts `Use georeferenced placement? [Yes/No]`. **Yes** (default): places at the world-file origin automatically. **No**: interactive placement with pre-filled width and rotation.
BEZIER	BEZ	Draw a Bézier curve.

continues on next page

Table 2 – continued from previous page

Command	Alias	Description
TABLE	TBL	Place a table. Click an insertion point to create a default 3 × 3 table. After placement, edit cells, row/column counts, and style in the Object Inspector. Cells support formulas and clipboard paste from Excel / LibreOffice.
MTEXT	MT	Place multiline text.
SOLID	SO	Draw a 2D filled solid.
WIPEOUT	WO	Mask an area with a background-coloured polygon, hiding what lies beneath. Options: pick points and right-click to close (default); Polyline converts a polyline — the selected one, or it prompts for one — closing it automatically if it is open, and optionally erasing it; Frames turns every wipeout's outline on or off (the WIPEOUT-FRAME system variable).
-HATCH	BH	Create a hatch from the command line (no dialog).

34.1.3 Modify

Command	Alias	Description
ERASE	E	Delete selected objects. If no objects are pre-selected, prompts "Select objects:" — pick objects then right-click to confirm or press Escape to cancel (matches AutoCAD behaviour).
MOVE	M	Move selected objects (base point → destination point).
COPY	CO	Copy selected objects.
ROTATE	RO	Rotate selected objects about a base point.
SCALE	SC	Scale selected objects from a base point.
OFFSET	O	Offset an entity by a distance or through a point. Supports Line, Circle, Arc, XLine, Polyline, Ellipse, Spline, and Rectangle. The default distance is stored in the OFFSETDIST system variable. Sub-options: **T** = offset through a picked point, **E** = erase the source entity after offsetting, **U** = undo last offset. A phantom preview shows the offset result while picking the side.
MIRROR	MI	Mirror selected objects across a two-point axis.
TRIM	TR	Trim objects to cutting edges. Phase 1: select cutting edges (or type ALL). Phase 2: click the portion to remove.
EXTEND	EX	Extend a line to a boundary edge. Phase 1: select boundary edges. Phase 2: click the end to extend.
CHAMFER	CHA	Chamfer (bevel) two lines. Sub-option D sets chamfer distances ($CHAMFERA / $CHAMFERB). Zero distance trims to intersection.
FILLET	F	Fillet (round) two lines with an arc. Sub-option R sets the radius ($FILLETRAD). Zero radius trims to intersection.
BREAK	BR	Break an entity between two points. Pick the object (the pick point is the first break point), then "Specify second break point or [First point]:". **F** = re-pick first point, @ = break at a single point with no gap. Lines, arcs and polylines split into two entities (or are shortened at an end); a circle becomes an arc (the portion counter-clockwise from the first to the second point is removed); closed polylines become one open polyline. Single undo step restores everything.

continues on next page

Table 3 – continued from previous page

Command	Alias	Description
JOIN	J	Join lines, arcs and polylines. Collinear touching lines merge into one line; touching arcs on the same circle merge into one arc; other touching chains merge into a polyline. Arc segments are preserved as true bulge (curved) segments. Reports "N object(s) joined into M."
ALIGN	AL	Align objects using 1–2 source/destination point pairs: first pair translates; second pair adds rotation (Enter after the first pair = move only). Then prompts "Scale objects based on alignment points? [Yes/No] <No>". Objects picked as destination reference are not affected.
DIVIDE	DIV	Place point entities dividing an entity into N equal segments (2–32767). Open entities get N−1 interior points; closed entities (circles, closed polylines) get N points. Works on Line, Polyline, Arc, Circle, Ellipse, and Spline entities.
MEASURE	ME	Place point entities along an entity at a fixed interval, starting from the end nearest the pick point. The final partial segment remains. Same entity types as DIVIDE.
ARRAY	ARR	Rectangular, polar or path array of the selected objects. Prompts "Enter array type [Rectangular/Polar/PAth] <Rectangular>". Rectangular: rows, columns, row/column spacing (negative spacing allowed). Polar: centre point, item count, angle to fill <360>, rotate items Yes/No. Path: pick a path entity, Divide (count) or Measure (spacing) placement, align items with the path Yes/No. Creates plain copies (alias is ARR because AR belongs to ARROW).
PEDIT	PE	Edit a polyline. Select a polyline (or Line/Arc/Circle to convert). Sub-options: **C**lose / **O**pen, **J**oin, **W**idth, **E**dit vertex (N/P/M/I/D/S sub-mode), **S**pline, **D**ecurve, **R**everse, e**X**it. Supports both 2D and 3D polylines.
UNDO	U	Undo the last action.
REDO	—	Redo the last undone action.
COPYCLIP	CC	Copy the selection to the clipboard, whichever tab (model or a layout) is active. Stamps the clipboard with the source space and drawing scale so a later paste can auto-scale.
CUTCLIP	CX	Cut the selection to the clipboard, whichever tab (model or a layout) is active. Stamps the clipboard with the source space and drawing scale so a later paste can auto-scale.
PASTECLIP	CV	Paste entities from the clipboard onto whichever tab is active. A ghost of the entities follows the cursor; click (or type x,y) to place them, Esc, a right-click, or pressing Enter on an empty command line cancels. Pasting between model and a layout auto-scales by the drawing scale, so text and tables land at the size they plot at; viewports are never carried across the clipboard (copying leaves them out). Legacy clipboards without scale metadata paste unscaled.
PASTEORIG	—	Paste entities from the clipboard at their original coordinates, onto whichever tab is active. Same auto-scaling as PASTECLIP, but non-interactive.
DUPLICATE	DU	Duplicate the selected entities in place.
SELECTALL	SA	Select every entity on the active tab.

34.1.4 Super Tools

Express/CadTools-style helpers. Full workflow notes: *Super Tools* (page 119).

Command	Alias	Description
FLATTEN	—	Set Z = 0 on the selected objects. Press Enter at the selection prompt to flatten the whole drawing. Also zeros polyline Elevation. Reports how many objects changed. Undoable. Ribbon / menu: **Super Tools → Cleanup**.
PLSIMPLIFY	PLSIM	Douglas–Peucker vertex reduction on selected 2D/3D polylines. Prompts for a deviation tolerance (drawing units; default 0.01, session-remembered). 2D polylines use XY deviation; 3D polylines use 3D. Arc bulges densify then clear; spline-fit is cleared. Optional `PLSIMPLIFY 0.05` when objects are pre-selected. Undoable. Ribbon / menu: **Super Tools → Polylines**.
PLGETZ	PGZ	Sample Z from a surface or point cloud along selected paths (2D/3D polylines and **lines**; multi-select OK). Sample mode **Vertices** (default) / Interval / Both; **Vertex interval** (default 1) is prompted only for Interval or Both. Missing samples use **Z = 0**. 2D polylines are **replaced** by 3D polylines; 3D polylines and lines (vertices mode) update Z in place; densified lines become 3D polylines. Undoable. **Super Tools → Polylines → Get Z…**.
BESTFITLINE	BFL	Best-fit 3D line (PCA) through vertices of the selection (points, lines, polylines; multi-select). Creates a new LINE on the current layer spanning the point cloud. Needs ≥ 2 distinct points. Undoable. **Super Tools → Polylines → Best Fit Line…**.
DGRID	DG	Draw a coordinate grid in model space. A *Create Grid* dialog sets the spacing, text height, target layer, style (**Draw lines** or **Draw ticks**) and whether to label the coordinates; you then pick two opposite corners of the area. The grid is real geometry, so it plots and can be edited.

34.1.5 View

Command	Alias	Description
ZOOM	Z	Zoom the viewport. Sub-options: `E` = extents, `W` = window, `+` / `-` = in / out, `P` = previous.
ZOOMOBJECT	—	Pick an entity on the canvas; the viewport zooms to its bounding box with a small margin. Also available from the Civil Prospector right-click menu on Surface, Alignment, Profile View and Point Cloud nodes.
PAN	—	Activate real-time pan (or use middle-mouse drag).
REGEN	—	Force a full redraw of the canvas.
UCSICON	—	UCS icon display: `ON`, `OFF`, `ORigin`, `NOorigin`, `Size`.
3DORBIT	ORBIT	Interactive 3D orbit mode — rotate the view around the model.
PLAN	—	Return to the standard top-down plan view.
VPOINT	VP	Set the viewpoint to a preset direction.
VIEWTWIST	VT	Set a 2D view rotation angle.
ALIGNVIEW	AV	Align the view to two picked points on the canvas.

continues on next page

Table 5 – continued from previous page

Command	Alias	Description
PLANVIEW	—	Reset the view twist back to north-up (0° rotation).
DRAWORDER	DR	Display order menu. Sub-options: F = front, B = back, A = above, U = below (under).
FRONT	—	Bring selected objects to front (drawn last).
BACK	—	Send selected objects to back (drawn first).
ABOVE	—	Move selected objects above a reference object (click to pick).
BELOW	—	Move selected objects below a reference object (click to pick).
ZW	—	Zoom window (pick two corners).
REDRAW	—	Refresh the display.

34.1.6 Inquiry

Command	Alias	Description
LIST	LI	Print all properties of the selected object.
ID	—	Click a point and report its world X, Y, Z.
DIST	DI	Measure the distance between two clicked points.
AREA	AA	Measure an area (pick points or select an object).
ANGLE	AN	Measure an angle (pick vertex + two ray points).

34.1.7 Blocks

Command	Alias	Description
BLOCK	B	Create a block definition from selected entities.
INSERT	INS	Insert a block reference.
BLOCKS	BL	List all defined blocks in the drawing.
BEDIT	BE	Edit a block definition in-place.
BCLOSE	BC	Close the block editor and return to the drawing.
BLOCKMANAGER	BM	Open the Block Manager dialog.
EXPLODE	—	Explode blocks, multilines, and polyface meshes into individual entities.
MESH2SURFACE	M2S	Convert a polyface mesh entity into a TIN surface.
SURFACE2MESH	S2M	Convert a TIN surface into a polyface mesh entity.

34.1.8 Dimensions

Command	Alias	Description
DIMLINEAR	DIML	Create a linear (horizontal/vertical) dimension.
DIMALIGNED	DIMAL	Create an aligned dimension (measures true distance).
DIMANGULAR	DIMANG	Create an angular dimension.

continues on next page

Table 8 – continued from previous page

Command	Alias	Description
DIMRADIUS	DIMRAD	Create a radius dimension.
DIMDIAMETER	DIMDIA	Create a diameter dimension.
DIMORDINATE	DIMORD	Create an ordinate dimension.
DIMARC	—	Create an arc length dimension.
DIMBASELINE	DIMBASE	Create a baseline dimension series.
DIMCONTINUE	DIMCONT	Create a continue dimension chain.
DIMJOGGED	DIMJOG	Create a jogged radius dimension.
DIMSTYLE	—	Open the Dimension Style Manager dialog.

34.1.9 Settings & Drawing Variables

Rows whose description begins with "Set $NAME" are **system variables**, not standalone commands — change them with SETVAR <name> <value> (alias SV), for example SETVAR LUNITS 2. The remaining rows (UNITS, GRID, ORTHO, SNAP, OSMODE, LIMITS, MAKELAYER, …) are regular commands.

Command	Alias	Description
UNITS	—	Multi-step wizard to set linear/angular units and precision.
LUNITS	—	Set $LUNITS — linear format (1 = Scientific, 2 = Decimal, 3 = Engineering, 4 = Architectural, 5 = Fractional).
LUPREC	—	Set $LUPREC — linear precision (0–8 decimal places).
AUNITS	—	Set $AUNITS — angular format (0 = Decimal degrees, 1 = DMS, 2 = Grads, 3 = Radians, 4 = Surveyor).
AUPREC	—	Set $AUPREC — angular precision (0–8 decimal places).
ANGDIR	—	Set $ANGDIR — angle direction (0 = CCW, 1 = CW).
ANGBASE	—	Set $ANGBASE — base angle in degrees (0 = East, 90 = North).
INSUNITS	—	Set $INSUNITS — insertion units (0 = Unitless, 1 = Inches, 2 = Feet, 4 = mm, 5 = cm, 6 = m, 7 = km).
MEASUREMENT	—	Set $MEASUREMENT — 0 = Imperial, 1 = Metric.
UNITMODE	—	Set $UNITMODE — 0 = stacked display, 1 = single string.
LIMCHECK	—	Set $LIMCHECK — 0 = off, 1 = enforce drawing limits.
LIMITS	—	Set $LIMMIN / $LIMMAX — drawing limits (lower-left, upper-right).
LTSCALE	—	Set $LTSCALE — global linetype scale factor.
TEXTSIZE	—	Set $TEXTSIZE — default text height.
DIMSCALE	—	Set $DIMSCALE — dimension scale factor.
OSMODE	—	Set $OSMODE — object snap bitmask (1 = END, 2 = MID, 4 = CEN, 8 = NOD, 16 = QUA, 32 = INT, 128 = PER, 256 = TAN, 512 = NEA).
GRID	—	Set grid spacing or toggle ON / OFF (also **F7**).
GRIDUNIT	—	Set $GRIDUNIT — grid spacing (saved with drawing).
GRIDMAJOR	—	Set $GRIDMAJOR — minor lines between bold major lines (default 5).

continues on next page

Table 9 – continued from previous page

Command	Alias	Description
GRIDTEXT	—	Toggle $GRIDTEXT — X/Y coordinate labels on major grid lines.
GRIPSIZE	—	Set $GRIPSIZE — grip square half-size in pixels (default 5).
ORTHO	—	Toggle orthogonal constraint (also **F8**).
SNAP	—	Toggle grid snap (also **F9**).
ORTHOMODE	—	Set $ORTHOMODE — 0 = off, 1 = on (saved with drawing).
SNAPMODE	—	Set $SNAPMODE — 0 = off, 1 = on (saved with drawing).
PICKADD	—	0 = replace selection, 1 = additive selection.
PICKAUTO	—	0 = no implied window, 1 = implied windowing.
SELECTIONCY-CLING	—	Overlapping pick (0 = off, 1 = badge only, 2 = badge and list, the default). Negative values remember the last on-mode. Toggle with **Ctrl+Shift+W**.
LAYERS	LA	Open the Layer Manager dialog.
TABLESTYLE	—	Open the Table Style Manager (create, edit, and delete table styles).
SETVAR	SV	Set any system variable by name and value.
SCALELISTEDIT	SLST	Edit the scale list (add, remove, rename annotation scales).
CURRENTLAYER	CLAYER	Set the active layer: CURRENTLAYER name.
MAKELAYER	—	Create a layer: MAKELAYER name [color].
LINETYPES	LT	Open the Linetypes dialog.
TEXTSTYLE	ST	Open the Text Styles dialog.
PLOTSTYLE	—	Open the Plot Style dialog.
POINTSTYLE	—	Open the Point Style dialog.
PROPERTIES	PR	Toggle the Object Inspector (properties) panel.

34.1.10 Layout & Paper Space

Command	Alias	Description
LAYOUT	LO	Layout management. Sub-options: N = new, D = delete, L = list (* marks active), S = set active.
MVIEW	MV	Create a viewport in paper space (Fit or 2-point). Must be on a Layout tab.
MVSETUP	MVS	Reserved for viewport rotate/scale/title setup. **Not implemented in this release** — use the Object Inspector for viewport scale and rotation.
VPCLIP	—	Clip a viewport boundary to a custom shape.
VPFADE	VPF	Toggle viewport fade (dimmed display of inactive viewports).
VPSCALEBAR	VPSB	Toggle scale bar display in a viewport.
PAGESETUP	—	Open the Page Setup dialog for the active layout.
PLOTPDF	PDF	Unified PDF export command. Exports the active layout to PDF. Replaces the older EXPORTPDF alias.
PLOT	PRINT	Open the Print dialog (Ctrl+P).
PLOTALL	—	Export all layouts to a multi-page PDF.

34.1.11 Utility

Command	Alias	Description
PURGE	—	Remove all unused items (layers, blocks, styles, linetypes).
PURGELAYERS	—	Remove unused layers only.
PURGEBLOCKS	—	Remove unreferenced block definitions.
PURGESTYLES	—	Remove unused text, dimension, and table styles.
PURGELINETYPES	—	Remove unused linetypes.
AUDIT	—	Check drawing file integrity and report issues.
RECOVER	—	Fix drawing integrity issues found by AUDIT.
RECOVERCREATELAYERS	—	Recover and recreate any missing layers referenced by entities.
DWGPROPS	DRAWINGPROPS	Print drawing statistics (entities, layers, blocks, styles, civil objects, model summary) to the command line. Also available as a dialog on **File › Drawing Utilities › Properties…**.

34.1.12 Display Performance (experimental)

Command	Alias	Description
RETAINEDGEOMETRY	RETAINED	Toggle the retained-geometry display mode (ON / OFF, or no argument to toggle). Speeds up pan/zoom on very large drawings. Off by default.
RETLOD	RETAINEDLOD	Tune the retained level of detail live: RETLOD FACE <n> / DENSITY <n> / PTSIZE <n>; no argument prints the current values.
RETAINEDSTATS	—	Print retained-geometry diagnostics (bucket and vertex counts, level of detail, view mode).

34.1.13 Miscellaneous

Command	Alias	Description
HELP	?	Show command help in the command line.
MAPBG	—	Toggle background map tile layer behind the drawing canvas.
MAPSETTINGS	—	Open the Map Settings dialog to add, edit, delete, and select map tile providers (XYZ and WMS).

34.1.14 Diagnostics and Test Commands

Support and development aids. They are not part of a normal drawing workflow, but they are the fastest way to answer “is this my machine or the drawing?” when reporting a problem.

Command	Alias	Description
GPUTEST	—	Test the machine's GPU/VBO capability and benchmark it. Run this when display performance is poor, before changing the **GPU rendering** setting (*Application Settings* (page 113)).
ADDSAMPLEENTITIES	—	Add one of every entity type to the drawing. Useful for checking rendering, export round-trips and style changes against the full set.
ADDTUNNELSECTION	—	Add a sample tunnel section line.
ADDREALTUNNELSECTION	—	Add a real (MEIS) tunnel section line, for testing deviation analysis against measured data.

34.2 Civil Commands

34.2.1 Survey Points

Command	Alias	Description
SURVEYPOINT	SRVPT	Place an AEC survey point (Northing, Easting, Elevation).
TSLINE	TSL	Draw a tunnel section line (measured cross-section profile for deviation analysis).
TSOPTIMIZE	TSOPT	Optimize the fit between a measured tunnel profile and its theoretical section. Supports five optimization levels: 2P (translate), 3P (translate + uniform scale), 4P (translate + non-uniform scale), 5P (translate + scale + rotation), and per-vertex radial adjustment. Constraints: max overbreak/underbreak DR, max overbreak/underbreak area. Uses the Nelder-Mead simplex algorithm for 2P-5P modes.
SURVEYPOINTSTYLE	SPS	Edit survey point styles.
TSLADDPOINT	TSLADD	Add points to a tunnel section line.
TSLDELETEPOINT	TSLDEL	Delete points from a tunnel section line.
TSLCLOSEINVERT	TSLINV	Add invert points near the axis to close a tunnel section line.
TSLCLOSETOLEVEL	TSLCLV	Extend the section line ends to a target elevation.
TSLSORT	—	Sort tunnel section line points radially around the axis.

34.2.2 Surfaces

Command	Alias	Description
CREATESURFACE	CS	Create a TIN surface. If survey points exist in the drawing they are triangulated automatically; otherwise an empty surface definition is created so point groups and boundaries can be attached later.

continues on next page

Table 16 – continued from previous page

Command	Alias	Description
IMPORTDEM	IDEM	Import a GeoTIFF DEM (`.tif`) as a grid-backed surface. Decodes the raster natively — uncompressed, Deflate or LZW; single-band Float32 or Int16; strips or tiles — and georeferences it from the GeoTIFF tags (model tiepoint, pixel scale, CRS); no world file or external conversion is required. Elevations are answered by bilinear interpolation on the grid (no triangulation), so the DEM can be sampled by section sheets exactly like a TIN. NoData, voids and non-finite cells return no elevation (honest gaps, never fabricated ground). The CRS is reported but not reprojected — the DEM must already share the drawing's coordinate system. The surface is persisted in the drawing and re-decoded from its source file on reopen (a warning is shown if the file has moved).
CLOUDSURFACE	CLS	Build a grid surface from a loaded point cloud, for comparison against a DEM or design surface. Bins the cloud's ground points (LAS class 2) into a regular grid, keeping the lowest Z per cell (bare earth); if the cloud has no ground classification it falls back to all points and suggests running PCGROUND first. Cell size defaults to about twice the average point spacing; empty cells stay void (no fabricated ground from stray returns). Produces a surface named Cloud_<name> that can be sampled in a section sheet alongside the DEM, with cut/fill reported between the two — the intended way to validate a DEM against survey data.
DEMTOTIN	D2T	Resample a DEM surface into a **new** editable TIN surface, leaving the DEM intact. Pick the DEM, choose a region (whole DEM, a rectangle, or an existing closed polyline), a cell size, and an optional decimation tolerance. The DEM is sampled (bilinear) onto a regular grid tightened to the region — so a small high-detail region does not hit the whole-DEM cell cap — and triangulated; voids and out-of-region cells become gaps. The result is a normal TAECC_SURFACE named <DEM>_TIN (source type *Derived*) that can be edited, used in corridors, or contoured like any TIN. (Full-resolution DEMs are far too large to triangulate directly, so a sensible cell size is required.)
ADDSURFACE-POINTS	ASP	Add all survey points to a named surface and build (or rebuild) the TIN.
ADDSURFACEEN-TITIES	ASE	Add selected drawing objects to a surface. Supported entity types: 3DFace (loaded as TIN triangles directly), Line, Polyline, Spline, Point, and Survey Point (vertices triangulated). If only one surface exists it is used automatically; otherwise prompts for the surface name.
REBUILDSUR-FACE	RS	Rebuild an existing surface TIN from the current survey points.
ADDSURFACE-BOUNDARY	ASB	Add a boundary polyline to a surface (outer or hole).
DECIMATESUR-FACE	DS	Decimate (simplify) a TIN surface. Prompts for the surface name (auto-selects if only one exists) and a max elevation tolerance. Two algorithms are selected automatically: **grid-based O(n)** for surfaces > 100K points (single-pass min/max Z per cell, capped at 50K output points, < 5 sec for 4M points); **greedy insertion** for ≤ 100K points (convex hull seed, binary max-heap, batched refinement).

continues on next page

Table 16 – continued from previous page

Command	Alias	Description
DELETESURFACE	—	Delete a surface by name.
SURFACEPROP-ERTIES	—	Open the **Surface Properties** dialog (name, description, style, lock, and the Statistics tab). A selected surface is used; otherwise the command asks for the name. SURFACEPROPERTIES TEXT prints the short report instead: points, triangles, breaklines, elevation range, area, style, **source type** (Survey Points / Point Cloud / DEM / LandXML / Merged / Derived) and the edit-log count. The source type is also shown next to each surface in the Civil Toolspace tree.
EXPLODECON-TOURS	EXC	Extract a surface's contour lines into editable polyline entities, placed on <Name>-Contour-Major / -Minor layers at each contour elevation (DXF-exportable). Works for any surface with contours — including grid-backed DEM surfaces, whose contours are generated by marching squares directly on the elevation grid (no triangulation). If the surface style has no contour intervals set, the command reports that rather than producing nothing.
SURFDIFF	—	Compare two TIN surfaces to build a **difference (diff) surface** whose elevation is the vertical deviation dz between them. Opens the **Compare Surfaces** dialog (sources, cell size, tolerance, max deviation, sign convention, insert/stats options) with a live overlap/cell-count preview. The result is a deviation-banded surface DIFF_<A>-vs-<B>, saved with its comparison provenance so a volume report works later without the sources. Basis for earthwork QC.
PCDIFF	—	As SURFDIFF but compares a surface against a loaded point cloud (as-built). Adds a **PC reducer** choice (Median / Minimum / Mean) for collapsing multiple returns per cell to one ground value.
VOLUMEREPORT	VOL-RE-PORT	Earthwork **volume report** for a comparison (diff) surface: cut, fill and net volumes, cut/fill/total areas, deviation statistics, and a per-band breakdown, integrated directly from the diff TIN. Uses the selected diff surface, or prompts when several exist. Opens as an HTML report in the system browser.
MERGESURFACES	MS	Merge two surfaces (paste or union) into a new or replaced surface.
SURFACESTYLE	SUS	Edit surface styles.
SURFBUILDSET-TINGS	SBS	Edit per-surface build settings (triangulation algorithm and cleanup filters).
SLOPEPATH	SLP	Trace a maximum-slope gradient line along a sketched polyline. Select a 2D/3D polyline (or pre-select it); its vertices become waypoints and the command chains a gradient path through them across the surface, never exceeding the given maximum slope (%). Draws one 3D polyline draped on the terrain (collinear runs merged), keeps the sketch, and reports lengths, climb/descent, per-leg status and a per-vertex slope list.
FACE2SURFACE	F2S	Convert selected 3D faces into a surface.
ADDDXFFACES	ADF	Add 3DFACE entities imported from DXF to a surface. When the DXF carries 3DFACEs on several layers (typically one exported surface per layer), a dialog asks which layer(s) to import — either merged into the target surface, or, with **Create a separate surface per selected layer** checked, one new surface per layer named after it. Single-layer files import directly, no dialog.

continues on next page

Table 16 – continued from previous page

Command	Alias	Description
ELEVTRACK	—	Toggle the surface **Elevation Tracker** overlay — a live Z readout of every visible surface under the cursor. See *Elevation Tracker* (page 156).

34.2.3 Surface Editing

Command	Alias	Description
SURFADDPOINT	SAP	Add a single point to a surface TIN interactively.
SURFDELETE-POINT	SDP	Delete a point from a surface TIN.
SURFMOVEPOINT	SMP	Move a surface TIN point to a new position.
SURFMODI-FYELEV	SME	Modify the elevation of a surface point.
SURFSWAPEDGE	SSE	Swap a TIN edge between two adjacent triangles. Runs continuously (click edges one after another) until cancelled with Escape or right-click.
SURFADDLINE	SAL	Force an edge (breakline) between two surface points.
SURFDELETE-LINE	SDL	Remove a TIN edge.
SURFRAISELOWER	SRL	Raise or lower the entire surface by a specified elevation offset.
SURFUNDO	—	Undo the last surface edit operation.
SURFCLEARED-ITS	SCE	Clear the surface edit log.
SURFLISTEDITS	SLE	List all surface edit operations performed since the last clear.
SURFADDBREAK-LINE	SAB	Add the selected polyline(s) to the active surface as breaklines. Prompts for selection if nothing is pre-selected.
SURFDELETE-LONGEDGES	SDLE	Delete TIN edges longer than a threshold.
SURFHIDETRI	SHT	Hide TIN triangles: click inside each one, right-click to finish. The faces are kept in the edit log, not deleted.
SURFSHOWTRI	SWT	Restore every hidden triangle on the picked surface.
SURFREGRID	SRG	Resample a surface onto a regular grid and rebuild it as a gridded TIN.

34.2.4 Spot Elevations & Labels

Command	Alias	Description
SPOTELEV	SE	Place a spot elevation label at a clicked point on a surface.
ADDCONTOURLA-BEL	ACL	Place contour labels along surface contour lines with collision avoidance and uphill-readable rotation.

34.2.5 Alignments

Command	Alias	Description
ALIGNMENT	ALG	Create a horizontal alignment from PI (Point of Intersection) picks.
ALIGNREVERSE	—	Reverse the direction of an alignment.
ALIGNPROPS	—	Display alignment properties (element table, total length, etc.).
ALIGN-FROMPOLYLINE	AFP	Create an alignment from a selected polyline with optional curve radius at interior PIs. Accepts 2D and 3D polylines; when the vertices carry differing elevations, optionally also builds the vertical profile from the vertex Z values (one PVI per vertex, with an optional vertical curve length at interior PVIs).
ADDPI	API	Click to insert a horizontal PI into an alignment. After picking the alignment, prompts once for a default curve radius (and, if > 0, a spiral length and type); the defaults are applied to every PI clicked in the run. Enter through the prompts for sharp radius-0 PIs. Enter finishes; every click is one undo step.
DELPI	DPI	Click a PI to delete it (nearest PI within the pick tolerance). End PIs may be deleted; an alignment keeps at least 2 PIs.
EDITPI	EPI	Click a PI to edit its curve radius and spiral (current values as defaults); end PIs have no curve.
ALIGNMENTRE-PORT	ALR	HTML alignment report, opened in the browser: plan sketch, PIs, horizontal elements, curve and spiral data, profile sketch, PVIs, vertical curve data, cross-section model and a station table.
ALIGN-MENTSTYLE	ALS	Edit alignment styles.
POINTCHAINAGE	PCH	Report chainage and offset of picked points against an alignment.

34.2.6 Profiles

Command	Alias	Description
QUICKPROFILE	QP	Create a quick profile along a polyline or line by sampling a surface. Select source entity, then surface, then click insertion point.
CREATEPRO-FILEVIEW	CPV	Create a profile view from an alignment. Select alignment, then surface, then click insertion point. Samples ground profile and includes design vertical alignment if present.
DESIGNPROFILE	DP	Create a design profile (vertical alignment) inside a profile view. Click PVIs in the view, then enter curve lengths at each interior PVI.
ADDPVI	APV	Click inside a profile view to insert a design-profile PVI (curve length 0) at the clicked station/elevation, kept in station order.
DELPVI	DPV	Click a design-profile PVI in a profile view to delete it; deleting all PVIs clears the design line.
PROFILEPROPS	—	Display profile view properties (station range, elevation range, profile count).

34.2.7 Cross-Sections

Command	Alias	Description
CREATESECTIONSHEET	CSS	Create a cross-section sheet: select alignment, surfaces, interval, widths, grid layout, cut/fill surfaces, and click insertion point.
EARTHWORKVOLUMES	EWV	Compute and display earthwork volume table using the average end area method (cut/fill areas, volumes, cumulative totals).
CREATESECTIONSHEETDLG	CSSD	Create a cross-section sheet via dialog.

34.2.8 Parcels

Command	Alias	Description
CREATEPARCEL	PARCEL	Create a parcel from a closed polyline. Prompts for a parcel name, then click a closed polyline to define the boundary. Area and perimeter are computed automatically, and every corner is assigned a vertex name (see *Parcel Vertex Names* (page 177)).
PARCELCHECK	PCHK	Check-and-fix hub for parcel vertex issues. Sub-options **Report** (default, read-only listing of duplicate vertices, unnamed vertices, mismatched shared corners, and duplicate names), **Fix** (repairs everything Report finds; offers to set the drawing's vertex prefix), **reNumber** (confirmation-guarded full drawing-wide re-sequence), and **Prefix** (directly change the drawing's vertex-name prefix; affects newly assigned names only).
PARCELTABLE	—	Insert a coordinate table (Point/X/Y/Dist. plus Perimeter and Area) for the selected parcels, or all parcels if none are selected. The Point column shows each corner's vertex name, falling back to its index when unnamed.
PARCELREPORT	—	Generate an HTML report (sketch plus coordinate table) for the selected parcels, or all parcels if none are selected, and open it in the system browser.

34.2.9 Assemblies & Corridors

Command	Alias	Description
ASSEMBLIES	—	Open the Assembly Editor.
SAMPLEASSEMBLIES	SAS	Seed the assembly library with the six standard road templates (same set as the shipped standard-roads.iqal).
BUILDCORRIDOR	BC	Build the corridor for an alignment. Prints cut/fill, excavation-to-formation and per-material quantities.
CORRIDORAUTO	CA	Toggle corridor auto-rebuild (live update on alignment/surface edits).

34.2.10 Grading

Command	Alias	Description
GRADING	GRD	Open the Grading dialog and create/build a grading: project cut/fill slopes from a footprint polyline at a design elevation out to a target surface, producing a graded surface + cut/fill volumes. Supports flat / vertex-Z / drainage pad elevation, cut & fill slopes (as ratio or %), and height-triggered benching with an optional bench cross-slope. The grading is a self-drawing dynamic object that auto-rebuilds when its footprint / target / criteria change.
BUILDGRADING	BG	Rebuild a stale grading (used when Auto rebuild is turned off).
GRADINGBAL-ANCE	GRDB	Solve the pad's design elevation for a target net cut/fill volume (default 0 = balanced earthwork). Prompts for the target and a tolerance; reports the solved elevation, or that the target is not achievable for the pad. Flat/drainage pads only.
GRADINGEX-TRACT	GRDX	Clone a grading's internal graded surface into a standalone editable AECC_SURFACE (<Grading>_Extracted) for sectioning, contouring, differencing or corridor use.

34.2.11 Culverts

Command	Alias	Description
CULVERT	CULV	Create a parametric box culvert from a picked flow-line polyline (box barrel, slabs, subbase/protection, headwalls, wing walls, aprons and an optional fall chamber), built with default parameters; reports the part count and total volume.
CULVERTEDIT	CUL-VEDIT	Open the tabbed Culvert editor for a selected culvert (same as double-clicking it): edit box, wing, fall-chamber and material parameters with a live schematic and a live quantities table.
CULVERTSEC-TION	CUL-VSEC	Cut the culvert into a dimensioned 2D cross-section (or a longitudinal section with the LONG option), with outlines, material hatches and dimensions placed at a picked point.
CULVERTQUAN-TITIES	CUL-VQTY	Report concrete volumes grouped by component and material (grand total) and place a quantities table on the drawing.
CULVERTREPORT	CUL-VRPT	Write a PDF report of a selected culvert: identity, plan sketch with numbered stake-out points, cross and long sections, a volumes table, and an X/Y/Z coordinates table.

34.2.12 3D Object Viewer

Command	Alias	Description
OBJVIEW	OBJECTVIEWER	Open the read-only 3D Object Viewer on the current selection (surfaces, gradings, culverts, 3D polylines): orbit/pan/zoom, display modes, show/hide and isolate objects, hover to inspect elevation / slope / aspect / area / part name, with live update while you edit in model space.

34.2.13 Point Groups

Command	Alias	Description
ADDTOGROUP	ATG	Add selected civil points to a point group. Select points first, then run the command. Lists existing groups by number; type a number to pick one or type a new name to create a group.
REMOVEFROMGROUP	RFG	Remove selected civil points from a point group. Select points first, then run the command and pick the group.
CREATEGROUP	CRG	Create a new point group. Prompts for a name, then lets you click individual points to add. Press Enter to finish.
CONVERTPOINTS	CPT	Convert selected AutoCAD point markers into AEC civil survey points. Optionally assigns a group and deletes the original point entities.

34.2.14 Coordinate Transformations

Command	Alias	Description
CSTRANSFORM	CST	Global coordinate transformation using common/control points. Supports Helmert 2D (4-parameter similarity), Affine 2D (6-parameter), and Polynomial degree 1/2/3. Interactive point picking on canvas, keyboard coordinate entry (`E,N` or `E,N,Z`), survey point lookup (`#PointNumber`), and file loading (`.cpt` JSON / `.csv`). Produces least-squares residuals, RMS statistics, and transformation reports (`.trpt`). Common points can be saved for reuse.
RUBBERSHEET	RBS	Non-uniform rubber-sheet transformation using control points. Two modes: **Linear** (Delaunay piecewise affine — C0 continuous) and **Smooth** (Thin Plate Spline — C2 continuous). Both achieve exact interpolation at control points (zero residual). Points outside the convex hull are extrapolated using the nearest triangle (linear) or affine far-field (smooth). Reports outside-hull count. Saves common points and transformation reports.

34.3 Point Cloud Commands

Command	Alias	Description
IMPORTPOINT-CLOUD	IPC	Import point cloud file (LAS, LAZ, XYZ/CSV/TXT, PTS/PTX, PLY, E57). Opens a file dialog, reads points, builds octree, creates `.iqpc` cache.
ATTACHPOINT-CLOUD	APC	Attach an existing `.iqpc` cache file without re-importing.
EXPORTPOINT-CLOUD	EPC	Export point cloud to file. Supported formats: LAS, LAZ, XYZ, PTS, PLY, E57. Runs in background with progress.
PCCLIP		Elevation clip for point clouds. `PCCLIP <zmin> <zmax>` enables clipping to the specified Z range. `PCCLIP OFF` disables. `PCCLIP` alone shows current state.
PCSECTION	PCS	Create a point-cloud section (vertical or horizontal cut) as a persistent CAD entity. Prompts for kind, two pick points, slab thickness, Z-range mode, and section name. See *Civil → Point Cloud Sections* for the full workflow.
PCSECTIONVIEW	PCSV	Open the floating viewer for a selected section entity. Multiple sections can have viewers open simultaneously; each viewer auto-refreshes when the underlying section changes.
PCSECTION-TRACE	PCST	Run line-tracing on a section. Three modes: `None` (clear cache), `Simple` (PCA + Douglas-Peucker), `Manhattan` (orthogonal-axis snap on top of Simple). Tolerance defaults to 2 cm. Equivalent to clicking **Trace** in the viewer toolbar.
PCSECTIONCOM-MIT	PCSC	Flatten a section's traced polylines into the drawing at a picked insertion point with optional rotation and section label. Each commit produces an independent copy; committed geometry does not auto-update when the source section changes.
PCSECTIONS	PCSS	Open the modeless *Point Cloud Sections Along Alignment* viewer. Supports **Ground** mode (planimetric slab filter) and **Tunnel** mode (angular-bin filter around the alignment center — the retired standalone Tunnel Section Viewer is subsumed here). Shared toolbar controls: **Plane** combo (Vertical / Perpendicular) and **Absolute Z** toggle. **Generate sheet…** places the results on a grid `TAECC_PointCloudSectionSheet` entity via canvas pick; tunnel sheets carry a `SheetKind` discriminator and store tunnel filter parameters for re-sampling. See *Civil → Point Cloud Sections Along Alignment*.
PCGROUND	PCG	Slope-based ground classifier for a loaded point cloud (marks LAS class 2 ground points).
PCSURFACE	PC-SURF	Build a surface from a point cloud (filter by classification, voxel decimate).
PCSECTION-SHEETSTYLES	PCSSS	Edit point-cloud section sheet styles.
PCELEVTRACK	—	Toggle the point cloud **Elevation Tracker** overlay — a live Z readout of the nearest point in every visible point cloud under the cursor. Shares the surface tracker's overlay box. See *Elevation Tracker* (page 156).

34.4 Drone Commands

Command	Alias	Description
DRONESURVEY	DSV	Open the Drone Survey workspace. Load a GeoTIFF DEM or build one from LAS or LAZ, drape an orthophoto, pick points, breaklines and boundaries, sample a regular or low-pass grid, build a TIN and contours, and send them into the drawing. Named sessions are stored in the IQZ. Requires the Drone module. See *Drone Survey* (page 249).

34.5 Scripting & Automation

Command	Alias	Description

SCRIPT	SCR	Run a text script file of commands. Invoke SCRIPT with no arguments to open a file picker (*.scr), or SCRIPT <path> to load a specific file directly (relative paths resolved against the working directory; quotes around paths with spaces are tolerated). Also reachable from **Tools → Run Script…**.
		File format — one input per line, fed to the dispatcher as if typed at the prompt. Lines beginning with ; are comments (leading whitespace allowed). Blank lines act as Enter, which repeats the last command — same as AutoCAD's empty-Enter behaviour. No space-as-Enter, no LISP, no DIESEL.
		Typical drawing pattern:
		`LINE`
		`0,0`
		`@10,0`
		
		`CIRCLE`
		`20,0`
		`25,0`
		Three sample scripts ship under samples\scripts\ — hello.scr (single-shot view commands, smoke test), setup_view.scr (reset to north-up zoomed extents), and draw_demo.scr (LINE + CIRCLE driven by typed coordinates).

PYTHON	PY	Run a Python script file (.py) in the embedded interpreter.
PYEVAL	—	Evaluate a Python expression and print the result.
PYLOAD	—	Load a Python module (session-resident).
PYSCRIPTS	—	List registered Python commands.
PYPIP	—	Run pip against the bundled Python (e.g. PYPIP install <pkg>).
PASCAL	PAS	Run a Pascal Script file (.pas).
PASEVAL	—	Evaluate a Pascal Script expression.
PASLOAD	—	Load a Pascal Script module (session-resident).
PASSCRIPTS	—	List registered Pascal Script commands.
BLADE	BLADERNER	Open the Blade Runner script editor.

continues on next page

Table 31 – continued from previous page

Command	Alias	Description
PLUGINS	PLUG-IN-MAN-AGER	Open the Plugin Manager dialog.

35 Keyboard Shortcuts

35.1 File

Shortcut	Action
Ctrl+N	New drawing
Ctrl+O	Open drawing
Ctrl+S	Save drawing
Ctrl+Shift+S	Save As…
Ctrl+W	Close drawing (opens a new blank if it was the last tab)
Ctrl+P	Print / Export PDF
Alt+F4	Exit IQcad

35.2 Edit

Shortcut	Action
Ctrl+Z	Undo
Ctrl+Y	Redo
Ctrl+A	Select all
Ctrl+D	Duplicate selected
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+V	Paste
Ctrl+Shift+V	Paste to Original Coordinates (places pasted objects at their original world position instead of requiring an insertion point)
Ctrl+Shift+W	Toggle selection cycling (overlap badge / pick list)
Del	Erase selected objects

35.3 View

Shortcut	Action
+	Zoom in
−	Zoom out
Ctrl+Shift+E	Zoom extents (fit all objects)
Ctrl+Shift+G	Toggle grid on/off

35.4 Format

Shortcut	Action
Ctrl+Shift+D	Point Style editor
Ctrl+Shift+T	Text Style editor
Ctrl+Shift+M	Multiline Style editor

35.5 Drawing Tools

These aliases are entered in the command line (type and press **Enter**):

Key	Alias	Tool
L	LINE	Line
PL	PLINE	Polyline
ML	MLINE	Multiline
A	ARC	Arc
C	CIRCLE	Circle
EL	ELLIPSE	Ellipse
SP	SPLINE	Spline
REC	RECTAN-GLE	Rectangle
H	HATCH	Hatch
T	TEXT	Text
MLD	MLEADER	Multileader
PO	POINT	Point marker
XL	XLINE	Infinite construction line

35.6 Modify Tools

Alias	Tool
E	Erase
M	Move
CO	Copy
RO	Rotate
SC	Scale
O	Offset
MI	Mirror
TR	Trim
EX	Extend
CHA	Chamfer
F	Fillet
PE	PEDIT (polyline edit)

35.7 General

Key	Action
Escape	Cancel current operation; return to Select mode
Enter / Space	Confirm current prompt or repeat last command
F1	Open this User Manual (HTML help)

36

Program Defaults Reference

This chapter lists every factory default in IQcad. Use it as a quick reference when you need to know the out-of-the-box value for any setting, style, or variable. All values are applied when you create a new drawing or choose **Reset to Defaults** in a settings dialog.

36.1 Application Settings

Table 1: Window & UI

Setting	Default	Notes
Theme	MetroDark	SpkToolbar ribbon theme
Menu bar visible	True	Toggle with **File › Toggle Menu Bar**
Debug log	False	Writes to `iqcad_debug.log`
Prospector width	220 px	Left panel
Inspector width	360 px	Right panel
Command panel height	120 px	Bottom panel

Table 2: Canvas Display

Setting	Default	Notes
Background colour	Black	
Crosshair size	50 px	Arm length
Grip size	5 px	
Pick aperture	5 px	Half-side of pick box
Snap colour	Yellow	
OSNAP aperture	15 px	
OSNAP mode	575	END + MID + CEN + NOD + QUA + INT + NEA

Table 3: Ruler

Setting	Default	Notes
Visible	True	
Size	28 px	
Colour	clBtnFace	System button-face colour
Text colour	Black	
Opacity	200	0 = transparent, 255 = opaque

Table 4: Grid

Setting	Default	Notes
Visible	True	
Grid size	10.0	World units
Major interval	5	Lines per major grid
Grid text	False	Coordinate labels on major lines
Grid style	Lines	Lines / Dashed / Dots

Table 5: UCS Gizmo

Setting	Default	Notes
Visible	True	
At origin	False	Corner placement by default
Size	80 px	
Corner	Bottom-left	
Background opacity	80	0–255

Table 6: North Arrow

Setting	Default	Notes
Visible	False	
Style	Simple	Simple / Compass Rose / Ornate / Circle
Corner	Top-right	
Size	48 px	
Colour	Black	

Table 7: ViewCube

Setting	Default	Notes
Visible	True	
Corner	Top-right	
Size	100 px	

Table 8: DWG

Setting	Default	Notes
Reader backend	`native`	`native` | `libredwg-cli` | `oda-cli`; the CLI converters stay in the fallback chain
Save version	`R2018`	`R2000` | `R2004` | `R2007` | `R2010` | `R2013` | `R2018`
Survey export mode	`labelled`	`civil` | `labelled` | `geometry` | `omit` — see *DWG Compatibility* (page 109)

Table 9: Rendering

Setting	Default	Notes
GPU mode	`Auto`	`Auto` | `On` | `Off`; `GPUTEST` reports what the machine supports

36.2 Drawing Variables

These are per-drawing system variables, equivalent to AutoCAD system variables. They are stored with each drawing.

