



by Ralph Grabowski

Customizing BricsCAD[®] V20

Copyright Information

Copyright © 2019 by upFront.eZine Publishing, Ltd.

All rights reserved worldwide.

This book is covered by copyright. As the owner of the copyright, upFront.eZine Publishing, Ltd. gives you permission to make one print copy. You may not make any electronic copies, and you may not claim authorship or ownership of the text or figures herein.

Visit the *Customizing BricsCAD* Web site at <http://www.worldcadaccess.com/ebooksonline/2015/04/cb15.html>. At this Web page, you will find editions of this book for BricsCAD V8 through to V19.

This twelfth edition is based on BricsCAD V20

16 December 2019

Technical Writer **Ralph Grabowski**

Technical Editing **Bricsys Staff**

All brand names and product names mentioned in this book are trademarks or service marks of their respective companies. Any omission or misuse (of any kind) of service marks or trademarks should not be regarded as intent to infringe on the property of others. The publisher recognizes and respects all marks used by companies, manufacturers, and developers as a means to distinguish their products.

This book is sold as is, without warranty of any kind, either express or implied, respecting the contents of this book and any disks or programs that may accompany it, including but not limited to implied warranties for the book's quality, performance, merchantability, or fitness for any particular purpose. Neither the publisher, authors, staff, or distributors shall be liable to the purchaser or any other person or entity with respect to any liability, loss, or damage caused or alleged to have been caused directly or indirectly by this book.

Summary of Contents

Full Table of Contents	v
----------------------------------	---

Part I — Customizing the BricsCAD Environment

Introduction to How to Customize BricsCAD	3
Adjusting BricsCAD's Settings	21
Changing BricsCAD's Environment	31
Adapting the User Interface To You	55

Part II — Working with the Customize Dialog Box

Introduction to the Customize Dialog Box	77
Customizing the Menu Bar & Context Menus	93
Customizing Toolbars and Button Icons	113
Writing Macros and Diesel Code	135
Customizing Ribbon Tabs and Panels	161
Customizing Keystroke Shortcuts, Aliases, & Shell Commands	183
Customizing Mouse, Double-click, & Tablet Buttons	201
Customizing the Quad	221
Customizing Rollover Properties	237
Customizing Multiple UIs with Workspaces	245

Part III — Other Customizations in BricsCAD

Designing Tool & Structure Panels	261
Creating Simple & Complex Linetypes	291
Patterning Hatches	303
Decoding Shapes & Fonts	315
Coding with Field Text	329

Part IV — Programming BricsCAD

Writing Scripts	359
Programming with LISP	367
Designing Dialog Boxes with DCL	399
Dabbling in VBA	431

Part V — Appendices

Command Summary	463
Summary of Variables & Settings	489
Concise DCL Reference	515
Concise LISP Reference	563

Full Table of Contents

Part I — Customizing the BricsCAD Environment

1. Introduction to How to Customize BricsCAD 3

The Many Ways to Customizing	4
Which Customization Do You Use?	5
Versions of BricsCAD	6
61 Tips for BricsCAD Users	7
For Further Reference	18
Reference and Tutorial Books	18
BricsCAD API References	19
DWG, DXF, and DWF References	20

2. Adjusting BricsCAD's Settings 21

Touring the Settings Dialog Box	22
Settings Dialog Box: Toolbar	23
<i>Categorized/Alphabetic Sorting</i>	23
<i>Show Differences</i>	24
<i>Dialog Configuration</i>	24
<i>Finding Variables</i>	25
<i>Export Settings</i>	26
<i>Exporting Variables</i>	27
Accessing Variables and Changing Values	28
<i>Variables Specific to Windows</i>	29
Changing Variables at the Command Prompt	29

3. Changing BricsCAD's Environment	31
Starting BricsCAD	32
Command Line Options.	32
Catalog of Command-Line Switches	34
<i>No Switch - Load Drawings</i>	35
<i>B Switch - Script Files</i>	35
<i>L Switch - No Logo</i>	35
<i>LD Switch - Application Load</i>	36
<i>S Switch - Search Support Paths</i>	36
<i>P Switch - User Profiles</i>	36
<i>PL Switch - Batch Plotting</i>	36
<i>T Switch - Template Files</i>	37
<i>Regserver and Unregserver Switches</i>	37
Other Startup Controls	37
Changing the Colors of the User Interface	38
Theme Color	38
<i>Background Color</i>	39
SETTINGS AT THE COMMAND LINE	40
<i>Changing Cursor Color and Size</i>	40
DISPLAY SETTINGS	41
Snap Marker Options.	43
Hyperlink Cursor Options	44
Dynamic Dimension Options.	45
Support File Paths	45
Summary of Files Settings	48
<i>Files (and Paths)</i>	48
<i>Project Paths</i>	50
<i>Printer Support Paths and Files</i>	50
<i>Templates Paths and Files</i>	50
<i>Tool Palettes Path</i>	50
<i>Dictionaries Section</i>	50
<i>Log Files Paths and Files</i>	51
<i>File Dialogs</i>	51
<i>Places Bar (Windows only)</i>	51
Reusing User Preferences	52
Launching the User Profile Manager.	52
<i>Using the Profile Manager</i>	53

4. Adapting the User Interface To You	55
Customizing the Command Line	56
The Parts of the Command Bar	56
<i>Resizing and Hiding the Command Line</i>	56
<i>Changing Command Bar Actions</i>	57
<i>Additional Command Line Variables</i>	59
<i>Even More Command Line Variables</i>	63
Customizing the Look of the Ribbon	64
Handling the Ribbon	64
<i>Related System Variables</i>	66
Customizing the Look of Drawing Tabs	66
<i>Related System Variables</i>	67
Customizing the Look From Control	67
LookFrom Command	69
<i>Related System Variables</i>	70
Maximizing the Drawing Area	71
Using Multiple Monitors	71
Customizing Other UI Elements	73

Part II — Working with the Customize Dialog Box

5. Introduction to the Customize Dialog Box	77
Touring the Customize Dialog Box	79
ABOUT CUI FILES	80
Customize's Menu Bar	81
ABOUT MAIN AND PARTIAL CUSTOMIZATION	81
<i>CUI Customization Files</i>	82
Search For Commands	84
Tabs of the Customize Dialog Box	85
Shortcut Menus	85
Apply and OK Buttons	86
Viewing Changes Made to Customize	86
<i>Additional Management Options</i>	88
Using Partial Menus to Customize BricsCAD Correctly	89
Setting Up a New Partial Menu	89
<i>Sharing Customizations</i>	90
<i>Removing Partial CUI Files</i>	92

6. Customizing the Menu Bar & Context Menus93

Modifying the Menu Bar	94
QUICK SUMMARY OF MENU COMMANDS & VARIABLES.	94
<i>Touring the Menu Tab</i>	95
QUICK SUMMARY OF MENU PARAMETERS.	96
<i>Opening and Closing Nodes</i>	97
<i>Gray Dots and Separator Lines</i>	97
Understanding Menu Title Conventions.	97
<i>Keyboard Shortcut - &</i>	98
<i>Dialog Box - ...</i>	98
<i>Menu Titles.</i>	98
Commands Use Macros.	99
<i>Cancel - ^c</i>	99
<i>Transparent - '</i>	99
<i>Internationalize - _</i>	99
<i>Enter - ;</i>	99
<i>Pause - \</i>	100
Editing the Help String.	100
Tutorial: Adding Menu Items	101
Tutorial: Deleting Menu Items.	103
Tutorial: Adding Tools to Menus	104
Context Menus	106
Tutorial: Customizing Context Menus	107
Tutorial: Sharing Menus	111
Importing AutoCAD Menus.	112

7. Customizing Toolbars and Button Icons113

QUICK SUMMARY OF TOOLBAR COMMANDS & VARIABLES.	114
Customizing the Look of Toolbars	114
Rearranging Toolbars.	114
<i>Tutorial: Dragging and Moving Toolbars.</i>	115
QUICK SUMMARY OF TOOLBAR PARAMETERS.	116
<i>Tutorial: Turning Toolbars On and Off.</i>	117
Making New Toolbars, and Modifying Them	118
Tutorial: How to Create A New Toolbar	118
<i>Tutorial: Alternative Method</i>	121
Adding Controls, Flyouts, and Separators	122
About Controls (Droplists)	122
<i>Tutorial: Adding Controls (Droplists) to Toolbars.</i>	123
<i>Customizing Controls (Droplists)</i>	124

About Flyouts.	125
<i>Tutorial: Adding Flyouts to Toolbars</i>	125
About Separators.	127
<i>Tutorial: Adding Separators to Toolbars</i>	127
Removing Buttons, Renaming and Deleting Toolbars	128
<i>Tutorial: Removing Buttons and Toolbars</i>	128
<i>Tutorial: Renaming Toolbars and Buttons</i>	129
Customizing Buttons	129
SIZING BUTTONS	130
Modifying Button Parameters	130
<i>Tutorial: Editing the Title Name and the Help String</i>	131
<i>Tutorial: Changing the Command Macro</i>	131
<i>Tutorial: Replacing Button Images</i>	132
8. Writing Macros and Diesel Code	135
QUICK SUMMARY OF METACHARACTERS IN MACROS	136
Simple Macros.	137
Transparent Commands in Macros	138
<i>Dashed Commands</i>	138
Options & User Input.	138
<i>Options</i>	138
<i>Pausing for User Input</i>	139
<i>Combining Options and Pauses</i>	139
<i>Other Control Keys</i>	140
Menu-Specific Metacharacters	141
Diesel Coding	141
About Diesel.	141
QUICK SUMMARY OF DIESEL FUNCTIONS	142
How to Toggle Check marks	143
<i>Toggling Grayouts</i>	144
Reporting Values of System Variables	145
Applying Variables Everywhere	147
<i>How to Add Units</i>	147
<i>How to Solve Check Marks that Conflict with Icons</i>	147
<i>How to Deal with Two Sysvars</i>	148
<i>Reporting Through Diesel</i>	149
<i>Formatting Units</i>	149
Formatting Diesel Output.	149
Formatting Numbers	149
<i>Fix</i>	149
<i>Index</i>	150

<i>Nth</i>	150
<i>Rtos</i>	150
<i>Formatting Angles</i>	151
Formatting Text	151
<i>Upper</i>	151
<i>StrnLen</i>	151
Variables in Diesel	152
Complete Catalog of Diesel Functions	152
Math Functions	152
Logic Functions	153
Conversion Function	155
String Functions	155
System Functions	157
Diesel Programming Tips	159
<i>Debugging Diesel</i>	159
<i>ModeMacro: Displaying Text on the Status Bar</i>	160

9. Customizing Ribbon Tabs and Panels161

QUICK SUMMARY OF RIBBON COMMANDS AND VARIABLES 162

The Structure of Ribbons	163
Tutorial: How to Add Panels to Ribbon Tabs	164
<i>Moving Panels</i>	166
<i>Copying Panels — Not</i>	167
<i>Removing Panels</i>	167
Tutorial: Making New Tabs	167
QUICK SUMMARY OF CONTEXTUAL TABS	168
<i>Adding Panels to A New Ribbon Tab</i>	169
<i>Moving Tabs Along the Ribbon</i>	169
<i>Making Copies of Tabs</i>	169
<i>Hiding Tabs in a Workspace</i>	169
Customizing Ribbon Panels	169
<i>Panel Design Tips</i>	171
Tutorial: Populating a new Panel	171
Catalog of Panel Elements	174
<i>Append Ribbon Panel / Insert Ribbon Panel</i>	175
<i>Delete</i>	175
<i>Add Launcher</i>	176
<i>Append Row / Insert Ribbon Row / Insert Row Panel</i>	176
<i>Append Break / Insert Ribbon Break / Append Separator</i>	177
<i>Append Split Button</i>	178
<i>Append Toggle Button</i>	180

10. Customizing Keystroke Shortcuts, Aliases, & Shell Commands	183
QUICK SUMMARY OF SHORTCUT KEYSTROKES	184
Tutorial: Defining Shortcut Keys	187
Tutorial: Editing & Deleting Keyboard Shortcuts	190
<i>Tutorial: How to Assign Multiple Commands</i>	190
Customizing Command Aliases.	191
Tutorial: Customizing Aliases.	192
Tutorial: Creating New Aliases.	192
<i>Tutorial: Editing & Deleting Aliases.</i>	193
BRICSCAD ALIASES SORTED BY COMMAND NAME	194
Rules for Writing Aliases	197
<i>Tutorial: Hand-Coding Aliases.</i>	197
Customizing Shell Commands	198
Tutorial: Editing Shell Commands	200
11. Customizing Mouse, Double-click, & Tablet Buttons	201
About Mice and Their Buttons	203
QUICK SUMMARY OF DEFAULT BUTTONS	204
<i>About the Pick Button</i>	205
<i>About the Right Button</i>	205
<i>About the Middle Button</i>	205
<i>Troubleshooting.</i>	206
Other Input Devices.	207
<i>Digitizing Tablets.</i>	207
<i>3D Mice.</i>	207
<i>Touch Pads.</i>	209
Defining Actions for Mouse Buttons	210
Tutorial: Button Assignment	210
Tutorial: Assigning Shortcut Menus to Buttons	212
<i>Tutorial: Writing Macros for Buttons</i>	213
Customizing Double-click Actions	213
<i>Changing a Double-click Action.</i>	214
<i>Making a New Double-click Action</i>	215
Defining Actions for Tablet Buttons.	216

12. Customizing the Quad	221
QUICK SUMMARY OF QUAD VARIABLES.	222
About The Quad	223
Step 1: Move Cursor Onto an Entity.	223
<i>Step 2: Expand the Quad</i>	224
<i>Step 3: Move Into Groupings</i>	224
Tutorial: Drawing with Quad	225
<i>Tutorial: Dimensioning with Quad</i>	225
Modifying the Quad's Behavior.	226
Customizing the Quad	226
<i>Tutorial: Customizing Quad Buttons</i>	227
Customizing Quad Tabs	228
<i>Where's My New Tab?</i>	230
<i>Tutorial: Turning On Quad Groups (Tabs)</i>	230
<i>Toggling Quad Tabs</i>	232
About Quad ENTity Filters	233
<i>Tutorial: Changing Entity Filters</i>	233
<i>How the Quad Works. Or, How Does It Know What Entity Is There?</i>	236
13. Customizing Rollover Properties	237
QUICK SUMMARY OF ROLLOVER PROPERTY SETTINGS.	238
QUICK SUMMARY OF ROLLOVER PROPERTIES	240
Customizing Rollover Properties	241
Tutorial: How to Change Properties Displayed by Rollovers	241
14. Customizing Multiple UIs with Workspaces	245
Workspace Customization Elements	247
Adding and Removing Workspaces.	247
<i>Removing Workspaces</i>	247
<i>About Insert Separator</i>	249
Toggling the Display of UI Elements	249
<i>Workspace Property Toggles</i>	249
<i>Show Menus.</i>	252
Toggling Visibility of UI Elements	252
<i>Toggling Menus</i>	252
<i>Toggling Toolbars</i>	253
<i>Toggling Panels</i>	253
<i>Toggling Ribbons.</i>	253
<i>Toggle the Quad</i>	253

Fine-Tuning UI Elements	253
<i>Workspace Properties for Menus</i>	254
<i>Properties of Toolbars</i>	254
<i>Properties of Panels</i>	255
<i>Properties of Ribbon Tabs</i>	257
<i>Properties of Quad Items</i>	258

Part III — Other Customizations in BricsCAD

15. Designing Tool & Structure Panels261

About the Tool Palettes Panel	263
QUICK SUMMARY OF VIEW OPTIONS.	264
Navigating Tools Palettes	265
Icon Customization	266
Palette Customization	267
Customizing Tools	268
<i>Customizing Tools Properties</i>	268
<i>Adding Programs and Macros to Tools</i>	272
Organizing Tools with Groups.	273
Creating Palette Groups.	274
Importing Tool Palettes from AutoCAD	276
<i>Sharing Tool Palette Groups by Exporting Them</i>	276
<i>Alternative Sharing Method</i>	277
Customizing the Structure Panel	278
<i>Structure Configurations</i>	279
Customizing the Structure Panel.	279
STRUCTURE OF .CST FILES	280
<i>Group/Sort Tab</i>	281
<i>Examining Rules</i>	282
<i>Constructing Rules</i>	283
<i>Show/Skip Tab</i>	289
<i>Options Tab</i>	290

16. Creating Simple & Complex Linetypes291

QUICK SUMMARY OF LINETYPE DEFINITIONS	292
About Simple and Complex Linetypes.	293
Commands Affecting Linetypes.	293
<i>Loading Linetypes</i>	294
<i>Scaling Linetypes</i>	294

System Variables Affecting Linetypes	295
<i>The Special Case of Paper Space</i>	295
<i>The Special Case of Polylines</i>	296
Customizing Linetypes	297
At the Command Prompt	297
<i>Testing the New Linetype</i>	298
Creating Linetypes with Text Editors	299
Linetype Format (.lin)	300
Line 1: Header	300
Line 2: Data	300
Complex (2D) Linetypes	300
Embedding Text in Linetypes	301
<i>Text</i>	301
<i>Text Style</i>	301
<i>Text Scale</i>	301
<i>Text Rotation</i>	302
<i>Absolute</i>	302
<i>X and Y Offset</i>	302

17. Patterning Hatches303

QUICK SUMMARY OF PATTERN DEFINITIONS..... 304

Where Do Hatch Patterns Come From?	305
How Hatch Patterns Work	306
<i>System Variables that Control Hatches</i>	307
Creating Custom Hatch Patterns	307
-Hatch Command	308
Hatch Command	309
Understanding the .pat Format	310
Comment and Header Lines	310
<i>Comment</i>	310
<i>Start of Definition</i>	310
<i>Pattern Name</i>	310
<i>Description</i>	311
The Hatch Data	311
<i>Angle</i>	311
<i>xOrigin and yOrigin</i>	311
<i>xOffset and yOffset</i>	311
<i>Dash1,.....</i>	312
Adding Samples to the Hatch Palette	312
Tips on Creating Pattern Codes	312

18. Decoding Shapes & Fonts.315

QUICK SUMMARY OF SHAPE DEFINITIONS. 316

Fonts, Complex Linetypes, and Shapes	317
SHX Fonts.	317
<i>About Fonts in BricsCAD</i>	318
Using SHX in Complex Linetypes	318
SHX in Shapes	318
SHX in GD&T	319
Shape Compatibility with AutoCAD	319
About Shape Files	319
The Shape File Format.	320
Header Fields	321
<i>Definition Start</i>	321
<i>shapeNumber</i>	321
<i>totalBytes</i>	321
<i>shapeName</i>	321
Definition Lines	321
<i>bytes</i>	321
Vector Codes	322
<i>Hexadecimal Conversion</i>	322
Instruction Codes.	323
<i>End of Shape - 0/000</i>	323
<i>Draw Mode - 1/001</i>	323
<i>2/002: Move Mode</i> -	324
<i>Reduced Scale - 3/003</i>	324
<i>Enlarged Scale - 4/004</i>	324
<i>Save (Push) - 5/005</i>	324
<i>Recall (Pop) - 6/006</i>	324
<i>Subshape - 7/007</i>	324
<i>X,y Distance - 8/008</i>	325
<i>X,y Distances - 9/009</i>	325
<i>Octant Arc - 10/00A</i>	325
<i>Fractional Arc - 11/ 00B</i>	326
<i>Bulge Arc - 12/00C</i>	326
<i>Polyarc - 13/00D</i>	327
<i>Flag Vertical Text Flag - 14/00E</i>	327

19. Coding with Field Text	329
FIELD COMMANDS & VARIABLES	330
Placing Field Text	331
Field Command	331
Fields in MText	333
Fields in Attributes	335
Changing Field Text	337
Double-clicking Fields in MText	337
Editing Fields in Attribute Definitions	338
Controlling the Way Fields Update	339
UpdateField Command	339
FieldEval Command	339
FieldDisplay Command	340
Another Field Text Example	340
<i>Updating the Field Text</i>	341
COMPATIBILITY WITH AUTOCAD FIELD CODES	342
Understanding Field Codes	343
Complete Field Code Reference	344
Groups	344
Metawords	344
Formatting	344
Complete Format Code Reference	345
<i>%tcn — Text Case</i>	345
<i>%lun — Linear Units</i>	345
<i>%dsn — Decimal Separator</i>	345
<i>%aun — Angular Units</i>	345
<i>%lwn — Line Weight units</i>	346
<i>%qfn — scale Factor</i>	346
<i>%ctn — ConverT</i>	346
<i>%ptn — PointTs (xyz coordinates)</i>	346
<i>%n — decimal places</i>	346
<i>%prn — display PRecision</i>	347
<i>%fnn — File Names</i>	347
<i>%byn — BYtes (file size)</i>	347
href - Hyperlinks	347
QUICK SUMMARY OF FIELD DATE AND TIME CODES	348
Date & Time Format Codes	349
Objects and Property Names	350
Properties in Common	350

Object Properties	350
<i>Arcs</i>	350
<i>Attribute Definition</i>	351
<i>Associative Dimensions</i>	351
<i>Blocks, Block Placeholders, and External References</i>	351
<i>Circles</i>	351
<i>Ellipses</i>	351
<i>Hatches</i>	352
<i>Leaders</i>	352
<i>Lines</i>	352
<i>Mtext</i>	352
<i>OLE (object linking and embedding) objects</i>	352
<i>Polylines</i>	352
<i>Polygon Meshes</i>	353
<i>Polyface Meshes</i>	353
<i>Raster Images</i>	353
<i>Regions</i>	353
<i>Rays and Xlines</i>	353
<i>Shapes</i>	353
<i>Single-line Text</i>	353
<i>Splines</i>	354
<i>Tables</i>	354
<i>Tolerances</i>	354
<i>Viewports</i>	354
<i>3D Faces</i>	354
<i>3D Polylines</i>	354
<i>3D Solids</i>	354
<i>Sheet SetS</i>	355
Named Object Properties	355

Part IV — Programming BricsCAD

20. Writing Scripts359

What are Scripts?	360
Drawbacks to Scripts	361
<i>Strictly Command-Line Oriented.</i>	361
Recording with RecScript	362
Writing Scripts by Hand	363
Script Commands and Modifiers	364
Script	364
RScript	364
Resume	364
Delay	365
Special Characters	365
<i>Enter - (space)</i>	365
<i>Comment - ;</i>	365
<i>Transparent - '</i>	366
<i>Pause - Backspace</i>	366
<i>Stop - ESC</i>	366

21. Programming with LISP367

The History of LISP in BricsCAD	368
BLADE Environment	368
Compatibility between LISP and AutoLISP	368
<i>Additional LISP Functions.</i>	369
<i>Different LISP Functions</i>	369
<i>Missing AutoLISP Functions.</i>	369
The LISP Programming Language	370
Simple LISP: Adding Two Numbers	370
LISP in Commands	371
Remembering the Result: setq	372
LISP Function Overview	373
Math Functions	373
Geometric Functions	374
<i>Distance Between Two Points</i>	374
<i>The Angle from 0 Degrees</i>	374
<i>The Intersection of Two Lines.</i>	375
<i>Entity Snaps</i>	375
Conditional Functions	375
<i>Other Conditionals</i>	376

String and Conversion Functions	376
<i>Joining Strings of Text</i>	376
<i>Converting Between Text and Numbers</i>	376
<i>Other Conversion Functions</i>	377
External Command Functions	377
GetXXX Functions	379
Selection Set Functions	381
Entity Manipulation Functions	381
Advanced LISP Functions	381
Writing a Simple LISP Program	382
Why Write a Program?	382
<i>The Id Command</i>	382
The Plan of Attack	382
<i>Obtaining the Coordinates</i>	382
Placing the Text	384
Putting It Together	385
Adding to the Simple LISP Program	385
Conquering Feature Bloat	386
<i>Wishlist Item #1: Naming the Program</i>	386
<i>Defining the Function - defun</i>	386
<i>Naming the Function - C:</i>	386
<i>Local and Global Variables - /</i>	387
<i>Wishlist Item #2: Saving the Program</i>	387
<i>Wishlist Item #3: Automatically Loading the Program</i>	387
<i>Wishlist #4: Using Car and Cdr</i>	388
Saving Data to Files	391
The Three Steps	391
<i>Step 1: Open the File</i>	391
<i>Step 2: Write Data to the File</i>	392
<i>Step 3: Close the File</i>	392
Putting It Together	393
<i>Wishlist #5: Layers</i>	393
<i>Wishlist #6: Text Style</i>	394
Tips in Using LISP	394
<i>Tip #1: Use an ASCII Text Editor</i>	394
<i>Tip #2: Loading LSP Code into BricsCAD</i>	394
<i>Tip #3: Toggling System Variables</i>	395
<i>Tip #4: Be Neat and Tidy</i>	395
<i>Tip #5: UPPER vs. lowercase</i>	396
<i>Tip # 6: Quotation Marks as Quotation Marks</i>	396
<i>Tip #7: Tabs and Quotation Marks</i>	397

22. Designing Dialog Boxes with DCL399

A QUICK HISTORY OF DCL	400
What Dialog Boxes Are Made Of	402
How DCL Operates	402
Your First DCL File	402
DCL Programming Structure	403
<i>Start Dialog Box Definition</i>	403
A QUICK SUMMARY OF DCL METACHARACTERS	404
<i>Dialog Box Title</i>	404
<i>OK Button</i>	404
<i>The Default Tile</i>	405
Testing DCL Code	405
LISP CODE TO LOAD AND RUN DIALOG BOXES	406
Displaying Data from System Variables	408
Adding the Complimentary LISP Code	410
Clustering Text	410
<i>Supplying the Variable Text</i>	411
<i>Leaving Room for Variable Text</i>	413
Fixing the Button Width	413
<i>Centering the Button</i>	413
Testing the Dialog Box	414
<i>Defining the Command</i>	414
Examples of DCL Tiles	416
Buttons	416
<i>Making Buttons Work</i>	417
<i>Check Boxes</i>	419
<i>Radio Buttons</i>	421
Clusters	424
<i>Columns and Rows</i>	424
<i>Boxed Row</i>	425
<i>Boxed Row with Label</i>	426
<i>Special Tiles for Radio Buttons</i>	426
Debugging DCL	427
Dcl_Settings	427
DCL Error Messages	427
<i>Semantic error(s) is DCL file</i>	427
<i>Dialog has neither an OK nor a CANCEL button</i>	427
<i>Error in dialog file "filename.dcl", line n</i>	428

<i>Dialog too large to fit on screen</i>	428
Additional Resources	428
23. Dabbling in VBA	431
QUICK SUMMARY OF VBA PROGRAM COMPONENTS	432
QUICK SUMMARY OF VBA COMMANDS IN BRICSCAD	432
Introduction to VBA	433
Accessing VBA Programs	433
Sending Commands	433
EMBEDDED OR EXTERNAL	434
Writing and Running VBA Routines	435
Displaying Messages	437
<i>Constructing Dialog Boxes</i>	438
<i>BricsCAD V20 Automation Object Model</i>	440
Object-Oriented Programming	441
Common Object Model	441
Object Browser	441
Line Entity	443
<i>Properties</i>	443
<i>Methods</i>	444
<i>Events</i>	444
Dialog Box with Code	444
Designing the Dialog Box	445
Adding the Code	450
<i>Clicking Cancel</i>	450
QUICK SUMMARY OF VBA DATA TYPES	451
QUICK SUMMARY OF VBA DATA TYPE RETURN VALUES	451
LastInput.Dvb	452
QUICK SUMMARY OF VBA STRING MANIPULATION	452
Conversion Routines	454
PointToString Conversion Function	454
<i>Private Function PointToString(vIn As Variant) As String</i>	455
QUICK SUMMARY OF VBA PREDEFINED CONSTANTS	455
<i>Dim sPt As String: sPt = vbNullString</i>	456
<i>Dim iPrecision As Integer</i>	456
<i>iPrecision = ThisDrawing.GetVariable("LUPREC")</i>	456

<i>If VarType(vIn) > vbArray Then</i>	456
<i>sPt = StringFromValueFixedDecimal(vIn(0), iPrecision) & ", "</i>	457
<i>sPt = sPt & StringFromValueFixedDecimal(vIn(1), iPrecision) & ", "</i>	457
<i>sPt = sPt & StringFromValueFixedDecimal(vIn(2), iPrecision)</i>	458
<i>End If</i>	458
<i>PointToString = sPt</i>	458
<i>End Function</i>	458
StringToPoint Conversion Function.....	458
<i>Dim sCoords() As String: sCoords = Strings.Split(sIn, ",")</i>	459
<i>If UBound(sCoords) = 0 Then</i>	459
<i>tmpPt(0) = Val(sCoords(0))</i>	459
Loading and Running LastInput.Dvb.....	459
QUICK SUMMARY OF VBA VARIABLE DECLARATIONS	460

Part V — Appendices

A. Command Summary	463
B. Summary of Variables & Settings.	489
C. Concise DCL Reference	515
QUICK REFERENCE OF DCL TILE NAMES	516
QUICK REFERENCE OF DCL ATTRIBUTES	518
<i>Exiting Dialog Boxes</i>	519
QUICK REFERENCE OF LISP FUNCTIONS FOR DIALOG BOXES	520
QUICK REFERENCE OF DIALOG BOXES DISPLAYED BY LISP FUNCTIONS	520
SUMMARY OF TILE REFERENCES	522
<i>Multiple Radio Buttons.</i>	527
<i>Multiple_Select</i>	536
<i>Errtile</i>	544
<i>Value and Mnemonic</i>	546
LISP Functions for Dialog Boxes	550
Dialog Boxes Displayed by LISP Functions	560
<i>Alert</i>	560
<i>Help</i>)	560
<i>Acad_HelpDlg</i>	560
<i>AcadColorDlg</i>	560
<i>Acad_TrueColorDlg</i>	560
<i>InitDia</i>	561
D. Concise LISP Reference	563
LISP Function Summary	564

PART I

Customizing the BricsCAD Environment

Introduction to How to Customize BricsCAD

If you are a messy sketcher like me, then you appreciate how computer software makes your work neater. For some drafters, that's what BricsCAD amounts to: a neater drafting machine.

The real power behind CAD (computer-aided design) is, however, its ability to be customized to the way **you** work. *Customize* is jargon for letting CAD do some of the drafting for you. This ranges from employing line patterns that are specific to your discipline to generating 3D staircases to fit between two floors — and more.

The benefit? You get your work done in less time, or, if you are a free-lancer, you get more work done in the same time.

The drawback? Customizing takes a bit of time:

- You need time to learn *how* to customize BricsCAD — that's what this tutorial book is all about
- Then you need more time to *create* the customization

Time isn't something most professionals have a lot of. I sometimes find myself doing repetitive editing under the false belief that it would take longer to write (and debug) a macro for automating the task than doing it by hand repetitively. So, I have this rule-of-thumb:

Write a macro (automation) when the same action is repeated more than three times.

There lies the responsibility of programmers to make automation easier for the end-user. Still, the time you invest in automation makes you a more productive BricsCAD user, even in the short run.

The information in this reference applies equally to BricsCAD running on Linux, MacOS, and Windows. When there are differences from Windows, the Linux and MacOS portions are indicated by gray colored text.

The Many Ways to Customizing

By my count, there are more than two dozen ways by which to customize BricsCAD. Some of these methods depend on the edition of BricsCAD installed on your computer: the Pro and Platinum editions provide more options than does the Classic edition.

Here I list the customization tasks supported by BricsCAD in alphabetical order, along with related three-letter file extensions. Those covered by this book are highlighted in blue:

BRX/TX	BricsCAD and Teigha runtime extensions, similar to AutoCAD's ARX Customizing the environment through command-line switches and other settings	Chapter 3
CUI	User interface elements like ribbon, LookFrom widget, and drawing tabs	Chapter 4
DCL	Dialog Control Language for customizable dialog boxes	Chapter 22
DWG	DraWinG for storing drawings and creating custom symbols — see <i>Inside BricsCAD</i>	
DXF	Drawing Interchange Format Field text	Chapter 19
FMP	Font Mapping	Chapters 2 and 18
PGP	Aliases and shell commands	Chapter 10
CUI	Customizable keystrokes, buttons, menus, toolbars, and ribbon	Chapters 5 - 14
LIN	Customizable simple and complex linetypes	Chapter 16
LSP	List processing language, similar to AutoLISP	Chapter 21
OLE	Object linking and embedding (not available in Linux or MacOS)	
PAT	Hatch patterns Quad cursor Rollover tooltips	Chapter 17 Chapter 12 Chapter 13
SCR	Script files	Chapter 20
SDS	Solutions Development System, similar to AutoCAD's ADS (SDS and ADS are deprecated*)	
SHP,SHX	Shapes and customizable text fonts**	Chapter 18
SLD	Slides	
TIP	Tip of the day *** Variables, Settings dialog box, and SetVar command	Chapter 2
VBA	Visual Basic for Applications	Chapter 23

* *Deprecated* is a programmer's term that means, yes, SDS is still in BricsCAD, but it is so old that Bricsys recommends you use BRX instead.

** BricsCAD cannot compile *.shp* files into *.shx*. Sorry!

*** Tip of the Day was removed from BricsCAD V15. Sorry, sorry!

Some methods of customization are designed for end-users, such as modifying toolbar macros, menus, and LISP routines, all of which you learn about in this tutorial book. Others are meant for professional programmers, such as BRX/TX and VBA.

Between the two levels, there are many other customization possibilities. For instance, the coding for hatch patterns is hard to figure out, but some enthusiastic users enjoy tinkering with them. You learn about all these, as well.

WHICH CUSTOMIZATION DO YOU USE?

That said, you need to make some decisions along the way. As you draft with BricsCAD, make a mental or written record of your work. In particular, you should **chronicle repetitive drafting tasks**, because these are prime candidates for customization. As a pioneer in the CAD world emphasized, “You should never have to draw the same line twice.” (In practice, we do, of course.)

Next, decide which of BricsCAD’s customization possibilities apply to the repetitive tasks you uncovered. Some solutions are obvious, such as writing *.lin* files for custom line patterns. Others are less obvious: to draw that 3D staircase, should you use a script file? (Perhaps.) A LISP routine? (Yes.) Or a menu macro? (Maybe.)

For these reasons, it’s good to become familiar with most of BricsCAD’s customization possibilities — even if you rarely use most of them. This way you craft a solution employing the best tools. You will know when to give the job over to a professional programmer, yet maintain intelligent oversight of the result.

A third solution is to learn about add-on programs available from amateur and professional programmers. Bricsys has an entire portion of its Web site dedicated to add-ons that works with BricsCAD at <https://www.bricsys.com/common/applications>.

The screenshot displays the 'Applications' page on the BricsCAD website. On the left, there are filters for 'COMPATIBILITY' (listing versions V20, V19, V18, V17, and V16 or lower with counts), 'CATEGORY' (listing various CAD-related categories like 3D, AEC, and Civil with counts), and 'PRICE' (listing Free and Paid with counts). The main content area shows a grid of application cards, each with a title, a brief description, and a 'Read more' link. The cards include: 'Express Tools' (described as well-known tools from AutoCAD), 'Civil Site Design' (an all-inclusive civil design package), 'PON CAD Scaffolding Design software' (a BIM-oriented tool for scaffolding), 'CADprofi Electrical' (a module for designing complex electrical installations), 'CADPower V20' (a domain-neutral productivity tool for DWG users), and 'GeoTools V20' (a geo-data CAD application for GIS data).

Applications that run on BricsCAD from third-party developers

As well, you may find utilities written for AutoCAD may well work in BricsCAD.

The bulk of the add-ons were written by programmers to solve their own problems with CAD. By knowing how to customize BricsCAD, you can modify their routines to suit your needs, which is a lot easier than writing it from scratch.

VERSIONS OF BRICSCAD

There are several versions of BricsCAD, and this ebook is for all of them:

- ▶ **BricsCAD Classic** — handles nearly all of the customizations listed in this book; does not support VBA programming (found in this book), as well as COM, BRX, and .Net (not in this book)
- ▶ **BricsCAD Pro** — supports all of the customizations described by this ebook, and handles most APIs provided by Bricsys
- ▶ **BricsCAD Platinum** — identical to Pro, as far as this book is concerned
- ▶ **BricsCAD BIM** and **BricsCAD Mechanical** — identical to Pro, as far as this book is concerned
- ▶ **BricsCAD Ultimate** — identical to Pro, as far as this book is concerned
- ▶ **BricsCAD Shape** is a free 3D design program that has limited customization ability (not in this book)

See this Web page for tables that list the differences between the three primary editions:

https://www.bricsys.com/en_INTL/bricscad/compare/. The image below shows a small part of the comparison table.

	Classic	Pro	Platinum
100% Real DWG performance	✓	✓	✓
Familiar CAD functionality	✓	✓	✓
Dynamic Blocks	✓*	✓*	✓*
Cloud Connectivity	✓	✓	✓
Network licensing	✓**	✓**	✓**
Perpetual licensing	✓	✓	✓
Full LISP, VBA, BRX (ARX) & .NET support	LISP only	✓	✓
Access to Third Party Applications	✗	✓	✓
3D Direct Modeling	✗	✓	✓

Comparing the three editions of BricsCAD

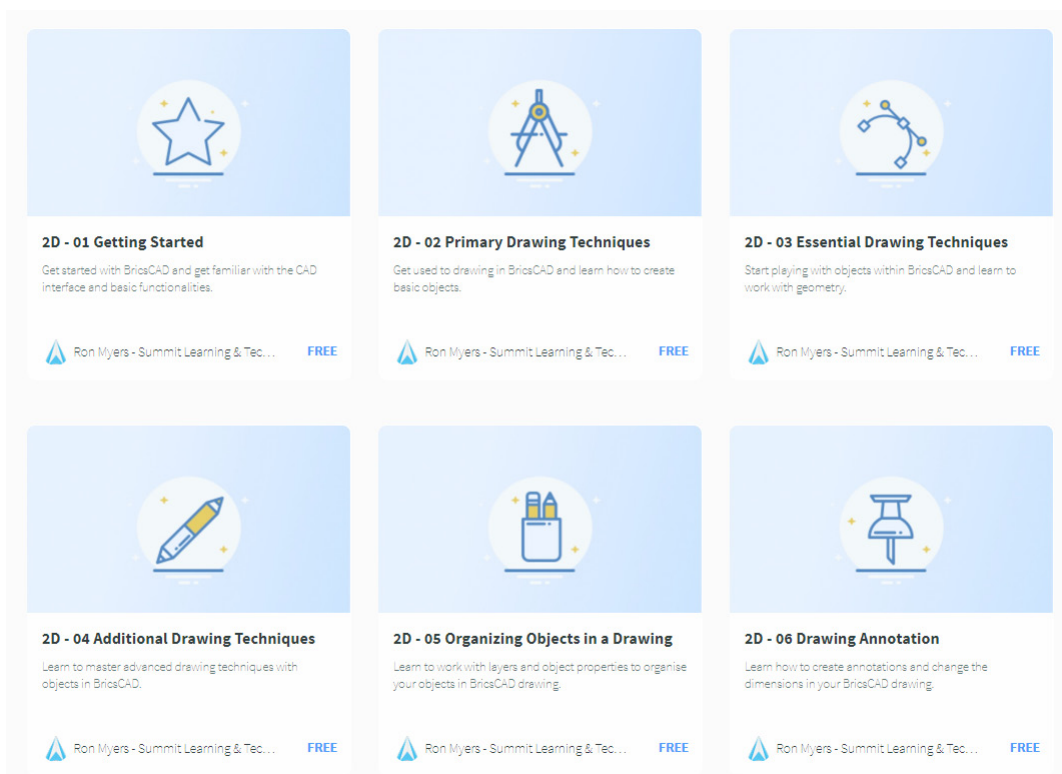
The free 30-day version of BricsCAD is the Ultimate edition, which consists of the Platinum version plus the BIM and Mechanical add-ons. It is handy for trying out all of the exercises in this book. Download your copy from https://www.bricsys.com/en_INTL/.

61 Tips for BricsCAD Users

TIP 1. To change menus, ribbon, and other interface elements, enter the **Customize** command. You can modify the currently active customization file, or load another CUI file (for example, from AutoCAD). *See Chapter 5.*

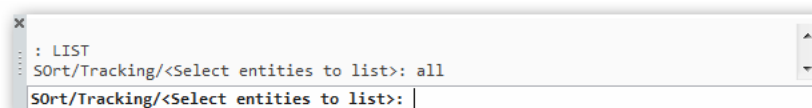
TIP 2. BricsCAD groups all settings variables in a single dialog box, the Settings dialog box. Enter the **Settings** command to open the dialog box. *See Chapter 2.*

TIP 3. Watch BricsCAD tutorial movies at <https://lessons.bricsys.com/>. Some of the ones dealing with customization were produced by the author of this book, Ralph Grabowski.



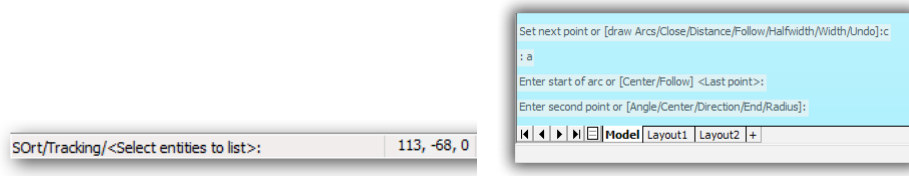
Web page listing tutorial videos for BricsCAD

TIP 4. To turn the display of the command bar on and off, enter the **CommandLine** command or press **Ctrl+9**. You can dock the command bar at the top or bottom of the BricsCAD application window or place it anywhere on any monitor attached to your computer. Hold down the **Ctrl** key (**Cmd** key on Macs) to prevent the command bar from docking.

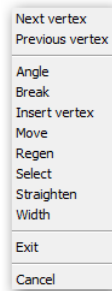


Docked command bar

When the command bar is off, then the prompts are displayed by the status bar, and in the drawing area. (See figures below)



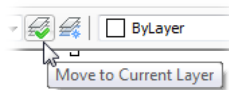
Left: Command prompt displayed on the status bar; right: ...and in the drawing area



TIP 5. When you launch a command, a Prompt menu displays all options current for the command. (See figure at left.) Use the mouse to select an option in the prompt menu.

The display of the prompt menu is controlled through the **PromptMenu** variable. In the Settings dialog box under **Program Options Display**, choose **Prompt Menu**, and then select a location for the prompt menus to display. Choose 'Don't display prompt menu' to suppress the display of prompt menus.

TIP 6. When you hover the cursor over a tool button, a tooltip is displayed, from which you can read a brief explanation of the tool's purpose. In addition, a line of help text is displayed on the status bar at the bottom of the BricsCAD application window.



Tooltip explaining the function of a button on the toolbar

TIP 7. When you press **F1** while a command is active, then BricsCAD displays help specific to that command. You can also access the complete command reference online from <https://help.bricsys.com/hc/en-us/categories/360000679494-Command-Reference>

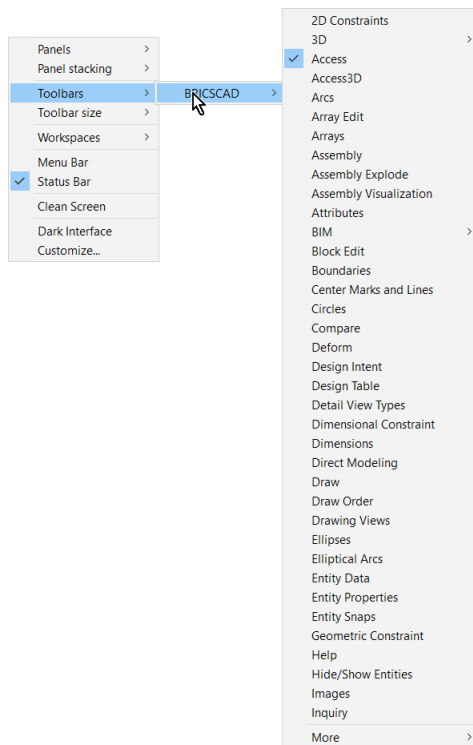
TIP 8. When you are familiar with typing AutoCAD commands and aliases at the command prompt, you can use exactly the same names in BricsCAD. These are called "aliases," and you can create, edit, and delete them with the Customize command.

TIP 9. The best way to open additional toolbars is to right-click a toolbar or the ribbon, and then click on **Toolbar > BricsCAD** to see a list of all available toolbars. (See figure.) Click on the name of the toolbar you want to open. Toolbars that are already open have a check mark next to them. When you click a toolbar that is open, it closes.

TIP 10. User interface elements, such as toolbars, ribbon, the command bar, and the Properties panel, can be placed anywhere on the screen, including on a second monitor. Press and hold the left mouse button to drag them to another location.

To prevent a toolbar or panel from docking against the edge of the BricsCAD window, press and hold the **Ctrl** key (**Cmd** key on MacOS) when positioning the item.

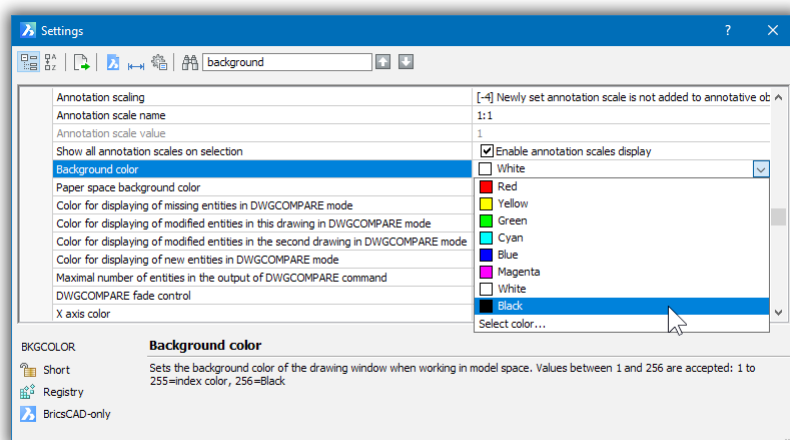
The **LockUI** variable determines if toolbars and panels are to be locked into place.



Accessing toolbars in BricsCAD

TIP 11. To change the color of the screen background, follow these steps:

1. Enter the **Settings** command
2. In the search field, enter “background color” and then press **Enter**
3. Select the color you want for the background



Alternatively, enter variable names at the command prompts. For instance:

BkgColor directly controls the background of the model space background

BkgColorPs directly controls the background of the paper space background

TIP 12. Use the tools on the **Inquiry** toolbar to measure the distance between two points, to find the area of closed entity, or to read the x,y,z coordinates of a point.

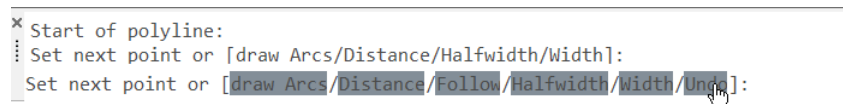


From left to right: Distance, Area, Mass Properties, ID Coordinates | List Entity Data, Drawing Status, Time Variables

TIP 13. To choose an option quickly during a command, just type the capitalized letter of the option’s name. When drawing a polyline, for instance, type **a** to start drawing arc segments and type **l** to switch back to drawing line segments.

```
: PLINE
Start of polyline:
Set next point or [draw Arcs/Distance/Halfwidth/Width]:a
Set end of arc or [draw Lines/Angle/Center/Direction/Halfwidth/Radius/Second point/Width]:l
Set next point or [draw Arcs/Distance/Halfwidth/Width]:
```

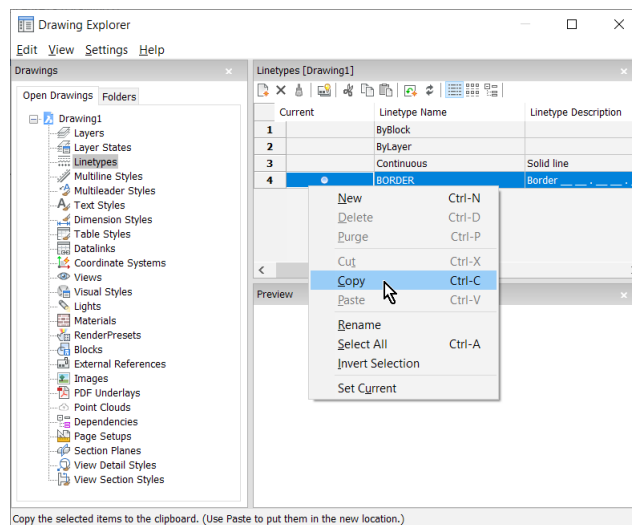
(NEW IN V20) You can use your cursor to pick the name of the option directly in command bar area.



Selecting a command option in the command bar with the cursor

TIP 14. BricsCAD’s drawing elements, such as layers, linetypes, text styles, dimension styles, and blocks, can be copied between open drawings using the Drawing Explorer. Follow these steps:

1. Enter the **Explorer** command, and then choose the element you want to copy, such as “Linetypes”
2. Right-click “Linetypes,” and then choose **Copy**
3. Switch to the other drawing, and then paste the item



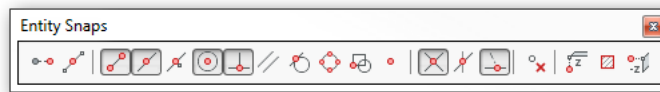
Pasting a linetype into another drawing

TIP 15. An *alias* for a command can be a single letter, such as **L** for “Line,” and it can be a different name, such as **axis** for “infiline”. To edit existing aliases or add new aliases, enter the **Customize** command, and then select **Command Aliases** tab on the Customize dialog.

TIP 16. To recall the previous selection set for the next command, choose **Previous Selection** in the Prompt bar or type **p** in the command bar to, for instance, move a previously copied selection set.

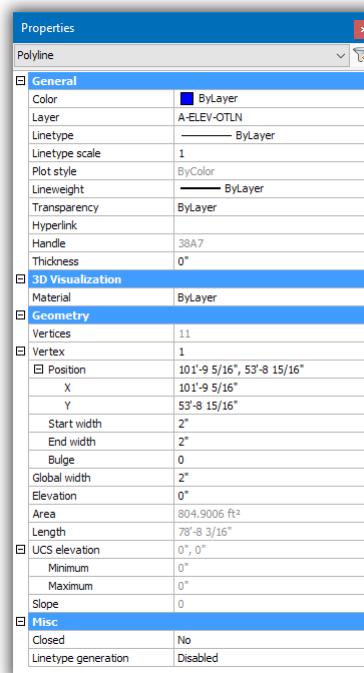
TIP 17. You can use any drawing you want as the *template* for future drawings. Template drawings can contain blocks, borders, and any other geometry, in addition to all your preferred settings. To set the default template drawing in the Settings dialog, go to **Program Options | Files | Templates | Template**.

TIP 18. Using entity snaps, you will draw faster and more accurately by snapping to their geometry, such as end points and mid points of lines. To quickly set and unset specific snap types, click a button on the Entity Snaps toolbar. The recessed button (or blue a border) indicates that the entity snap is active. (See figure below.) Click again to deactivate.



Toggling entity snaps

TIP 19. The Properties panel performs three tasks:



Properties being reported for a polyline entity

- When nothing is selected, then it sets the working properties for entities (color, layer, and so on)
- When one entity is selected, it edits the properties of the entity
- When two or more entities are selected, it edits the shared properties of them

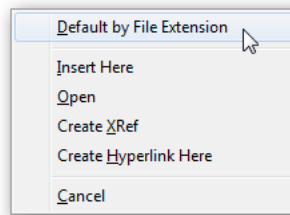
To open the Properties panel, enter the **Properties** command or else press **Ctrl+1**.

TIP 20. You can resize the height of any panel by dragging its top edge (when docked at the bottom) or its bottom edge (when docked at the top).

TIP 21. In each drawing you can define your own coordinate systems (UCS), which can then be saved and recalled as you need them. This is useful when you need to draw at angles other than the regular orthographic x,y-plane. To have BricsCAD automatically place the working plane, turn on **DUCS** (dynamic UCS) on the status bar.

(NEW IN V20) BricsCAD can also use DUCS when working with most 2D entities and dimensions.

TIP 22. Documents from other programs (e.g. text document and spreadsheets) can be dragged from the Windows Explorer window into your BricsCAD drawings (Windows only). To see the options available, drag the file from Explorer into BricsCAD using the **right** mouse button. In this case, the following dialog box is displayed:




Dialog box displayed by right-button dragging into BricsCAD

Double-clicking the inserted document opens the source application.

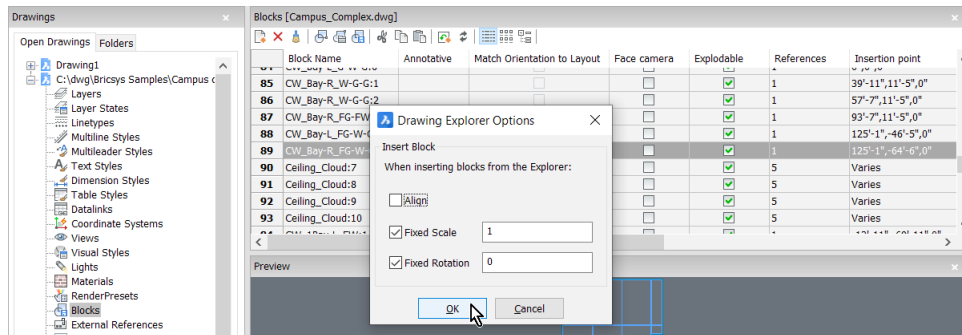
TIP 23. To toggle the display of the status bar, enter the **StatBar** command, or else press **Shift+F3**.

TIP 24. When trying to select one entity from many overlapping ones, you can cycle through possible snap points by pressing the **Tab** key repeatedly. The entity that is highlighted is the one selected.

TIP 25. Polylines consist of a chain of line and/or arc segments. Polylines have properties that ordinary lines lack, such as width, elevation and thickness. Draw polylines with the **PLine** command.

TIP 26. In the Drawing Explorer, you can see a thumbnail image of all blocks in the drawing. To insert the block in the current drawing, click the  **Insert** button.

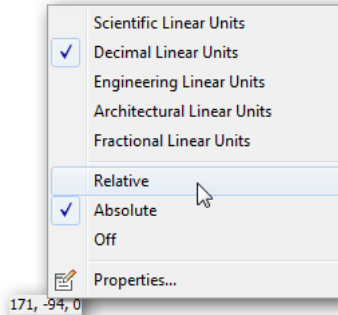
To preset parameters for blocks being inserted, right-click any block and then choose Options.



Presetting options for inserting blocks

TIP 27. Use the **Follow** option if you want to continue drawing lines, arcs, or polylines from the last point and in the same tangent direction.

TIP 28. To see the length and angle of the current line segment while drawing lines or polylines, right click the coordinates field in the status bar and then choose **Relative** from the context menu.



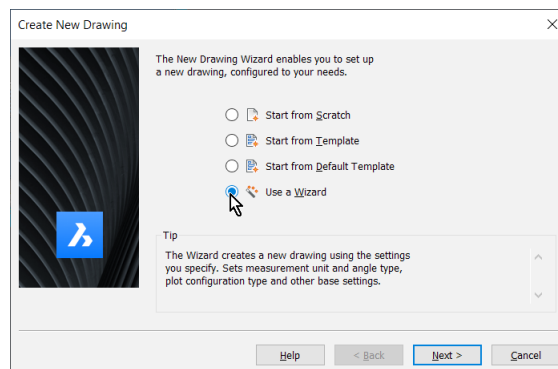
Turning on relative coordinates

TIP 29. Use the  **Match** tool (the ‘brush’) to apply the properties of one entity, such as color, linetype, and thickness, to other entities.

TIP 30. Click **ORTHO** in the status bar or press **F8** to toggle the orthogonal setting. Holding down the **Shift** key while you draw temporarily reverses the ortho setting — on or off.

TIP 31. Double-click the left end of the status bar to toggle the display of the command bar.

TIP 32. Choose **New Wizard** from the **B** menu, and then choose the **Use a Wizard** option to create a new drawing from scratch. The new drawing wizard guides you through all the basic drawing settings.



Starting a new drawing with the assistance of a wizard

TIP 33. BricsCAD provides dynamic view control using the mouse:


Mouse motion

- Hold down middle button
- Roll middle button (roller wheel)
- Hold down **Shift** key with middle button

BricsCAD Action

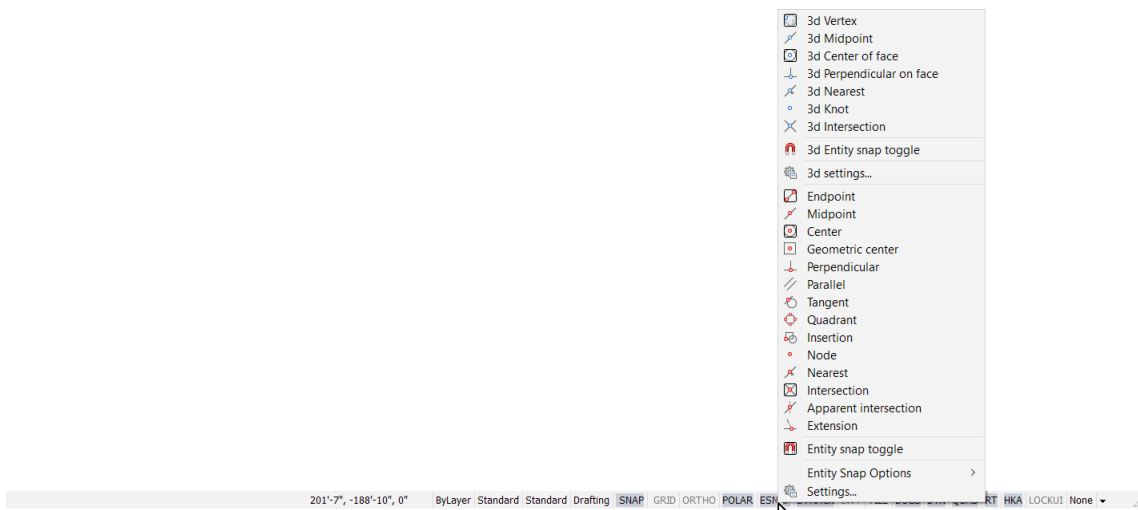
- Real-time pan
- Real-time zoom
- Real-time orbit (3D rotation)

TIP 34. The **Explode** command breaks complex entities, such as blocks and polylines, into their component pieces.

TIP 35. Purging unused definitions might reduce the size of your drawings dramatically. Enter the **Purge** command, then choose **All** to purge all unused definitions. Purging can also be done in the Drawing Explorer with the  Purge button.

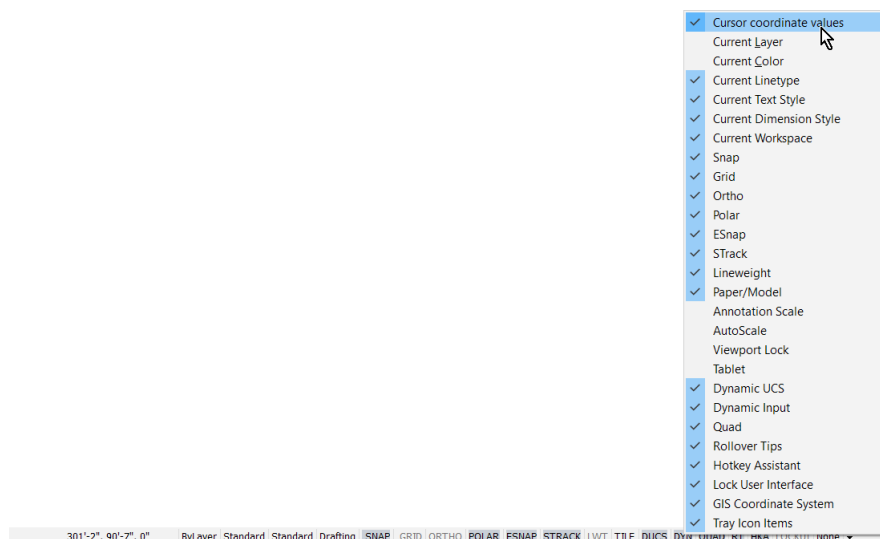
TIP 36. The **CENter** entity snap recognizes the center point of closed polylines — as well as circles and arcs. This works for even non-circular curves.

TIP 37. Many settings can be changed by clicking or right-clicking on each of the fields in the status bar at the bottom of the window.



Right-clicking a status bar item to access options

Click the small  down arrow at the right hand side of the status bar to control the display of the various fields in the status bar.





Toggling what gets displayed by the status bar

TIP 38. To copy or move entities between drawings, start the command in the source drawing, then switch to the target drawing when prompted for the displacement point. Press and hold the **Ctrl** key (Cmd key on Macs), then hit the **Tab** key to cycle through all open drawings.

TIP 39. The **Bisect** option of the **Ray** command bisects angles, lines, arcs, and polylines:

- ▶ **Vertex** option bisects an angle
- ▶ **Entity** option draws the ray perpendicular to the midpoint of the entity

TIP 40. The  **Save Block** tool in the Drawing Explorer saves the selected block to a separate drawing, which can then be inserted into other drawings.

TIP 41. In the Drawing Explorer, you can click in the **Current** column to make an item current. For example, to set a layer as the current layer, click in the Current column of that layer. A  blue dot appears in this column to mark the layer as current.

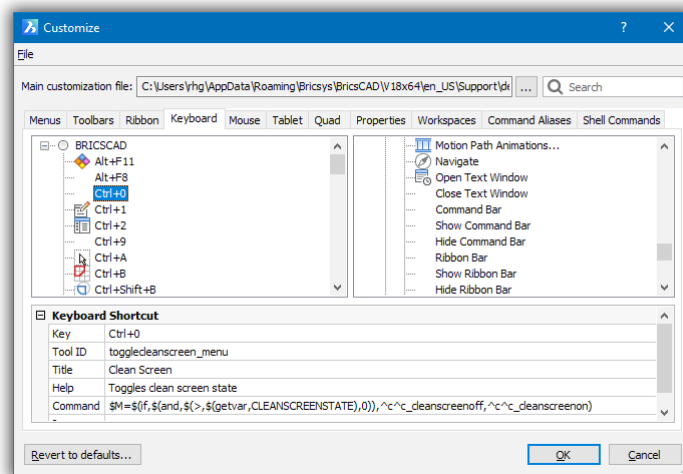
TIP 42. Use the **Mail** command to compose an email with the current drawing as an attachment.

TIP 43. How to turn an arc into a circle: use the Circle command's **turn Arc into circle** option.

TIP 44. Scroll bars allow to pan horizontally and vertically. To toggle the display of the scroll bars, enter the **ScrollBar** command or press **Shift+F4**.

TIP 45. Use the **Join** command to turn several entities with common endpoints into a single one: lines, polylines, 3D polylines, circular and elliptical arcs, splines, and helixes.

TIP 46. How to create keyboard shortcuts: enter the **Customize** command, and then choose the **Keyboard** tab. You can use any key on the keyboard and any function key, together combinations with **Shift**, **Alt**, and **Ctrl**. See Chapter 10 more information.



Customizing keyboard shortcuts

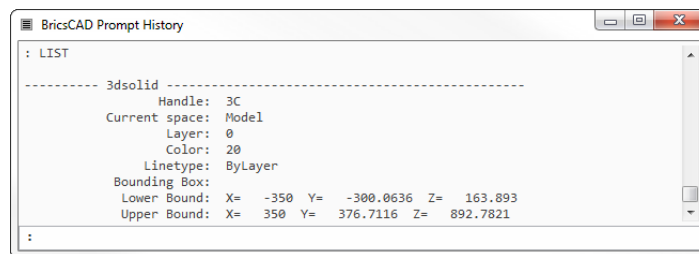
Linux and Windows versions of BricsCAD use the same **Cmd** and **Alt** keys for shortcuts. In the MacOS version, **Ctrl** and **Alt** are not used; instead, mentally map them to the Mac's **Cmd** and **Option** keys:

<u>Linux, Windows</u>	<u>MacOS</u>
Ctrl	Cmd
Alt	Options

TIP 47. The **Plan** command restores the plan view of the current coordinate system, as well as in the World Coordinate System (WCS) or any saved User Coordinate System (UCS). From the **View** menu, choose **Plan View**.

If the **UcsFollow** variable is on, plan view is restored automatically when the coordinate system changes.

TIP 48. Select an entity, then type **List** in the command bar to display the information about the selected entity in the Prompt History window.



List command displaying its report in the Prompt History window

TIP 49. Block attribute data is exported to external data files with the **DataExtraction** command, which can then be imported into spreadsheets and database programs — as well as brought back into the drawing as a table using the **Table** command's From Data option.

TIP 50. Select the **Explode** option of the Insert Block dialog to break a block into its component pieces upon insertion. Use the **BEdit** command to edit blocks in the block editor environment, and the **RefEdit** command to edit externally-referenced drawings in the reference editor.

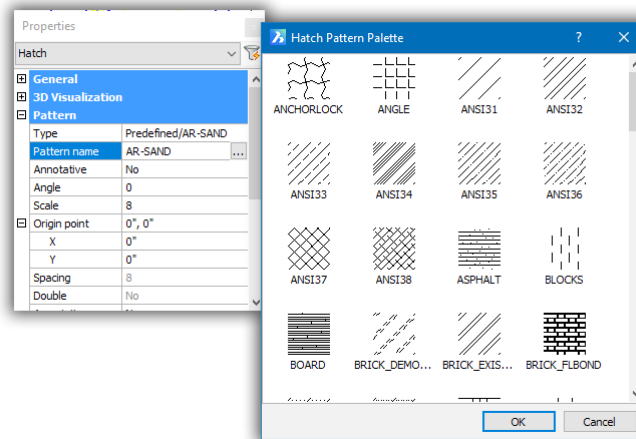
TIP 51. The **QSelect** command creates a selection set using properties, such as layer, color, and/or entity type.

TIP 52. BricsCAD automatically saves your work at a specified time interval. In the Settings dialog, under **Program Options | Open and Save**, choose **Save time interval**, then specify the time interval in minutes.

In the Settings dialog, under **Program Options | Files** you can specify the folder in which to store automatic saves.

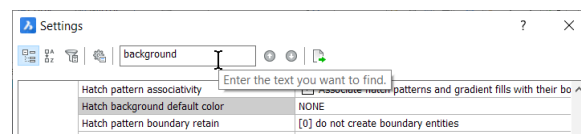
TIP 53. You can change the pattern style of an existing hatch in the Properties panel:

1. Select the hatch
2. Choose **Pattern name** in the Properties panel
3. Click the **Browse** button to select a different hatch pattern in the Hatch Pattern palette



Changing a hatch pattern quickly

TIP 54. You can easily find any variable or program setting using the search field in the Settings dialog.



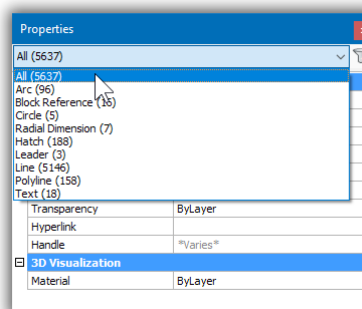
Searching for variable names in the Settings dialog box

To open the Settings dialog, enter the **Settings** command. Click the binoculars button to define the search target: variable names, titles, and/or help text.

TIP 55. To open a drawing in BricsCAD quickly, drag it from File Explorer (Finder on Macs) to the title bar of BricsCAD.

To insert the drawing as a block, drag it instead into the drawing area.

TIP 56. To get a count of entities in the drawing quickly, enter **Ctrl+A** to select all entities, and then cast a glance at the Properties panel's droplist.

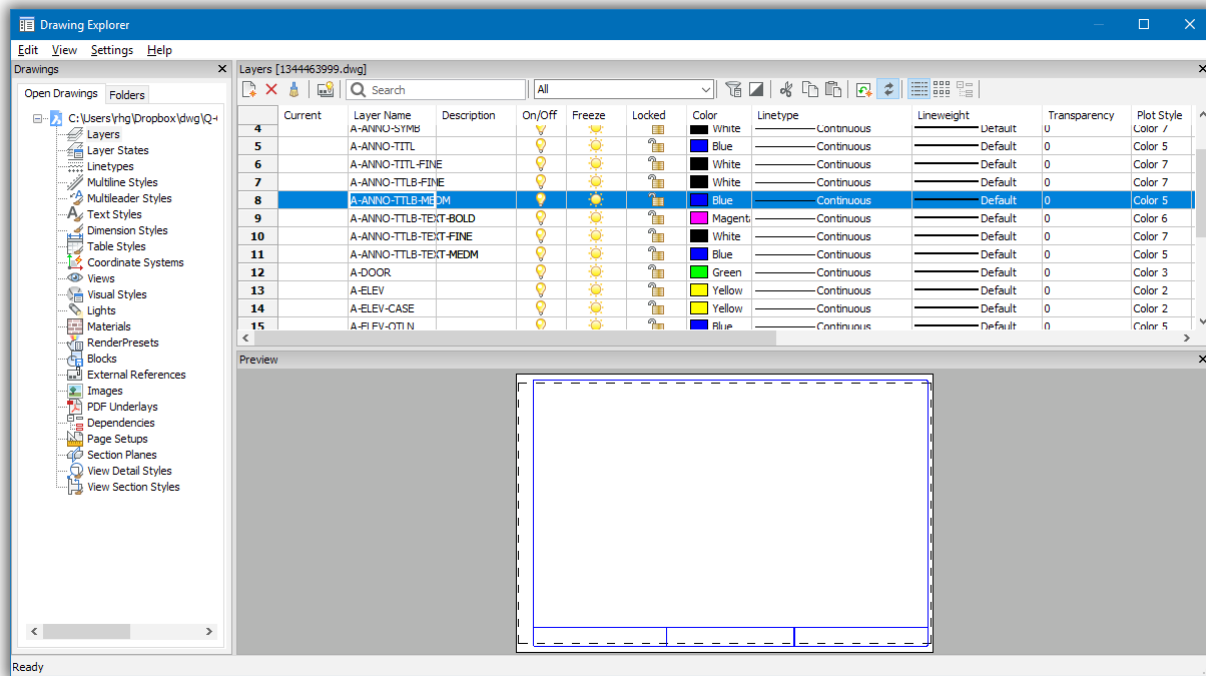


Counting the types of entities in the drawing

TIP 57. To make a layer current by selecting an entity on that layer, click the  **Set Layer by Entity** tool button on the Layer toolbar.

TIP 58. To override the currently active entity snaps while you are drawing, press and hold the **Shift** key, then right-click and choose an entity snap from the context menu, which will then be used to specify the next point.

TIP 59: To view the contents of each layer individually, open the Drawing Explorer to the Layers node, and then click on each layer name.



TIP 60: To restore all of the user interface to its default state, click the **Revert to defaults** button in the Customize dialog box.

TIP 61: To learn more about customization, read this book!

For Further Reference

This book provides much documentation and many tutorials for customizing most aspects of BricsCAD, but even at 600 pages it doesn't cover everything. Here are additional references:

REFERENCE AND TUTORIAL BOOKS

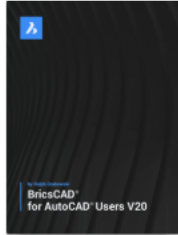
Bricsys offers these titles for learning how to operate BricsCAD. Each is a PDF file that you download for free from <https://help.bricsys.com/hc/en-us> and then scroll down to "Download Free Books."

Inside BricsCAD — teaches newcomers how to draft with BricsCAD through many step-by-step tutorials. Nearly 400 color pages.

BricsCAD for AutoCAD Users — clearly explains the similarities and differences between BricsCAD and AutoCAD. Over 300 color pages.

Customizing BricsCAD — additional copies of this book are available from Bricsys.

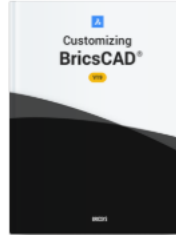
Download free books and guides



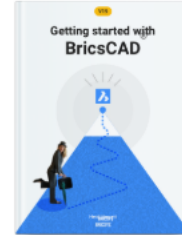
BricsCAD for AutoCAD users
330 pages - English
by Ralph Grabowski
[Download >](#)



Inside BricsCAD
386 pages - English
by Ralph Grabowski
[Download >](#)



Customizing BricsCAD V19
560 pages - English
by Ralph Grabowski
[Download >](#)



Getting Started with BricsCAD
107 pages - English
by Bricsys
[Download >](#)

BRICSCAD API REFERENCES

Bricsys provides online references for programmers at https://bricsys.com/bricscad/help/en_US/V20/DevRef/index.html. The Web site provides information on the following APIs:

- ▶ LISP (LISt Processing)
- ▶ Additional LISP functions, such as VLA, VLAX, and VLE
- ▶ DCL (Dialog Control Language) — incomplete
- ▶ DIESEL (Direct Interpretively Evaluated String Expression Language)
- ▶ VBA (Visual Basic for Applications) — incomplete
- ▶ COM (Component Object Model)
- ▶ BRX (BricsCAD Runtime eXtension) and TX (Teigha eXtension; formerly DRX) — incomplete
- ▶ .Net
- ▶ SDS (Softdesk Development System; deprecated)

DWG, DXF, AND DWF REFERENCES

The Open Design Alliance provides the specifications of the **DWG** file format, as they understand it to be. Covers Release 13 through 2013. Download in PDF format (free; 262 pages) from https://www.opendesign.com/files/guestdownloads/OpenDesign_Specification_for_.dwg_files.pdf
Autodesk does not document DWG.

Autodesk provides references for the **DXF** format (drawing interchange format). Covers Release 2012. Download in PDF format (free; 270 pages) from http://images.autodesk.com/adsk/files/autocad_2012_pdf_dxf-reference_enu.pdf

Adjusting BricsCAD's Settings


The Settings dialog box is BricsCAD's control central. This is where you make adjustments to the settings of over 1.000 variables. BricsCAD uses variables (settings) to control, change, and remember the states of drawings, dimensions, the user interface, and the program itself.

For instance, through this dialog box you change the background color of the drawing area, or specify the name and path for the default template file. When you want to change the default radius of fillets, you select the Settings option of the Fillet command: this action opens the Settings dialog box at the Chamfer/Fillet section: you are changing the value of variable FilletRad.

CHAPTER SUMMARY

The following topics are covered by this chapter:

- Touring the Settings dialog box
- Understanding system variables and preferences
- For the complete list of variables and preferences, see Appendix A
- Additional system variables and preferences

Many BricsCAD variables have names that are the same as in AutoCAD and IntelliCAD, such as FilletRad. But BricsCAD also has its own set of unique variables that it names “preferences,” such as **BkgColor** to set the background color of the drawing area. BricsCAD-only variable names are tagged  **BricsCAD-only** in the Settings dialog box.

Certain commands take you directly to the related section of the Settings dialog box, such as the **DdPMode** for setting the display style of points. Other commands have a **Settings** or **Options** option that does the same thing, such as the Fillet command’s Settings option that I mentioned earlier.

A handy way to get to some settings is by right-clicking a button on the status bar, and then choosing Settings from the shortcut menu. For example, you can change the settings of grid, snap, and Quad in this way.

Touring the Settings Dialog Box

To access the dialog box, enter the **Settings** command. When the dialog box appears, notice that it sorts variables into groups:

- ▶ **Drawing** — settings affecting how drawings are created
- ▶ **Dimensions** — settings affecting the styles of dimensions
- ▶ **Program Options** — settings affecting how the program looks and operates

Plus, there are additional settings for the optional add-ons:

- ▶ **Compare** — setting for 3D model comparison, part of the Mechanical add-on
- ▶ **Sheet Metal** — settings for sheet metal design, part of the Mechanical add-on
- ▶ **Communicator** — settings for file translation, part of the Communicator add-on
- ▶ **Standard Parts** — settings for sheet metal design, part of the Mechanical add-on
- ▶ **BIM** — settings for building information modeling, part of the BIM add-on

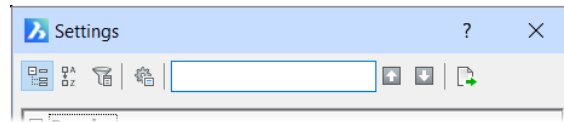


Settings dialog box

First, we examine the functions of the Settings dialog box’s toolbar, and then we tackle the body of the dialog box.

SETTINGS DIALOG BOX: TOOLBAR


The toolbar is at the top of the Settings dialog box. It changes the way the Settings dialog box presents information and accesses utility commands.



Toolbar in the Settings dialog box

Let's take a look at the functions of the toolbar buttons, beginning at the left end.

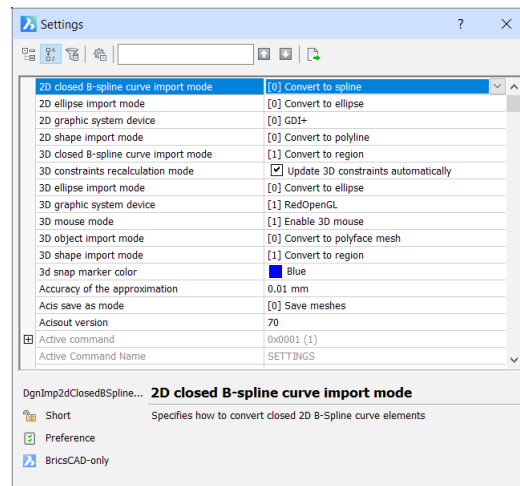
Categorized/Alphabetic Sorting

 You can search for settings by browsing through categories, or you can look through them alphabetically, or you can use the search field. I tend to use the search field, which I describe later.

The first two buttons on the toolbar switch the dialog box's listing of variables between Categorized and Alphabetic modes:

- ▶ **Categorized** mode is illustrated on the facing page, and lists variables in related groups
- ▶ **Alphabetic** mode is shown below, and lists them alphabetically by description (rather than by name)

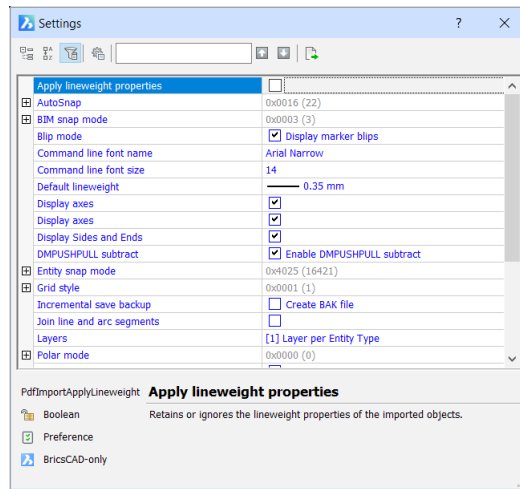
Because BricsCAD lists variables alphabetically by their *description* (instead of by their actual names), “2D closed B-spline curve import mode” is the first one on the list, even though its variable name is DgnImp2dCloseBSpline.



Settings dialog box displaying settings in alphabetical order

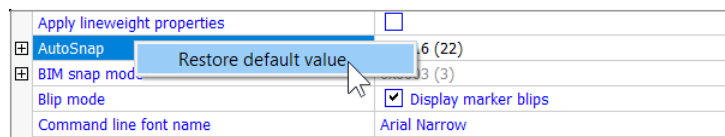
Show Differences

The **Show Differences** button clears the dialog box of all settings, except for those whose values have changed from the initial (default) values. Make a change to a variable, and it will show up here. This is handy for figuring out what might have changed.



Settings dialog box showing differences from initial values

We generally don't know what the default value of a variable is, and so BricsCAD helps us out here. To change a value back to the default, right-click its name, and then choose **Restore Default Value** from the shortcut menu.

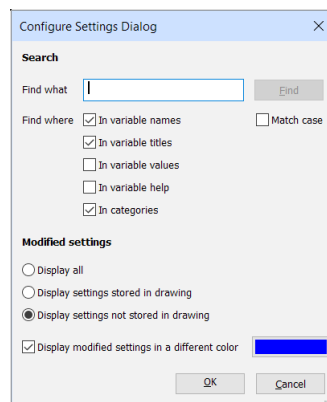


Changing a variable back to its default value

Notice that the variables with changes are shown in blue. The same happens in the regular dialog box. You can change this color and the kinds of variables listed, as described next.

Dialog Configuration

The **Dialog Configurations** button lets you search for variables, configure what is searched, and how the differences are shown. When you click the button, then BricsCAD displays this dialog box:



BricsCAD searches for the values of variables when the **In variable values** option is turned on.




Search. The Search options are useful in limiting where BricsCAD performs its searches. I tend to leave **In Variable Names** and **In Variables Titles** turned on, with the others turned off.

While you can use the **Find What** field that's right here in this sub-dialog box, it doesn't display color change or do real-time searches, and so I don't find it useful. Use the toolbar's **Search** field instead, as described next.

Modified Settings. The Modified Settings section lets you change the color of the changed variables, as well as to determine which variables are displayed by the Show Differences dialog box.



- ▶ **Display All** — lists all variables whose values have changed
- ▶ **Display Settings Stored in Drawing** — lists only those variables that affect the current drawing
- ▶ **Display Settings Not Stored in Drawing** — lists those that affect all drawings

Finding Variables

   The best item in the toolbar is the real-time search field. It lets you directly access variables when you know the first few letters of their names or descriptions. I find this the easiest way to navigate the 900+ entries in the dialog box.

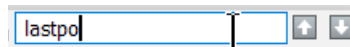
As you type into the search field, BricsCAD immediately jumps to the first item that matches the letters. For instance, when entering "lastpoint," the following occurs:

- Type **l** — focus jumps to Insertion Unit, because the description contains the letter 'l'
- Type **la** — focus jumps to Unit Mode, because 'la' is in "displayed" in the description
- Type **las** — focus jumps to Text Angle, because of 'las' in "last" of the description
- Type **lastp** — focus stays in Last Point, because of 'lastp' in "lastpoint" variable name

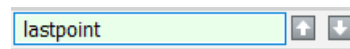
If the LastPoint variable is not the one that I want, then I click the  down arrow to move to the next instance of a candidate that matches "lastp." Continuing with this example, when I click  the focus jumps to **LastPrompt**.

Sometimes the color of the search field changes. The colors report the status of the search term that you entered:

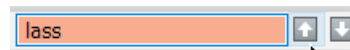
Snow — two or more words match the search phrase



Lime — only one word matches the search phrase, or a repeated search has returned to the start



Tangerine — no words match the search phrase



Export Settings

The last button is **Export** and it saves all settings and values to a CSV file, short for “comma-separated values.” The file contains the names of variables in true alphabetical order, their current values, and other information.

Each value is separated by a comma, as shown by this sample:

```
AUTOSNAP,reg,int,RTSHORT,63,63,,AutoSnap
AttractionDistance,prf,int,RTNONE,3,3,,Grips attraction distance
AutoTrackingVecColor,prf,int,RTNONE,171,171,,Auto tracking vector color
AutosaveChecksOnlyFirstBitDBMOD,prf,bool,RTNONE,1,1,,Ignore all but first bit of DBMOD for autosave
BACKZ,drw,real,RTREAL,0,0,,Back clipping plane offset
```

TIP Fields are separated by commas. Because coordinates normally use commas — such as 2,3,4 — in this CSV file the commas are replaced by semi-colons, such as 2;3;4.

The meaning of each field is explained by the table below using the example of AutoSnap.

```
AUTOSNAP,reg,int,RTSHORT,63,63,,AutoSnap
```

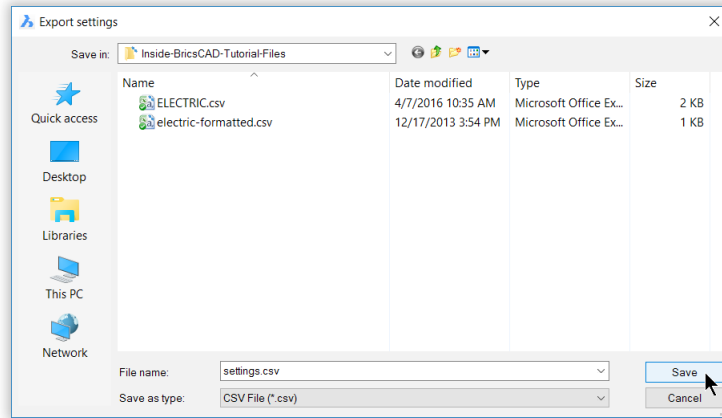
Data	Example	Field	Options	Meaning
Name	AUTOSNAP	Variable name	UPPERCASE MixedCase	System variable Unique to BricsCAD
Save mode	reg	Location where the value is saved	not prf reg drw	Not saved In BricsCAD preferences In the Windows registry In the drawing
Save type	0 4 10000000 0.5;0.5 1;0;0 25.4 ANSI31	Type of value	bool int long pt2d pt3d real str	Boolean (a toggle, such as 0 or 1) Integer (no decimal places) Long integer (greater than 2 ¹⁶) 2D point (x,y) 3D point (x,y,z) Real number (with decimal places) Strings (text)
Restype	4 10000000 0.5;0.5 25.4 ANSI31	Numerical type of value	RTSHORT RTLONG RTPOINT RTREAL RTSTR RTNONE	Short integer (same as integer) Long integer 2D and 3D point Real number String Stored in preferences
Default value	63	Specifies the default value, as found in the default template file		
Current value	63	Specifies the current value		
Status		Reports when the value is read-only (cannot be changed by the user)		
Title	AutoSnap	Briefly describes the purpose of the variable		

As a file format, CSV is “universal,”because it can be imported easily into spreadsheets and databases. These programs use the comma to identify where to separate the fields into columns. In a word processor, I use the Find and Replace command to change commas to tabs.

Exporting Variables

To export the settings data, follow these steps:

1. Click the **Export** button. Notice the Export Settings dialog box.

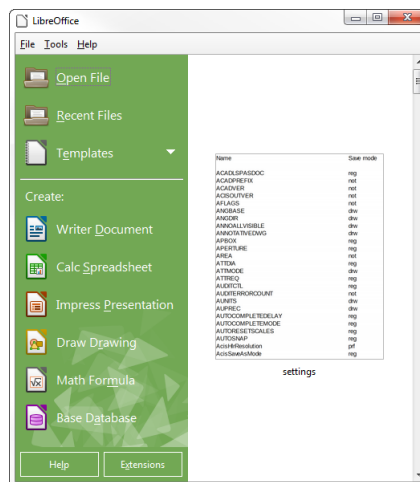


Saving variables to a .csv file

2. Choose a folder in which to store the file. You can change the file’s name, but make sure you leave the extension set to “.csv”.
3. Click **Save**.

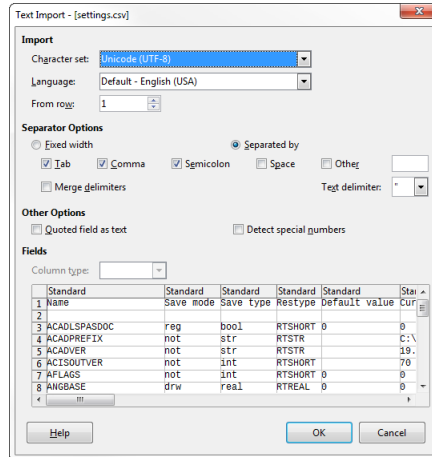
The data is exported from BricsCAD in alphabetical order, whether or not the current setting is Alphabetical or Categorized. To open the exported file in an application, such as the Calc spreadsheet program from LibreOffice, continue with these steps:

1. Start LibreOffice. (You can download this software free of charge from <http://www.libreoffice.org>. It is available for Windows, Linux, and MacOS — just like BricsCAD!)



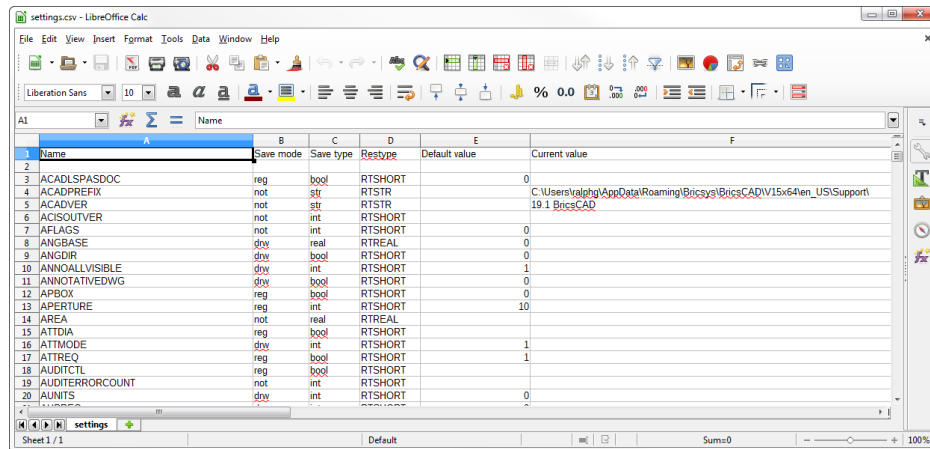
The initial LibreOffice interface

- From the File Manager (Finder on the Mac), drag the `settings.csv` file into the LibreOffice window shown above.
- Notice the Text Import dialog box. It shows you how LibreOffice proposes to separate the comma-delimited data into columns. Under Separator Options, ensure that **Comma** is selected, and then click **OK**.



Formatting imported text

- Notice that the settings data appears in columns in the spreadsheet. Format and edit the text as you wish.



Settings imported successfully into a spreadsheet

ACCESSING VARIABLES AND CHANGING VALUES

To access the *value* of a variable, you can use the **Find** field described above, or else click the **+** node boxes to open sections. (Click **-** nodes to collapse sections.) Notice the nodes:

- Modeler properties for viewonly ACIS(Classic license)**
- Modeler properties for full ACIS(Pro license and higher)**
- Use FACETRES system variable

To change the value of a variables, follow these steps:

- Navigate to the variable you want to change.
- Click on the name of the variable.
- Depending on the nature of the variable, you take one of these actions to change the value:
 - Text/numerical** variable — type in a new piece of text or a new number



- **Toggle** variable — click a check box to turn it on (green check mark) or off (none)

Grips	<input checked="" type="checkbox"/> Turn on grips
-------	---

- **Option** variable — select an option from a droplist

Entity dragging	Whenever possible
	No dragging
	When requested
	Whenever possible

- **Value** variable — enter a new value

Grip size	6
-----------	---

- **File** variable — click the  Browse button to open the file dialog box

Save file path	C:\Users\fhg\AppData\Local\Temp\
Cloud temporary folder	C:\Users\fhg\AppData\Local\Temp\Bricsys_24_7\

After it is changed, the variable name and its value turns to **boldface** — a way of alerting you to changes. Not all variables can be changed.