Table 10: Units & Angles

Variable	Default	Notes
LUNITS	2 (Decimal)	Linear unit format
LUPREC	4	Linear unit precision
AUNITS	0 (Decimal Degrees)	
AUPREC	0	Angular precision
ANGBASE	0.0	Angle base (east)
ANGDIR	0	0 = CCW, 1 = CW
INSUNITS	4 (Millimetres)	5 = cm, 6 = m, 7 = inches
MEASUREMENT	1 (Metric)	0 = Imperial

Table 11: Linetype & Point Display

Variable	Default	Notes
LTSCALE	1.0	Global linetype scale
CELTSCALE	1.0	Current entity linetype scale
MSLTSCALE	1	Model-space linetype scale
PDSIZE	0.15	Point marker size (world units)
PDMODE	Cross + Circle	Point display style
WIPEOUTFRAME	0 (off)	Draw the boundary outline of every wipeout. Drawing-wide, not per entity; the `WIPEOUT` command's `Frames` option sets it.
TEXTSIZE	2.5	Default text height
DIMSCALE	1.0	Overall dimension scale

Table 12: Drawing Limits

Variable	Default	Notes
LIMMIN	0, 0	Lower-left limit
LIMMAX	420, 297	Upper-right limit (A3 paper)
LIMCHECK	0	No limit enforcement

Table 13: Hatch Defaults

Variable	Default	Notes
HPNAME	SOLID	Default hatch pattern
HPSCALE	1.0	
HPANG	0.0	Pattern angle
HPASSOC	True	Associative hatch
HPMAXLINES	1 000 000	Line limit

Table 14: Selection

Variable	Default	Notes
PICKADD	1 (add)	0 = replace selection, 1 = additive
PICKAUTO	1 (on)	Implied windowing on empty-space click
SELECTIONCYCLING	2 (badge + list)	Overlapping pick: 0 = off, 1 = badge only, 2 = badge and list. Application-wide. Toggle with **Ctrl+Shift+W**.

36.3 Layer Defaults

Every new layer starts with these defaults:

Property	Default	Notes
Colour	White	
Linetype	Solid	
Line width	1	
Visible	True	
Frozen	False	
Locked	False	
Printable	True	
Transparency	0	0–90 %
Plot style	Normal	

36.4 Text Styles

Table 15: Built-in Text Styles

Name	Font	Bold	Italic	Description
Standard	Arial	No	No	Default text style — variable height
Title	Arial	Yes	No	Bold title text
Subtitle	Arial	No	Yes	Italic subtitle text
Annotative	Arial	No	No	Annotative text style
Mono	Courier New	No	No	Monospaced (dimensions, tables)

36.5 Dimension Styles

Table 16: Standard Dimension Style

Property	Default	Notes
Arrow type	Closed filled	Both sides + leader
Arrow size	2.5	
Text height	2.5	
Text placement	Above, centred	
Text gap	0.625	
Baseline spacing	3.75	
Extension line extend	1.25	
Extension line offset	0.625	
Decimal precision	2	
Overall scale	1.0	
Unit format	Decimal	
Angular format	Decimal degrees	

36.6 Surface Styles

Table 17: Standard Surface Style

Component	Colour	Weight	Notes
Border	Yellow	2	Visible by default
Triangles	Grey	1	Hidden by default
Major contours	Brown (180, 60, 0)	2	Interval: 5.0
Minor contours	Dark green (0, 140, 0)	1	Interval: 1.0
Points	Gold (255, 200, 0)	—	Hidden by default, size 3
Contour labels	—	—	Style: Contour - Standard, interval 50
Spot elevations	—	—	Style: Spot Elevation - Standard
Elevation bands	—	—	Hidden by default. Use the Elevation Bands tab in the Surface Style Editor to generate bands: set count, choose a color ramp (Green→Red, Blue→Red, Grayscale), and click **Generate Bands from Surface**. This opens a small dialog — a **Surface** dropdown lists every surface in the drawing (preselected to one already using the style being edited, else the first surface), and **Min elevation / Max elevation** boxes are pre-filled from that surface's actual range. Edit either value to an exact figure before generating, or accept the surface's own range as-is; switching the surface dropdown refills both boxes. GPU rendering provides smooth gradient interpolation between bands.
Elevation band legend	—	—	Hidden by default (`ShowLegend=False`). Enable per-style in the Surface Style Editor's Elevation Bands tab.

Built-in presets:

- **Standard** — contours + border visible, triangles hidden
- **Triangles & Contours** — both triangles and contours visible
- **Contours Only** — border hidden
- **Elevation Banding** — contours + elevation bands visible (generate bands via style editor)
- **Empty** — all components hidden

36.7 Alignment Styles

Table 18: Standard Alignment Style

Component	Colour	Weight	Notes
Centreline	Red	2	Linetype: ByLayer
Station ticks	Red	1	Major 100 m, minor 20 m
PI markers	Red	—	Size: 2.0
PI lines	Green	1	Hidden, dashed linetype
Direction arrows	Red	—	Interval 200 m, size 2.0
Label text	White	—	Height: 2.0 mm

Built-in presets:

- **Standard** — centreline + ticks + PI markers + arrows
- **Centreline Only** — ticks, PI markers, arrows hidden
- **Design Phase** — centreline as CENTER linetype, PI details + PI lines visible
- **Empty** — all components hidden

36.8 Parcel Styles

Table 19: Built-in Parcel Styles

Style	Boundary	Fill	Notes
Standard	ByLayer, width 1	Hidden	Centroid label: name + area
Property	Dark brown, width 2	Light tan, 20% transparent	Full label, segment distances + bearings
Easement	Red, dashed	ANSI31 hatch, red	Name label
ROW	Green	Lime, 15% transparent	Name label
Empty	Hidden (BoundaryVisible=False)	Hidden	All components hidden

36.9 Tunnel Section Line Styles

Table 20: Built-in Tunnel Section Line Styles

Style	Description
Standard	Measured line with deviation annotations visible, no area hatch.
Hatched	Measured line with coloured area hatch (green = overbreak, red = underbreak) between measured and theoretical profiles.
Simple	Measured line only, no annotations or hatch.

36.10 Civil Object Layer Defaults

When civil object layers are enabled (default: on, prefix C), each civil entity automatically creates dedicated layers for its components. The following table shows all layer name patterns and their default colours. The group, suffix and colour of every component can be changed per drawing in **Civil › Civil Settings… › Layers**.

Table 21: Surface Layers

Component	Layer Pattern	Default Colour
Geometry (TIN)	C-TOPO-{Name}	Grey
Major contours	C-TOPO-{Name}-MAJR	Brown (180, 60, 0)
Minor contours	C-TOPO-{Name}-MINR	Dark green (0, 140, 0)
Boundary	C-TOPO-{Name}-BNDY	Yellow
Breakline	C-TOPO-{Name}-BRKL	Magenta (255, 0, 255)
Points	C-TOPO-{Name}-PNTS	Gold (255, 200, 0)
Labels	C-TOPO-{Name}-LABL	White
Spot elevations	C-TOPO-{Name}-SPOT	Cyan

Table 22: Alignment Layers

Component	Layer Pattern	Default Colour
Centreline	C-ROAD-{Name}	Red
PI markers	C-ROAD-{Name}-PI	Magenta
Labels	C-ROAD-{Name}-LABL	White
Offset axis	C-ROAD-{Name}-OAX	Lilac (210, 170, 255)

Table 23: Profile View Layers

Component	Layer Pattern	Default Colour
Grid	C-PROF-{Name}	Dark grey (60, 60, 60)
Axis	C-PROF-{Name}-AXIS	White
Title	C-PROF-{Name}-TITL	White
Existing profile	C-PROF-{Name}-PROF	Green (0, 160, 0)
Design profile	C-PROF-{Name}-DSGN	Red

Table 24: Parcel Layers

Component	Layer Pattern	Default Colour
Boundary	C-PROP-{Name}	Magenta
Fill	C-PROP-{Name}-FILL	Light pink (255, 200, 255)
Labels	C-PROP-{Name}-LABL	White
Segment labels	C-PROP-{Name}-SEGL	Cyan

Table 25: Survey Point Layers

Component	Layer Pattern	Default Colour
Markers	`C-PNTS`	Gold (255, 200, 0)
Labels	`C-PNTS-LABL`	White

Table 26: Section View Layers

Component	Layer Pattern	Default Colour
Grid	`C-SECT-{Name}`	Dark grey (60, 60, 60)
Axis	`C-SECT-{Name}-AXIS`	White
Section lines	`C-SECT-{Name}-SECT`	Green (0, 160, 0)

Table 27: Point Cloud Section Sheet Layers

Component	Layer Pattern	Default Colour
Sheet geometry (frame, axis guides, trace polylines, labels)	`C-PCSC-{Name}`	Yellow

Every visual element painted by `TAECC_PointCloudSectionSheet` — the cell frames, the optional vertical/horizontal axis guides, the section trace polylines, the cell labels and the optional cell fill — defaults to **ByLayer** in the style entity, so all of them inherit the layer's colour. Override an individual element's colour via the PC Section Sheet Style editor when you need to break out of ByLayer (for example, a contrasting trace colour against a coloured frame).

36.11 Civil Label Styles

All label styles use paper-mm text height, scaled by the civil drawing scale.

Table 28: Built-in Label Styles

Name	Height (mm)	Placement	Components
Point - Standard	2.0	Anchor TR	Number, Elevation
Point - Full	1.5	Anchor TR	Number, E, N, Elevation, Description
Point - Number Only	2.0	Anchor TR	Number
Point - Elevation Only	2.0	Anchor TR	Elevation (%.2f)
Point - No Label	—	—	Hidden
Contour - Standard	2.0	Linear aligned	Elevation (%.1f), uphill readable
Contour - Horizontal	2.0	Linear horizontal	Elevation (%.1f)
Spot Elevation - Standard	1.5	Anchor TR	Elevation (%.2f)
DR - Standard	1.5	Linear perpendicular	Deviation (%.1f cm)
DR - Millimeters	1.5	Linear perpendicular	Deviation (%.0f mm)
Station - Standard	2.0	Linear perpendicular	Station chainage
PI - Standard	1.5	Anchor TR	Station, Radius, Deflection

The **Tunnel** category provides label styles for deviation (DR) labels on tunnel section lines. The two built-in presets are:

- **DR - Standard** — deviation value in centimetres (`%.1f cm`), placed perpendicular to the measured profile line.
- **DR - Millimeters** — deviation value in millimetres (`%.0f mm`), placed perpendicular to the measured profile line.

36.12 Civil Settings

Table 29: Drawing Scale & Units

Setting	Default	Notes
Drawing scale	1:1000	
Linear unit	Metre	Metre / Foot / US Survey Foot
Distance precision	3	
Elevation precision	3	
Coordinate precision	3	
Angle precision	4	
Angle format	Degrees	
Direction format	North azimuth	
Station increment	1000	
Station precision	3	

Table 30: Abbreviations

Abbreviation	Default
Station	STA
Right	RT
Left	LT
North	N
South	S
East	E
West	W
Elevation	EL
Grade Break	GB

Table 31: Civil Draw Order

Setting	Default	Notes
Inter-object priority (back → front)	Surface → Parcel → Alignment → Point	Configurable in Civil Settings > Draw Order

Table 32: Object Layers

Setting	Default	Notes
Enabled	True	
Prefix	C	NCS discipline designator
Use object name	True	Append entity name to layer

Part IX

SDK

Automating and extending IQcad: command scripts, the embedded Python and Pascal scripting environments, the native plugin SDK, and the built-in Blade Runner script editor.

37 Scripting

Scripting lets you replay a sequence of IQcad commands from a text file — useful for repetitive drawing setup, batch operations, demos, and recording sequences of inputs for later reuse.

This chapter documents the `.scr` text-script support and the embedded **Python** scripting host that ship with IQcad today. A third backend — **Pascal** — is still planned and will gain its own section when it lands.

For full programmability (variables, loops, functions, and a real drawing API) jump to *Python scripting* (page 301) below. The `.scr` engine documented first is the simpler input-replay mechanism.

> **Note**
>
> The SCRIPT (`.scr`) engine replays *user inputs* — command names, coordinates, and prompt responses — exactly as if you had typed them at the command line. It has no concept of variables, loops, or conditionals. For those, use the Python backend (*Python scripting* (page 301)).

37.1 Overview

A script is a UTF-8 text file with a `.scr` extension (`.txt` also works) containing IQcad command-line inputs, one operation per logical step. Each script line — or each whitespace-separated token within a line — is fed to the dispatcher in turn, exactly as if you had typed it manually and pressed Enter. This means anything you can do interactively can be scripted, with a few documented exceptions in *Limitations* (page 301) below.

37.2 Running scripts

Three ways to invoke a script:

- **Command** — type `SCRIPT` at the command line with no arguments to open a file picker filtered to `*.scr` / `*.txt`. Or type `SCRIPT <path>` to run a specific file directly. Relative paths resolve against the working directory; quote the path if it contains spaces (`SCRIPT "C:\Project Setup\boilerplate.scr"`).
- **Alias** — `SCR` is the short form of `SCRIPT`.
- **Menu** — **Tools → Run Script…** opens the file picker.

While the script runs, each executed line is echoed to the command line as `[script] <line>` in grey so you can follow what's happening. A successful run ends with `Script complete.` in green.

Stopping a script. Pressing **Esc** cancels both the currently running CAD command and the script. The script also stops automatically on the first unknown command or unhandled exception, reporting the line number:

```
Script stopped at line 9 (dispatch failed).
```

37.3 File format

Plain UTF-8 text:

- Lines beginning with `;` are comments (leading whitespace allowed).
- **Spaces inside a line act like Enter** — the line is split into whitespace-separated tokens and each token is fed to the dispatcher in turn. `LINE 0,0 10,10` is three inputs at once. This matches AutoCAD's script convention.
- **A blank line acts like a right-click** — it finishes the active drawing tool (POLYLINE, LINE chain, SPLINE, etc.). At the empty command prompt with no tool running, a blank line is a no-op.
- A double-quoted string (`"…"`) is one token even if it contains spaces — useful for file paths. The surrounding quotes are stripped.

37.3.1 Coordinate forms

A line of typed coordinates is fed to whatever drawing tool is currently active and treated as a synthetic mouse click at the parsed world point. Five forms are accepted; see *Entering Coordinates* (page 81) in chapter 10 for the full reference. Briefly:

- `X,Y` — absolute 2D, Z inherits from the last point.
- `X,Y,Z` — absolute 3D.
- `@dX,dY[,dZ]` — relative to the last point.
- `@dist<angle` — polar relative, angle in degrees.
- `#N` — survey point number from the active drawing's database.

37.4 Examples by category

The samples below ship under `samples\scripts\` and are intended as copy-and-modify starting points. Each runs end-to-end on a fresh drawing (`01_setup.scr` first if the file mentions named layers).

37.4.1 Setup

Create layers, set the active layer, reset the view. This is the boilerplate every other script in the folder assumes.

File: `samples\scripts\01_setup.scr`

```
MAKELAYER Existing       3
MAKELAYER Design         5
MAKELAYER Annotations    7
```

(continues on next page)

(continued from previous page)

```
MAKELAYER Construction   1
MAKELAYER Centerlines    2
MAKELAYER Hatching       6

CURRENTLAYER Design

PLANVIEW
VIEWTWIST 0
ZE
REGEN
```

Layer colours use the AutoCAD Colour Index (1=red, 2=yellow, 3=green, 4=cyan, 5=blue, 6=magenta, 7=white). Named shortcuts (`RED` / `YELLOW` / …) are also accepted by `MAKELAYER`.

37.4.2 Drawing primitives

LINE / CIRCLE / ARC / RECTANGLE / ELLIPSE / POLYLINE / SPLINE / POINT all use the same point-input convention.

File: `samples\scripts\02_primitives.scr`

```
LINE 0,0 10,0
LINE 0,5 10,5
LINE 0,10 @10,0

CIRCLE 20,0 25,0
CIRCLE 30,0 @5<0

ARC 40,0 45,2.5 50,0

RECTANGLE 0,20 10,30
RECTANGLE 15,20 @10,10

ELLIPSE 30,25 35,25 32.5,28

; Polyline finished by the blank line below.
PLINE 0,40 5,45 10,40 15,45 20,40
```

A few specifics that catch newcomers:

- **CIRCLE** wants two world points (centre + rim point); a bare scalar radius like `CIRCLE 0,0 10` is *not* accepted — use `@10,0` for the rim point.
- **RECTANGLE** wants two opposite corners.
- **ELLIPSE** wants centre + major-axis endpoint + minor-axis endpoint.
- **POLYLINE / SPLINE** chain-mode: each point adds a vertex; a blank line (or another command name) terminates the entity.

37.4.3 Annotation

TEXT and MTEXT take their content via a prompt and then a placement point. Use the prompted form (one line for the command, one for the content, one for the position) so multi-word text isn't confused with multi-token tokenisation:

File: `samples\scripts\03_annotation.scr`

```
TEXT
Hello, IQcad scripting!
0,0

MTEXT
First paragraph of MText.
Second paragraph continues on the next line.
0,5
```

37.4.4 Hatching

HATCH is a pick-points tool: each typed coordinate adds a vertex to the hatch boundary, and the blank line terminates the boundary, at which point the hatch fills the closed region. Pattern and scale come from HP* system variables set via SETVAR.

File: `samples\scripts\04_hatching.scr`

```
SETVAR HPName SOLID
HATCH 0,0 10,0 5,8

SETVAR HPName ANSI31
SETVAR HPScale 1
HATCH 15,0 25,0 20,8
```

37.4.5 Dimensions

Linear, aligned, radius, diameter, and angular dimensions all take points the same way the drawing tools do. Each command lists the specific input order in *Command Reference* (page 257).

File: `samples\scripts\05_dimensions.scr`

```
LINE 0,0 10,0
DIMLINEAR 0,0 10,0 5,-3

CIRCLE 30,5 35,5
DIMRADIUS 30,5 33,7

; Angle between two lines sharing a vertex at (0,20)
LINE 0,20 10,20
LINE 0,20 7,27
DIMANGULAR 0,20 10,20 7,27
```

37.4.6 Civil entities

Civil flows that take input as coordinates (SURVEYPOINT chain) or run single-shot (CREATESURFACE) script cleanly. Flows that need to *click* an existing entity (e.g. CREATEPARCEL picks a closed polyline) are better driven from the GUI for now.

File: `samples\scripts\06_civil.scr`

```
SURVEYPOINT 0,0,100.0
0,10,100.5
0,20,101.0
10,0,99.8
10,10,100.2
```

(continues on next page)

(continued from previous page)

```
; Build a TIN surface from the survey points just placed
CREATESURFACE
```

37.4.7 Guided tour

A short walkthrough combining several entity types — useful as a quick starting template.

File: `samples\scripts\07_tour.scr`

This sample creates layers, draws a building footprint as a closed polyline, dimensions it, hatches the interior, and adds a label — everything in one self-contained script.

37.5 Patterns

- **Setup boilerplate.** Start every script with a call to `01_setup` or its equivalent so layer names and view defaults are predictable.
- **One operation, one logical block.** Group the lines for each drawing operation together with a comment line above; keep blank lines reserved for "finish current tool" right-clicks.
- **Mix manual and scripted input.** A script can leave you with an active tool — handy for "set things up, then take it from here".
- **Reuse via copy-paste.** Scripts are linear; for now, the way to reuse logic is to copy it. Variables and loops will arrive with the Python / Pascal backends.

37.6 Limitations

- **Scalar radius / distance shortcuts.** AutoCAD's `CIRCLE 0,0 10` (bare scalar radius) is *not* supported — IQcad expects two world points. Use a rim point at the desired distance instead.
- **Modal dialogs** opened by a scripted command (e.g. `LAYERS`, `MAPSETTINGS`) block until you close them manually. The script pauses, then continues. Avoid such commands in unattended scripts.
- **Pick-an-existing-entity flows.** Commands that need to *click* on an entity already in the drawing (CREATEPARCEL, ALIGNFROMPOLYLINE, etc.) work only if the script's synthetic click happens to fall on the right entity's nearest vertex. More reliable to drive from the GUI.

37.7 Python scripting

IQcad now embeds a complete **Python 3.12** interpreter — a private, bundled CPython that ships in the `python\` folder next to `IQcad.exe`. There is nothing to install and no system Python is required or used.

Python is the recommended choice for anything beyond a linear replay of typed inputs. Unlike `.scr`, Python gives you variables, loops, functions, conditionals, the full Python standard library, and a proper drawing API through the built-in `iqcad` module.

37.7.1 Running Python

```
PYTHON drawing_setup.py     ; run a .py file (alias: PY)
PYEVAL 2 + 2                ; evaluate a single expression, print result
PYLOAD extensions.py        ; load a file and keep it resident
PYSCRIPTS                   ; list registered Python commands
PYPIP install requests      ; run pip in the bundled interpreter
```

- **PYTHON** *file* (alias **PY**) runs a `.py` file once.
- **PYEVAL** *expr* evaluates a single Python expression and prints the result at the command line — handy as a calculator or for quick API probing (`PYEVAL iqcad.active_drawing.name`).
- **PYLOAD** *file* runs a file and leaves it **session-resident**: any custom commands or event handlers it registers stay active until you close IQcad. This is how you load an extension by hand.
- **PYSCRIPTS** lists the Python-defined commands currently registered.
- **PYPIP** *args* forwards its arguments to `pip` running inside the bundled interpreter, so you can add third-party packages.

Note

`PYPIP install <pkg>` downloads from PyPI and therefore needs network access. Behind a TLS-intercepting proxy or antivirus, the SSL handshake may fail. In that case install from an offline wheel (`PYPIP install C:\wheels\somepkg.whl`) or pass pip's proxy / trust options (`PYPIP install --proxy http://host:port pkg` or `--trusted-host pypi.org --trusted-host files.pythonhosted.org`).

37.7.2 The `iqcad` module

Every script runs with a built-in `iqcad` module already imported. It exposes an object-oriented drawing API rooted at `iqcad.active_drawing` (a `Drawing`), plus a set of flat convenience helpers.

```
import iqcad

dwg = iqcad.active_drawing          # the current Drawing

dwg.create_layer("Design", color=5)
dwg.active_layer = "Design"

h = dwg.add_line((0, 0), (10, 0))   # returns an Entity
c = dwg.add_circle((5, 0), 2.5)

h.color = 1                         # AutoCAD colour index (1 = red)
h.layer = "Design"
print(h.handle, h.type_name, h.points)
print(h.get_property("length"))

dwg.zoom_extents()
dwg.invalidate()
```

`Drawing` offers `add_line`, `add_circle`, `add_arc`, `add_ellipse`, `add_rectangle`, `add_polyline`, `add_text`, `add_hatch`, `add_survey_point`, `add_tunnel_section_line`, `create_layer`, `find_by_handle`, `entities`, `selection`, `zoom_extents`, `invalidate`, and the properties `active_layer`, `name`, `layer_names`.

Creatable entities: line, circle, arc, ellipse, rectangle, polyline, text, hatch, survey point, tunnel-section-line.

Each `Entity` exposes the properties `handle`, `type_name`, `layer`, `color`, `points` and the methods `delete`, `get_property`, `set_property`.

The same operations are also available as **flat helpers** for quick one-liners — `iqcad.cmd`, `iqcad.write_line`, `iqcad.point`, `iqcad.add_line` / `add_circle` / `add_arc` / `add_ellipse` / `add_rectangle` / `add_text` / `add_hatch` / `add_tunnel_section_line` (each returns a handle string), `iqcad.create_layer`, `iqcad.set_active_layer`, `iqcad.zoom_extents`:

```
import iqcad
iqcad.create_layer("Grid", color=8)
iqcad.set_active_layer("Grid")
for x in range(0, 101, 10):
    iqcad.add_line((x, 0), (x, 100))
iqcad.zoom_extents()
```

37.7.3 Surfaces

`Drawing` exposes three surface methods: `create_surface` builds a new TIN from a list of 3-D seed points, `find_surface` looks up an existing surface by name (returns `None` if not found), and `surfaces` enumerates all surfaces in the drawing.

Each `Surface` has read-only `name` and `handle` properties and four methods:

- `elevation_at(x, y)` — interpolates the TIN at the given plan coordinates; returns `None` when the point lies outside the triangulation.
- `add_point(x, y, z)` — inserts a new vertex and rebuilds the TIN.
- `profile_along(pts)` — samples the surface along a polyline defined by 2-D plan points; returns a list of `(station, z)` tuples.
- `volume_to(other, cell=1.0)` — computes cut and fill volumes between this surface and *other* using a grid cell size of *cell* metres; returns `(cut, fill)` as a 2-tuple of floats.

```
import iqcad

d = iqcad.active_drawing

# Build an existing-ground surface from survey points
eg = d.create_surface("EG", [(0, 0, 10), (10, 0, 11), (5, 8, 10.5)])

z = eg.elevation_at(4, 2)                  # float, or None if outside the␣
↪TIN
prof = eg.profile_along([(0, 0), (50, 0)]) # [(station, z), ...]

# Compare against a finished-grade surface already in the drawing
fg = d.find_surface("FG")
if fg is not None:
    cut, fill = eg.volume_to(fg)
    iqcad.write_line(f"cut={cut:.1f}  fill={fill:.1f}")
```

37.7.4 Alignments

`Drawing` exposes three alignment methods: `create_alignment_from_polyline` builds a new alignment from an existing polyline entity (located by handle), `find_alignment` looks up an alignment by name (returns `None` if not found), and `alignments` enumerates every alignment in the drawing.

Each `Alignment` has read-only `name`, `handle`, `length`, `start_station`, and `end_station` properties and five methods:

- `point_at_station(s)` — returns the `(x, y, z)` world point at chainage *s*.
- `direction_at_station(s)` — bearing at chainage *s*, in radians.
- `elevation_at_station(s)` — profile elevation at chainage *s* (from the attached vertical alignment, if any).
- `station_offset_at(x, y)` — nearest chainage and signed offset for a plan point; offset is negative to the left and positive to the right of the alignment direction.
- `elements()` — list of dicts describing the horizontal geometry elements; each dict has keys `kind` (`"line"` or `"arc"`), `length`, `start_station`, and `radius` (`0` for tangent runs).

```
import iqcad

d = iqcad.active_drawing

# Create an alignment from a polyline already in the drawing.
# radius=0 builds straight tangents through every vertex; a non-zero
# radius fillets the interior corners with curves instead.
al = d.create_alignment_from_polyline(poly_handle, radius=25.0, name="CL-1
↪")

s, off = al.station_offset_at(x, y)        # chainage + offset (-left /␣
↪+right)
x, y, z = al.point_at_station(120.0)
z = al.elevation_at_station(120.0)         # from the profile

for el in al.elements():
    print(el["kind"], el["length"])

# Iterate all alignments in the drawing
for a in d.alignments():
    print(a.name, f"{a.length:.1f} m")
```

Note

`create_alignment_from_polyline` with `radius=0` (the default) builds straight tangents through every vertex, including polylines with **3 or more vertices**. Pass a non-zero *radius* (≥ 0.01) only if you want the interior corners filleted into curves.

Full signature:

```
create_alignment_from_polyline(handle, radius=0, name="", use_z=False,␣
↪vcurve_len=0)
```

`use_z=True` builds the vertical profile from the polyline's per-vertex elevations — the scripted equivalent of accepting the `AFP` command's `Create vertical profile from ver-`

`tex elevations?` prompt. `vcurve_len` sets the vertical curve length applied to every interior PVI (`0`, the default, keeps an angular profile). The PascalScript equivalent is `CreateAlignmentFromPolylineZ(polyHandle, radius, name, vcurveLen)` (see *Pascal Scripting* (page 323)), which always builds the profile from vertex elevations.

37.7.5 Custom commands

Decorate a function with `@iqcad.command` to register a new IQcad command. Once registered (load it with `PYLOAD` or drop it in the autoload folder) it appears in `HELP` and can be typed at the command prompt like any built-in. **Command arguments arrive as strings** — cast them yourself.

```
import iqcad

@iqcad.command("GRID", aliases="GR", category="Drafting",
               description="Draw an N x N construction grid")
def grid(n, spacing):
    n = int(n)                  # args are strings — cast them
    step = float(spacing)
    for i in range(n + 1):
        iqcad.add_line((0, i * step), (n * step, i * step))
        iqcad.add_line((i * step, 0), (i * step, n * step))
    iqcad.zoom_extents()
```

Then at the prompt: `GRID 5 10`.

37.7.6 Events

Register handlers that fire when the drawing changes. Handlers run **synchronously** on the main thread; an exception in a handler is logged to the command line, not fatal.

```
import iqcad

@iqcad.on_entity_created
def created(entity):
    iqcad.write_line(f"created {entity.type_name} {entity.handle}")

@iqcad.on_entity_deleted
def deleted(entity):
    iqcad.write_line(f"deleted {entity.handle}")

@iqcad.on_selection_changed
def selected(entities):
    iqcad.write_line(f"selection: {len(entities)} object(s)")
```

Handlers can also be attached and detached explicitly with `iqcad.on(event, func)` and `iqcad.off(event, func)`.

37.7.7 Long-running scripts

A tight Python loop blocks IQcad's UI for its whole duration. In any loop that may run for a while, call `iqcad.yield_()` periodically: it pumps the message queue so the window stays responsive, and lets **Esc** cancel the script (which raises `KeyboardInterrupt` in your code).

```
import iqcad
```

(continues on next page)

(continued from previous page)

```
try:
    for i in range(100000):
        iqcad.add_circle((i, 0), 1)
        if i % 100 == 0:
            iqcad.yield_()      # note the trailing underscore
except KeyboardInterrupt:
    iqcad.write_line("cancelled by user")
```

Note

The method is spelled `iqcad.yield_()` with a trailing underscore, because `yield` is a reserved keyword in Python.

37.7.8 Installed extensions (autoload) and trust

Any `.py` file dropped in `%APPDATA%\IQcad\scripts\` is loaded automatically at startup — the place to keep your personal commands and event handlers.

Because autoloaded code runs without prompting, IQcad shows a one-time **content-trust dialog** the first time it sees a script (and again whenever its contents change). The dialog shows the file's SHA-256 hash and offers **Always**, **This session**, or **Never**; your choice is remembered in `%APPDATA%\IQcad\scripts\script_trust.json`. A script you have never trusted is not executed.

37.7.9 Type stubs and IDE support

For autocomplete and type checking in your editor, point it at the type stub `samples\python\iqcad.pyi` (or at the bundled interpreter in `python\`). Worked examples ship alongside it:

- `samples\python\01_primitives.py` — drawing geometry with the API.
- `samples\python\02_command.py` — registering a custom command.
- `samples\python\03_events.py` — wiring up event handlers.

37.7.10 Known limitations (Python)

- **Command arguments are strings.** `@iqcad.command` passes every argument as a `str`; cast to `int` / `float` in your function.
- **Limited event coverage.** Only entity-created, entity-deleted, and selection-changed events exist today. There is no `on_entity_modified` and no drawing-lifecycle or command events yet.
- **No background / async execution.** Scripts run on the UI thread; there is no `iqcad.background()` worker API yet. Use `iqcad.yield_()` to keep long loops responsive.

Note

The Greek edition of this manual does not yet include the Python scripting section; its translation is a follow-up.

37.8 Future scripting languages

The SCR engine is intentionally minimal — it's a faithful replay of typed inputs. For richer scripting:

- **Python** — *available now.* A Python 3.12 interpreter embedded in IQcad with bindings to the drawing API, command dispatcher, and entity model, supporting full control flow, computations, user-defined functions, and external libraries. See *Python scripting* (page 301) above.
- **Pascal** — *planned.* A Free Pascal script host using the same compiler that builds IQcad, for users who already write extensions in Pascal and want the same syntax for scripting. When it lands, this chapter will gain a *Pascal scripting* section.

38

Python Scripting Reference

This chapter is the complete reference for IQcad's embedded `iqcad` Python module: every class, method, property, decorator, and event, with text-based class diagrams and worked examples.

For a getting-started introduction (running scripts, the trust model, custom commands, the big picture) see *Python scripting* (page 301) in chapter 18. This chapter is the API reference you reach for once you know the basics.

38.1 Intro

IQcad embeds a private, bundled **Python 3.12** interpreter that ships in the `python\` folder next to `IQcad.exe`. There is nothing to install and no system Python is required or used. Every script runs with a built-in `iqcad` module already imported, exposing an object-oriented drawing API rooted at `iqcad.active_drawing` together with a set of flat convenience helpers.

The reference below is generated from the type stub `samples\python\iqcad.pyi` and matches it exactly. Point your IDE at that stub (or the bundled interpreter) for autocomplete.

38.2 Running Python

Five command-line entry points drive the Python host:

Command	Description
`PYTHON <file.py>` (alias `PY`)	Run a `.py` file once. Anything it registers is discarded when it finishes.
`PYEVAL <expr>`	Evaluate a single Python expression and print the result at the command line (e.g. `PYEVAL iqcad.active_drawing.name`).
`PYLOAD <file.py>`	Run a file and leave it **session-resident**: any `@iqcad.command` commands or event handlers it registers stay active until you close IQcad.
`PYSCRIPTS`	List the Python-defined commands currently registered.
`PYPIP <args>`	Forward arguments to `pip` running inside the bundled interpreter.

Warning

`PYPIP install <pkg>` downloads from PyPI and therefore needs network access. Behind a TLS-intercepting proxy or antivirus the SSL handshake may fail. Install from an offline wheel (`PYPIP install C:\wheels\somepkg.whl`) or pass pip's proxy / trust options (`PYPIP install --proxy http://host:port pkg` or `--trusted-host pypi.org --trusted-host files.pythonhosted.org`).

38.2.1 Autoload and trust

Any `.py` file dropped in `%APPDATA%\IQcad\scripts\` is loaded automatically at startup. Because autoloaded code runs without prompting, IQcad shows a one-time **content-trust dialog** the first time it sees a script (and again whenever its SHA-256 content hash changes). The dialog offers **Always**, **This session**, or **Never**; your choice is remembered in `%APPDATA%\IQcad\scripts\script_trust.json`. A script you have never trusted is not executed.

38.2.2 Responsiveness and Esc

Scripts run on the GUI thread, so a tight loop blocks the UI for its whole duration. In any loop that may run for a while, call `iqcad.yield_()` periodically: it pumps the message queue so the window stays responsive, and lets **Esc** cancel the script (raising a catchable `KeyboardInterrupt`). The method is spelled with a trailing underscore because `yield` is a Python keyword.

38.2.3 Type stubs

For autocomplete and type checking, point your editor at the stub `samples\python\iqcad.pyi` or at the bundled interpreter in `python\`.

38.3 The object model

The `iqcad` module is the root. It carries flat helper functions, the command/event decorators, and the live `active_drawing` attribute. `active_drawing` resolves fresh on every access (PEP 562) and returns the current `Drawing`. The `Drawing` is the factory for everything else: it *creates* and *returns* `Entity`, `Surface`, and `Alignment` objects.

```
iqcad  (module)
|
|-- flat helpers ....... cmd, write_line, point,
|                        add_line/add_circle/add_arc/add_ellipse/
|                        add_rectangle/add_hatch/
|                        add_tunnel_section_line  (return handle str),
|                        create_layer, set_active_layer, zoom_extents
|-- decorators ......... command, on_entity_created,
|                        on_entity_deleted, on_selection_changed,
|                        on / off, unregister_command
|-- yield_() ........... pump UI / allow Esc
|
`-- active_drawing  --->  Drawing   (resolves fresh each access)
                           |
                           |  create/returns
                           |
          +---------------+----------------+
          |               |                |
          v               v                v
```

(continues on next page)

(continued from previous page)

```
        Entity          Surface          Alignment
        (geometry)      (TIN)            (horizontal/vertical)

        add_line/circle/arc/ellipse/rectangle/polyline/
        text/hatch/survey_point/tunnel_section_line  --> Entity
        create_surface / find_surface / surfaces       --> Surface
        create_alignment_from_polyline /
        find_alignment / alignments                    --> Alignment
```

Note

`iqcad.add_line(...)` (flat) returns a handle **string**, whereas `iqcad.active_drawing.add_line(...)` (OO) returns an `Entity`. `add_polyline` is OO-only: there is no flat `iqcad.add_polyline`.

38.4 Class reference

38.4.1 Drawing

Obtained from `iqcad.active_drawing`. The factory for entities, surfaces, and alignments.

Member	Signature	Returns	Description
`active_layer`	property (read/write)	`str`	Name of the current layer.
`name`	property (read-only)	`str`	Drawing name.
`layer_names`	property (read-only)	`List[str]`	All layer names.
`add_line`	`add_line(a, b)`	`Entity`	Line from point *a* to *b*.
`add_circle`	`add_circle(c, r)`	`Entity`	Circle, centre *c*, radius *r*.
`add_text`	`add_text(pos, text, height)`	`Entity`	Text at *pos* with *height*.
`add_polyline`	`add_polyline(pts, closed=False)`	`Entity`	Polyline through *pts* (OO-only).
`add_survey_point`	`add_survey_point(p number)`	`Entity`	Survey point with *number*.
`add_arc`	`add_arc(start, mid, end)`	`Entity`	Three-point arc.
`add_ellipse`	`add_ellipse(center radx, rady, rotation=0.0)`	`Entity`	Ellipse with semi-axes.
`add_rectangle`	`add_rectangle(c1, c2)`	`Entity`	Rectangle from opposite corners.
`add_hatch`	`add_hatch(points, pattern="ANSI31", angle=0.0, scale=1.0)`	`Entity`	Hatch filling the *points* boundary.
`add_tunnel_secti`	`add_tunnel_section closed=False)`	`Entity`	Tunnel section line.
`create_layer`	`create_layer(name, color=...)`	`bool`	Create a layer; *color* is an ACI index.
`find_by_handle`	`find_by_handle(h)`	`Optional[En`	Look up an entity by handle.
`entities`	`entities()`	`List[Enti`	All entities in the drawing.
`selection`	`selection()`	`List[Enti`	Currently selected entities.
`zoom_extents`	`zoom_extents()`	`None`	Zoom to drawing extents.
`invalidate`	`invalidate()`	`None`	Force a redraw.
`surfaces`	`surfaces()`	`List[Surf`	All surfaces.
`find_surface`	`find_surface(name)`	`Optional[Su`	Look up a surface by name.
`create_surface`	`create_surface(nam pts)`	`Surface`	Build a TIN from `(x, y, z)` points.
`alignments`	`alignments()`	`List[Alig`	All alignments.
`find_alignment`	`find_alignment(nam`	`Optional[Al`	Look up an alignment by name.
`create_alignment_fr`	`create_alignment_f radius=0.0, name="")`	`Alignment`	Build an alignment from a polyline entity.

38.4.2 Entity

A single drawing object returned by the OO `add_*` methods.

Member	Signature	Returns	Description
`handle`	property (read-only)	`str`	Persistent handle.
`type_name`	property (read-only)	`str`	`'Line'` | `'Circle'` | `'Polyline'` | `'Text'` | `'SurveyPoint'` | …
`layer`	property (read/write)	`str`	Layer name.
`color`	property (read/write)	`int`	ACI colour index.
`points`	property (read/write)	`List[Tuple[float, float]]`	Defining points.
`delete`	`delete()`	`None`	Remove the entity. Any later access raises `RuntimeError`.
`get_property`	`get_property(name)`	`bool | int | float | str | None`	Read a typed property (e.g. circle `'Radius'`).
`set_property`	`set_property(name, value)`	`None`	Write a typed property.

Common `get_property` / `set_property` names: circle `'Radius'`; ellipse `'RadiusX'` / `'RadiusY'`; text `'Text'` / `'Height'`; survey point `'PointNumber'` / `'Elevation'`.

38.4.3 Surface

A TIN surface created by `Drawing.create_surface` (or looked up via `find_surface` / `surfaces`).

Member	Signature	Returns	Description
`name`	property (read-only)	`str`	Surface name.
`handle`	property (read-only)	`str`	Persistent handle.
`elevation_at`	`elevation_at(x, y)`	`Optional[float]`	Interpolated Z, or `None` outside the TIN.
`add_point`	`add_point(x, y, z)`	`None`	Insert a vertex and rebuild the TIN.
`profile_along`	`profile_along(pts)`	`List[Tuple[float, float]]`	Long-section: list of `(station, z)`.
`volume_to`	`volume_to(other, cell=1.0)`	`Tuple[float, float]`	Grid-approx `(cut, fill)`; cut where self is above *other*.

38.4.4 Alignment

A horizontal alignment created by `create_alignment_from_polyline` (or looked up via `find_alignment` / `alignments`).

Member	Signature	Returns	Description
`name`	property (read-only)	`str`	Alignment name.
`handle`	property (read-only)	`str`	Persistent handle.
`length`	property (read-only)	`float`	Total length.
`start_station`	property (read-only)	`float`	Start chainage.
`end_station`	property (read-only)	`float`	End chainage.
`point_at_stati`	`point_at_station`	`Tuple[float, float, float]`	`(x, y, z)` at chainage *s*.
`direc-tion_at_statio`	`direc-tion_at_station(`	`float`	Bearing at *s*, in radians.
`eleva-tion_at_statio`	`eleva-tion_at_station(`	`float`	Profile elevation at *s*.
`sta-tion_offset_at`	`station_offset_a y)`	`Tuple[float, float]`	`(station, offset)`; offset - left / + right.
`elements`	`elements()`	`List[Dict[str, object]]`	Geometry elements; keys `kind` / `length` / `start_station` / `radius`.

38.4.5 Module-level functions

The flat helpers and the live `active_drawing` attribute.

Member	Signature	Returns	Description
active_drawing	attribute (read-only)	Drawing	The current drawing; resolves fresh on each access.
cmd	cmd(s)	None	Run a command-line string.
write_line	write_line(s)	None	Print a line to the command line.
point	point(x, y, z=...)	Tuple[fl float, float]	Build a point tuple.
add_line	add_line(a, b)	str	Flat line; returns a handle string.
add_circle	add_circle(c, r)	str	Flat circle; returns a handle string.
add_text	add_text(pos, text, height)	str	Flat text; returns a handle string.
add_arc	add_arc(start, mid, end)	str	Flat arc; returns a handle string.
add_ellipse	add_ellipse(center, radx, rady, rotation=0.0)	str	Flat ellipse; returns a handle string.
add_rectangle	add_rectangle(c1, c2)	str	Flat rectangle; returns a handle string.
add_hatch	add_hatch(points, pattern="ANSI31", angle=0.0, scale=1.0)	str	Flat hatch; returns a handle string.
add_tunnel_secti	add_tunnel_section_l closed=False)	str	Flat tunnel section line; returns a handle string.
create_layer	create_layer(name, color=...)	bool	Create a layer (ACI *color*).
set_active_layer	set_active_layer(nar	None	Set the current layer.
zoom_extents	zoom_extents()	None	Zoom to drawing extents.
unregis-ter_command	unregis-ter_command(name)	None	Remove a registered custom command.
on	on(event, func)	Callable	Attach an event handler by name.
off	off(event, func)	None	Detach an event handler.
yield_	yield_()	None	Pump the UI; raises KeyboardInterrupt on Esc.
show_message	show_message(text)	None	Show a message box.
confirm	confirm(prompt)	bool	Yes/No prompt; True if confirmed.
input_string	input_string(prompt, default="")	Op-tional[s	Text input; None if cancelled.
select_file	select_file(title="' filter="", save=False)	str	File chooser; '' if cancelled.

38.5 Decorators and events

38.5.1 Custom commands

Decorate a function with `@iqcad.command` to register a new IQcad command. Load it with `PYLOAD` (or drop it in the autoload folder) and it can be typed at the command prompt like any built-in.

```
@iqcad.command("NAME", aliases="N", category="Tools",
               description="What it does")
def handler(*args):
    ...
```

Warning

Command arguments arrive as **strings**. Cast them yourself (`int(args[0])`, `float(args[1])`).

Remove a registration with `iqcad.unregister_command("NAME")`.

38.5.2 Events

Three event decorators register handlers that fire when the drawing changes:

Decorator	Handler receives	Fires when
`@iqcad.on_entity_created`	`fn(entity)` – an `Entity`	An entity is created.
`@iqcad.on_entity_deleted`	`fn(handle)` – a handle `str`	An entity is deleted.
`@iqcad.on_selection_changed`	`fn(drawing)` – the `Drawing`	The selection changes.

Handlers can also be attached and detached explicitly with `iqcad.on(event, func)` and `iqcad.off(event, func)`.

Note

Handlers run **synchronously** on the GUI thread. An exception inside a handler is logged to the command line, not fatal. There is no `on_entity_modified` and no drawing-lifecycle or command events yet.

38.6 Dialogs

Four module-level dialog functions prompt the user. They are shared with the Pascal host (see *Pascal Scripting* (page 323)). Run headlessly – or wherever no GUI provider is installed – they return safe defaults so an unattended script never blocks: `confirm` returns `False`, `input_string` returns `None`, and `select_file` returns `''`.

Function	Signature	Returns	Description
`show_message`	`show_message(text)`	`None`	Show a message box with *text*.
`confirm`	`confirm(prompt)`	`bool`	Yes/No prompt; `True` if the user confirmed.
`input_string`	`input_string(prompt, default="")`	`Optional[st`	Single-line text input; `None` if cancelled, else the string.
`select_file`	`select_file(title="" filter="", save=False)`	`str`	File chooser; `''` if cancelled. *filter* is a pipe-delimited `Caption|Mask` (e.g. `"CSV files|*.csv"`); *save* selects an Open vs Save dialog.

```
import iqcad

iqcad.show_message("Starting import.")
if iqcad.confirm("Proceed?"):
    name = iqcad.input_string("Layer name", "Design")
    if name is not None:
        path = iqcad.select_file("Open points file", "CSV files|*.csv",␣
↪False)
        if path:
            iqcad.write_line("import %s onto %s" % (path, name))
```

38.7 Examples

Worked example scripts ship under `samples\python\`:

- `samples\python\01_primitives.py` – drawing geometry with the API.
- `samples\python\02_command.py` – registering a custom command.
- `samples\python\03_events.py` – wiring up event handlers.
- `samples\python\04_surface.py` – TIN surfaces: create, query, profile, volume.
- `samples\python\05_alignment.py` – alignments from a polyline.
- `samples\python\06_civil_workflow.py` – chainage along an alignment while sampling a surface (the highlight).

38.7.1 Surfaces

Create a TIN from seed points, interpolate it, take a long-section, then compare two surfaces by volume (`04_surface.py`):

```
import iqcad

d = iqcad.active_drawing

eg = d.create_surface("EG", [
    (0, 0, 10.0), (50, 0, 11.0), (50, 40, 12.5),
    (0, 40, 10.5), (25, 20, 11.2),
```

(continues on next page)

(continued from previous page)

```
])

z = eg.elevation_at(25, 20)            # float, or None outside the TIN
eg.add_point(25, 40, 11.8)             # rebuilds the TIN

for station, pz in eg.profile_along([(0, 20), (50, 20)]):
    iqcad.write_line("STA %.3f  Z %.3f" % (station, pz))

fg = d.create_surface("FG", [
    (0, 0, 11.0), (50, 0, 12.0), (50, 40, 13.5),
    (0, 40, 11.5), (25, 20, 12.2),
])
cut, fill = eg.volume_to(fg, cell=1.0) # cut where EG is above FG
iqcad.write_line("cut=%.2f  fill=%.2f" % (cut, fill))
```

38.7.2 Alignments

Build an alignment from a polyline (fillet radius required for 3+ vertices), then query it (`05_alignment.py`):

```
import iqcad

d = iqcad.active_drawing

# add_polyline is OO-only; capture the handle for the alignment factory.
ph = d.add_polyline([(0, 0), (100, 0), (100, 50), (160, 50)], False).handle
al = d.create_alignment_from_polyline(ph, 20.0, "CL-1")

iqcad.write_line("len=%.3f  STA %.3f..%.3f" % (
    al.length, al.start_station, al.end_station))

x, y, z = al.point_at_station(50.0)
d_rad = al.direction_at_station(50.0)
s, off = al.station_offset_at(50.0, 5.0)   # offset -left / +right

for el in al.elements():
    iqcad.write_line("%s len=%.3f R=%.3f" % (
        el["kind"], el["length"], el["radius"]))
```

38.7.3 Combined civil workflow

The highlight: step chainage along an alignment and sample the ground surface at each station (`06_civil_workflow.py`):

```
import iqcad

d = iqcad.active_drawing

surf = d.create_surface("EG", [
    (-20, -20, 10.0), (180, -20, 14.0), (180, 80, 16.0),
    (-20, 80, 11.0), (80, 25, 13.0),
])

ph = d.add_polyline([(0, 0), (100, 0), (100, 50), (160, 50)], False).handle
al = d.create_alignment_from_polyline(ph, 20.0, "CL-1")
```

(continues on next page)

(continued from previous page)

```
step = 10.0
s = al.start_station
while s <= al.end_station + 1e-6:
    x, y, z = al.point_at_station(s)
    gz = surf.elevation_at(x, y)           # None outside the TIN
    iqcad.write_line("STA %.1f  surf=%.3f" % (
        s, gz if gz is not None else float('nan')))
    s += step
```

38.8 Dialogs and message boxes

Scripts can pop a parameter dialog or a message box through the same host-built UI the plugin SDK uses (so there is no Tk/Qt dependency).

`iqcad.create_dialog(title)` returns a dialog object; describe fields, `execute()`, then read the values:

```
dlg = iqcad.create_dialog("Draw a shape")
dlg.add_choice("shape", "Shape", ["Rectangle", "Circle"], 0)
dlg.add_float("size",  "Size (m)", 10.0, decimals=2)   # min=0, max=0 →␣
↪unbounded
dlg.add_text ("layer", "Layer name", "SHAPES")
dlg.add_color("color", "Color", 0x00B400)              # 0x00BBGGRR
dlg.add_bool ("filled","Filled?", False)
if dlg.execute():
    shape  = dlg.get_choice("shape")   # selected index
    size   = dlg.get_float("size")
    layer  = dlg.get_text("layer")
    color  = dlg.get_color("color")
    filled = dlg.get_bool("filled")
```

Field builders: `add_text`, `add_multiline`, `add_int(…, min=0, max=0)`, `add_float(…, min=0.0, max=0.0, decimals=3)`, `add_bool`, `add_choice(key, label, options_list, default_index=0)`, `add_file(…, filter='', save=False)`, `add_folder`, `add_color`. Getters: `get_text` (also returns the selected choice's text), `get_int`, `get_float`, `get_bool`, `get_choice` (index), `get_color`. `execute()` returns `False` if the user cancels.

`iqcad.message_box(text, caption="", buttons=iqcad.MB_OK, icon=iqcad.MB_ICON_NONE)` shows a message box and returns the pressed button. Buttons: `MB_OK`, `MB_OKCANCEL`, `MB_YESNO`, `MB_YESNOCANCEL`; icons: `MB_ICON_NONE/INFO/WARNING/ERROR/QUESTION`; results: `MB_RESULT_OK/CANCEL/YES/NO` (a dismissed box returns `MB_RESULT_CANCEL`).

```
if iqcad.message_box("Delete selection?", "Confirm",
                     iqcad.MB_YESNO, iqcad.MB_ICON_QUESTION) == iqcad.MB_
↪RESULT_YES:
    pass   # … proceed with the delete
```

38.9 Live forms

Where a dialog is modal, a **live form** is a non-modal, host-owned tool window that stays open: you build controls, keep the handles, mutate them at runtime, and react to the user through a handler object. It is the scripting face of the plugin SDK's *live tool windows* (page 351).

`iqcad.create_form(title, handler=None)` returns a live form. Build it with `add_button`, `add_label`, `add_text`, `add_multiline`, `add_int`, `add_float`, `add_bool`, `add_choice`, `add_color`, `add_file`, `add_folder`, `add_list` (plus `begin_row` / `end_row`, `find_control`), then `show()`. Each `add_*` returns a control object with `get_*` / `set_*` accessors, `set_caption` / `set_enabled` / `set_visible`, and the list ops `append_line` / `clear` / `item_count` / `get_selected` / `set_selected`. The form also offers `hide()`, `close()`, `is_visible()` and `set_title()`.

The optional `handler` is any object whose `on_button_clicked(key)`, `on_value_changed(key)` and `on_form_closed()` methods the host calls (on the main thread) when the user interacts with the form:

```
class Handler:
    def __init__(self, form):
        self.form = form
    def on_button_clicked(self, key):
        if key == "add":
            n = self.form.find_control("n").get_int()
            self.form.find_control("log").append_line("Added: %d" % n)
            self.form.find_control("status").set_text("Last added: %d" % n)

form = iqcad.create_form("Form demo")
form.add_label("status", "Click Add to append a value")
form.add_int("n", "Value", 1, 0, 1000)
form.add_button("add", "Add")
form.add_list("log", "Log")
form.handler = Handler(form)        # or pass handler= to create_form
form.show()
```

`iqcad.create_panel(title, handler=None)` is identical to `create_form` but **docks** the panel into the main window — it appears as a tab in the right-side dock strip instead of a floating window. The builder, control objects and handler surface are exactly the same; only the container differs:

```
panel = iqcad.create_panel("Form demo", handler)   # docked twin of create_
↪form
```

38.10 Ribbon & menu

`iqcad.add_ribbon_button(group, command, caption, hint='', icon='', large=True)` and `iqcad.add_menu_item(submenu, command, caption)` add GUI entries that **run a registered command** — a button in the **Plugins** ribbon tab and an item in the **Plugins** menu. Scripts appear under a single **Scripts** group/submenu; `icon` is an **absolute** path to a PNG (omit it for a generic icon). See *Ribbon & menu contributions* (page 354).

```
@iqcad.command("GREET")
def greet():
    iqcad.message_box("Hello!", "Greet", iqcad.MB_OK, iqcad.MB_ICON_INFO)
```

(continues on next page)

(continued from previous page)

```
iqcad.add_ribbon_button("Demo", "GREET", "Greet", hint="Say hello",␣
↪large=True)
iqcad.add_menu_item("Demo", "GREET", "Greet")
```

38.11 Limitations and tips

- **Command arguments are strings.** `@iqcad.command` passes every argument as a `str`; cast to `int` / `float` in your function.
- **``elevation_at`` returns None outside the TIN.** Always guard the result before using it in arithmetic.
- **Alignment fillet radius.** `create_alignment_from_polyline` with `radius=0` (the default) builds straight tangents through every vertex, including polylines with **3 or more vertices**. Pass a non-zero radius (at least `0.01`) only if you want the interior corners filleted into arcs.
- **Offset sign.** `station_offset_at` returns an offset that is **negative to the left** and **positive to the right** of the alignment direction.
- **Limited event coverage.** Only entity-created, entity-deleted, and selection-changed events exist today. There is no `on_entity_modified` and no drawing-lifecycle or command events yet.
- **No background / async execution.** Scripts run on the GUI thread; use `iqcad.yield_()` to keep long loops responsive and Esc-cancelable.

39 Pascal Scripting

This chapter is the complete reference for IQcad's embedded **Pascal Script** host: every script-facing class, global, command, and event, with a text class diagram and worked examples.

For the big-picture scripting concepts (the trust model, autoload, custom commands, events) see *Python scripting* (page 301) in chapter 18 and the *Python Scripting Reference* (page 309) in chapter 19. The Pascal host shares the same Layer-2 drawing model as Python, so those concepts carry over directly; this chapter documents the Pascal-specific surface.

39.1 Intro

Pascal Script is a Pascal-syntax interpreter that compiles your script **into IQcad** at run time. Unlike the Python host – which embeds a bundled CPython interpreter – the Pascal host has **zero external runtime dependency**: there is nothing to install and nothing ships alongside `IQcad.exe`. If you already know Object Pascal (or Delphi), this is the lowest-friction way to automate IQcad.

Both hosts talk to the **same Layer-2 drawing model**: a surface, alignment, or entity created from Pascal is identical to one created from Python or from the GUI. The two languages differ only in syntax and in how they pass collections (see *The object model* (page 324)).

39.2 Running Pascal

Four command-line entry points drive the Pascal host:

Command	Description
`PASCAL <file.pas>` (alias `PAS`)	Run a `.pas` file once. Anything it registers is discarded when it finishes (unless it registers a command – then it stays resident, like `PASLOAD`).
`PASEVAL <expr>`	Evaluate a **single** Pascal expression and print its value at the command line (e.g. `PASEVAL ActiveDrawing.Name`).
`PASLOAD <file.pas>`	Run a file and leave it **session-resident**: any commands or event handlers it registers stay active until you close IQcad.
`PASSCRIPTS`	List the Pascal-defined commands currently registered.

A script is a normal Pascal `program`:

```
program demo;
begin
  WriteLine('hello from Pascal');
end.
```

A bare snippet without a `program ... begin ... end.` header is auto-wrapped, so `WriteLine('hi');` runs on its own.

Note

The wrapper detects a full program by the first `program` (or `unit`) keyword, **skipping any leading comments**. You can open a full-program file with a header comment block (`{ ... }`, `//` or `(* ... *)`) before the `program` line, as the bundled samples do.

39.2.1 Autoload and trust

Any `.pas` file dropped in `%APPDATA%\IQcad\scripts\` is loaded automatically at startup, using the **same content-hash trust model** as Python. The first time IQcad sees a script (and again whenever its SHA-256 content hash changes) it shows a one-time trust dialog offering **Always**, **This session**, or **Never**; your choice is remembered in `%APPDATA%\IQcad\scripts\script_trust.json`. A script you have never trusted is not executed.

39.2.2 Responsiveness and Esc

Scripts run on the GUI thread. In any loop that may run for a while, call `Yield` periodically: it pumps the message queue so the window stays responsive and lets **Esc** cancel the script. Cancellation is cooperative – it takes effect at the next `Yield` (or at the next script line between statements).

39.3 The object model

`ActiveDrawing` returns the current `Drawing`, which is the factory for everything else: it *creates* and *returns* `Entity`, `Surface`, and `Alignment` objects. Three helper objects – `Points`, `Profile`, and `Elements` – carry collections.

```
globals
|
|-- flat creators ...... AddLine/AddCircle/AddArc/AddEllipse/
|                        AddRectangle/AddText/AddSurveyPoint/
|                        AddPolyline/AddHatch/AddTunnelSectionLine
|                        (return a handle STRING),
|                        CreateLayer/SetActiveLayer/ZoomExtents
|-- WriteLine / Cmd / Yield
|-- RegisterCommand / UnregisterCommand
|-- OnEvent / OffEvent
|-- dialogs ............ ShowMessage/Confirm/InputString/SelectFile
|-- NewPoints  --------> Points   (the array-free input builder)
|
`-- ActiveDrawing  ----> Drawing
                          |
                          |  create/returns
                          |
           +--------------+----------------+
```

(continues on next page)

(continued from previous page)

```
          |              |               |
          v              v               v
       Entity         Surface         Alignment
       (geometry)     (TIN)           (horizontal)

       AddLine/Circle/Arc/Ellipse/Rectangle/Polyline/Text/
       SurveyPoint/Hatch/TunnelSectionLine     --> Entity
       CreateSurface / FindSurface / SurfaceAt --> Surface
       CreateAlignmentFromPolyline /
       FindAlignment / AlignmentAt             --> Alignment

       Surface.ProfileAlong  --> Profile  (Count + Station(i)/Z(i))
       Alignment.Elements    --> Elements (Count + Kind(i)/...)
```

The array-free design. PascalScript handles dynamic arrays poorly, so the Pascal API never takes or returns an array. Point *input* uses a `Points` builder (`NewPoints` then `Add(x, y, z)`); collection *results* come back as `Profile` and `Elements` objects that expose a `Count` plus indexer methods (`Station(i)` / `Z(i)` / `Kind(i)` …). This is the key difference from Python, which uses native lists and tuples.

Note

A **flat** global creator (e.g. `AddCircle(...)`) returns the new entity's **handle string**, whereas the **OO** method `ActiveDrawing.AddCircle(...)` returns an `Entity` object you can keep editing.

39.4 Class and global reference

39.4.1 Drawing

Obtained from `ActiveDrawing`. The factory for entities, surfaces, and alignments.

Member	Signature	Returns	Description
`Name`	property (read-only)	`string`	Drawing name.
`ActiveLayer`	property (read/write)	`string`	Name of the current layer.
`AddLine`	`AddLine(x1, y1, z1, x2, y2, z2)`	`Entity`	Line between two points.
`AddCircle`	`AddCircle(cx, cy, cz, r)`	`Entity`	Circle, centre + radius.
`AddArc`	`AddArc(sx, sy, sz, mx, my, mz, ex, ey, ez)`	`Entity`	Three-point arc.
`AddEllipse`	`AddEllipse(cx, cy, cz, radx, rady, rot)`	`Entity`	Ellipse with semi-axes and rotation.
`AddRectangle`	`AddRectangle(x1, y1, x2, y2)`	`Entity`	Rectangle from opposite corners.
`AddText`	`AddText(x, y, z, s, h)`	`Entity`	Text *s* at the point, height *h*.
`AddSurveyPoint`	`AddSurveyPoint(x, y, z, num)`	`Entity`	Survey point numbered *num*.
`AddPolyline`	`AddPolyline(Pts, closed)`	`Entity`	Polyline through a `Points` list.
`AddHatch`	`AddHatch(Pts, pattern, angle, scale)`	`Entity`	Hatch over a `Points` boundary.
`AddTunnelSectionLine`	`AddTunnelSectionLine(closed)`	`Entity`	Tunnel section line.
`CreateLayer`	`CreateLayer(n, color)`	`Boolean`	Create a layer (*color* is an ACI index).
`EntityByHandle`	`EntityByHandle(h)`	`Entity`	Look up an entity by handle (`nil` if absent).
`ZoomExtents`	`ZoomExtents`	–	Zoom to drawing extents.
`CreateSurface`	`CreateSurface(name, Pts)`	`Surface`	Build a TIN from a `Points` list of `(x, y, z)`.
`FindSurface`	`FindSurface(name)`	`Surface`	Look up a surface by name (`nil` if absent).
`SurfaceCount`	`SurfaceCount`	`Integer`	Number of surfaces.
`SurfaceAt`	`SurfaceAt(i)`	`Surface`	Surface by zero-based index.
`CreateAlignmentFromPolyline`	`CreateAlignmentFromPo radius, name)`	`Alignment`	Build an alignment from a polyline entity.
`CreateAlignmentFromPolylineZ`	`CreateAlignmentFromPo radius, name, vcurveLen)`	`Alignment`	Like `CreateAlignmentFromPolyline`, but also builds the vertical profile from the polyline's vertex elevations (one PVI per vertex; `vcurveLen` applied to interior PVIs, 0 = angular profile).
`FindAlignment`	`FindAlignment(name)`	`Alignment`	Look up an alignment by name (`nil` if absent).
`AlignmentCount`	`AlignmentCount`	`Integer`	Number of alignments.
`AlignmentAt`	`AlignmentAt(i)`	`Alignment`	Alignment by zero-based index.

39.4.2 Entity

A single drawing object returned by the OO `Add*` methods or by `EntityByHandle`.

Member	Signature	Returns	Description
`Handle`	property (read-only)	`string`	Persistent handle.
`TypeName`	property (read-only)	`string`	`'Line'` | `'Circle'` | `'Text'` | …
`Layer`	property (read/write)	`string`	Layer name.
`Color`	property (read/write)	`Integer`	ACI colour index.
`GetProperty`	`GetProperty(n)`	`string`	Read a typed property (e.g. circle `'Radius'`).
`SetProperty`	`SetProperty(n, v)`	–	Write a typed property.
`Delete`	`Delete`	–	Remove the entity; later access is a no-op guard.

39.4.3 Surface

A TIN surface created by `Drawing.CreateSurface` (or looked up via `FindSurface` / `SurfaceAt`).

Member	Signature	Returns	Description
`Name`	property (read-only)	`string`	Surface name.
`Handle`	property (read-only)	`string`	Persistent handle.
`ElevationAt`	`ElevationAt(x, y, out z)`	`Boolean`	Interpolated Z in *z*; **False** outside the TIN.
`AddPoint`	`AddPoint(x, y, z)`	–	Insert a vertex and rebuild the TIN.
`ProfileAlong`	`ProfileAlong(Pts)`	`Profile`	Long-section along a `Points` polyline.
`VolumeTo`	`VolumeTo(Other, cell, out cut, out fill)`	`Boolean`	Grid-approx cut/fill; cut where self is above *Other*.

39.4.4 Alignment

A horizontal alignment created by `CreateAlignmentFromPolyline` (or looked up via `FindAlignment` / `AlignmentAt`).

Member	Signature	Returns	Description
`Name`	property (read-only)	`string`	Alignment name.
`Handle`	property (read-only)	`string`	Persistent handle.
`Length`	property (read-only)	`Double`	Total length.
`StartStation`	property (read-only)	`Double`	Start chainage.
`EndStation`	property (read-only)	`Double`	End chainage.
`PointAtStation`	`PointAtStation(s, out x, out y, out z)`	–	World point at chainage *s*.
`DirectionAtStation`	`DirectionAtStation(s)`	`Double`	Bearing at *s*, in **radians**.
`ElevationAtStation`	`ElevationAtStation(s)`	`Double`	Profile elevation at *s*.
`StationOffsetAt`	`StationOffsetAt(x, y, out station, out offset)`	–	Nearest station + offset (`-` left / `+` right).
`Elements`	`Elements`	`Elements`	Geometry-elements collection.

39.4.5 Points, Profile, Elements

The array-free helper objects. `Points` is the input builder; `Profile` and `Elements` are result collections.

Object	Member	Signature	Returns
`Points`	`Add`	`Add(x, y, z)`	–
`Points`	`Count`	`Count`	`Integer`
`Profile`	`Count`	`Count`	`Integer`
`Profile`	`Station`	`Station(i)`	`Double`
`Profile`	`Z`	`Z(i)`	`Double`
`Elements`	`Count`	`Count`	`Integer`
`Elements`	`Kind`	`Kind(i)`	`string`
`Elements`	`Length`	`Length(i)`	`Double`
`Elements`	`Radius`	`Radius(i)`	`Double`
`Elements`	`StartStation`	`StartStation(i)`	`Double`

39.4.6 Module-level globals

Call these unqualified. The flat creators mirror the `Drawing` methods but return a handle **string** instead of an object.

Member	Signature	Returns	Description
ActiveDrawing	ActiveDrawing	Drawing	The current drawing.
NewPoints	NewPoints	Points	A fresh, empty Points builder.
WriteLine	WriteLine(s)	–	Print a line to the command line.
Cmd	Cmd(s)	Boolean	Run a command-line string.
Yield	Yield	–	Pump the UI; allow Esc to cancel.
AddLine	AddLine(x1, y1, z1, x2, y2, z2)	string	Flat line; returns a handle.
AddCircle	AddCircle(cx, cy, cz, r)	string	Flat circle; returns a handle.
AddArc	AddArc(sx, sy, sz, mx, my, mz, ex, ey, ez)	string	Flat arc; returns a handle.
AddEllipse	AddEllipse(cx, cy, cz, radx, rady, rot)	string	Flat ellipse; returns a handle.
AddRectangle	AddRectangle(x1, y1, x2, y2)	string	Flat rectangle; returns a handle.
AddText	AddText(x, y, z, s, h)	string	Flat text; returns a handle.
AddSurveyPoint	AddSurveyPoint(x, y, z, num)	string	Flat survey point; returns a handle.
AddPolyline	AddPolyline(Pts, closed)	string	Flat polyline; returns a handle.
AddHatch	AddHatch(Pts, pattern, angle, scale)	string	Flat hatch; returns a handle.
AddTunnelSectionLine	AddTunnelSectionLine(P closed)	string	Flat tunnel section line; returns a handle.
CreateLayer	CreateLayer(n, color)	Boolean	Create a layer (ACI *color*).
SetActiveLayer	SetActiveLayer(n)	–	Set the current layer.
ZoomExtents	ZoomExtents	–	Zoom to drawing extents.
RegisterCommand	RegisterCommand(name, aliases, category, desc, procname)	–	Register a command bound to a script proc.
UnregisterCommand	UnregisterCommand(name)	–	Remove a registered command.
OnEvent	OnEvent(eventName, procname)	–	Attach an event handler by proc name.
OffEvent	OffEvent(eventName, procname)	–	Detach an event handler.

39.5 Commands and events

39.5.1 Custom commands

RegisterCommand binds a new IQcad command to a script procedure **by name**. Load the script resident with PASLOAD (or drop it in the autoload folder) and the command can be typed at the prompt like any

built-in.

```
program ext;
procedure DoGrid(arg: string);
begin
  { arg arrives as a string -- parse it yourself }
  AddCircle(0, 0, 0, StrToFloatDef(arg, 1.0));
end;
begin
  RegisterCommand('PASGRID', 'PG', 'Tools', 'Draw a grid', 'DoGrid');
end.
```

Warning

Command arguments arrive as a **string**. Parse them yourself with `StrToFloatDef` / `StrToIntDef`.

Remove a registration with `UnregisterCommand('PASGRID')`.

39.5.2 Events

`OnEvent` attaches a handler procedure (by name) to a drawing event.

Event name	Handler signature	Fires when
`'entity_created'`	`procedure(handle: string)`	An entity is created.
`'entity_deleted'`	`procedure(handle: string)`	An entity is deleted.
`'selection_changed'`	`procedure` (no args)	The selection changes.

Detach a handler with `OffEvent(eventName, procname)`.

Note

Handlers run **synchronously** on the GUI thread. An exception inside a handler is logged to the command line, not fatal. A handler that itself creates entities re-fires `entity_created` and can recurse – guard with a module-level boolean (see `samples\pascal\03_events.pas`).

39.6 Dialogs

Four dialog globals – shared with the Python host – prompt the user. Run headlessly (no GUI), they return safe defaults so an unattended script never blocks.

Function	Signature	Headless default
`ShowMessage`	`ShowMessage(s)`	no-op
`Confirm`	`Confirm(s): Boolean`	`False`
`InputString`	`InputString(prompt, def: string; out v: string): Boolean`	`False` (`v` = *def*)
`SelectFile`	`SelectFile(title, filter: string; save: Boolean): string`	`''`

`SelectFile`'s *filter* is a pipe-delimited `Caption|Mask` pair (e.g. `'CSV files|*.csv'`); *save* selects an Open vs Save dialog.

39.7 Available RTL

The host registers a subset of the Pascal RTL plus a few additions. **Locale-invariant** float conversions are provided so that scripts behave identically regardless of the system decimal separator.

- **Strings:** `Copy`, `Pos`, `Length`, `UpperCase`, `LowerCase`, `Trim`.
- **Numbers:** `IntToStr`, `StrToInt`, `StrToIntDef`, `FloatToStr`, `StrToFloat`, `StrToFloatDef` (the float ones are **locale-invariant** – always `.` as the decimal point).
- **Math:** `Sin`, `Cos`, `Sqrt`, `Round`, `Trunc`, `Abs`, `Pi`, `ArcTan`, `Power`.
- **Formatting:** `FmtFloat(v: Double; decimals: Integer): string` – a locale-invariant fixed-decimal formatter.

Warning

`Format` (the `array of const` variant) is **not available**. Build strings with `FmtFloat` + `IntToStr` + concatenation, e.g. `WriteLine('STA ' + FmtFloat(s, 1) + ' Z=' + FmtFloat(z, 3))`.

39.8 Examples

Seven worked scripts ship under `samples\pascal\`:

- `samples\pascal\01_primitives.pas` – geometry, layers, the OO vs flat creators, `Points` for polyline + hatch.
- `samples\pascal\02_command.pas` – registering a custom command.
- `samples\pascal\03_events.pas` – wiring up event handlers.
- `samples\pascal\04_surface.pas` – TIN surfaces: create, query, profile, volume.
- `samples\pascal\05_alignment.pas` – alignments from a polyline.
- `samples\pascal\06_civil_workflow.pas` – chainage along an alignment while sampling a surface (the highlight).
- `samples\pascal\07_dialogs.pas` – the four user-interaction dialogs.

39.8.1 Surfaces

Create a TIN from seed points, interpolate it, take a long-section, then compare two surfaces by volume (`04_surface.pas`):

```
program surface;
var
  d: Drawing;
  pts, line: Points;
  eg, fg: Surface;
  prof: Profile;
  z, cut, fill: Double;
  i: Integer;
begin
  d := ActiveDrawing;

  pts := NewPoints;
  pts.Add(0, 0, 10.0);  pts.Add(50, 0, 11.0);  pts.Add(50, 40, 12.5);
  pts.Add(0, 40, 10.5); pts.Add(25, 20, 11.2);
  eg := d.CreateSurface('EG', pts);

  if eg.ElevationAt(25, 20, z) then       { False outside the TIN }
    WriteLine('Z at (25,20) = ' + FmtFloat(z, 3));

  eg.AddPoint(25, 40, 11.8);              { rebuilds the TIN }

  line := NewPoints;
  line.Add(0, 20, 0); line.Add(50, 20, 0);
  prof := eg.ProfileAlong(line);
  for i := 0 to prof.Count - 1 do
    WriteLine('STA ' + FmtFloat(prof.Station(i), 3) +
              '  Z ' + FmtFloat(prof.Z(i), 3));

  pts := NewPoints;
  pts.Add(0, 0, 11.0);  pts.Add(50, 0, 12.0);  pts.Add(50, 40, 13.5);
  pts.Add(0, 40, 11.5); pts.Add(25, 20, 12.2);
  fg := d.CreateSurface('FG', pts);
  if eg.VolumeTo(fg, 1.0, cut, fill) then
    WriteLine('cut=' + FmtFloat(cut, 2) + '  fill=' + FmtFloat(fill, 2));
end.
```

39.8.2 Alignments

Build an alignment from a polyline (fillet radius required for 3+ vertices), then query it (`05_alignment.pas`):

```
program alignment;
var
  d: Drawing;
  pts: Points;
  e: Entity;
  al: Alignment;
  el: Elements;
  x, y, z, dir, st, off: Double;
  i: Integer;
begin
  d := ActiveDrawing;
```

(continues on next page)

(continued from previous page)

```
  pts := NewPoints;
  pts.Add(0, 0, 0); pts.Add(100, 0, 0);
  pts.Add(100, 50, 0); pts.Add(160, 50, 0);
  e := d.AddPolyline(pts, False);

  al := d.CreateAlignmentFromPolyline(e.Handle, 20.0, 'CL-1');
  WriteLine('len=' + FmtFloat(al.Length, 3) +
            '  STA ' + FmtFloat(al.StartStation, 3) +
            '..' + FmtFloat(al.EndStation, 3));

  al.PointAtStation(50.0, x, y, z);
  dir := al.DirectionAtStation(50.0);             { radians }
  al.StationOffsetAt(50.0, 5.0, st, off);         { offset - left / +␣
↪right }

  el := al.Elements;
  for i := 0 to el.Count - 1 do
    WriteLine(el.Kind(i) +
              ' len=' + FmtFloat(el.Length(i), 3) +
              ' R=' + FmtFloat(el.Radius(i), 3));
end.
```

39.8.3 Combined civil workflow

The highlight: step chainage along an alignment and sample the ground surface at each station (`06_civil_workflow.pas`):

```
program civil_workflow;
var
  d: Drawing;
  pts: Points;
  e: Entity;
  surf: Surface;
  al: Alignment;
  step, s, x, y, z, gz: Double;
begin
  d := ActiveDrawing;

  pts := NewPoints;
  pts.Add(-20, -20, 10.0); pts.Add(180, -20, 14.0); pts.Add(180, 80, 16.0);
  pts.Add(-20, 80, 11.0);  pts.Add(80, 25, 13.0);
  surf := d.CreateSurface('EG', pts);

  pts := NewPoints;
  pts.Add(0, 0, 0); pts.Add(100, 0, 0);
  pts.Add(100, 50, 0); pts.Add(160, 50, 0);
  e := d.AddPolyline(pts, False);
  al := d.CreateAlignmentFromPolyline(e.Handle, 20.0, 'CL-1');

  step := 10.0;
  s := al.StartStation;
  while s <= al.EndStation + 1e-6 do
  begin
    al.PointAtStation(s, x, y, z);
```

(continues on next page)

(continued from previous page)

```
    if surf.ElevationAt(x, y, gz) then     { False outside the TIN }
      WriteLine('STA ' + FmtFloat(s, 1) + '  surf=' + FmtFloat(gz, 3))
    else
      WriteLine('STA ' + FmtFloat(s, 1) + '  surf=(outside TIN)');
    s := s + step;
    Yield;                                 { keep responsive / allow Esc }
  end;
end.
```

39.9 Dialogs and message boxes

Pascal scripts can pop a host-built parameter dialog or message box — the same UI the plugin SDK uses.

`CreateDialog(title): TScriptDialog` returns a dialog; describe fields, `Execute`, then read the values:

```
var
  dlg: TScriptDialog;
  shape: Integer; size: Double; layer: string; col: Integer; filled:␣
↪Boolean;
begin
  dlg := CreateDialog('Draw a shape');
  dlg.AddChoice('shape', 'Shape', 'Rectangle'#9'Circle', 0);  { TAB␣
↪separates options }
  dlg.AddFloat ('size',  'Size (m)', 10, 0, 0, 2);           { Min=Max=0␣
↪→ unbounded }
  dlg.AddText  ('layer', 'Layer name', 'SHAPES');
  dlg.AddColor ('color', 'Color', $00B400);                  { 0x00BBGGRR␣
↪}
  dlg.AddBool  ('filled','Filled?', False);
  if dlg.Execute then
  begin
    shape  := dlg.GetChoice('shape');   { selected index }
    size   := dlg.GetFloat('size');
    layer  := dlg.GetText('layer');
    col    := dlg.GetColor('color');
    filled := dlg.GetBool('filled');
  end;
end.
```

Builders: `AddText`, `AddMultiline`, `AddInt(…, Min, Max)`, `AddFloat(…, Min, Max, Decimals)`, `AddBool`, `AddChoice(Key, Caption, Options, DefaultIndex)`, `AddFile(…, Filter, SaveDlg)`, `AddFolder`, `AddColor`. Getters: `GetText` (also the selected choice's text), `GetInt`, `GetFloat`, `GetBool`, `GetChoice` (index), `GetColor`. `Execute` returns `False` if the user cancels.

`MessageBox(Text, Caption: string; Buttons, Icon: Integer): Integer` shows a message box and returns the pressed button. Integer codes — buttons: `0` OK, `1` OK/Cancel, `2` Yes/No, `3` Yes/No/Cancel; icons: `0` none, `1` info, `2` warning, `3` error, `4` question; results: `0` OK, `1` Cancel, `2` Yes, `3` No (a dismissed box returns `1`):

```
if MessageBox('Delete selection?', 'Confirm', 2, 4) = 2 then  { YesNo,␣
↪question → Yes }
  WriteLn('confirmed');
```

39.10 Live forms

A **live form** is a non-modal, host-owned tool window that stays open: you build controls, keep references, mutate them at runtime, and react to the user through event handlers. It is the scripting face of the plugin SDK's *live tool windows* (page 351).

`CreateForm(title): TScriptForm` returns a live form. Its `Add*` builders (`AddButton`, `AddLabel`, `AddText`, `AddMultiline`, `AddInt`, `AddFloat`, `AddBool`, `AddChoice`, `AddColor`, `AddFile`, `AddFolder`, `AddList`) each return a `TScriptControl` with typed `Get*` / `Set*` accessors and list ops; the form also has `BeginRow` / `EndRow`, `FindControl`, `Show` / `Hide` / `Close` / `IsVisible` / `SetTitle`. Assign the **name of a script procedure** to `OnButtonClicked` / `OnValueChanged` / `OnFormClosed` and the host calls it back:

```
var frm: TScriptForm;

procedure HandleClick(const Key: string);
var n: Integer;
begin
  if Key = 'add' then
  begin
    n := frm.FindControl('n').GetInt;
    frm.FindControl('log').AppendLine('Added: ' + IntToStr(n));
    frm.FindControl('status').SetText('Last added: ' + IntToStr(n));
  end;
end;

begin
  frm := CreateForm('Form demo');
  frm.AddLabel('status', 'Click Add to append a value');
  frm.AddInt('n', 'Value', 1, 0, 1000);
  frm.AddButton('add', 'Add');
  frm.AddList('log', 'Log');
  frm.OnButtonClicked := 'HandleClick';   { name of the script procedure }
  frm.Show;
end.
```

`CreatePanel(title): TScriptForm` is identical to `CreateForm` but **docks** the panel into the main window — it appears as a tab in the right-side dock strip instead of a floating window. The `Add*` builders and `On*` event handlers are exactly the same; only the container differs:

```
frm := CreatePanel('Form demo');   { docked twin of CreateForm }
```

39.11 Ribbon & menu

`AddRibbonButton(group, command, caption, hint, iconfile, large): Boolean` and `AddMenuItem(submenu, command, caption): Boolean` add GUI entries that **run a registered command** — a button in the **Plugins** ribbon tab and an item in the **Plugins** menu. Scripts appear under a single **Scripts** group/submenu; `iconfile` is an **absolute** path to a PNG (empty for a generic icon). See *Ribbon & menu contributions* (page 354).

```
AddRibbonButton('Demo', 'GREET', 'Greet', 'Say hello', '', True);
AddMenuItem('Demo', 'GREET', 'Greet');
```

39.12 Limitations and tips

- **PASEVAL is single-expression.** It evaluates one Pascal expression and prints its value; it is not a place for statements.
- **No ``Format`` / ``array of const``.** Build strings with `FmtFloat` + `IntToStr` + concatenation.
- **Array-free API.** Pass points via a `Points` builder; read collection results via `Profile` / `Elements` `Count` + indexers. There are no array parameters or array results.
- **Alignment fillet radius.** `CreateAlignmentFromPolyline` with `radius = 0` (the default) builds straight tangents through every vertex, including polylines with **3 or more vertices**. Pass a non-zero radius (at least `0.01`) only if you want the interior corners filleted into arcs.
- **``ElevationAt`` returns False outside the TIN.** Always check the result before using the out-parameter.
- **Offset sign.** `StationOffsetAt` returns an offset that is **negative to the left** and **positive to the right** of the alignment direction.
- **Command args are strings.** Parse with `StrToFloatDef` / `StrToIntDef`.
- **Limited event coverage.** Only `entity_created`, `entity_deleted`, and `selection_changed` exist; there is no `on_entity_modified` and no drawing-lifecycle or command events yet.
- **No background / async execution.** Scripts run on the GUI thread; call `Yield` to keep long loops responsive and Esc-cancelable.

40

Plugins (Native SDK)

IQcad can be extended in two complementary ways. **Scripting** (Python or Pascal, chapters *Python scripting* (page 301), *Python Scripting Reference* (page 309) and *Pascal Scripting* (page 323)) runs interpreted code inside IQcad and is ideal for automation, batch work, and one-off tools. **Plugins** are *native compiled* libraries (`.dll` / `.so` / `.dylib`) that IQcad loads at startup; they run at full machine speed, can be distributed as binaries without exposing source, and add first-class commands, drawing operations, and event handlers exactly like built-in features.

This chapter is the complete reference for the **native plugin SDK**: the design and ABI, the trust model and Plugin Manager, the manifest format, two step-by-step tutorials (Pascal and C), the bundled samples, and the full interface reference.

- *Plugins versus scripting* (page 339)
- *Architecture and design* (page 340)
  - *The contract* (page 340)
  - *The vtable-interface ABI* (page 340)
  - *The out-parameter convention* (page 340)
  - *Frozen interfaces and versioning* (page 341)
  - *Threading* (page 341)
  - *Memory and lifetime* (page 341)
- *The plugin lifecycle* (page 342)
  - *Loader states* (page 342)
- *Trust and the Plugin Manager* (page 343)
  - *Opening the manager* (page 343)
  - *Enabling, disabling, reloading* (page 343)
  - *State persistence* (page 343)
- *Installing a plugin* (page 344)
- *The manifest (`plugin.json`)* (page 344)

- *Tutorial 1 — your first plugin (Free Pascal)* (page 345)
  - *Prerequisites* (page 345)
  - *Step 1 — write the source* (page 345)
  - *Step 2 — write the manifest* (page 347)
  - *Step 3 — build* (page 348)
  - *Step 4 — install and enable* (page 348)
  - *Step 5 — run it* (page 348)
  - *Troubleshooting* (page 348)
- *Tutorial 2 — drawing geometry* (page 348)
- *Parameter dialogs and message boxes* (page 350)
  - *The rich message box* (page 350)
  - *A parameter dialog* (page 350)
  - *The same dialogs from scripting* (page 351)
- *Live tool windows* (page 351)
  - *A live window with a button that updates a list and a label* (page 352)
  - *The same live forms from scripting* (page 353)
  - *Docking a panel* (page 354)
- *Ribbon & menu contributions* (page 354)
  - *The same contributions from scripting* (page 355)
- *Writing a plugin in C or C++* (page 355)
- *The bundled samples* (page 357)
- *API reference* (page 359)
  - *Result codes (`TIqResult`)* (page 359)
  - *Core types* (page 359)
  - *Runtime helpers (`uIqSdkRtl`)* (page 359)
  - *Entry point* (page 360)
  - `IIqPlugin` (page 360)
  - `IIqHost` (page 360)
  - `IIqHostUI` (page 360)
  - `IIqDialog` (page 361)
  - `IIqForm` (page 361)
  - `IIqControl` (page 362)
  - `IIqFormSink` (page 363)

  - *IIqCommand and IIqCommandRegistry* (page 363)
  - *IIqDrawing* (page 364)
  - *IIqEntity and IIqLayer* (page 365)
  - *Civil: IIqSurface, IIqProfilePoint, IIqAlignment* (page 365)
  - *Events: IIqEventSource and IIqEventSink* (page 366)
  - *Strings, enumerators, runnables* (page 366)
- *Debugging plugins* (page 367)
  - *Enabling the debug log* (page 367)
  - *What gets logged* (page 367)
  - *Discovering plugin commands* (page 367)
- *Building for other platforms* (page 368)
- *Versioning and compatibility — summary* (page 368)

Note

The plugin contract is shared with the scripting hosts — a plugin and a Pascal/Python script see the *same* drawing, layer, entity, surface and alignment model. If you only need automation, scripting is simpler; reach for a plugin when you need native speed, binary distribution, or you are shipping a tool to other users.

40.1 Plugins versus scripting

Aspect	Scripting (Python / Pascal)	Native plugin
Language	Python 3.12 or PascalScript	Any language that can produce a C-ABI shared library (Free Pascal, C, C++, Rust, …)
Distribution	Source text (`.py` / `.pas`)	Compiled binary + a small JSON manifest
Speed	Interpreted	Native machine code
Loading	Run on demand from the editor or command line	Discovered and loaded at startup (once enabled)
Trust	Sandboxed-ish; runs in-process	**Full process trust** — must be enabled with consent
API	`IScript*` host objects	`IIq*` interfaces (a thin, frozen ABI overlay on the same engine)

Both share the same underlying engine, so concepts such as *commands*, *events*, *layers*, and the *civil model* behave identically.