Those with the “read-only” setting cannot be changed and so are shown with gray text. (See figure below.) When you click on them, they do not react.

Modification status	0x0001 (1)
---------------------	------------

Variables Specific to Windows

Some preferences are specific to the Windows version of BricsCAD, such as those related to OLE. They have no effect in Linux or Mac.OS

Changing Variables at the Command Prompt

Outside of the Settings dialog box, you can change the values of variables at the ‘ : ’ command prompt. There are two ways to do this.

- ▶ Enter the variable name at the command prompt just like a command name:

```
: gripblock
New current value for GRIPBLOCK (Off or On) <Off>: on
```

The values in the parentheses report the valid range of values, such as (*Off or On*); the value inside the angle brackets report the current (default) value, such as <*Off*>. I find this useful for determining the range of allowable values.

- ▶ Use the **SetVar** command. The only advantage to this command is that it also lets you list the names of variables, if you are unsure of the exact spelling. Here’s how to do this:

1. First, enter a ? (question mark) at the prompt:

```
: setvar
Variable name or ? <GRIPBLOCK>: ?
```

- Then, type the part of the name that you know, and use an * (asterisk) to represent the unknown part of the name.

Variable(s) to list <*>: **grip***

GRIPBLOCK	0
GRIPCOLOR	72
GRIPDYNCOLOR	140
GRIPHOT	240
GRIPHOVER	150
GRIPOBJLIMIT	100
GRIPS	1

TIPS To get a list of variable names and their current values, press **Enter** at the 'Variable(s) to list <*>:' prompt.

To capture the list to a file, use the **LogFileOn** command before entering **SetVar**, and then **LogFileOff** afterwards. The location of the log file is given by variable **LogFilePath**.

(Historical note: The Settings dialog box was added with BricsCAD V8, replacing the Options dialog box of earlier releases.) See Appendix B for a complete list of all BricsCAD variable names and default values in alphabetical order.

CHAPTER 3

Changing BricsCAD's Environment

BricsCAD allows you great flexibility in changing the way it looks and works. The first few chapters of this book show you how to change the *look* of BricsCAD; later ones concentrate on changing how it *works*.

This chapter tells you how to change the way in which BricsCAD starts up, as well as how to use the Settings dialog box to change the look of BricsCAD's user interface. For example, you can change things like the background color of the drawing area and the size of the cross hair cursor to suit you.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Starting BricsCAD
- Setting command line options (not available in Mac)
- Changing screen and other colors of the user interface
- Specifying support file paths
- Launching the user profile manager (not available in Linux or Mac)

Starting BricsCAD

You probably know about these ways to start BricsCAD:

- ▶ Double-click the BricsCAD icon found on your computer's desktop
- ▶ Or, on the taskbar of most releases of Windows, click the **Start** button, and then select **Bricsys | BricsCAD V20 | BricsCAD** or something similar
 - In Windows 8 and Windows 10 tablet mode, click the BricsCAD icon in the Start screen.
 - In Linux, click the **Main Menu** button, and then select **Graphics | BricsCAD**
 - In the MacOS dock, click the **Applications** folder, and then select **BricsCAD**
- ▶ Or, in Windows Explorer (or Linux File Browser or MacOS Finder), double-click the name of a *.dwg* file; this option works only when BricsCAD is the default program assigned to *.dwg* files

But there are other ways to launch the program. These variations are described next.

COMMAND LINE OPTIONS

It was common knowledge in the days of the DOS and Unix operating systems that programs could use options to start up. Windows, MacOS, and Linux hide much of what goes on behind their graphical user interfaces, and so *command-line options* are no longer common. They are, nevertheless, still available, and here I show you how to use them with BricsCAD.

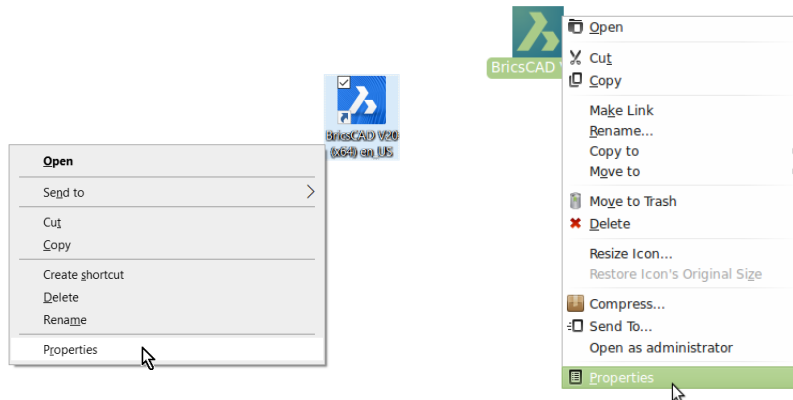
Normally, BricsCAD starts with a new, blank drawing. You can, however, have BricsCAD start with a specific file by editing a *target value*. BricsCAD can be made to load specific drawing files — and other kinds of files — as it starts; you just need to specify the file name to the OS (operating system) at the command line. In a moment, I'll tell you how to access that command line.

The following table lists the file types that can be used at the command line, and what they do when BricsCAD starts up:

File Type	Meaning
<i>.CUI</i>	Loads the file that customizes the user interface
<i>.DLL</i>	Loads ADS/SDS or DRX/ARX programs (dynamic link libraries)
<i>.DSD</i>	Plots files specified earlier by the Publish command
<i>.DWG</i>	Opens a drawing file made by BricsCAD, AutoCAD, and other CAD programs
<i>.DWT</i>	Opens a template file that specifies the initial settings of new drawings
<i>.DXF</i>	Opens drawing interchange format files from other CAD programs
<i>.LSP</i>	Loads LISP and AutoLISP routines
<i>.MNU</i>	Loads menu files from older releases of BricsCAD and AutoCAD
<i>.SCR</i>	Runs a script file
<i>.SLD</i>	Displays a slide file

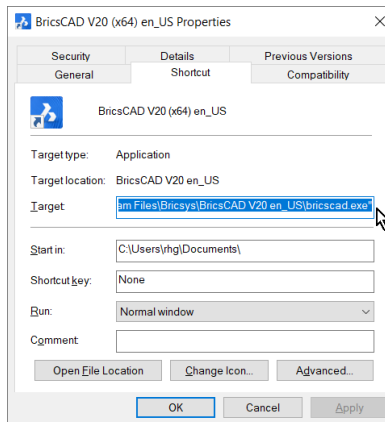
Here's how to access the command line in Windows and Linux. This feature is not available on MacOS.

1. On the desktop of Windows or Linux, right-click the BricsCAD icon.
2. Notice the shortcut menu. Select **Properties**.



Left: Accessing icon properties in Windows 7... right: ...and in Linux Mint

3. In the Properties dialog box of Windows, select the **Shortcut** tab.
In the Properties dialog box of Linux, select the Launcher tab.



Properties dialog box in Windows

Notice that for Windows the default command-line text is similar to the path listed below:

Target "C:\Program Files\Bricsys\BricsCAD V20 en_US\bricscad.exe"

Linux is straightforward: it knows the location of programs, and so programs do not need a path.

The elements of a path, such as "C:\Program Files\Bricsys\BricsCAD V20 en_US\bricscad.exe", have the following meaning:

Path	Meaning
C:	Name of the disk drive
\Program Files	Name of a folder
bricscad.exe	Name of the BricsCAD program

Quotation marks (" and ") are needed when there are spaces in the names of folders and programs

Colon (:) identifies the names of disk drives, such as C:

Back slashes (\) separate the names of folders. The folder names may vary, depending on where BricsCAD was installed on your computer

Linux has these differences from Windows:

- Linux uses forward slashes (/) to separate folder names, rather than backslashes (\)
 - The path starts from the *home* folder, not the root folder
 - Path and file names are case-sensitive in Linux, so “Documents” is a different name from “documents”
4. Here is how to start BricsCAD with a specific .dwg drawing file. The same procedure works with .dxf translation files, as well.

In the Properties dialog box in Windows, edit the text in the **Target** box by adding the text shown in **color**:
"C:\Program Files\Bricsys\BricsCAD V20 en_US\bricscad.exe" "c:\folder\file name.dwg"

In Linux, add the path to the drawing file to the Command box:

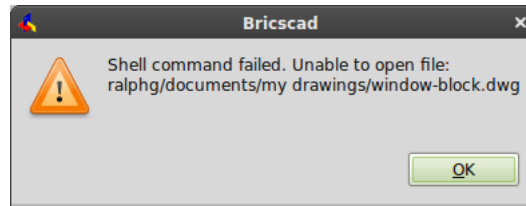
```
bricscad.exe "home/<login>/Documents/My Drawings/file name.dwg"
```

Replace “<login>” with the name by which you logged into your Linux computer. In my case, it is “ralphg,” so the path looks like this:

```
bricscad.exe "home/ralphg/Documents/My Drawings/file name.dwg"
```

Notice that:

- The full path name to the drawing is required
 - Separate pairs of quotation marks are needed for the names of the program and the file. (Quotation marks are needed only when paths and file names contain spaces.)
5. Click **OK** in Windows.
In Linux, click **Close**.
6. Test your modification by double-clicking the BricsCAD icon. The program should start, and then open the file. If you made an error, you will receive a complaint from a dialog box, such as this one in Linux:



Warning message from the operating system

Reasons for errors include the following ones:

- The file name was incorrectly spelled
- The path is incorrect or incomplete
- The file is missing and does not exist
- The quotation marks are unbalanced, with the starting or ending " missing

CATALOG OF COMMAND-LINE SWITCHES

As it starts up, BricsCAD can open a number kinds of file types, as listed by the table earlier. For instance, after starting it could run a script, or it could plot a number of drawings automatically in *batch* mode.

When it comes to file types other than drawings, however, you need to use a *command-line switch* to alert BricsCAD as to the type of file. It is called a “switch” because it switches the way BricsCAD operates; for example, the /b switch instructs BricsCAD to run a script file following start-up.

To indicate a switch, Windows uses the forward slash (/), borrowed from DOS. Linux uses a dash (-), borrowed from Unix. (This feature is not available on Mac.) Here is the complete list of switches that BricsCAD accepts:

Windows	Linux	Meaning
/b filename.scr	-b filename.scr	Runs an .scr script file following start up
/l	-l	Suppresses the BricsCAD logo (splash screen)
/ld app.arx	-ld app.arx	Loads an ARx, BRx, or DRx application
/p profile.arg	-p profile.arg	Loads an alternate user profile
/pl plotlist.dsd	-pl plotlist.dsd	Plots silently in the background
/s path	-s path	Specifies alternative search paths for support files
/t path	-t path	Specifies the path and name of a .dwt template file
<i>The following switches apply to Windows only; Linux does not support COM:</i>		
regserver	...	Registers BricsCAD's COM (common object model)
unregserver	...	Unregisters COM

You can use any number of switches in a row to make multiple things happen as BricsCAD starts up. For example, it could plot a number of drawings and then open a specific template file.

Let's now go through a detailed description of each switch.

No Switch - Load Drawings

BricsCAD uses no switch to load one or more .dwg and/or .dxf files specified at the OS command line:

```
"c:\program files\bricsys\bricscad\bricscad.exe" "c:\my documents\filename.dwg" c:\dwg\filename.dxf
```

B Switch - Script Files

The **b** switch specifies the name of a .scr script file to run immediately after BricsCAD starts. The "b" is short for *batch*. The switch is followed by the path and name of the script file. Here is an example of the usage in Windows:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /b "c:\BricsCAD\script file.scr"
```

See the later chapter on scripts to learn how to write your own script files.

L Switch - No Logo

The **l** switch suppresses the logo at startup. The "l" is short for *logo*. This means that the splash screen bearing the BricsCAD name and version number does not appear. Notice that this switch appears by itself; no path or file name is associated with it.

In Windows, it looks like this:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /l
```

In Linux, it looks like this:

```
bricscad.exe -l
```

LD Switch - Application Load

The **ld** switch specifies the names of applications to load, specifically those written with Bricsys's BRx, the ODA's DRx and Tx, or Autodesk's ARx application programming interfaces — APIs. The "ld" is short for *load*. This switch is useful when you want to load add-on programs right away as BricsCAD starts.

```
"c:\program files\bricsys\bricscad\bricscad.exe" /ld "appname.brx"
```

S Switch - Search Support Paths

The **s** switch specifies alternative search paths for support files. The "s" is short for *search*. This switch is useful when you want to load linetypes, patterns, and menu files provided by clients; you don't want to mess up your own setup, and so you place these files in their own folder, and then point to the folder with this switch.

Notice that this switch specifies only paths, not any file names:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /s "c:\client\support"
```

You can specify multiple paths by separating them with semicolons (;), like this:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /s "c:\client1\support;c:\client2\support"
```

P Switch - User Profiles

The **p** switch loads an *.arg* user profile file. The "p" is short for *profile*. This file changes the way that BricsCAD looks, as described more fully later in this chapter.

Here is an example of its use:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /p "c:\bricscad\myui.arg"
```

TIP BricsCAD includes a separate utility command for creating and editing user profiles, **ProfileManager**, described at the end of this chapter.

PL Switch - Batch Plotting

The **pl** switch plots drawings in the background without showing the BricsCAD program window. The "pl" is short for *plot*. It reads the files to be plotted from *.dsd* files, which are created earlier by the Publish command in BricsCAD.

Here is an example of its use:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /pl "c:\bricscad\plotlist.dsd"
```

BricsCAD reads the *.dsd* file and then plots the drawing according the instructions contained therein. The *.dsd* file saved by the Publish command specifies the file name, layouts, page setups, plotter and printer names, orientation, plot scale, number of copies, optional plot stamp, and the order in which to plot the files.

The *.dsd* extension is optional. When the file name is missing, however, BricsCAD simply exits.

TIP When BricsCAD starts with the `/pl` switch, it ignores the setting of SingletonMode, and so multiple instances of the program can be launched irregardless.

T Switch - Template Files

The `t` switch opens BricsCAD with a new drawing based on the `.dwt` template file specified by this switch. The “t” is short for *template*. This file changes the way that the drawing initially looks.

Here is an example of its use:

```
"c:\program files\bricsys\bricscad\bricscad.exe" /t "c:\drawings\officetemplate.dwt"
```

Regserver and Unregserver Switches

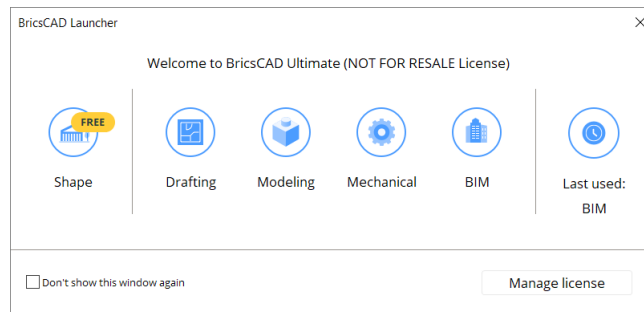
The `regserver` and `unregserver` switches register and unregister BricsCAD’s COM common object model. They operate only with Windows, because Microsoft does not provide COM for Linux or MacOS systems.

OTHER STARTUP CONTROLS

(**NEW IN V20**) BricsCAD offers variables that control which elements are displayed during startup. These elements include the Launcher dialog box and the Start screen. Note that the splash screen is controlled by the `/l` switch described above. (Functions new in V20 are shown in [blue](#).)

GetStarted variable toggles the display of the Launcher dialog box:

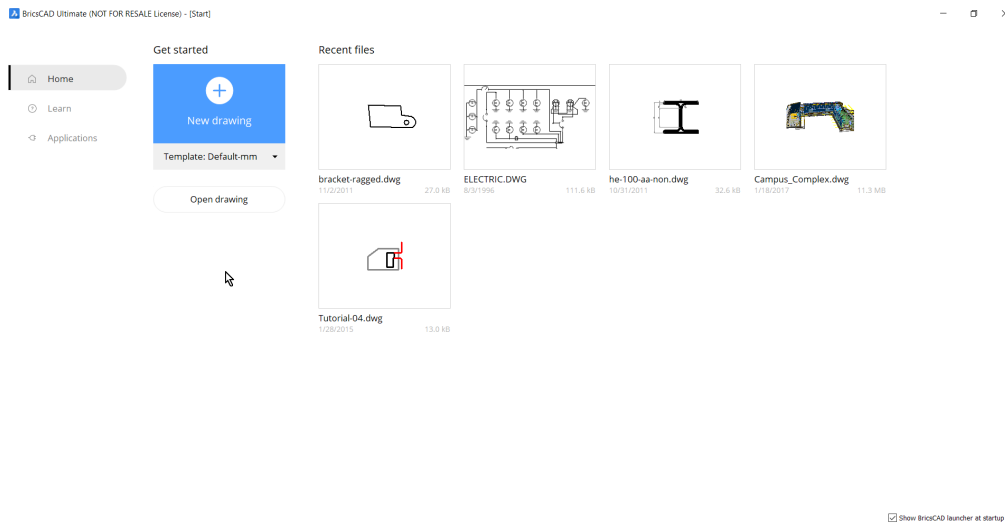
GetStarted	Meaning
0	Do not display
1 (default)	Display the Launcher dialog box



The Launcher dialog box

Startup variable displays one of several startup dialogs:

Startup	Meaning
0	Start new drawing with template file specified by BaseFile
1	Display Startup dialog box
2	Display the Start window without the ribbon (see figure below)
3 (default)	Display the Start window with the ribbon



The Launcher dialog box

Changing the Colors of the User Interface

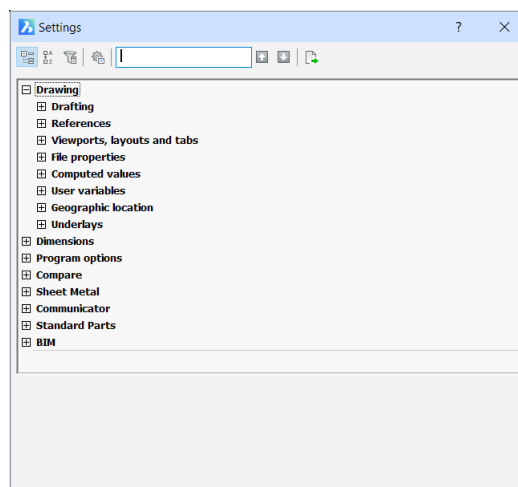
BricsCAD allows you to change some aspects of its user interface directly, while other aspects are controlled by Windows. With BricsCAD, the changes are made through the Settings dialog box.

THEME COLOR

(NEW IN V20) You can change the colorization of the entire interface of BricsCAD between light and dark. This is called the “theme color.” Prior to V20, the theme color was light; as of V20, the default is dark. The change is made with the **ColorTheme** setting:

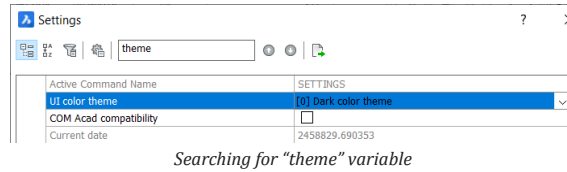
In BricsCAD, you can change a settings like this:

1. Enter the **Settings** command. Notice the Settings dialog box.

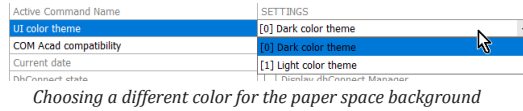


Settings dialog box

- In the dialog box, enter “theme” in the search field. If the field turns orange, it’s because you have misspelled the word(s). Notice that dialog box jumps to the **UI Color Theme** settings.

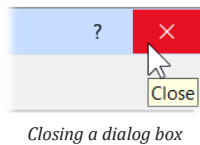


- Click the droplist adjacent to **UI Color Theme**, and then choose a theme, either light or dark:



The changes you make in the Settings dialog box have immediate effect in BricsCAD, and so the theme color changes as soon as you select the other one.

There is no “OK” or “Apply” button to click, because changes take effect immediately. Instead, click the dialog box’s **Close** button:



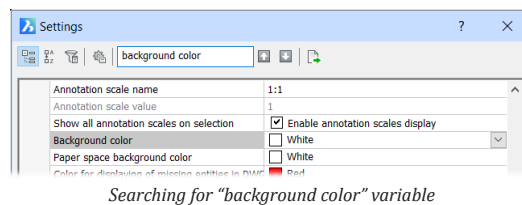
Background Color

The first change I always make to a new CAD installation is to the background color of the drawing area. Usually, it are colored black by default.

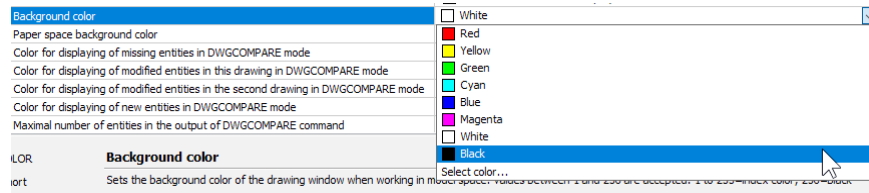
(History note: Black was the traditional color in the days when CAD ran on the DOS operating systems. Some users prefer the dark color, because entity colors look more vibrant against it. Others prefer a white background, because that most closely resembles the paper upon which the drawing will be printed. I prefer the white background color.)

In BricsCAD, you the change background color like this:

- In the Settings dialog box, enter “background color” in the search field. Notice that dialog box jumps to the Background Color variable.



- Click the droplist adjacent to **Background Color**, and then choose a color, such as “White.”



Choosing a different color for the paper space background

TIP If the color you are looking for is not on the droplist, then click **Select Color** to access the Color dialog box. Choose a color from the dialog box, and then click **OK**.

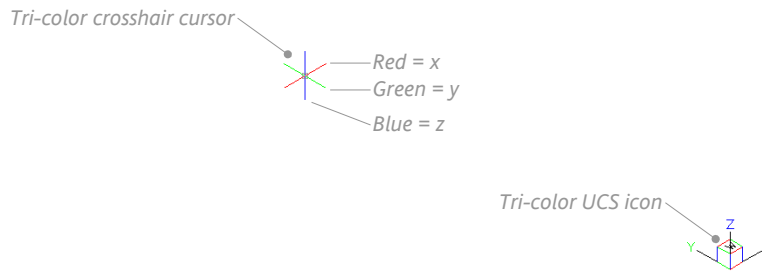
In a similar manner, you can make changes to the background color of paper space (layouts) — and many other areas of BricsCAD. For layouts, this is done with the **Paper Space Background Color** option. (It affects the color of the “paper” in layouts, not the color of the background.)

If, after exiting the Settings dialog box, you want to revert the value, just enter the **U** (undo) command

TIP The changes you make to the color of cursor’s axes have no effect the colors of the UCS icon’s axes, which cannot be customized.

Changing Cursor Color and Size

In the drawing area, BricsCAD displays a “tri-color” crosshair cursor that has a different color for each of the axes. The UCS icon follows the same color scheme.



Colors of the cursor and UCS icon

SETTINGS AT THE COMMAND LINE

Many times, using the BricsCAD command line is faster than the dialog box. You can avoid opening the Settings dialog box (and then searching for variable names) by changing values directly at the command prompt: just type the name of a variable.

For example, I find it faster to change the size of the crosshair cursor by entering the following variable name at the command prompt, and then changing its value:

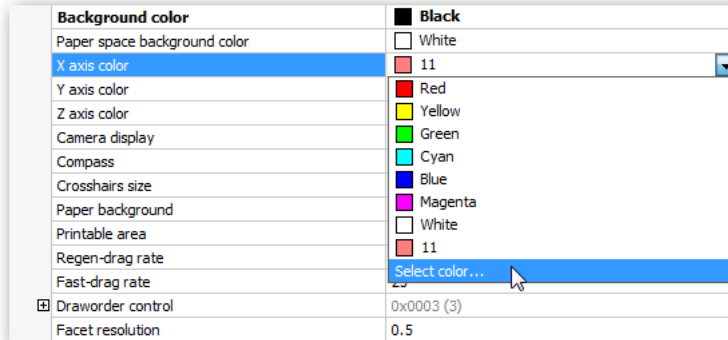
```
: cursorsize
New current value for CURSORSIZE (1 to 100): <3>: 100
```

By default, the colors are assigned as follows:

Axis	Default Color	Color Number
x	Red	11
y	Green	112
z	Blue	150

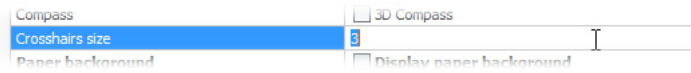
You change its color and size of the crosshair cursor like this:

1. In the Setting dialog box's **Program Options** section, enter "x axis color" in the **Search** field.
2. Choose **X axis Color**, and then change its color.



Changing the color of the cursor axes

3. Repeat for **Y axis Color** and **Z axis Color**.
4. If you wish, you can also change the size of the cursor with the **Crosshairs Size** setting. The default is 3, which means 3% of the screen size. A value of 100 (percent) makes the cursor full-size.



Changing the size of the crosshair cursor

Normally, the crosshair cursor appears only when you use a drawing or editing command. If you want the crosshair cursor to always be on, then turn on the "Pointer defaults to crosshairs" option of the **Always Use Crosshairs** option.

DISPLAY SETTINGS

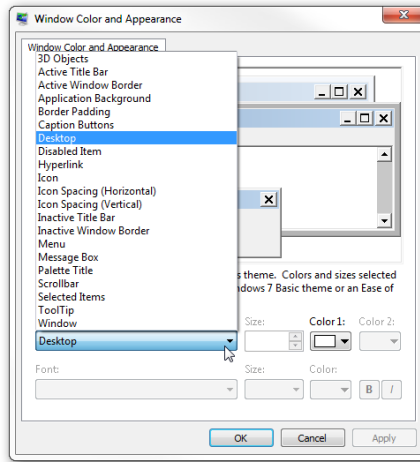
Here are settings that affect the BricsCAD display:

Display Setting	Variable Name	Meaning
Background Color	BkColor	Sets background color of model space drawing area
Paper Space Background Color	BkColorPs	Sets layout mode's paper color
X Axis Color	ColorX	Specifies color of crosshair cursor's x axis
Y Axis Color	ColorY	Specifies color of crosshair cursor's y axis
Z Axis Color	ColorZ	Specifies color of crosshair cursor's z axis
Always Use Crosshairs	AlwaysUseCrosshair	Displays crosshair cursor in place of pointer cursor
Crosshairs Size	CursorSize	Sizes cursor cross hairs, as a percentage of drawing area

Interface Parameters Controlled By the OS

Operating systems control some of aspects of BricsCAD's user interface, such as fonts and colors used by menus. This level of personalization is not available in Windows 8 or 10. In Windows 7, follow these steps:

1. Right-click the desktop, and then select **Personalize**.
2. Click **Windows Color** and then choose **Advanced Appearance Properties**.

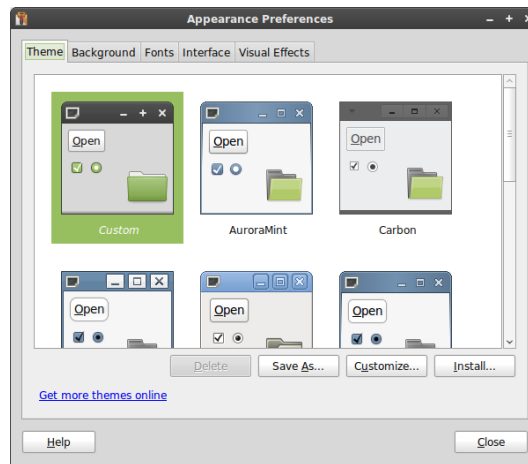


Customizing the appearance of Windows

3. In the Window Color and Appearance dialog box, change the fonts and color user interface elements.

In Linux, follow these steps:

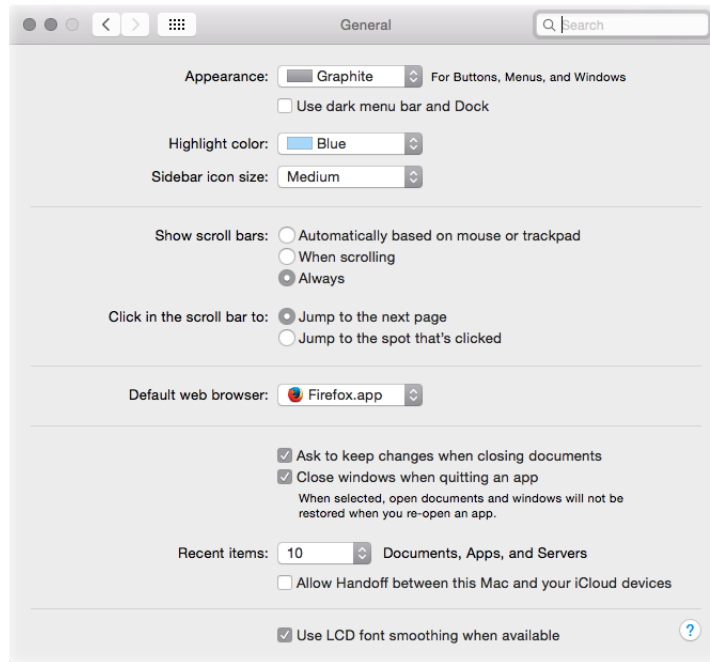
1. Right-click the desktop, and then select **Change Desktop Background**.
2. Click **Fonts** to change the default fonts used for the interface.
3. Choose **Theme** to set the overall look and color of dialog boxes and other UI elements.



Customizing the appearance of Linux

In MacOS, follow these steps:

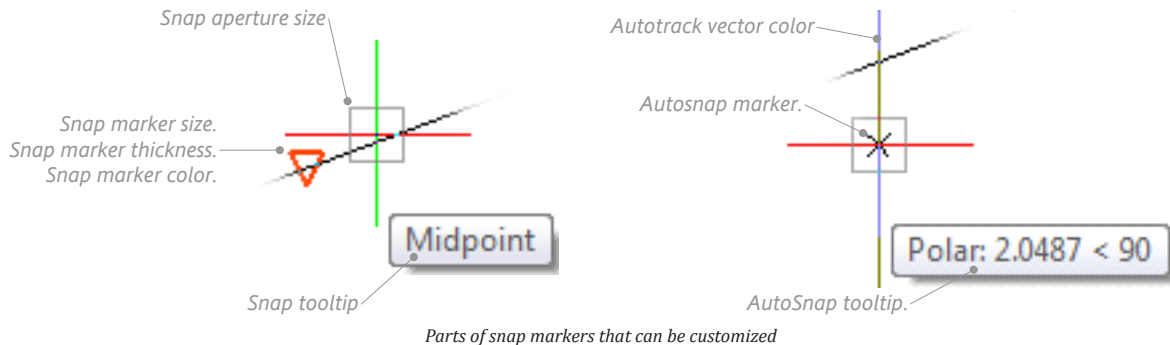
1. From the dock, choose **Settings**.
2. In the System Preferences folder, choose **General**.
3. Make changes as desired.





General settings for MacOS

SNAP MARKER OPTIONS

Snap markers display information at the cursor about current object (or entity) snap modes. BricsCAD allows you to choose the color, and other options, for snap markers and snap cursors. Some of the settings are illustrated below:



Most of these options are found near the end of the **Display** section of the Settings dialog box — and not the Snap/Grid section!

Snap marker size	5
Snap marker thickness	2
Snap marker color	 20
Auto tracking vector color	 171
Snap marker in all views	<input type="checkbox"/> Display snap marker in all views
Snap tooltips	<input checked="" type="checkbox"/> Enable snap tooltips

Settings for snap markers

Other snap-related options are found at the start of the **Snap Tracking** section:

Snap tracking	
AutoSnap	0x0000 (0)
1	<input type="checkbox"/> AutoSnap marker
2	<input type="checkbox"/> AutoSnap tooltips
4	<input type="checkbox"/> AutoSnap magnet
8	<input type="checkbox"/> Polar tracking
16	<input type="checkbox"/> Entity snap tracking
32	<input type="checkbox"/> Tooltips for polar tracking and entity s

Additional settings for snap markers

Finally, there are aperture options in the **Entity Snaps** section:

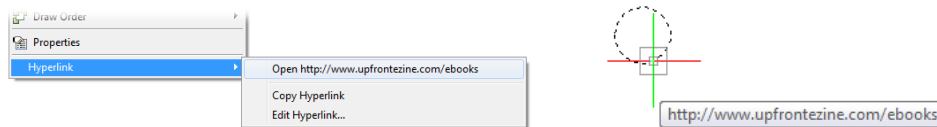
Entity snaps	
Entity snap mode	0x0020 (32)
Entity snap options	0x0003 (3)
1	<input checked="" type="checkbox"/> Entity snap ignores hatches
2	<input checked="" type="checkbox"/> Entity snap ignores negative Z values in Dynamic UCS mode
Entity snap aperture box	<input type="checkbox"/> Display Aperture box
Entity snap aperture	10
Dgn entity snap	<input checked="" type="checkbox"/> Enable DGN entity snap
Dwf entity snap	<input checked="" type="checkbox"/> Enable DWF entity snap
Pdf entity snap	<input checked="" type="checkbox"/> Enable PDF entity snap

Settings for the aperture

HYPERLINK CURSOR OPTIONS

Entities can contain *hyperlinks*, which are links to Web sites and other documents. When the cursor passes over them, a tooltip reports the name of the link. A shortcut menu that displays options useful for working with hyperlinks, which are added with the **Hyperlinks** command.)

The hyperlink menu and the tooltip are illustrated below:



*Left: Hyperlinks options added to right-click menu.
Right: Tooltip reporting hyperlink attached to circle.*

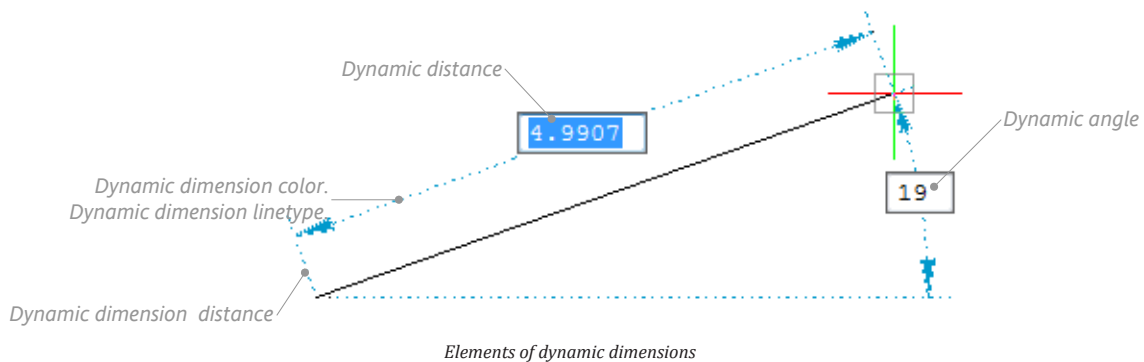
The Settings dialog box has the options for hyperlinks near the end of the Display section.

Hyperlink menu	<input checked="" type="checkbox"/> Enable hyperlink menu
Hyperlink tooltip	<input type="checkbox"/> Enable hyperlink tooltip

Settings for hyperlinks

DYNAMIC DIMENSION OPTIONS

Dynamic dimensioning shows the lengths and angles of entities as they are being drawn. You turn on this function by clicking **DYN** on the status bar.



Through the Settings dialog box, you can adjust the look of dynamic dimensions. The quick way to access these is by right-clicking **DYN** on the status bar, and then choosing **Settings**.

<input type="checkbox"/> Dynamic input	
<input type="checkbox"/> Dynamic input mode	0x0003 (3)
negative	<input type="checkbox"/> Switch all off temporarily
1	<input checked="" type="checkbox"/> Dynamic input at pointer (not supported)
2	<input checked="" type="checkbox"/> Editable dynamic dimensions
4	<input type="checkbox"/> Tracking dynamic dimensions
<input type="checkbox"/> Show dynamic dimensions	0x001F (31)
1	<input checked="" type="checkbox"/> Resulting length
2	<input checked="" type="checkbox"/> Extended length
4	<input checked="" type="checkbox"/> Absolute angle
8	<input checked="" type="checkbox"/> Relative angle
16	<input checked="" type="checkbox"/> Arc radius
Dynamic dimension visibility	[1] Only the first two dynamic dimensions
Default mode for dynamic coordinates input	[0] Relative
Dynamic dimension hover color	142
Dynamic dimension hot color	142
Dynamic dimension distance	1
Dynamic dimension linetype	[0] Continuous
Transparency of dynamic input fields	90
Dynamic dimension aperture	20

Settings for dynamic dimensions

Support File Paths

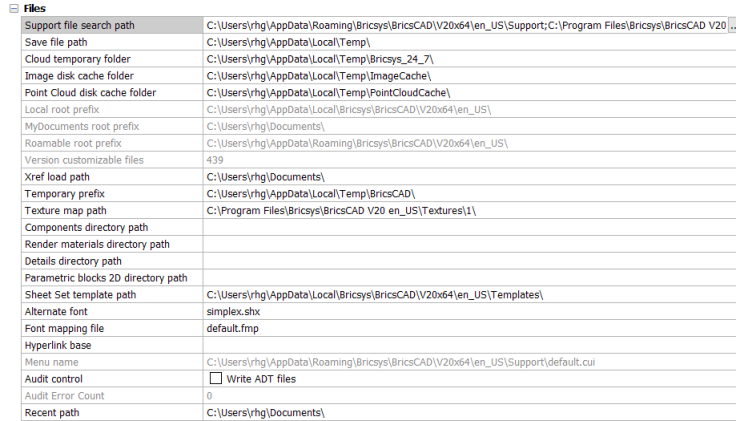
BricsCAD uses a number of folders in which to store support files, such as those needed for fonts, on-line help, and hatch patterns. BricsCAD locates the support files by consulting “paths” to the folders. A *path* specifies the name of the drive and the folder, such as *c:\BricsCAD*. There are several reasons why you may want to change the file paths that BricsCAD uses:

- ▶ Your firm has clients with different standards for fonts, layers, and so on
- ▶ You are a third-party developer, and need to have paths pointing to sets of different files
- ▶ You import drawings from other CAD packages, and need to map different sets of fonts via *fmp* files

You are not limited to specifying one single path; you can have BricsCAD search along multiple paths, including those on networks. Multiple paths are separated by semicolons (;).

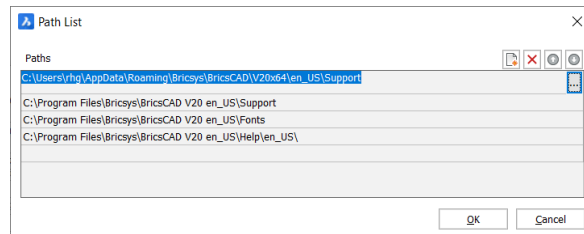
Here is how to change paths and file locations:

1. Enter the **Settings** command.
2. In the Search field, enter “support file.” Notice the Files section.
3. To change a path, select a path adjacent to a heading. For example, click the field next to **Support File Search Path**.



All of the paths to files




4. Notice the  **Browse** button. Click it.



Adding and removing paths


5. Notice that BricsCAD displays the Folder List dialog box, which lets you add, remove, and reorder multiple paths. Choose a path, and then close the file dialog box.

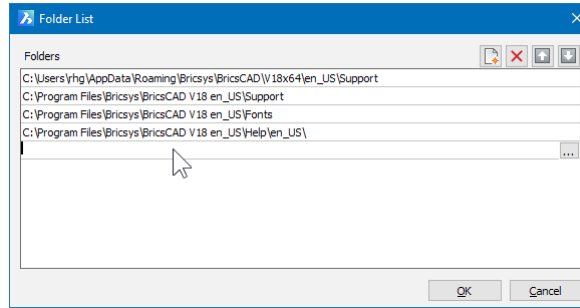
The Path List dialog box has buttons for controlling the list of folder names:

Button	Name	Meaning
	Add Folder	Adds a blank entry to the list
	Remove Folder	Removes the selected folder from the list. Caution! BricsCAD does not ask for deletion confirmation. If you erase a folder by accident, click the Cancel button to exit the dialog box with no changes preserved.
	Move Up / Down	Moves the selected folder up or down the list


TIP BricsCAD searches paths by the order in which they are listed this dialog box. By changing the order, BricsCAD will find files in folders listed higher up sooner. For example, you might want BricsCAD to search folders on your computer before searching the network — or the other way around.

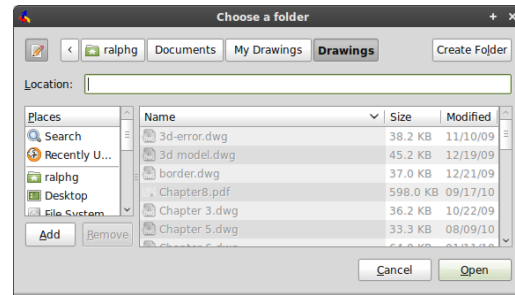
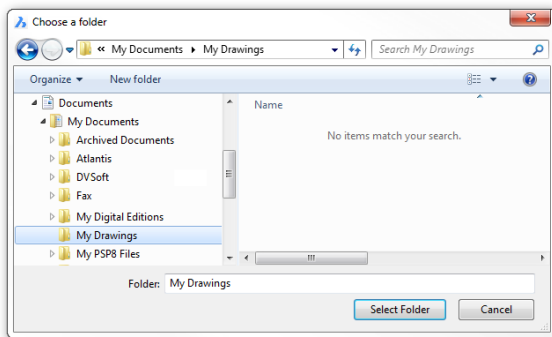
To add a folder, follow these steps:

1. Click the  **Add Folder** button. Notice that a blank entry is added to the dialog box.

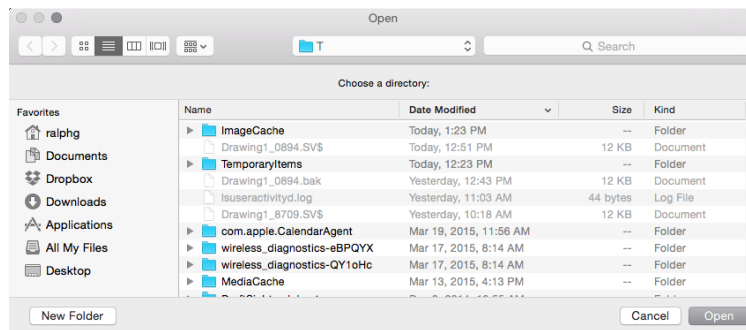


Blank line added to folders list

2. Click the  **Browse** button. In Windows, notice the Browse for Folder dialog box. (In Linux, the Choose a Folder dialog box appears.)



Left: Choose a Folder dialog box in Windows; right: Choose a Folder in Linux
Below: Open a folder in MacOS



3. You can choose a folder on your computer, or from a computer connected over the network. Internet locations cannot, however, be chosen.

TIP To create a new folder while in the files dialog box, just click the **New Folder** (or **Create Folder**) button, and then give it a name.

4. Click **OK** sufficient times to exit dialog boxes. Notice that the selected path is added to the list.

SUMMARY OF FILES SETTINGS

Here are some details on the paths and support files specified through the Setting dialog box's **Files** section. You can modify all paths, except for ones that are read-only, such as "Local Root Prefix" and "Menu Name."

System variable names are shown in SMALL CAPS; those in **blue** are specific to BricsCAD; those in **black** are found in both BricsCAD and AutoCAD.

Files (and Paths)

Support File Search Path (**SRCHPATH**) — folders that store fonts, customization, plug-ins, blocks, linetypes, and hatch patterns. You can specify multiple paths, each separated by a semi-colon.

Save File Path (**SAVEFILEPATH**) — path for storing temporary and automatically saved files.

Cloud Temporary Folder (**CLOUDTEMPFOLDER**) — folder in which BricsCAD stores temporary download files from its online 24/7 service (formerly called "chapootempfolder").

Image Disk Cache Folder (**IMAGECACHEFOLDER**) — folder which temporarily stores cached raster files placed as images in drawings.

Point Cloud Disk Cache Folder (**POINTCLOUDCACHEFOLDER**) — folder which temporarily stores point cloud files placed in drawings.

Local Root Prefix (**LOCALROOTPREFIX**; read-only, set by Windows) — folder in which BricsCAD stores support files for the program specific to the local computer.

My Documents Root Prefix (**MYDOCUMENTSPREFIX**; read-only, set by Windows) — folder in which BricsCAD accesses the logged-in user's specific documents and other files.

Roamable Root Prefix (**ROAMABLEROOTPREFIX**; read-only, set by Windows) — folder which stores support files for the program when the user signs onto other (roamed) computers.

Version Customizable File (**VERSIONCUSTOMIZABLEFILES**; ready-only, set by Bricsys) — current version number of CUI and PGP files.

Xref Load Path (**XLOADPATH**) — path to temporary copies of demand-loaded xrefs.

Temporary Prefix (**TEMPPREFIX**) — path to the folders in which BricsCAD stores temporary files, such as automatic backup files (*filename.sv\$*).

Texture Map Path (**TEXTUREMAPPATH**) — path to the folder holding texture map files.

Components Directory Path (**COMPONENTSPATH**) — path to component files used for mechanical parts.

Render Material Directory Path (**RENDERMATERIALPATH**) — path to material files used for rendering.

▣ Projects	
Project name	
Project search paths	
▣ Printer support	
Plot styles path	C:\Users\rhg\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\PlotStyles\
Plotter configuration path	C:\Users\rhg\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\PlotConfig\
Plot output path	C:\Users\rhg\Documents\
Print file	.
▣ Templates	
Template	Default-mm.dwt
Template path	C:\Users\rhg\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\Templates\
Sheet Set template path	C:\Users\rhg\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\Templates\
Default new sheet template	
BMFORM template path	
▣ Tool palettes	
Tool Palettes bar state	[0] Tool Palettes bar is invisible
Tool palettes path	C:\Users\rhg\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\ToolPalettes\
▣ Dictionaries	
Custom spelling dictionary	C:\Users\rhg\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\Sample.cus
Main spelling dictionary	en_US.dic
▣ Log files	
Log file mode	<input type="checkbox"/> Maintain log file
Log file name	
Log file path	C:\Users\rhg\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\
Cloud log	[0] No log

Additional specifications for paths

Render Material Directory Path ([RENDERMATERIALSPATH](#)) — path to material files used for previews.

Details Directory Path ([DETIALSPATH](#)) — path to details files used for generated drawings.

Parametric Blocks 2D Directory Path ([PARAMETRICBLOCKS2DPATH](#)) — path to where BricsCAD stores 2D parametric blocks.

Sheet Set Template Path ([SHEETSETTEMPLATEPATH](#)) — path to the folder holding sheetset template files.

Alternate Font ([FONTALT](#)) — font to use when a font cannot be found. Default is the *simplex.shx* font file.

Font Mapping File ([FONTMAP](#)) — file that maps fonts. When a font cannot be found, BricsCAD consults this *fmp* file for the name of a matched font; if the matched font cannot be found, it uses the alternative font specified above.

TIP If unexpected fonts appear for text in a drawing, this means that BricsCAD was unable to find the correct fonts. Either use the *default.fmp* file to match equivalent fonts, or add the appropriate path to the **Fonts** path.

Hyperlink Base ([HYPERLINKBASE](#)) — default path for relative hyperlinks in drawings.

Menu Name ([MENUNAME](#); read-only, specified by BricsCAD) — path and name to the menu file


Audit Control ([AUDITCTL](#)) — toggles whether the Audit command creates reports in *.adt* files.

Audit Edit Count ([AUDITERRORCOUNT](#); read-only, set by BricsCAD) — reports the number of errors found in the last audit of a drawing file.

Recent Path ([RECENTPATH](#)) — most-recent path used to access a file.

Project Paths

Project Search Path (`PROJECTNAME`) — paths to the folders in which BricsCAD searches for raster image files and externally-referenced drawing files.

Project Search Paths (`PROJECTSEARCHPATHS`) — names of projects and related paths. Click  button to create new project settings.

TIP Projects allow you to assign one or more paths to a drawing project.

Printer Support Paths and Files

Plot Styles Path (`PLOTSTYLEPATH`) — path to the folders in which BricsCAD accesses `.ctb` and `.stb` plot style files.

Plotter Configuration Path (`PLOTCFGPATH`) — path to the folders in which BricsCAD stores `.pc3` plot configuration files.

Plot Output Path (`PLOTOUTPUTPATH`) — path to the folders in which BricsCAD stores plot files.

Print File (`PRINTFILE`) — default name to use plot files; “” means to use the drawing’s file name.

Templates Paths and Files

Template (`BASEFILE`) — name of the `.dwt` or `.dwg` drawing file used to start new drawings upon starting BricsCAD or when using the New command.

Template Path (`TEMPLATEPATH`) — path to `.dwt` template files.

Sheet Set Template Path (`SHEETSETTEMPLATEPATH`) — path to folder holding sheetset template files.

Default New Sheet Template (`DEFAULTNEWSHEETTEMPLATE`) — name of the `.dwg` or `.dwt` file from which to create new sheets.

BM Form Template Path (`BMFORMTEMPLATEPATH`) — path the folder holding template drawings for BricsCAD Mechanical forms.

Tool Palettes Path

Tool Palettes Bar State (`TPSTATE`; read-only, set by BricsCAD) — reports whether the Tools Palette is open or closed.

Tool Palettes Path (`PROJECTNAME`) — path to the folder holding XTP tool palette definition files.

Dictionaries Section

Custom Spelling Dictionary (`DICTCUST`) — path and name of the `.cus` file used to store words added by the user during the Spell command. No, it has nothing to do with cussing.

Main Spelling Dictionary (`DCTMAIN`) — path and name of the `.dic` file used for checking the spelling of text in drawings.

Log Files Paths and Files

Log File Mode (LOGFILEMODE) — toggles the creation of *.log* files, which record all keystrokes entered at the command prompt.

Log File Name (LOGILENAME;read-only, set by BricsCAD — name of the file with which to record the command-line text, after the LogFileOn command turns on logging. When this entry is blank, log files are named after the drawing.

Log File Path (LOGFILEPATH) — path to folder that stores log files.

Cloud Log (CLOUDLOG) — determines whether 24/7 log files are stored (formerly called “chapoolog”).

File Dialogs

Remember Folders (REMEMBERFOLDERS) — default path for file-oriented dialog boxes, such as Open and Save As. Options are:

- ▶ 0 — **Use Start In Path** uses the Start In path specified in the Windows Properties dialog box
- ▶ 1 — **Use Last Path Used** uses the path stored in the Recent Path option

Use Standard Open File Dialog (USESTANDARDOPENFILEDIALOG) — toggles the display of path icons in file-related dialog boxes.

Drawings Path (DRAWINGPATH) — path to the folders in which to search for *.dwg* drawing files.

Blocks Path (BLOCKSPATH) — path to the folders in which search for *.dwg* block files.

Thumbnail Preview Image Size (THUMBSIZE) — ranges from 64x64 pixels to 2560x2560.

Preview Window in Open Dialog (PREVIEWWNDINOPENDLG) — toggles the display of the preview image of *.dwg* files in dialog boxes.

Preview Type (PROJECTNAME) — view to show in thumbnails:

- ▶ **Last Saved View** of the drawing last time it was saved
- ▶ **Home View** of the drawing in its default state

Places Bar (Windows only)

First thru Fourth Folder (PLACEBARFOLDER1 thru PLACEBARFOLDER4) — specifies the order which places appear in file-related dialog boxes. Choose from the following places:

PlaceBarFolder	Meaning
1	Desktop
2	My Computer
3	My Documents
4	Favorites
5	Network
6	My Recent Documents

Drawings Path (DRAWINGPATH) — lists additional folders for Open and SaveAs commands.

Reusing User Preferences

After you customize BricsCAD through the instructions given by this book, you can save the changes to *.arg* files. You can make changes to how BricsCAD looks and works, and save them, and then reuse them later.

For example, here are some reasons why you might want to create *.arg* files:

Projects. It may be necessary to customize BricsCAD for specific projects. Through *.arg* files, you can BricsCAD to look in project-specific folders, use specified plot settings, and so on.

Portability. Through *.arg* files, you can make any copy of BricsCAD your own. Import the *.arg* file into the CAD program running on another computer, activate it, and when BricsCAD restarts, it will look just like your own. In this case, you might be setting the size of the cursor and colors of user interface elements, such as the background color of the screen.

The *.arg* file stores settings in the following areas of BricsCAD:

- All settings that are reported by the Settings dialog
- All settings that are stored in the Windows registry
- Plotter settings used for model space
- Settings made in dialog boxes
- Project settings
- Recent paths
- Status bar settings
- Tip of the Day setting
- Toolbar settings

To make it easier to work with *.arg* files, BricsCAD includes the User Profile Manager utility program. It displays a dialog box that lets you save and recall settings. More importantly, the dialog box's **Start** button launches BricsCAD with the selected user profile. (This is like using the `/p` switch discussed earlier.)

TIP The *.arg* file format used by BricsCAD is compatible with AutoCAD. This means you can swap user profiles between AutoCAD and BricsCAD.

LAUNCHING THE USER PROFILE MANAGER

You launch the manager from inside or outside BricsCAD:

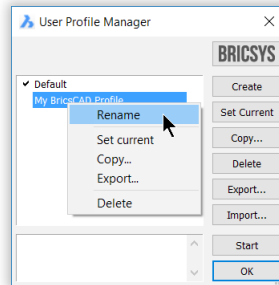
- Inside of BricsCAD through **Tools | User Profile Manager**, or enter the **ProfileManager** command.
- Outside of BricsCAD through the Windows **Start** button: **All Programs | Bricsys | User Profile Manager**. (This option is not available on Linux or Mac.)

Notice the dialog box that appears. On the right is a column of buttons for creating and editing user profile names. The buttons have the following meaning:

Create... creates new profiles, prompting you for a name. It uses the current settings found in BricsCAD.

Set Current sets the selected profile as the current one. The next time BricsCAD starts, it will use this profile.

Copy... makes a copy of a selected profile.



User Profile Manager is an external program

Delete removes the profile. (In-use profiles cannot be erased.)

Export... exports the selected profile as an .arg file.

Import... imports .arg files in to this copy of BricsCAD.

Start launches BricsCAD with the selected user profile.

OK closes the dialog box.

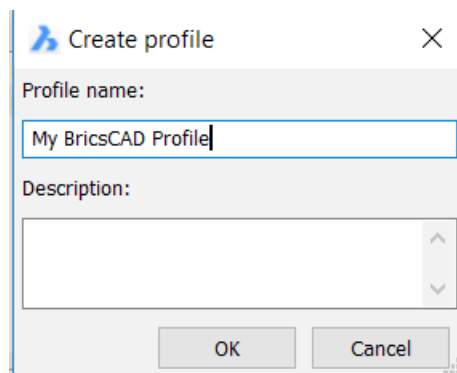
To rename a profile, right click its name, and then choose **Rename** from the shortcut menu.

There is one more button: clicking **Bricsys** opens the BricsCAD Web site in your computer's default Web browser.

Using the Profile Manager

To use the user profile manager, follow these steps:

1. Make changes to BricsCAD, such as in the Setting dialog box or in the Plot dialog box.
2. From the **Tools** menu, choose **User Profile Manager**.
3. In the User Profile Manager dialog box, follow these steps:
 - a. Click **Create** to create a new profile. Notice the Create Profile dialog box:



Naming the new profile

- b. Enter a name and description of the profile. It is a good idea to list the specifics of the profile in the Description field, because that is the only way you will be able to determine what is in this profile.
 - c. Click **OK**.
4. With the profile saved, you now have two primary options:
 - Launch another copy of BricsCAD with the new profile — click **Start**.
 - Simply exit the dialog box — click **OK**.

TIP You can create multiple desktop icons for BricsCAD, each associated with a different user profile. Use the `/p` argument, as described earlier under “Command Line Switches.”

CHAPTER 4

Adapting the User Interface To You

Some of BricsCAD's customization fall outside the realm of the Settings and Customization dialog boxes, as described in other chapters of this book. Some aspects of the user interface can be changed directly in the interface, instead indirectly through a dialog box.

In this chapter, you learn how to directly change the looks of drawing tabs, LookFrom widget, ribbon, and command bar.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Accessing commands in the drawing tabs
- How to modify the LookFrom widget
- Taking full advantage of the command bar
- Repositioning the ribbon

Customizing the Command Line

The command line is the primary way in which BricsCAD communicates with you. Here you can enter the names of commands, their options, and specify positions in the drawing. The command line (a.k.a. “command bar” or “command panel”). is usually found along the bottom of the BricsCAD window, but it doesn’t need to be there or even look the way it does.



Command bar's default location at the bottom of the BricsCAD window

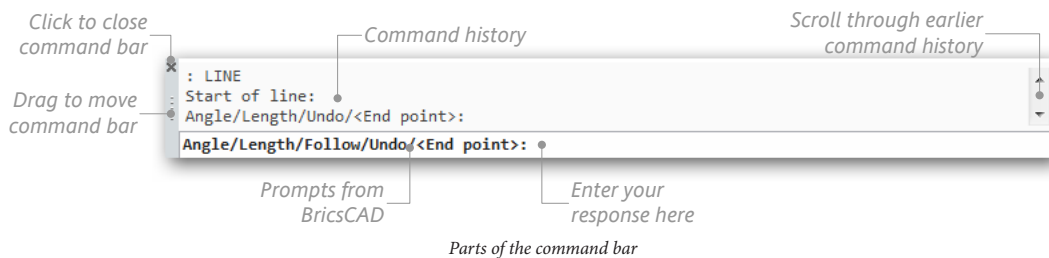
Since this is a book about customization, let’s see how we can customize the command line. For instance, you can make the text look different. Here I changed the font to Bauhaus and the size to 24 points (1/3" tall).



Command bar with different font and font size

THE PARTS OF THE COMMAND BAR

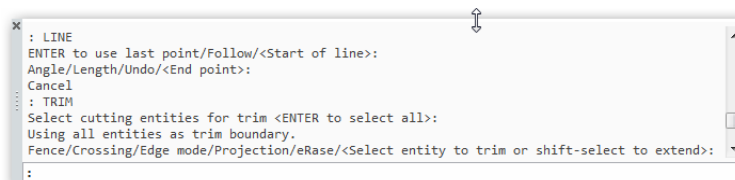
These are the parts to the command bar, and the controls that are embedded in it:



Resizing and Hiding the Command Line

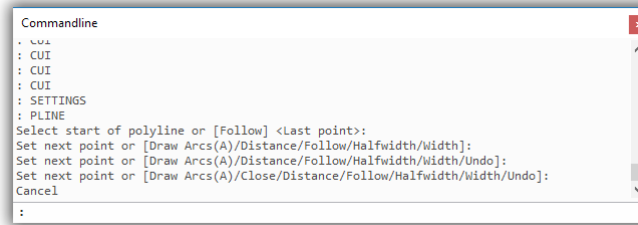
The command bar typically displays three or four lines of *history*, which is the text of previously displayed prompts. When you need to see more lines of history, then you have these choices:

- ▶ Drag the bar’s top border to stretch it taller or shorter. For the exact location to do this, see the double-ended arrow cursor shown below.



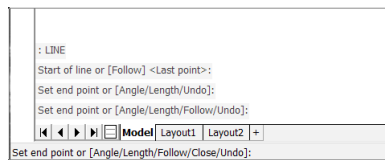
Resizing the command bar

- ▶ Drag the bar away from its docked position, and then resize it, as shown below.



Floating command line dragged away from its docked position

You can turn off the Command bar with the **CommandLineHide** command, but I don't recommend this; there is no good reason to do so — the BIM workspace does this by default, and I find it unhelpful! When the command line is turned off, it still displays the command history in the drawing area. It looks like this:



Prompts scrolling into the drawing area

After a few moments, the history fades from view. To change the number of lines of history appearing in the drawing area, enter the **CliPromptLine** variable and then change its value. Here are the valid values:

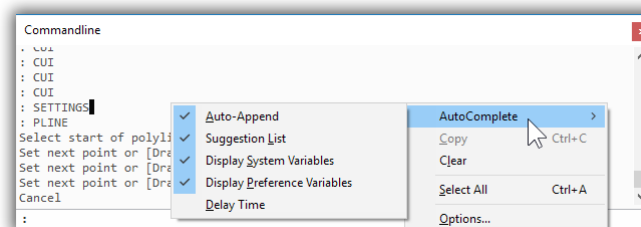
CliPromptLine	Meaning
0	Turns off the prompt history
1	Minimum number of lines
4	Default number of lines of prompt history
64	Maximum number

When the command bar is turned off, the current prompt is always displayed on the status bar.

The **CommandLine** command turns it back on. You can use the **Ctrl+9** (Cmd+9 on Macs) shortcut keystroke to toggle its visibility.

Changing Command Bar Actions

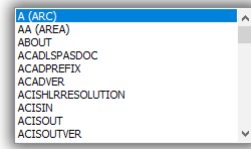
The command line can be made to present different kinds of information. The easiest way to do this is by right-clicking the command line:



Shortcut menu presenting command line options

The options are sometimes governed by the variables, as described here.

AutoComplete. The **AutoCompleteMode** variable controls the actions of the pop-up menu that displays a list of command and variable names that begin with the same letter(s) — auto-complete mode. Illustrated below is what happens when users type an “a”: they can then select one of the names or aliases from the list, and then BricsCAD executes the command or variable.



Autocomplete entries for 'a'

The default value of this variable is 15, which is the sum of 1+2+4+8, representing the options that are turned on by default. When set to 0, auto-complete is turned off.

AutoCompleteMode	Meaning
0	Disabled
1	Enable auto-complete (default)
2	Auto-append names (default)
4	Display names that begin with the same letter(s) (default)
8	Display icons (unsupported; default)
16	Exclude the display of system variables
32	Display preference variables found only in BricsCAD

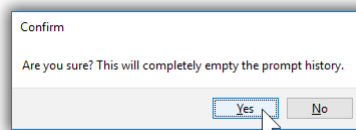
While the display of icons seems to be an option, BricsCAD does not do this at the time of writing. The value (8) is included for compatibility, so that routines imported from other DWG editors will work smoothly.

Delay Time. The **AutoCompleteDelay** variable specifies how long BricsCAD should wait before displaying the auto-complete list. The time is measured in seconds.

AutoCompleteDelay	Meaning
0	Minimum value; no delay
0.3	Default value; brief delay
10	Maximum value; long delay

Copy copies the selected text from the command line to the Clipboard. Some text has to be selected for this command to become available.

Clear erases all text from the command line, including the history. BricsCAD asks if you are sure:



Checking if you really want to erase the command history

Select All selects all text in the command line, including the *history*.

The history consists of previous command entries. By default, the program retains 256 lines of history; you can change this number to be larger or smaller with the **ScrHist** variable.

Paste pastes text from the clipboard into the command line. This option appears only when the cursor is next to the ':' prompt, and the Clipboard contains data that can be pasted; text, not images.

You can paste text previously copied to the Clipboard with the Copy option, as well as text from other programs formatted as scripts (see Chapter 20) or LISP routines (see Chapter 21).

Options displays the Command Line section of the Settings dialog box. Many of the settings are discussed next.

TIP To record the command history in a file, use the **LogFileOn** command to begin saving it, and then use **LogFileOff** to turn off the recording.

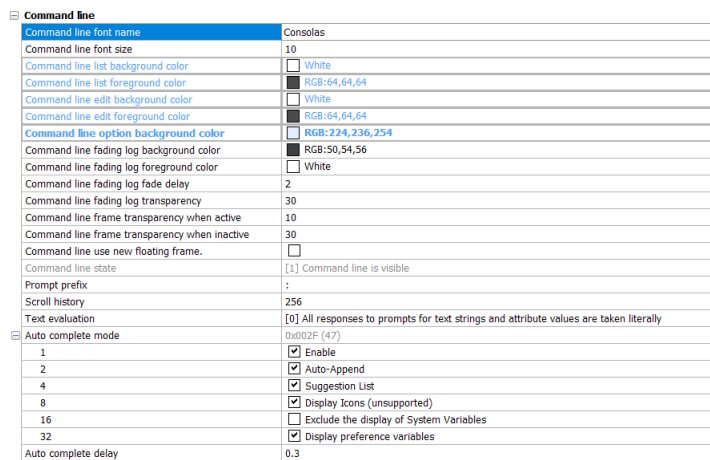
Use the **LogFileName** to specify a different name for the .log file (the default is the name of the drawing), and **LogFilePath** to change the folder in which the file is stored (the default is C:\Users\<login>\AppData\Local\Bricsys\BricsCAD\V20x64\en_US).

The log file system is independent of the **ScrHistory** variable's limit on lines, and so keeps recording everything until you turn it off.

Additional Command Line Variables

You change the look of the command line and its close cousin Prompt History through variables in the Command Line section of the Settings dialog box. The settings descriptions use some unusual terms:

Command Bar Element	Term Used by Settings	Abbreviation Used by Variables
Command Line	Edit	CmdLine
Prompt History	List	CmdLineList
Text	Foreground	Fg (Background = Bg)
Text scrolling into drawing area	Fading log	FadingLog
Text in drawing area background	Frame or floating frame	CmdLineFrame

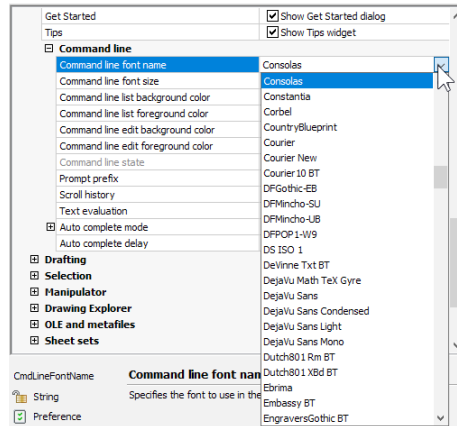


Command Line section of the Settings dialog box

Command Line Font Name. The **CmdLineFontName** variable specifies the font used by the command line and Prompt History window. Any TrueType font installed on your computer can be used.

CmdLineFontName	Meaning
Consolas	Default font name

To change the font displayed by the command line and text window, click the field, and then choose a font from the droplist.



Selecting a font for the command line

Command Line Font Size. The **CmdLineFontSize** variable sets the size of the text in the command bar and Prompt History window. I would like it I could set different font sizes for the command line (smaller) and Prompt History window (larger), but this is not possible.

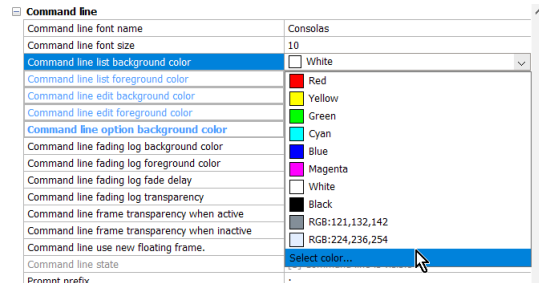
TIP To display the Prompt History window, press F2.

Command Line List Background Color. The **CmdLineListBgColor** variable specifies the background color of the prompt history window (“list”). The default is 250,250,250 — a light gray.

The triplet of numbers specify levels of red, green, and blue. The numbers indicate the strength of each component color, and ranges from 0 (black or no color) to 255 as the maximum strength for each color.

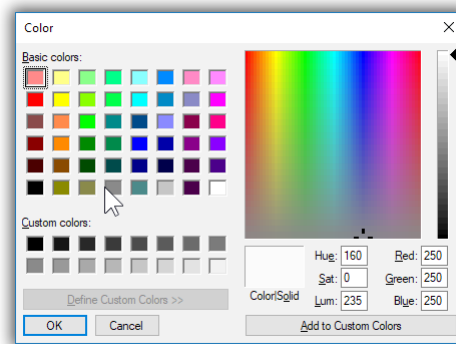
CmdLineEditBgColor	Meaning
0,0,0	Black
255,0,0	Red
0,255,0	Green
0,0,255	Blue
250,250,250	Light gray (default value)
255,255,255	White

To change the color, click the field in the Settings dialog box, and then choose a color.



Choosing a color

If the color you want isn't in the list, then click **Select Color** to access the Color dialog box. Choose a color, and then click **OK**.



Selecting a color

(Early releases of BricsCAD used hexadecimal numbers to assign colors to these variables. Hexadecimal numbers are natural to computer systems, are base 16, and are signaled by the # prefix, such as #fe00ee.)

Command Line List Foreground Color. The **CmdLineListFgColor** variable specifies the foreground color of the history window. By “foreground,” BricsCAD means the color of the text. The default is 64,64,64 — a very, very dark blue.

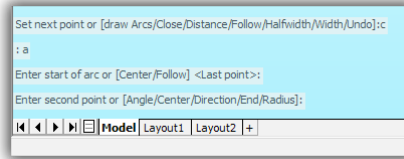
Command Line Edit Background Color. The **CmdLineEditBgColor** variable specifies the background color of the command bar (“edit”). The default is 250,250,250, a light gray.

Command Line Edit Foreground Color. The **CmdLineEditFgColor** variable sets the foreground (text) color of the command line panel. The default is 32,32,32.

(NEW IN V20) Command Line Option Background Color. The **CmdLineOptionBgColor** variable sets the background color of option names, which can be selected with the cursor.

Command Line Fading Log Background Color. The **CmdLineFadingLogBgColor** variable sets the background color for text that appears in the drawing area (“fading log”).

When the command bar is off, the last four lines of command text appear in the drawing area. After a few seconds, the text fades away. When you next enter a command or pick an option, the on-screen text reappears. This works whether or not cleanscreen is on.



Command line text in the drawing area

Command Line Fading Log Foreground Color. The **CmdLineFadingLogFgColor** variable sets the color of the text that scrolls in the drawing area.

Command Line Fading Log Fading Delay. The **CmdLineFadingLogFadeDelay** variable determines how long the text appears in the drawing area before it starts to fade away; default is 2 seconds.

Command Line Fading Log Transparency. The **CmdLineFadingLogTransparency** variable determines the transparency of text that appears in the drawing area; default is 30%, where 0% is opaque.

Command Line Frame Transparency When Active. The **CmdLineFadingActiveTransparency** variable determines the transparency of the background (“fading frame”) to text that appears in the drawing area; default is 10%, where 100% is fully transparent.

Command Line Frame Transparency When InActive. The **CmdLineFadingInactiveTransparency** variable determines the transparency of the background when the command line is not being used; default is 10%, where 100% is fully transparent.

(NEW IN V20) Command Line Use New Floating Frame. The **CmdLineUseNewFrame** variable toggles the appearance of a new frame for text that scrolls into the drawing area; this is a test variable and may be changed in the future; default is off.

Command Line State. The **ClsState** (read-only) variable reports whether the command bar is open, or not. Because it is a read-only variable, users cannot change it.

ClsState	Meaning
0	Command bar is hidden
1	Command bar is visible (default)

TIP Use **Ctrl+9** (Cmd+9 on MacOS) to turn the command line on and off quickly.

Prompt Prefix. The **CmdLnText** variable specifies the prompt character(s) displayed by the command bar. While the default is a colon (:), BricsCAD allows you to change it to something else, such as AutoCAD's traditional prompt, 'Command:'.

CmdLnText	Meaning
:	Prompt text displayed by command bar

Scroll History. The **ScrlHist** variable determines how many lines of command history BricsCAD remembers. This affects both the command line and the Prompt History window.

ScrlHist	Meaning
0	No history is kept
256	Default value
2147483647	Maximum value

Even More Command Line Variables

There are additional variables located elsewhere in the Settings dialog box:

CliPromptLines variable determines how many lines of command history appear in the drawing area when the Command panel is turned off.

LastPrompt variable reports the name of the command entered most recently. It is read-only.

PromptOptionFormat variable determines how command options are displayed on the command line and in the prompt menu; option 4 is meant for international versions of the software:

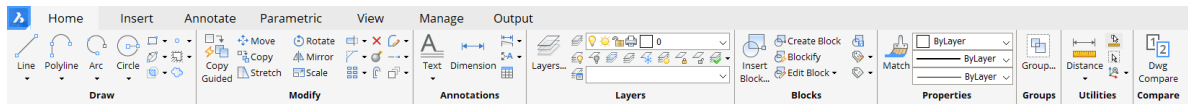
PromptOptionFormat	Meaning
0 (default)	Show description only Set end of arc or [draw Lines/Angle/CEnter/CLose/...
1	Show keywords only Set end of arc or [Line/Angle/CEnter/CLose/...
2	Show description, with keywords in brackets Set end of arc or [Draw lines(Line)/Angle/Center(CEnter)/Close(Close)/...
3	Show description, with shortcuts in brackets Set end of arc or [Draw lines(L)/Angle/Center(CE)/Close(CL)/...
4	Show local keyword, with global keyword in brackets

PromptOptionTranslateKeywords variable toggles the use of international commands. When off, the underscore (_) prefix is not needed during command input; default = on.

TIP To change the color of the drawing area, use the

Customizing the Look of the Ribbon

The ribbon is like a series of overlapping toolbars, where a row of *tabs* segregate the overlapping “toolbars” that group similar functions. A tab is further segregated into a row of adjacent *panels*, each panel containing a group of buttons, flyouts, and/or droplists — just like toolbars!)

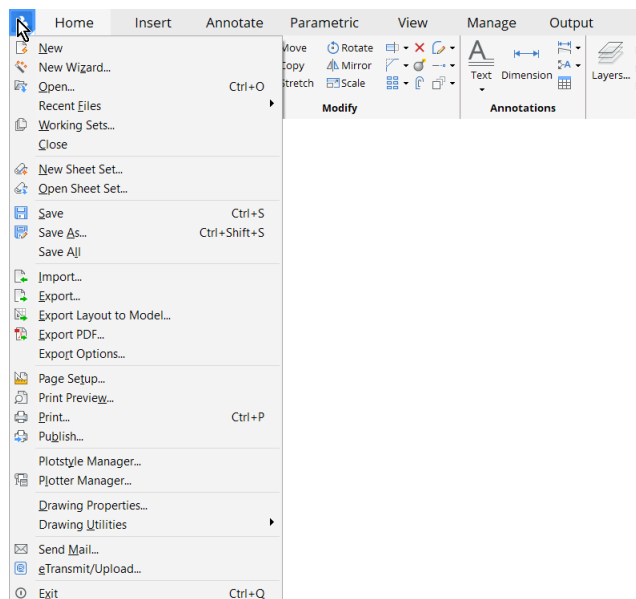


Ribbon displayed by the 2D Drafting workspace

Because Bricsys wrote its own version of the ribbon interface, it is equally available on the Windows, MacOS, and Linux versions — unlike most other CAD systems. You customize the ribbon through the Customize command. (See Chapter 9 for tutorials on creating and editing ribbon tabs and panels with the Customize command. The chapter also includes a complete panel design reference.)

When no drawing is open, all buttons on the ribbon turn gray, to indicate that they are unavailable.

The blue **B** item is not part of the ribbon, but is a menu that accesses file-related function, such as opening and saving drawings.



B button displaying file-related functions

HANDLING THE RIBBON

BricsCAD comes with several ribbons. To switch between them, you don't use the Ribbon command, as you might think. Instead, you use the **Workspaces** command. Ironically, a Workspaces droplist is by default not available on the ribbon, although it is on the status bar.

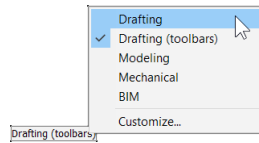
To switch between ribbons, you can change the workspace using a toolbar, the status bar, or the command line, as follows:

- ▶ **Workspaces droplist** on a toolbar:



Choosing a workspace from the toolbar

- ▶ **Workspaces button** on the status bar:



Choosing a workspace from the status bar

- ▶ **Workspace command** in the command bar:

: workspace

Workspace: setCurrent/SAveas/Rename/Delete/SEttings/? <setCurrent>: ?

Workspace Name

BIM

Drafting

Drafting (toolbars)

Mechanical

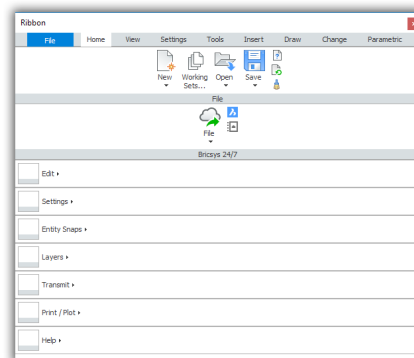
Modeling

Modeling (toolbars)

The names of workspaces changed with BricsCAD V19 and V20:

Old Workspace Name	New Workspace Name	Default Display
2D Drafting	Drafting	Displays the ribbon
...	Drafting (Toolbars)	Displays the menu bar and toolbars, no ribbon
...	Modeling	Displays the ribbon
3D Modeling	Modeling (Toobars)	Displays the menu bar and toolbars, no ribbon
BIM	BIM	Displays the ribbon
Mechanical	Mechanical	Displays the ribbon

You can drag the ribbon away from its docked position; the floating ribbon looks like this:



Floating ribbon

While it is possible to float the ribbon, I find no useful purpose to it.

The **Ribbon** command displays the ribbon; **RibbonClose** closes it.

Related System Variables

RibbonState (read-only) variable reports whether the ribbon palette is open or closed. Users cannot change this variable, because it is read-only.

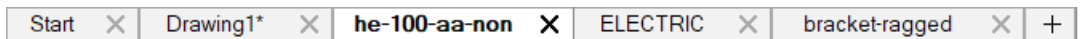
RibbonState	Meaning
0	Ribbon is closed
1	Open

RibbonDockedHeight variable determines the height of the ribbon when docked. The height is measured in pixels. I recommend that you do not change this number.

RibbonState	Meaning
0	Ribbon sizes itself to the height of the selected tab
120	Default value
500	Maximum value

Customizing the Look of Drawing Tabs

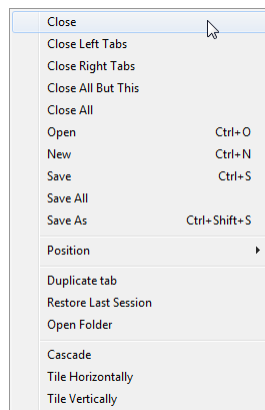
Drawing tabs let you switch quickly between open drawings. BricsCAD calls them “document tabs.”



Each tab names the open drawing

(NEW IN V20) The first tab accesses the Start screen, which you can also access with the **GoToStart** command. This tab can be turned off by clicking the gray x.

As well, the tabs provide a shortcut to file-related commands, such as Open and Close, plus some commands you won't find elsewhere in BricsCAD. Right-click any drawing tab. although the list of commands you see depends on which tab you right-click.



Shortcut menu of commands for controlling tabs and drawings

Among the unique commands, I find Open Folder particularly useful. The unique commands perform these functions:

- ▶ **Close Left Tabs** — closes all drawings to the left of this tab. This is useful for closing older drawings. Drawings that were opened earlier tend to appear at the left end of the row of tabs
- ▶ **Close All But This** — closes all other drawings, except the current one. I find this useful when I open an entire folder's worth of drawings, and then want to keep just one open
- ▶ **Save All** — saves all drawings at once
- ▶ **Duplicate Tab** — makes a copy of the current drawing, and then names it *copy_name.dwg*
- ▶ **Open Folder** — opens the folder from which the drawing was opened

Related System Variables

The look of drawing tabs is customized through variables. Be aware that the changes made to these variables do not take effect until the next time BricsCAD is started. So to apply the change(s), exit BricsCAD, and then start it again.

DocTabPosition variable — places the tabs at the top, bottom, left, or right of the drawing area.

DocTabPosition	Meaning
0	Position tabs at the top of the drawing area (default)
1	Position tabs at the bottom of the drawing area
2	Position tabs at the left edge of the drawing area
3	Position tabs at the right edge of the drawing area

The **ShowDocTabs** variable turns the tab row on and off.

ShowDocTabs	Meaning
0	Does not display tabs
1	Displays tabs (default)

You can also access these variables in the Settings dialog box.

Customizing the Look From Control

BricsCAD has a LookFrom widget in the upper right corner of the drawing area. I find it very useful for quickly changing the 3D viewpoint.



LookFrom widget at rest

Click on one of the triangles to see a 3D model from a different point of view:



Viewing a 3D model from an isometric viewpoint

Here is how to use it:

1. Pass the cursor over the widget. Notice that small triangles appear, as does the preview image of a simple chair appears.



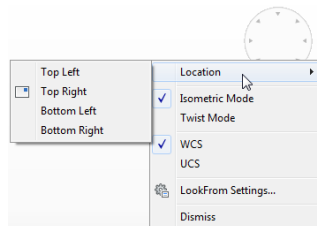
Cursor activating the LookFrom widget

2. Pause the cursor over a triangle:
 - You get a preview of what the 3D view will look like via the chair icon
 - A tooltip tells you name of the view, such as “Top Front Left:
 - The green dot indicates the cursor position, kind of like a laser pointer.
3. Click the triangle to change the 3D viewpoint.

TIPS To see the bottom views: hold down the **Ctrl** (or **Cmd** in Mac) key while the cursor is in the LookFrom widget.

To return to the home (default) view: click the center of the LookFrom control. (This is particularly helpful in Twist mode.) Or, press the **Home** key on the keyboard to return to the home view.

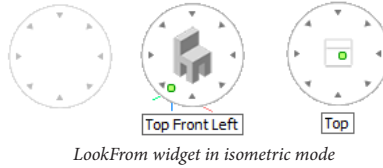
There are two ways to customize the way the LookFrom control operates. The easier one is right-click the control, and then choose an option from the shortcut menu:



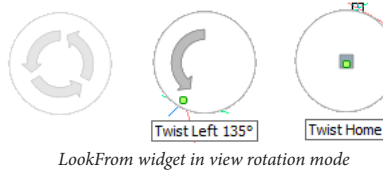
Shortcut menu of LookFrom options

Most of the options in the shortcut menu are straight-forward, but I do want to explain the difference between Isometric and Twist modes:

- ▶ **Isometric** mode is like using the Viewpoint or View commands; clicking a triangle jumps to the viewpoint



- ▶ **Twist** mode is like using the RtRotF (real time view rotation) command; clicking an arrow *rotates* the viewpoint



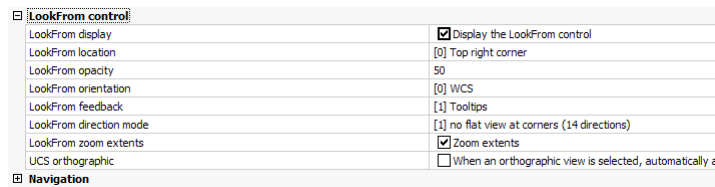
LOOKFROM COMMAND

The other way to customize the widget is through the **LookFrom** command, from which you can turn the widget off (and on) and access its settings:

: **lookfrom**
 LookFrom [ON/Off/Settings] <ON>: (Enter an option)

The **ON** and **Off** options turn the widget on and off.





The **Settings** option opens the Settings dialog box at the LookFrom section.



LookFrom variables in the Settings dialog box

With Settings, you can adjust properties of the widget, such as its translucency and position.

Of particular interest is the number of isometric viewpoints it can display, which is set through “Direction Mode” or the **LookFromDirectionMode** variable. The following table illustrates these modes:

LookFromDirectionMode	Number of Views
0 	6 orthogonal views
1 	14 views; no flat views of corners (default)
2 	18 views; top down corners
3 	26 views; eight top down corners

Related System Variables

The following variables control the look and action of the LookFrom widget. Some of the names begin with “NavVCube.” This is AutoCAD’s name for its “navigation view cube” widget.

The **LookFromFeedback** variable toggles the feedback display between tooltips and the status bar:

LookFromFeedback	Meaning
0	No feedback
1	Tooltips near the LookFrom widget (default)
2	On the status bar

The **LookFromDirectionMode** variable selects the type of display, as illustrated above.

The **LookFromZoomExtents** variable toggles the use of Zoom Extents when a viewpoint changes:

LookFromZoomExtents	Meaning
0	Zoom is unchanged
1	Zoom extents is executed when the view direction changes (default)

The **NavVCubeDisplay** variable toggles the display of the LookFrom widget:

NavVCubeDisplay	Meaning
0	Not displayed
1	Displayed (default)

The **NavVCubeLocation** variable positions the widget in one of the four corners of the drawing area:

NavVCubeLocation	Meaning
0	Top right corner of the drawing area (default)
1	Top left corner
2	Bottom left corner
3	Bottom right corner

The **NavVCubeOpacity** variable determines the “see through-ness” of the widget:

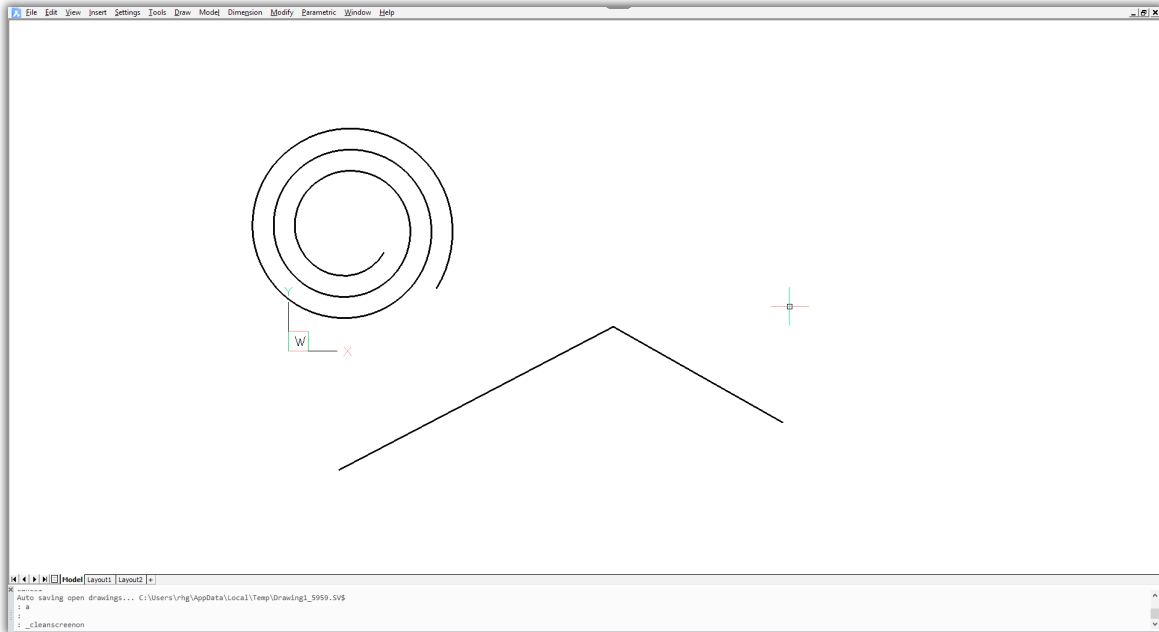
NavVCubeOpacity	Meaning
0	Invisible
50	Semi-transparent (default value)
100	Opaque

The **NavVCubeOrient** variable determines whether view changes are relative to the world coordinate system or the current user-defined coordinate system:

NavVCubeOrient	Meaning
0	Relative to WCS (default)
1	Relative to UCS

Maximizing the Drawing Area

To maximize the screen means to minimize the number of user interface elements. Use the **CleanScreenOn** command to maximize the drawing area, **CleanScreenOff** to return the UI to normal. Pressing **Ctrl+0** (**Cmd+0** on MacOS) does the same thing much more quickly.



BricsCAD's drawing area maximized

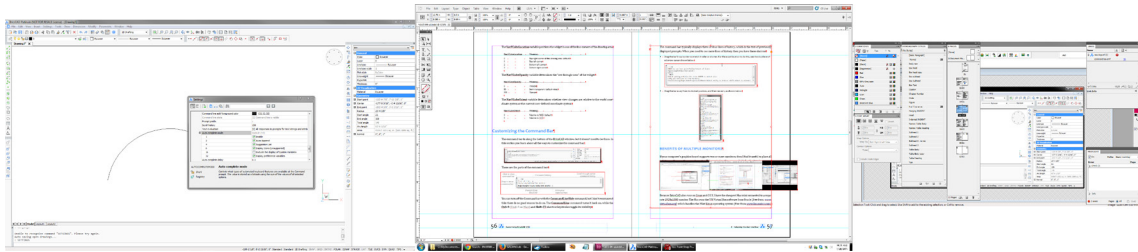
The **CleanScreenOptions** variable specifies which UI elements to keep on during clean screen mode. The default value is 15, which means the drawing tabs, panels, toolbars, and ribbon are hidden, while the command line, status bar, and menu bar remain visible.

CleanScreenOptions	Meaning
0	Hide no elements
1 (default)	Hide document (drawing) tabs
2 (default)	Hide dockable panels (palettes)
4 (default)	Hide toolbars
8 (default)	Hide ribbon
16	Hide command line panel (bar)
32	Hide status bar
64	Hide menu bar

USING MULTIPLE MONITORS

Most computers, even today's laptops, support two or three monitors. I find it useful to place all palettes and bars on the second monitor. This maximizes the area available for the BricsCAD drawing screen.