40.2 Architecture and design

40.2.1 The contract

Everything a plugin can see or do is expressed through one Pascal unit and its hand-maintained C mirror:

- `cad/plugins/uIqSdk.pas` — the **contract**: every `IIq*` interface, the result codes, the records, and the entry-point type. This file is the single source of truth.
- `cad/plugins/uIqSdkRtl.pas` — small **runtime helpers** a Pascal plugin uses to build strings and enumerators (`IqStr`, `IqRGB`, `IqEnumStrings` …).
- `cad/plugins/csdk/iqcad_sdk.h` — the **C/C++ header**, a 1:1 mirror of the contract for plugins written in C or C++.

A plugin never links against IQcad. It only `uses uIqSdk` (Pascal) or `#include "iqcad_sdk.h"` (C), exports one function, and talks to the host entirely through interface pointers handed to it at load time.

40.2.2 The vtable-interface ABI

The SDK uses a **COM-style vtable ABI**:

- Every object is an interface deriving from `IInterface` (`IUnknown`): the first three slots are `QueryInterface`, `AddRef`, `Release`. Objects are **reference-counted**; the last `Release` frees them.
- Every method uses the **C calling convention** (`cdecl`). This is the one convention shared by all three shipped 64-bit targets, so a Pascal host and a C plugin (or vice-versa) interoperate without glue.
- **No managed types ever cross the boundary.** There are no Pascal `string`/`array`/`Variant` parameters in the contract. Variable length data is exposed as ref-counted interfaces (`IIqString`, `IIqEnumerator`); fixed-size data is plain records (`TIqPoint3D`, `TIqColor`, `TIqVariant`). Text is always UTF-16 (`PWideChar` / `IIqString`).

> **Note**
>
> There is **no Windows COM runtime** involved — no `CoCreateInstance`, no registry, no apartments. "COM-style" here means only *the binary layout*: a vtable pointer followed by `cdecl` methods, with `QueryInterface`/`AddRef`/`Release` at the front. This works identically on Windows, Linux, and macOS, including Apple silicon.

40.2.3 The out-parameter convention

A function that *returns a value type* (a `Double`, a `TIqColor`, a `LongInt`) returns it directly:

```
function Color: TIqColor; cdecl;
function ElevationAt(const X, Y: Double; out Z: Double): TIqResult; cdecl;
```

A function that *returns an interface or a string* returns it through an `out` parameter and yields a *result code* (page 359) instead:

```
function ActiveDrawing(out D: IIqDrawing): TIqResult; cdecl;
function Name(out S: IIqString): TIqResult; cdecl;
```

This is mandatory for ABI safety: returning a ref-counted interface *by function result* is not C-ABI-compatible across the Free Pascal / C boundary. **Always check both** the result code and the out-parameter — for example `ActiveDrawing` returns `IQ_OK` with `D = nil` when no drawing is open.

40.2.4 Frozen interfaces and versioning

Once an interface has shipped, **its layout never changes** — methods are never reordered, removed, or re-signed. The contract evolves only by:

- **adding** methods to the *end* of an interface in a new minor version (existing plugins keep working — they simply do not call the new slots), or
- **adding** an entirely new interface with a new GUID (`IIqDrawing2` …) when a breaking change is unavoidable.

Two constants in `uIqSdk.pas` track this:

- `IQSDK_API_MAJOR` — bumped only on a breaking change.
- `IQSDK_API_MINOR` — bumped on every additive change.

A plugin declares the *minimum* minor it needs in its manifest (`apiVersion.min_minor`); the loader admits it only when `host.major = manifest.major` **and** `host.minor >= manifest.min_minor` (see *The manifest (plugin.json)* (page 344)). The current contract is **API 1.6**.

Warning

While the SDK is pre-1.0 it may still change *in place* between IQcad builds. A plugin compiled against an older contract can crash a newer IQcad. **Rebuild your plugins from the current SDK whenever you update IQcad**, until the contract is declared stable.

40.2.5 Threading

The drawing model is **single-threaded**. All plugin entry points (command `Invoke`, event callbacks) are called on IQcad's main thread, and you must only touch the drawing from that thread. If you spin up a worker thread, marshal back with:

```
Host.PostToMainThread(myRunnable);   // myRunnable.Run executes on the␣
↪main thread
```

Calling drawing methods from another thread returns `IQ_E_WRONGTHREAD` or causes undefined behaviour.

40.2.6 Memory and lifetime

Objects are reference-counted. In Pascal this is automatic: assign an interface variable and the compiler emits `AddRef`/`Release` for you. In C you call `lpVtbl->AddRef` / `lpVtbl->Release` yourself (the samples use static singletons whose lifetime equals the library, so their refcount methods are no-ops).

Note

The plugin module is never unloaded while IQcad runs. When you disable a plugin the loader deactivates it (it unregisters its commands and drops its references) but does **not** `dlclose` / `FreeLibrary` the binary. The host may still hold ref-counted objects whose *code lives in your library*; unmapping it

would turn the next `Release` into a crash. The module stays mapped until IQcad exits. A disabled plugin is genuinely inert — its commands are gone — it simply remains in memory.

40.3 The plugin lifecycle

A plugin binary exports exactly one function. Its lifetime is a fixed sequence driven by the loader:

```
discover  ──►  load (dlopen)  ──►  IqcadPluginInit(host, out plugin)
                                              │
                                              ▼
                                   plugin.Describe(out info)   ◄── id must␣
↪match manifest
                                              │
                                              ▼
                                   plugin.Activate(host)       ◄── register␣
↪commands, subscribe events
                                              │
                                    … commands run …
                                              │
                                              ▼
                                   plugin.Deactivate           ◄──␣
↪unregister, release cookies
                                              │
                                              ▼
                                   references dropped (module stays mapped)
```

1. **Discover** — at startup the loader scans the plugins directory for sub-folders containing a `plugin.json`, reads each manifest, and version-gates it.
2. **Load** — for each *enabled and consented* compatible plugin, the loader `dlopen`s the binary and looks up the export `IqcadPluginInit`.
3. **Init** — `IqcadPluginInit(host, out plugin)` creates your plugin object and returns it. The `host` you receive is your handle to everything.
4. **Describe** — the loader calls `plugin.Describe` and verifies the reported `Id` matches the manifest (a mismatch quarantines the plugin).
5. **Activate** — `plugin.Activate(host)` is where you register commands and subscribe to events.
6. **Deactivate** — on shutdown or when the user disables the plugin, `plugin.Deactivate` runs; unregister your commands and release any event-subscription cookies here.

40.3.1 Loader states

Each discovered plugin carries a state visible in the Plugin Manager:

State	Meaning
Discovered	Manifest read, not yet gated.
Eligible	Compatible with this IQcad, not (yet) loaded.
Incompatible	No binary for this platform, or API mismatch — never loaded.
Active	Loaded and activated; its commands are live.
Failed	Load or activation raised an error — quarantined, with the error recorded.

40.4 Trust and the Plugin Manager

A native plugin runs with the **full privileges of IQcad**: it can read and write files, open network connections, and a faulty plugin can crash the application. IQcad therefore treats plugins as untrusted until you say otherwise.

Plugins are disabled by default. A freshly dropped-in plugin is discovered and listed but does *not* load until you enable it once and accept a trust prompt. Your choice is remembered, so the plugin autoloads on subsequent launches.

40.4.1 Opening the manager

- **Command** — type **`PLUGINS`** (alias `PLUGINMANAGER`).
- **Ribbon** — **Tools → Scripting → Plugins**.
- **Menu** — **Tools → Plugin Manager…**

The dialog lists every discovered plugin with its *Enabled* flag, name, version, publisher and status, plus a details pane showing the id, the required API version, the install path, and any last error.

40.4.2 Enabling, disabling, reloading

- **Enable** — select a plugin and click *Enable*. The first time, a trust dialog appears:

  ```
  Native plugins run with full process trust: they can read/write
  files, access the network, and a faulty plugin can crash IQcad.
  Only enable plugins you trust.  Enable 'com.example.myfirst'?
  ```

  Click *Yes* to grant consent (remembered) and load the plugin. Its commands become available immediately and are announced on the command line.

- **Disable** — deactivates the plugin (its commands disappear) and clears the enabled flag so it will not autoload next time.
- **Reload** — re-scans the plugins directory and reconciles state; use it after installing or rebuilding a plugin without restarting IQcad.
- **Open Folder** — opens the plugin's install directory (or the plugins root) in your file manager.

40.4.3 State persistence

Each plugin's *enabled* and *consented* flags are stored per-id in IQcad's `settings.json` and reapplied at startup. At launch IQcad *reconciles*: it loads only plugins that are eligible **and** enabled **and** consented.

40.5 Installing a plugin

Plugins live under a per-user directory:

Platform	Plugins directory
Linux / macOS	`~/.config/IQcad/plugins/`
Windows	`%APPDATA%\IQcad\plugins\`

Each plugin gets its **own sub-folder** containing the binary and the manifest:

```
~/.config/IQcad/plugins/
    com.example.myfirst/
        plugin.json
        libMyFirstPlugin.so        (Linux)
        MyFirstPlugin.dll          (Windows)
        libMyFirstPlugin.dylib     (macOS)
```

The sub-folder name is free-form — the loader reads the real id from `plugin.json`. After copying the files, either restart IQcad or click **Reload** in the Plugin Manager, then enable the plugin.

40.6 The manifest (`plugin.json`)

Every plugin ships a `plugin.json` next to its binary. It tells the loader who the plugin is, which IQcad API it needs, and which binary file to load on each platform.

Field	Type	Meaning
`schemaVersion`	int	Manifest format version — currently always `1`.
`id`	string	**Required.** Reverse-DNS unique id (e.g. `com.example.myfirst`). Must match the id returned by `Describe`.
`name`	string	Display name shown in the Plugin Manager.
`version`	string	The plugin's own version (informational, e.g. `"1.0.0"`).
`apiVersion.major`	int	Required SDK major version (must equal the host major).
`apiVersion.min_minor`	int	Minimum SDK minor version the plugin needs.
`publisher`	string	Optional publisher name.
`description`	string	Optional one-line description.
`icon`	string	Optional icon filename (reserved).
`entry`	object	**Required.** Map of *platform key* → *library filename* for that platform.

The recognised platform keys are `x86_64-win64`, `x86_64-linux` and `aarch64-darwin`. A complete manifest:

```
{
  "schemaVersion": 1,
  "id": "com.example.myfirst",
  "name": "My First Plugin",
  "version": "1.0.0",
```

(continues on next page)

(continued from previous page)

```
  "apiVersion": { "major": 1, "min_minor": 0 },
  "entry": {
    "x86_64-win64":   "MyFirstPlugin.dll",
    "x86_64-linux":   "libMyFirstPlugin.so",
    "aarch64-darwin": "libMyFirstPlugin.dylib"
  },
  "publisher": "Example",
  "description": "Adds the HELLO command."
}
```

Warning

The filename under `entry` must **exactly** match the binary you produce for that platform. The loader builds the path as *plugin-folder* + *entry filename* and loads precisely that file. A mismatch (e.g. manifest says `libMyFirstPlugin.so` but the build produced `MyFirstPlugin`) is the single most common "my plugin won't load" mistake.

40.7 Tutorial 1 — your first plugin (Free Pascal)

This tutorial builds a complete, working plugin that adds a `HELLO` command. It is the smallest useful plugin and shows every required piece.

40.7.1 Prerequisites

- **Free Pascal / Lazarus** (the same toolchain IQcad is built with).
- The IQcad/ICAD source tree, for the SDK units. Note the path to `cad/plugins/` — your project's unit search path must point there so `uIqSdk` and `uIqSdkRtl` resolve.

40.7.2 Step 1 — write the source

Create `MyFirstPlugin.lpr`:

```
library MyFirstPlugin;

{$mode ObjFPC}{$H+}
{$interfaces COM}          // ref-counted COM-style interfaces

uses
  Classes, SysUtils,
  uIqSdk, uIqSdkRtl;       // the SDK contract + runtime helpers

type
  { The command the user runs by typing HELLO. A command is just an
    object implementing IIqCommand. }
  TGreetCommand = class(TInterfacedObject, IIqCommand)
  private
    FHost: IIqHost;
  public
    constructor Create(const AHost: IIqHost);
    function Name(out S: IIqString): TIqResult; cdecl;
    function Aliases(out S: IIqString): TIqResult; cdecl;
```

(continues on next page)

(continued from previous page)

```
    function Category(out S: IIqString): TIqResult; cdecl;
    function HelpText(out S: IIqString): TIqResult; cdecl;
    function Invoke(const Args: IIqEnumerator): TIqResult; cdecl;
  end;

  { The plugin object — created by IqcadPluginInit, owns the command. }
  TMyFirstPlugin = class(TInterfacedObject, IIqPlugin)
  private
    FHost: IIqHost;
  public
    function Describe(out Info: TIqPluginInfo): TIqResult; cdecl;
    function Activate(const Host: IIqHost): TIqResult; cdecl;
    function Deactivate: TIqResult; cdecl;
  end;

{ TGreetCommand }

constructor TGreetCommand.Create(const AHost: IIqHost);
begin
  inherited Create;
  FHost := AHost;
end;

function TGreetCommand.Name(out S: IIqString): TIqResult; cdecl;
begin S := IqStr('HELLO'); Result := IQ_OK; end;

function TGreetCommand.Aliases(out S: IIqString): TIqResult; cdecl;
begin S := IqStr('HI'); Result := IQ_OK; end;   // comma-separated for␣
↪several

function TGreetCommand.Category(out S: IIqString): TIqResult; cdecl;
begin S := IqStr('Tutorial'); Result := IQ_OK; end;

function TGreetCommand.HelpText(out S: IIqString): TIqResult; cdecl;
begin S := IqStr('Print a friendly greeting'); Result := IQ_OK; end;

function TGreetCommand.Invoke(const Args: IIqEnumerator): TIqResult; cdecl;
begin
  // Args yields the command's arguments as IIqString items (none here).
  FHost.Log(illInfo, 'Hello from my first IQcad plugin!');
  Result := IQ_OK;
end;

{ TMyFirstPlugin }

function TMyFirstPlugin.Describe(out Info: TIqPluginInfo): TIqResult;␣
↪cdecl;
begin
  Info.Id := IqStr('com.example.myfirst');   // MUST match plugin.json
  Info.Name := IqStr('My First Plugin');
  Info.Version.Major := 1;
  Info.Version.Minor := 0;
  Result := IQ_OK;
end;

function TMyFirstPlugin.Activate(const Host: IIqHost): TIqResult; cdecl;
```

(continues on next page)

(continued from previous page)

```
var reg: IIqCommandRegistry;
begin
  FHost := Host;
  if Host.Commands(reg) <> IQ_OK then Exit(IQ_E_FAIL);
  reg.Register(TGreetCommand.Create(Host));
  Result := IQ_OK;
end;

function TMyFirstPlugin.Deactivate: TIqResult; cdecl;
var reg: IIqCommandRegistry;
begin
  if (FHost <> nil) and (FHost.Commands(reg) = IQ_OK) then
    reg.Unregister('HELLO');
  FHost := nil;
  Result := IQ_OK;
end;

{ The one exported symbol the loader looks for. }
function IqcadPluginInit(const Host: IIqHost; out Plugin: IIqPlugin):␣
↪TIqResult; cdecl;
begin
  Plugin := TMyFirstPlugin.Create;
  Result := IQ_OK;
end;

exports
  IqcadPluginInit;

begin
end.
```

That is the entire plugin. Note the four required ingredients:

1. `{$interfaces COM}` so interfaces are reference-counted.
2. One class implementing *IIqPlugin* (page 360).
3. One or more classes implementing *IIqCommand* (page 363).
4. The single exported `IqcadPluginInit` returning the plugin via its `out` parameter.

40.7.3 Step 2 — write the manifest

Create `plugin.json` beside the source (see *The manifest (plugin.json)* (page 344) for all fields):

```
{
  "schemaVersion": 1,
  "id": "com.example.myfirst",
  "name": "My First Plugin",
  "version": "1.0.0",
  "apiVersion": { "major": 1, "min_minor": 0 },
  "entry": {
    "x86_64-linux":   "libMyFirstPlugin.so",
    "x86_64-win64":   "MyFirstPlugin.dll",
    "aarch64-darwin": "libMyFirstPlugin.dylib"
  }
}
```

40.7.4 Step 3 — build

With a Lazarus project file (`.lpi`) whose *other unit files* path includes `cad/plugins`, build the library:

```
lazbuild MyFirstPlugin.lpi
```

or directly with the compiler, pointing `-Fu` at the SDK units:

```
fpc -Fu/path/to/ICAD/cad/plugins MyFirstPlugin.lpr
```

On Linux this produces `libMyFirstPlugin.so` — the exact name the manifest's `x86_64-linux` entry expects. On Windows build a `.dll`, on macOS a `.dylib`, each matching its `entry` filename.

40.7.5 Step 4 — install and enable

Copy the binary and `plugin.json` into a sub-folder of the plugins directory:

```
mkdir -p ~/.config/IQcad/plugins/com.example.myfirst
cp libMyFirstPlugin.so plugin.json ~/.config/IQcad/plugins/com.example.
↪myfirst/
```

Start IQcad (or click **Reload** in the Plugin Manager), open the manager with **`PLUGINS`**, select *My First Plugin*, click **Enable**, and accept the trust prompt. The command line confirms:

```
Plugin command available: HELLO  (aliases: HI)
```

40.7.6 Step 5 — run it

Type `HELLO` (or the alias `HI`) at the command line. You will see:

```
Hello from my first IQcad plugin!
```

Type **`HELP`** and scroll to the **Loaded plugin commands** section to see your command listed alongside any other plugins'.

40.7.7 Troubleshooting

- **Plugin not listed** — the folder must contain a valid `plugin.json` with a non-empty `id`. Click *Reload*.
- **Status "Incompatible"** — no `entry` for your platform key, or the required `apiVersion` is newer than this IQcad. Check the details pane.
- **Status "Failed"** — the binary could not be loaded, the `IqcadPluginInit` export is missing, or the `Describe` id does not match the manifest. The exact error is in the details pane and the *debug log* (page 367).
- **Enabled but nothing happens** — confirm the command was announced on the command line; check `HELP`.

40.8 Tutorial 2 — drawing geometry

A more useful plugin draws into the active drawing. This `DRAWDEMO` command creates a layer and adds a few entities. Only the command's `Invoke` differs from Tutorial 1; everything else (the plugin class, the manifest, the build) is identical in shape.

```
function TDrawDemoCommand.Invoke(const Args: IIqEnumerator): TIqResult;␣
↪cdecl;
var
  d: IIqDrawing;
  e: IIqEntity;
  a, b, c: TIqPoint3D;
begin
  // 1. Get the active drawing (check BOTH the code and the out-param).
  if (FHost.ActiveDrawing(d) <> IQ_OK) or (d = nil) then
  begin
    FHost.Log(illInfo, 'DRAWDEMO: open a drawing first');
    Exit(IQ_OK);
  end;

  // 2. Create a green layer and make it current.
  //    TIqColor is packed RGB 0x00BBGGRR — build it with IqRGB, NOT an
  //    ACI index. IqRGB(0,180,0) is green.
  d.CreateLayer('DEMO', IqRGB(0, 180, 0));
  d.SetActiveLayer('DEMO');

  // 3. Add entities. Points are TIqPoint3D records.
  a.X := 0;  a.Y := 0;  a.Z := 0;
  b.X := 10; b.Y := 10; b.Z := 0;
  d.AddLine(a, b, e);

  c.X := 5; c.Y := 5; c.Z := 0;
  d.AddCircle(c, 3, e);

  // 4. Read something back: the new circle's handle.
  if e <> nil then
  begin
    var h: IIqString;
    if e.Handle(h) = IQ_OK then
      FHost.Log(illInfo, PWideChar(WideString('DRAWDEMO: circle handle = '
        + WideString(h.Chars))));
  end;

  // 5. Repaint so the new geometry shows immediately.
  d.Invalidate;
  Result := IQ_OK;
end;
```

Key points:

- `IIqDrawing` is reached through `Host.ActiveDrawing`. It offers `AddLine`, `AddCircle`, `AddArc`, `AddRectangle`, `AddText`, `AddHatch` and more (see *IIqDrawing* (page 364)).
- **Colours are RGB, not ACI indices.** Use `IqRGB(r, g, b)` (Pascal) or `IQ_RGB(r, g, b)` (C). Passing a small integer like `3` yields `RGB(3,0,0)` — almost black — a classic mistake.
- Call `Invalidate` (and optionally `ZoomExtents`) so the view refreshes.

Civil work follows the same pattern: `AddSurveyPoint`, `CreateSurface`, `IIqSurface.ElevationAt`, alignment queries — see *Civil: IIqSurface, IIqProfilePoint, IIqAlignment* (page 365) and the `civilui` sample.

40.9 Parameter dialogs and message boxes

A plugin cannot create its own forms — IQcad is an LCL application and a second LCL instance inside the plugin would crash the process (see *Plugins (Native SDK)* (page 337) intro). Instead the host builds the UI: you **describe** a dialog and IQcad creates, shows, and reads back every widget.

40.9.1 The rich message box

`IIqHostUI.MessageBox` takes a caption, a button set, and an icon, and returns which button the user pressed:

```
var ui: IIqHostUI; ans: TIqMsgResult;
begin
  FHost.UI(ui);
  ui.MessageBox('Delete the selection?', 'Confirm', imbYesNo, imiQuestion,␣
↪ans);
  if ans = imrYes then
    ...;   // proceed
end;
```

Button sets are `imbOK` / `imbOKCancel` / `imbYesNo` / `imbYesNoCancel`; icons `imiNone` / `imiInfo` / `imiWarning` / `imiError` / `imiQuestion`; results `imrOK` / `imrCancel` / `imrYes` / `imrNo`. A dismissed dialog (or no provider) returns `imrCancel`.

40.9.2 A parameter dialog

Build a form, run it, read the values:

```
var ui: IIqHostUI; dlg: IIqDialog; ok: LongBool;
    shape: LongInt; size: Double; layer: IIqString; col: TIqColor; filled:␣
↪LongBool;
begin
  FHost.UI(ui);
  ui.CreateDialog('Draw a shape', dlg);
  dlg.AddChoice('shape', 'Shape', 'Rectangle'#9'Circle', 0);  // TAB␣
↪separates options
  dlg.AddFloat ('size',  'Size (m)', 10, 0, 0, 2);            // Min=Max=0␣
↪→ unbounded, 2 decimals
  dlg.AddText  ('layer', 'Layer name', 'SHAPES');
  dlg.AddColor ('color', 'Color', IqRGB(0, 180, 0));
  dlg.AddBool  ('filled','Filled (hatch)?', False);

  dlg.Execute(ok);
  if not ok then Exit(IQ_OK);          // user cancelled (or headless)

  dlg.GetChoice('shape', shape);
  dlg.GetFloat ('size',  size);
  dlg.GetText  ('layer', layer);
  dlg.GetColor ('color', col);
  dlg.GetBool  ('filled', filled);
  // … create the layer and draw the rectangle/circle …
end;
```

Each field kind maps to the natural editor (text → edit, multiline → memo, int/float → spin, bool → checkbox, choice → dropdown, file/folder → a path field with *Browse*, color → a swatch + colour picker). See the full full `IIqDialog` method list in the *API reference* (page 359), and the complete worked example in the

`dialogdemo` / `cdialogdemo` sample.

40.9.3 The same dialogs from scripting

The dialog engine is shared with the scripting hosts, so Python and Pascal scripts get the same parameter dialogs and rich message box.

Python (see *Python Scripting Reference* (page 309)):

```
dlg = iqcad.create_dialog("Draw a shape")
dlg.add_choice("shape", "Shape", ["Rectangle", "Circle"], 0)
dlg.add_float("size", "Size (m)", 10.0, decimals=2)
dlg.add_color("color", "Color", 0x00B400)        # 0x00BBGGRR
if dlg.execute():
    shape = dlg.get_choice("shape")
    size  = dlg.get_float("size")
answer = iqcad.message_box("Done", "DialogDemo", iqcad.MB_OK, iqcad.MB_
↪ICON_INFO)
```

PascalScript (see *Pascal Scripting* (page 323)):

```
var dlg: TScriptDialog;
begin
  dlg := CreateDialog('Draw a shape');
  dlg.AddChoice('shape', 'Shape', 'Rectangle'#9'Circle', 0);
  dlg.AddFloat('size', 'Size (m)', 10, 0, 0, 2);
  if dlg.Execute then
    WriteLn(dlg.GetFloat('size'));
end;
```

40.10 Live tool windows

A parameter dialog (*Parameter dialogs and message boxes* (page 350)) is **modal**: it blocks, the user fills it in, `Execute` returns. A **live tool window** is the opposite — a *non-modal*, host-owned window that stays open after the command returns. The plugin builds controls, **retains the handles**, mutates them at runtime, and reacts to the user through a callback object. This is how you build a side panel, a running log, or an interactive tool.

As with dialogs, the **host owns every widget** — the plugin never touches LCL. `IIqHostUI.CreateForm` (API 1.4) returns an *IIqForm* (page 361) builder; each `Add*` creates a real widget *now* and hands back a retained *IIqControl* (page 362) handle. You pass an *IIqFormSink* (page 363) your plugin implements; the host calls it (on the main thread, boundary-guarded) when a button is clicked, a value changes, or the window closes.

Lifetime is the careful part:

- The host keeps the shown window in a registry, so it stays open even if your plugin releases its `IIqForm` handle.
- When the window closes — the user clicks `[X]`, you call `Close`, your plugin is deactivated, or IQcad exits — the host fires `OnFormClosed` once, then marks the form **and all its control handles invalid**.
- A call on a stale (post-close) handle safely returns `IQ_E_FAIL` — never a dangling dereference.

- **Release your handles in ``Deactivate``** (and you may drop them in `OnFormClosed`). Because plugin modules are never unloaded, even a leaked handle is safe — the wrapper is host-owned memory.

40.10.1 A live window with a button that updates a list and a label

This is the heart of the `formdemo` sample. The command builds a form with a status label, an integer spin, an *Add* button, and a read-only log list, then shows it. The sink's `OnButtonClicked` reads the spin value *live*, appends a line to the log, and updates the label — all while the window stays open:

```
{ The sink — implemented by the plugin; holds a plain ref to the form so
  its callbacks can drive the live controls. }
type
  TFormDemoSink = class(TInterfacedObject, IIqFormSink)
  private
    FForm: IIqForm;          // plain ref — the host registry owns the␣
↪window
  public
    procedure SetForm(const AForm: IIqForm);
    procedure OnButtonClicked(const Key: PWideChar); cdecl;
    procedure OnValueChanged(const Key: PWideChar); cdecl;
    procedure OnFormClosed; cdecl;
  end;

procedure TFormDemoSink.SetForm(const AForm: IIqForm);
begin FForm := AForm; end;

procedure TFormDemoSink.OnButtonClicked(const Key: PWideChar); cdecl;
var c: IIqControl; n: LongInt;
begin
  if (FForm = nil) or (WideString(Key) <> 'add') then Exit;

  // Read the spin value LIVE through its handle.
  n := 0;
  if (FForm.FindControl('n', c) = IQ_OK) and (c <> nil) then c.GetInt(n);

  // Append a line to the log list.
  if (FForm.FindControl('log', c) = IQ_OK) and (c <> nil) then
    c.AppendLine(PWideChar(WideString(WideFormat('Added: %d', [n]))));

  // Update the status label.
  if (FForm.FindControl('status', c) = IQ_OK) and (c <> nil) then
    c.SetText(PWideChar(WideString(WideFormat('Last added: %d', [n]))));
end;

procedure TFormDemoSink.OnValueChanged(const Key: PWideChar); cdecl;
begin end;

procedure TFormDemoSink.OnFormClosed; cdecl;
begin FForm := nil; end;   // the host already freed the window

{ The command — builds and shows the form. }
function TFormDemoCommand.Invoke(const Args: IIqEnumerator): TIqResult;␣
↪cdecl;
var ui: IIqHostUI; frm: IIqForm; c: IIqControl; sink: TFormDemoSink;
begin
  ui := nil; FHost.UI(ui);
```

(continues on next page)

(continued from previous page)

```
  if ui = nil then Exit(IQ_OK);                // headless — nothing to show

  sink := TFormDemoSink.Create;
  FSink := sink;                               // keep the sink alive

  frm := nil;
  if (ui.CreateForm('Form demo', FSink, frm) <> IQ_OK) or (frm = nil) then
  begin FSink := nil; Exit(IQ_OK); end;        // no form provider
  sink.SetForm(frm);
  FForm := frm;                                // retain — release in␣
↪Deactivate

  frm.AddLabel ('status', 'Click Add to append a value', c);
  frm.AddInt   ('n', 'Value', 1, 0, 1000, c);
  frm.AddButton('add', 'Add', c);
  frm.AddList  ('log', 'Log', c);

  frm.Show;                                    // non-modal — it outlives␣
↪Invoke
  Result := IQ_OK;
end;
```

Release the retained `FForm` and `FSink` handles in `Deactivate` (the already-shown window stays alive in the host registry regardless). The complete, compilable version — including the plugin object and the `min_minor: 4` manifest — is the `formdemo` / `cformdemo` sample.

40.10.2 The same live forms from scripting

Live forms are part of the shared UI engine, so Python and Pascal scripts open them too.

Python (see *Python Scripting Reference* (page 309)) — `iqcad.create_form(title, handler)` returns a live form; the optional `handler` object's `on_button_clicked(key)` / `on_value_changed(key)` / `on_form_closed()` methods are called by the host:

```
form = iqcad.create_form("Form demo", handler)
form.add_label("status", "Click Add to append a value")
form.add_int("n", "Value", 1, 0, 1000)
form.add_button("add", "Add")
form.add_list("log", "Log")
form.show()
```

PascalScript (see *Pascal Scripting* (page 323)) — `CreateForm(title)` returns a `TScriptForm`; assign script procedure names to its event handlers:

```
frm := CreateForm('Form demo');
frm.AddInt('n', 'Value', 1, 0, 1000);
frm.AddButton('add', 'Add');
frm.AddList('log', 'Log');
frm.OnButtonClicked := 'HandleClick';   { name of a script procedure }
frm.Show;
```

40.10.3 Docking a panel

A live tool window normally floats. `IIqHostUI.CreatePanel` (API 1.5) is the **docked** alternative: it is *identical* to `CreateForm` — same arguments, same returned *IIqForm* (page 361) builder, same *IIqControl* (page 362) handles and the same *IIqFormSink* (page 363) callbacks — but instead of a floating window the form's widgets are hosted as a **tab in a docked strip on the right of the main window**, between the canvas and the Object Inspector. The strip appears when the first panel docks and hides again when its last tab closes. `Show` adds (or activates) the tab; `Close` — or the user closing the tab — removes it and fires `OnFormClosed` exactly once. Floating versus docked is purely the plugin's choice: call `CreateForm` for a floating window, `CreatePanel` for a docked tab; nothing else in your code changes.

```
{ Exactly the live-form example, but docked. Build the same sink and␣
↪controls. }
frm := nil;
if (ui.CreatePanel('Form demo', FSink, frm) <> IQ_OK) or (frm = nil) then
begin FSink := nil; Exit(IQ_OK); end;   // no dock provider (headless)
sink.SetForm(frm);
FForm := frm;                           // retain — release in Deactivate
{ … AddLabel / AddInt / AddButton / AddList exactly as for CreateForm … }
frm.Show;                               // appears as a tab in the right-
↪side dock strip
```

The same docked panel is available from scripting: Python (`iqcad.create_panel(title, handler)` — see *Python Scripting Reference* (page 309)) and PascalScript (`CreatePanel(title)` — see *Pascal Scripting* (page 323)), each mirroring its floating `create_form` / `CreateForm` counterpart.

40.11 Ribbon & menu contributions

Commands typed at the command line are powerful but not *discoverable*. To surface a plugin's commands in the GUI, a plugin **adds ribbon buttons and menu items that run a registered command**. Two `IIqHostUI` methods (API 1.6) do this:

```
function AddRibbonButton(const Group, CommandName, Caption, Hint,␣
↪IconFile: PWideChar;
                         Large: LongBool): TIqResult; cdecl;   // (1.6)
function AddMenuItem(const Submenu, CommandName, Caption: PWideChar):␣
↪TIqResult; cdecl;  // (1.6)
```

Every contribution lands in a **dedicated location** — a single **Plugins** ribbon tab and a single **Plugins** top-level menu. Each plugin gets its own **pane** (on the ribbon) and **submenu** (in the menu), named by the `Group` / `Submenu` argument; pass an empty string to use the plugin's display name. There is no contributing into existing host tabs or menus, so plugins never collide.

- **Command-bound.** `CommandName` is a command you registered through the *command registry* (page 363); clicking the button or menu item runs it through the host's normal command path. There is no separate click callback. The contribution is created even if the command is not yet registered (it resolves by name at click time); clicking an unknown command is a harmless no-op logged to the *debug log* (page 367).
- **Icons** are a PNG **relative to the plugin folder** (the same folder as `plugin.json`); the host loads and scales it into the ribbon image lists. Omit it (empty `IconFile`) for a generic plugin icon; `AddMenuItem` is text-only and ignores icons. `Large` selects a large versus small ribbon button.
- **Auto-cleanup.** Contributions are tagged to the plugin that added them and are removed automatically when the plugin is disabled, reloaded, or IQcad exits. The **Plugins** tab and menu hide again once no

contributions remain.

`AddRibbonButton` / `AddMenuItem` are normally called in `Activate`, *after* registering the command:

```
function TMyPlugin.Activate(const Host: IIqHost): TIqResult; cdecl;
var reg: IIqCommandRegistry; ui: IIqHostUI;
begin
  FHost := Host;
  // 1. Register the command (must succeed).
  if Host.Commands(reg) <> IQ_OK then Exit(IQ_E_FAIL);
  reg.Register(TMyCommand.Create(Host));      // command name 'MYCMD'

  // 2. Make it discoverable in the GUI (tolerate a headless host).
  ui := nil; Host.UI(ui);
  if ui <> nil then
  begin
    ui.AddRibbonButton('Demo', 'MYCMD', 'Hello', 'Say hello', 'icon.png',␣
↪True);
    ui.AddMenuItem('Demo', 'MYCMD', 'Hello');
  end;
  Result := IQ_OK;
end;
```

A headless host (or a build with no contributions provider) returns `IQ_E_FAIL` from these methods; tolerate it — the command itself is still registered and usable from the command line.

40.11.1 The same contributions from scripting

The contribution engine is shared with the scripting hosts, so Python and Pascal scripts add ribbon buttons and menu items too. Scripts have no install folder or id, so their contributions appear under a single **Scripts** group / submenu, and `icon` is an **absolute** path.

Python (see *Python Scripting Reference* (page 309)) — `iqcad.add_ribbon_button(group, command, caption, hint='', icon='', large=True)` and `iqcad.add_menu_item(submenu, command, caption)`.

PascalScript (see *Pascal Scripting* (page 323)) — `AddRibbonButton(group, command, caption, hint, iconfile, large)` and `AddMenuItem(submenu, command, caption)`.

40.12 Writing a plugin in C or C++

Because the ABI is plain C vtables, you can write a plugin in C/C++ (or any language with C FFI). Include `iqcad_sdk.h` and implement the same interfaces as vtable structs.

Where Pascal gives you language interfaces and `IqStr`, in C you build the vtable by hand and supply UTF-16 strings yourself. A minimal command plugin:

```
#include "iqcad_sdk.h"
#include <string.h>

/* ---- a reusable static UTF-16 string object ------------------------ */
typedef struct { const IIqStringVtbl* lpVtbl; const iq_wchar* s; iq_int n;␣
↪} CStr;
```

(continues on next page)

(continued from previous page)

```
static iq_result IQ_CALL str_QI(void* self, const IqGUID* iid, void** o) {
    if (!memcmp(iid, &IID_IIqString, sizeof(IqGUID))) { *o = self; return␣
↪IQ_OK; }
    *o = NULL; return IQ_E_NOTFOUND;
}
static iq_uint IQ_CALL str_AddRef(void* self)  { (void)self; return 1; }
static iq_uint IQ_CALL str_Release(void* self) { (void)self; return 1; }
static iq_int     IQ_CALL str_Length(void* self){ return ((CStr*)self)->n;␣
↪}
static const iq_wchar* IQ_CALL str_Chars(void* self){ return ((CStr*)self)-
↪>s; }
static const IIqStringVtbl g_strVtbl = {
    str_QI, str_AddRef, str_Release, str_Length, str_Chars
};
#define CSTR(VAR, LIT, LEN) static CStr VAR = { &g_strVtbl, u##LIT, LEN }

/* ---- the command ------------------------------------------------------ */
CSTR(g_name,    "CHELLO", 6);
CSTR(g_aliases, "CHI",    3);
CSTR(g_cat,     "Tutorial", 8);
CSTR(g_help,    "C greeting", 10);

typedef struct { const IIqCommandVtbl* lpVtbl; IIqHost* host; } CmdObj;

static iq_result IQ_CALL cmd_QI(void* self, const IqGUID* iid, void** o) {
    if (!memcmp(iid, &IID_IIqCommand, sizeof(IqGUID))) { *o = self; return␣
↪IQ_OK; }
    *o = NULL; return IQ_E_NOTFOUND;
}
static iq_uint IQ_CALL cmd_AddRef(void* self)  { (void)self; return 1; }
static iq_uint IQ_CALL cmd_Release(void* self) { (void)self; return 1; }
static iq_result IQ_CALL cmd_Name   (void* s, IIqString** o){␣
↪*o=(IIqString*)&g_name;    return IQ_OK; }
static iq_result IQ_CALL cmd_Aliases(void* s, IIqString** o){␣
↪*o=(IIqString*)&g_aliases; return IQ_OK; }
static iq_result IQ_CALL cmd_Category(void* s,IIqString** o){␣
↪*o=(IIqString*)&g_cat;     return IQ_OK; }
static iq_result IQ_CALL cmd_Help   (void* s, IIqString** o){␣
↪*o=(IIqString*)&g_help;    return IQ_OK; }
static iq_result IQ_CALL cmd_Invoke (void* self, IIqEnumerator* args) {
    CmdObj* c = (CmdObj*)self; (void)args;
    c->host->lpVtbl->Log(c->host, IQ_LOG_INFO, u"Hello from a C plugin!");
    return IQ_OK;
}
static const IIqCommandVtbl g_cmdVtbl = {
    cmd_QI, cmd_AddRef, cmd_Release,
    cmd_Name, cmd_Aliases, cmd_Category, cmd_Help, cmd_Invoke
};
static CmdObj g_cmd = { &g_cmdVtbl, NULL };

/* ---- the plugin object (Describe/Activate/Deactivate) ---------------- */
/* … same pattern: a struct with const IIqPluginVtbl*, fill the slots,
   Activate() calls host->Commands(host,&reg); reg->Register(reg, &g_cmd)␣
↪… */

IQ_EXPORT iq_result IqcadPluginInit(IIqHost* host, IIqPlugin** out_plugin)
```

(continues on next page)

(continued from previous page)

```
↪{
    g_cmd.host = host;
    *out_plugin = /* address of your IIqPlugin singleton */;
    return IQ_OK;
}
```

Build it as a shared library and install exactly as a Pascal plugin:

```
gcc -shared -fPIC -std=c11 -I/path/to/ICAD/cad/plugins/csdk \
    -o libCHello.so chello.c
```

See the `chello` sample for the complete, compilable version including the plugin object and `QueryInterface` details.

Note

In C you call every method as `obj->lpVtbl->Method(obj, args…)` — the object pointer is always the implicit first argument. Strings are `iq_wchar*` (UTF-16); use `u"…"` literals. Colours use the `IQ_RGB(r,g,b)` macro.

40.13 The bundled samples

The SDK ships eighteen worked samples under `cad/plugins/samples/` — nine in Free Pascal, each with a C twin. They form a learning path from the trivial to the civil:

#	Pascal	C twin	Command (alias)	Demonstrates
1	`hello`	`chello`	`HELLO` (`HI`) / `CHELLO` (`CHI`)	The minimal skeleton: entry point, `IIqPlugin`, one command, `Host.Log`. The Pascal version also subscribes to an event.
2	`geom`	`cgeom`	`DRAWDEMO` (`DD`)	Drawing geometry: `CreateLayer` + `SetActiveLayer`, `AddLine` / `AddCircle` / `AddArc` / `AddRectangle` / `AddHatch`, reading entity handles, `Invalidate`. *Needs API ≥ 1.2.*
3	`entsel`	`centsel`	`RECOLOR` (`RC`)	Selection and entities: iterate `Selection / Entities` with an `IIqEnumerator`, `QueryInterface` each to `IIqEntity`, `SetColor(IqRGB(…))`.
4	`events`	`cevents`	*(none — a listener)*	Events: `Subscribe` an `IIqEventSink`, implement all nine callbacks, release the cookie on `Deactivate`. *Needs API ≥ 1.2.*
5	`civilui`	`ccivilui`	`SURVEYDEMO` (`SD`)	Civil + UI: `IIqHostUI.PromptString` / `MessageBox`, `AddSurveyPoint`, `CreateSurface` (TIN), `ElevationAt`.
6	`dialogdemo`	`cdialogdemo`	`DIALOGDEMO` (`DLG`)	Parameter dialog: `CreateDialog` + `Add*` (choice/float/text/ color/bool) + `Execute` + `Get*`, then draws the shape and shows a rich `MessageBox`. *Needs API ≥ 1.3.*
7	`formdemo`	`cformdemo`	`FORMDEMO` (`FORM`)	Live tool window: `CreateForm` + controls + an `IIqFormSink` button callback updating a list/label live. *Needs API ≥ 1.4.*
8	`dockdemo`	`cdockdemo`	`DOCKDEMO` (`DOCK`)	Docked panel: same as `formdemo` but via `CreatePanel` — lives as a tab in the right-side dock strip. *Needs API ≥ 1.5.*
9	`ribbondemo`	`cribbondemo`	`RIBBONDEMO` (`RBN`)	Adds a ribbon button + menu item (with a PNG icon) that run a command: `AddRibbonButton` / `AddMenuItem` into the **Plugins** tab/menu. *Needs API ≥ 1.6.*

Recommended order: **hello → geom → entsel → events → civilui → dialogdemo → formdemo → dockdemo → ribbondemo**, with the C twin alongside each. `hello` / `chello` are the smallest complete examples; `samples/README.md` has build and install notes per sample.

40.14 API reference

This section lists the public contract. Signatures are Pascal (`uIqSdk.pas`); the C header `iqcad_sdk.h` mirrors each as a vtable slot with an explicit `self` first argument.

40.14.1 Result codes (`TIqResult`)

Constant	Value	Meaning
`IQ_OK`	0	Success.
`IQ_E_FAIL`	-1	Generic failure.
`IQ_E_NOTIMPL`	-2	Method not implemented for this object.
`IQ_E_INVALIDARG`	-3	A bad argument was supplied.
`IQ_E_NOTFOUND`	-4	Requested item does not exist.
`IQ_E_OUTOFBOUNDS`	-5	Index out of range.
`IQ_E_WRONGTHREAD`	-6	Called from a thread other than the main thread.

40.14.2 Core types

```
TIqVersion = record Major, Minor: Word; end;
TIqPoint3D = record X, Y, Z: Double; end;            PIqPoint3D = ^
↪TIqPoint3D;
TIqColor   = LongInt;                                 // packed RGB␣
↪0x00BBGGRR
TIqLogLevel = (illDebug, illInfo, illWarning, illError);
TIqVariantKind = (ivkNull, ivkInt, ivkFloat, ivkBool, ivkString);
TIqVariant = record Kind: TIqVariantKind; Num: Double; Str: IIqString; end;
TIqPluginInfo = record Id: IIqString; Name: IIqString; Version: TIqVersion;
↪ end;
// message-box enums (API 1.3)
TIqMsgButtons = (imbOK, imbOKCancel, imbYesNo, imbYesNoCancel);
TIqMsgIcon    = (imiNone, imiInfo, imiWarning, imiError, imiQuestion);
TIqMsgResult  = (imrOK, imrCancel, imrYes, imrNo);
```

Colour — `TIqColor` is a packed RGB value, `0x00BBGGRR` (red in the low byte, like a Win32 `COLORREF`). It is **not** an ACI palette index. Build it with `IqRGB(r, g, b)` / `IQ_RGB(r, g, b)`.

Variant — `TIqVariant` carries a typed scalar for entity properties. `Num` holds the integer, float, or boolean (0/1) value; `Str` is valid only when `Kind = ivkString`. It is always passed by `const`/`out` reference, never by value.

40.14.3 Runtime helpers (`uIqSdkRtl`)

```
function IqStr(const S: WideString): IIqString;
function IqRGB(R, G, B: Byte): TIqColor;
function IqEnumStrings(const Items: array of WideString): IIqEnumerator;
function IqEnumInterfaces(const Items: array of IInterface): IIqEnumerator;
function IqProfilePoint(AStation, AZ: Double): IIqProfilePoint;
```

These wrap the common cases (a string, a colour, an enumerator). To expose your own variable-length data, derive `TInterfacedObject` and implement `IIqString` or `IIqEnumerator` directly.

40.14.4 Entry point

```
// exported under the name 'IqcadPluginInit'
function IqcadPluginInit(const Host: IIqHost; out Plugin: IIqPlugin):␣
↪TIqResult; cdecl;
```

C: `iq_result IqcadPluginInit(IIqHost* host, IIqPlugin** out_plugin);`

40.14.5 IIqPlugin

The object your `IqcadPluginInit` returns.

```
function Describe(out Info: TIqPluginInfo): TIqResult; cdecl;  // id MUST␣
↪match manifest
function Activate(const Host: IIqHost): TIqResult; cdecl;      // register␣
↪commands/events here
function Deactivate: TIqResult; cdecl;                         //␣
↪unregister/unsubscribe here
```

40.14.6 IIqHost

Your handle to IQcad, passed to `IqcadPluginInit` and `Activate`.

```
function ApiVersion: TIqVersion; cdecl;
function ActiveDrawing(out D: IIqDrawing): TIqResult; cdecl;       // D␣
↪may be nil (no drawing)
function Commands(out R: IIqCommandRegistry): TIqResult; cdecl;
function Events(out S: IIqEventSource): TIqResult; cdecl;
function UI(out U: IIqHostUI): TIqResult; cdecl;
function Log(Level: TIqLogLevel; const Msg: PWideChar): TIqResult; cdecl;
function PostToMainThread(const Callback: IIqRunnable): TIqResult; cdecl;
```

40.14.7 IIqHostUI

Prompts, message boxes, and parameter dialogs. All UI is built and owned by the host (the plugin never touches LCL — see *Parameter dialogs and message boxes* (page 350)).

```
function MessageBox(const Text, Caption: PWideChar; Buttons: TIqMsgButtons;
                    Icon: TIqMsgIcon; out Answer: TIqMsgResult): TIqResult;
↪ cdecl;
function PromptString(const Prompt, Default: PWideChar; out Value:␣
↪IIqString): TIqResult; cdecl;
function PickFile(const Title, Filter: PWideChar; SaveDlg: LongBool; out␣
↪Path: IIqString): TIqResult; cdecl;
function CreateDialog(const Title: PWideChar; out Dlg: IIqDialog):␣
↪TIqResult; cdecl;  // (1.3)
function CreateForm(const Title: PWideChar; const Sink: IIqFormSink; out␣
↪Form: IIqForm): TIqResult; cdecl;  // (1.4)
function CreatePanel(const Title: PWideChar; const Sink: IIqFormSink; out␣
↪Form: IIqForm): TIqResult; cdecl;  // (1.5)
function AddRibbonButton(const Group, CommandName, Caption, Hint,␣
↪IconFile: PWideChar; Large: LongBool): TIqResult; cdecl;  // (1.6)
function AddMenuItem(const Submenu, CommandName, Caption: PWideChar):␣
↪TIqResult; cdecl;  // (1.6)
```

40.14.8 IIqDialog

A host-built **parameter dialog** (API 1.3), obtained from `IIqHostUI.CreateDialog`. You describe fields with the `Add*` methods, call `Execute`, then read the entered values with the typed `Get*` methods. The host builds and owns every widget; your plugin only describes the form, so there is no cross-runtime UI risk.

```
{ builders — Key identifies the field for the getters; Label_ is the␣
↪prompt }
function AddText     (const Key, Label_, Default: PWideChar): TIqResult;␣
↪cdecl;
function AddMultiline(const Key, Label_, Default: PWideChar): TIqResult;␣
↪cdecl;
function AddInt      (const Key, Label_: PWideChar; Default, Min, Max:␣
↪LongInt): TIqResult; cdecl;
function AddFloat    (const Key, Label_: PWideChar; Default, Min, Max:␣
↪Double; Decimals: LongInt): TIqResult; cdecl;
function AddBool     (const Key, Label_: PWideChar; Default: LongBool):␣
↪TIqResult; cdecl;
function AddChoice   (const Key, Label_, Options: PWideChar; DefaultIndex:␣
↪LongInt): TIqResult; cdecl;  // Options TAB-separated
function AddFile     (const Key, Label_, Default, Filter: PWideChar;␣
↪SaveDlg: LongBool): TIqResult; cdecl;
function AddFolder   (const Key, Label_, Default: PWideChar): TIqResult;␣
↪cdecl;
function AddColor    (const Key, Label_: PWideChar; Default: TIqColor):␣
↪TIqResult; cdecl;
{ run modal — OK = True if the user accepted }
function Execute(out OK: LongBool): TIqResult; cdecl;
{ typed getters — valid after Execute }
function GetText  (const Key: PWideChar; out Value: IIqString): TIqResult;␣
↪cdecl;  // text/multiline/file/folder, and the selected choice text
function GetInt   (const Key: PWideChar; out Value: LongInt): TIqResult;␣
↪cdecl;
function GetFloat (const Key: PWideChar; out Value: Double): TIqResult;␣
↪cdecl;
function GetBool  (const Key: PWideChar; out Value: LongBool): TIqResult;␣
↪cdecl;
function GetChoice(const Key: PWideChar; out Index: LongInt): TIqResult;␣
↪cdecl;    // selected index
function GetColor (const Key: PWideChar; out Value: TIqColor): TIqResult;␣
↪cdecl;
```

Semantics: `Min = Max` means an unbounded numeric field; `Decimals` sets float display precision; choice `Options` is one TAB-separated string and `GetChoice` returns the selected index (`GetText` returns the selected option's text). A mismatched-kind getter returns `IQ_E_INVALIDARG`, an unknown key `IQ_E_NOTFOUND`, and a duplicate `Add` key `IQ_E_INVALIDARG`. `Execute` is single-shot — a second call returns the first result. When no dialog provider is available (headless), `Execute` returns `OK = False` and `MessageBox` returns `imrCancel`.

40.14.9 IIqForm

A host-built **live, non-modal tool window** (API 1.4), obtained from `IIqHostUI.CreateForm`. Unlike the modal `IIqDialog`, a form stays open after the call returns: you add controls (each `Add*` builds a real widget **now** and returns a retained *IIqControl* (page 362) handle), `Show` it, then mutate the controls at runtime and react to user input through an *IIqFormSink* (page 363). The host owns the window and every widget — see *Live tool windows* (page 351).

```
{ builders — each creates a widget now and returns its retained handle;
  Key identifies the control for events and FindControl }
function AddButton   (const Key, Caption: PWideChar; out C: IIqControl):␣
↪TIqResult; cdecl;
function AddLabel    (const Key, Caption: PWideChar; out C: IIqControl):␣
↪TIqResult; cdecl;
function AddText     (const Key, Label_, Default: PWideChar; out C:␣
↪IIqControl): TIqResult; cdecl;
function AddMultiline(const Key, Label_, Default: PWideChar; out C:␣
↪IIqControl): TIqResult; cdecl;
function AddInt      (const Key, Label_: PWideChar; Default, Min, Max:␣
↪LongInt; out C: IIqControl): TIqResult; cdecl;
function AddFloat    (const Key, Label_: PWideChar; Default, Min, Max:␣
↪Double; Decimals: LongInt; out C: IIqControl): TIqResult; cdecl;
function AddBool     (const Key, Label_: PWideChar; Default: LongBool; out␣
↪C: IIqControl): TIqResult; cdecl;
function AddChoice   (const Key, Label_, Options: PWideChar; DefaultIndex:␣
↪LongInt; out C: IIqControl): TIqResult; cdecl;  // Options TAB-separated
function AddColor    (const Key, Label_: PWideChar; Default: TIqColor; out␣
↪C: IIqControl): TIqResult; cdecl;
function AddFile     (const Key, Label_, Default, Filter: PWideChar;␣
↪SaveDlg: LongBool; out C: IIqControl): TIqResult; cdecl;
function AddFolder   (const Key, Label_, Default: PWideChar; out C:␣
↪IIqControl): TIqResult; cdecl;
function AddList     (const Key, Label_: PWideChar; out C: IIqControl):␣
↪TIqResult; cdecl;
{ layout grouping (optional) — controls added between Begin/EndRow share a␣
↪row }
function BeginRow: TIqResult; cdecl;
function EndRow: TIqResult; cdecl;
{ lookup }
function FindControl(const Key: PWideChar; out C: IIqControl): TIqResult;␣
↪cdecl;
{ lifecycle }
function Show: TIqResult; cdecl;
function Hide: TIqResult; cdecl;
function Close: TIqResult; cdecl;
function IsVisible(out Visible: LongBool): TIqResult; cdecl;
function SetTitle(const Title: PWideChar): TIqResult; cdecl;
```

Controls are auto-stacked vertically (label + editor per row) in the order added; `BeginRow` / `EndRow` group several controls onto one line. A duplicate `Add` key returns `IQ_E_INVALIDARG`; `FindControl` with an unknown key returns `IQ_E_NOTFOUND`. After the window closes the form handle (and all its control handles) go **inert** — every call returns `IQ_E_FAIL` rather than dereferencing a freed widget.

40.14.10 IIqControl

A retained handle to one live widget (API 1.4), returned by the `IIqForm` builders or `FindControl`. It is a single generic handle whose accessors are **kind-checked**: a getter or setter that does not fit the control's kind returns `IQ_E_INVALIDARG` (e.g. `GetInt` on a label, or a list operation on a button).

```
function Key(out S: IIqString): TIqResult; cdecl;
function GetText  (out Value: IIqString): TIqResult; cdecl;   // text/
↪multiline/file/folder; selected choice text; label/button caption
function SetText  (const Value: PWideChar): TIqResult; cdecl;
function GetInt   (out Value: LongInt): TIqResult; cdecl;
```

(continues on next page)

(continued from previous page)

```
function SetInt   (Value: LongInt): TIqResult; cdecl;
function GetFloat (out Value: Double): TIqResult; cdecl;
function SetFloat (Value: Double): TIqResult; cdecl;
function GetBool  (out Value: LongBool): TIqResult; cdecl;
function SetBool  (Value: LongBool): TIqResult; cdecl;
function GetChoice(out Index: LongInt): TIqResult; cdecl;
function SetChoice(Index: LongInt): TIqResult; cdecl;
function GetColor (out Value: TIqColor): TIqResult; cdecl;
function SetColor (Value: TIqColor): TIqResult; cdecl;
function SetCaption(const Caption: PWideChar): TIqResult; cdecl;  //␣
↪button/label text
function SetEnabled(Enabled: LongBool): TIqResult; cdecl;
function SetVisible(Visible: LongBool): TIqResult; cdecl;
{ list-only ops (other kinds → IQ_E_INVALIDARG) }
function AppendLine (const Text: PWideChar): TIqResult; cdecl;
function Clear: TIqResult; cdecl;
function ItemCount  (out Count: LongInt): TIqResult; cdecl;
function GetSelected(out Index: LongInt): TIqResult; cdecl;
function SetSelected(Index: LongInt): TIqResult; cdecl;
```

Once the form closes, the handle is inert and every method returns `IQ_E_FAIL` — it is always safe to keep (and even call) a stale handle.

40.14.11 IIqFormSink

Implemented by your plugin and handed to `CreateForm` (it may be `nil` for a form that needs no event handling). The host calls it back when the user interacts with the form.

```
{ IIqFormSink — implemented by your plugin }
procedure OnButtonClicked(const Key: PWideChar); cdecl;   // a button with␣
↪that Key was clicked
procedure OnValueChanged(const Key: PWideChar); cdecl;    // an editor's␣
↪value changed; read it via the control
procedure OnFormClosed; cdecl;                            // the window␣
↪closed (fires once)
```

Every callback fires on the **main thread** and is wrapped by a boundary guard in the host, so an exception in your handler cannot escape into IQcad's message loop (identical to the `IIqEventSink` bridge). `OnValueChanged` carries the control key; read the new value through the control handle (`FindControl` + a typed getter).

40.14.12 IIqCommand and IIqCommandRegistry

You implement `IIqCommand`; the host provides `IIqCommandRegistry`.

```
{ IIqCommand — implemented by your plugin }
function Name(out S: IIqString): TIqResult; cdecl;
function Aliases(out S: IIqString): TIqResult; cdecl;     // comma-
↪separated, e.g. 'RC,REC'
function Category(out S: IIqString): TIqResult; cdecl;
function HelpText(out S: IIqString): TIqResult; cdecl;
function Invoke(const Args: IIqEnumerator): TIqResult; cdecl;   // Args␣
↪yields IIqString

{ IIqCommandRegistry — provided by Host.Commands }
```

(continues on next page)

(continued from previous page)

```
function Register(const Cmd: IIqCommand): TIqResult; cdecl;
function Unregister(const Name: PWideChar): TIqResult; cdecl;
function Execute(const Input: PWideChar): TIqResult; cdecl;     // run another command line
```

Aliases are fully supported — typing an alias dispatches the command just like its canonical name.

40.14.13 IIqDrawing

The active drawing. Methods marked *(1.2)* were added in API 1.2.

```
function AddLine(const A, B: TIqPoint3D; out E: IIqEntity): TIqResult; cdecl;
function AddCircle(const C: TIqPoint3D; R: Double; out E: IIqEntity): TIqResult; cdecl;
function AddPolyline(Pts: PIqPoint3D; Count: LongInt; Closed: LongBool; out E: IIqEntity): TIqResult; cdecl;
function AddText(const P: TIqPoint3D; const S: PWideChar; H: Double; out E: IIqEntity): TIqResult; cdecl;
function AddArc(const AStart, AMid, AEnd: TIqPoint3D; out E: IIqEntity): TIqResult; cdecl;        // (1.2)
function AddEllipse(const Center: TIqPoint3D; RadX, RadY, Rotation: Double; out E: IIqEntity): TIqResult; cdecl;
function AddRectangle(const A, B: TIqPoint3D; out E: IIqEntity): TIqResult; cdecl;                // (1.2)
function AddHatch(Pts: PIqPoint3D; Count: LongInt; const Pattern: PWideChar; Angle, Scale: Double; out E: IIqEntity): TIqResult; cdecl;  // (1.2)
function AddTunnelSectionLine(Pts: PIqPoint3D; Count: LongInt; Closed: LongBool; out E: IIqEntity): TIqResult; cdecl;
function Entities(out E: IIqEnumerator): TIqResult; cdecl;       // yields IIqEntity
function Selection(out E: IIqEnumerator): TIqResult; cdecl;      // yields IIqEntity
function FindByHandle(const H: PWideChar; out E: IIqEntity): TIqResult; cdecl;
function Layers(out E: IIqEnumerator): TIqResult; cdecl;         // yields IIqLayer
function ActiveLayer(out S: IIqString): TIqResult; cdecl;
function SetActiveLayer(const Name: PWideChar): TIqResult; cdecl;
function CreateLayer(const Name: PWideChar; Color: TIqColor): TIqResult; cdecl;                   // (1.2)
function GetName(out S: IIqString): TIqResult; cdecl;            // (1.2)
function BeginEdit: TIqResult; cdecl;
function EndEdit: TIqResult; cdecl;
function AddSurveyPoint(const Pos: TIqPoint3D; Number: LongInt; out E: IIqEntity): TIqResult; cdecl;
function Surfaces(out E: IIqEnumerator): TIqResult; cdecl;       // yields IIqSurface
function CreateSurface(const Name: PWideChar; Pts: PIqPoint3D; Count: LongInt; out S: IIqSurface): TIqResult; cdecl;
function FindSurface(const Name: PWideChar; out S: IIqSurface): TIqResult; cdecl;                 // (1.2)
function Alignments(out E: IIqEnumerator): TIqResult; cdecl;     // yields IIqAlignment
function CreateAlignmentFromPolyline(const PolyHandle: PWideChar; Radius:
```

(continues on next page)

(continued from previous page)

```
↪Double; const Name: PWideChar; out A: IIqAlignment): TIqResult; cdecl;
function FindAlignment(const Name: PWideChar; out A: IIqAlignment):␣
↪TIqResult; cdecl;            // (1.2)
function ZoomExtents: TIqResult; cdecl;                          // (1.2)
function Invalidate: TIqResult; cdecl;                           // (1.2)
```

40.14.14 IIqEntity and IIqLayer

```
{ IIqEntity }
function Handle(out S: IIqString): TIqResult; cdecl;
function EntityType(out S: IIqString): TIqResult; cdecl;
function Layer(out S: IIqString): TIqResult; cdecl;
function SetLayer(const Name: PWideChar): TIqResult; cdecl;
function Color: TIqColor; cdecl;
function SetColor(C: TIqColor): TIqResult; cdecl;
function PointCount: LongInt; cdecl;
function GetPoint(Index: LongInt; out P: TIqPoint3D): TIqResult; cdecl;
function SetPoint(Index: LongInt; const P: TIqPoint3D): TIqResult; cdecl;
function GetProp(const Name: PWideChar; out V: TIqVariant): TIqResult;␣
↪cdecl;
function SetProp(const Name: PWideChar; const V: TIqVariant): TIqResult;␣
↪cdecl;
function Delete: TIqResult; cdecl;

{ IIqLayer }
function Name(out S: IIqString): TIqResult; cdecl;
function Color: TIqColor; cdecl;
```

Survey-point properties are accessed through `GetProp` / `SetProp` using the well-known names `PointNumber`, `PointName`, `Northing`, `Easting`, `Elevation`, `RawDescription`, `FullDescription`, `PointGroup`, `LockPoint`, `LabelStyleName` (constants `IQ_PROP_*` in the contract).

40.14.15 Civil: IIqSurface, IIqProfilePoint, IIqAlignment

```
{ IIqSurface }
function Name(out S: IIqString): TIqResult; cdecl;
function Handle(out S: IIqString): TIqResult; cdecl;
function ElevationAt(const X, Y: Double; out Z: Double): TIqResult; cdecl;
function SlopeAt(const X, Y: Double; out SlopePct, SlopeRatio: Double):␣
↪TIqResult; cdecl;
function AddPoint(const P: TIqPoint3D): TIqResult; cdecl;
function ProfileAlong(Pts: PIqPoint3D; Count: LongInt; out Profile:␣
↪IIqEnumerator): TIqResult; cdecl;  // yields IIqProfilePoint
function VolumeTo(const Other: IIqSurface; Cell: Double; out Cut, Fill:␣
↪Double): TIqResult; cdecl;

{ IIqProfilePoint }
function Station: Double; cdecl;
function Z: Double; cdecl;

{ IIqAlignment }
function Name(out S: IIqString): TIqResult; cdecl;
function Handle(out S: IIqString): TIqResult; cdecl;
```

(continues on next page)

(continued from previous page)

```
function Length: Double; cdecl;
function StartStation: Double; cdecl;
function EndStation: Double; cdecl;
function PointAtStation(const S: Double; out P: TIqPoint3D): TIqResult;␣
↪cdecl;
function DirectionAtStation(const S: Double; out Radians: Double):␣
↪TIqResult; cdecl;
function ElevationAtStation(const S: Double; out Z: Double): TIqResult;␣
↪cdecl;
function StationOffsetAt(const X, Y: Double; out Station, Offset: Double):␣
↪TIqResult; cdecl;
function Elements(out E: IIqEnumerator): TIqResult; cdecl;       // yields␣
↪IIqAlignmentElement

{ IIqAlignmentElement }
function Kind(out S: IIqString): TIqResult; cdecl;   // 'Line' | 'Arc' |
↪'Spiral'
function Length: Double; cdecl;
function StartStation: Double; cdecl;
function Radius: Double; cdecl;
```

40.14.16 Events: `IIqEventSource` and `IIqEventSink`

Implement `IIqEventSink`; subscribe via `Host.Events` → `Subscribe`. Releasing the returned cookie unsubscribes.

```
{ IIqEventSource — from Host.Events }
function Subscribe(const Sink: IIqEventSink; out Cookie: IInterface):␣
↪TIqResult; cdecl;

{ IIqEventSink — implemented by your plugin (nine callbacks) }
procedure OnEntityCreated(const E: IIqEntity); cdecl;
procedure OnSelectionChanged(const Sel: IIqEnumerator); cdecl;
procedure OnDrawingOpened(const D: IIqDrawing); cdecl;
procedure OnDrawingSaved(const D: IIqDrawing); cdecl;
procedure OnDrawingClosed(const Name: PWideChar); cdecl;
procedure OnEntityModified(const E: IIqEntity); cdecl;
procedure OnEntityDeleted(const Handle: PWideChar); cdecl;
procedure OnCommandStarted(const Name: PWideChar); cdecl;
procedure OnCommandFinished(const Name: PWideChar; Success: LongBool);␣
↪cdecl;
```

To stop listening, set your stored cookie variable to `nil`. Each callback is wrapped by a boundary guard in the host, so an exception in your handler cannot escape into IQcad.

40.14.17 Strings, enumerators, runnables

```
{ IIqString }
function Length: LongInt; cdecl;
function Chars: PWideChar; cdecl;        // NUL-terminated UTF-16, valid␣
↪for the object's life

{ IIqEnumerator — call MoveNext BEFORE Current }
function MoveNext: LongBool; cdecl;
function Current(out Obj: IInterface): TIqResult; cdecl;   //␣
```

(continues on next page)

(continued from previous page)

```
↪QueryInterface to the concrete type
procedure Reset; cdecl;

{ IIqRunnable — for PostToMainThread }
procedure Run; cdecl;
```

Iterate an enumerator like this:

```
var en: IIqEnumerator; obj: IInterface; ent: IIqEntity;
begin
  if d.Selection(en) = IQ_OK then
    while en.MoveNext do
      if (en.Current(obj) = IQ_OK) and Supports(obj, IIqEntity, ent) then
        ent.SetColor(IqRGB(220, 30, 30));   // recolour each selected␣
↪entity
end;
```

40.15 Debugging plugins

IQcad routes plugin and scripting activity to a **debug log** so you can diagnose load failures and trace behaviour without a debugger.

40.15.1 Enabling the debug log

Set `"debugLog": true` in IQcad's `settings.json`. IQcad then writes `iqcad_debug.log` next to the executable, flushed after every line so entries survive a crash.

40.15.2 What gets logged

- **Plugin lifecycle** (category `Plugin`) — discovery, each load and activation stage, every `FAIL/EXCEPTION` with its reason, and deactivation. If a plugin won't load, this log pinpoints exactly where it stopped.
- **Plugin command register/unregister** (category `Plugin`).
- **Your ``Host.Log`` calls** — every `Host.Log(level, msg)` is written with its level. `illDebug` goes *only* to the log (no command-line clutter); `illInfo` / `illWarning` / `illError` go to the log **and** the command line. This gives you a quiet debug channel plus user-visible messages.
- **Scripting** (category `Script`) — script output and command execution.

40.15.3 Discovering plugin commands

- On load, each registered command is announced on the command line: `Plugin command available: HELLO (aliases: HI)`.
- **`HELP`** ends with a **Loaded plugin commands** section listing every command added by plugins, with aliases and help text, so users can always see which external commands are available.

40.16 Building for other platforms

The same source builds on all three targets; only the output filename and the manifest `entry` key differ:

Platform key	Toolchain	Output name	Example
`x86_64-linux`	lazbuild / fpc, gcc	`lib<Name>.so`	`libMyFirstPlugin.so`
`x86_64-win64`	lazbuild / fpc, MinGW gcc	`<Name>.dll`	`MyFirstPlugin.dll`
`aarch64-darwin`	lazbuild / fpc, clang	`lib<Name>.dylib`	`libMyFirstPlugin.dylib`

You may ship one plugin folder containing all three binaries plus a single `plugin.json` whose `entry` map names each; IQcad loads only the one matching the running platform.

40.17 Versioning and compatibility — summary

- The contract is **API 1.6** (`IQSDK_API_MAJOR` = 1, `IQSDK_API_MINOR` = 6). 1.6 added ribbon & menu contributions (`IIqHostUI.AddRibbonButton` / `AddMenuItem`); 1.5 added in-window dockable panels (`IIqHostUI.CreatePanel`); 1.4 added live non-modal forms (`IIqForm` / `IIqControl` / `IIqFormSink` + `CreateForm`); 1.3 added the parameter dialog (`IIqDialog` / `CreateDialog`) and the rich `MessageBox`; 1.2 added the extra `IIqDrawing` methods and the full nine-event `IIqEventSink`.
- Declare the minimum you need in `apiVersion` — same `major`, and a `min_minor` no higher than the host's.
- Shipped interface layouts are frozen; new functionality arrives as appended methods (minor bump) or new interfaces.
- Until the SDK is declared stable, **rebuild plugins against the current SDK whenever you update IQcad**.

41 The Blade Runner Script Editor

Blade Runner is IQcad's built-in editor for Python and Pascal scripts. It runs *inside* IQcad against the open drawing, so a script can be written, run, corrected and re-run without leaving the application or reloading anything.

Open it with the BLADE command (alias BLADERUNNER). The window is non-modal: leave it open beside the drawing while you work.

The languages themselves are documented in *Python Scripting Reference* (page 309) and *Pascal Scripting* (page 323); this chapter covers the editor.

41.1 Buffers and Files

Open scripts appear as tabs across the top of the editor. Each tab keeps its own caret position, scroll offset, undo history and language; switching tabs swaps the editor content.

Action	Shortcut	Notes
New Python	Ctrl+N	Opens an empty buffer with Python syntax highlighting.
New Pascal	—	The same for Pascal.
Open…	Ctrl+O	The language follows the file extension: `.py` or `.pas`.
Open Recent	—	The ten most recently opened scripts, kept across sessions.
Save	Ctrl+S	
Save As…	Ctrl+Shift+S	
Close	Ctrl+W	Closes the current tab; you are asked about unsaved changes.

41.2 Running a Script

Action	Shortcut	Notes
Run	F5	Runs the whole buffer against the active drawing.
Run Selection	Shift+F5	Runs only the highlighted text — the fastest way to try one call without splitting the file.

Output appears in the pane below the editor: everything the script prints (`iqcad.write_line` and its

Pascal equivalent) is tee'd there **as well as** to IQcad's command line, so nothing is lost if the editor is closed. Runtime errors are reported in the same pane with their message and, where the host provides it, the line number.

Esc interrupts a long-running script — see *Responsiveness and Esc* in the scripting chapters.

Toggle the output pane from **View › Output**; drag the splitter to resize it.

41.3 The Sidebar

An activity bar down the left edge switches the sidebar between four panels. **Ctrl+B** shows and hides the sidebar; the panel you were last on, and the sidebar width, are remembered between sessions.

Panel	Contents
Recent	The recent-files list. Double-click, or select and press **Enter**, to open one.
Snippets / Samples	The sample scripts shipped with IQcad, Python first and then Pascal, each group sorted by name. Open one to use it as a starting point.
Outline	The symbols in the current buffer: `def` and `class` for Python (indented by nesting), procedures, functions and type declarations for Pascal. Click a symbol to jump the caret to its line.
Explorer	A file tree rooted at a folder of your choosing — point it at your script library and open files straight from it.

41.4 Find, Replace and Navigation

Action	Shortcut	Notes
Find	Ctrl+F	Opens the find bar with **Aa** (match case), **W** (whole word) and **.*** (regular expression) toggles, and a live match count.
Replace	Ctrl+H	Adds the replace field, with **Replace** and **All**.
Find Next / Previous	F3 / Shift+F3	
Go to Line	Ctrl+G	

Esc closes the find bar.

41.5 The Command Palette

F1 opens a command palette over the editor. Type any part of a command name — the match is fuzzy, so `rsel` finds *Run Selection* — and press **Enter** to run the top hit. Every editor action is reachable this way, with its shortcut shown beside it.

41.6 Appearance

View › Theme switches between the **dark** theme (the default, a VS-Code-style palette) and a **light** one. The choice applies to the editor, both syntax highlighters and the surrounding panels, and is remembered.

Window position and size, the output-pane height, the sidebar state and the theme are all stored in IQcad's `settings.json` — see *Application Settings* (page 113).

Part X

Tutorials

42 Tutorials

Step-by-step guides for common civil engineering workflows in IQcad. Each tutorial starts from scratch and walks through the complete process.

- *Tutorial 1: Working with Survey Points* (page 374)
  - *Step 1 — Open a New Drawing* (page 374)
  - *Step 2 — Import Points from CSV* (page 374)
  - *Step 3 — View Points in the Point Manager* (page 376)
  - *Step 4 — Create Point Groups* (page 376)
  - *Step 5 — Change Point Styles* (page 377)
  - *Step 6 — Export Points to Excel* (page 377)
  - *Step 7 — Lock Points* (page 377)
  - *Step 8 — Transform Points* (page 378)
- *Tutorial 2: Working with TIN Surfaces* (page 378)
  - *Step 1 — Import Survey Points* (page 378)
  - *Step 2 — Create a Surface* (page 378)
  - *Step 3 — View Contour Lines and Triangulation* (page 379)
  - *Step 4 — Change the Surface Style* (page 379)
  - *Step 5 — Add a Boundary to Clip the Surface* (page 380)
  - *Step 6 — Add Spot Elevations* (page 380)
  - *Step 7 — Decimate a Large Surface* (page 380)
  - *Step 8 — Edit Surface Points* (page 381)
  - *Step 9 — Export to LandXML* (page 382)
- *Tutorial 3: Working with Alignments* (page 382)
  - *Step 1 — Create an Alignment from PIs* (page 382)

  - *Step 2 — Add Curves at PIs* (page 383)
  - *Step 3 — Add Spiral Transitions* (page 383)
  - *Step 4 — View Station Labels and PI Details* (page 384)
  - *Step 5 — Create a Profile View* (page 384)
  - *Step 6 — Add a Design Profile* (page 385)
  - *Step 7 — Create Cross-Sections* (page 386)
  - *Step 8 — View Earthwork Volumes* (page 386)
  - *Step 9 — Import an Alignment from LandXML* (page 387)
- *Tutorial 4: Working with Parcels* (page 387)
  - *Step 1 — Draw a Closed Polyline* (page 387)
  - *Step 2 — Create a Parcel* (page 388)
  - *Step 3 — Set Parcel Name and Number* (page 388)
  - *Step 4 — View Area and Perimeter* (page 388)
  - *Step 5 — Add User Data* (page 389)
  - *Step 6 — Change the Parcel Style* (page 389)
  - *Step 7 — Import Parcels from LandXML* (page 390)

42.1 Tutorial 1: Working with Survey Points

This tutorial covers importing survey data, organising points into groups, applying styles, and performing coordinate transformations.

42.1.1 Step 1 — Open a New Drawing

1. Choose **File > New** or press **Ctrl+N** to create a blank drawing.

2. The drawing opens with a default empty canvas. The coordinate system defaults to metres (metric).

42.1.2 Step 2 — Import Points from CSV

Survey points are typically collected in the field and exported as CSV files. IQcad can import these directly.

1. Open the **Point Manager** (**Civil › Points › Point Manager…**) and click the **Import** button on its toolbar.
2. In the Import dialog, click **[…]** to browse to your CSV file.
3. Select the **Format** from the dropdown. The most common format is **P, N, E, Z, Desc** (Point Number, Northing, Easting, Elevation, Description). If unsure, choose **Auto-detect (by column count)**.
4. Check **Skip first line** if your file has a header row.
5. Optionally enter a **Point group** name (e.g., `Topo`) to assign all imported points to a group automatically.
6. Review the preview at the bottom of the dialog to verify the data looks correct.
7. Click **Import**. The status bar shows how many points were imported.

CSV file format example:

```
1,538345.123,4872341.456,412.500,BM-001
2,538350.200,4872345.100,413.200,EP
3,538360.500,4872350.800,414.100,CL
4,538370.000,4872355.300,413.800,EP
```

Each line contains: PointNumber, Northing, Easting, Elevation, Description. Lines starting with `#` or `;` are treated as comments. Accepted file types: `.csv`, `.txt`, `.pts`.

42.1.3 Step 3 — View Points in the Point Manager

1. Open the **Point Manager** from **Civil > Points > Manager** or the ribbon button.
2. The grid displays all imported points with columns: #, Easting, Northing, Elevation, Raw Desc, Full Desc, and Lock.
3. Click any cell to edit its value directly. Changes apply immediately to the database and the canvas.
4. Use **Zoom Extents** (`Z` then `E` in the command line, or double-click the mouse wheel) to see all points on the canvas.

42.1.4 Step 4 — Create Point Groups

Groups let you organise points by category (e.g., boundary, topo, control).

1. In the Point Manager, **right-click a sheet tab** at the bottom and choose **New Group…**.
2. Enter a group name (e.g., `Boundary`) and click OK. A new tab appears.
3. Switch to the **All Points** tab. Select the rows you want to add to the group (click the first row, hold **Shift** and click the last row for a range, or hold **Ctrl** and click individual rows).
4. Click the **To Group** button on the toolbar. Select the target group from the dialog and click OK.
5. Switch to the group tab to verify the points appear there.

42.1.5 Step 5 — Change Point Styles

Point styles control the marker shape and label appearance for survey points on the canvas.

1. Open the style editor via **Civil > Styles** (Civil Style Manager) and select the **Point Styles** category, or use **Format > Point Style…** (**Ctrl+Shift+D**).
2. Select a point in the drawing, then in the Object Inspector change the **PointStyleName** property to one of the built-in styles: Standard, Benchmark, Control Point, GPS Rover, or Iron Rod.
3. The marker on the canvas updates immediately to reflect the new style.

42.1.6 Step 6 — Export Points to Excel

1. In the Point Manager, switch to the tab you want to export (a specific group tab, or **All Points**).
2. Click the **Export** button on the toolbar.
3. Choose the format: **CSV**, **XLSX** (Excel), or **ODS** (LibreOffice).
4. Choose a save location and click **Save**.

42.1.7 Step 7 — Lock Points

Locking prevents accidental modification of control points or benchmarks.

1. In the Point Manager, select the rows you want to lock.
2. Click the **Lock** button on the toolbar. The Lock column changes to `Yes` for the selected points.
3. Locked points cannot be moved on the canvas. To unlock, select them again and click **Lock** to toggle.

42.1.8 Step 8 — Transform Points

IQcad provides several coordinate transformation tools accessible from the Point Manager toolbar.

1. Select the points you want to transform in the Point Manager grid.

2. Click one of the transform buttons on the toolbar:
   - **Translate** — shift by dE, dN, dZ offsets.
   - **Rotate** — rotate around a base point by a specified angle.
   - **Scale** — scale from a base point by a factor.
   - **Swap E/N** — swap Easting and Northing values.
   - **Helmert** — 2D conformal transformation (translate + rotate + scale) using control point pairs.

3. Enter the required parameters in the dialog and click **OK**. The points are transformed and the canvas updates.

For more advanced transformations (Affine, Polynomial, Rubber-sheet), use the `CSTRANSFORM` or `RUBBERSHEET` commands from the command line.

42.2 Tutorial 2: Working with TIN Surfaces

This tutorial covers creating a terrain surface from survey points, customising its display, adding boundaries, and performing analysis.

42.2.1 Step 1 — Import Survey Points

If you do not already have survey points in the drawing, import them following the steps in Tutorial 1 (Steps 1-2). You need at least 3 non-collinear points to create a TIN surface.

42.2.2 Step 2 — Create a Surface

1. Choose **Civil > Surfaces > Create Surface** from the menu, click the **Create Srf** ribbon button on the Civil tab, or type `CREATESURFACE` (alias `CS`) in the command line.

2. A surface is created with an auto-generated name (`Surface_1`). If survey points exist in the drawing, they are triangulated automatically using the Bowyer-Watson Delaunay algorithm.

3. The surface appears on the canvas showing triangulation edges and contour lines (depending on the active style).

4. Use **Zoom Extents** to see the full surface.

42.2.3 Step 3 — View Contour Lines and Triangulation

1. Select the surface on the canvas by clicking on it.

2. In the Object Inspector, look at the surface properties. The **ShowContours** and **ShowTriangles** flags control what is visible.

3. Toggle **ShowContours** to `True` to display contour lines.

4. Adjust the **MajorContourInterval** and **MinorContourInterval** values to control contour spacing (e.g., 5.0 m major, 1.0 m minor).

42.2.4 Step 4 — Change the Surface Style

1. With the surface selected, change the **StyleName** property in the Object Inspector.

2. Three built-in presets are available:
   - **Standard** — triangles, contours, boundaries, and labels.
   - **Triangles & Contours** — triangles and contours visible.
   - **Contours Only** — only contour lines and labels.

3. To create or edit styles, open the **Surface Style Editor** via **Format > Surface Styles…** or right-click the surface in the Civil Toolspace and choose **Surface Style…**.

42.2.5 Step 5 — Add a Boundary to Clip the Surface

1. Draw a closed polyline around the area you want to keep (use the `PLINE` command, alias `PL`). Close it by typing `C` at the last vertex prompt.
2. Type `ADDSURFACEBOUNDARY` (alias `ASB`) in the command line.
3. Enter the surface name (or press Enter if only one surface exists).
4. Click the boundary polyline.
5. Choose **Outer** to clip everything outside the boundary, or **Hole** to clip everything inside.
6. The TIN is rebuilt with triangles outside (or inside) the boundary removed.

42.2.6 Step 6 — Add Spot Elevations

1. Type `SPOTELEV` (alias `SE`) in the command line, or choose **Civil > Surfaces > Spot Elevation**.
2. Click points on the surface. At each click, the tool queries the TIN to interpolate the elevation and places a label annotation.
3. Right-click or press **Escape** to exit the command.

42.2.7 Step 7 — Decimate a Large Surface

If your surface has a very large number of points (e.g., from LiDAR or photogrammetry), decimation reduces the triangle count while preserving the surface shape.

1. Type `DECIMATESURFACE` (alias `DS`) in the command line.

2. Enter the surface name (auto-selected if only one exists).

3. Enter the **max elevation tolerance** in drawing units (e.g., `0.05` for 5 cm accuracy).

4. The system automatically selects the best algorithm: grid-based for surfaces > 100K points, greedy insertion for smaller surfaces.

5. Contours and boundaries are regenerated automatically after decimation.

42.2.8 Step 8 — Edit Surface Points

1. Use the surface editing commands to modify the TIN interactively:
   - `SURFADDPOINT` (`SAP`) — add a point to the TIN.
   - `SURFDELETEPOINT` (`SDP`) — remove a point.
   - `SURFMOVEPOINT` (`SMP`) — move a point.
   - `SURFMODIFYELEV` (`SME`) — change a point's elevation.
   - `SURFSWAPEDGE` (`SSE`) — swap a TIN edge between triangles.

2. Use `SURFUNDO` to undo the last edit, or `SURFLISTEDITS` (`SLE`) to review the edit history.

42.2.9 Step 9 — Export to LandXML

1. Choose **File > Export LandXML…** or use the **Output** ribbon tab.
2. In the export dialog, check the surfaces you want to include.
3. Choose a file location and click **Export**.
4. The surface is written as a `<Surface>` element with `<Pnts>` and `<Faces>` sub-elements in LandXML format.

42.3 Tutorial 3: Working with Alignments

This tutorial covers creating a horizontal alignment, adding curves and spirals, and generating profile views and cross-sections.

42.3.1 Step 1 — Create an Alignment from PIs

1. Type `ALIGNMENT` (alias `ALG`) in the command line, or choose **Civil > Alignments > Create Alignment**.
2. Click points on the canvas to place each PI (Point of Intersection). A rubber-band preview line follows the cursor as you click.
3. Right-click or press **Enter** to finish. The alignment is created with straight tangent segments only (no curves yet).

Alternatively, convert an existing polyline: type `ALIGNFROMPOLYLINE` (alias `AFP`) and click on a polyline. Its vertices become the alignment PIs.

42.3.2 Step 2 — Add Curves at PIs

1. Open the **PI Table Editor** from the **Civil ribbon > Edit PIs** button, or double-click an alignment in the Civil Toolspace.
2. In the PI data grid, find the interior PI where you want a curve.
3. Enter a **Radius** value (e.g., `200` for a 200 m curve). The radius of `0` means no curve (tangent intersection).
4. Click **Apply** to rebuild the alignment geometry. The canvas updates to show the curved alignment.

42.3.3 Step 3 — Add Spiral Transitions

1. In the PI Table Editor grid, locate the PI with a curve.
2. Enter a value in the **Spiral In** column (e.g., `50` for a 50 m entry spiral length).
3. Select the **Type (In)** from the dropdown: Clothoid L, Clothoid A, Cubic L, or Cubic A.
4. Similarly, enter a **Spiral Out** value and select its type.
5. Click **Apply** to rebuild. The alignment now shows smooth spiral transitions between the tangent and the circular curve.

42.3.4 Step 4 — View Station Labels and PI Details

1. Select the alignment on the canvas.
2. In the Object Inspector, change the **Style Name** to **Design Phase** to see full labels and PI detail tables.
3. Station labels appear at major and minor tick positions along the centreline. PI detail tables display computed curve parameters (radius, arc length, spiral parameter, tangent distance).
4. Tables are placed on alternating sides of the alignment with leader lines and auto-hide when zoomed out.

42.3.5 Step 5 — Create a Profile View

1. Type `CREATEPROFILEVIEW` (alias `CPV`) in the command line, or choose **Civil > Profile > Create Profile View**.
2. If multiple alignments exist, enter the alignment name. If only one exists, it is selected automatically.
3. Enter the surface name to sample (or press **Enter** to skip).
4. Click a point on the canvas to place the profile view.
5. The profile view appears as a graph with station on the horizontal axis and elevation on the vertical axis. If a surface was selected, the existing ground profile is drawn as a line in the view.

42.3.6 Step 6 — Add a Design Profile

1. Type `DESIGNPROFILE` (alias `DP`) in the command line.
2. Click on a profile view in the drawing to select it.
3. Click points inside the profile view to place PVI (Point of Vertical Intersection) points. Each click defines a station and elevation.
4. Right-click or press **Enter** to finish placing PVIs.
5. For each interior PVI, you are prompted for a **vertical curve length**. Enter `0` for a sharp grade break, or a positive value (e.g., `100`) for a smooth parabolic vertical curve.
6. The design profile is drawn in the profile view as a line overlaying the ground profile.

42.3.7 Step 7 — Create Cross-Sections

1. Type `CREATESECTIONSHEET` (alias `CSS`) in the command line, or choose **Civil > Cross-Sections > Create Section Sheet**.
2. Select the alignment to use.
3. Enter surface names to sample (comma-separated for multiple surfaces, e.g., `Surface_1, Design_Surface`).
4. Set the **station interval** (default 20 m) — how often cross- sections are cut along the alignment.
5. Set the **left and right widths** (default 20 m each) — how far each section extends from the centreline.
6. Set the number of **columns and rows** per sheet for the grid layout.
7. Optionally select two surfaces for **cut/fill comparison** (e.g., existing ground vs. design surface).
8. Click an insertion point on the canvas. The section sheet is generated with all cross-sections arranged in a grid layout.

42.3.8 Step 8 — View Earthwork Volumes

1. Type `EARTHWORKVOLUMES` (alias `EWV`) in the command line.
2. Select the section sheet to analyse.
3. The command computes volumes using the **average end area method** and displays a table showing, for each station: cut area, fill area, cut volume, fill volume, cumulative cut, cumulative fill, and net volume.
4. Totals are printed at the bottom of the table.

42.3.9 Step 9 — Import an Alignment from LandXML

1. Choose **File > Import LandXML…** or use the LandXML import button on the Civil ribbon tab.
2. Browse to the `.xml` file and click **Open**.
3. The import dialog shows the contents of the file. Check the alignments you want to import.
4. Click **Import**. The alignment(s) appear on the canvas with their horizontal geometry (tangents, curves, spirals) and vertical profile (if present in the LandXML data).

42.4 Tutorial 4: Working with Parcels

This tutorial covers creating parcel boundaries, attaching metadata, and customising parcel display.

42.4.1 Step 1 — Draw a Closed Polyline

1. Type `PLINE` (alias `PL`) in the command line to start a polyline.
2. Click points on the canvas to define the parcel boundary corners.
3. After placing the last corner, type `C` and press **Enter** to close the polyline (connect the last point back to the first).

42.4.2 Step 2 — Create a Parcel

1. Type `CREATEPARCEL` (alias `PARCEL`) in the command line, or choose **Civil > Parcels > Create Parcel**.
2. When prompted, enter a **parcel name** (e.g., `Lot-1`).
3. Click on the closed polyline you drew in Step 1.
4. The parcel entity is created and appears on the canvas with the default **Standard** style. The source polyline remains in the drawing.

42.4.3 Step 3 — Set Parcel Name and Number

1. Select the parcel on the canvas by clicking on it.
2. In the Object Inspector, set the **ParcelName** property (e.g., `Lot-1` or `Parcel A`).
3. Set the **ParcelNumber** property to a numeric identifier (e.g., `1`).

42.4.4 Step 4 — View Area and Perimeter

1. With the parcel selected, look at the Object Inspector.
2. The **Area** property shows the computed area of the boundary polygon in square drawing units (e.g., square metres).
3. The **Perimeter** property shows the total boundary length.
4. These values are read-only and update automatically if the boundary geometry is edited.

42.4.5 Step 5 — Add User Data

1. With the parcel selected, expand the **UserData** category in the Object Inspector.

2. Add key-value pairs for the parcel metadata. Common fields include:
   - **Owner** — name of the property owner.
   - **Description** — free-text description of the parcel.
   - **Zoning** — zoning classification (e.g., Residential, Commercial).
   - **LandUse** — current land use description.
   - **TaxID** — tax parcel identifier.

3. User data is saved with the drawing and included in LandXML exports.

42.4.6 Step 6 — Change the Parcel Style

1. With the parcel selected, change the **StyleName** property in the Object Inspector.

2. Four built-in styles are available:
   - **Standard** — boundary line only, no fill.
   - **Property** — boundary with semi-transparent fill and bearing/distance labels on each segment.
   - **Easement** — dashed boundary with light fill for restricted zones.
   - **ROW** — heavy boundary with cross-hatch fill for rights-of-way.

3. To create or edit parcel styles, open the **Civil Style Manager** via **Civil > Styles** and select the **Parcel Styles** category, or use **Format > Parcel Styles…**.

42.4.7 Step 7 — Import Parcels from LandXML

1. Choose **File > Import LandXML…** from the menu.
2. Browse to the `.xml` file containing parcel data and click **Open**.
3. In the import dialog, check the parcels you want to import.
4. Click **Import**. The parcels appear on the canvas with their boundary geometry, names, and any associated metadata from the LandXML file.

Part XI

Features and Changelog

43 Features and Changelog

A summary of all features available in IQcad, followed by a changelog of recent releases. This manual is for IQcad 2026.

- *Changelog* (page 393)
- *Drawing & Editing* (page 426)
- *Civil / AEC* (page 426)
- *Styles* (page 427)
- *Layout & Printing* (page 427)
- *File I/O* (page 428)
- *User Interface* (page 428)

43.1 Changelog

43.1.1 2026-11-08

Help

- **Get Support — Help > Get Support…** and the **Get Support** button on the Help ribbon open a form. Subject, what happened, and an email address are required. Name is optional. Up to 8 files, 20 MB altogether: png, jpg, jpeg, webp, pdf, txt, log, csv, xml, dxf, dwg, zip and iqz. Attach the drawing when IQSoft needs the file. **Include version, platform and license** starts ticked and sends the version, the system, the active modules and a masked serial. On success the ticket number is shown, for example `IQ-2026-0001`. IQSoft replies by email. The same form is at https://iqcad.iqsoft.xyz/support/ and https://iqcad.iqsoft.xyz/el/support/. See *Licensing* (page 11).
- **Website in About — Help > About** shows iqcad.iqsoft.xyz. The address opens in the browser.

Manual

- **Object snaps** — the chapter lists the nine snaps: Endpoint, Midpoint, Centre, Node, Quadrant, Intersection, Nearest, Perpendicular and Tangent. Centre and Quadrant apply to a circle or an arc. Node snaps to a Point entity. See *Drawing Tools* (page 41).
- **Point Group Properties** — the chapter follows the dialog: Information, Include and Point List. See *AEC Survey Points* (page 139).
- **Surface Properties** — the chapter follows the Information and Statistics tabs. `SURFACEPROPERTIES` opens the dialog. `SURFACEPROPERTIES TEXT` prints the short report. See *TIN Surfaces* (page 149).

43.1.2 2026-11-06

Files

- **DXF files open in AutoCAD on every version** — exports to R12, R2000, R2004, R2007, R2010, R2013 and R2018 now open in AutoCAD. A wrong code page name, a wrong linetype count and invalid dimension style colors could crash it or make it discard the drawing. The active layout's viewport and materials are now written in the sections AutoCAD expects, and each layout has its sheet viewport.
- **Layers frozen in a viewport** — saved as references to the layer and read back on every DXF version. They were lost on import before.
- **R12 export** — Greek text and layer names are kept, stored in the system code page. R12 names are limited, so they are written in capitals, a space becomes `_` and long names are cut to 31 bytes. R12 has no custom linetypes or linetype scale in this export, so those are left out.
- **Civil 3D surfaces in DWG** — with the "Civil 3D objects" export setting, a surface now opens in Civil 3D with a surface style called `Standard`, which draws 1 m and 5 m contours on layer 0. Its triangles now run counter-clockwise like Civil 3D's own, and the surface has the bounding box Civil 3D stores.
- **Saving a Civil 3D drawing in the debugger** — the check after a save no longer raises a harmless read error on the point group data.
- **DWG R2000 and R2004 export** — files saved in these two versions are no longer refused on opening. Table style colors in R2000 and several file sections in R2004 were written in the layout of newer versions.
- **Layer lineweight in DWG** — layers are now saved with the Default lineweight on every version. They were saved as 0.00 mm before.
- **Tables in DWG** — a drawing with a table is no longer refused on opening, on any DWG version. The table object was only partly written, and its cells had no text. Cell text is now stored in the table itself.
- **Table cell formatting in DWG** — fills, text colors, alignment, text height, border colors and hidden borders are now saved on every DWG version, along with the grid lines of the table. Fills and text colors are read back when the file is opened. Bold, italic, cell margins and border lineweights are not saved yet.
- **Meshes and attributes in DWG R2000** — all vertices of a mesh and all attributes of a block are now kept. Only the first of each could be read back.

- **Greek text in DWG R2000 and R2004** — Greek and other non-Latin text and layer names now survive both saving and opening. Text is saved in the system code page, and anything that page cannot hold is saved as `\U+XXXX`. Opening uses the code page the file declares.

Drawing

- **Block inserts in MOVE, ROTATE and SCALE** — attributes now stay with the block, and ROTATE and SCALE change the block's rotation and scale. MINSERT spacing scales too. ROTATE now also turns text and multiline text, and images and tables turn by the right angle.
- **MOVE, ROTATE, SCALE and ALIGN can be undone** — for every entity type, as one step. MOVE, ROTATE and SCALE were never recorded, so undo skipped them. ALIGN skipped undo when it scaled, or when it rotated entities that store their own angle, such as text and blocks.
- **Block bounding box** — it now includes attribute text in drawings read from a file. It could stop short of text reaching past the block.
- **CAD points in IQZ** — every point entity reopened at 0,0. They now keep their position.
- **Object snap on the entity being drawn** — a polyline, 3D polyline, TS line, multiline or spline in progress is now a snap target for every snap type: its vertices, segment midpoints, nearest and perpendicular points, and where it crosses another object or itself. It also no longer needs another object near the cursor.

Survey and Civil

- **Coordinate transformations are more accurate** — Helmert and Affine now solve around the centroid of the points. At grid coordinates in the millions they lost precision: on a test drawing every point landed about 20 cm (Helmert) or 3 cm (Affine) from the correct solution. Transformations will shift by that much compared with earlier runs.
- **Survey points follow CSTRANSFORM and RUBBERSHEET** — they snapped back to their old position after the command, and the point database kept the old coordinates.
- **Zoom Extents after moving points** — the drawing limits are recalculated after CSTRANSFORM, RUBBERSHEET and any coordinate change or transformation in the Point Manager. Zoom Extents was fitting the old positions.
- **Transformations move every entity correctly** — CSTRANSFORM and RUBBERSHEET now transform surfaces directly, with their triangles, boundaries, breaklines and labels, along with parcels, gradings, culverts, circles, arcs, text, hatches, dimensions, block inserts, tables and images. Text and block rotation and scale follow the transformation, and on a rubber sheet they are measured where each entity sits. Table text height and dimension text height do not scale.
- **What is not transformed** — alignments stay as designed, because a transformation would change their tangents, curves and spirals. Corridors, profile and section views, section sheets, DEM surfaces, point clouds, ACIS solids and section planes stay too, as do locked surfaces and parcels. The command line lists any types it left out.
- **Undo after a transformation** — one undo now restores everything the command changed, including the drawing limits. Lines and polylines return to their exact coordinates.
- **Rubber sheet accuracy** — control points now land exactly on their targets with the linear method. At grid coordinates they missed by about 2 mm.
- **Transformation report** — the `.trpt` file now holds the solution: the model, all parameters with standard deviations, the inverse transformation, residuals, check points and correct RMS values.

Rubber sheet reports list the triangles or the spline weights. Polynomial residuals were those of an affine solve, and now are the polynomial's own.

- **Common points file** — coordinates in a `.cpt` file are plain decimals with 6 places instead of scientific notation. Older files still load.
- **Point Group Properties** — right-click a point group and choose Properties for the Information, Include and Point List tabs. Include selects points by number, elevation, name or description, or all of them, and the list previews the result. Not available for `_All Points`.
- **Add Point Group under Points** — it did nothing, and now creates a new group.
- **Other point group commands** — Show and Hide work on a surface's groups, where Delete is now called Remove from Surface. A group cannot be deleted while it is locked or a surface uses it.
- **Surface Properties** — a new dialog with Information and Statistics tabs. A button beside the style list opens the Surface Style Editor. The description is saved in IQZ and DWG.

Interface

- **Window and panel sizes** — the main window, Civil Toolspace, Properties and command line now keep their sizes between sessions. They were losing 20% at every start on screens below the form's design DPI, so panels that have already shrunk need dragging back once.

43.1.3 2026-11-01

Manual

- **Licensing** — the license manager, and the commands in IQcad Basic, Pro, Civil, Point Cloud and Drone. See *Licensing* (page 11).
- **Check for Updates** — Help checks the signed manifest and downloads one archive of the program folder. Confirming the update closes IQcad. A helper extracts that archive onto the program folder and starts IQcad again. See *Licensing* (page 11).

43.1.4 2026-10-29

Files

- **DXF tables, meshes and styles** — a table saved to DXF is now an `ACAD_TABLE`, with its text height and cell margin in a `TABLESTYLE`. The grid, the cell text and the rotation come back. A polyface mesh is saved as a `MESH` on R2000 and R2018. A multiline style and a multileader style are written and read, including a custom Standard multiline. R12 export is unchanged. See *Exporting and Importing* (page 97).

43.1.5 2026-10-28

Point Cloud

- **Drone Survey DEM from LAS/LAZ** — **Load DEM** also builds a grid from a LAS or LAZ point cloud. Choose ground (class 2), every class, or one class, and keep the lowest or the average Z in each cell. The cell size is stored with the session. See *Drone Survey* (page 249).

Files

- **Esri JSON import** — **File > Import > GeoJSON** also reads an ArcGIS feature-set JSON file, including a download from maps.ktimatologio.gr. The file's `wkid` is the source CRS. A Κτηματολόγιο extract is EPSG:2100, so leave the drawing CRS at 2100 and the Greek Grid metres stay on the grid. Each ring, including holes, is a closed polyline on the `LAYER` attribute. `KAEK` becomes the entity name and `AREA` the description. GeoJSON files still come in from WGS 84, and export still writes GeoJSON. See *Exporting and Importing* (page 97).

43.1.6 2026-10-26

Point Cloud

- **Drone Survey** — **`DRONESURVEY`** (alias `DSV`) opens from the Point Cloud menu and ribbon. Load a GeoTIFF DEM, drape an orthophoto, pick survey points, breaklines and boundaries, sample a regular or low-pass grid, build a TIN and contours, and send them into the drawing as civil points, 3D polylines and a surface. Sent points go into a point group on the surface definition. Named sessions are stored in the IQZ. Orbit, pan, zoom, walk and fly are in the same window.

43.1.7 2026-09-07 — version 1.5.10.385

DWG

- **Native DWG is now the default** — the in-process reader and writer handle R2000, R2004, R2007, R2010, R2013 and R2018 with no external converter and no temporary DXF. LibreDWG and the ODA File Converter stay in the fallback chain for the versions the native engine does not read. Choose the backend in **File > Settings > DWG > Reader Backend**.
- **Save version** — the target version for DWG output is set once, in **File > Settings > DWG > Save Version** (default R2018).
- **Survey export modes** — **File > Settings > DWG > Survey objects** chooses how COGO points and TIN surfaces leave the drawing: as genuine **Civil 3D objects** (`AeccDbPointsNode`, `AeccDbSurfaceTin`, `AeccDbFace`, `AeccDbAlignment`, listed in the Civil 3D Prospector), as **points + surface with labels** (the default: POINT, TEXT and 3DFACE geometry any CAD can read), as **geometry only**, or **not at all**.
- **ACIS write** — REGION, 3DSOLID and BODY are now written as well as read. R2000–R2010 get encrypted SAT; R2013 and R2018 get the SAB prototype section, so a solid opened here and saved again is still the same solid.
- **New encoders** — 2D polylines and their vertices, ATTDEF, polygon meshes, a Standard table style, and proxy headers. 61 object decoders in total; Civil 3D parcels and alignments now decode.
- **SHAPE** — `AcDbShape` entities are read and drawn, with the glyph stroked from the style's shapefile.
- **Verbatim round-trip** — a drawing opened and saved again without being edited is re-emitted byte for byte.

Civil

- **Sites** — a Civil 3D-compatible named topology container. Parcels and grading groups always belong to a site (`Site 1` by default); alignments may belong to one or stay in the top-level collection. Right-click the Prospector's **Sites** node for **Create Site…**, or an alignment for **Move to Site…**.

- **Alignment defaults** — a new alignment is named `Alignment_1`, `Alignment_2`, … (one past the highest such number in the drawing) by `ALIGNMENT`, `ALIGNFROMPOLYLINE` and the PI editor's **New** button. The default PI layer (`C-ROAD-{Name}-PI`) is now magenta and the offset-axis layer (`C-ROAD-{Name}-OAX`) lilac (210, 170, 255) instead of yellow, and the built-in **Standard** alignment style shows the PI tangent lines.
- **Import points from another drawing** — the Point Manager toolbar gains **Import from Drawing**: pick an `.iqz` or `.ixml` file, tick the point groups to bring in (or **_All Points** for everything), and they are merged into the current drawing. Only the survey database section of the file is read, so large drawings open quickly. Clashing point numbers are reassigned above both drawings and reported.
- **Spot Elevation Dot point style** — a new built-in point style: a small dot marker with the elevation value as the only label. Older drawings gain it on load next to the other presets.
- **Editable civil object layers** — **Civil › Civil Settings… › Layers** gains a grid of every civil object component with its layer group, suffix, resulting layer pattern and colour. Rename a layer by typing in the Group or Suffix cell, click the swatch to pick a colour, or **Reset Layers** to return to the built-in table. Only the overrides are stored in the drawing, so untouched components keep following the defaults.
- **Hide and restore TIN triangles** — `SURFHIDETRI` (`SHT`) hides triangles by clicking inside them; `SURFSHOWTRI` (`SWT`) brings them all back. Hidden faces are kept in the surface edit log, not deleted, and surfaces read from a Civil 3D DWG arrive with their hidden faces already flagged.
- **Alignment report sketches** — **`ALIGNMENTREPORT`** (`ALR`) now opens with a plan sketch of the alignment: tangents, numbered PIs, element boundaries with their stations, and lines, arcs and spirals in distinct colours. An alignment with a design profile also gets a profile sketch with grade lines, PVIs, BVC/EVC marks and high/low points. The PVI table gains grade in, grade out and algebraic difference columns, and a new **Vertical Curve Data** section lists BVC/EVC, K value, external distance and the high or low point of every vertical curve.

Plotting

- **Profile views plot to PDF** — a layout viewport that shows a profile view now prints it: grid, datum, sampled and design profiles, PVI markers and labels, axis labels, the strip stack, title and border, at the same size and position as on screen. Before, the viewport plotted only the profile's outline.
- **Alignments plot through their style** — a layout exported to PDF now renders an alignment exactly as the layout viewport shows it: centerline, offset axis, station ticks and labels, direction arrows, PI markers, PI lines and the PI detail tables, each following the alignment style and the alignment's component layers, with the plot style table applied. Previously only the bare centerline was written, whatever the style said.
- **PI detail tables on layouts** — the tables were drawn in white, which vanished against the white sheet of a layout viewport. They now take the PI layer colour, or the style's PI marker colour, like the markers and PI lines, so they show on layouts and on paper.

Documentation

- New chapters: **Blocks and Attributes**, **Layouts, Viewports and Printing**, **DWG Compatibility**, **Application Settings**, **Sites**, and **The Blade Runner Script Editor**.
- The DWG export and import sections were rewritten — they still described the external-converter flow that the native engine replaced.
- The **Layers** chapter now describes the twelve-column Layer Manager (On, Frozen, Locked, Transparency, Plot Style, Plot, Description) and the ribbon layer dropdown with its state icons.

43.1.8 2026-09-02

Selection

- **Overlapping objects** — when several objects sit under the pick box (corridor linework on an alignment, stacked polylines, …) a gold two-square **badge** appears at the cursor. Click opens a pick list (type, name, layer); hover a row to preview, click the row or press Enter to select, Esc to cancel. Clicking the same spot again, or **Alt+click**, still cycles to the next object without starting a move. **Ctrl+drag** still pans.
- **SELECTIONCYCLING** — new system variable (`0` = off, `1` = badge only, `2` = badge and list, the default). Toggle with **Ctrl+Shift+W** (a negative value remembers the last on-mode). **Ctrl+W** still closes the drawing.

Culverts

- **Culvert Report** — **`CULVERTREPORT`** (`CULVRPT`) writes a PDF of a selected culvert: identity, plan sketch with numbered stake-out points, cross and long sections, a volumes table, and an X/Y/Z coordinates table. **Civil → Culverts → Culvert Report** on the menu and ribbon.

Files

- **Recent files** — saving a new untitled drawing with **Save** or **Save As** now adds the file to the recent-files list.

Corridors

- **Auto-rebuild on by default** — new corridors rebuild automatically when the alignment or assembly changes (`CORRIDORAUTO` still toggles it). Previously the default was off.

43.1.9 2026-08-24

Draw

- **WIPEOUT** — (alias `WO`) new command that masks an area of the drawing with a polygon filled in the background colour, hiding the geometry beneath it without erasing it. *Draw → Wipeout* on the menu and the **Wipeout** button on the *Draw* ribbon tab under **Shapes**.

  Three modes, matching AutoCAD:

  - pick the polygon corners and right-click to close (the default);
  - `Polyline` converts a pre-selected closed polyline, optionally erasing it. A polyline with arc (bulged) segments is refused, because a wipeout boundary is straight-sided and the mask would not cover the same area;
  - `Frames` turns the outline of every wipeout in the drawing on or off. This is drawing-wide, stored in the new `WIPEOUTFRAME` system variable (`0` = off, the default).
- **WIPEOUT import** — wipeouts now read from **DWG** as well as DXF, on all six supported versions. The clip boundary is transformed out of image space into world coordinates, so a wipeout authored in AutoCAD lands at the right place and size.
- **TOLERANCE import** — geometric tolerance frames (GD&T feature control frames) now read from **DWG** as well as DXF, on all six supported versions, and are drawn as a proper boxed frame: one box per cell, divided on the `%%v` separators.

Previously a tolerance was imported as a plain text object holding the raw string, so the drawing showed literal gibberish such as `{\Fgdt;j}%%v{\Fgdt;n}0.05` instead of a frame.

AutoCAD writes the GD&T symbols as font escapes into `gdt.shx`, which IQCAD cannot read. The 20 symbols are mapped to their Unicode equivalents for display — position, diameter, perpendicularity, the material-condition circles and so on. A symbol with no Unicode equivalent shows its code letter, so nothing disappears silently. The original string is kept untouched and is what gets written back out, so a round-trip through IQCAD loses nothing. Tolerances also save to and load from `.iqz` drawings.

- **Layer on/off, frozen, locked and plot state on import** — layers now come in with the state the file actually records. Previously every layer loaded switched **on**, whatever the drawing said, so a drawing whose hidden layers were part of how it reads — construction lines, an underlay, a superseded revision — opened with everything visible at once.

  Applies to **DWG** (all six supported versions) and **DXF**. Two related faults were fixed at the same time:

  – in DXF, a layer that was switched off *and* carried a true colour came in visible, because the off state was only checked when no true colour was present;

  – the **do not plot** flag was neither read nor written, so it was lost both on import and on export.

  Layer state already survived saving to `.iqz`; it now survives a DXF export as well. Exporting to **DWG** still does not carry layer state.

- **Circles drawn on a tilted plane** — a circle whose plane is not the drawing's XY plane (one drawn in a rotated UCS, or on the side of a 3D object) now lands where AutoCAD puts it, and is drawn standing in its own plane instead of lying flat. Previously its position was read as though its local coordinates were world coordinates, which could place it hundreds of units away.

  The plane survives export to DXF and saving to `.iqz`. Text on a tilted plane is not covered yet.

- **Mirrored ellipses placed correctly** — an ellipse that a drawing stores mirrored was moved to the wrong side of the drawing on DXF import. In one real file this put it about 28,000 units from the identical ellipse directly above it. DWG and DXF import now agree on these.

- **Mirrored arcs from DWG** — arcs stored mirrored came in unmirrored from **DWG** files, while the same drawing read from DXF was correct. Both now agree. Arcs standing on a tilted plane are still not supported.

- **Hatch transparency on import** — a hatch read from **DWG** or **DXF** now keeps the transparency it was given, instead of coming in fully opaque. Exporting to DXF writes it back, including a fill that lives on a polyline rather than on a hatch object of its own.

  A hatch whose transparency is set to *ByLayer* — which is AutoCAD's default — now follows its layer, and keeps following it: changing the layer's transparency updates the hatch, rather than the value being fixed at the moment the drawing was opened. *ByBlock* follows the layer too.

  Layer transparency itself was being read and written on the wrong scale: AutoCAD's 50 % came in as 45 %, and 50 % went out as 55 %. Values read from DXF files written by **older IQCAD versions** will therefore shift slightly; values from AutoCAD are now correct.

  Exporting to **DWG** does not yet carry transparency.

43.1.10 2026-08-22

Linux

- **Wayland startup crash fixed** — on a Wayland desktop the program died while building the main window, inside the Qt binding's X11 display lookup (the OpenGL canvas needs it and the binding dereferences a null pointer when no X11 interface exists). IQcad now selects the `xcb` platform plugin itself when the session is Wayland, an X display is reachable and `QT_QPA_PLATFORM` is not already set. An explicit `QT_QPA_PLATFORM` is respected as before.

43.1.11 2026-08-07

Super Tools

New **Super Tools** menu and ribbon tab for Express/CadTools-style drawing helpers. Commands are grouped under **Cleanup** and **Polylines** split buttons (and matching menu submenus). Full chapter: **Super Tools** in the user manual.

- **FLATTEN moved** — still `FLATTEN` on the command line, but the UI entry is now **Super Tools → Cleanup** (no longer under Modify menu / Modify ribbon "More").
- **PLSIMPLIFY** — (alias `PLSIM`) Douglas–Peucker vertex reduction on selected 2D and 3D polylines. Prompts for a deviation tolerance (default `0.01`, remembered for the session). 2D uses XY deviation; 3D uses true 3D. Arc bulges densify then clear; spline-fit is cleared. Undoable. Ribbon / menu: **Super Tools → Polylines**.
- **PLGETZ** — (alias `PGZ`) sample elevations from a **surface** or **point cloud** along selected paths. Supports **lines** and **multi-select**. Sample mode **Vertices** (default) / Interval / Both; **Vertex interval** (default `1`) is asked only when Interval or Both is selected. Missing samples set **Z = 0**. **2D polylines are replaced** by 3D polylines; 3D polylines and lines (vertices mode) update Z in place; densified lines become 3D polylines. Undoable. **Super Tools → Polylines → Get Z…**.
- **BESTFITLINE** — (alias `BFL`) best-fit **3D line** (PCA) through vertices of the multi-selection (points, lines, polylines). Creates a new LINE spanning the point set. Undoable. **Super Tools → Polylines → Best Fit Line…**.

43.1.12 2026-08-05

User interface

- **Layer dropdown state icons** — every row of the ribbon **Layer** dropdown (*Home* tab) now shows a bulb (On/Off), a sun/snowflake (thawed/frozen) and a padlock (unlocked/locked), as in the AutoCAD layer control. Click an icon to toggle the state; the list stays open so several layers can be changed in one visit. Freezing the current layer is refused with a status-bar message. The Layer Manager, if open, updates at once.

Drawing performance & engine indexes

Large drawings (many surfaces, alignments, polylines, point clouds, and civil entities) spend less time on full-document scans during pick, OSNAP, prospector rebuilds, civil commands, and export.

- **Document index** — the canvas keeps non-owning type side-lists and name maps (`TDocumentIndex`): surfaces, alignments, corridors, point clouds, parcels, gradings, survey points, culverts, XLines, CAD points, polylines, profile views, section sheets, block inserts, plus handle and civil name maps for O(log N) resolve.

- **Civil Prospector** — tree rebuild, style grids, and surface naming walk the type lists instead of every entity in the drawing.
- **Commands & dialogs** — profile views, section sheets, earthwork, LandXML export, PC sections, grading cleanup, corridor spawn cleanup, alignment/volume reports, surface style apply, corridor dialogs, and block insert count/rename use the lists or `Find*ByName` / `FindPointCloudByHandleInt`.
- **TRIM ALL** — cutting-edge candidates come from a **spatial query** around the object being trimmed (padded bounding box); EXTEND ALL still considers the full document so distant boundaries remain valid.
- **OSNAP** — geometry helpers and the multi-tier search path live in dedicated units (`UCADOsnapMath`, `UCADOsnapEngine`); aperture queries use the spatial index and always include infinite XLines.
- **Surfaces** — sequential elevation query prep and cheaper tombstone compaction on TIN local edits; band-legend stacking uses `SurfaceList`.
- **``QueryWorldRegion``** — public canvas API for world-AABB spatial candidates (used by TRIM and proximity picks).

No user-facing command names or file formats change; behaviour of pick/OSNAP/trim is intended to match previous results with less work on large files.

- **CAD primitives lists** — `LineList` / `CircleList` / `ArcList` for GIS export and line-based picks without scanning civil-heavy drawings; point-cloud deviation-from-surface colouring resolves the target surface via `FindSurfaceByName` (hot path on draw).
- **Tunnel and PC section lists** — `TunnelSectionLineList` and `PointCloudSectionList`; PDF tunnel deviation overlays and PC section auto-names no longer walk the full object list.
- **Layout viewports** — `ViewportList` speeds activate-by-click, plot viewport passes, DXF layout staging, and IQML clip-entity resolve on paper-space layouts.
- **Plot / PDF** — elevation-band legend surface lookup uses `SurfaceList`; layout PDF no longer rebuilds the viewport frozen-layer set from every model entity (uses the viewport's own list).
- **Load / inspector** — IQML dependency reconnect and legacy profile migration resolve alignments and surfaces by name map; DXF insert bbox update walks `InsertList` only.
- **Index rebuild** — rebuilding type side-lists after bulk load is linear in entity count (no per-add duplicate scan); grading resolve is O(log N) via a name map.
- **Surface SURFUNDO memory** — undoing surface edits frees the mesh snapshot storage after restore (large TINs keep less peak memory).
- **Layout PDF** — paper-space non-viewport entities and viewports are rendered in separate passes (viewports via `ViewportList`).
- **Survey point sync** — `SyncSurveyPointsToCanvas` and `UpdateSurveyPointsFromDB` walk `SurveyPointList` only (not the full document) when building existing-number sets and pushing DB fields.
- **Window / crossing selection** — `SelectObjectsInRect` uses `QueryWorldRegion` so large drawings only test entities near the drag rectangle (same locked/hidden filters as before).
- **Layout viewport paint** — model content inside paper-space viewports is drawn from a spatial candidate set for the viewport's model window (matches the main 2D `CollectVisible` approach).

- **Layout / window PDF** — viewport and model-window plot passes query spatial candidates (rotation uses a diagonal pad so corners stay covered).
- **3D visible set** — `CollectVisibleObjects` in 3D mode prefilters with an expanded spatial XY query, then frustum-tests candidates (falls back to a full frustum walk when nearly the whole drawing is in view).
- **IQML reconnect** — dependency rewire after load walks `ProfileViewList` / `SectionViewList` / `SectionSheetList` / `CorridorList` / `GradingList` only.
- **3D hidden-line PDF** — mesh and edge export use `Mesh3DList` / `Edges3DList` instead of scanning every entity.
- **``SectionViewList``** — new type side-list for section views.
- **``TextList``** — `TCADText` side-list (SVG export and style audits).
- **Scene extents cache** — one pass for `IncludeInZoomExtents` AABB; shared by `ZoomExtents`, layout viewport fit, DXF layout staging, drawing statistics, SVG viewBox, and 3D near/far (`TryGetZoomExtents`).
- **Draw order** — bring/send selection uses an O(N) partition rebuild instead of O(S·N) `IndexOf` + `Move`.
- **Handle counter** — `SyncHandleCounter` uses max handle tracked during handle-map rebuild/add (no second full scan).
- **SVG export** — writes from type lists (points/lines/circles/arcs/ polylines/text).
- **DXF paper space** — viewports written from `ViewportList`.
- **Entity name map** — `FindByName` rebuilds from type side-lists (named polylines/civil/etc.), not every mesh face.
- **3D billboards** — shaded-mode billboard groups use the per-paint visible set (spatial/frustum culled) instead of the full object list.
- **Point-style refresh** — `RefreshObjectPointDefaults` pushes to CAD primitive type lists only (not faces/meshes).
- **Bulk bbox / REGEN** — `EndLoad` and `RegenAll` update bounding boxes (and caches on REGEN) in parallel when entity count ≥ 2000 (`UParallel` / `mtprocs`).
- **Print zone extents** — template drawing-zone fit uses `TryGetZoomExtents` instead of a full object walk.
- **Layer membership** — each `TLayer` tracks entities assigned to it; `RemoveLayer` reassigns only those members (not the full drawing); drawing statistics layer counts use membership size.
- **Layer name lookup** — `FindLayer` uses a sorted name map (rebuilt when layer names change via `LayerNameEpoch`).
- **SURFCLEAREDITS memory** — clearing the surface edit log also frees the SURFUNDO mesh snapshot (peak TIN memory after commit).
- **Block name map** — `FindBlock` / insert resolve use a sorted name map (O(log N) instead of scanning every block definition).
- **DXF export handles** — seed from the canvas handle counter (no full model-space handle scan); block definitions still scanned.

- **EndLoad layer membership** — re-registers entities on their layers after bulk import so `RemoveLayer` membership stays correct.
- **SelectAll** — adds only entities on unlocked/renderable layers via membership lists (skips locked/frozen layers wholesale).
- **Purge / AUDIT** — used-name collection and audit walk layer membership + type lists instead of a flat full-object style filter.
- **EXTEND ALL** — boundary candidates from line/circle/arc/polyline/ xline type lists (not every 3DFACE in the drawing).
- **Drawing statistics by type** — tallies via layer membership.

43.1.13 2026-08-03

Parcel vertex names

- **Survey-style corner names** — every parcel corner now carries a drawing-wide vertex name (prefix + running number, e.g. `A1`, `A2`, …); the sequence continues across every parcel in the drawing, and a corner shared by two adjacent parcels always carries the same name on both sides.
- **Automatic on creation** — `CREATEPARCEL` names every new corner; older drawings keep their corners unnamed until repaired with `PARCELCHECK`.
- **Editable per vertex** — the Object Inspector's **Geometry › Vertex** row for a parcel now shows **Index / Name / X / Y / Bulge**. Renaming a shared corner updates the neighbouring parcel live; an empty name is rejected; a name already used elsewhere gets a command-line warning instead of a blocking dialog.
- **Grip-drag behaviour** — pulling a shared corner apart leaves the name with the vertex that stayed put and issues the moved corner a fresh number; dragging a corner onto another parcel's named corner adopts that name.
- **Labels, tables and reports** — a new **Vertex Labels** group on the parcel style's Labels tab (**Vertex Names Visible** toggle + a civil label style picker; new built-in **Parcel Vertex** label style) draws the names on screen and in PDF plots; the built-in **Property** parcel style shows them by default. `PARCELTABLE` and the `PARCELREPORT` HTML report now list corner names instead of a bare `1..N` index, falling back to the index when unnamed.
- **``PARCELCHECK`` becomes a check-and-fix hub** — reworked with **Report** (read-only listing of duplicate vertices, unnamed vertices, mismatched shared corners, and duplicate names), **Fix** (repairs all of the above; offers to set the drawing's vertex prefix), and **reNumber** (confirmation-guarded full drawing-wide re-sequence). The previous duplicate-vertex cleanup is now part of Fix.
- Vertex names and the per-drawing prefix round-trip through `.iqz` saves and DXF export/import.

43.1.14 2026-08-02

Point cloud Elevation Tracker

- **``PCELEVTRACK``** — a point cloud counterpart to the surface Elevation Tracker: toggle it from the Point Cloud ribbon tab or menu (or the command line) for a live readout of the nearest point in every visible point cloud under the cursor, sharing the same overlay box as the surface tracker. No averaging or interpolation — the reading is always a real measured point.

Elevation band picker

- **Generate Bands from Surface** now opens a small dialog instead of silently guessing a surface: a **Surface** dropdown lists every surface in the drawing (preselected to one already using the style being edited), with **Min elevation / Max elevation** boxes pre-filled from that surface's actual range and editable to an exact figure before generating.

Parcel styles

- **Parcel Style Editor** — boundary, fill and label settings for parcels are now organised in a dedicated style editor (**Format › Parcel Styles…** or the Civil Style Manager), matching the surface and alignment style editors; styles persist in the drawing (`.iqz`).
- Four built-in presets — **Standard**, **Property**, **Easement**, **ROW** — plus **Empty**, selectable via the parcel's **StyleName** property.

43.1.15 2026-08-01

Command-line suggestion popup

- **AutoCAD-style autocomplete** — typing at an idle command prompt now opens a popup listing matching commands (name, aliases, description), ranked exact match first, then prefix, then substring. Navigate with **Up/Down**, run with **Enter** or a click, fill without running with **Tab**, dismiss with **Esc**. Always on, no setting required; it steps aside during an active tool's own sub-prompts.

43.1.16 2026-07-31

Drawing Properties

- **Drawing statistics report** — a new **File › Drawing Utilities › Properties…** dialog presents a read-only, collapsible tree of what the current drawing contains: entity counts by type; layers with their per-layer entity count and on/off, frozen and locked state, colour and linetype; block definitions with their reference counts; text, dimension, multiline and point styles and linetypes; a civil breakdown (surfaces with TIN point/triangle counts, DEM surfaces, alignments, corridors, gradings, culverts); and a model summary (totals, extents, units, file info).
- **Copy / Save** — the report can be copied to the clipboard or saved to a text file for QA and hand-over.
- **Command line** — the same report prints to the command line with the `DWGPROPS` command (alias `DRAWINGPROPS`).

Retained geometry display mode (experimental)

- **Faster pan/zoom on very large drawings** — an optional display mode batches static wireframe geometry into GPU buffers and projects it on the GPU, so panning and zooming a drawing with hundreds of thousands to millions of entities (for example an exploded surface with 1M+ 3D faces) no longer re-projects every vertex on the CPU each frame.
- **Culling + level of detail** — only the visible parts are drawn when zoomed in, and a point-cloud-style level of detail draws far-away geometry as subsampled points, so the view stays responsive at any zoom.
- **Commands** — `RETAINEDGEOMETRY ON/OFF` (alias `RETAINED`) toggles it, `RETLOD` tunes the level-of-detail look, and `RETAINEDSTATS` prints diagnostics. **Off by default**; it does not change normal rendering and is intended for reviewing very large drawings.

43.1.17 2026-07-23

Corridor section styles (Corridor v1.3 Phase 4)

- **Named corridor section styles** — a per-drawing style library (built-ins **Standard** and **Monochrome**) now drives the appearance of both model-space section views (`TAECC_SECTIONVIEW`) and generated section sheets (`TAECC_SECTIONSHEET`); a view/sheet references a style by name, defaulting to **Standard** when blank or missing.
- **Material fills** — each course shape resolves solid or hatch fill through a per-material override list (pattern name, scale, angle, colour — e.g. `ANSI31`), falling back to the material's colour from the drawing's material library, then to a per-row-kind default.
- **Per-row-kind outlines + silhouette** — independent outline colours for Lane/Shoulder/Median/Curb/Sidewalk/Slope/Generic course shapes, plus a dedicated pen for the corridor's outer silhouette.
- **Annotations** — four style-toggled additions: width/slope labels (`W=… S=…%`) on top-surface segments; point marks + names; a per-station area block (cut/fill/excavation + per-material areas); a material legend (per view, or one per sheet with the new per-sheet option).
- **Section View Styles editor** — new category in the Civil Style Manager: style list with New/Duplicate/Rename/Delete (**Standard** protected), and View/Ground/Corridor/Annotations/Preview tabs, the last rendering a live sample through the real section renderer.
- **Monochrome** — a plot-ready built-in: black pens, white background, and hatch fills for all seven default materials.
- Styles persist in the drawing (IQML); no legacy migration is required (assemblies and corridor sections were not previously released).

Conditional rows, ditch and benching (Corridor v1.3 Phase 5)

- **Per-row Cut/Fill condition** — a new **Cond** grid column (Always/Cut/Fill); a conditional row contributes geometry, area, and section shapes only at stations where the corridor resolver decides that side is actually in the matching state, probed once at the daylight hinge — the losing branch contributes nothing anywhere. The editor preview draws both branches schematically (dashed orange for cut, dashed green for fill) so a conditional design can be reviewed without building a corridor.
- **Ditch row kind** — a new trapezoidal **Ditch** row (**+Ditch** button), Depth/BottomWidth (0 = V-ditch)/ForeSlope in the grid, BackSlope in the row params dialog; authored Cut-only by default. It springs from the excavation edge, its rising backslope becomes the cut daylight's starting slope, and its invert extends the excavation-to-formation envelope.
- **Benching** — Slope rows gain per-branch MaxHeight/BenchWidth/ BenchSlope (row params dialog, double-click a P column); tall daylights break into outward-draining lifts (named `DL.B1`, `DL.B1E`, …) instead of one continuous face; a 20-lift cap flags the station **Ungraded**. The preview shows one schematic bench jog per enabled branch.
- **Excavation-edge daylight hinge (bug fix)** — both the cut and fill daylight now spring from the true end of excavation (the outer bottom corner of the outermost course stack) instead of the shoulder's finished top edge, closing a gap where a widened subbase or taper foot previously stuck out past the daylight line; a connector link closes the Top chain down the stack face. Sides without course stacks fall back to the previous anchor unchanged.
- **Upgraded standard templates** — all six shipped templates now carry a Cut-only ditch (0.5 m/0.5 m/1:2/1:1.5) and benching (5 m lifts, 1.5 m benches, 4%) on both branches before the closing Slope

row. On Urban-2x3.5+walks and Urban-Boulevard the ditch ends up behind the sidewalk (no shoulder to place it after) — unusual practice, worth reviewing per project.

- Condition, ditch, and bench parameters persist in `.iqal`/IQML as an additive extension of the existing v2 schema.

43.1.18 2026-07-22

Pavement structures (Corridor v1.3 Phase 2)

- **Multi-course pavement stacks** — Lane, Shoulder, and Sidewalk rows now carry an ordered stack of **courses** instead of one `PaveDepth`. Each course is a parametric shape (material, width, thickness, per-side edge treatment) attached with a 9-point anchor (`TL/TC/TR/ML/MC/MR/BL/BC/BR`) to an earlier point — by default the row's top link or the previous course, or explicitly to any other structure in the assembly (cross-row anchoring, e.g. `LaneL.Subbase.BR`). Tapered edges produce widening course feet; wider/centered courses use `SelfPt=TC/TargetPt=BC`.
- **Drawing-level material library** — every drawing now has its own material library (name + colour), opened via the new **Materials…** button in the Assembly Editor, seeded with seven defaults (Asphalt Wearing, Asphalt Binder, Asphalt Base, Base (crushed stone), Subbase, Concrete, Drainage gravel). Each course references a material by name.
- **Per-material quantities + excavation-to-formation** — `BUILDCORRIDOR` now prints, after the cut/fill/net line, an **Excavation (to formation)** volume (ground above the datum surface) and a per-material quantity table integrated along the whole corridor.
- **Datum at the true bottom of the stack** — the datum surface now follows the outer bottom corners of a row's last course (the deepest course by template convention), folding in the Phase-1 M1 fix so Curb/Median/Generic rows correctly advance the datum chain and chain starts always resolve a proper bottom corner.
- **``.iqal``/``.iqml`` XML v2** — the assembly library and drawing formats gain a `<Materials>` block and per-course `<Course>` elements (root `version="2"`). Files saved as v1 (including a v1 `<AssemblyLibrary>` embedded in an older `.iqml`) still load: a legacy `PaveDepth` row migrates to a single course named 'Pavement'.
- **Regenerated standard library** — `standard-roads.iqal` ships with real course stacks: lanes carry 4 cm wearing + 6 cm binder + 25 cm base (stepped, W+0.6 m) + 30 cm subbase (tapered 1:1, W+1.2 m); shoulders carry 5 cm wearing + 25 cm base + 30 cm subbase; sidewalks carry 5 cm concrete + 10 cm base.
- **Courses…/Materials… editor dialogs** — the Assembly Editor gains a **Courses…** button per side (lane/shoulder/sidewalk rows) for a per-row course grid with Add/Delete/Move, and the **Materials…** button for the drawing's material list with a colour picker.

Assembly editor display (Corridor v1.3 Phase 3)

- **Material-colored preview** — course shapes fill with their material's library color (curb/median use fixed fallbacks); course outlines draw as thin grey lines; recoloring a material updates the preview and thumbnails immediately.
- **Annotations** — width and cross-slope above each pavement row, cut/fill slope ratios at daylight points, with automatic decluttering below 40 px.
- **Dual daylight branches** — both the cut (dashed) and fill (solid) branches draw schematically from the hinge.

- **Thumbnail template list** — the assembly combo is replaced by an owner-drawn thumbnail list with per-assembly caching.
- **Course highlighting** — selecting a row in the Courses dialog outlines that course in the preview.
- **Mirror auto-rename** — mirroring a side now renames the copied rows by swapping the trailing L/R suffix (appending it when absent); only true residual name collisions are refused.
- Manual images regenerated with the new rendering.

43.1.19 2026-07-21

Alignment from 3D polyline with vertical profile

- **ALIGNFROMPOLYLINE** — **`ALIGNFROMPOLYLINE`** (`AFP`) now also accepts **3D polylines**. When the picked polyline's vertices carry differing elevations (a 3D polyline with real elevations, or a 2D polyline with per-vertex Z), the command asks `Create vertical profile from vertex elevations? [Yes/No] <Yes>` and, on Yes, `Vertical curve length <0>`: the alignment's vertical profile is built with one PVI per polyline vertex, the given vertical curve length applied to every interior PVI (0 keeps an angular profile; lengths are clamped so neighbouring curves never overlap). Each PVI is placed at the station where its vertex projects onto the finished alignment, so profile elevations line up with station queries even where corner curves cut inside the sketched polyline. Flat polylines keep the old horizontal-only flow. The resulting profile can be viewed and refined with **`CREATEPROFILEVIEW`** and **`DESIGNPROFILE`**.
- **Scripting** — Python `create_alignment_from_polyline` gains optional `use_z` / `vcurve_len` arguments, and Pascal scripts get the new `CreateAlignmentFromPolylineZ(polyHandle, radius, name, vcurveLen)` function.

Graphical alignment vertex editing

- **ADDPI / DELPI / EDITPI / ADDPVI / DELPVI** — five new Civil commands to add, delete or edit alignment PIs on the canvas and to add or delete design-profile PVIs inside a profile view, one click per vertex, each click (or edit) a single undo step. **ADDPI** prompts once, after the alignment pick, for a default curve radius (and, if greater than 0, a spiral length and type); the defaults apply to every PI clicked in that run, and Enter through the prompts gives sharp radius-0 PIs. **EDITPI** lets you click an interior PI to change its curve radius and spiral, offering the current values as defaults; end PIs are refused since they have no curve.

Embedded icons and menubar icons

- All toolbar/menu icons are now **compiled into the IQcad executable** — installations no longer need an `icons` folder (an `icons` folder next to the executable still takes precedence, so custom icon sets keep working).
- The **Civil menu** now shows command icons on its items (alignments, profiles, and the new vertex-editing commands).

43.1.20 2026-07-20

Slope Path traces along a sketched polyline

- **SLOPEPATH reworked** — **`SLOPEPATH`** (`SLP`) no longer asks for a start and destination point: you now sketch the route as a 2D/3D polyline (or pre-select one) and its vertices become waypoints. The command chains a maximum-slope gradient path through every waypoint across the surface,

merges collinear runs, keeps the sketch, and reports lengths, climb and descent, per-leg status and a per-vertex slope list.

43.1.21 2026-07-19

Seven new modify commands

- **BREAK** — (alias BR) breaks an entity between two points. Pick the object (the pick point is the first break point), then "Specify second break point or [First point]:" — type F to re-pick the first point, or @ to break at a single point with no gap. Lines, arcs and polylines split into two entities (or are shortened at an end); a circle becomes an arc; closed polylines become one open polyline.
- **JOIN** — (alias J) joins lines, arcs and polylines. Collinear touching lines merge into one line; touching arcs on the same circle merge into one arc; other touching chains merge into a polyline. Reports "N object(s) joined into M."
- **ALIGN** — (alias AL) aligns objects using one or two source/destination point pairs: the first pair translates, a second pair adds rotation, with an optional final scale step.
- **DIVIDE** — (alias DIV) places point entities dividing an entity into 2–32767 equal segments.
- **MEASURE** — (alias ME) places point entities along an entity at a fixed interval, starting from the end nearest the pick point.
- **FLATTEN** — sets Z = 0 on the selected objects, or the whole drawing if nothing is selected.
- **ARRAY** — (alias ARR) creates a rectangular, polar or path array of the selected objects.
- **BREAK** is fully undoable in a single step, and **JOIN** preserves arc curvature (bulge) exactly when building a polyline from mixed line/arc chains.

All seven commands are available from the **Modify menu**, a second "More" pane on the **Modify ribbon tab**, and the command line.

Command changes

- **ALIGNMENT alias renamed** — the alias of **ALIGNMENT** is now ALG (was ALIGN). The names ALIGN and AL are now used by the new object-align command (see above).

43.1.22 2026-07-18

AutoCAD-style ViewCube

- **New 3D navigation ViewCube** — a corner navigation cube (top-right by default) that mirrors the current 3D view. Click a **face** for a straight orthographic view, an **edge** for a 45° view or a **corner** for an isometric view, each with a smooth animated transition. **Drag** the cube to free-orbit, hover to highlight the zone under the cursor, and use the **Home** button to return to the default isometric view.
- **Compass ring** — an N/E/S/W ring around the base of the cube turns with the view; click a cardinal to snap to that side view.
- **Options** — toggle it from the **View** ribbon, or set its visibility, corner position and size persistently under **Settings ▯ ViewCube** (the same options the UCS gizmo has).

Ribbon reorganisation — dropdown command buttons

- **Draw → Dimensions** is now a single **Dimension** split button — click it for a linear dimension, or open the dropdown for every dimension type plus the dimension style manager, each with its own icon.
- **Civil tab** panes are collapsed into split/dropdown buttons (Points, Surfaces, Surface Edits, DEM, Alignment, Profiles, Sections, Assemblies, Culverts, Map, Transform): a normal click runs the primary command and the dropdown lists the rest. Stateful toggles (Elevation/Station Tracker, Map background, Toolspace) stay as their own buttons.
- **Civil menu & commands** — new **Assembly**, **Grading**, **Transformations** and **Parcels** sub-menus; the Civil menu and ribbon now expose the same commands, and around twenty civil commands that were previously command-line only (grading build/extract/balance, corridor auto-rebuild, alignment reverse/properties, surface rebuild / add-points / contour-labels / earthwork volumes, point groups, coordinate transforms, …) are now reachable from the UI, with new command icons.

Slope Path (max-slope gradient line)

- **New Slope Path command** — **SLOPEPATH** (SLP) traces a 3D polyline from a start point A to a destination B across a terrain surface so that no segment exceeds a maximum slope you specify — the reconnaissance "gradient line" used to find an alignment path up or down grade. It prompts for the max slope (%) and a step length (auto-defaulted from the surface size), then you pick A and B; the path climbs or descends automatically and snaps its final vertex to B.
- **Terrain source** — uses a selected (or the drawing's only) surface, or, if none is present, triangulates a temporary surface on the fly from selected contour polylines, 3D faces or points.
- **Output & report** — draws one 3D polyline draped on the terrain and reports the vertex count, 3D and plan length, total climb and descent, the steepest segment, and whether it reached B, stalled short of it, or ran off the surface; it warns if the closing segment (or any segment) exceeds the requested slope. Reachable via **Civil → Surfaces → Slope Path (Gradient Line)**.

43.1.23 2026-07-15

3D Object Viewer

- **New 3D Object Viewer** — isolate the selected objects (surfaces, gradings, culverts, 3D polylines) in a separate, read-only window and view them in true 3D: orbit / pan / zoom, six display modes, per-object show/hide and double-click isolate, and hover-to-inspect (elevation, slope %, aspect and area for TIN faces; part name for culvert parts). Open with **OBJVIEW** (OBJECTVIEWER), the **View** ribbon **Obj Viewer** button, or the canvas right-click.
- **Live update** — the viewer tracks its objects by handle and re-renders automatically when you edit them in model space, keeping the camera angle and any hidden/shown state; a **Live** toggle disables it, and **Reload** re-syncs to the current selection or refreshes the objects already shown.
- **Double-sided, smooth surfaces** — surfaces are rendered two-sided with smooth per-vertex normals, so a TIN no longer drops triangles when rotated. Navigation: left-drag pans, right-drag orbits, middle-drag pans, the wheel zooms.

Culverts (parametric box culverts)

- **New Culvert module** — **CULVERT** (CULV) builds a reinforced-concrete box culvert as a dynamic 3D civil object swept along a flow-line polyline: box barrel (segmented), bottom/top slabs and haunches, subbase and protection layers, inlet/outlet headwalls, splayed wing walls with footings and aprons, and an optional fall (drop) chamber — each a solid part on its own layer, with a computed concrete volume.
- **Tabbed editor** — double-click a culvert (or **CULVERTEDIT**, CULVEDIT) to edit box, wing, fall-chamber and material parameters with a live cross/long schematic and a live quantities table; **Load / Save** `.cvt` templates and **Import** legacy GCulvert files.
- **Sections & quantities** — **CULVERTSECTION** (CULVSEC) cuts a dimensioned 2D cross- or longitudinal section; **CULVERTQUANTITIES** (CULVQTY) reports concrete volumes by component and material and places a quantities table.

43.1.24 2026-07-10

Grading (site-grading pads)

- **New Grading module** — design a graded pad by projecting cut/fill slopes from a footprint polyline at a design elevation out to a target surface, producing a graded terrain surface and earthwork cut/fill volumes. **GRADING** (GRD) opens the Grading dialog. Supports flat / footprint-vertex-Z / drainage pad elevation; separate cut & fill outslopes entered as a ratio (3:1) or grade (%); auto cut-vs-fill per side; and **height-triggered benching** for cut and fill (width + interval) with an optional **bench cross-slope** back-drain.
- **Dynamic, self-drawing object** — a grading owns and draws its own geometry (graded TIN, daylight line, bench lines) with no stray entities, and **auto-rebuilds live** when you grip-drag its footprint or edit a criterion in the Object Inspector (toggle **Auto rebuild** off for very large pads and rebuild on demand with **BUILDGRADING**).
- **Volume balance** — **GRADINGBALANCE** (GRDB) solves the pad design elevation for a target net volume (default 0 = balanced earthwork).
- **Extract Surface** — **GRADINGEXTRACT** (GRDX) clones the graded surface into a standalone AECC_SURFACE for downstream use.
- **Show/Hide and Lock** — a grading has **Visible** and **Locked** properties and prospector toggles, consistent with surfaces and alignments. **Locked civil objects (surfaces, alignments, parcels, points, gradings) are now excluded from selection** — a locked object cannot be clicked, window-selected or moved; unlock it from its Civil Toolspace node.

  See *the Grading section* (page 219) in the Civil chapter.

43.1.25 2026-07-08

Earthwork volume report for comparison surfaces (VOLUMEREPORT / VOLREPORT)

- **Volume report from a diff surface** — a comparison (diff) surface built by **SURFDIFF** / **PCDIFF** can now be turned into a printable HTML earthwork report: cut, fill and net volumes, cut/fill/total areas, deviation statistics (min/max/mean/RMS `dz`), and a per-deviation-band breakdown with areas, signed volumes and area shares. Every figure is integrated directly from the diff surface's own triangulation, so the report is self-contained and works on any saved diff surface even after the source surfaces are gone; the band table matches the surface's on-screen style.

- **Four entry points** — **Civil › Surfaces › Volume Report…**, the **Earthwork Volume Report** ribbon button, the **Volume Report** action on a selected diff surface in the Object Inspector, and an offer to open the full report right after a compare. Command **`VOLUMEREPORT`** (alias `VOLREPORT`). See *Comparing Surfaces* (page 159) and the Surfaces command table.

43.1.26 2026-06-08

Plugin SDK — ribbon & menu contributions (API 1.6)

- **Ribbon buttons and menu items for plugin commands** — a plugin (and Python/PascalScript) can now make its commands **discoverable in the GUI**. `IIqHostUI.AddRibbonButton` / `AddMenuItem` (and `iqcad.add_ribbon_button` / `add_menu_item` and the `AddRibbonButton` / `AddMenuItem` PascalScript globals) add a button and a menu item that run a registered command. They land in a dedicated **Plugins** ribbon tab and **Plugins** menu, each plugin getting its own pane/submenu, with a plugin-shipped PNG icon (generic fallback); contributions are auto-removed when the plugin is disabled and the tab/menu hide when empty.
- **New ``ribbondemo`` sample** (Pascal + C twin) registers a command and adds a ribbon button + menu item bound to it. See *Ribbon & menu contributions* (page 354).

Plugin SDK — in-window dockable panels (API 1.5)

- **Dockable plugin panels** — a plugin (and Python/PascalScript) can now dock a panel **inside the IQcad main window** instead of opening a floating window. `IIqHostUI.CreatePanel` (and `iqcad.create_panel` / `CreatePanel` from scripting) returns the same `IIqForm` builder as `CreateForm`, but its widgets live as a tab in a docked strip on the right of the window, between the canvas and the Object Inspector. The strip appears when the first panel docks and hides when its last tab closes; closing a tab fires `OnFormClosed`. The control and event surface is identical to a floating form.
- **New ``dockdemo`` sample** (Pascal + C twin) is the docked twin of `formdemo`. See *Docking a panel* (page 354).

43.1.27 2026-06-07

Plugin SDK — live tool windows (API 1.4)

- **Live, non-modal tool windows** — the SDK gains `IIqHostUI.CreateForm` returning an `IIqForm` builder. A native plugin (and Python/PascalScript) can open a host-owned, non-modal window, add controls (buttons, the input editors as live controls, a label, and a read-only list), **retain handles** to them, and mutate them at runtime — all without linking the GUI toolkit. Controls are reached through one kind-checked `IIqControl` handle, and the plugin reacts to user input via an `IIqFormSink` it implements (`OnButtonClicked` / `OnValueChanged` / `OnFormClosed`, fired on the main thread and boundary-guarded). The host owns the window; stale handles go inert after it closes.
- **New ``formdemo`` sample** (Pascal + C twin) demonstrates a live window whose *Add* button reads a spin value and updates a list and label live. See *Live tool windows* (page 351).

Plugin SDK — parameter dialogs and rich message box (API 1.3)

- **Host-built parameter dialogs** — the SDK gains `IIqHostUI.CreateDialog` returning an `IIqDialog` builder: a plugin describes fields (text, multi-line, integer, float, boolean, choice, file, folder, colour), calls `Execute`, and reads the values back with typed getters. The host creates and owns every widget, so a native plugin can show a real parameter dialog without linking the GUI

toolkit. `IIqHostUI.MessageBox` is now a rich message box (caption, button set, icon, returned result).

- **Same dialogs from scripting** — Python (`iqcad.create_dialog` / `iqcad.message_box`) and PascalScript (`TScriptDialog` / `CreateDialog` / `MessageBox`) expose the same parameter dialog and rich message box, sharing the host's dialog engine.
- **New ``dialogdemo`` sample** (Pascal + C twin) demonstrates building a parameter dialog and drawing the chosen shape. See *Parameter dialogs and message boxes* (page 350).

Native plugin SDK

- **Native plugin SDK (API 1.2)** — IQcad can now be extended with native compiled plugins (`.dll` / `.so` / `.dylib`) that add commands, drawing operations, civil operations, and event handlers exactly like built-in features. Plugins use a COM-style vtable ABI (`cdecl`, no managed types) so they can be written in Free Pascal, C, C++, or any language with a C FFI, and distributed as binaries. See *Plugins (Native SDK)* (page 337) for the full design, two step-by-step tutorials, the manifest format, the bundled samples, and the complete interface reference.
- **Plugin Manager** — **`PLUGINS`** (alias `PLUGINMANAGER`; ribbon **Tools → Scripting → Plugins**) lists discovered plugins and lets you enable/disable/reload them. Plugins are **disabled by default** and require a one-time trust consent before they load, since a native plugin runs with full process privileges. Enabled/consented state persists in `settings.json` and autoloads on the next launch.
- **Ten worked samples** — five Free Pascal plugins each with a C twin (hello, geom, entsel, events, civilui), forming a learning path under `cad/plugins/samples/`.

Scripting and command-line improvements

- **Debug log for plugins and scripting** — with `"debugLog": true` in `settings.json`, plugin lifecycle (discover / load / activate / deactivate), command registration, and every `Host.Log` call are written to `iqcad_debug.log`, along with script output (categories `Plugin` / `Script`). `illDebug` messages go only to the log; info/warning/error also appear on the command line.
- **HELP lists plugin commands** — **`HELP`** now ends with a **Loaded plugin commands** section so users can see which external commands a plugin has added. Each command is also announced on the command line when its plugin loads.
- **Command aliases fixed for plugin/script commands** — typing a plugin or script command's alias (e.g. `RC` for `RECOLOR`) now dispatches correctly.

43.1.28 2026-06-01

Section Sheet dialog (CREATESECTIONSHEETDLG / CSSD)

- **Create Section Sheet dialog** — **`CREATESECTIONSHEETDLG`** (alias `CSSD`) replaces the ribbon **Civil → Section Sheet** button with a two-tab modal dialog that gathers all parameters at once before asking for an insertion point. Settings tab: alignment, one or more surfaces each with a per-surface line color, station interval, left/right width, optional station range (From / To), columns and rows per sheet, optional cut/fill base and comparison surfaces with cut/fill colors. Advanced display tab: vertical exaggeration, grid major/minor X·Y intervals, grid colors, auto-fit elevation band (or manual datum/top), show area labels, and title. Per-surface colors, cut/fill colors, grid settings, title, and station range are persisted in `.iqz` and survive alignment edits and resample (older files load with default palette and default display settings).

- **CSS command-line wizard retained** — the existing **CREATESECTIONSHEET** (alias CSS) wizard is unchanged and continues to work for scripting and power users.

43.1.29 2026-05-31

DEM to editable TIN, and a Civil ribbon DEM group

- **DEM → editable TIN** — **DEMTOTIN** (alias D2T) resamples a DEM surface into a *new* editable TIN at a chosen cell size, leaving the DEM intact. You can convert the whole DEM or a region (rectangle or closed polyline), with an optional decimation tolerance; the working grid tightens to the region so a small high-detail area does not hit the whole-DEM cell cap.
- **Civil ribbon DEM group** — the Civil ribbon tab gains a **DEM** group with buttons for **IMPORTDEM**, **DEMTOTIN**, **CLOUDSURFACE**, and **EXPLODECONTOURS** — the DEM commands are no longer command-line only.

Surface source-type provenance and DEM contours

- **Surface source type** — every surface now records where it came from (*Survey Points*, *Point Cloud*, *DEM*, *LandXML*, *Merged*, or *Derived*), set when it is created and persisted in the drawing. The label is shown by **SURFACEPROPERTIES** and next to each surface in the Civil Toolspace tree, so a DEM and a point-cloud comparison surface are no longer indistinguishable. Drawings saved before this release infer a sensible type on load.
- **Contours from DEM surfaces** — grid-backed DEM surfaces now generate contours by marching squares run directly on the elevation grid (no triangulation, NoData-aware, decimated for speed). They render through the normal surface style (`ContourVisible` + major/minor intervals), exactly like a TIN surface's contours.
- **Extract contours to polylines** — **EXPLODECONTOURS** (alias EXC) turns any surface's contours into editable polyline entities on `<Name>-Contour-Major` / `-Minor` layers, ready for drafting or DXF export.

43.1.30 2026-05-30

DEM cross-sections, point-cloud comparison, and ortho backdrop

- **Import GeoTIFF DEMs as surfaces** — **IMPORTDEM** (alias IDEM) reads a GeoTIFF elevation raster natively (uncompressed, Deflate or LZW; Float32 or Int16; strips or tiles) and adds it as a grid-backed surface, georeferenced from the GeoTIFF's own tags. The DEM is sampled by section sheets exactly like a TIN (bilinear interpolation, no triangulation), so cross-sections along an alignment can be taken straight from a DEM. NoData/void cells return honest gaps rather than fabricated ground; the CRS is reported but not reprojected.
- **DEM surfaces persist in the drawing** — a DEM survives save and reopen by re-decoding from its source `.tif` (a warning is shown if the file has moved).
- **Point-cloud comparison surfaces** — **CLOUDSURFACE** (alias CLS) bins a loaded point cloud's ground points into a min-Z grid surface (`Cloud_<name>`) so it can be overlaid on the DEM in a section sheet with cut/fill between the two — validating a DEM against survey data. Falls back to all points (and suggests **PCGROUND**) when the cloud has no ground classification.
- **Large orthophoto backdrops display** — gigapixel JPEG orthophotos attached with **IMAGEATTACH** now decode at reduced scale instead of rendering as a white placeholder; combine with **DRAWORDER** → Back to drape an ortho under the plan.

43.1.31 2026-05-26

Tunnel Section Viewer merged into Point Cloud Sections Along Alignment

- **Standalone Tunnel Section Viewer retired** — the separate viewer window and its ribbon entry have been removed. All its capabilities — angular-bin filtering, plane-mode toggle, and tunnel section placement — are now native to the **Point Cloud Sections Along Alignment** viewer (`PCSECTIONS`) under the **Tunnel** radio button.
- **Full tunnel filter panel** — Tunnel mode now exposes all eleven filter controls in the viewer: selection rule per bin (Outermost / Innermost / Median / Percentile), angular bin width, min / max radius, minimum points per bin, percentile, plane proximity %, invert-skip zone, min / max elevation cutoff, and radial gap threshold.
- **Shared toolbar controls** — a **Plane** combo (Vertical / Perpendicular) and an **Absolute Z** checkbox are available in both Ground and Tunnel modes. Absolute Z flips the Y axis between CL-relative ΔZ and absolute world Z without recomputing the point data; toggling it resets pan/zoom so the new extents are visible. Perpendicular plane mode in Ground mode is a placeholder — it shows a warning and falls back to the vertical slice.
- **Axis info strip** — a six-cell band above the station slider shows X / Y / Z / Dir / Grade / R for the active station and updates on every scrub.
- **SheetKind discriminator** — the `TAECC_PointCloudSectionSheet` entity now carries a `SheetKind` field (Ground / Tunnel). Tunnel sheets store the `TAECC_TunnelSectionLine` handles and the eleven tunnel filter parameters separately; cell thumbnails render the actual cross-section profile using a PCA-derived local frame. Both kinds round-trip through IQML and the compact streaming format; legacy IQZ files load as Ground.
- **Inspector** — the Object Inspector exposes a read-only **SheetKind** field on the sheet proxy.
- **Re-sample for tunnel sheets** — the Re-sample action on a tunnel sheet re-fits every section using the stored filter parameters.
- **Bug fixes** —
  - Vertical-mode tunnel placement in `PlaceTunnelSectionsAlongAlignment` was double-counting the alignment Z (off by the alignment elevation in world units); the reconstructed section was placed at twice the correct height. Fixed.
  - Absolute Z painting: grid labels were offset from the actual data points by `AlignmentZ` pixels. Fixed — labels now align with data.
  - Tunnel sheet cell rendering was producing a thin diagonal line instead of the cross-section profile. The cell now fits the tunnel profile to the cell rectangle via a PCA-derived local frame.
  - Catmull-Rom spline ghost control points: Z was hard-coded to 0, causing massive Z overshoot for polylines anchored at high world Z (most visibly: sliver/empty tunnel sheet thumbnails). Z is now extrapolated the same way as X and Y.

43.1.32 2026-05-24

Point Cloud Sections Along Alignment — pick insertion point on canvas

- **Generate sheet pick-on-canvas** — the *Point Cloud Sections Along Alignment* viewer's **Generate sheet…** button used to ask for the sheet insertion X/Y through two `InputBox` dialogs. It now hides the viewer, switches the main canvas into a one-shot pick mode (status bar reads *"Pick insertion point (Esc to cancel):"*), and creates the sheet at the point you click. Object snap is honoured, so endpoint

/ midpoint / intersection snaps work for sheet placement. Press **Esc** or **right-click** on the canvas to cancel — the viewer reappears with a *"Sheet generation cancelled."* message and no sheet is added.

- **Locale fix as a side effect** — the old `InputBox` path parsed the typed coordinates with `StrToFloatDef`, which is locale-sensitive and silently corrupted Greek-locale input (comma vs. dot decimal). The new pick path has no string parsing at all, eliminating the locale hazard.
- **Reusable canvas primitive** — the change adds a new public `TCADCanvas.BeginPointPick(ACallback)` / `CancelPointPick` API, mirroring the existing `BeginBoundaryPick` pattern. Future dialogs that need to solicit a single world-coordinate pick can reuse this primitive instead of opening their own coordinate input dialogs.

43.1.33 2026-05-08

Surface build pipeline — algorithms + filters + dialog

- **Per-surface build settings** — every `TAECC_SURFACE` now owns a `TSurfaceBuildSettings` object that captures the triangulation algorithm and four optional cleanup filters (**MaxEdgeLength**, **MinTriangleArea**, **MaxSlope**, **MaxAspectRatio**) plus a **Clip to outer boundary** toggle. All filters default off so existing surfaces are unaffected; users opt in via the new dialog. Settings round-trip through IQML and are re-applied automatically on every `CREATESURFACE` and `REBUILD`. Loading an IQML file does **not** re-apply the filters — the saved geometry is preserved.
- **SURFBUILDSETTINGS** / `SBS` — new command opens an all-code dialog with the algorithm dropdown and one row per filter (checkbox + numeric input, greyed out when disabled). Dialog has `[Reset] [Apply Now] [Cancel] [OK]`; `Apply Now` applies filters in place and keeps the dialog open for iterating threshold values. Reachable from the Surfaces menu, ribbon, and Civil Prospector right-click on a surface.
- **Constrained Delaunay** — new algorithm option `taConstrainedDelaunay`. Runs the standard Bowyer-Watson sweep and then forces every breakline polyline and outer/hole boundary segment into the TIN as constrained edges, so ridges, ditches, road centrelines, and boundary loops survive subsequent local edits and Delaunay flips. Uses the existing breakline / boundary-reconform infrastructure under the hood.
- **Gridded TIN** — new algorithm option `taGriddedTIN`. When the input points form a regular `NX × NY` lattice (within 1 mm), the surface is built by emitting two triangles per cell with a Z-aware diagonal split (the diagonal whose endpoints are closer in Z is preferred — smoother contours where ridges cross cells). ~10× faster than Bowyer-Watson on lattice-aligned data; falls back to Bowyer-Watson with a warning if the points are not on a grid.
- **SURFREGRID** / `SRG` — new command resamples an existing irregular surface onto a regular grid. Default cell size = median edge length of current TIN. Cells outside the existing surface footprint sample to *NaN*; cells with three valid corners emit a single triangle (preserves the boundary outline). Hard cap at 25 M cells refuses pathological cell-size choices with a clear error.
- **SURFDELETELONGEDGES** / `SDLE` — new command for ad-hoc cleanup outside the build pipeline (e.g. on a TIN imported from LandXML or 3DFACEs that we can't rebuild). Walks every unique edge, removes those longer than the user threshold, undoable as one step via `SURFUNDO`. Stats line shows `min / median / 95th / max` so the user can pick a sensible threshold; `>50%-of-edges` confirmation prompt prevents fat-fingered deletes.

Point cloud → surface pipeline

- **PCGROUND** / `PCG` — new slope-based ground classifier. Two-pass cell-grid algorithm: pick the lowest-Z point per cell as a ground seed, validate against 3×3-neighborhood minimum (drops rooftop low points), promote seeds to LAS class 2, then accept every other point whose slope from its nearest seed is below the user threshold. **Vendor LAS classifications are preserved** — only points currently at class 0 (Created) or 1 (Unclassified) are touched. Defaults: cell = 5 m, max slope = 45°.
- **PCSURFACE** / `PCSURF` — new command builds a `TAECC_SURFACE` from a point cloud filtered to ground-only points (or all points via `A`). Voxel-decimates dense clouds before triangulation; defaults to **Gridded TIN** so the lattice fast path engages automatically when the cloud is grid-aligned. Reachable from Point Cloud menu, ribbon, and right-click context.

3DFACE import / convert

- **FACE2SURFACE** / `F2S` — bulk-convert `3DFACE` entities in the drawing into a single `TAECC_SURFACE`. Welds shared corners with `DeduplicateAndRemapTIN` (1 µm tolerance) so a topographic export of N independent triangles collapses to roughly N/2 unique points and N triangles.
- **ADDDXFFACES** / `ADF` — new command streams a DXF file for its 3DFACE entities and merges them straight into a target surface, **without ever materialising TCADFace entities on the canvas**. Lets a 1.5 GB / 4 M-face topographic export land directly inside the surface object. Civil Prospector's surface popup gains an **Add 3D Faces from DXF…** entry.
- **Surface popup wiring** — Civil Prospector right-click on a surface name now has a context menu with the surface name as a header and the new commands as items (Add 3D Faces, Build Settings, Resample to Grid, Filter Long Edges, etc.).

Surface editing improvements

- **SURFDELETELINE** / `SDL` rewritten — now runs as a **continuous loop** (click edges one after another, right-click or Enter to finish) and supports a **window-pick mode** (type `W` then click two corners; every TIN edge with both endpoints inside the rectangle is removed in one batch via `Edit-DeleteLine` so `SURFUNDO` can revert the lot). A dashed cyan rectangle rubber-bands between the two clicks for visual feedback.
- **TINDeleteLine — real removal** — fixed a long-standing UX trap where `TINDeleteLine` actually swapped the edge with the opposite quadrilateral diagonal (and silently no-op'd on boundary edges) instead of removing it. Single-click `SDL` now physically removes the edge and the triangles touching it, leaving a hole. `SURFSWAPEDGE` / `SES` is the explicit command for the swap operation.

Performance / precision fixes

- **2D-mode LOD for 3DFace + PolyfaceMesh** — previously these entities were excluded from sub-pixel dot-LOD because their XY-projected bbox can lie in 3D-orbit views. The exclusion now applies **only** in 3D mode; 2D plan view collapses sub-pixel 3DFaces to a single `SetPixel` call. The 4.2 M-face `TRIGWNA.DXF` import is now navigable in 2D pan/zoom.
- **Contour precision at large coordinates** — `UploadContours` now derives the GL local origin from the contour bbox centre when the triangle VBO hasn't been uploaded yet. Previously the `GLfloat` cast at world coords ~4.4×10^6 (EGSA87 / UTM) lost ~0.5 m of precision and contour lines snapped to a coarse grid; the visible stair-step zigzag at large Eastings/Northings is now gone.

43.1.34 2026-05-02

Zoom to Object + multi-scan PTS import

- **Zoom to Object** — new `ZOOMOBJECT` command frames a single picked entity in the viewport with a small margin, computed from the entity's bounding box. The same action is exposed in the Civil Prospector right-click menu on **Surface**, **Alignment** and **Profile View** nodes (the Point Cloud node already had its own octree-based *Zoom To*). Implemented as a new `TCADCanvas.ZoomToObject` method that all four call sites share.
- **PTS multi-scan import** — the Leica PTS reader now reads every scan in a multi-scan file (`count1` / points / `count2` / points / …). Previously the loop broke at the first scan's declared count, silently dropping every later scan. After consuming a scan's declared point count, a single-integer line is recognised as the next scan's count and parsing continues; if the inter-scan integer is missing, the reader leniently keeps appending point lines. All scans are merged into a single point cloud cache.

43.1.35 2026-05-01

Surface — breaklines, 3D editing, GL fixes

- **Breaklines** — polylines can now be attached to a surface as forced (constrained) edges. `SURFADDBREAKLINE` / `SAB` consumes the current selection or prompts for one; the Civil Prospector's *Definition* popup gains an **Add Breakline…** entry; each surface shows a **Breaklines (N)** node with a per-entry right-click **Delete**. Breakline geometry is kept on its own component layer (default `C-TOPO-{Name}-BRKL`, magenta) and can't be flipped by a later swap-edge / delete-line operation. Visual style (visibility, colour, weight, linetype, LT scale) is configurable on a new **Breaklines** tab in the Surface Style editor.
- **Edge swap** — the convexity test in `TINSwapEdge` was rewritten so swaps no longer fail silently when the local triangulation winding put the opposite vertex on the "wrong" side of the diagonal. New triangles are always emitted in CCW order.
- **3D wireframe rendering** — in any rotated 3D view, the whole TIN now renders correctly under pan / zoom / orbit. The previous render used a screen-rect XY grid which dropped triangles outside the projected XY rectangle even when they were inside the camera frustum.
- **GPU mesh cache** — captures the dirty marker before `GetMesh3D`, so the cache invalidates correctly after any edit. Previously, edits were rendered only after a coincidental cache miss (display-mode switch, etc).
- **Contour labels** — skip the LCL text pass on GL contexts so labels at oblique angles no longer appear doubled (same fix pattern as the tunnel DR-label fix).
- **Edit log** — `Add Line` and `Delete Line` operations now appear in the log and round-trip through save/load. Individual entries can be deleted via right-click on the *Edit Log* node in the Civil Prospector tree.

LandXML enrichment

Surfaces now round-trip through LandXML with their full IQcad-level detail. In addition to points and faces, the writer emits:

- `<Boundaries>` (outer + island) with inline NEZ point lists.
- `<BreakLines>` carrying every breakline polyline.
- `<Feature code="IQcadEdits">` per surface with the edit log.

- Top-level `<Feature code="IQcadPointGroups">` capturing the survey database's groups + point-number membership.
- `<Feature code="IQcadSourceGroups">` per surface listing the source point groups, so the *Definition* tree repopulates on reimport.

The reader handles both the legacy point-ID and the new inline-NEZ `<PntList3D>` formats and parses every new `Feature`. Polylines already attached to a surface as breaklines are no longer re-exported as separate alignments.

Civil Settings

The surface object-layer table gains an 8th row: `Surface / Breakline / TOPO / BRKL` (magenta, Continuous). The **Civil Settings → Object Layers** dialog picks the row up automatically.

Documentation

E57 (ASTM E2807) is now documented as a supported point-cloud import / export format alongside LAS / LAZ / XYZ / PTS / PLY. Requires the `libE57CWrapper` runtime library next to the executable; absent it, the format is silently disabled.

43.1.36 2026-04-30

- Shapefile import / export — new entries under Files → Import / Export and the corresponding ribbon panes. Drawings can now be exchanged with QGIS, ArcGIS and GDAL. See *Exporting and Importing* (page 97) for details.

43.1.37 2026-04-25

DXF / DWG round-trip

- **Full dimension round-trip** — `DIMSTYLE` and `STYLE` tables now read and write every typed field plus an opaque tail for forward compatibility (~80 dimvars). `DIMENSION` entities round-trip with per-subtype defpoints, oblique angle, horizontal direction, attachment, line-spacing style/factor, source-block reuse, and per-dim overrides via `ACAD_DSTYLE` xdata. The NILA reference drawing now round-trips 14/14 DIMSTYLE, 14/14 STYLE, and 1457/1457 DIMENSION entities.
- **ATTRIB / ATTDEF preservation** — block attributes and block-template attribute definitions survive a full DXF round-trip including invisible placeholders, empty-text placeholders, tag names, prompt strings, flag bits (invisible / constant / verify / preset), text style, width factor, oblique angle, horizontal/vertical alignment and the second alignment point. NILA round-trips 3420/3420 ATTRIB entities and the full ATTDEF schema for every editable block.
- **MULTILEADER (MLEADER) round-trip** — read native `MULTILEADER` records by walking the nested `CONTEXT_DATA` / `LEADER` / `LEADER_LINE` blocks and reconstructing a `TCAD-Multileader` with tip / anchor / text / arrow size / dogleg length / text height preserved. Writer emits a real `MULTILEADER` (subclass `AcDbMLeader`) record instead of decomposing to LINE/SOLID/TEXT primitives. Inline MText escape codes (`\fArial|b0|...;`) strip cleanly when rendering single-line leader text.
- **Generic xdata + APPID auto-registration** — third-party application data attached to entities via DXF codes 1001..1071 is now preserved verbatim through import/export. The APPID table auto-registers every unique APPID name found in any entity's xdata so AutoCAD / BricsCAD accept the data on read.

- **MINSERT array rendering** — `INSERT` entities with column count > 1 or row count > 1 now tile the block contents over the full `ColumnCount × RowCount` grid at the configured spacing, matching AutoCAD's MINSERT convention (rotated, scale-independent spacing). Bounding box and hit testing cover all cells.
- **OCS extrusion (Arbitrary Axis Algorithm)** — INSERT, ARC, CIRCLE, ELLIPSE, TEXT, LWPOLYLINE, and POINT entities encoded with extrusion vector `(0, 0, -1)` (common for mirrored entities in AutoCAD-written DXF) are converted to WCS via the AAA at read time so they render at the correct mirrored position instead of the opposite side of the drawing. Surfaced on NILA where two of four arrows in each diffuser cell were rendering at wrong positions.
- **Nested block matrix composition** — `TCADInsert.RebuildCache` composes parent and child transforms as a 2×2 linear matrix instead of trying to compress them back into `ScaleX/ScaleY/Rotation`. Handles arbitrary nesting depth with any combination of rotation, non-uniform scale, reflection (mirror), and shear. Fixed the case where mirrored parents (e.g. `scale(-1, 1)` + `rot(180)`) produced arrows at the wrong angle inside nested diffuser blocks.
- **Full INSERT / BLOCK / ATTRIB code coverage** — every DXF group code the spec calls out is now read, written, and survives `IQML` / `IQZ` round-trip:
  - `INSERT`: `43` Z-scale, `70/71/44/45` MINSERT params, plus `210/220/230` extrusion (handled via AAA).
  - `BLOCK`: `70` flag bits (anonymous / xref / overlay), `1` xref path, `4` description.
  - `ATTRIB`: `7` style, `41` width factor, `51` oblique, `72/74` H/V alignment, `11/21/31` second alignment point.
- **DXF table coverage** — the LTYPE table now emits user-defined linetypes from the canvas in addition to the hardcoded standard set, preserving custom linetype references on round-trip. LAYER, STYLE, DIMSTYLE, and BLOCK_RECORD tables already round-trip the common path (color, true color, lineweight, plot style, transparency, frozen, locked, fixed-length text style, full dimstyle vars).
- **Handle stability (Phase 4)** — entity handles (DXF code 5) and owner pointers (code 330) are preserved on import so external references (xref targets, dimension associations, ATTRIB→INSERT linkage, layout viewport ownership, third-party cross-references) round-trip without breaking. Canvas `SyncHandleCounter` advances the auto-allocator past the highest preserved handle so newly created entities don't collide.

Editing UI

- **Dimension Style Editor preview** — the live preview pane was rewritten as an AutoCAD-faithful 2 × 2 grid of sample dimensions (linear, aligned, radius, angular). Honours the full set of editable fields: arrow heads / sizes, ext-line offset and extend, dim-line and ext-line colors, suppress flags, centermark size, prefix / suffix, alternate units, tolerances, text height (scaled by `OverallScale`), text fill colour, and the angular format (decimal degrees / DMS / grad / radian) — every change in the editor reflects on the next paint.
- **Dimensions ribbon pane** — 11 buttons covering linear, aligned, rotated, radius, diameter, angular, ordinate, arc-length, jogged, baseline, and continued dimensions, all wired through the existing command-line dispatch.
- **Dimension Style Manager extension** — the editor's tabs cover all ~80 `TCADDimStyle` fields across seven AutoCAD-style tabs (Lines, Symbols, Text, Fit, Primary Units, Alternate Units, Tolerances).

- **Properties inspector — dimension Advanced category** — exposes ObliqueAngle, HorizDir, TextRotation, Attachment (1..9), LineSpaceStyle, LineSpaceFactor, SourceBlockName, OverrideCount.

IQML / IQZ

- **Comprehensive IQML coverage** — every new entity field added in this release survives a full `DXF → IQML → DXF` round-trip in both the DOM and the streaming I/O paths: text style / alignment / width / oblique / align-point / attrib-tag / prompt / flags, insert ScaleZ and MINSERT params, block flags / xref / description, owner pointer and generic xdata.
- **Streaming chop bug fixed** — `LoadFromIQML` matched the first `<Entities>` element when chopping out the top-level entities section, but block definitions also contain `<Entities>` children. The chop now matches `<Entities count="..."` specifically so block contents survive an IQML reload.

Civil engineering

- **Section sheet from canvas selection** — the `CSS` (`CREATESECTIONSHEET`) command now prompts the user to click an alignment object and one-or-more surface objects on the canvas instead of typing names at the command line. A new **Civil > Sections** ribbon pane exposes a **Section Sheet** button that dispatches the same command.
- **Civil DXF export** — alignments, profile views, section views, and section sheets now export to DXF as drawable primitives:
  - `TAECC_ALIGNMENT` writes a 3D polyline centerline with proper `AcDbEntity` / `AcDb3dPolyline` subclass markers and per-vertex `AcDbVertex` / `AcDb3dPolylineVertex` markers.
  - `TAECC_PROFILEVIEW` emits a frame rectangle plus one polyline per sampled profile and design profile, with `NaN` gaps split into separate polyline segments instead of being bridged by a phantom line dropping to `Z = 0`. Title text is included.
  - `TAECC_SECTIONVIEW` emits a frame plus one polyline per surface section and a station label.
  - `TAECC_SECTIONSHEET` renders each grid cell inline (frame + sections + station label) so the entire sheet survives a DXF round-trip even though the host `TAECC_SECTIONVIEW` instances aren't placed on the canvas.
- **Profile view Z-elevation strip** — two extra label rows below the bottom axis show the design and ground elevations at every major station tick (`Design` / `Ground`), matching the AutoCAD Civil 3D profile-view convention. `NaN` (gap) elevations are skipped.
- **Civil label colours follow the canvas** — profile-view and section-view axis labels, station / offset / Z-strip labels, and titles now use a canvas-aware colour: white on IQcad's dark model canvas, black inside paper-space viewports. The previous behaviour routed through `EffectiveColor`, which produced black-on-black text whenever the civil layer was assigned `clBlack`.
- **Civil DXF text colour** — exported civil labels emit ACI 7 (`BYLAYER` automatic) plus an explicit 24-bit white via DXF code `420 = $00FFFFFF` so BricsCAD (which ignores ACI 7 for `TEXT`) still renders them white on its dark model background.

43.1.38 2026-04-04

- **Point cloud support** — import, export, and display large point clouds (hundreds of millions of points). Supported formats: LAS, LAZ, XYZ/CSV, PTS/PTX, PLY. Octree-based LOD with adaptive threshold. Six colour modes. `.iqpc` binary cache for fast reload. Background export with progress bar.
- **LAZ read/write** — full LASzip 3.5 DLL integration for compressed LAS files. Read and write with proper struct alignment and inventory tracking.
- **Elevation clipping** — `PCCLIP` command filters visible points by Z range. Useful for viewing floors, isolating terrain, or inspecting cross-sections.
- **Prospector delete** — right-click a point cloud in the Prospector for Delete (with confirmation, undoable via Ctrl+Z).
- **Portable IQZ paths** — point cloud source file saved as relative path, cache file as filename only. Projects can be moved between folders.
- **Display fix** — fixed critical rendering bug where only ~1/3 of the cloud was visible (vertex Z exceeded glOrtho clip range).

43.1.39 2026-04-02

- **Binary DXF support** — DXF import and export now support both ASCII and binary DXF formats. Export dialog offers five options: ASCII R2018/R2000/R12 and Binary R2018/R2000. Binary files are auto-detected on import.
- **GeoJSON import/export** — import and export GeoJSON files via **File > Import** and **File > Export** or the Output ribbon. Supports points, lines, polygons, and all Multi* geometry types. CRS transformation with 9 EPSG presets.
- **KML/KMZ import/export** — import and export KML and KMZ files. KMZ archives are handled transparently. Features grouped into Folders by layer name on export.
- **MTEXT editor improvements** — double-click on MTEXT opens the editor dialog. Bold/Italic/Underline/Strikethrough toolbar buttons now toggle correctly. The Object Inspector Content property shows a `[...]` button to open the editor.
- **Inspector window selection fix** — fixed a crash (GPF) when using window selection to select multiple objects.
- **Rotated viewport cross grid** — grid crosses now rotate with the viewport rotation instead of staying horizontal. Crosses outside the viewport boundary are properly clipped.
- **Grid text properties** — new TextOffsetX, TextOffsetY (mm) and TextStep properties control the position and density of coordinate labels at grid intersections. TextStep=2 labels every other cross in both X and Y.
- **PDF export: grid text and outer border** — grid coordinate labels at intersections and the outer border rectangle are now included in PDF export.
- **PDF export: rotated viewports** — PDF export now correctly renders rotated viewport content matching the on-screen layout view.

43.1.40 2026-03-30

- **LOD rendering system** — screen-space aware rendering for drawings with 100K–1.4M+ entities. Sub-pixel entities (< 1.5px) drawn as colored dots; ellipse tessellation adapts from 8–72 segments based on screen radius; spline points decimated when < 1px apart; hatch fill suppressed below 10px; civil point labels hidden below 5px.
- **Surface LOD** — three-layer optimization for surfaces with 1M+ triangles: (A) sub-pixel triangle culling before coordinate transforms; (B) $\sqrt{N} \times \sqrt{N}$ spatial render grid for viewport-aware iteration; (C) bitmap cache for instant pan drag (zero triangle processing during interaction).
- **Selection performance** — SelectAll uses O(1) flag check instead of O(n) IndexOf. 140K entities: 14.7s → 0.5s.
- **Ribbon reorganization** — new View tab (Navigate, Display, Select panes moved from Home); Clipboard pane moved from Modify to Home; Erase removed from Home (in Modify).
- **Full benchmark** — 14 entity types × 10K each. Reports per-type creation speed, operation timings (Repaint, Zoom Extents, Select All), and grand total.

43.1.41 2026-03-29

- **Blender-style gizmo rotation rings** — 3 colored rings for constrained axis rotation.
- **Point group commands** — ADDTOGROUP, REMOVEFROMGROUP, CREATEGROUP for managing groups from the canvas.
- **Right-click context menu** — select points, right-click to add to group instantly.
- **Association highlighting** — selecting profile/section views highlights associated surfaces and polylines.
- **Label style editor fix** — Tunnel and Parcels categories added; DR label text scaling corrected.
- **PDF export unified** — single PLOTPDF command with overwrite prompt.
- **UI reorganization** — Explode to Modify; Mesh/Surface conversions to Civil > Surfaces.
- **Tunnel test improvements** — CivilScale 100, ISO axes linetype, acadiso.lin in resources.

43.1.42 2026-03-28

- **UCS Gizmo** — interactive 3D navigation gizmo replacing the flat 2D UCS icon. Click an axis to snap the view; drag background to orbit. Depth-sorted rendering with cone arrowheads and semi-transparent background circle. Configurable corner, size, and opacity in Settings.
- **3D rendering** — rulers auto-hide in 3D mode; 3D crosshair with colored axes (red X, green Y, blue Z); zoom-at-mouse-point and pan in 3D mode; fixed entity clipping (spatial index uses ScreenToWorld3D).
- **Polyface Mesh** (`TCADPolyfaceMesh`) — shared vertex pool + face index array. DXF import (POLYLINE flag 64), DXF export, IQML serialization, wireframe rendering, grip editing (<500 vertices).
- **ACIS Pass-Through** (`TCADAcisEntity`) — opaque round-trip for 3DSOLID, REGION, BODY, SURFACE and subtypes. Stores raw SAT data, renders as labeled placeholder. Preserves Civil 3D data.
- **MESH2SURFACE** (`M2S`) — convert polyface mesh to TIN surface.

- **SURFACE2MESH** (S2M) — convert TIN surface to polyface mesh.
- **EXPLODE** — implemented for polyface mesh (produces 3DFace entities).
- **Civil label style fixes** — all built-in label styles updated to paper-mm TextHeight values; constructor default changed from 1.0 to 2.5 mm.
- **UI** — Draw menu: Polyface Mesh. Modify menu: Mesh to Surface, Surface to Mesh, Explode. Ribbon: Convert pane. Test menu: AddSampleEntities includes polyface mesh and ACIS samples.
- **SpkToolbar ribbon** — migrated from IQRibbon to SpkToolbar with 5 built-in themes (Office 2007 Blue/Silver/Black, Windows 10, Metro Light). Theme selection is persisted in settings.json.
- **Per-tab command line history** — each drawing tab now maintains its own command history, so switching tabs restores the command context for that drawing.
- **Parcel entity** (TAECC_PARCEL) — closed boundary polygons with computed area/perimeter, user data fields, 4 built-in styles (Standard, Property, Easement, ROW). CREATEPARCEL command. Full IQML and LandXML I/O support.
- **Unified CAD Style Manager** — single window for Text, Dimension, Table, Multiline, Point, and Plot styles with embedded editors.
- **Unified Civil Style Manager** — single window for Point, Surface, Tunnel Section Line, Alignment, Parcel, and Label styles with embedded editors.
- **Point Style Editor redesign** — checkbox-based marker shape selection replacing the old grid picker.
- **Help tab** — new ribbon tab with links to documentation, keyboard shortcuts reference, and About dialog.

43.1.43 2026-03-27

- **Cross-section system** — multi-surface cross-sections perpendicular to alignments with cut/fill computation and earthwork volumes.
- **CREATESECTIONSHEET** (CSS) — interactive command to create section sheets from alignment + surfaces with grid layout.
- **EARTHWORKVOLUMES** (EWV) — earthwork volume table using average end area method.
- **Multi-surface model** — each section can show multiple surface profiles with cut/fill computed between any two selected surfaces.
- **Dependency reconnection** — profile views and section views now reconnect to source entities after save/load.
- **Prospector** — added “Section Sheets” root node.

43.1.44 2026-03-25

- **CSTRANSFORM** (CST) — global coordinate transformation using common/control points. Supports Helmert 2D, Affine 2D, and Polynomial degree 1/2/3 with least-squares adjustment and residual analysis.
- **RUBBERSHEET** (RBS) — local non-uniform rubber-sheet transformation. Linear (Delaunay piecewise affine) and Smooth (Thin Plate Spline) modes.

- **Object Inspector redesigned** — now uses TVirtualStringTree with collapsible categories, owner-drawn combo editors (colour swatches, linetype patterns, lineweight previews), multi-selection support ("Varies" display), value type colour-coding, keyboard navigation, human-friendly property labels.
- **Selection filter** — combo at the top of the inspector limits picks to a specific entity type.
- **Quick Access Toolbar** — New, Open, Save, Save As, Undo, Redo, Properties toggle.
- **Layout persistence** — panel sizes saved to settings.json.
- **IQZ compressed format** — gzip-compressed IQML as default save format.

43.1.45 2026-03-23

- **Point Manager paste** — paste tab-separated text (from Excel) into the Point Manager to create survey points.

43.1.46 2026-03-22

- **Hatch system variables** — 20 AutoCAD-compatible HP* variables.
- **Hatch transparency** — per-entity HatchBackgroundColor and HatchTransparency.
- **Drawing Variables dialog** — System Variables grid tab with editable values and Scope column.
- **Tunnel section line styles** — separate overbreak/underbreak hatch patterns; scanline-based hatch rendering.

43.1.47 2026-03-21

- **Codebase audit** — 122 findings resolved across core engine, entity system, and civil/AEC subsystem.
- **DXF locale fix** — `DXFFloat()` helper ensures valid DXF on non-English systems.
- **Scale/Rotate overrides** — Circle, Ellipse, Text, MText, Multileader, Arrow now update entity-specific fields on transform.
- **Block insert caching** — eliminates 10K+ allocations per frame for block-heavy drawings.
- **Entity class registry** — runtime registration for all 24 entity types.
- **Selection manager** — extracted from TCADCanvas into separate unit.

43.1.48 2026-03-19

- **3D wireframe view** — `3DORBIT/ORBIT` command with arcball rotation, `PLAN` and `VPOINT` commands.
- **VIEWTWIST / ALIGNVIEW / PLANVIEW** commands.
- **Elevation banding** — colour ramps (green-to-red, blue-to-red, grayscale) on TIN surfaces.
- **Civil linetypes** — configurable linetypes for surface, alignment, and tunnel section line style components.
- **Alignment Style Editor** — new dialog for editing alignment display styles.

43.1.49 2026-03-18

- **TSOPTIMIZE / TSOPT command** — optimize tunnel section line fit using Nelder-Mead simplex minimizer. Five optimization levels.
- **Trim/Extend** — now supports arc, polyline, ellipse, and spline entities.
- **Chamfer** — second distance defaults to first distance.
- **Hatch fill engine** — pattern-based and solid hatch generation.
- **Spatial index** — grid-based spatial index for viewport-culled rendering.
- **Background tasks** — parallel infrastructure with modal progress dialog.

43.1.50 2026-03-16

- **OFFSET command** — full implementation with phantom preview and `OFFSETDIST` system variable.
- **ERASE enhancement** — prompts "Select objects:" when nothing selected.
- **Cursor modes** — full crosshair / pickbox-only based on context.
- **Escape key** — priority-based cancel (command > tool > selection).
- **OSNAP for edit commands** — MOVE, COPY, ROTATE, SCALE, OFFSET use OSNAP when picking points.
- **dmZoneExtremes** — new DR display mode for Tunnel Section Lines.

43.2 Drawing & Editing

- All CAD entity types: Line, Polyline, Circle, Arc, Ellipse, Spline, Rectangle, Polygon, Text, MText, Hatch, Image, Table, Dimensions, Multileader, Multiline, XLine, Point, 3D Face, 3D Polyline, Bezier, Arrow
- Grip editing, move, copy, rotate, scale, mirror, offset, explode
- Trim, extend, chamfer, fillet
- PEDIT (polyline vertex editing — close, join, width, vertex edit, spline, decurve, reverse)
- Block creation, insertion, in-place editing (BEDIT/BCLOSE), Block Manager
- Drawing order (front, back, above, below)
- Undo/redo per drawing
- Multiple drawing tabs (MDI)
- PURGE (layers, blocks, styles, linetypes), AUDIT, RECOVER

43.3 Civil / AEC

- **Survey Points** — import/create with style-driven markers and labels; Point Manager with paste, lock, transform, renumber, group management
- **TIN Surfaces** — Bowyer-Watson triangulation, contours (Chaikin smoothing), spot elevations, boundary clipping, decimation, surface editing commands (add/delete/move points, swap edges, add/remove lines, raise/lower), elevation banding, contour label placement

- **Tunnel Section Lines** — measured cross-section deviation analysis with DR labels, smart display modes (All, ZoneExtremes, Threshold, Interval), area hatch, style-driven rendering, TSOPTIMIZE fit optimization
- **Horizontal Alignments** — tangents, circular curves, spiral (clothoid) transitions; PI-based creation, station labelling, style-driven rendering
- **Cross-Sections** — multi-surface section sampling perpendicular to alignments, section views with cut/fill shading, section sheets with grid layout, earthwork volumes (average end area method)
- **Coordinate Transformations** — CSTRANSFORM (Helmert, Affine, Polynomial) and RUBBERSHEET (Linear, Smooth TPS) with common point management, residual analysis, and transformation reports
- **Civil Toolspace / Prospector** — tree view with surfaces, points, sections, alignments, and property grid
- **Civil Settings** — units, coordinate system (182 EPSG), transformation, ambient settings, abbreviations
- **LandXML** — import/export for surfaces, survey points, and alignments

43.4 Styles

- Text Style Editor
- Dimension Style Manager + Editor (5 property areas)
- Table Style Editor (Title/Header/Data tabs)
- Multiline Style Editor
- Point Style Editor (19 marker shapes + named library)
- Surface Style Editor (3 tabs: Border & Triangles, Contours, Contour Labels)
- Tunnel Section Line Style Editor (measured line, deviation/hatch, center/labels/DR display)
- Alignment Style Editor (centerline, ticks, PI markers, arrows)
- Alignment Label Style Editor (stations, PI data, curve data)
- Civil Label Style Editor (unified system for all civil label types)
- Plot Style Tables (CTB)

43.5 Layout & Printing

- Multiple layouts per drawing with paper-space viewports
- Viewport border: ticks, ladder, crosses with coordinate labels
- Viewport grid, north arrow, scale bar
- Page setup: paper size, orientation, margins
- Print dialog with layout selector
- PDF export (single layout, all layouts, or from layout tab right-click)
- Plot style tables (CTB)

- Viewport setup (MVSETUP), clipping (VPCLIP), fade (VPFADE)

43.6 File I/O

- Native IQZ (compressed) and IQML (uncompressed) formats with streaming writer for large files
- DXF export (R12 / R2000) and import
- DWG export/import (via LibreDWG or ODAFileConverter)
- SVG export
- Image export (PNG, BMP)
- PDF export
- LandXML import/export dialog
- Recent Files menu (up to 10 entries, persisted)

43.7 User Interface

- Ribbon toolbar with Home, Draw, Modify, Annotate, Civil, Output tabs
- Quick Access Toolbar (QAT) with Save As and Properties toggle
- AutoCAD-style command line with aliases
- Object Inspector (TVirtualStringTree-based, categorized, multi-select, owner-drawn combos, keyboard navigation)
- Selection filter by entity type
- Layer Manager
- Civil Toolspace / Prospector panel with property grid
- Status bar with progress indicator
- Dark theme editor dialogs
- Layout persistence (panel sizes saved to settings.json)
- Keyboard shortcuts
- 3D orbit view, viewpoint presets, view twist

Index

Symbols

-HATCH (*command*), 257
3D Behaviour, 36
3D Object Viewer, 217, 223, 272
3D Polyline Properties, 66
3D View Commands, 36
3DFACE (*command*), 257
3DORBIT (*command*), 261
3DPOLY (*command*), 257

A

A parameter dialog, 350
Abbreviations, 135
ABOVE (*command*), 261
ADDCONTOURLABEL (*command*), 79, 269
ADDDXFFACES (*command*), 266
Adding a Custom WMS Provider, 39
Adding a Custom XYZ Provider, 39
Adding Data to a Surface, 150
ADDPI (*command*), 270
ADDPVI (*command*), 270
ADDREALTUNNELSECTION (*command*), 265
ADDSAMPLEENTITIES (*command*), 265
ADDSURFACEBOUNDARY (*command*), 79, 266
ADDSURFACEENTITIES (*command*), 79, 266
ADDSURFACEPOINTS (*command*), 79, 266
ADDTOGROUP (*command*), 273
ADDTUNNELSECTION (*command*), 265
AEC Survey Point Properties, 68
AEC Survey Points, 137
ALIGN (*command*), 77, 259
ALIGNFROMPOLYLINE (*command*), 79, 168, 270
Alignment, 313, 327
ALIGNMENT (*command*), 79, 168, 270
Alignment Commands, 79, 168
Alignment Editor, 172
Alignment Label Styles, 93, 169
Alignment Properties, 68, 167
Alignment Styles, 92, 168, 286
ALIGNMENTREPORT (*command*), 270
Alignments, 164, 269, 303, 318, 332
Alignments and Profiles, 164
ALIGNMENTSTYLE (*command*), 270
ALIGNPROPS (*command*), 79, 168, 270
ALIGNREVERSE (*command*), 79, 168, 270
ALIGNVIEW (*command*), 261
Ambient Settings, 135
ANGBASE (*system variable*), 263
ANGDIR (*system variable*), 263
ANGLE (*command*), 78, 262
Annotation, 45, 299
API reference, 358
Application Settings, 112, 281
ARC (*command*), 76, 257
Arc / Circle Properties, 67
ARC_CONT (*command*), 257
ARC_CSA (*command*), 257
ARC_CSE (*command*), 257
ARC_CSL (*command*), 257
ARC_SCA (*command*), 257
ARC_SCE (*command*), 257
ARC_SCL (*command*), 257
ARC_SEA (*command*), 257
ARC_SED (*command*), 257
ARC_SER (*command*), 257
Architecture, 63, 339
Architecture and design, 339
AREA (*command*), 78, 262
Area Calculation Algorithm, 182
ARRAY (*command*), 77, 259
ARROW (*command*), 257
Assemblies, 191, 271
ASSEMBLIES (*command*), 271
Assemblies & Corridors, 271

Assemblies and Corridors, 191
Assembly elements, 194
Association Highlighting, 176
Attaching a Cache, 233
ATTACHPOINTCLOUD (*command*), 273
ATTDEF, 72
ATTRIB, 72
Attributes, 70, 72
AUDIT (*command*), 265
AUNITS (*system variable*), 263
AUPREC (*system variable*), 263
AutoCAD compatibility, 108
Autocomplete, 75
Autoload, 310, 324
Autoload and trust, 310, 324
Automation, 275
Available RTL, 331
Axis Info Strip, 239

B

BACK (*command*), 261
Balance Volumes, 161
BCLOSE (*command*), 262
BCLOSE command, 72
BEDIT (*command*), 262
BEDIT command, 72
BELOW (*command*), 261
Benching, 206, 208
Best fit line, 127
BESTFITLINE (*command*), 127, 261
BEZIER (*command*), 257
BFL (*command*), 127, 261
BLADE (*command*), 275
BLADE command, 368
Blade Runner, 368
Blade Runner — buffers, 369
Blade Runner — command palette, 370
Blade Runner — find and replace, 370
Blade Runner — running scripts, 369
Blade Runner — sidebar, 370
Blade Runner — theme, 370
BLOCK (*command*), 262
BLOCK command, 71
Block Definitions, 70
Block Editor, 72
Block Manager, 72
BLOCKMANAGER (*command*), 262
BLOCKMANAGER command, 72
Blocks, 70, 262
BLOCKS (*command*), 262
Blocks and Layers, 73
Blocks in export formats, 73
Boundaries, 152
BREAK (*command*), 77, 259
Breaklines, 163
BUILDCORRIDOR (*command*), 271
BUILDGRADING (*command*), 221, 272
Building a corridor, 216
Building for other platforms, 367
Built-in Providers, 38
ByLayer Colour Picker, 243

C

CAD Commands, 257
Canvas / Drawing Defaults, 65
CHAMFER (*command*), 77, 259
CIRCLE (*command*), 76, 257
CIRCLE_2P (*command*), 257
CIRCLE_3P (*command*), 257
CIRCLE_CD (*command*), 257
Civil 3D objects, 111
Civil Commands, 266
Civil Drawing Tools, 47
Civil entities, 300
Civil Label Styles, 93, 289
Civil Object Layer Defaults, 287
Civil Object Layers, 136
Civil Settings, 135, 290
Civil Sites, 225
Civil Toolspace, 131, 133
Civil Toolspace and Settings, 131
Civil: IIqSurface, 365
Class, 325
Class and global reference, 325
Class reference, 311
Cleanup (*Super Tools*), 122
Clipboard, 52
CLOSE (*command*), 257
CLOUDSURFACE (*command*), 266
COGO points in DWG, 111
Colour, 61
Colour and Linetype Inheritance, 61
Colours, 225
Combined civil workflow, 318, 333
Command Line, 73
Command Reference, 75, 255
Command Suggestions, 75
Commands, 329
Commands and events, 329
Common Points File Format (*.cpt*), 187
Common Properties (*All Objects*), 65
Comparing Surfaces (*Surface Differencing*), 159
compatibility — summary, 368

Condition and ditch/bench persistence, 209
Conditional rows, 206
Contour Generation Algorithm, 152
Contour Label Placement, 164
Converting AutoCAD Points, 144
CONVERTPOINTS (*command*), 273
Coordinate forms, 298
Coordinate Transformation Commands, 80
Coordinate Transformations, 179, 184, 273
COPY (*command*), 77, 259
COPYCLIP (*command*), 259
Core types, 359
Corridor section style library, 204
CORRIDORAUTO (*command*), 271
Corridors, 191, 271
Course highlighting, 216
Courses, 201
CREATEGROUP (*command*), 273
CREATEPARCEL (*command*), 271
CREATEPROFILEVIEW (*command*), 80, 172, 270
CREATESECTIONSHEET (*command*), 80, 176, 271
CREATESECTIONSHEETDLG (*command*), 80, 271
CREATESURFACE (*command*), 79, 266
Creating a Block, 71
Creating a culvert, 222
Creating a Design Profile, 171
Creating a grading, 219
Creating a Layout, 105
Creating a Parcel, 176
Creating a Profile View, 170
Creating a Section Sheet, 175
Creating a Site, 227
Creating a Surface, 149
Creating a Tunnel Section Line, 181
Creating an Alignment, 165
Cross-Section Commands, 80, 175
Cross-Sections, 175, 270
Cross-Sections from a DEM, 158
CSTRANSFORM (*command*), 80, 273
CSTRANSFORM – Global Transformation, 184
CULVERT (*command*), 223, 272
Culvert commands, 223
Culvert report, 223
CULVERTEDIT (*command*), 223, 272
CULVERTQUANTITIES (*command*), 223, 272
CULVERTREPORT (*command*), 223, 272
Culverts, 217, 222, 272
CULVERTSECTION (*command*), 223, 272
Curb, 197
CURRENTLAYER (*command*), 78, 263
Cursor Modes, 56
Curves, 41
Custom commands, 305, 316, 329
Cut/Fill condition, 206
CUTCLIP (*command*), 259

D

Debug log, 114
Debugging plugins, 367
DECIMATESURFACE (*command*), 79, 266
Decimating (Simplifying) a Surface, 156
Decimation Algorithms, 157
Decorators, 315
Decorators and events, 315
DELETESURFACE (*command*), 79, 266
Deleting a Surface, 158
DELPI (*command*), 270
DELPVI (*command*), 270
DEMTOTIN (*command*), 266
design, 339
DESIGNPROFILE (*command*), 80, 172, 270
Deviation Colour Coding, 182
DGRID (*command*), 261
Diagnostics and Test Commands, 265
Dialogs, 316, 319, 330, 334
Dialogs and message boxes, 319, 334
DIMALIGNED (*command*), 262
DIMANGULAR (*command*), 262
DIMARC (*command*), 262
DIMBASELINE (*command*), 262
DIMCONTINUE (*command*), 262
DIMDIAMETER (*command*), 262
Dimension Properties, 46, 67
Dimension Styles, 47, 88, 285
Dimensions, 46, 262, 300
DIMJOGGED (*command*), 262
DIMLINEAR (*command*), 262
DIMORDINATE (*command*), 262
DIMRADIUS (*command*), 262
DIMSCALE (*system variable*), 263
DIMSTYLE (*command*), 262
disabling, 343
Discovering plugin commands, 367
Display modes, 224
Display Performance, 40
DIST (*command*), 78, 262

Ditch, 206
Ditch row kind, 207
DIVIDE (*command*), 77, 259
Docking a panel, 353
DR Display Modes, 182
Draw Order, 107, 136
Drawing, 257, 311, 325
Drawing Commands, 76
Drawing primitives, 299
Drawing Tabs, 34
Drawing Tools, 40, 278
Drawing Variables, 60, 85, 263, 283
DRAWINGPROPS (*command*), 265
Drawings, 31
DRAWORDER (*command*), 261
DRAWORDER command, 107
Drone Commands, 274
Drone Survey, 247
Drone Survey DEM, 249
Drone Survey grid, 250
Drone Survey keys, 251
Drone Survey navigation, 249
Drone Survey points, 250
Drone Survey send, 250
Drone Survey sessions, 251
Drone Survey surface, 250
DRONESURVEY (*command*), 247, 275
DUPLICATE (*command*), 259
DWG, 108
DWG backend, 110
DWG entity support, 109
DWG Export, 98
DWG limitations, 111
DWG survey export, 111
DWG versions, 109
DWGPROPS (*command*), 265
DXF Export, 97
Dynamic editing, 220
Dynamic editing and rebuild, 220

E

Earthwork Volume Report, 160
Earthwork Volumes, 175
EARTHWORKVOLUMES (*command*), 80, 176, 271
Edit, 277
Editable Fields, 242
Editing vertexes graphically, 167
Editing – the Culvert editor, 222
Editor display, 216
EDITPI (*command*), 270
Elements, 328
Elevation Band Legend, 137
Elevation Clipping, 234
Elevation Query, 155
Elevation Tracker, 155
ELEVTRACK (*command*), 266
ELLIPSE (*command*), 76, 257
enable, 348
Enabling, 343
Enabling the debug log, 367
Entering Coordinates, 81
Entity, 312, 327
Entry point, 359
enumerators, 366
ERASE (*command*), 77, 259
Esri JSON Import, 100
Events, 305, 316, 330
events, 315, 329
Events: IIqEventSource, 366
Events: IIqEventSource and IIqEventSink, 366
Examples by category, 298
Excavation-edge daylight hinge, 208
EXPLODE (*command*), 262
EXPLODE command, 72
EXPLODECONTOURS (*command*), 266
Exploding a Block, 72
EXPORTDXF (*command*), 257
EXPORTIMAGE (*command*), 257
Exporting, 95, 97
Exporting a Point Cloud, 233
Exporting and Importing, 95
Exporting Surfaces, 158
EXPORTPDF (*command*), 257
EXPORTPOINTCLOUD (*command*), 273
EXTEND (*command*), 77, 259
Extract, 221
EXTRUDE (*command*), 257

F

FACE2SURFACE (*command*), 266
File, 257, 277
File format, 82, 298
FILLET (*command*), 77, 259
Fills, 42
FLATTEN, 123
FLATTEN (*command*), 77, 122, 261
Format, 278
Format Settings, 83
Freeway-2x3x3.75, 214
FRONT (*command*), 261
Frozen interfaces, 341
Frozen interfaces and versioning, 341

Future scripting languages, 306

G

General, 279
Generating a Section Sheet, 241
Generic, 200
GeoJSON Export, 99
GeoJSON Import, 100
Geometry Elements, 165
Get Support, 12
Get Z, 126
Gizmo Settings, 37
global reference, 325
GPU rendering, 114
GPUTEST (*command*), 265
Grading, 217, 219, 271
GRADING (*command*), 221, 272
Grading commands, 221
GRADINGBALANCE (*command*), 221, 272
GRADINGEXTRACT (*command*), 221, 272
Grid, 35
GRID (*command*), 263
GRIDMAJOR (*command*), 263
GRIDTEXT (*command*), 263
GRIDUNIT (*command*), 263
Grips, 53
GRIPSIZE (*command*), 263
Ground Filter Panel, 239
Group Management, 146
Guided tour, 301

H

HATCH (*command*), 76, 257
Hatch Properties, 66
Hatching, 300
Hellenic Cadastre, 100
HELP (*command*), 265
Hidden triangles, 162
Hiding TIN faces, 162
Highway-2x2x3.5+median, 213
Horizontal Alignments, 165
How This Manual Is Organized, 6

I

ID (*command*), 78, 262
IDE support, 306
IIqAlignment, 365
IIqCommand, 363
IIqCommand and IIqCommandRegistry, 363
IIqCommandRegistry, 363
IIqControl, 362
IIqDialog, 360
IIqDrawing, 364
IIqEntity, 365
IIqEntity and IIqLayer, 365
IIqEventSink, 366
IIqForm, 361
IIqFormSink, 363
IIqHost, 360
IIqHostUI, 360
IIqLayer, 365
IIqPlugin, 360
IIqProfilePoint, 365
Image Export, 97
Image Properties, 45, 69
IMAGEATTACH (*command*), 257
Import Formats, 170
IMPORTDEM (*command*), 266
Importing, 95, 99
Importing a Point Cloud, 233
Importing Large Surfaces (*LandXML Fast Path*), 158
Importing Survey Points from CSV, 140
IMPORTPOINTCLOUD (*command*), 273
Inline Editing, 147
Inquiry, 262
Inquiry Commands, 78
Insert, 45
INSERT (*command*), 262
INSERT command, 71
Inserting a Block, 71
Inspecting, 224
Installed extensions (*autoload*), 306
Installing a plugin, 343
INSUNITS (*system variable*), 263
Interface Overview, 27
Intro, 309, 323
IQcad Basic, 13
IQcad Civil, 19
IQcad Drone, 24
IQcad Point Cloud, 23
IQcad Pro, 18
IQML Streaming Save, 156

J

JOIN (*command*), 77, 259

K

Keyboard Shortcuts, 276
KML / KMZ Export, 99
KML / KMZ Import, 101
Known limitations (*Python*), 306

L

label, 352
Labels, 269
LandXML Export, 98
LandXML Import, 100
Lane, 194
Layer Defaults, 284
Layer Dropdown, 59
Layer Manager, 59
Layers, 57, 85
LAYERS (*command*), 78, 263
Layout, 264
LAYOUT (*command*), 81, 264
Layout & Paper Space, 264
LAYOUT command, 105
Layout Commands, 80
Layout Persistence, 30
Layouts, 103
LibreDWG, 110
License Manager, 11
License modules, 12
Licensing, 9
lifetime, 341
LIMCHECK (*system variable*), 263
Limitations and tips, 321, 335
LIMITS (*command*), 263
LINE (*command*), 76, 257
Line Properties, 66
Lines, 41
Lines and Curves, 41
Linetype Inheritance, 61
Linetype Manager, 60
Linetypes, 57, 85
LINETYPES (*command*), 263
LIST (*command*), 78, 262
Live forms, 319, 334
Live tool windows, 351
Live update, 225
live window with a button that updates a list, 352
Loader states, 342
lock, 221
Locking a Viewport, 106
Long-running scripts, 305
LTSCALE (*system variable*), 263
LUNITS (*system variable*), 263
LUPREC (*system variable*), 263

M

Main Areas, 29
MAKELAYER (*command*), 78, 263
Map Background, 37
Map Settings Dialog, 38
MAPBG (*command*), 265
MAPBG Command, 40
MAPSETTINGS (*command*), 265
Material fills and overrides, 206
Materials (*assembly*), 201
MEASURE (*command*), 77, 259
MEASUREMENT (*system variable*), 263
Median, 196
Memory, 341
Memory and lifetime, 341
menu, 320, 335
menu contributions, 354
MERGESURFACES (*command*), 266
Mesh, 52
Mesh and Surface Conversion, 52
MESH2SURFACE (*command*), 262
message boxes, 319, 334, 349
MIRROR (*command*), 77, 259
MLEADER (*command*), 76, 257
Model Space, 103
Modify, 259
Modify Commands, 49, 76
Modify Tools, 278
Modifying Objects, 47
Module-level functions, 314
Module-level globals, 328
MOVE (*command*), 77, 259
Moving a culvert, 223
Moving an object to a site, 228
MTEXT (*command*), 257
Multi-Selection, 64
Multileader Properties, 67
MULTILINE (*command*), 76, 257
Multiline Properties, 66
Multiline Styles (*Ctrl+Shift+M*), 88
Multiple Drawings, 33
MVIEW (*command*), 81, 264
MVIEW command, 106
MVSETUP (*command*), 264
MVSETUP command, 106

N

Native DWG engine, 108
Native File Formats, 97
Navigating, 224
Navigation, 34
NEW (*command*), 257
New Drawing, 33
Number, 388

O

Object Inspector, 61
Object Layers, 136
Object list – show / hide / isolate, 224
Object Snapping (*Osnap*), 41
Object Viewer commands, 225
OBJVIEW (*command*), 225, 272
ODA File Converter, 110
OFFSET (*command*), 77, 259
OPEN (*command*), 257
Open / Import DWG, 99
Open a Drawing, 33
Open DXF, 99
Opening the manager, 343
Opening the Section Style Editor, 204
Opening the Style Editor, 242
Opening the viewer, 224
ORTHO (*command*), 263
ORTHOMODE (*system variable*), 263
OSMODE (*command*), 263
OSNAP in Edit Commands, 53

P

Page Setup, 105
PAGESETUP (*command*), 81, 264
PAGESETUP command, 105
PAN (*command*), 78, 261
Paper Size, 105
Paper Space, 103, 264
Parameter dialogs, 349
Parameter dialogs and message boxes, 349
Parcel Properties, 177
Parcel Styles, 91, 177, 287
Parcel Vertex Names, 177
PARCELCHECK (*command*), 271
PARCELREPORT (*command*), 271
Parcels, 173, 176, 271
PARCELTABLE (*command*), 271
PASCAL (*command*), 275
Pascal Scripting, 321
PASEVAL (*command*), 275
PASLOAD (*command*), 275
PASSCRIPTS (*command*), 275, 323
PASTECLIP (*command*), 259
PASTEORIG (*command*), 259
Patterns, 301
Pavement structure, 201
PCCLIP (*command*), 273
PCDIFF (*command*), 266
PCELEVTRACK (*command*), 273
PCGROUND (*command*), 273
PCSECTION (*command*), 273
PCSECTIONCOMMIT (*command*), 273
PCSECTIONS (*command*), 273
PCSECTIONSHEETSTYLES (*command*), 273
PCSECTIONTRACE (*command*), 273
PCSECTIONVIEW (*command*), 273
PCSURFACE (*command*), 273
PDF Export, 98, 108
PEDIT (*command*), 77, 259
Performance, 156
Performance & Rendering, 156
Perimeter, 388
Persistence, 243
PGZ (*command*), 126, 261
PI Detail Tables, 169
PI Details, 383
PI Table Editor, 166
PICKADD (*system variable*), 263
PICKAUTO (*system variable*), 263
PLAN (*command*), 261
PLANVIEW (*command*), 261
PLGETZ (*command*), 77, 126, 261
PLOT (*command*), 81, 264
PLOT command, 107
Plot Style Tables (*CTB*), 90
PLOTALL (*command*), 81, 264
PLOTALL command, 108
PLOTPDF (*command*), 81, 264
PLOTPDF command, 108
PLOTSTYLE (*command*), 263
Plotting, 103
PLSIM (*command*), 123, 261
PLSIMPLIFY (*command*), 77, 123, 261
Plugin Manager, 343
PLUGINS (*command*), 275
Plugins (*Native SDK*), 336
Plugins versus scripting, 339
POINT (*command*), 76, 257
Point Cloud Commands, 273
Point Cloud Display, 234
Point Cloud Export, 103
Point Cloud Import, 103
Point Cloud Management, 234
Point Cloud Section Sheet Styles, 242
Point Cloud Sections, 235, 237
Point Cloud Sections Along Alignment, 238

Point Cloud Sections and Sheets, 235
Point Clouds, 231
Point Group Context Menu, 57
Point Group Include Rules, 142
Point Group Point List, 143
Point Group Properties, 141
Point Groups, 140, 273
Point Manager, 145
Point Style (*Ctrl+Shift+D*), 85
Point Style Library, 139
POINTCHAINAGE (*command*), 270
Points, 328
POINTSTYLE (*command*), 263
POLYGON (*command*), 257
Polyline, 66
POLYLINE (*command*), 76, 257
Polyline and Multiline Properties, 66
Polylines (*Super Tools*), 123
Practical Guide: Which Transformation to Use, 188
Preferences, 112
Prerequisites, 38, 345
Print dialog, 107
Printing, 103
Profile, 328
Profile Commands, 79
Profile View Commands, 172
Profile View Properties, 171
Profile Views, 170
PROFILEPROPS (*command*), 80, 270
Profiles, 164, 270
Program Defaults Reference, 279
PROPERTIES (*command*), 263
Prospector Integration, 234
Prospector Style Columns, 176
Provider Persistence, 40
PURGE (*command*), 265
PURGEBLOCKS (*command*), 265
PURGELAYERS (*command*), 265
PURGELINETYPES (*command*), 265
PURGESTYLES (*command*), 265
PYEVAL (*command*), 275
PYLOAD (*command*), 275
PYPIP (*command*), 275
PYSCRIPTS (*command*), 275, 309
PYTHON (*command*), 275
Python scripting, 301
Python Scripting Reference, 307

Q

quantities, 223
QUICKPROFILE (*command*), 80, 172, 270

R

rebuild, 220
REBUILDSURFACE (*command*), 79, 266
RECOVER (*command*), 265
RECOVERCREATELAYERS (*command*), 265
RECTANGLE (*command*), 76, 257
Redo, 53
REDO (*command*), 259
REDRAW (*command*), 261
REGEN (*command*), 78, 261
reloading, 343
REMOVEFROMGROUP (*command*), 273
Rendering, 156
Responsiveness, 310, 324
Responsiveness and Esc, 310, 324
Result codes (*TIqResult*), 359
Retained Geometry, 40
RETAINEDGEOMETRY (*command*), 265
RETAINEDSTATS (*command*), 265
RETLOD (*command*), 265
Ribbon, 320, 335, 354
  Layer dropdown, 59
Ribbon & menu, 320, 335
Ribbon & menu contributions, 354
Ribbon Tabs, 30
Right-Click Behaviour, 57
ROTATE (*command*), 77, 259
Rotation Rings, 37
RUBBERSHEET (*command*), 80, 273
RUBBERSHEET – Local Non-Uniform Transformation, 186
Rulers, 35
runnables, 366
Running Pascal, 323
Running Python, 301, 309
Running Scripts, 81
Running scripts, 297
Runtime helpers (*uIqSdkRtl*), 359
Rural-2x3.5, 210
Rural-Narrow-2x3.0, 209

S

SAMPLEASSEMBLIES (*command*), 271
SAVE (*command*), 257
SAVEAS (*command*), 257
Saving, 33
SCALE (*command*), 77, 259
SCALELISTEDIT (*command*), 263
SCRIPT (*command*), 81, 275
Script editor, 368

Scripting, 275, 295
Scripting & Automation, 275
Scripting Commands, 81
Section style editable fields, 204
Section style persistence, 206
Section styles, 204
Section View Styles, 204
Sections, 173, 223
Sections and Parcels, 173
Sections and quantities, 223
SELECTALL (*command*), 259
Selecting a grading, 221
Selection, 54
Selection Feedback, 55
Selection Filter, 55, 64
Selection Methods, 55
SELECTIONCYCLING (*system variable*), 263
Set Properties, 70
Settings, 131, 263
Settings & Drawing Variables, 263
Settings Commands, 78
Settings file location, 115
Settings – Advanced, 114
Settings – Display, 113
Settings – DWG, 115
Settings – General, 113
Settings – Grid, 113
Settings – Ruler, 114
Settings – UCS Gizmo, 114
Settings – ViewCube, 114
settings.json, 112
Setup, 298
SETVAR (*command*), 78, 263
Shapefile, 101
Shapes, 42
Shapes and Fills, 42
Sheet Tabs, 146
Sheet Tabs & Group Management, 146
Sheets, 235
Shoulder, 195
show/hide, 221
Sidewalk, 198
Simplify polyline, 125
Sites, 225
Sites in file formats, 228
Sites in the Prospector, 227
Slab Thickness (*shared*), 239
Slope, 199
SLOPEPATH (*command*), 266
SNAP (*command*), 263
SNAPMODE (*system variable*), 263
SOLID (*command*), 257
SPLINE (*command*), 76, 257
Spot Elevations, 163, 269
Spot Elevations & Labels, 269
SPOTELEV (*command*), 79, 269
Standard sample library, 209
State persistence, 343
Station Format, 170
Step 1 – Create an Alignment from PIs, 382
Step 1 – Draw a Closed Polyline, 387
Step 1 – Import Survey Points, 378
Step 1 – Open a New Drawing, 374
Step 1 – write the source, 345
Step 2 – Add Curves at PIs, 382
Step 2 – Create a Parcel, 387
Step 2 – Create a Surface, 378
Step 2 – Import Points from CSV, 374
Step 2 – write the manifest, 347
Step 3 – Add Spiral Transitions, 383
Step 3 – build, 347
Step 3 – Set Parcel Name, 388
Step 3 – Set Parcel Name and Number, 388
Step 3 – View Contour Lines, 378
Step 3 – View Points in the Point Manager, 375
Step 4 – Change the Surface Style, 379
Step 4 – Create Point Groups, 376
Step 4 – install, 348
Step 4 – install and enable, 348
Step 4 – View Area, 388
Step 4 – View Area and Perimeter, 388
Step 4 – View Station Labels, 383
Step 5 – Add a Boundary to Clip the Surface, 379
Step 5 – Add User Data, 388
Step 5 – Change Point Styles, 376
Step 5 – Create a Profile View, 384
Step 5 – run it, 348
Step 6 – Add a Design Profile, 384
Step 6 – Add Spot Elevations, 380
Step 6 – Change the Parcel Style, 389
Step 6 – Export Points to Excel, 377
Step 7 – Create Cross-Sections, 385
Step 7 – Decimate a Large Surface, 380
Step 7 – Import Parcels from

LandXML, 389
Step 7 — Lock Points, 377
Step 8 — Edit Surface Points, 381
Step 8 — Transform Points, 377
Step 8 — View Earthwork Volumes, 386
Step 9 — Export to LandXML, 381
Step 9 — Import an Alignment from LandXML, 386
Stopping a script, 82
Strings, 366
Styles, 150
Super Tools, 119, 260
Super Tools Commands, 77
Super Tools menu, 119
Super Tools ribbon, 119
Supported Platforms, 5
Surface, 313, 327
Surface Commands, 78
Surface Conversion, 52
Surface Display, 150
Surface Display & Styles, 150
Surface Editing, 269
Surface Editing Commands, 162
Surface Properties, 68, 152
Surface Properties — Information, 152
Surface Properties — Statistics, 153
Surface Statistics, 155
Surface Styles, 86, 285
SURFACE2MESH (*command*), 262
SURFACEPROPERTIES (*command*), 79, 266
Surfaces, 266, 303, 317, 331
SURFACESTYLE (*command*), 266
SURFADDBREAKLINE (*command*), 162, 269
SURFADDLINE (*command*), 162, 269
SURFADDPOINT (*command*), 162, 269
SURFBALANCE (*command*), 161
SURFBUILDSETTINGS (*command*), 266
SURFCLEAREDITS (*command*), 162, 269
SURFDELETELINE (*command*), 162, 269
SURFDELETELONGEDGES (*command*), 162, 269
SURFDELETEPOINT (*command*), 162, 269
SURFDIFF (*command*), 266
SURFHIDETRI (*command*), 162, 269
SURFLISTEDITS (*command*), 162, 269
SURFMODIFYELEV (*command*), 162, 269
SURFMOVEPOINT (*command*), 162, 269
SURFRAISELOWER (*command*), 162, 269
SURFREGRID (*command*), 269
SURFSHOWTRI (*command*), 162, 269
SURFSWAPEDGE (*command*), 162, 269
SURFUNDO (*command*), 162, 269
Survey Points, 266
SURVEYPOINT (*command*), 266
SURVEYPOINTSTYLE (*command*), 266
SVG Export, 98
SWEEP (*command*), 257

T

TABLE (*command*), 257
Table Properties, 69
Table Styles, 87
TABLESTYLE (*command*), 263
TEXT (*command*), 76, 257
Text Properties, 67
Text Style (*Ctrl+Shift+T*), 86
Text Styles, 57, 60, 285
TEXTSIZE (*system variable*), 263
TEXTSTYLE (*command*), 263
The bundled samples, 357
The contract, 340
The grading object, 219
The iqcad module, 302
The manifest (*plugin.json*), 344
The object model, 310, 324
The out-parameter convention, 340
The plugin lifecycle, 342
The rich message box, 350
The same contributions from scripting, 355
The same dialogs from scripting, 351
The same live forms from scripting, 353
The vtable-interface ABI, 340
Threading, 341
Thumbnail list, 216
TIN Surfaces, 147
Tips, 238
tips, 321, 335
Toolbar Controls, 239
Toolbar Operations, 145
Topology container, 225
Transformation, 135
Transformation Report (*.trpt*), 187
Triangulation, 378
Triangulation Algorithm, 149
TRIM (*command*), 77, 259
Troubleshooting, 348
Trust, 343
trust, 306, 310, 324
Trust and the Plugin Manager, 343
TSLADDPOINT (*command*), 266
TSLCLOSEINVERT (*command*), 266

TSLCLOSETOLEVEL (*command*), 266
TSLDELETEPOINT (*command*), 266
TSLINE (*command*), 266
TSLSORT (*command*), 266
TSOPTIMIZE (*command*), 266
Tunnel Analysis, 179
Tunnel Filter Panel, 240
Tunnel Section Line Properties, 70, 181
Tunnel Section Line Styles, 90, 287
Tunnel Section Line – Measured Profile, 181
Tutorial 1 – your first plugin (*Free Pascal*), 345
Tutorial 1: Working with Survey Points, 374
Tutorial 2 – drawing geometry, 348
Tutorial 2: Working with TIN Surfaces, 378
Tutorial 3: Working with Alignments, 382
Tutorial 4: Working with Parcels, 387
Tutorials, 373
Type stubs, 306, 310
Type stubs and IDE support, 306

U

UCS Gizmo, 36
UCSICON (*command*), 261
Undo, 53
UNDO (*command*), 259
Undo and Redo, 53
Ungraded, 206, 208
Unified Style Managers, 94
UNITMODE (*system variable*), 263
Units, 135
UNITS (*command*), 78, 263
Units & Zone, 135
Updates, 12
Upgraded standard templates, 209
Urban-2x3.5+walks, 211
Urban-Boulevard-2x2x3.25, 212
Using the Gizmo, 36

V

Verbatim round-trip, 110
Versioning, 368
versioning, 341
Versioning and compatibility – summary, 368
Vertex Names, 177
View, 261, 277
View Commands, 77
Viewing a culvert in 3D, 223
Viewport Properties, 69
Viewport Scale, 106
Viewports, 106
VIEWTWIST (*command*), 261
Volume balance, 220
VOLUMEREPORT (*command*), 266
VPCLIP (*command*), 264
VPCLIP command, 106
VPFADE (*command*), 264
VPFADE command, 106
VPOINT (*command*), 261
VPSCALEBAR (*command*), 264
VPSCALEBAR command, 106

W

Welcome to IQcad, 5
What gets logged, 367
What IQcad Does, 5
Wipeout, 42
WIPEOUT (*command*), 257
WIPEOUTFRAME, 42
Writing a plugin in C or C++, 355

X

XLINE (*command*), 76, 257

Z

Zone, 135
Zoom, 35
ZOOM (*command*), 78, 261
ZOOMOBJECT (*command*), 261
ZW (*command*), 261