Indeed, when I write these books, I have five screens surrounding me. My Windows 7 workstation has three:



Three monitors at different resolutions

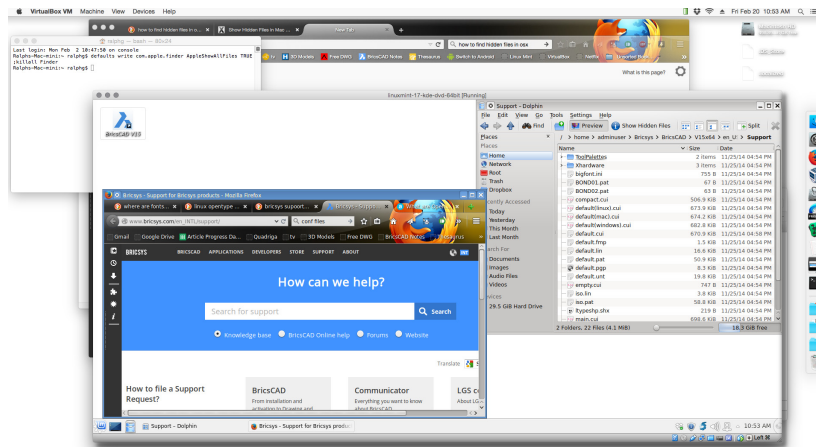
- ▶ Main monitor (2048x1152 resolution) for the InDesign desktop publishing software
- ▶ Second monitor (1360x768) for InDesign's many palettes and PaintShop Pro for editing figures
- ▶ Third monitor (1920x1080) for displaying BricsCAD

I find it beneficial to have the secondary monitor run at a lower resolution, because it makes the user interface larger and so easier to read.

To check how BricsCAD works with Windows 10, I have a separate laptop running that operating system.

Because BricsCAD also runs on Linux and MacOS, I have the cheapest Mac mini connected to a separate 1920x1080 monitor. The Mac runs the VM Virtual Box software from Oracle (free from <https://www.virtualbox.org>) which handles the Mint Linux operating system (download free from <https://www.linuxmint.com/download.php>).

In the figure below, the “linuxmint” window is running on the Mac desktop.



Mac running MacOS natively and Linux in a virtual machine window

When I make screen grabs on them or the Mac-Linux system using WinSnap, pCloud captures the images automatically, and then places them in a folder on my Windows 7 computer for placement in the InDesign document.

TIP Even if your computer’s graphics board is limited to working with one (or two) monitors, there is a workaround. **DisplayLink** is a USB dongle that allows you to add a monitor without needing a video port. Windows sees the dongle as another screen.

Several manufacturers make the hardware for under \$100; see <http://www.displaylink.com>. Software is included that runs on the computer to redirect the “second screen” graphics to the dongle.

Customizing Other UI Elements

To customize the look of other user interface elements, see the following chapters:

To change the look of...	See Chapter...
Menus	6
Toolbars	7
Quad cursor	12
Rollover tooltips	13
Palettes	15

PART II

Working with the Customize Dialog Box

Introduction to the Customize Dialog Box

The Customize dialog box is the primary place in which to customize BricsCAD — to change the way it looks and works. This dialog box handles many customization tasks, including the following ones:

- ▶ Creating and changing toolbars, menus, shortcut menus, mouse and digitizer buttons, ribbon tabs and panels, workspaces, Quad cursor, rollover properties, and tablet menus
- ▶ Writing and editing macros
- ▶ Creating and modifying keyboard shortcuts, command aliases, and shell commands
- ▶ Importing and exporting full and partial menu files in *.cui* , *.cuix* , *.icm* , *.mns* , and *.mnu* formats

CHAPTER SUMMARY

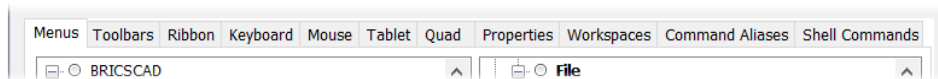
The following topics are covered in this chapter:

- Touring the Customize dialog box
- Understanding the CUIX customization file
- Accepting and rejecting changes to customization
- Reviewing the XML format
- Working with partial CUI files

The Customize dialog box is your one-stop shop for customizing ribbons, macros, and more, and so here are some of the ways to access this important dialog box:

- ▶ Enter **Customize** at the command prompt
- ▶ Or, enter the alias **cui** (this is my preferred method)
- ▶ Or, from the **Tools** menu, select **Customize**
- ▶ Or, right-click any toolbar or ribbon, and then from the shortcut menu select **Customize**

This chapter introduces the Customize dialog box by providing an overview of its functions. Each of the nine following chapters describe in turn how to customize the user interface as indicated by the tab name — Menus, Toolbars, Ribbon, and so on.



Tabs segregating customization of interfaces elements

The chapters follow roughly in order of how the customizations appear in the dialog box:

Chapter	Tab Name(s)	Topic
6	Menus	Menus and context (right-click) menus
7	Toolbars	Toolbars
8	<i>Applies to most tabs *</i>	Macros and diesel code
9	Ribbon	Ribbon tabs and panels
10	Keyboard	Keyboard shortcuts
11	Mouse, Tablet	Mouse buttons, double-click actions, and tablet menu
12	Quad	Quad cursor
13	Properties	Rollover properties
14	Workspaces	Workspaces
15	Command Aliases, Shell Commands	Aliases and and shell commands

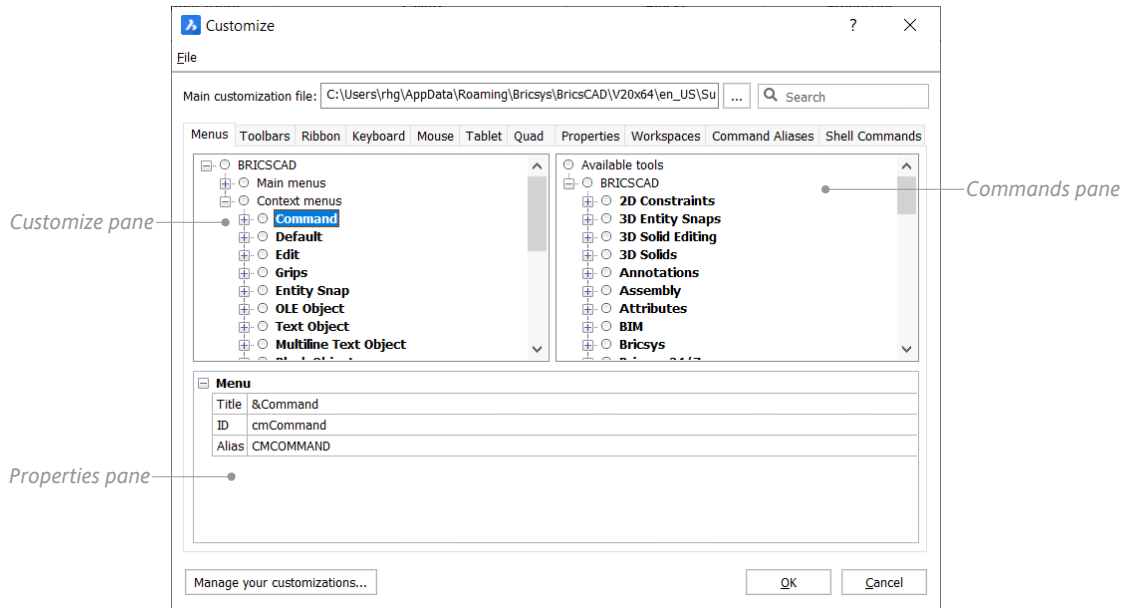
*) Macros and Diesel code are used by menu and menu items, toolbar buttons, ribbon buttons, mouse actions, tablet menus, and Quad buttons

BricsCAD stores information about the ribbons, macros, and all the rest in *.cui* files, where “cui” is short for *customize user interface*. The program also reads menu files from AutoCAD (*.cuix*, *.mnu* and *.mns*) and IntelliCAD (*.icm*).

Aliases and shell commands are stored in a different format and in *.pgp* files.

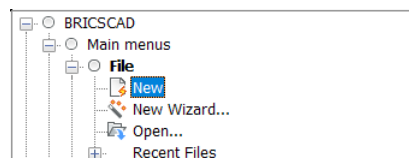
Touring the Customize Dialog Box

The Customize dialog box has three primary areas (a.k.a. *panes*): customize, tools, and properties.



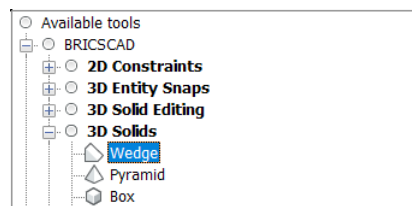
Customize dialog box

- ▶ **Customize** pane (on the left) lists items that can be customized; the content of this pane varies, depending on which tab is selected, whether “Menu” or “Shell Commands”



Customize pane, found in the left half of the dialog box

- ▶ **Tools** pane (on the right) lists all of the commands found in BricsCAD; they are sorted according to menu order, for example, all file-related commands are listed under “File”



Tools pane, found in the right half of the dialog box

- ▶ **Properties** pane (at the bottom) edits the properties, such as title, Diesel code, Help text, command macro, and image associated with the currently-selected item; the content of this pane varies, depending on the tab and item

Menu Item	
Title	&NNew
Diesel	
Tool ID	qnew
Help	Creates a new drawing from the current default drawing template
Command	^c^c_qnew
Image	qnew

Properties pane, found at the bottom of the dialog box

ABOUT CUI FILES

Default.cui is the name of the file that defines the menus and toolbars of BricsCAD. It was developed by Autodesk, which then switched to CUIX. The “X” indicates CUIX is a zipped package file that holds all the files needed by the user interface including icons. (It can be viewed with software like PkZIP or 7-Zip.)

CUI files are written in XML, which is short for eXtended Markup Language. XML is a file format that is an extension of HTML, the hyper text markup language used for Web pages. The format is in ASCII, and looks just like HTML, but uses custom tags. Custom tags are used to identify the data stored in CUI files. For example, **<Macro>** identifies the start of a macro, while **</Macro>** marks its end. Because it is like HTML, you can use any Web browser to parse CUI files. Because the format is written in plain text (as shown below), and because the XML specification requires that every piece of data be identified, CUI files are human-readable. Well, in theory; in practice, it quickly becomes tedious trying to read the content of CUI files, because of the repetitive nature of tags.

Bellow you see the first part of BricsCAD's *default.cui* file as written in XML format. This is the first of several dozen lines of a file that's 343 pages long in BricsCAD V20. I've boldfaced some of the macro-related items to help them stand out from the XML tags.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<CustSection xml:lang="en-US">
  <FileVersion IncrementalVersion="2" MajorVersion="0" MinorVersion="3" UserVersion="0"/>
  <Header>
    <CommonConfiguration>
      <CommonItems>
        <PartialMenuFile>C:\Users\rhg\Desktop\partial.cui</PartialMenuFile>
      </CommonItems>
    </CommonConfiguration>
    <WorkspaceRoot>
      <WorkspaceConfigRoot/>
    </WorkspaceRoot>
  </Header>

  <MenuGroup Name="BRICSCAD">
    <MacroGroup Name="File">
      <MenuMacro UID="qnew">
        <Macro>
          <Name>QNew</Name>
          <Command>^c^c_qnew</Command>
          <HelpString>Creates a new drawing from the current default template</HelpString>
          <Image ID="qnew"/>
        </Macro>
      </MenuMacro>

      <MenuMacro UID="new">
        <Macro>
          <Name>New...</Name>
          <Command>^c^c_new</Command>
          <HelpString>Creates a new drawing</HelpString>
          <Image ID="new"/>
        </Macro>
      </MenuMacro>

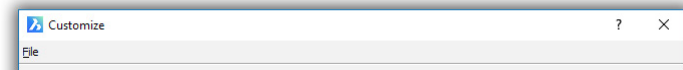
      <MenuMacro UID="newwiz">
        <Macro>
          <Name>New Wizard...</Name>
          <Command>^c^c_newwiz</Command>
          <HelpString>Creates a new drawing using 'Create New Drawing' wizard</HelpString>
          <Image ID="newwiz"/>
        </Macro>
      </MenuMacro>
    </MacroGroup>
  </MenuGroup>
</CustSection>
```

et cetera...

For the remainder of this chapter, I'll describe the parts of the Customize dialog box common to all areas. I begin at the top, and then work my way to the bottom of the dialog box. Later chapters describe unique content.

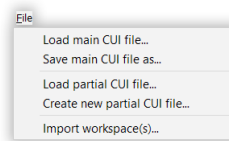
CUSTOMIZE'S MENU BAR

The top of the dialog box sports a menu bar with a single, lonely-looking menu item. The **File** menu lets you open and save full and partial *.cui* files.



The lonely File menu on the menu bar

Click **File** to view the menu:



Options displayed by the File menu

Here are the tasks the menu items perform:

Load Main CUI File opens a *.cui*, *.cuix*, *.mnu*, or *.icm* file:

File Type	Meaning
<i>.cui</i>	Format in which BricsCAD stores customizations; also used by older releases of AutoCAD
<i>.cuix</i>	Format in which AutoCAD currently stores customizations; "x" refers to an archive file, which includes CUI, image files, and so on
<i>.mnu</i>	Format in which the oldest releases of AutoCAD stored customization; also used by many AutoCAD work-alike programs
<i>.icm</i>	Format in which IntellCAD originally stored customizations

Warning A new Main Cui file **completely** replaces the current one, thereby replacing all existing menus, toolbars, shortcut menus, and so on.

ABOUT MAIN AND PARTIAL CUSTOMIZATION

The difference between "Main" and "Partial" customization files is subtle, but crucial:

Main file covers all the user interface elements governed by the Customize dialog box. Change the Main file and the entire user interface changes — except for the portions defined by the Partial files, if any are loaded.

Partial file is like an appendix, an independent addition. Change the Partial file, and only the parts it defines are changed; the rest of the user interface is unaffected. Partial files are used by third-party add-ons (and you!) for customizing the CAD program.

BricsCAD must have a Main file loaded for the user interface to exist; no Partial files need be loaded.

Save Main CUI File saves the current `.cui` file by another name.

TIP You don't need to use the Save Main CUI File option to save changes to customization, because BricsCAD saves them automatically when you click **OK** to exit the dialog box.

Load Partial CUI File opens a partial `.cui` file. The difference between a *main* and a *partial* file is that the contents of a partial `.cui` file are added to the existing user interface. This option is useful for adding menus and toolbars that were customized for add-on applications.

Create New Partial CUI File creates a (nearly) empty `.cui` file. Its sparse content is shown below:

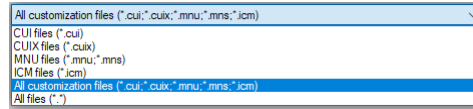
```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<CustSection xml:lang="en-US">
  <MenuGroup Name="DEFAULT"/>
</CustSection>
```

(I would rather that this command creates a partial file of the selected item, such as a toolbar or a menu. The workaround is to copy and paste the items I want into the file. This is not, however, a great solution, considering the complexity of the CUI format.)

Import Workspaces — imports workspace info from `.cui` files.

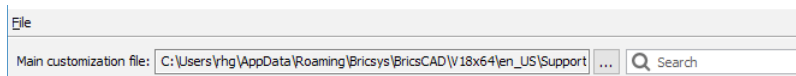
TIP BricsCAD can import menu files from other CAD systems, such `.cuix` from recent releases of AutoCAD, `.mnu` from older AutoCAD systems, and IntelliCAD's `.icm` files.

These are imported through the **File | Load Main CUI File** option. In the **Files of Type** droplist, choose the other format, as illustrated below.




CUI Customization Files

As BricsCAD store customizations in `.cui` files, it shows the name of the current one in the field next to **Main Customization File**:




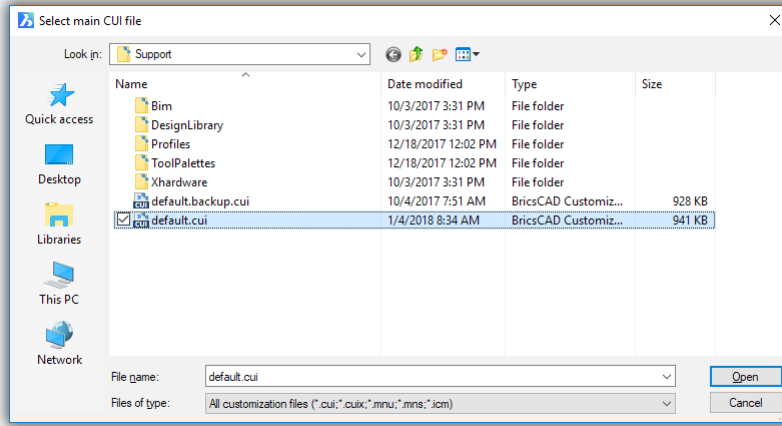
Location of the primary customization file in Windows

This single *default.cui* file contains everything to do with menus, shortcut menus, toolbars, buttons, tablets, Quad cursor, ribbon, and keyboard shortcuts— except for aliases and shell commands, which are saved in a separate *default.pgp* file. Both of the CUI and PGP files can be exchanged with other BricsCAD users and with other CAD programs that read them, such as AutoCAD.

The **Browse**  button (at the end of the **Main Customization File** field) lets you load a different `.cui` file. You want to do this when you need to quickly change the user interface of BricsCAD.

Here is how to do this:

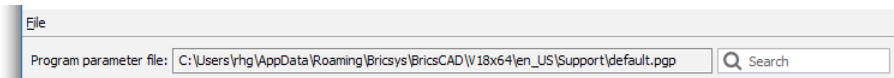
1. Click the  button.
2. Choose a *.cui*, *.cuix*, *.mnu*, or *.icm* file from the Select a CUI File dialog box.



Selecting another user interface by opening another *.cui* file

3. Click **Open**. Notice that all of the user interface of BricsCAD changes immediately!

Aliases and shell command definitions are stored in a different file, called *default.pgp*. “PGP is short for program parameters, but rumor has it that it was nicknamed the “pigpen” file. Click the “Command Aliases” or “Shell Commands” tab to see the location of the *.pgp* file:



Path to the *default.pgp* file

In Windows, the *default.cui* files are found in these folders:

Original files	C:\Program Files\Bricsys\BricsCAD V20 en_US\UserDataCache\Support\en_US
Working copy	C:\Users<login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support

The “original files” are the ones BricsCAD uses when you use the Windows Repair facility, as well as when you click the Revert to Defaults button. The “working copies” are the ones that get modified when you make changes using the Customize dialog box.

In Linux, the *default.cui* files are located in these folders:

Original file	/opt/bricsys/bricscad/V20/UserDataCache/Support/en_US
Working copy	/home/<login>/Bricsys/BricsCAD/V20x64/en_US/Support

In MacOS, the *default.cui* files are stored in these folders:

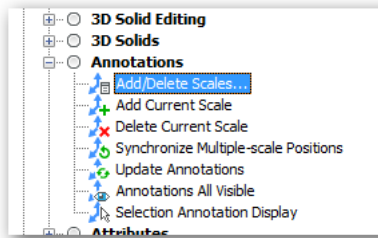
Original file	/Applications/BricsCAD V20.app/Contents/MacOS/UserDataCache/Support/en_US
Working copy	/Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/support

The *default.cui* file has a different name on MacOS and Linux systems:

```
default(windows).cui  
default(linux).cui  
default(mac).cui
```

SEARCH FOR COMMANDS

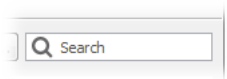
BricsCAD has hundreds of commands, but they are not listed alphabetically in the Customize dialog box's Tools pane, unfortunately. The pane is a bit of a pain by listing the commands in groups of related functions, such as "2D Constraints" and "Annotations."



Guessing which description is for the ObjectScale command

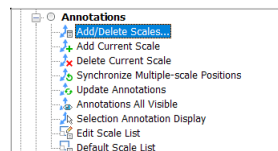
Worse, the Tools pane does not use the actual command names, but command *descriptions*. You'll never find the ObjectScale command by scrolling through the Tools pane, because it is named "Add/Delete Scales...". Sigh. It can be a little bit painful to find the command (a.k.a. tool) you want.

So BricsCAD provides a search field to look for command names.



Searching for commands and descriptions

1. Enter a command name, and it jumps to the command name in the Tools pane. So, you can type "objectscale" and it goes to the "Add/Delete Scales" item.
2. Press **Enter** to find the next use of the command name.

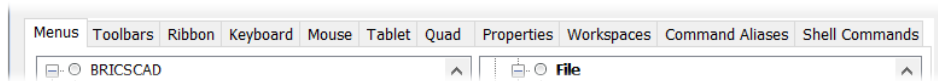


Searching for the ObjectScale command

Disappointingly, the search for command names does not work in the Customize pane.

TABS OF THE CUSTOMIZE DIALOG BOX

The Customize dialog box has a row of tabs that access the primary user interface elements:



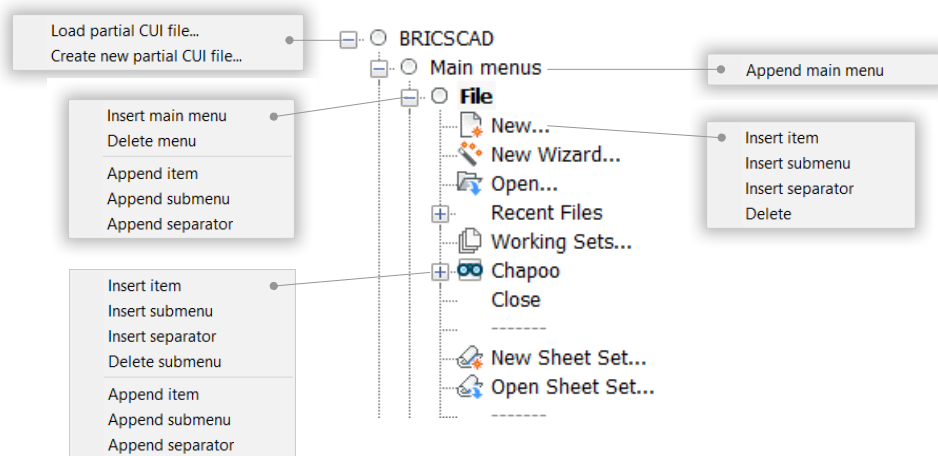
Tabs segregating the customizations for different areas of the user interface

- ▶ **Menu** tab — customizes menus, sub-menus, and shortcut menus
- ▶ **Toolbars** tab — customizes toolbars, buttons, flyouts, and icons
- ▶ **Ribbon** tab — customizes tabs and panels
- ▶ **Keyboard** tab — customizes keyboard shortcuts
- ▶ **Mouse** tab — customized mouse buttons and double-click actions
- ▶ **Tablet** tab — customizes digitizer buttons and tablet menus
- ▶ **Quad** tab — customize the Quad cursor
- ▶ **Properties** tab — customizes rollover properties
- ▶ **Workspace** tab — customizes the look of workspaces
- ▶ **Aliases** tab — customizes command abbreviations
- ▶ **Shell Commands** tab — customizes commands that run programs external to BricsCAD

When you choose a tab, the Customize dialog box displays the associated customizable content, as described fully in the following chapters.

SHORTCUT MENU

When you right-click different areas of the Customize dialog box, a number of shortcut menus become available. Different ones appear, depending on where you right-click in the dialog box. Some of these are illustrated by the figure below.



Shortcut menu that appear when different elements are right-clicked

Most of the options are self-explanatory, but there are two whose subtlety can get lost on me. These are *Insert* and *Append*. The difference between them is as follows:

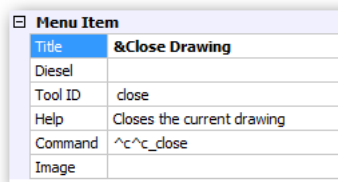
Append — adds the new item at the end of the list

Insert — places the new item before the selected item

The process of customizing toolbars and menus is an identical process; the only difference is that menus have a few more options, such as check marks and gray text.

APPLY AND OK BUTTONS

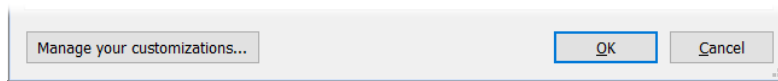
When you make a change in the Customize dialog box, BricsCAD highlights it, turning the change to **boldfaced** text, such as for the Title field shown below. This is handy as it reminds you what has changed.



Menu Item	
Title	&Close Drawing
Diesel	
Tool ID	close
Help	Closes the current drawing
Command	^c^c_close
Image	

Changed parameters displaying boldface text

The boldfacing remains visible, however, only until the dialog box closes; the next time you open it, the text again looks normal. You commit changes to the customization by clicking the **OK** button:



OK and Cancel buttons

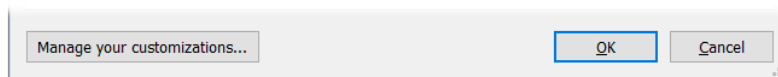
Here's what the buttons mean:

- ▶ **OK** — applies the changes, and then exits the dialog box
- ▶ **Cancel** — reverses (undoes) the changes, and then exits the dialog box

After you click **OK**, BricsCAD applies the changes to a copy of the *default.cui* file. This way, the original *default.cui* file is kept untouched.

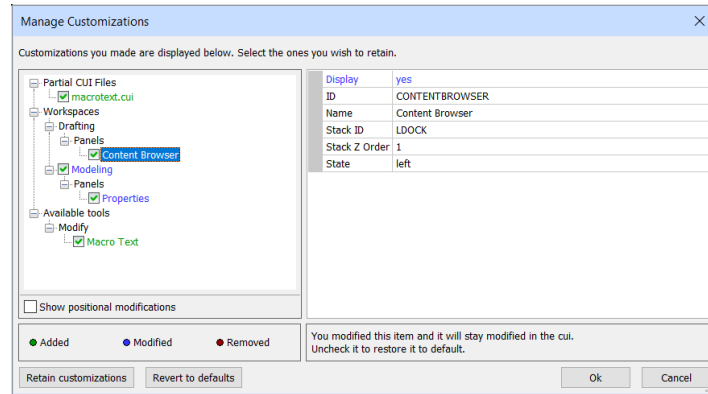
VIEWING CHANGES MADE TO CUSTOMIZE

You can see a list of the changes made to the Customize dialog box, just as with the Settings dialog box. To see them, click the **Manage Your Customizations** button:



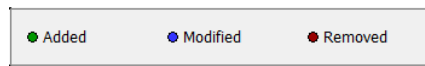
Accessing the changes to customizations

The Manage Customizations dialog box might initially look intimidating.



Managing customizations

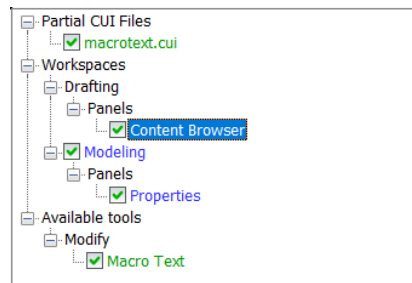
The key understanding it is to use the color coding to figure out what is going on. Near the bottom of the dialog box is a list of the colors and their meanings:



Color coding of changes to customizations

Color	Meaning
Green	Element was added to the default customization set up
Blue	Element was changed
Red	Element was removed

So when you look at the left pane, you see in green the list of elements that changed, modified in blue, and removed in red, if any.



List of changes by category

Here we see that the **macrotext.cui** and **Macro Text** elements were added, because they are shown in green, while **Content Browser** and **Properties** were changed (blue).

The next step is to examine what changes were made to the blue elements. To do so, follow these steps:

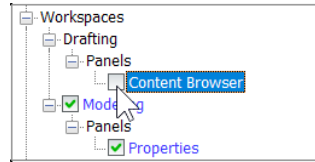
1. Select an element that is shown in green, such as Content Browser.
2. Cast a glance at the right-hand pane, which lists the properties of the selected element.

Display	yes
ID	CONTENTBROWSER
Name	Content Browser
Stack ID	LDOCK
Stack Z Order	1
State	left

Changes to an element shown in blue

3. The properties that changed are shown in blue. In this case, the **Display** parameter was changed to “Yes.” The right-hand panel is view-only; you cannot change anything here, other than to accept or reject changes. To make a change, you have to go back to the Customizations dialog box.

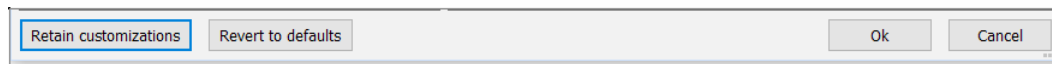
- To accept the change, do nothing
- To reject the change, uncheck the element:



Check boxes accepting and rejecting changes to customization

Additional Management Options

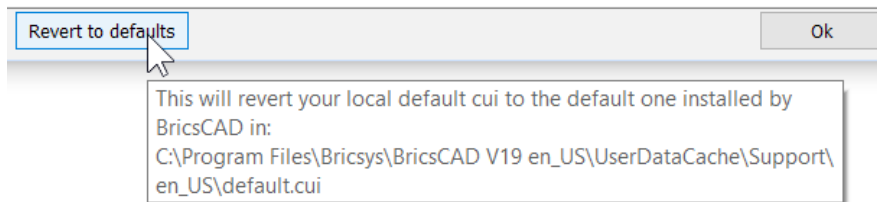
The two options in the lower-left of the dialog box offer these possibilities:



Buttons at the bottom of the dialog box

Retain Customizations button is a shortcut to turn on all the check boxes next to elements. Should you have turned off any of them, then clicking this button checking the boxes next to Content Browser, Properties, and so on

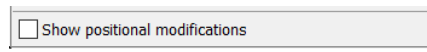
Revert to Defaults button is a shortcut to turn off all checkboxes. BricsCAD pops up a tooltip to ask if you are sure, because it will copy the untouched *default.cui* file over the modified ones.



Erasing all your changes, and reverting to the original CUI content

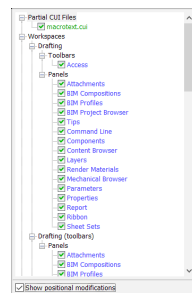
Notice that the check boxes are cleared next to.

Show Positional Modifications check box toggles the display of elements that have only changed their position in the Customize dialog box, a trivial change that otherwise clutters the content of this dialog box.



Toggling the display of position changes

When you turn it on, the dialog box looks like this:



All changes displayed, all of them

Click **OK** to exit the dialog box.

Using Partial Menus to Customize BricsCAD Correctly

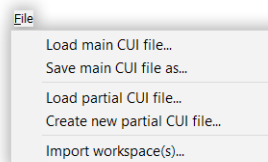
The following seven chapters show you how to change the contents of the Customize dialog box to modify the user interface. It is best, however, to make changes to a partial customization file, rather than the main “default.cui” one for a couple of good reasons:

- The primary *default.cui* file remains unchanged
- Your customization can be shared with other BricsCAD users

SETTING UP A NEW PARTIAL MENU

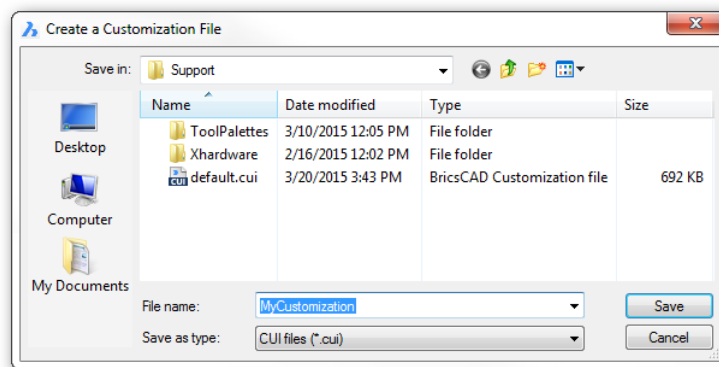
Before carrying out changes in the Customization dialog box, create first a new partial customization file in which you carry out your work. In the Customize dialog box, take the following steps:

1. From the **File** menu, choose **Create New Partial CUI File**.



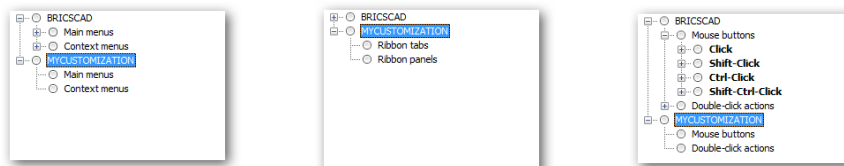
Starting to create a new partial customization file

2. Notice the Create a Customization File dialog box. Enter a name for the new .cui file, such as “MyCustomization,” and then click **Save**.



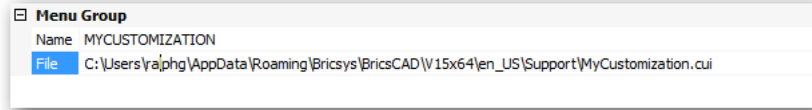
Naming the new CUI file

3. Back in the Customize dialog box, notice the new item called “MYCUSTOMIZATION.” The same item appears in each tab controlled by the .cui file. Below, I show the new item in the Menu, Ribbon, and Mouse tabs.



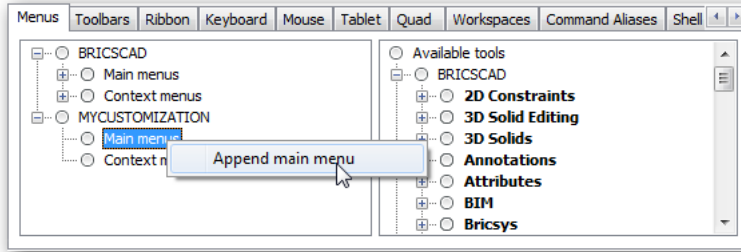
Left to right: MyCustomization item added to Menu, Ribbon, and Mouse tabs

The other thing to notice is that the path to your new .cui file is listed down below in the parameters pane:

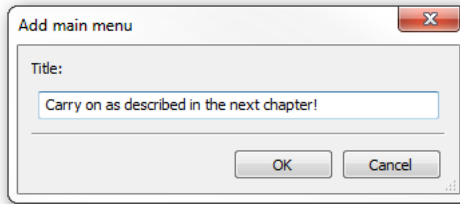


File parameter indicating path to the partial .cui file

4. Now, to actually use the partial CUI file for customization. Here are the steps for writing, for instance, a new menu:
 - a. In the Customization dialog box, click the **Menus** tab.
 - b. Scroll down to the **MyCustomization** section.
 - c. Right-click **Main Menu**.



- d. From the shortcut menu, choose **Append Main Menu**.
 - e. Carry on as described in the next chapter.



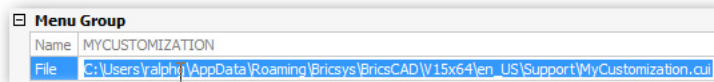
Naming the new menu

Sharing Customizations

By using partial customization files, you easily share customizations you make with others in your office, your clients, and maybe even on the Bricsys eStore at [https://www.bricsys.com/applications/!](https://www.bricsys.com/applications/)

Follow these steps:

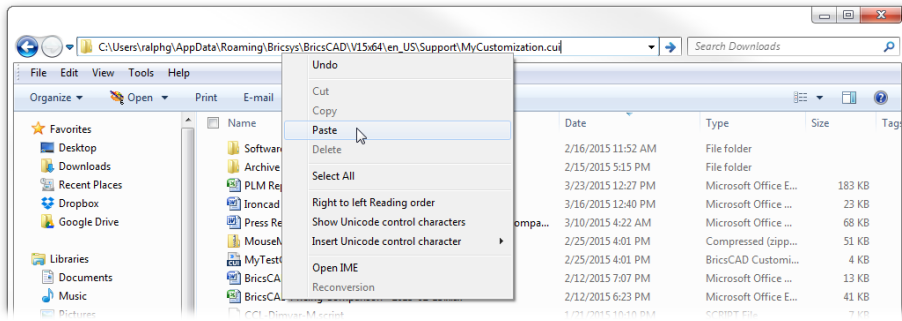
1. Go to the folder that holds your partial customization file. To locate the folder, click on the name of the partial item, then copy and paste the path from the File parameter:
 - a. Select all of the path by dragging the cursor across all of the text.



Selecting the text of the path

- b. Use the **Ctrl+C** (Cmd+C on Macs) shortcut to copy the path to the clipboard. (You can try to right-click, but no shortcut menu will appear.)

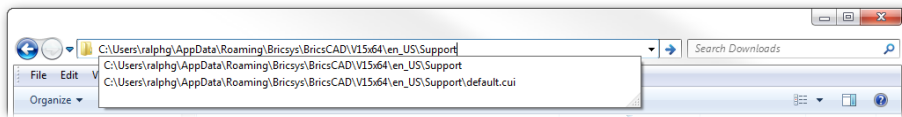
c. In the File Manager, paste the text into the address bar (Windows shown here):



Pasting the path into the address bar

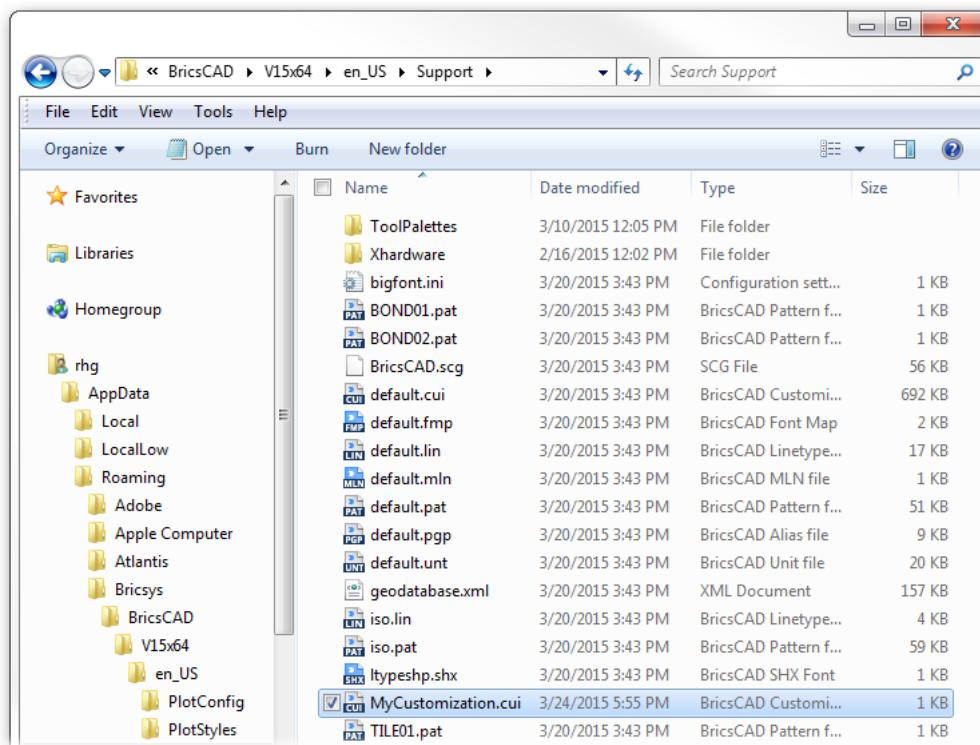
d. At this point, one of two things can happen:

- If you press **Enter**, then the .cui file will be opened by a text editor
 - If you press **Backspace** to erase the file name from the path, then the file manager goes to the folder
- For this tutorial, press Backspace to erase the file name, such as “MyCustomization.cui”.




Erasing the file name from the path

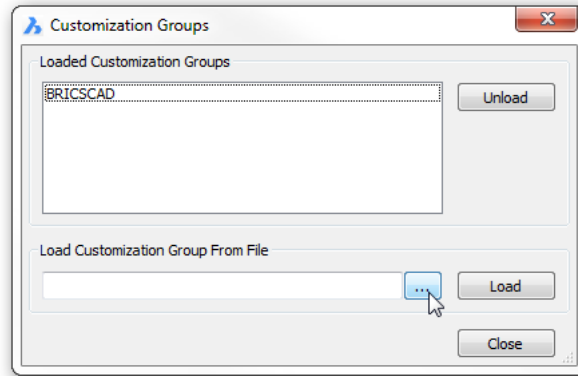
e. Now press **Enter**. Notice that the file manager displays the contents of the folder.



Files displayed in the support folder

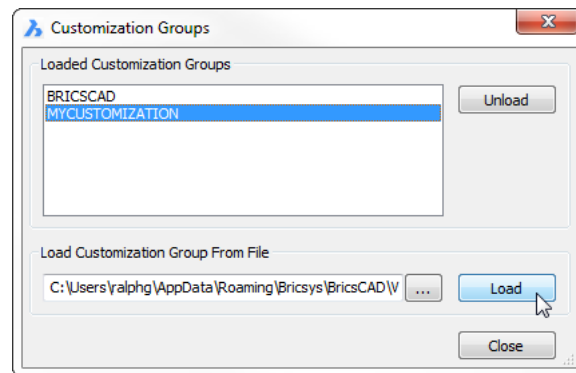
2. You can now copy the partial .cui file to a USB drive or to a central file server like Dropbox or attach it to an email message or...

3. To load a partial customization file, you can use the Customize dialog box's **File | Load Partial CUI File** option, or you can use the **MenuLoad** command. The dialog box is not exactly clearly laid out, so here are the steps to follow:
 - a. Enter the **MenuLoad** command.
 - b. Notice the Customization Group dialog box. Click the  **Browse** button.



Clicking the Browse button in the Customization Groups dialog box

- c. In the Choose a Customization File dialog box, navigate to the folder, drive, or network location that holds the .cui file you want.
- d. Click **Open**.
- e. Back in the Customization Groups dialog box, click **Load**.



Loading the partial customization file into BricsCAD

- f. Notice that the partial customization is added to the list of groups loaded into BricsCAD. Click **Close** to exit the dialog box.

The added customization should now appear in the BricsCAD user interface. If it contains menus, then the new menus will appear at the end of the menu bar. If ribbon tabs, then at the end of the ribbon. And so on.

Removing Partial CUI Files

You use this same dialog box to unload partial customizations that you no longer want in the CAD program. To do so, start the **MenuLoad** command, choose the Customization Group (such as “My Preferences”), and then click **Unload**.

Customizing the Menu Bar & Context Menus

Menus arrange commands in logical groups. The menu uses words primarily, with pictures as an afterthought. The logical arrangement and use of words makes it easy to find specific commands, more so than any other interface, especially for new users and for commands that we rarely use.

In this chapter, you learn how to modify the menu bar's menus and of context menus. You make changes to menus via the Customize dialog box.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Modifying menu items
- Adding new menu items
- Adding new tools (commands)
- Creating context menus
- Sharing menus
- Importing menus from AutoCAD

MODIFYING THE MENU BAR

BricsCAD lists nearly all of its commands on the menu, organizing them by categories. For instance, the **Draw** menu is where you find most drawing commands; most file commands in the **File** menu.

Sometimes, however, you may want to change the content of menus or add a menu — something that is common for third-party developers particularly.

In addition, you may want to add and remove parts of menus, without affecting the original menu structure. These parts of menus are known as *partial menus*. For example, Bricsys adds “Parametric” as a partial menu to the Platinum edition of the software. It appears as another word on the menu bar.

Technical Note The items shown in the Customize dialog box reflect the contents of *default.cui* and other customization files. As you make changes in the dialog box, BricsCAD records the changes in the appropriate *.cui* or *.pgp* file, and then adjusts the looks and actions in the user interface of BricsCAD.

QUICK SUMMARY OF MENU COMMANDS & VARIABLES

The following commands work with menus:

Menu — loads menu files into the program; supports the following file formats:

Format	Meaning
CUI	Standard menu file used by AutoCAD since release 2007, and BricsCAD since V8
CUIX	Packaged menu files used by AutoCAD since release 2010
MNU	Legacy menu files used by AutoCAD and AutoCAD LT prior to release 2008
MNS	LISP source code used by MNU files
ICM	IntelliCAD menu file used by BricsCAD V7 and earlier

MenuLoad — loads menu groups

MenuUnload — unloads menu groups

The following variables work with menus:

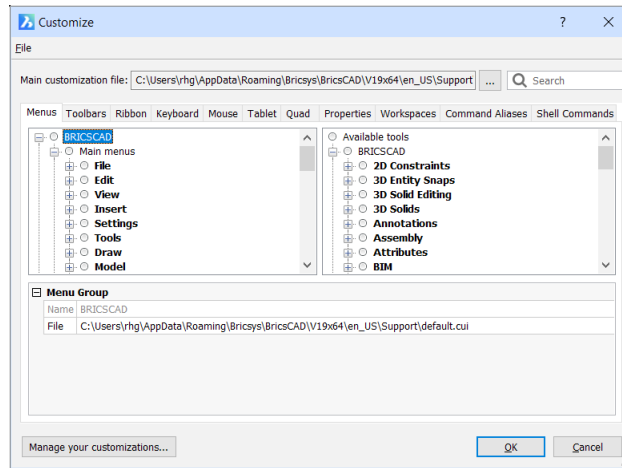
MenuBar — toggles the display of the menu bar

MenuName — reports the path and name of the current menu file

Touring the Menu Tab

Menu customization takes place in the Customize dialog box. Enter the **Customize** command or **Cui** alias at the ‘:’ command prompt, or else right-click any toolbar and then select **Customize**.

Notice the Customize dialog box. If necessary, click the **Menus** tab.

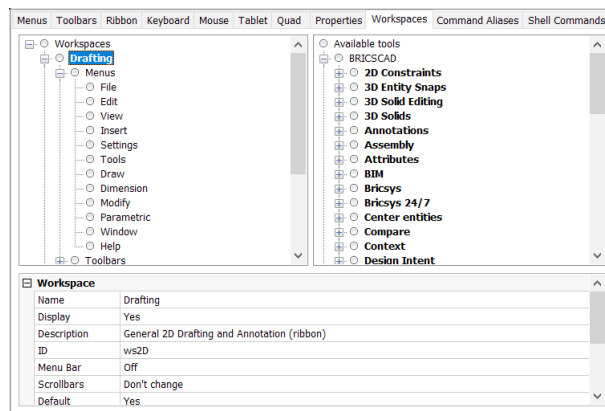


Customize dialog box showing the Menus tab

On the left side you see the Main Menu pane. The names, such as **File**, **Edit**, and through to **Help**, represent the default menus available in BricsCAD. You can change nearly all of them, naturally.

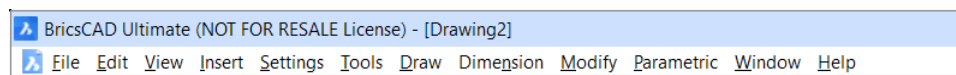
To see the menus actually displayed by BricsCAD at this moment, however, you need to switch to the Workspaces tab, because the purpose of the Menus tab is to *define* menus; the Workspaces tab determines which menus are seen on the menu bar:

1. Click the **Workspaces** tab.
2. Under the **Drafting** node, open the **Menu** node, and there is the list of active menu items.



Workspaces tab showing the menus of the Drafting workspace

It matches what you see on the menu bar, in the same order. Shown here is the menu bar from the “Drafting” workspace. The menu bar will probably change when you switch to another workspace.

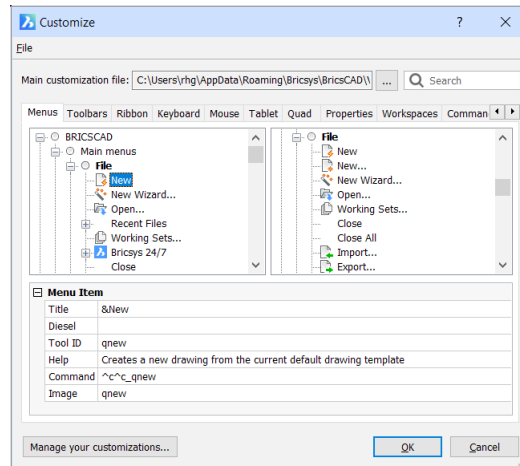


Menu bar for Drafting workspace

QUICK SUMMARY OF MENU PARAMETERS

The *look* of every menu item is defined by parameters found in the Customize dialog box's **Menu** tab. The *position* of menus, submenus, menu items, and separators is defined by their position in the dialog box.

Here is the meaning of the parameters:



Title — label that appears in the menu. The text is displayed literally, but other characters and metacharacters can be employed:

- ... (ellipsis) means the command opens a dialog box
- & (metacharacter) underlines the character following; used for keyboard shortcuts in conjunction with the **Alt** key

Diesel — code written in Diesel programming code

Tool ID — identifier assigned to the menu item by BricsCAD; do not adjust this ID

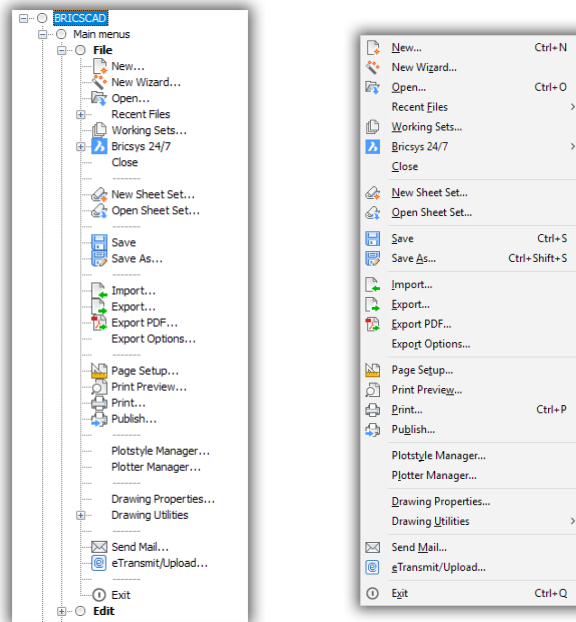
Help — sentence of text displayed on the status bar when you pause the cursor over the menu item

Command — macro to be executed when you click the button; the macro can consist of command names or aliases, option words, Diesel, and LISP code

Image — name of the bitmap (a.k.a. picture or icon) displayed to the left of the menu item; the image is changed by clicking the ... button that appears when this parameter is selected.


Opening and Closing Nodes

Notice that each menu title has a  next to it. For instance, click  next to **File** to reveal the items in the File dropdown menu. The items under File match the names you see in the File menu, as illustrated below.



Left: File menu tree displayed by the Customize dialog box; right: Identical menu items in the File menu.

Gray Dots and Separator Lines

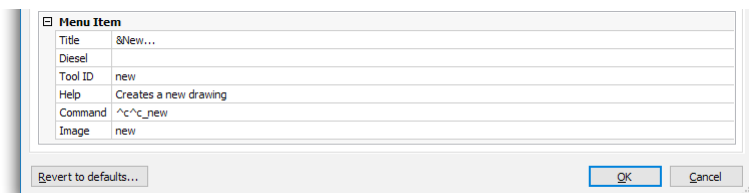
Notice that there are  gray dots that prefix items in the dialog box. These dots indicate “container” items, menu items which literally *contain* other items. Here are examples:

- BRICSCAD** is the name of the *menu group*. It contains *Main Menu* (the items seen on the menu bar) and *Context Menu* (the shortcut menus that appear when you right-click objects.)
- Main Menu** contains the items that appear on the menu bar.
- File** is the name of the first menu to appear along the menu bar, and it contains file-related commands.

The rows of dashes “----” indicates a separator bar, the gray line that separates groups of menu items. See figure above.

UNDERSTANDING MENU TITLE CONVENTIONS

Menu names employ special characters to define conventions. To see what they mean, choose the **New** command and then look at the bottom part of the dialog box — the **Menu Item** pane.



Menu Item pane showing parameter fields

The **Title** field contains the word “New” along with several characters, which I highlighted below in **blue boldface**:

&New...

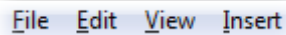
Let’s take a look at the meaning.

Keyboard Shortcut - &

The ampersand (&) is placed in front of the *keyboard shortcut letter* — *N* in this case (&N). This causes the letter N to be shown underlined in the menu when you press the **Alt** key.

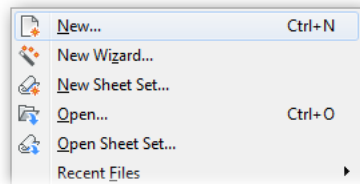
Keyboard shortcut letters allow you to access the menu without a mouse, just from the keyboard. To do so, you hold down the **Alt** key and then choose the underlined letters in the menus. For example, to access the New command in the menu, follow these steps:

1. Press the **Alt** key. Notice each menu name on the menu bar has one letter underlined, such as File.



Underlined letters on menu bar

2. To access the File menu, press **F** on the keyboard (for **F**ile). Notice now that items in the menu also have underlined names, such as New and New Wizard.



Underlined letters in File menu

3. To access the New command, press **N** on the keyboard (for **N**ew).

The convention is that the first letter should be underlined for mnemonic purposes. For example, **N**ew, **O**pen, and **S**ave each have the first letter underlined.

When two names in a menu start with the same letter, however, then the second name has to have a different letter underlined. For example, **N**ew has N underlined, and so **N**ew **W**izard is given Z.

Dialog Box - ...

The ellipsis (...) indicates that the command opens a dialog box. Note that **New...** displays a dialog box, whereas **Save** does not. By itself, the ellipsis does nothing; it is merely a user interface convention. This means that it’s your job to add the ellipsis when you know a command will open that dialog box.

Menu Titles

The name of a menu item can be the same as the command it operates — or it can be different. In most cases, the menu title should be the same as the command it carries out. For instance, selecting the **New...** item causes BricsCAD to execute the New command.

When the command name is somewhat cryptic, however, then it makes sense to switch to a descriptive title, such as using “Polyline” for the PLine command.

COMMANDS USE MACROS

Macros are the programming code behind menu picks. As I noted above, choosing File | New executes the New command. In the Customize dialog box’s Menu Item pane, the command is shown as `^^c_new` in the Command field.



Macro being shown in the Command field

TIPS You can type commands, options, and metacharacters directly into the **Command** text box. As an alternative, you can select commands from the **Available Tools** pane, and then click **Insert Tool**. The advantage to this alternative approach is that BricsCAD automatically adds the `^C` and `_` metacharacters for you.

Like titles, macros make use of metacharacters. The macro syntax has the following meaning:

Cancel - `^c`

The `^c` metacharacter means “cancel.” The caret (`^`) is the equivalent of the **Ctrl** key; together with `C`, `^c` is the same as pressing the **Esc** key to cancel a command. The convention is to start (almost) every macro with two `^c` so that nested commands are cancelled.

Transparent - `'`

You do not prefix macros with `^c` if the command is to be operated *transparently*, such as `'_Redraw`. The apostrophe metacharacter (`'`) means the command can be used during another command. Not all BricsCAD commands are transparent.

Internationalize - `_`

The underscore (`_`) “internationalizes” the command. BricsCAD is available in a variety of (human) languages. By prefixing commands with the underscore, the command word is understood, even if it is used by the Spanish or German releases of BricsCAD.

The **PromptOptionTranslateKeywords** variable toggles the use of international commands. When off, the underscore (`_`) prefix is not needed for command input; default = on.

Enter - `;`

The semicolon (`;`) is equivalent to pressing the **Enter** key. For example, the macro for the **View | Zoom | Zoom In** menu item looks like this:

```
'_zoom;2x
```

In this macro, the Zoom command accesses its 2x option to zoom into the drawing. You typically use the semicolon to separate commands from options.

The convention is to *not* include a semicolon at the end of the macro, because BricsCAD automatically adds the **Enter** for you. If, however, the macro needs to end with two or more Enters, then you do need to supply the two or three semi-colons, such as ;; and ;;;

Pause - \

The backslash (\) pauses the macro for user input, so that you can pick a point or select an entity. In the macro below, BricsCAD pauses twice, once for each backslash. (I show commands and options in blue, while pauses for user input are in cyan.)

```
^^c_dimlinear;\_rotated
```

This is how the DimLinear command appears in the command bar:

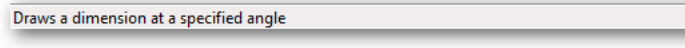
```
: dimlinear
ENTER to select entity/<Origin of first extension line>: (User picks first point.)
Origin of second extension line: (User picks second point.)
Angle/Text/Orientation of dimension line: Horizontal/Vertical/Rotated: _ROTATED
Angle of dimension line <0>: (And so on.)
```

Here is how the macro works:

1. The ^^c sequence cancels any existing command.
2. The DimLinear command begins.
3. The backslash metacharacter forces the macro to wait for input from the user, such as one of these:
 - ▶ The user picks a point on the screen
 - ▶ The user enters a value at the keyboard, and then presses **Enter**
4. The second backslash forces the macro to wait at the 'Origin of second extension line' prompt for the user to react.
5. The macro executes the Rotated option.

EDITING THE HELP STRING

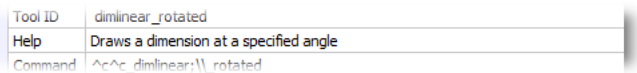
When you change the purpose of a menu item, then you may need to change the text of the help string as well. The help string is displayed on the status bar when the user selects the menu item.



Draws a dimension at a specified angle

Help text displayed on the status bar

You edit the text in the **Help String** textbox.



Tool ID	dimlinear_rotated
Help	Draws a dimension at a specified angle
Command	^^c_dimlinear;_rotated

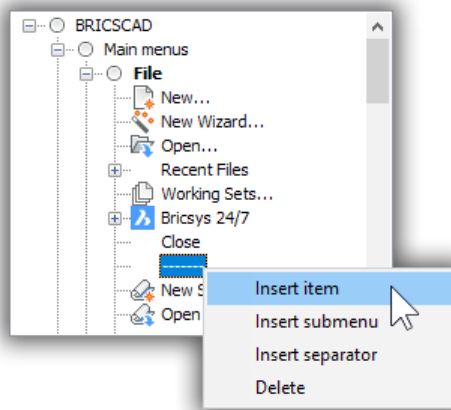
Help text specified in the Customize dialog box

Tutorial: Adding Menu Items

You can add new items to menus by right-clicking an existing menu item in the Customize dialog box, and then choosing an option from the shortcut menu. The shortcut menu allows you to create new menus, add commands and sub-menus to existing menus, and add separator bars.

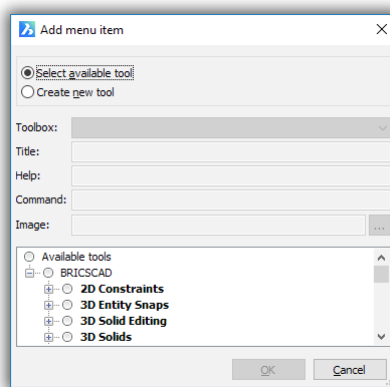
In this tutorial, you add the **CloseAll** command to the File menu; it will be located after the Close item. The CloseAll command closes all open drawings.

1. With the **Cui** alias, open the Customize dialog box, and then choose the **Menus** tab.
2. In the Main Menu pane, open the **File** item.
3. Find the Close item. Below it is a ----- (separator) item.
4. Right-click the separator to place the new item before it. Notice that BricsCAD displays a shortcut menu.



Adding an item to the menu by inserting it

5. To add a the new menu item above the currently-selected one, choose **Insert Item**. Notice the Add Menu Item dialog box.

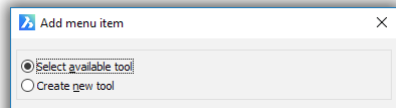


Add New Item dialog box

This dialog box lists *all* commands available in BricsCAD — just like that Available Tools pane. (I'm not sure why there's that duplication.) The dialog box lets you select existing commands and create new ones.

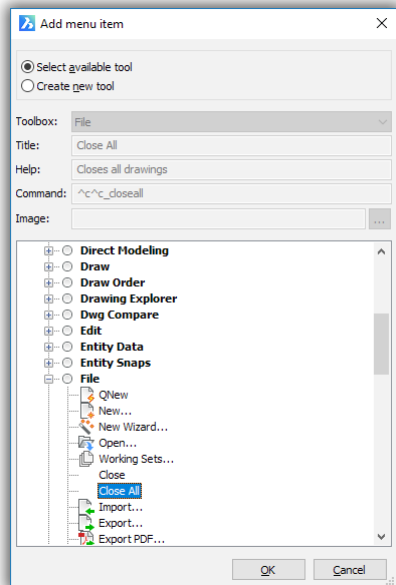
Historical Note Earlier releases of BricsCAD had an **Append Item** option, which added the new item to the end of the menu structure. It didn't make much sense, and was subsequently removed.

- In the dialog box, choose the **Select Available Tool** option. It lets you pick one of BricsCAD's built-in commands. (The other option, Create New Tool, is for creating new commands, and is described later.)



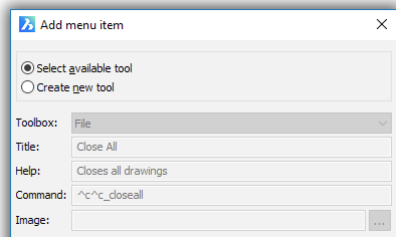
Choosing Select Available Tool option

- Under the list of Available Tools, open the **File** container, and then choose **Close All**.



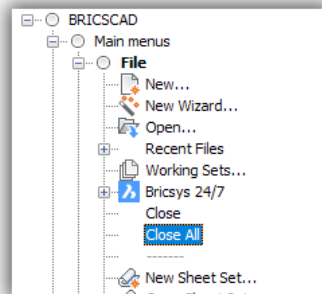
Choosing 'Close All' from the File section

At the top of the dialog box, notice that BricsCAD has filled in most of the parameters for you, such as Title, Help, and so on. They are, however, grayed out; if you wish to edit these values, you need to wait until you are back in the Customize dialog box.



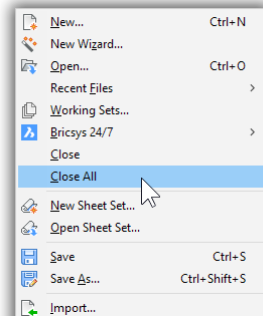
Grayed-out parameters

- Click **OK**. Notice that the **Close All** command is added to the list under Close.



Close All command added to the File menu

9. To ensure the new command works, follow these steps:
 - a. Close the Customize dialog box by clicking **OK**.
 - b. Choose the **File** menu. Notice that the **Close All** item has been added.



Close All command appearing in the File menu

- c. Click **Close All**. Does it work correctly? It should prompt you to save all open drawings that have changed since being loaded.

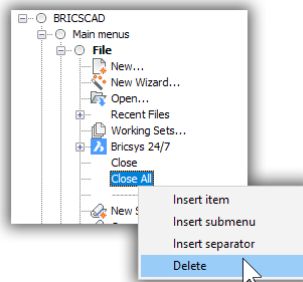
TIP Not sure which commands can be added to menus? Peruse the list in the Customize dialog box found under **Available Tools**. It lists all commands found in BricsCAD, sorted by menu order.

The Available Tools listing also allows you to add your own commands, which can be constructed from other commands or from LISP routines.

TUTORIAL: DELETING MENU ITEMS

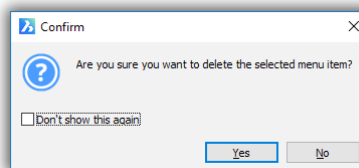
To delete a menu item, select it, and then right-click. You can delete individual menu items, as well as submenus and entire menus.

1. From the shortcut menu, choose **Delete**.



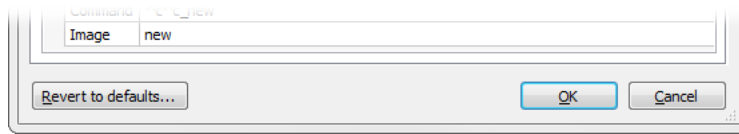
Choosing Delete from the shortcut menu

2. BricsCAD asks if you really want to do this. Click **Yes**.



Confirming the item should be erased

Did you make a horrible mistake? There is no undo button. Instead, click the **Revert to Defaults** button to return the menus to their fresh-out-of-the-box nature. This action, however, also undoes all other changes you made, including those you may want to keep.



Revert to Defaults button in the Customize dialog box

Tutorial: Adding Tools to Menus

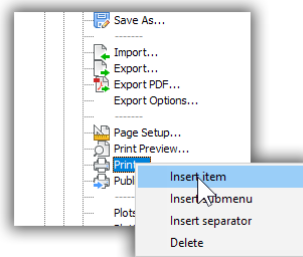
So far, you've seen how to add existing BricsCAD commands to menus and toolbars. You also make "new" commands, which BricsCAD calls *tools*. These are not so much commands as reworkings of existing commands — pieces of simple programming code that simulate commands — and are known as "macros." These are described in detail in the following chapter.

To show you how to create new tools, I'll write a macro that saves the drawing and then starts the Plot command. So, two commands combined into a single menu pick. The macro looks like this:

```
^C^C_qsav;_plot
```

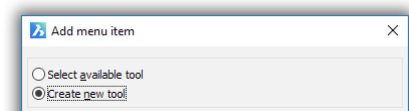
I'll name the macro "Save'n Print" and add it to the File menu, like this:

1. In the Customize dialog box's Menu tab, follow these steps:
 - a. Choose the **File** item.
 - b. Right-click **Print**.



Inserting an item

- c. From the shortcut menu, choose **Insert Item**.
2. Notice the Add New Item dialog box. Select the **Create New Tool** option.

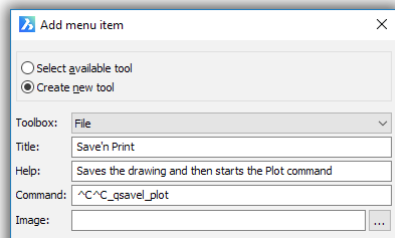


Creating a new tool (aka a new command)

- Fill in parameters, as follows:

Parameter	Entry	Comment
Toolbox	File	Adds the new command to the File category of available tools
Title	Save'n Print	Name that appears in the File menu
Help	Saves the drawing, and then starts the Plot command.	Help text that appears on the status bar
Command	^^^C_qlsave;_plot	Macro that cancels the current command, saves the drawing, and then starts the Plot command
Image	(leave blank)	No images are needed for menus

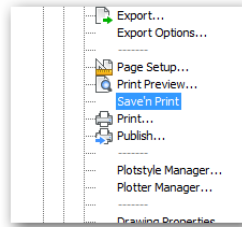
When the parameters are filled in, the dialog box looks like this:



Parameters filled out for the new tool

Ignore the bottom half of the dialog box, the one that lists all commands.

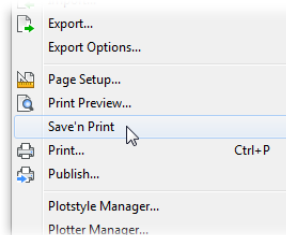
- Click **OK**. Back in the Customize dialog box, notice that the new tool is added to the File menu (on the left) and to the list of Available Tools (on the right).



New tool added to the menu

In addition, the new set of parameters is shown in the Menu Item pane (at the bottom of the dialog box.) You can edit the parameters here, just like with any other command.

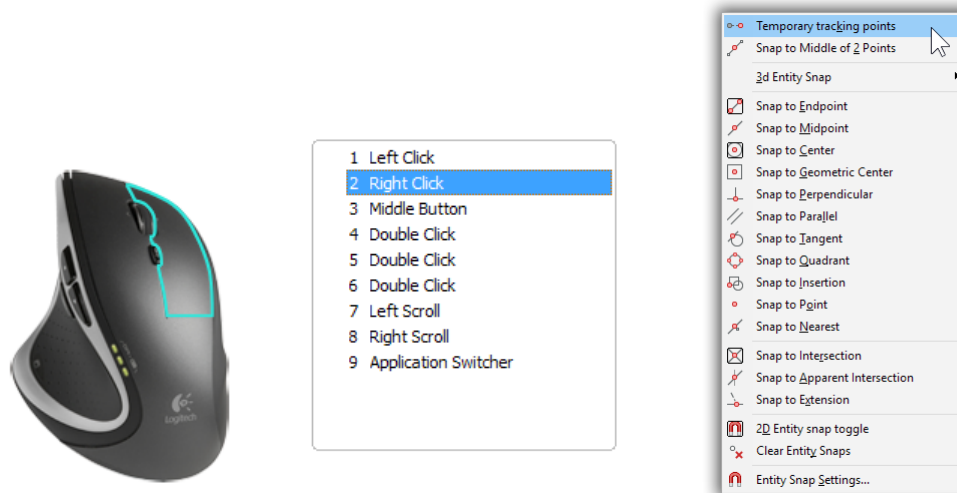
- Click **OK**.
- Test the new item by selecting **Save'n Print** from the **File** menu.



Testing the new tool

Context Menu

Context menus go by a number of names, such as “shortcut menu” or “right-click menu.” Whatever the name, they are the menus that appear when you press the mouse’s **right** button — the action known as “right-clicking.”



Left: Mouse highlighting right button; **right:** Example of a context menu

Different shortcut menus appear depending on *where* you right-click in BricsCAD: while drawing or while editing or in the user interface.

As well, the menu that appears depends on whether you hold down the **Shift** and/or **Ctrl** keys at the same time as right-clicking. The **Menu** tab’s **Context Menu** section defines the menus; the **Buttons** tab defines *some* other right-click actions. (See Chapter 10).

You can customize the content of some context menus, though not all of them. Specifically

- ▶ You can customize those that appear when you right-click inside the drawing area
- ▶ You cannot change the menus that appear when you right-click *outside* of the drawing area, such as on the status bar or a toolbar

TIP If shortcut menus do not appear when you right-click the mouse, then you need to turn on several related options in the Settings dialog box, like this:

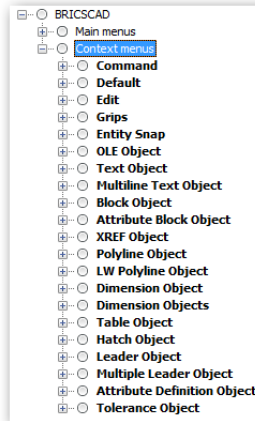
- Enter the **Settings** command.
- In the Search field, enter “shortcut menus.”
- Turn on all options, as illustrated below.

Shortcut menu	0x001F (31)
1	<input checked="" type="checkbox"/> Enable Default mode shortcut menus
2	<input checked="" type="checkbox"/> Enable Edit mode shortcut menus
4	<input checked="" type="checkbox"/> Enable Command mode shortcut menus (available whenever a command is active)
8	<input checked="" type="checkbox"/> Enable Command mode shortcut menus only when command options are currently available from the command line

TUTORIAL: CUSTOMIZING CONTEXT MENUS

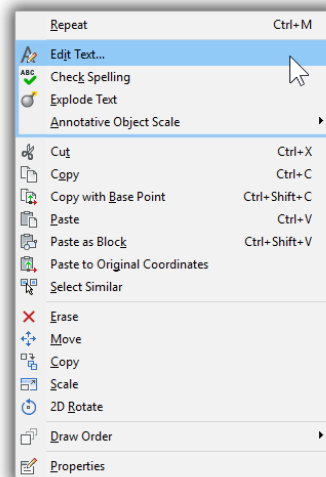
Context menus are customized in the much same way as regular menus.

1. Open the Customize dialog box (**Tools | Customize**), and then choose the **Menus** tab.
2. In the left hand pane, scroll down until you reach **Context Menus**.
3. Click the **+** to open the **Context Menus** tree. Notice the menus represented there, as illustrated below.



Accessing the list of context menus in the Customize dialog box

Below is the context menu that is displayed when you select a text object, and then right-click. Commands specific to text editing are added by BricsCAD, which I have emphasized in blue.



Context menu when a text object is selected

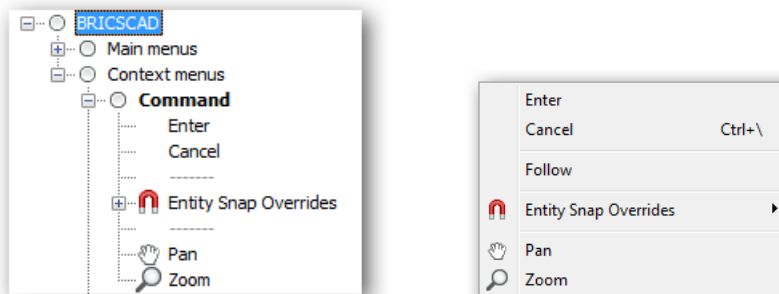
There are two types of context menus, *full* and *partial*:

Full menu — replace existing context menus, and all other items defined by the Customize dialog box.

Partial menu — add items to context menus, such as editing text, polylines, dimensions, and attributes.

Context Menu Name	Menu Appears While Right-clicking When...
<i>Full Context Menus</i>	
Command	A command is active
Default	No command is active
Edit	Object is selected, other than those listed below
Grips	A hot (red) grip is active
Entity Snap	The Shift key is held down
<i>Partial Context Menus</i>	
OLE Object	OLE object is selected
Text Object	Text object is selected
Multiline Text Object	Multi-line text object is selected
Block Object	Block is selected (formerly named "Insert")
Attribute Block Object	Attributed is selected (formerly "Attribute Block Reference")
XREF Object	Externally-referenced drawing is selected
Polyline Object	Old-style polyline is selected
LW Polyline Object	New-style lightweight polyline is selected
Dimension Object	Dimension is selected
Dimension Objects	One or more dimensions are selected
Table Object	Table is selected
Hatch Object	Hatch pattern is selected
Multiple Leader Object	Multi-line leader is selected
Attribution Definition Object	Attribute definition is selected
Tolerance Object	Tolerance is selected

The structure of the context menu's definition is similar to that of regular menus. For instance, items are listed in the order in which they appear, and there are submenus and separator lines.

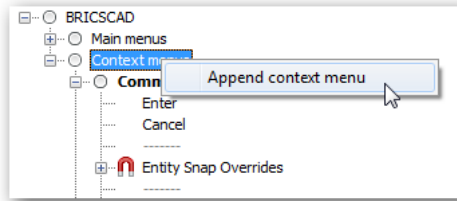


Left: Context menus in Customize dialog box; **right:** Context menu appears when right-clicking during an active command

- Adding commands is no different than before. Follow the previous two tutorials on adding existing commands and new tools to context menus.
- There is one thing different in creating new context menus: they are activated only when you right-click something specific. BricsCAD has a list of these specific actions, also known as "reactors."

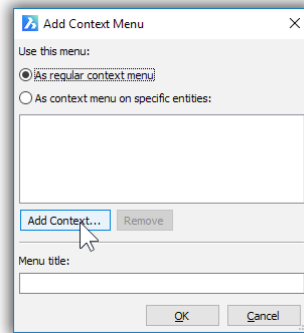
Right-click **Context Menus**.

6. From the shortcut menu that appears, choose **Append Context Menu**.



Appending a context menu

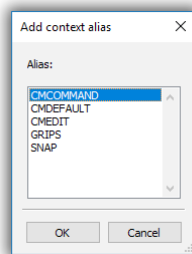
Here it gets complicated I find, for Bricsys redesigned the Add Context Menu dialog box to split options into two streams.



Add Context Menu dialog box

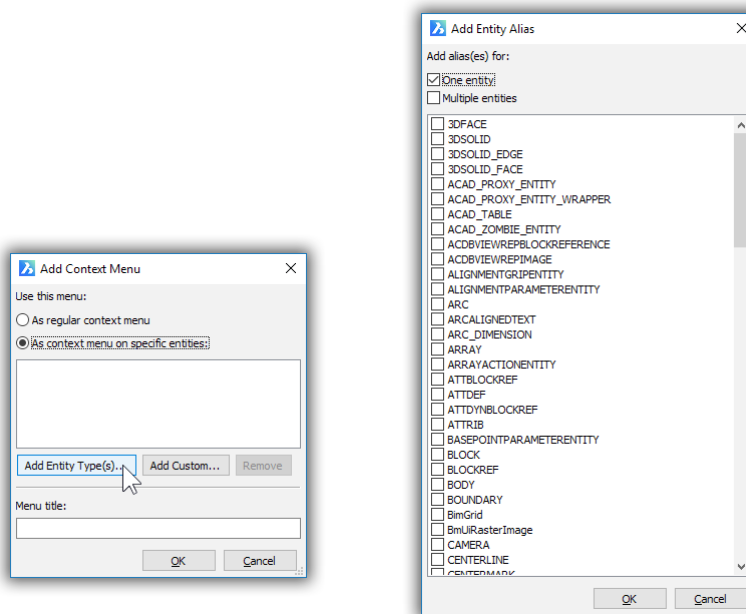
The **Use This Menu** options are as follows:

- ▶ **As Regular Context Menu** — this option accesses context menus that don't involve entities, such as during commands or right-clicked grips. There are just five of them. To choose one of them, click the **Add Context** button. The Add Context Alias dialog box appears, as shown below.



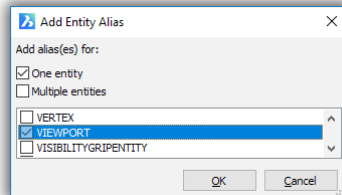
Selection of non-entity shortcut menus

- ▶ **As Context Menu on Specific Entities** — this option accesses context menus for entities, such as text and polylines. Click the **Add Entity Type(s)** button to select one from the Add Entity Alias dialog box.



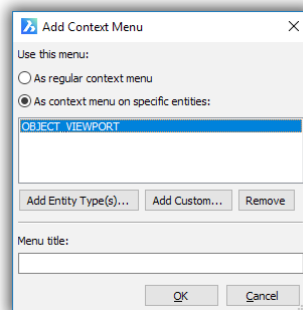
Selection of shortcut menus specific to entities

7. For this tutorial, you create add a shortcut menu that involves viewport objects. Follow these steps:
 - a. Choose **As Context Menu on Specific Entities**.
 - b. Click **Entity Type(s)**. Notice the Add Entity Alias dialog box.
 - c. Scroll down the list, and then choose **Viewport**. To get there quickly, press 'v'.



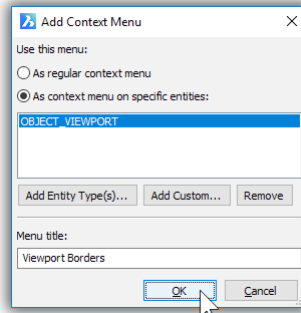
Selecting the Viewport as the entity

- d. Click **OK** to close the dialog box. Notice that “Object_Viewport” is added to the list of entities.



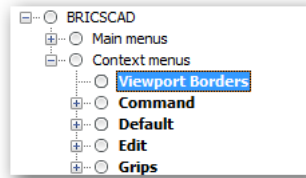
Viewport added as the specific entity

- e. Add text to the Menu Title, such as “Viewport Borders”...



Naming the menu

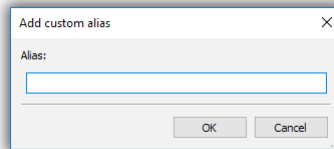
... and then click **OK**. Notice that BricsCAD adds the new menu to the list. It is, however, empty of commands.



Viewport Borders item with no commands

8. Your job now is to populate the context menu with commands. Follow the instructions given earlier for regular menus. When done, click **OK**, and then test the new shortcut (context) menu.

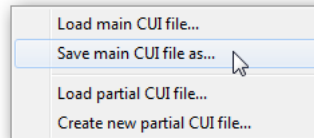
TIP The Custom Alias option is meant for third-party developers who create custom objects:



Tutorial: Sharing Menus

To share customized menus with other BricsCAD users, follow these steps:

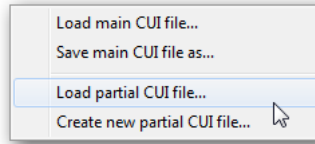
1. Open the Customize dialog box (**Customize**), and then from the **File** menu, choose **Save Main CUI File As**.



Saving the menu as a CUI file

2. In the dialog box, enter a file name, and then click **Save**. BricsCAD saves the menu structure as a *.cui* file. This action saves menus, toolbars, and so on in the same *.cui* file. It does not save aliases or shell commands, because they are stored in a *.pgp* file, which you cannot access from the Customize dialog box.

3. Copy the .cui file to the other computers via your network, email, or a USB thumbdrive.
4. On the other computer, import the .cui file through the Customize dialog box's **File** menu.



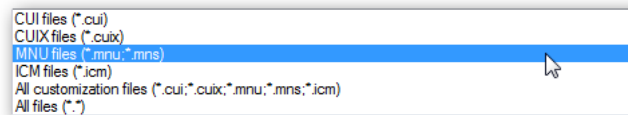
Loading the CUI file as a partial menu

The menu presents you with two options:

- ▶ **Load Main CUI File** — overwrites existing menus, toolbars, and keyboard shortcuts with the new file. (If you did not mean to, use the Revert to Defaults button to correct the mistake.)
- ▶ **Load Partial CUI File** — adds the contents of the file to the existing menus, toolbars, and so on.

IMPORTING AUTOCAD MENUS

The Customize dialog box's **File | Load** items import three kinds of menu files into BricsCAD. Choose them from the droplist in the Choose a Customization File dialog box:



CUI — standard menu file used by AutoCAD since release 2007, and BricsCAD since V8.

CUIX — packaged menu files used by AutoCAD since release 2010

MNU or **MNS** — legacy menu files used by AutoCAD and AutoCAD LT prior to release 2008.

ICM — IntelliCAD menu file used by BricsCAD V7 and earlier, as well as CAD systems based on IntelliCAD.

Careful: Although BricsCAD imports AutoCAD menu files effortlessly, menu picks sometimes do not work because AutoCAD macros can contain macro code and metacharacters unsupported by BricsCAD. For more information on writing macros for menus, see Chapter 8.

Customizing Toolbars and Button Icons

The best way to customize commands in BricsCAD is through toolbars, in my opinion. Toolbars give me single-click access to almost any command or group of commands (a.k.a. *macros*). Instead of hunting through the ribbon (is hatching in the *Draw* or *Tools* tab) or trying to recall the exact syntax of a typed command (was that *Viewpoint* or *VPoint*?), toolbars let us collect our most-used commands into one or more convenient strips.

BricsCAD lets you customize these aspects of toolbars:

- › Name, default position, size, and default visibility of toolbars
- › Flyouts and separator bars
- › Titles, macros, help strings, and button images

All of these elements are handled by the Customize dialog box and working with them is explained in this chapter.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Customizing the look of toolbars and buttons
- Creating new toolbars and flyouts
- Understanding controls and separators

Customizing the Look of Toolbars

There are two ways to customize toolbars in BricsCAD:

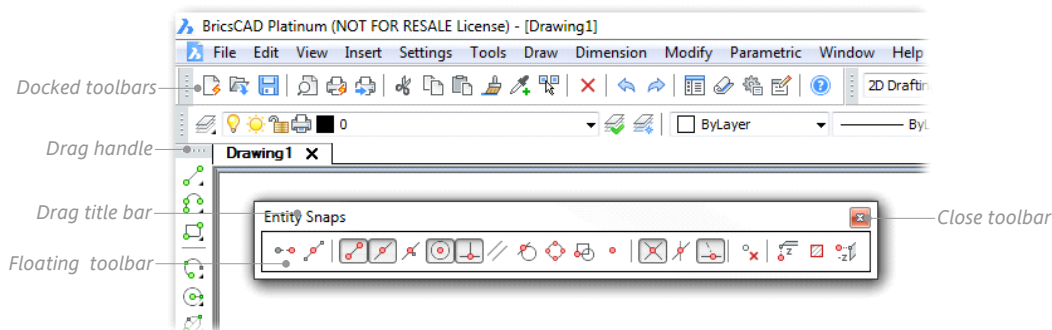
- ▶ Change the *look* of toolbars
- ▶ Change the *macros* executed by toolbar buttons

The first one is simpler, as it involves cosmetic changes, such as rearranging buttons, making new icons, or defining new toolbars. The second way involves programming by (re)writing macros that activate one or more commands when the toolbar button is clicked.

First, here is how to change the looks of toolbars.

REARRANGING TOOLBARS

The first time you start a fresh copy of BricsCAD in the toolbar workspace, you'll see that it has several toolbars *docked* along the edges of the drawing area. "Docked" means that when you move the main BricsCAD window, docked toolbars move along with it.



Docked and floating toolbars

Toolbars don't have to be docked; they can *float*. When toolbars float, they are independent of the BricsCAD window. When you move or resize the BricsCAD window, the floating toolbars remain where they are.

QUICK SUMMARY OF TOOLBAR COMMANDS & VARIABLES

The following command works with toolbars:

Toolbar and **-Toolbar** — displays and hides toolbars by name, at the command line

The following variables work with toolbars:

ToolbarIconSize — changes the size of icons between regular, large, and extra large

MenuName — reports the path and name of the current menu file

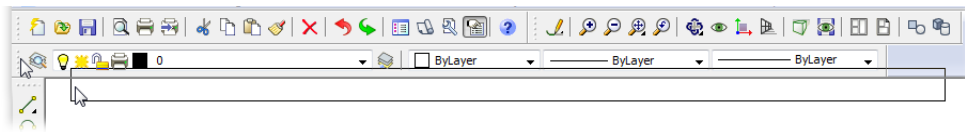
Tutorial: Dragging and Moving Toolbars

Look closely at the left end of a toolbar that is docked. There you see a line of dots, shown in the figure above. This is called the “drag handle.” By dragging the handle, you move the toolbar to another location in the BricsCAD window or make it float.

You move floating toolbars by dragging them by their title bars. Floating toolbars can be relocated to other edges of the drawing area — or left floating on the screen.

Here is how to move toolbars:

1. Position the cursor over the drag handle (line of dots).
2. Hold down the left mouse button, and then move the mouse.
3. Drag the toolbar away from the edge of the drawing area. Notice the thin, gray, rectangular outline. This is called the *dock indicator*, as shown by the figure below.



Moving a docked toolbar

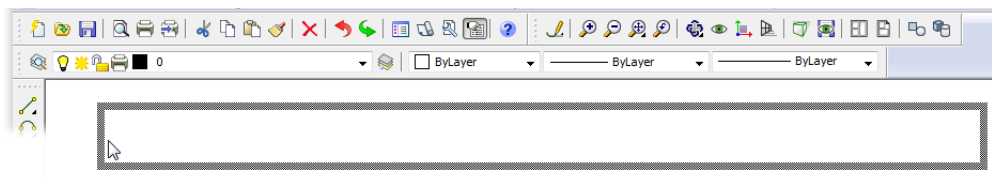
If you were to release the mouse button at this point, the toolbar would jump back to its docked position.

TIP You prevent the toolbar from docking inadvertently by holding down the **Ctrl** key.

To prevent any toolbar or panel (palette) from moving, use the **LockUI** variable. Its options are listed in the Settings dialog box, as follows:

Lock user interface elements	0x0000 (0)
negative	<input type="checkbox"/> Locking temporarily disabled
1	<input type="checkbox"/> Lock docked toolbars
2	<input type="checkbox"/> Lock docked panels
4	<input type="checkbox"/> Lock floating panels and toolbars

4. Drag the toolbar a bit further, and notice that the rectangular outline becomes thicker. This is called the *float indicator*.

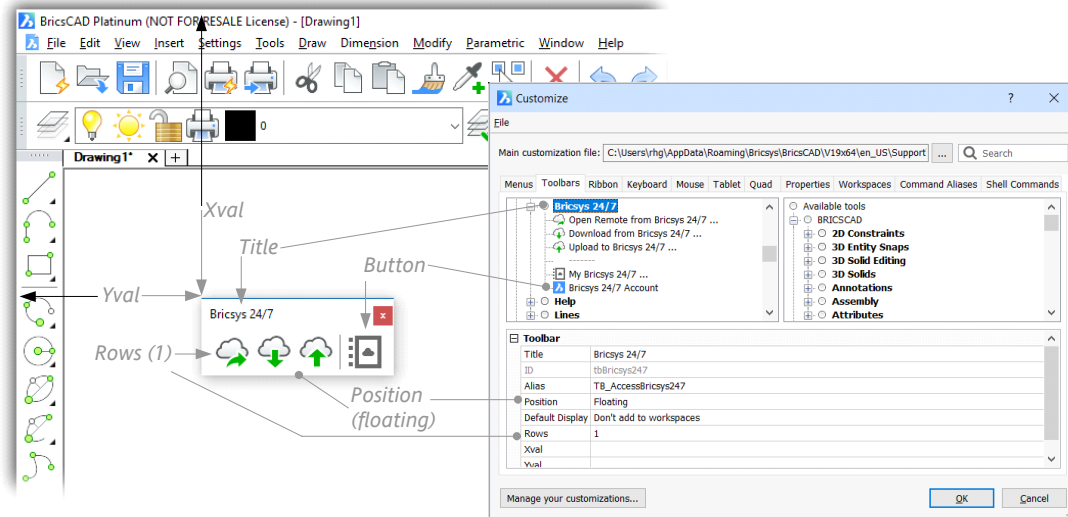


Symbology for a floating toolbar

5. Let go of the mouse button now, and the toolbar floats. With the toolbar floating, you move it around by dragging its title bar.
6. To dock the toolbar again, drag it by its title bar back against one edge of the drawing area, and then let go of the mouse button.

QUICK SUMMARY OF TOOLBAR PARAMETERS

The look and position of every toolbar is defined by an individual set of parameters defined by the Customize dialog box. Here is an overview of the meaning of the parameters.



Title — identifies the toolbar to the BricsCAD system, and appears on the title bar of the toolbar (you can make the title any descriptive phrase that you like)

ID — BricsCAD assigned identification of the UI element; read-only (cannot be edited)

Alias — code name assigned by BricsCAD to identify this toolbar (although you can edit this value, I suggest that this would be an unwise move)

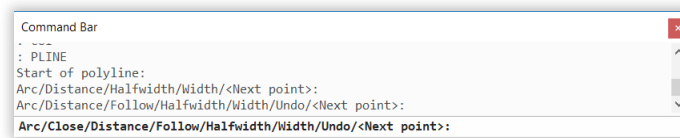
Position — determines whether the toolbar is floating or docked at one of the four edges of the screen; defines the default position when the toolbar is turned on; choose from Floating, Top, Bottom, Left, or Right (this parameter had no effect at the time of writing this ebook)

Default Display (formerly Visible) — determines whether the toolbar is displayed or hidden when BricsCAD starts up; choose from Add to Workspaces and Don't Add to Workspaces

Rows — number of rows of a floating toolbar

Xval and **Yval** — x and y coordinates of a floating toolbar's upper left corner, as measured from BricsCAD's upper left corner

TIP Although not a toolbar, the Command Bar can also be made to float and resize. To float, drag the left edge into the drawing area.



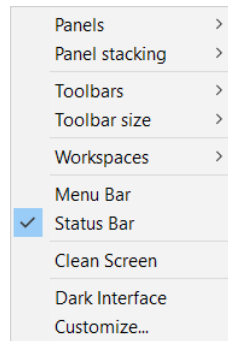
Once floating, you can move the Command Bar window by its title bar, and resize it by its edges — just like a toolbar. To dock it again, drag the Command Bar window back into place.

To turn the Command Bar on and off, press **Ctrl+9** (Cmd+9 on Mac).

Tutorial: Turning Toolbars On and Off

When you want to turn on a toolbar (or turn one off), then follow these steps:

1. Right-click any toolbar or the ribbon. Notice that a shortcut menu appears. (NEW IN V20) This shortcut menu was redesigned in BricsCAD V20.



Accessing UI elements from the shortcut menu

Panels — lists the names of panels

Panel Stacking — options for how multiple panels are displayed

Toolbars — lists the names of toolbars, illustrated at right

Toolbar size — change the size of icons between normal, large, and extra large

Workspace — select the workspace to make current

Menu Bar — toggles the menu bar on and off

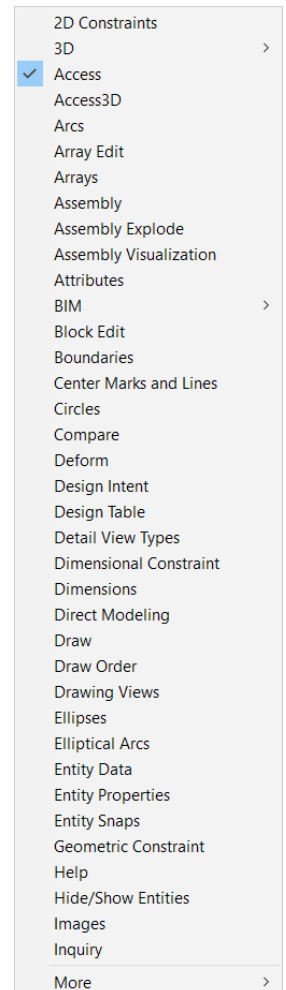
Status bar — toggles the status bar

Clean Screen — toggles the clean screen state

(NEW IN V20) **Dark interface** — toggles the theme between light and dark interface

Customize — opens the Customize dialog box

2. Click **Toolbars** and then click **BRICSAD**. A submenu lists the names of toolbars, as illustrated at right. The check mark means that the toolbar associated with the name is turned on (displayed):
 - ▶ **Turn on a toolbar** select its name from the submenu. Notice that the shortcut menu disappears, and the toolbar appears.
 - ▶ **Turn off a toolbar** click a name *with* a check mark.



TIPS You can turn on (or off) *all* toolbars at once through the **Toolbar** command, as follows:

```

: toolbar
Enter Toolbar name, or <ALL>: all
Enter an option [Show/Hide] <Show>: s

```

This command can also turn on and off individual toolbars, which is useful in macros and LISP routines.

When toolbars are floating, you can turn them off by clicking the red **x** in the upper right corner.

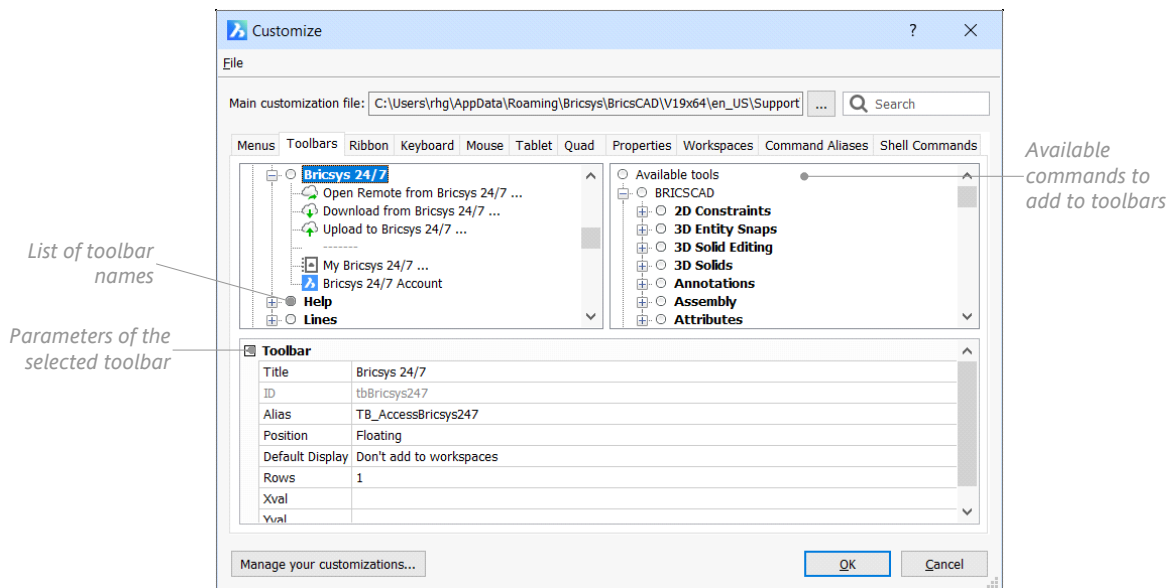
Making New Toolbars, and Modifying Them

You create new toolbars with any set of buttons of your choosing. You can change the content of the toolbar by adding and removing buttons, controls, flyouts, and separator bars — as well as designing your own icons. Let’s see what this means, and how it is done.

TUTORIAL: HOW TO CREATE A NEW TOOLBAR

For this tutorial, you make a toolbar with commands related to grouping, as BricsCAD doesn’t offer such a toolbar. *Groups* are like unnamed blocks, but unlike blocks are easily editable. They are made with the Group command.

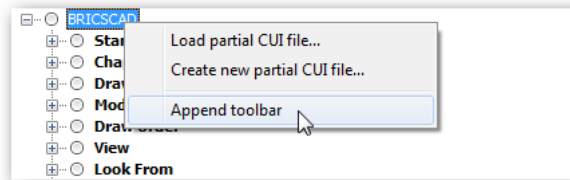
In this tutorial you create a new toolbar named “Group” that holds the Group command.



Three panes of customizing toolbars

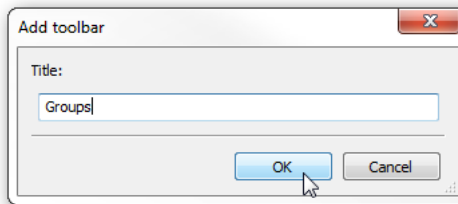
1. Open the Customize dialog box. I find the quickest way to do this is to type the **Cui** alias at the command prompt. Other methods include the following:
 - ▶ Right-click any toolbar, and then from the context menu, select **Customize**
 - ▶ From the Tools menu, select **Customize**
 - ▶ Enter the **Customize** command

2. When the Customize dialog box appears, choose the **Toolbars** tab. Notice the three panes and how they relate to toolbar customization:
 - ▶ **Toolbar** pane — (on the left) lists the names of all toolbars in BricsCAD, sorted by the order in which they were created. Within each toolbar name are names of commands, represented by icons.
 - ▶ **Command** pane — (on the right) lists the names of all commands available in BricsCAD, sorted by the order in which they appear in dropdown menus. For instance, file-related commands are listed under File.
 - ▶ **Parameters** pane — (at the bottom) lists parameters controlling the look and function of toolbars and their buttons. Here you edit the names, command macros, button images, help strings, and optional Diesel code for each button.
3. To create the new toolbar, right-click the **BRISCAD** node, and then choose **Append toolbar** from the short-cut menu.



Appending a new toolbar

4. Notice the Add Toolbar dialog box.

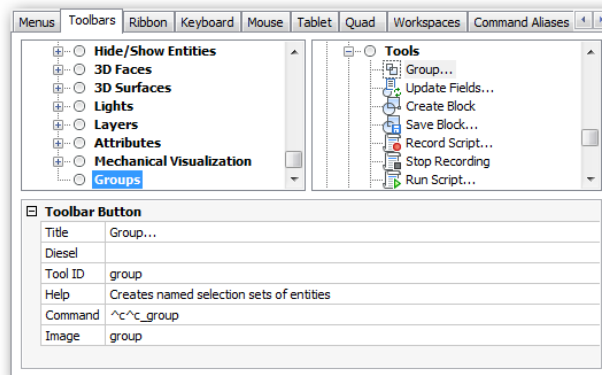


Naming the new toolbar

For this tutorial, follow these steps:

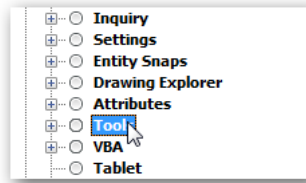
- a. Enter “Groups” for the name
Title: **Groups**
- b. Click **OK**.

The name will appear on the title bar of floating toolbars — as well as identify the toolbar to BricsCAD. Notice that BricsCAD adds the new (but empty) toolbar named Group to the end of the list. Also notice that it fills out the Macro pane (in the lower half of the dialog box) with some preset parameters.



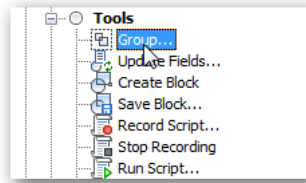
Newly created toolbar named “Groups”

5. With the toolbar created, you now add buttons. The easiest (I find) way is to drag and drop buttons from the Tools pane onto the new Groups node. Follow these steps:
 - a. Because the Groups command is listed under Tools, so scroll down to **Tools**.



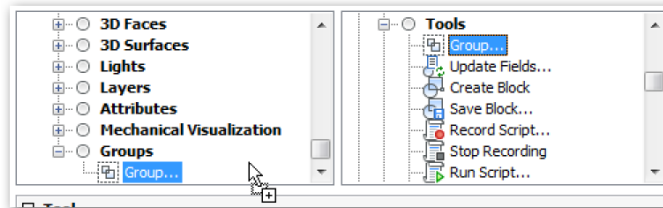
Finding a command in a collection

- b. Open the **Tools** group by clicking the + sign.
 - c. Choose the **Groups...** item.



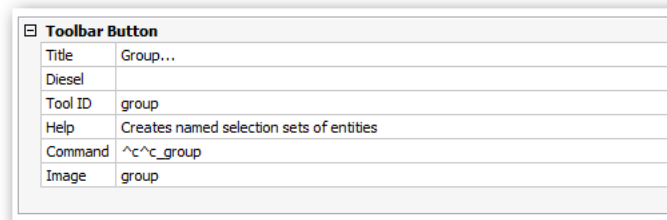
Choosing the Group tool

- d. Drag “Groups...” over to the Toolbars pane, and then deposit on the Groups toolbar.



Dragging the Group tool to the Groups toolbar

Notice that BricsCAD fills the Macro pane with preset parameters.



Preset parameters for the Group tool

6. Click **OK**. Notice the new toolbar appears.



New toolbar with its solitary Group button

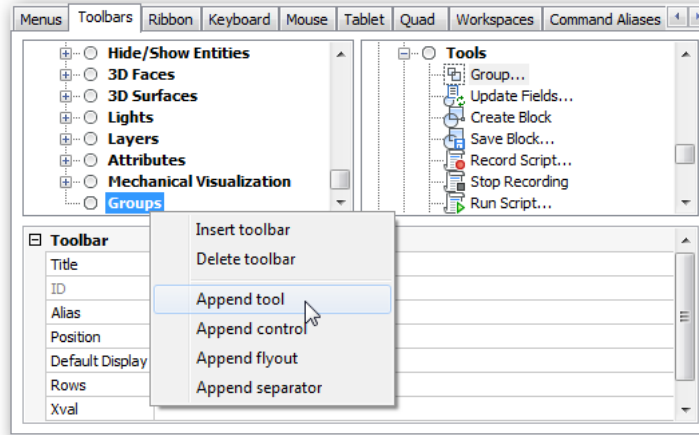
(If you do not see the toolbar, you may need to turn it on by right-clicking another toolbar, and then choosing **BRICSCAD | Groups** from the shortcut menu.)

7. Click the button to ensure it works correctly: the Group command should execute by displaying the Entity Grouping dialog box.

Tutorial: Alternative Method

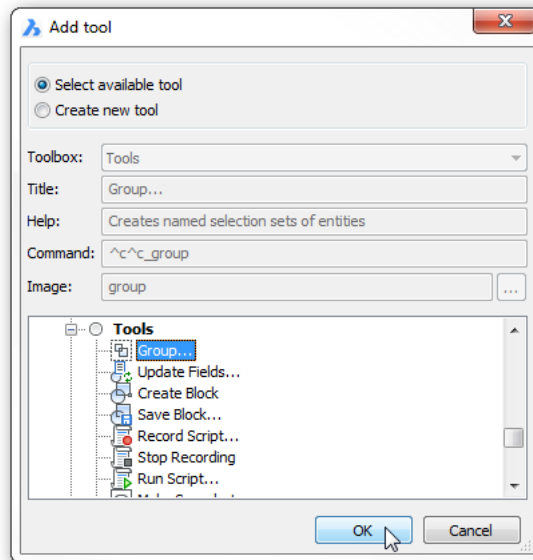
There is a second method for populating toolbars. It involves a dialog box, and is useful if you are not fond of dragging'n dropping. This method replaces Step 5 from above:

- 1 - 4. Follow the steps listed above.
5. With the blank toolbar created, it's time to add a button. Follow these steps:
 - a. In the Toolbars pane, right-click the **Groups** item.
 - b. From the shortcut menu, choose **Append Tool**.



Using the Append Tool alternative

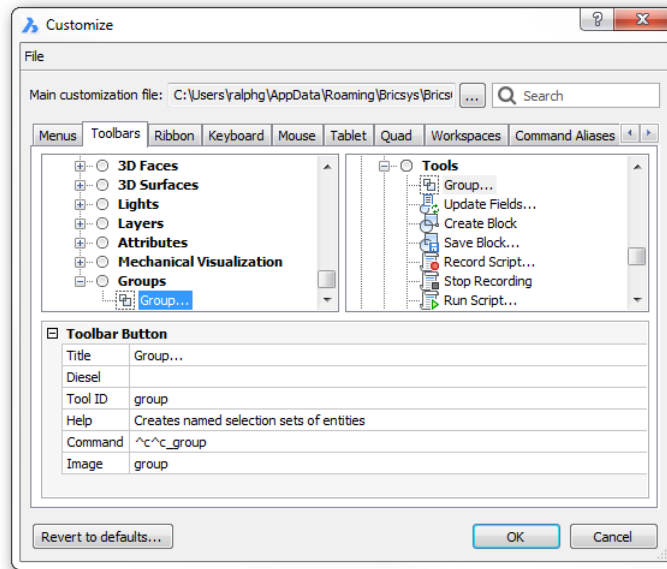
- c. Notice the Add Tool dialog box. Ensure that the **Select Available Tool** option is selected.



Choosing the Group tool

- d. As before, you can find the Groups command under Tools, so scroll down to **Tools**, and then open the Tools group by clicking the + sign.

- e. Choose **Groups**, and then click **OK**. Notice that BricsCAD fills the Macro pane with preset parameters.



Group command added to the Groups toolbar

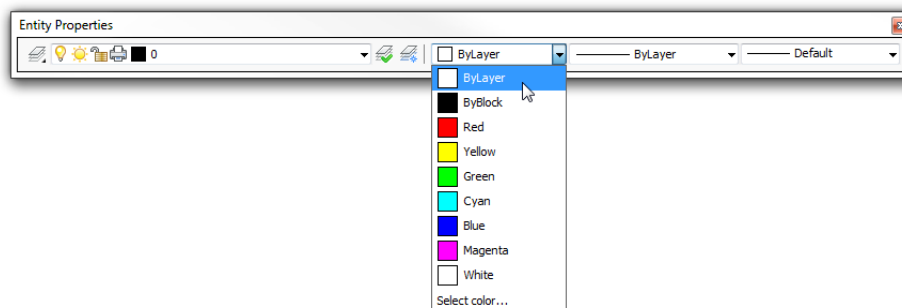
Adding Controls, Flyouts, and Separators

Toolbars can contain more than just buttons. There are other elements available, including *controls*, *flyouts*, and *separators*.

ABOUT CONTROLS (DROPLISTS)

Controls are better known as *droplists*. When the user clicks a control, BricsCAD drops a list of options, such as names of colors or linetypes. You cannot customize controls.

Four controls are illustrated below, with the Color control showing its droplist. From left to right, you see the controls for layers, colors, linetypes, and line weights.



Color control lists the names of colors

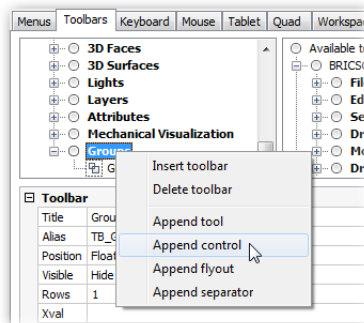
BricsCAD comes with these controls that you can add to and remove from toolbars:

- Color** — droplist of default and recently-used colors
- Layer** — droplist of layer names and settings
- Linetype** — droplist of loaded linetypes
- Lineweight** — droplist of standard line weights
- Text Style** — droplist of text style names
- Dimension Style** — droplist of dimension style names
- Plot Style** — droplist of table plot style names; available only when table-based styles are enabled
- Layer State** — droplist of named layer states
- Layer Filter** — droplist of named layer filters
- UCS** — droplist of named UCSes
- Perspective** — slider bar that toggles perspective mode and then zoom in or out
- Workspace** — droplist of available workspace names
- View** — droplist of named views
- MLeader Style** — droplist of multileader style names
- Visual Style** — droplist of visual style names

Tutorial: Adding Controls (Droplists) to Toolbars

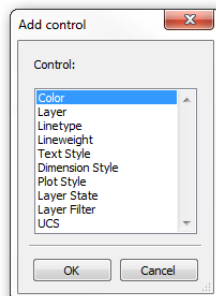
To add a control to a toolbar, follow these steps:

1. In the Customize dialog box's Toolbars tab, right-click an existing toolbar name. For this tutorial, choose **Groups**.
2. In the shortcut menu, choose **Append control**.



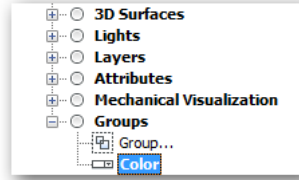
Choosing the Append tool from the shortcut menu

3. Notice the Add Controls dialog box. Choose the name of a control, such as “Color,” and then click **OK**.



List of controls

Notice that the control is added to the toolbar's list.

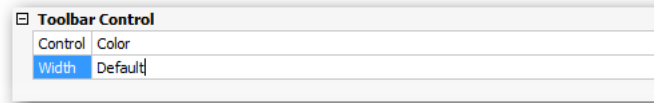


Control added to toolbar

4. Click **Apply** to see the control in the actual toolbar.

Customizing Controls (Droplists)

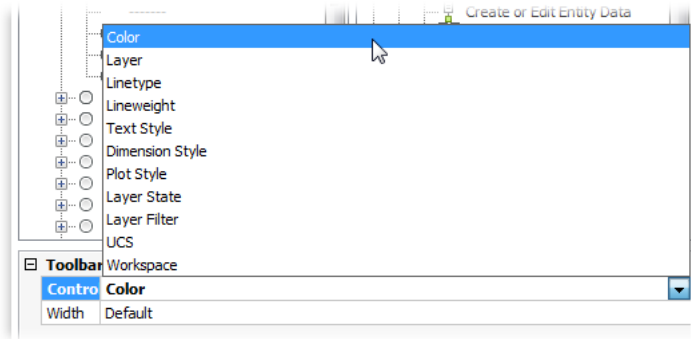
You cannot create new controls, but you can customize one aspect of them: their width.



Pair of parameters for controls

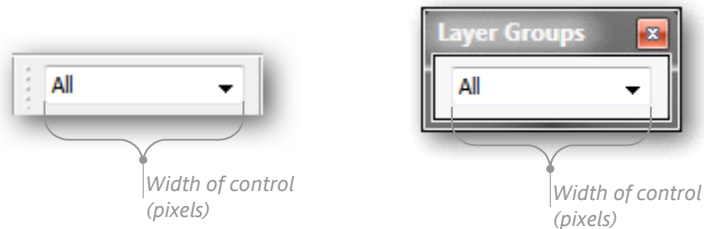
Here is what the parameters mean:

Control — changes the control displayed by the toolbar. Click to choose another one from the drop list:



Changing the control

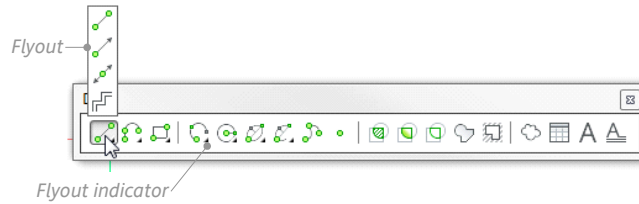
Width — specifies the width of the control. Click to enter a new width, which is measured in pixels. The figure below shows how the width is measured on docked and floating toolbars. The width measurement includes the gray line at either end of the white area.



Setting the width of the control

ABOUT FLYOUTS

Flyouts are sub-toolbars that “fly out” from a toolbar button, as illustrated below.



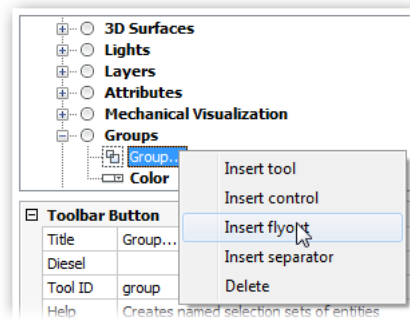
Flyout, and flyout indicator

The presence of a flyout is indicated by the small black triangle in the lower right corner of a toolbar button. Since flyouts are just toolbars within toolbars, you customize them kind of like a toolbar.

Tutorial: Adding Flyouts to Toolbars

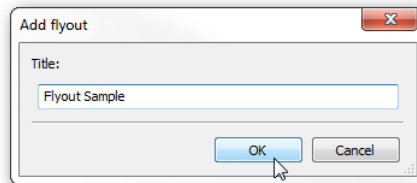
To add flyouts, it’s a bit tricky. Here are the steps involved:

1. Right-click an existing toolbar name. For this tutorial, choose **Group**.
2. In the shortcut menu, choose **Append Flyout**.



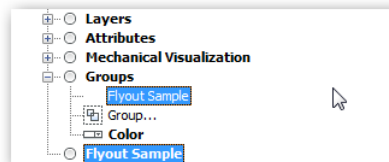
Inserting a flyout

3. Notice the Append Flyout dialog box. Give the flyout a name, and then click **OK**. (For this tutorial, I use the name “Flyout Sample.”)



Naming the new flyout

Notice that the flyout appears twice: once as a sub-toolbar and again as a toolbar in its own right.



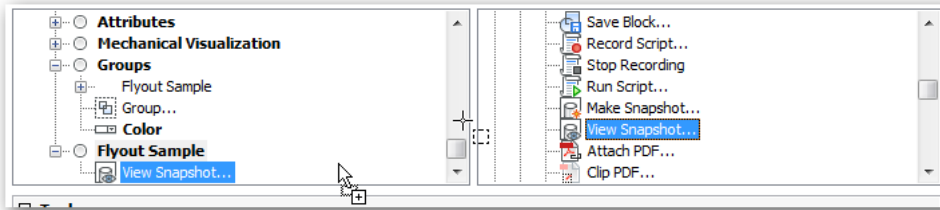
Double appearance of the flyout

4. You can now populate the flyout with tools in two places: the embedded Flyout Sample sub-toolbar or the vestigial Flyout Sample toolbar. The difference between them is as follows:

Embedded sub-toolbar — you must use the Append tool; you cannot use the Insert tool nor can you drag commands from the Commands pane

Vestigial toolbar — you can use the Insert tool and can drag commands from the Commands pane.

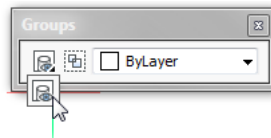
Any change you make to the vestigial toolbar appear in the embedded toolbar. I have no idea why Bricsys does things this way, but there you have it. I recommend using the vestigial toolbar, because you can simply drag'n drop. Drag tools from the Command pane onto the Flyout Sample toolbar.



Dragging and dropping commands into a toolbar

5. To add more commands to the flyout, repeat step 5. Notice that both toolbars contain the same list of commands.
6. Click **OK** to apply the changes and then view the changed toolbar.

The customized Groups toolbar now looks something like this:

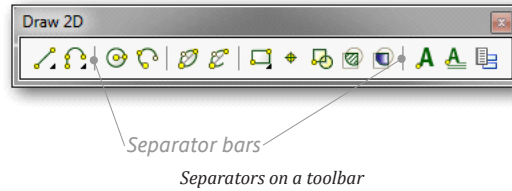


Toolbar with a button, a flyout, and a control

Because flyouts are simply toolbars, you can customize them just as you do toolbars. You cannot, unfortunately, simply drag existing toolbars on top of others to turn them into flyouts.

ABOUT SEPARATORS

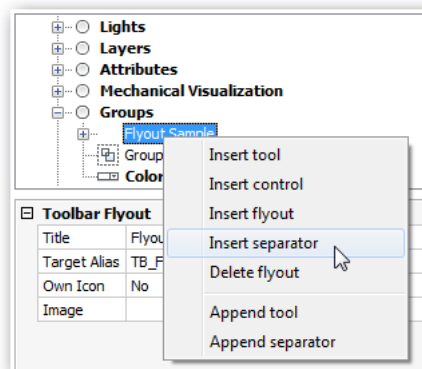
Separators are those lines that separate groups of buttons, as shown below. These are handy for visually segregating related groups of buttons. There is nothing to customize about separators: either you add them to a toolbar, or you do not.



Tutorial: Adding Separators to Toolbars

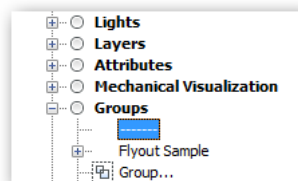
To add a separator to a toolbar, follow these steps:

1. Open a toolbar by clicking the **+** button. For this tutorial, choose Groups.
2. Select a tool name, such as the Color control. The separator bar is added in front of the selected tool.
3. Right-click, and then choose **Add separator**.



Choosing to add a separator

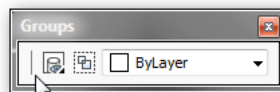
Notice a row of dashes (-----) is added to the Groups node, indicating the position of the separator bar.



Separator indicated by row of dashes

If you don't like the position of the separator, drag it elsewhere.

4. On the toolbar itself, a vertical gray line appears. Click **Apply** to see the change to the toolbar.



Separator added to toolbar

That's about as easy as it gets!

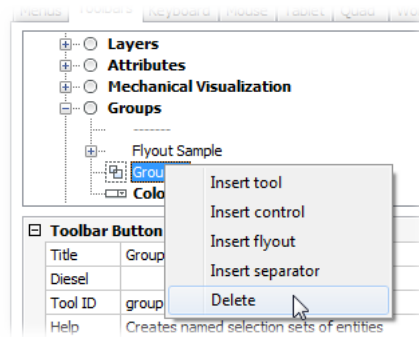
REMOVING BUTTONS, RENAMING AND DELETING TOOLBARS

You can remove buttons, rename toolbars, and delete them. To perform these actions, open up the Customize dialog box, go to the Toolbars tab, and then choose the toolbar you want to edit.

Tutorial: Removing Buttons and Toolbars

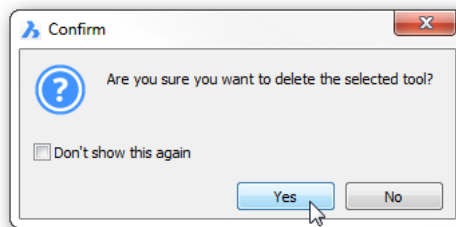
To remove a button from a toolbar:

1. Right-click a button's name, and then choose **Delete**.



Deleting a toolbar button

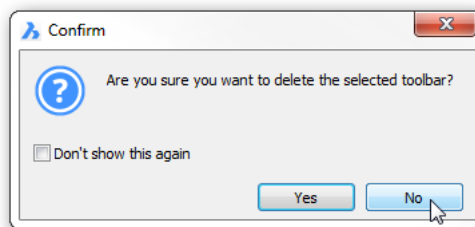
2. When BricsCAD asks whether you are sure, choose **Yes**.



Answering Yes (or No)

3. Click **Apply** to see the toolbar with one fewer button.

The same procedure is used to delete toolbars: right-click a toolbar name, and then choose **Delete Toolbar**.

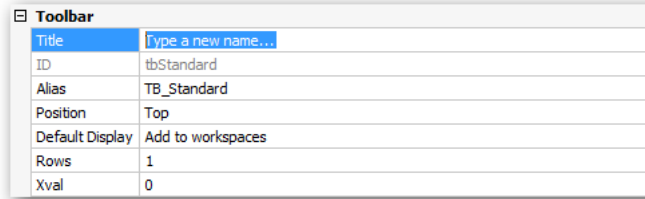


Being cautious about deleting an entire toolbar

Tutorial: Renaming Toolbars and Buttons

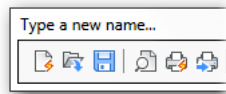
You can change the names displayed by toolbars and buttons. To rename a toolbar, follow these steps:

1. Select a toolbar in the Customize dialog box.
2. In the Macro pane, choose the **Title** parameter.
3. Edit the name.



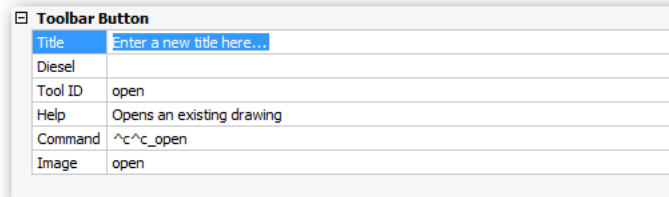
Renaming a toolbar

4. Click **Apply** to see the name change on the toolbar's title bar, if it is a floating toolbar.



Toolbar sporting its new name

You rename buttons in the same manner: select a button in the Customize dialog box, and then edit its **Title** parameter.



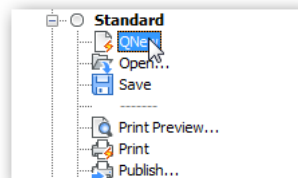
Renaming a button

The name appears in the tooltip when you hover the cursor over the button.

Customizing Buttons

Buttons are customized in a manner similar to that of toolbars. The parameters that can be changed are described by the boxed text, above. To start customizing buttons, follow these steps:

1. Enter the **Cui** command to open the Customize dialog box, and then select the **Toolbars** tab.
2. Open any toolbar by clicking the **+** next to its name. For example, open **Standard**.



Selecting a button to customize

3. Select a command name, such as **QNew**. Notice the button parameters that appear at the bottom of the dialog box, such as Title and Diesel.

Toolbar Button	
Title	QNew
Diesel	
Tool ID	qnew
Help	Creates a new drawing from the current default drawing template
Command	^c^c_qnew
Image	qnew

Parameters that can be modified on toolbar buttons

With the Customize dialog box ready to modify buttons, let's go on to see what can be done in regards to this.

MODIFYING BUTTON PARAMETERS

You can change the following button parameters:

Title — specifies the name displayed by the tooltip. The tooltip appears when you hover the cursor over the toolbar button.

Help — specifies the help text displayed on the status bar. The text appears in the status bar, again when you hover the cursor over the button.




Command — specifies macro executed by clicking the button. This macro can consist of a simple command name, like *line*, or multiple command names within a lengthy series of instructions.

Image — specifies the picture (a.k.a *icon*) displayed by the button. You can use icons provided by Bricsys or use your own.

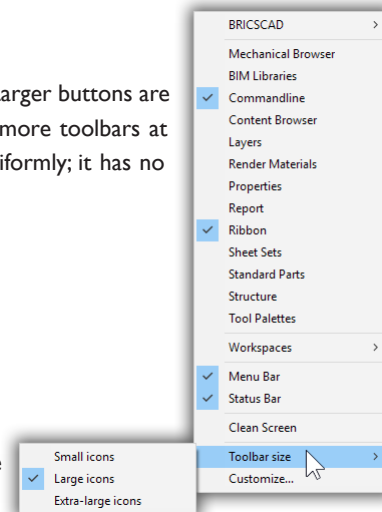
I recommend that you leave Diesel and Tool ID fields alone. Diesel, because its functions are carried out by the Command field; ToolID, because it's best not muck about with how BricsCAD identifies buttons internally.

SIZING BUTTONS

You can have three sizes of buttons on toolbars: regular, large, and extra-large. Larger buttons are easier to see on very-high resolution monitors, but smaller ones let you see more toolbars at a time. The **ToolbarIconSize** variable affects all buttons on every toolbar uniformly; it has no impact on ribbon buttons.

-  **Regular:** 16x16 pixels
-  **Large:** 32x32 pixels
-  **Extra-large:** 64x64 pixels

The easiest way to change the size is to right-click a UI element, and then choose **Toolbar Size**.



Tutorial: Editing the Title Name and the Help String

To change the name displayed by the button's tooltip, follow these steps:

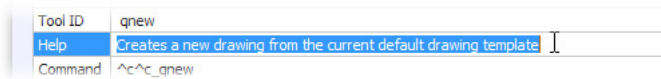
1. Click the field next to the **Title** parameter.



Editing the Title parameter

2. Edit or replace the text.
3. Click **Apply** to make the change stick.

Follow the same steps to change the help text displayed by the status bar: Click the Help field, edit the text, and then click **Apply**.

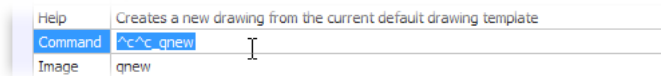


Editing the help text

Tutorial: Changing the Command Macro

To change the macro that is executed when you pick the toolbar button, follow these steps:

1. Click the field next to the **Command** parameter.



Editing the command's macro

2. Enter a new macro. If all you want to is to execute a single command, then use this template:
`^c^c_command`

Replace “command” with the command name of your choice. For example, to execute the PLine command, then enter `^c^c_pline`. (For details on writing macros, see the chapter on “Writing Macros and Diesel Code.”)

3. Click **Apply**.

TIPS After you change a parameter, it is shown in boldface to remind you that it has changed. The boldfacing goes away after you press **Apply**.

Click the **Apply** button to see the effect of changes you've made to the button(s).

Although it is not explicit, you can copy and paste text in the parameter fields, as follows:

- To copy: select text, and then press **Ctrl+C**.
- To paste: place cursor, and then press **Ctrl+V**.

Ctrl+X also works to cut text, as does **Ctrl+Z** for undo, **Ctrl+Y** for redo, **Ctrl+A** to select all text, and **Del** to delete.


Tutorial: Replacing Button Images

To change the picture displayed by the button, follow these steps:

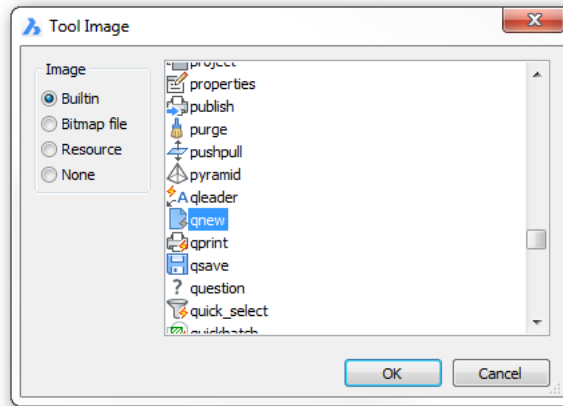
1. Pick the field next to the **Image** parameter.



Accessing the button editor


2. Notice the  button at the end of the field. Click it to access the Tool Image dialog box. This dialog box offers four ways to access collections of pictures (or icons):

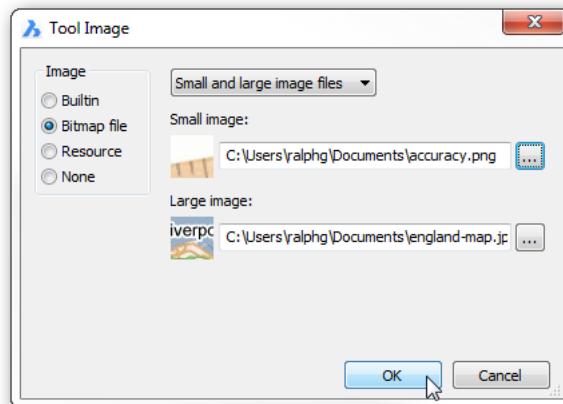
Builtin — lists images available within BricsCAD. Scroll through the list, choose an image, and then click **OK**.



Choosing an icon from those provided by BricsCAD

Bitmap file — selects an image on your computer. This takes two steps:

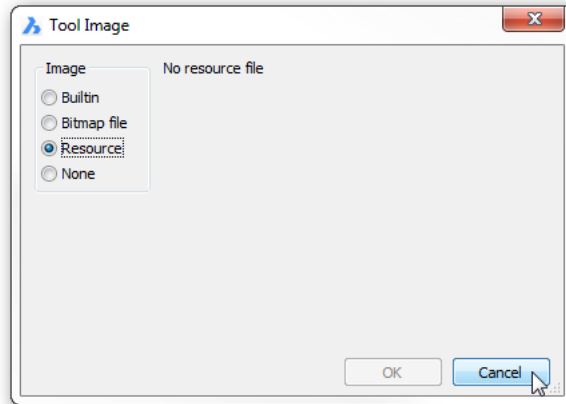
- a. Choose the size:
 - **One Image File** — standard size only (16x16 pixels)
 - **Small and Large Image Files** — both standard and large (24x24 pixels).
- b. Click the  button.
- c. From the Tool Image dialog box, choose a file in BMP (bitmap), GIF, JPEG, or PNG format. The image is automatically resized to fit the area of the button.



Choosing an icon from a file

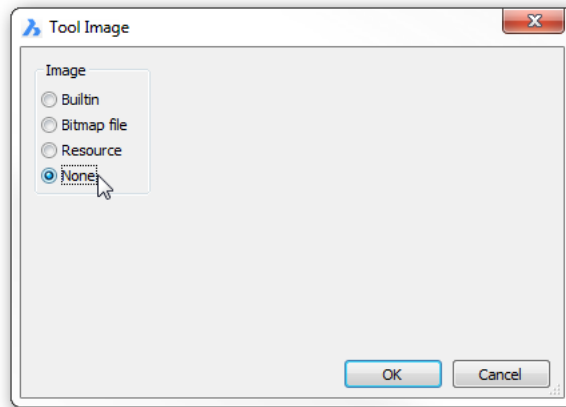
- d. Click **Open**, and then repeat for the large image, if necessary.
- e. Click **OK**.

Resource — chooses an image from a resource file. These *.dll* or *.exe* files are used by Windows to run programs, and often contain a small collection of icons. At time of writing this book, this option did not work, and so all you can do is click **Cancel**.



Choosing an icon from a resource file, if any

None — removes the image from the button. In this case, the button is blank.



Specifying no icon for the button

BricsCAD does not have a built-in icon editor. Instead, you can use a raster editor like PaintShop Pro to create images, and then use the **Bitmap File** option to load them into BricsCAD.

Writing Macros and Diesel Code

When you click a button or make a menu selection, BricsCAD behind the scenes executes a *macro*. This is a series of one or more commands assigned to the button or menu item.

With the Customize command, you can change the macros that lie behind buttons, menu selections, mouse clicks, and other actions. Toolbars, ribbons, and most other parts of the BricsCAD user interface all use the same format for macros. This is good news for this reason:

**When you learn to write a macro for one,
then you know how to write macros for all**

There is just one difference, however: menus have more user interface options than the others, and they so have more capabilities when it comes to macros.

Some macros use *Diesel*, a simplistic programming language. It is used for special effects, such as toggling check marks in front of menu items. The coding used by *Diesel* is really, *really* arcane. Fortunately, the same bits of Diesel code can be used over again, and so it is enough for you to recognize what the code does.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Learning the macro syntax
- Writing macros specific to menus
- Coding with Diesel
- Cataloging Diesel functions

QUICK SUMMARY OF METACHARACTERS IN MACROS

Macros use command and option names, metacharacters, Diesel code, and LISP functions in menus, toolbars, and other areas of the Customize dialog box. Metacharacters consist of punctuation that represents actions. Here are the metacharacters used by BricsCAD:

^ — (carat) represents the **Ctrl** key. These following control-key combinations are valid in macros:

- ^B Toggles snap mode
- ^C Cancels the current command
- ^D Toggles coordinate format
- ^E Changes to the next isometric plane
- ^G Toggles the grid display
- ^O Toggles orthographic mode
- ^S Selects the entity under the cursor
- ^T Toggles tablet mode

; — (semi-colon) represents the Enter key.

' — (single quote) forces the use of commands in transparent mode.

_ — (underscore) forces the use of English versions of command names.

- — (dash) forces the use of command-line versions of commands.

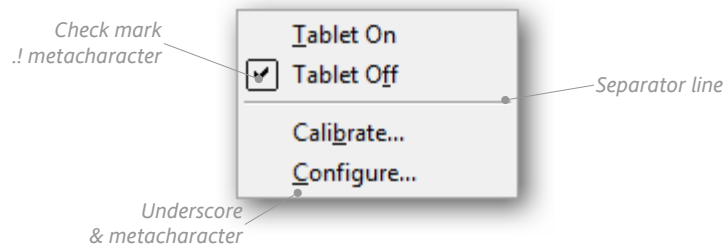
\ — (backslash) pauses the macro for user input.

(— (open parenthesis) signals the start of a LISP function.

\$(— signals the start of a Diesel statement.

) — (close parenthesis) signals the end of LISP functions and Diesel statements.

The following metacharacters are used only by menu macros:



\$M — signals the start of a complex macro.

.! — displays a check mark, to indicate the toggle is turned on.

~ — (tilde) grays out a menu item, to indicate it is not available.

& — (ampersand) signals the accelerator key, to access menu items with the **Alt** key.

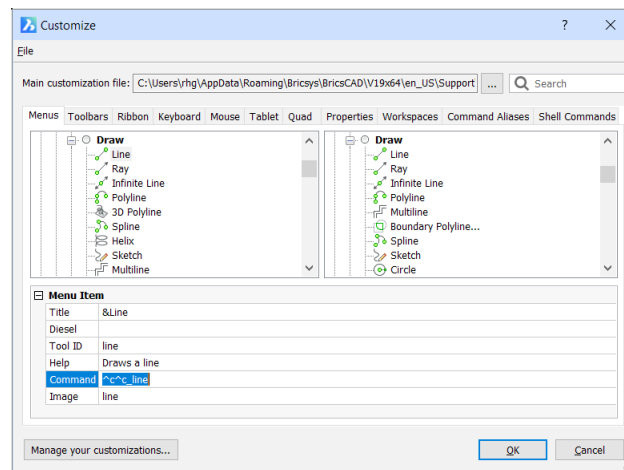
In this chapter, you learn how to write macros, and then how to add *more* power to macros through one-line programs written in Diesel.

Simple Macros

A simple macro consists of a single command, prefixed by some unusual-looking characters. For instance, here is the macro attached to the Line button in Draw toolbar:

```
^c^c_line
```

The exact same macro is used for toolbars and macros, as shown below by Line in the Draw menu:



A basic macro shown in the *Command* property

The `^c^c_` characters in the macro have the following meanings:

`^c` — is a *control character*. It imitates pressing **Esc** on the keyboard, canceling the command currently in progress. The carat symbol (`^`) alerts BricsCAD that the character following is a control character, and not part of a command name or an alias.

(What does ‘c’ have to do with the Esc key? Back in the 1980s and 1990s, when desktop computers used MS-DOS for their operating system, users pressed `Ctrl+C` to cancel a command; the **C** was short for “cancel.” With Windows, Microsoft changed the meaning of `Ctrl+C` to mean “copy to Clipboard,” but in macros, it continues to mean “cancel.”)

`^c^c` — most macros start with two `^cs` because many BricsCAD commands are two levels deep. Extra `^cs` do no harm; indeed, older releases of BricsCAD prefixed all macros with three `^cs`. to handle commands like `PEdit`, whose options can go three levels deep.

When a macro is *transparent* (starts with the ' apostrophe), then you can't prefix it with the Cancel characters; more on this later in this section.

`_` — the underscore is a convention that *internationalizes* the command. Prefixing the command name with the underscore ensures the English-language version of the command always works, whether used with a German, Japanese, or Spanish versions of BricsCAD.

`line` — is the name of the command to be executed. In macros, you type BricsCAD commands and their options exactly the way you would type them on the keyboard at the `:'` command prompt.

Nothing is needed at the end to terminate the command. BricsCAD automatically does the “pressing **Enter**” for you.

TIP Macros are case-insensitive. This means that the characters in macros can be upper or lowercase, or mixed case; it matters not to BricsCAD. The following have the same effect:

```
^^_LINE
^^c_line
^^c_Line.
```

TRANSPARENT COMMANDS IN MACROS

Most macros start by cancelling existing commands. But sometimes you want to use a command *transparently*; i.e, during another command. For example, you might want to zoom into the drawing during a command.

Transparent commands are indicated by the apostrophe prefix ('), like this:

```
'_zoom
```

Dashed Commands

A few commands in BricsCAD start with a dash; these are ones that operate at the command line, instead of displaying a dialog box.

One example is the **View** command: View displays a dialog box, while **-View** display prompts at the command line. To force View to display its prompts at the command line, enter this:

```
^^c_-view
```

OPTIONS & USER INPUT

Macro can specify commands options, as well as wait for input from users.

Options

Options are just like commands; you just write out the option name. The only thing to watch is that you should use semicolons (;) to separate options from commands and each other.

Here is an example with the Layout command and its New option:

```
^^c_layout;_new
```

Notice that options can also receive the underscore (_) prefix to internationalize them.

You can spell out the name of the option in full (_new) or use the approved abbreviation, such as _n). Recall that approved abbreviations are indicated by capitalized letters in the option names displayed at the command prompt, such as these:

```
: layout
Enter layout option [Copy/Delete/New/Rename/Set/SAve/Template/? to list] <Set>: n
```

You can use “n” for the New option and “s” for the Set option, but must use “sa” for the Save option.

It is perfectly valid to use “?” (as in “? to list”) in macros, but you cannot use spaces, because these are interpreted as pressing **Enter**.

Pausing for User Input

To allow users to input data, macros employ the backslash character (\). This forces the macro to wait for the user to do something. Commonly, the expected action is for the user to input one of these:

- ▶ Pick a point on the screen, or
- ▶ Enter x,y-coordinates at the keyboard

The macro waits for the user to enter the center point of the circle; the circle is always 1 unit in radius:

```
^c^c_circle;\1
```

When you execute this macro, such as from a menu pick or toolbar button, the following occurs at the command line:

Command Prompt	Comments
: _circle	<i>Macro begins the command</i>
2Point/.../<Center of circle>: (User picks a point)	<i>Macro waits for user to pick a point in the drawing (or enter a coordinate pair)</i>
Diameter/<Radius> <5>: 1	<i>Macro enters 1 (for the radius) and then ends the command</i>

But the expected action can be other things, too, depending on the command. Options in the Rotate command expect an angle (the user picks two points on the screen, or enters a single number), in the Text command a line of text (the user enters one or more words), and so on.

When a command expects more than one input from the user, you can type several back slashes in a row, as you see next.

Combining Options and Pauses

Options and pauses can be combined together. In this example, the macro draws an ellipse after the user specifies a center point and the rotation angle:

```
^c^c_ellipse;_c;\_r
```

Here is what the code means:

^c^c cancel any other command that might be active at the time.

Ellipse is the name of the command, while the underscore (_) prefix internationalizes it.

; (semicolon) is just like pressing **Enter** or the spacebar on the keyboard.

C is the Ellipse command's **Center** option. Just as you enter an abbreviation for options at the keyboard, so too you can use the same abbreviations in macros — or you can spell out the entire option name, such as “Center.”

**** (backslash) pauses the macro, waiting for the user. Two backslashes in a row means that the macro expects the user to make two picks.

R is short for the Ellipse command's **Radius** option.

Let's look again at the macro, this time in parallel with the command's prompts:

Macro	Command Prompt
^C^C	(Press <i>Esc, Esc.</i>)
_ellipse;	: ellipse
;	(Press <i>Enter.</i>)
_C;	Arc/Center/<First end of ellipse axis>: c (Press <i>Enter.</i>)
\	Center of ellipse: (Pick point.)
\	Endpoint of axis: (Pick point.)
_R	Rotation/<Other axis>: r (Press <i>Enter.</i>)
	Rotation around major axis: (Pick point.)

A final semicolon (i.e. Enter) and backslash (i.e. pause for user input) are not needed at the end of the macro, because the macro no longer needs to wait for the user.

TIPS You can include aliases, Diesel code, and LISP routines in toolbar and menu macros.

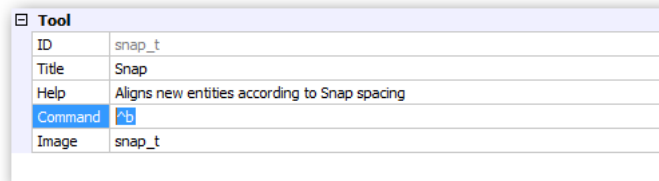
There is no “debugger” for macros, and so you have to figure out the errors on your own.

Other Control Keys

You've met ^C, the control key for cancelling a command. BricsCAD also supports all these control-key combinations using the ^-prefix:

Control Key	Meaning	Command Equivalent
^B	Toggle snap mode	_snap;_t
^C	Cancel command	<i>Press Esc</i>
^D	Toggle coordinates	_coordinate;_t
^E	Change isometric plane	_isoplane
^G	Toggle grid display	_grid;_t
^O	Toggle orthographic mode	_orthogonal;_t
^S	Selects the entity under the cursor	...
^T	Toggle tablet mode	_tablet;_t

Think of these control-key combos as abbreviations, like aliases. You can use these control keys as shortcuts in macros all by themselves, like this:



Macro consisting solely of a Ctrl-key macro

MENU-SPECIFIC METACHARACTERS

Menus use additional metacharacters that are not needed by toolbars. Here is the complete set:

Metacharacter	Meaning
.!	Displays a check mark to indicate the toggle is turned on.
~ (tilde)	Grays-out menu item to indicate it is not available.
& (ampersand)	Enables accelerator key to access menu item from the keyboard.
Other Metacharacters	
_ (underscore)	Internationalizes the command or option.
' (quote)	Starts a command transparently.
\ (backslash)	Waits for user input.
;	Equivalent to pressing Enter or the spacebar.
\$(Starts a Diesel statement.
(Starts a LISP routine.

TIP Some of AutoCAD's metacharacters don't work, such as [], +, \t, and *.

Diesel Coding

Sometimes you need additional code to help macro perform decisions. For example, the View menu lists three items that have check marks beside them: Command Bar, Status Bar, and Scroll Bars. When the three bars are displayed, the check marks appear in the menu; when not displayed, the check marks are not shown.



*Left: Check marks indicate UI elements are turned on.
Right: No check marks mean the elements are turned off.*

It is easy to get a menu to display the check mark: just add the !. metacharacter to the macro. It is difficult, however, to get BricsCAD to do the actual turning on and off, because the display of check marks is a logical function. It should appear in the menu only when the UI element is turned on. This is where Diesel comes in.

ABOUT DIESEL

Diesel has two purposes in macros: one is for making decisions, and the other is for customizing the status bar. The name is short for “direct interactively evaluated string expression language,” and its programming logic is as clear as the acronym’s meaning — as clear as mud.

QUICK SUMMARY OF DIESEL FUNCTIONS

The following Diesel functions are supported by BricsCAD:

MATH FUNCTIONS

+	Addition
-	Subtraction
*	Multiplication
/	Division

LOGIC FUNCTIONS

=	Equal
<	Less than
>	Greater than
!=	Not equal
<=	Less than or equal
>=	Greater than or equal
and	Logical bitwise AND
eq	Determines if all items are equal
if	If-then
or	Logical bitwise OR
xor	Logical bitwise XOR

NUMERIC CONVERSION FUNCTION

fix	Truncates real numbers to rounded-down integers
angtos	Formats angles (short for angle to string)
rtos	Formats numbers with units (short for real to string)

STRING (TEXT) FUNCTIONS

index	Extracts one element from a comma-separated series
nth	Extracts the nth element from one or more items
strlen	Returns the number of characters in the string (short for string length)
substr	Returns a portion of a string (short for sub string)
upper	Converts a text string to uppercase characters

SYSTEM FUNCTIONS

edtime	Formats the system time.
eval	Passes a string to Diesel.
getenv	Gets the value of an environment variable.
getvar	Gets the value of a system variable.
linelen	Returns the length of the display

Diesel has an unusual format for a macro language. Every function begins with a dollar sign and a parenthesis, like this:

```
$(function,variable)
```

The purpose of the initial \$-sign is to alert BricsCAD that a Diesel expression is on its way, just as the (symbol alerts BricsCAD that a LISP expression is coming up. The \$ symbol is often used by programmers to indicate a string of text.

The opening and closing parentheses signal the beginning and end of the Diesel function. Functions can be *nested*, where one Diesel function is inside the parentheses of a second one. You use nesting to have one function evaluate the result of the second one. Diesel is completely reentrant.

Because Diesel programs consist of just one line — at most! — nesting is the only way to carry out more than one function during a macro.

For some functions, Diesel can operate on as many as nine values at a time, such as adding several values together. The closing parenthesis alerts Diesel to the end of the list of values.

BricsCAD provides a catalog of 26 Diesel functions. Most of them use at least one variable, some as many as nine. A comma always separates the function name from its variable(s), as well as the variables themselves. *Diesel tolerates no spaces.*

Diesel functions can be run at the command line, in toolbar and menu macros, in LISP code, and in other areas of BricsCAD, such as the status bar. To work in the status bar, you use the **ModeMacro** command, followed by the Diesel expression.

(John Walker, the Autodesk programmer who created Diesel, notes that additional functions, such as **setvar** and **time**, could be implemented but never were. He provides instructions for accessing the Diesel source code and recompiling it with other functions at <http://www.fourmilab.ch/diesel/>. He named Diesel as “Dumb Interpretively Executed String Expression Language.”)

HOW TO TOGGLE CHECK MARKS

BricsCAD primarily uses Diesel to toggle check marks in menus:

- ▶ Check mark means the option is turned on
- ▶ No check mark means the option is turned off

To switch between the two states, BricsCAD uses the `!.!` metacharacter in code that looks like this:

```
$(if,$(=,$(getvar,FILLMODE),0),,!.)
```

It may look weird, but don't worry: you don't need to know how to write that code from scratch, ever. All you need to do is: (a) copy and paste it, and then (b) change just one word (FILLMODE, in this case).

Here is what the Diesel code says: “If the value of FillMode equals 0, display nothing; otherwise, display the check mark.” BricsCAD uses the `!.!` metacharacter to instruct menus to display check marks.

Here is another way of looking at the Diesel code. This is called “parsing,” where each line of code is given its own, indented line:

```

$(if,
  $(=,
    $(getvar,
      FILLMODE),
    0)
  ,
  ,!.
)

```

```

If...
...equal to
...get the value of
...system variable FillMode
... (equal to) zero
... then display nothing.
      Otherwise, display the check mark.
End of diesel statement.

```

Here is what the code does: it checks the value in system variable **FillMode**. If the value is 0, then the check mark is not displayed; if the value is **1**, then the check mark is displayed.

To use this code for other menu items, copy and paste the text, and then change the name of the system variable. For example, to add a check mark toggle to the Limits command, use **LimCheck** system variable. Simply copy, paste, and edit the Diesel string to make it look like this:

```
$(if,$(=,$(getvar,LIMCHECK),0),,!.)
```

Reuse the same code for the **Grid** command, which uses the GridMode system variable:

```
$(if,$(=,$(getvar,GRIDMODE),0),,!.)
```

So, you don’t really need to know what the Diesel code does; you just need to know which word to change!

Toggling Grayouts

To toggle the color of menu text between black and gray, you use the tilde (~) character:

- ▶ **Black** text means the menu item is available
- ▶ **Gray** text means the menu item is unavailable

For example, BricsCAD uses the tilde in Diesel code to check for valid sublicenses. If you’ve paid for the Pro and Platinum versions of BricsCAD, then you get access to solids modeling and Visual Basic programming. If not, then no. (Bricsys pays to license the ACIS and VBA technology from Spatial Technology and Microsoft, hence the higher cost of the Pro and Platinum versions.)

The following code is used to check for licenses:

```
$(if,$(=,$(and,$(getvar,LICFLAGS),0x1),0),~,)
```

If nonzero, then the submenu is available for your use. If zero, then the menus are grayed out. The read-only LicFlags system variable contains a bitcode that signals which licenses are valid:

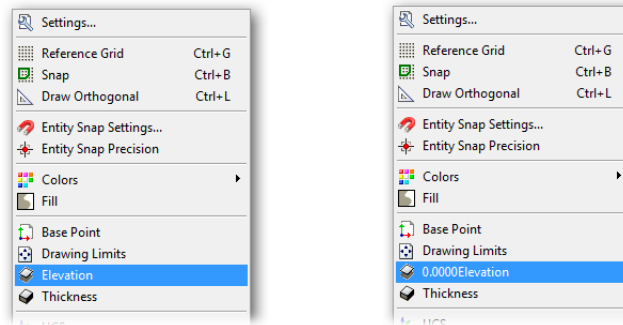
LicFlags	Meaning
1	Microsoft license for Visual Basic programming
2	Spatial license for 3D solid modeling and editing, ACIS import and export, regions
4	Bricsys license for Platinum edition

REPORTING VALUES OF SYSTEM VARIABLES

Being the hackers, er, customizers that we are, we won't stop at toggling mere check marks or text colors. We've figured out how to use Diesel to do more.

For instance, to display the *values* of system variables, we can use the `$(getvar` function. This Diesel function *gets* the value of a system *variable*, and then displays it in the menu.

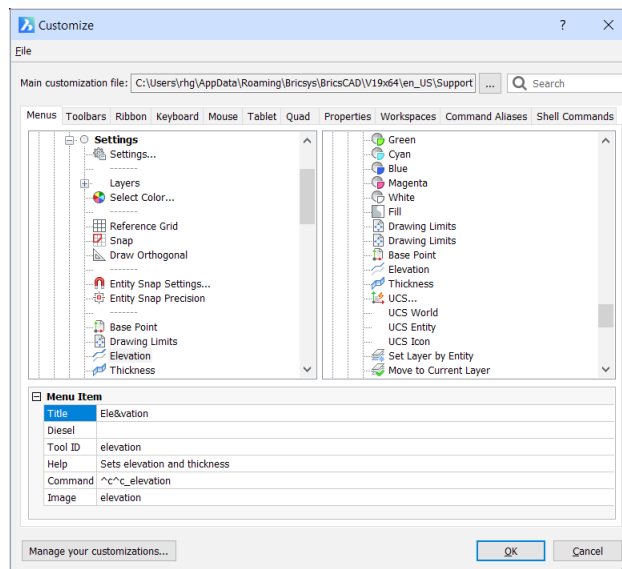
In the following tutorial, you change the Elevation menu item so that it reports its current value. (Elevation is found in the Settings menu.) The figures below illustrate how the menu looks before and after this tutorial. In the “after” picture, Elevation reports its current value of 10.9000:



Left: The default version of Elevation in the Settings menu; right: Elevation modified to show value, using Diesel code

To modify a menu item so that it reports values, follow these steps:

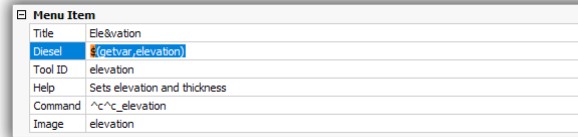
1. Enter the **Cui** alias, and then choose the **Menus** tab.
2. Expand the **Settings** node, and then select the **Elevation** item.



Elevation command in the Customize dialog box

3. Shift your attention to the Menu Item area, the macro pane at the bottom of the dialog box. Click the field next to **Diesel**, and then enter the following code:
`$(getvar,elevation)`

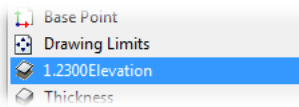
This piece of Diesel code gets the value of the Elevation system variable, and then displays it.



Menu Item	
Title	Ele&vation
Diesel	\$(getvar,elevation)
Tool ID	elevation
Help	Sets elevation and thickness
Command	^c^c_elevation
Image	elevation

Adding Diesel code to the macro

4. Click **OK** to apply the change and exit the Customize dialog box.
5. Choose the **Settings** menu, and then notice the change to the Elevation item. It will probably be prefixed with 0.0000 — the current elevation.
6. Choose **Elevation** to run the command, and then enter a different value, like 1.23:
: elevation
Enter current new value for ELEVATION <0.0000>: 1.23
7. Choose the **Settings** menu again. Notice that the value next to Elevation has changed to 1.2300.



Actual elevation value added to menu

You've made the menu more useful by customizing the display of the Elevation item! But there is a small problem with the display: it doesn't look very good, with the "1.2300" jammed up against the word "Elevation."

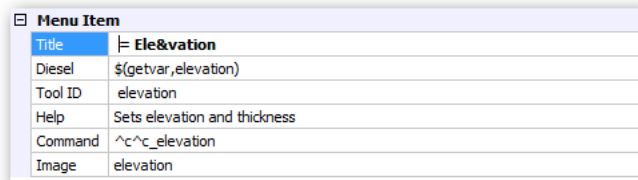
In this next tutorial, you fix the spacing problem:

1. Reenter the **Cui** alias, and then return to the **Settings | Elevation** item.
2. Edit the field next to **Title** so that it changes from
Ele&vation

to this:

= Ele&vation

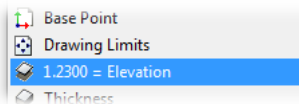
Just add a space, equals sign, and another space.



Menu Item	
Title	= Ele&vation
Diesel	\$(getvar,elevation)
Tool ID	elevation
Help	Sets elevation and thickness
Command	^c^c_elevation
Image	elevation

Enhancing the macro

3. Click **OK** to exit, and try the **Settings** menu again. That looks better!



Enhanced look of the menu item

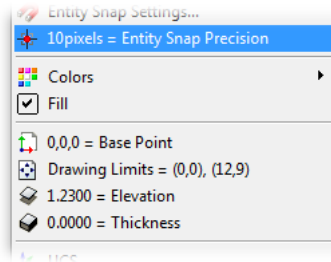
The number will always appear in front of the word. The reason? Recall that the Diesel code was meant to toggle check marks, which appear in front of words. Since displaying values is a *hack*, we are stuck with the backward looking "1.2300 = Elevation." There is, however, a workaround, as described later under "How to Deal with Two Sysvars."

APPLYING VARIABLES EVERYWHERE

You can apply the same sort of change to other items in the Settings menu. Here are the names of the system variables for some of them:

Settings Menu	System Variable(s)	Diesel Code
Entity Snap Precision	Aperture	<code>\$(getvar,aperture)</code>
Base Point	InsBase	<code>\$(getvar,insbase)</code>
Drawing Limits*	LimMin, LimMax	<code>\$(getvar,limmin),\$(getvar,limmax)</code>
Thickness	Thickness	<code>\$(getvar,thickness)</code>

After the changes are applied, the Settings menu looks like this:



Additional menu items reporting values

There are some special things to notice about the menu illustrated above. Let's go through them.

How to Add Units

The value of 10 shown for **Entity Snap Precision** is somewhat meaningless, so I added the word "pixels" to the Title parameter, like this:

```
pixels = Entity Snap &Precision
```

Menu Item	
Title	pixels = Entity Snap &Precision
Diesel	<code>\$(getvar,aperture)</code>
Tool ID	aperture
Help	Sets selection area for snapping to entities
Command	'_aperture
Image	aperture

Adding units to values

How to Solve Check Marks that Conflict with Icons

I found that the image (icon) for **Fill** seems to override the check mark. In the figures below, which is on and which is off?



Left: Fill is ???
Right: Fill is ???

Here's why there's a problem: when a toggle is on, the icon gets a thin black border, which I find is easy to miss with the Fill, because it already has a black border. This is why I prefer the bold-looking check mark over the essentially-invisible border.

One solution is to remove the icon from the Image field:

Menu Item	
Title	&Fill
Diesel	<code>\$(if,\$(=,\$(getvar,FILLMODE),0),,1.)</code>
Tool ID	fill_t
Help	When on, planar entities are drawn filled with color
Command	<code>^c^c_fill;_t</code>
Image	

Removing the icon from the menu item

Now the check mark is prominent:



*Left: Fill is off with blank icon
Right: Fill is on with checkmark icon*

How to Deal with Two Sysvars

At first, I could not get the **Drawing Limits** item to work correctly. It extracts values from two system variables, LimMin and LimMax, which is tricky. After some fiddling around, I found that I could get the Diesel code to work by placing part of it in the **Title** parameter, like this:

Menu Item	
Title	<code>&Drawing Limits = (\$(getvar,limmin)), (\$(getvar,limmax))</code>
Diesel	<code>\$(if,\$(eq,\$(getvar,LIMCHECK),ON),1.,)</code>
Tool ID	limits_t
Help	Sets the drawing limits, and turns them on or off
Command	<code>^c^c_limits;_t</code>
Image	limits_t

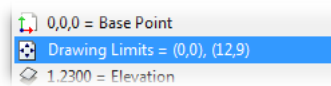
Code for reporting two variables

Notice that the pieces of Diesel code are surrounded by parentheses, and separated by a comma. This makes the pair of 2D coordinates more legible.

```
&Drawing Limits = ($(getvar,limmin)), ($(getvar,limmax))
```

This macro displays the limits as follows:

```
(0,0), (12,9)
```



Menu item reporting the values of two variables

Reporting Through Diesel

Other menus can take advantage of Diesel's reporting feature. Here are examples of what's possible:

- ▶ **File | Close** can report the name of the drawing file with the **DwgName** system variable.
- ▶ **Edit | Undo** can report the name of the command being undone with **CmdName**.
- ▶ **View | Set Viewpoint** can report the coordinates of the current view with **VPointX**, **VPointY**, and **VPointZ**.
- ▶ **Insert | Insert Block** can report the name of the last-inserted block with **InsBase**.
- ▶ **Draw | Circle** can report the current radius with **CircleRad**.
- ▶ **Dimension | Restore Dimension Style** can report the name of the current dimensions style with **DimStyle**.
- ▶ **Modify | Fillet** can report the current fillet radius with **FilletRad**.
- ▶ **Settings | TextStyle** can report the name of the current text style with **TextStyle**.
- ▶ **Tools | Inquiry | Time Variables** can report the duration the drawing has been open with **TdInDwg**.

Formatting Units

In the figure above, the values of Base Point and Drawing Limits are shown in architectural units. This should come as a surprise you, because normally they would be displayed by default as decimal units. In this case, I cheated: I didn't use Diesel, but simply changed format of the units with the **Units** command.

This example illustrates that some values in menus are affected by the current setting of Units. Other values, such as Elevation and Thickness, are, however, still shown by four decimal places. This can be overridden using Diesel code, as described next.

Formatting Diesel Output

You can apply formatting to the numbers and text generated by Diesel. Numbers and angles can be formatted for units, while text can be converted to uppercase, or be truncated.

FORMATTING NUMBERS

Diesel provides functions for rudimentary formatting of numbers and coordinates.

Fix

The **Fix** function truncates real numbers to rounded-down integers. For example, if a Diesel calculation returns the value of 5.321, then applying the Fix function changes the value to 4. "Rounding down" means that a value like 5.987 (which you would expect to be rounded up to 6) is also truncated to 4:

```
$(fix,4.321) returns 5
```

Index

The **Index** function extracts a single coordinate value from a comma-separated series. For example, the BasePoint system variable returns a x,y,z coordinate like this:

```
(4,11,16)
```

You use the Index function to extract the x coordinate from (4,11,16), like this:

```
$(index,0,($getvar,basepoint))           returns 4
```

Notice that Diesel uses a radix of 0, meaning it starts counting with 0, instead of 1 as we humans do. Thus, the **0** in the function above extracts the first coordinate, x:

Index Number	Coordinate Extracted
0	x
1	y
2	z

(When we count, we count like this: 1, 2, 3...; but when Diesel counts, it counts like this: 0, 1, 2... . This makes counting in Diesel complex, because you have use a digit that's one less than what you would expect to use.)

Nth

The **Nth** function extracts the nth element from one or more items. Here, the **2** returns the third element of the string of numbers, 8 (because Diesel starts counts with 0, not 1).

```
$(nth,2,10,9,8,7)                       returns 8
```

Both Index and Nth work with numbers and text.

Rtos

The **Rtos** function formats numbers with units. The function name is short for “real to string,” but has nothing to do with strings! (A similar function, **Angtos**, formats angles.) Here is how to use it. Say a drawing has units that are architectural, but you want Diesel to report numbers in decimal notation, with one decimal place of accuracy. In the following example, the Rtos function formats the first chamfer distance, ChamferA:

```
$(rtos,($getvar,chamfera),2,1)
```

Where:

`($getvar,chamfera)` — name of the system variable, the *source* of the real number that you want to format.
`2` — *format* of the number, decimal in this case. Diesel uses the same code as LUnits; see the table below for more info. When you leave out this digit, BricsCAD reads the value found in the LUnits (linear units) sysvar.

Mode (LUnits)	Number Display Format
1	Scientific notation (exponential format)
2	Decimal format (metric)
3	Engineering format (feet and decimal inches)
4	Architectural format (feet and fractional inches)
5	Fractional format (fractional inches, no feet)

1 — *precision* of the number, one decimal place in this case. When this digit is left out, then the value of Lu-Prec (linear units precision) is used by default. The range is 0 to 8, meaning zero to eight decimal places, but the precision itself varies depending on the units of the angle, as shown by the table below:

Angular Units	Range of Precisoin (AuPrec)
Decimal	0 to 0.00000000
DMS	0d to 0d00'00.0000"
Grads	0g to 0.00000000g
Radians	0r to 0.00000000r
Surveyor's units	N0dE to N0d00'00.0000"E

Formatting Angles

Angles are formatted through the **Angtos** function, short for “angle to string.” In this example, the Angtos function formats the chamfer angle, ChamferD:

```
$(angtos, ($getvar, chamferd), 2, 1)
```

Where:

(\$getvar, chamferd) — specifies the name of the system variable, the source of the angle.

2 — specifies the *format* of the angle, grads in this case. Diesel uses the same code as the AUnits (angle units) system variable. When this digit is left out, BricsCAD reads the current value in AUnits.

Mode (AUnits)	Displays Angles As
0	Decimal degrees (360.0 degrees per circle)
1	Degrees, minutes, seconds
2	Grads (400 grads per circle)
3	Radians (2pi radians per circle)
4	Surveyor's units (N and E coordinates)

1 — specifies the *precision* of the angle, one decimal place in this case. When this digit is left out, then AuPrec (angular units precision

FORMATTING TEXT

Diesel provides the most rudimentary of functions for formatting text.

Upper

Diesel includes an **Upper** function that converts the entire text string to uppercase. This useful for comparing two text strings, to ensure they are identical. There is no “Lower” function.

StrnLen

The **StrLen** function determines the number of characters in a string, while the **Substr** extracts a portion of a string. Details on these and other functions are found later in this chapter.

Other types of text formatting, such as boldface and coloring, are not available in Diesel.

VARIABLES IN DIESEL

You can use variables with Diesel functions. When you have the result of one calculation, you may wish to store it for use later on by another second calculation — kind of like using a memories on a calculator. Here is how you accomplish this:

1. First, you use the **SetVar** command to store the value in one of the *user* system variables, such as **UserR1**. (This must be done outside of the menu macro.)
Command: **setvar**
Enter variable name or [?]: **userr1**
Enter new value for USERR1 <0.0000>: **3.141**
2. Then you can access it inside the menu macro with the **\$(getvar)** function:
\$(+,\$(getvar,userr1),25)

The following user system variables can be used with Diesel:

- ▶ **UserR1** through **UserR5** to store *reals* (numbers with decimals)
- ▶ **UserI1** through **UserI5** to store *integers* (numbers without decimals)
- ▶ **UserS1** through **UserS5** to store *strings* (text)

Actually, you can store anything you want in these 15 sysvars; it's just handy that they are labelled with R (for real), I (for integer), and S (for string). Careful though: the contents of these sysvars are wiped clean when BricsCAD closes. The next time you start BricsCAD, their values are all 0.

Complete Catalog of Diesel Functions

Here are details on all Diesel functions supported by BricsCAD.

MATH FUNCTIONS

Diesel supports the four basic arithmetic functions.

+

The **+** (Addition) function adds together up to nine numbers:

\$(+,2,3.4,10,5) *returns 20.4*

The function works with as little as one value, adding the value to 0:

\$(+,2) *returns 2*

-

The **-** (Subtraction) function subtracts as many as eight numbers from a ninth. For example, the following equation should be read as $2 - 3.4 - 10 - 5 = -16.4$:

\$(-,2,3.4,10,5) *returns -16.4*

As another example, this equation should be read as $2 - 0 = 2$:

\$(-,2) *returns 2*

*

The * (Multiplication) function multiplies together up to nine numbers.

```
$(*,2,3.4,10,5)           returns 340
```

When you store the value of pi (3.141) in **UserR1**, you can perform calculations that involve circles. For instance, recall that to find the area of a circle the formula is **pi * r²**. Diesel doesn't support squares or exponents, so you need to multiple **r** by itself: **pi * r * r**.

To find the area of a 2.5"-radius circle:

```
$(*,$(getvar,userR1),2.5,2.5)  returns 19.63125
```

/

The / (Division) function divides one number by up to eight other numbers.

```
$(/,2,3.4,10,5)           returns 0.01176471
```

This one reads as $2 / 3.4 / 10 / 5 = 0.1176471$.

LOGIC FUNCTIONS

The logic functions test to see if two (or more) values are equal (or not).

=

The = (Equal) function determines if two numbers (or strings) are equal. If so, the function returns 1; if not, it returns 0.

```
$(=,2,2)                  returns 1
```

```
$(=,2,3.4)                returns 0
```

<

The < (Less than) function determines if one number is less than another. If so, the function returns 1; if not, it returns 0.

```
$(<,2,2)                   returns 0
```

```
$(<,2,3.4)                 returns 1
```

>

The > (Greater Than) function determines if one number is greater than another. If so, the function returns 1; if not, it returns 0.

```
$(>,2,2)                   returns 0
```

```
$(>,2,3.4)                 returns 1
```

!=

The != (Not Equal) function determines if one number is not equal to another. If not equal, the function returns 1; if equal, it returns 0.

```
$(!=,2,2)                  returns 0
```

```
$(!=,2,3.4)                returns 1
```

<=

The `<=` (Less Than or Equal) function determines if one number is less or equal than another. If so, the function returns 1; if not, it returns 0.

```
$(<=,2,2)           returns 1
$(<=,2,3.4)        returns 1
$(<=,9,0.5)        returns 0
```

>=

The `>=` (Greater Than or Equal) function determines if one number is greater than or equal to another. If so, the function returns 1; if not, it returns 0.

```
$(>=,2,2)           returns 1
$(>=,9,0.5)        returns 1
$(>=,2,3.4)        returns 0
```

AND

The **and** (Logical Bitwise AND) function returns the bitwise logical “AND” of two or more integers. This function operates on up to nine integers.

EQ

The **eq** (Equality) function determines if two numbers (or strings) are equal. If identical, the function returns 1; otherwise, it returns 0.

```
$(eq,2,2)           returns 1
$(eq,9,0.5)         returns 0
```

The values have to be *exactly* equal; for instance, a real number is not the same as an integer number, as the following example illustrates:

```
$(eq,2.0,2)         returns 0
```

Normally, you wouldn't test two numbers; instead, you would test a number and a value stored in a variable. For example, to check if **LUnits** is set to 4 (architectural units):

```
$(eq,$(getvar,lunits),4) returns 1 when LUnits = 4
                           returns 0 if LUnits = any other number
```

IF

The **if** function checks if two expressions are the same. If so, the function carries out the first option, and ignores the second option; if false, it carries out the second option. In generic terms:

```
$(if,test,true,false)
```

where:

test — specifies another logic function, such as `$(eq,clayer,0)`; *test* expects a value of 1 (true) or 0 (false).

true — indicates the action to take when the test is true.

false — indicates the action to take when the test is false.

For example, the following test checks to see if the current layer is not 0. If so, it then gets the name of the layer. Notice that the *true* parameter is missing.

```
$(if,$(eq,clayer,"0"),,$(getvar,clayer))
```

OR

The **or** (Logical OR) function returns the bitwise logical “OR” of two or more integers.

XOR

The **xor** (Logical Bitwise Xor) function returns the bitwise logical “XOR” (eXclusive OR) of two or more integers.

CONVERSION FUNCTION

The conversion functions change the state of numbers.

FIX

The **fix** function removes the decimal portion from real numbers, converting them to integers. This function can be used to extract the number before the decimal point from a real number. (There is no “round” function.)

```
$(fix,3.99) returns 3
```

STRING FUNCTIONS

The string functions manipulate text (and sometimes numbers).

INDEX

The **index** function extracts one element from a comma-separated series. Autodesk suggests using this function to extract the x, y, and z coordinates from variables returned by the (**\$getvar** function. In generic terms, the function looks like this:

```
$(index,item,string)
```

where:

item — a counter; starts with 0.

string — the text being searched; contains comma-separated items.

Note that the *item* counter starts with 0, instead of 1; the first item is #0:

```
$(index,0,"2,4,6") returns 2
```

String must be text surrounded by quotation marks; leave out the quotation marks, and Diesel ignores the function. The string consist of one or more items separated by commas.

Here is an example of extracting the y coordinate from the **LastPoint** system variable:

```
$(index,1,$(getvar,lastpoint)) returns 64.8721
```

(The result will differ, depending on the coordinate stored in **LastPoint**.) Use the following *item* values to extract specific coordinates:

Item	Coordinate Extracted
0	X
1	Y
2	Z

NTH

The **nth** function extracts the *nth* element from one or more items. This function handles up to eight items. Like **index**, the first item in the list is #0. In generic terms, the function looks like this:

```
$(nth,item,n1,n2,...)
```

where:

item — a counter; range is 0 to 7.

n — a list of items separated by comma; maximum of eight items in the list.

If *item* exceeds *n*, then Diesel ignores this function.

Here are examples of using the function with numbers and text:

```
$(nth,2,2.3,4.5,6.7) returns 6.7
```

```
$(nth,1,Tailoring,BricsCAD,CAD) returns BricsCAD
```

STRLEN

The **strlen** (String Length) function returns the number of characters in the string. This function is useful for finding the length of a string before applying another function, such as **substr**.

```
$(strlen,Tailoring BricsCAD) returns 18
```

If the string is surrounded by quotation marks, Diesel ignores them.

```
$(strlen,"Tailoring BricsCAD") also returns 18
```

This function also works with numbers and system variables:

```
$(strlen,3.14159) returns 7
```

```
$(strlen,$(getvar,platform)) returns 38
```

SUBSTR

The **substr** (SubString) function returns a portion of a string. This is useful for extracting text from a longer portion. Generically, the function looks like this:

```
$(substr,string,start,length)
```

where

string — specifies the text to be handled.

start — indicates the starting position of the substring; first character is #1.

length — specifies the length of the substring; optional. If left out, the entire rest of the string is returned.

Here are some examples of this function at work:

```
$(substr,Tailoring BricsCAD,5) returns oring BricsCAD
```

```
$(substr,Tailoring BricsCAD,5,7) returns oring B
```

If the string is surrounded by quotation marks, Diesel ignores them.

```
$(substr,"Tailoring BricsCAD",5) also returns oring BricsCAD
```

This function also works with numbers and system variables:

```
$(substr,3.14159,1,4) returns 3.14
```

```
$(substr,$(getvar,platform),5,15) returns osoft Windows N
```

UPPER

The **upper** (uppercase) function converts text strings to uppercase characters. (There is no “lower” function in Diesel.) It works with text and system variables, as follows:

```
$(upper,"Tailoring BricsCAD") returns TAILORING BRICSCAD
```

```
$(upper,$(getvar,platform)) returns MICROSOFT WINDOWS NT VERSION 5.0
```

The function also works with numbers, but leaves them unchanged.

SYSTEM FUNCTIONS

The system functions are a collection of miscellaneous functions.

EDTIME

The **edtime** (Evaluate Date Time) function formats the display of the system time. This function reads the date and time from the **Date** system variable, and then formats it according to your instructions. Generically, the function looks like this:

```
$(edtime,$(getvar,date),format)
```

where

format — specifies how the date and time should be displayed, as illustrated by the table below.

When *format* contains text that Diesel cannot interpret, it is displayed literally. The table shows date formatting codes for a date of September 5, 2006:

Date Formats	Meaning	Example
D	Single-digit date	5
DD	Dual-digit date	05
DDD	Three-letter day	Fri
DDDD	Full-letter day	Friday
M	Single-digit month	9
MO	Dual-digit month	09
MON	Three-letter month	Sep
MONTH	Full-letter month	September
YY	Dual-digit year	16
YYYY	Four-digit year	2016

The table below lists time formatting codes for a time of 1:51:23.702AM:

Time Formats	Meaning	Example
H	Single-digit hour	1
HH	Dual-digit hour	01
MM	Minutes	51
SS	Seconds	23
MSEC	Milliseconds	702
AM/PM	Uppercase AM or PM	AM
am/pm	Lowercase AM or PM	am
A/P	Abbreviated uppercase	A
a/p	Abbreviated lowercase	a

TIPS To use commas in the format code, surround them with "," so that Diesel does not read the comma as an argument separator.

The quotation-mark trick does not work for words like "Date" and "Month": Diesel returns *1date* and *7onth*.

The date and time codes are case-insensitive; **D** and **d** work the same. The exceptions are for the **AM/PM** and **am/pm** codes.

When the **AM/PM** and **A/P** format codes are used, Diesel displays the 12-hour clock; when they are left out, Diesel displays the 24-hour clock.

The **AM/PM** and **A/P** format codes must be entered with the slash. If, say, PM is entered, then Diesel returns P literally and reads **M** as the single-digit month code.

Here are some examples of using the **EdTime** function:

```
$(edtime,$(getvar,date),H:MMam/pm)           returns 11:58am
$(edtime,$(getvar,date),DDD", " DD-MO-YY)     returns Fri, 01-07-05
$(edtime,$(getvar,date), DDD", " d mon", " YYYY) returns Fri, 1 Jul, 2015
```

EVAL

The **eval** (Evaluate) function displays text on the status bar:

```
Command: modemacro
Enter new value for MODEMACRO, or . for none <"">: $(eval,"This is text")
```

It is equivalent to using the **ModeMacro** command without Diesel:

```
Command: modemacro
Enter new value for MODEMACRO, or . for none <"">: This is text
```

GETENV

The **getenv** (Get Environment) function gets the values stored in environment variables. This function was designed for use with AutoCAD LT, which has two commands not found in AutoCAD: **SetEnv** sets values in environment variables, and **GetEnv** reads the values. These environment variables were originally stored in a file named *aclt.ini*, but are now stored in the Windows Registry.

```
$(getenv,maxarray)           returns 10000
```

As of BricsCAD V12, the behavior of `$(getenv)` is now consistent with that of LISP and SDS/BRX: it searches for environment variables in BricsCAD environment registry; in Windows, Linux, or Mac process environment; and in BricsCAD CFG settings. The read sequence is:

1. BricsCAD Windows registry
2. Linux, Mac, or Windows process environment
3. BricsCAD configuration

The Write sequence is (a) BricsCAD configuration, if a key is present, and (b) BricsCAD Windows registry.

GETVAR

The **getvar** (Get Variable) function gets the values of system variables.

```
$(getvar, lunits)           returns 4
```

LINELEN

linelen (line length) function returns the maximum length of display.

```
$(linelen)                 returns 240
```

DIESEL PROGRAMMING TIPS

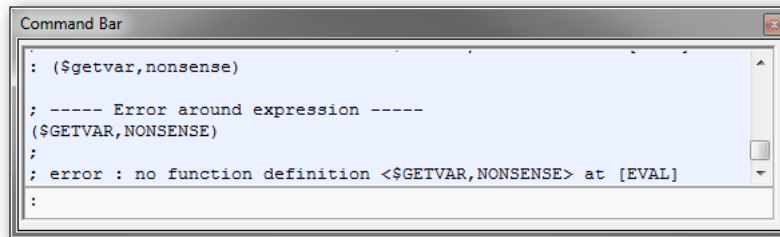
Here are some tips for working with Diesel:

- ▶ Each argument must be separated by a comma; there must be no spaces within the expression.
- ▶ The maximum length of a Diesel macro is 240 characters; the maximum display on the status bar is 32 characters.
- ▶ The **ModeMacro** system variable outputs text directly to the status bar until it reaches a `$(`, and then it begins evaluating the macro.
- ▶ Use the **MacroTrace** system variable to debug macros.
- ▶ Use LISP's (**strcat**) function to string together Diesel macros within LISP.
- ▶ Use the **\$M=** construct to use Diesel expressions in menu and toolbar macros.

Debugging Diesel

The purpose of the **MacroTrace** system variable is to help track down bugs in Diesel macros. When on, a step-by-step evaluation of the Diesel macro should be displayed in the Text window. Although MacroTrace exists in BricsCAD, it is not yet implemented.

Instead, BricsCAD displays errors directly, whether in a menu or on the command line. Below, I entered Diesel code with a non-existent sysvar, “nonsense.”



```
Command Bar
: ($getvar,nonsense)
; ----- Error around expression -----
($GETVAR, NONSENSE)
;
; error : no function definition <$GETVAR, NONSENSE> at [EVAL]
:
```

Error reported by Diesel

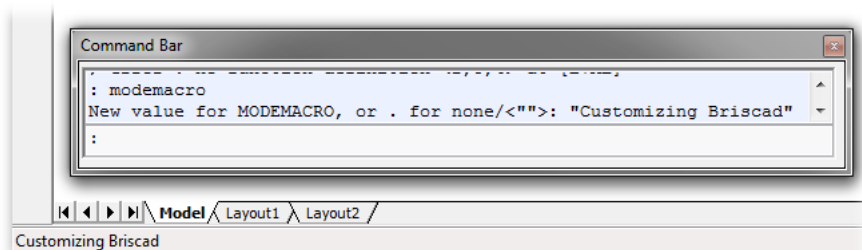
ModeMacro: Displaying Text on the Status Bar

The purpose of the ModeMacro command is to display text on the status bar.

Should BricsCAD ever get this function, then here is how to use it. First, let’s see how to display text to the status bar:

1. Enter the **ModeMacro** system variable at the ‘Command:’ prompt, and then type something:
Command: **modemacro**
New value for MODEMACRO, or . for none <"">: **Customizing BricsCAD**

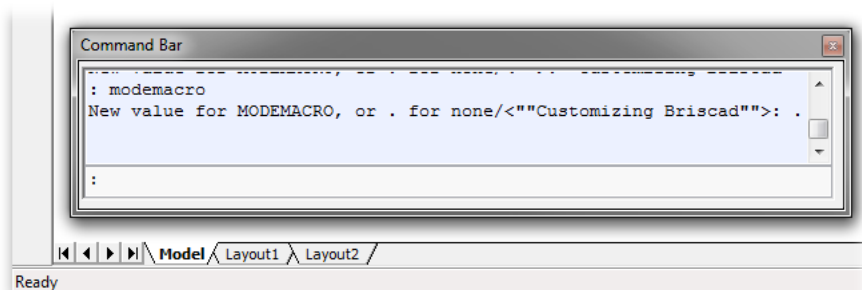
The words “Customizing BricsCAD” should appear at the far left of the status bar:



Using Diesel to display text on the status bar

(You cannot change the location where the text is positioned on the status bar.)

2. To remove the text from the status bar, type the **ModeMacro** system variable with a . (null string), as follows:
Command: **modemacro**
New value for MODEMACRO, or . for none <"Customizing AutoCAD">: .



Removing user-defined text from the status bar

Customizing Ribbon Tabs and Panels

The ribbon is a Microsoft-designed user interface that some love to hate, and others have come to like. Me, I don't care for its design inconsistencies and for the extra clicks needed to get at commands. Even though the ribbon is "unique" to Windows, Bricsys wrote a custom version so that the ribbon works identically with the Linux and MacOS versions of BricsCAD.

In this chapter you learn how to customize the tabs and panels of the ribbon.

CHAPTER SUMMARY

This chapter covers the following topics:

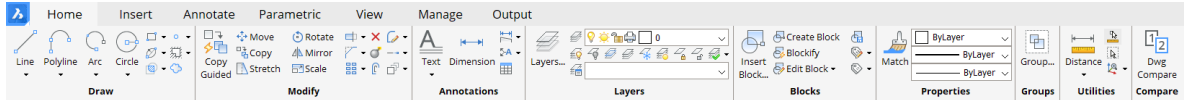
- Understanding the structure of ribbons
- Defining the look of the ribbon through workspaces
- Creating new tabs
- Adding panels to tabs
- Designing new panels

QUICK SUMMARY OF RIBBON COMMANDS AND VARIABLES

COMMANDS

Ribbon displays the ribbon.

RibbonClose closes the ribbon.



SYSTEM VARIABLES

RibbonState (read-only) reports whether the ribbon palette is open or closed:

- › 0 = ribbon is closed
- › 1 = open (default in most workspaces)

RibbonDockedHeight determines the height of the ribbon when docked:

- › 0 = ribbon sizes itself to the height of the selected tab
- › 120 = default value
- › 1 to 500 pixels = range

RibbonPanelMargin specifies the distance between buttons and the edges of the ribbon panel:

- › 0 = no spacing; default value
- › 50 = maximum value; in pixels

CleanScreenOptions determines whether the ribbon is displayed in clean screen mode:

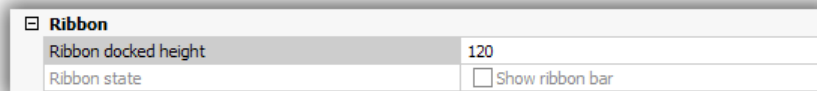
- › 8 = ribbon is not displayed

StartUp specifies whether the ribbon is displayed by the Start window:

- › 3 = display Start page without ribbon
- › 4 = display Start page with ribbon

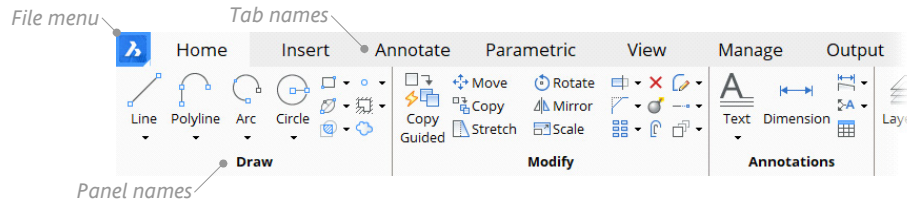
SETTINGS DIALOG BOX

The Ribbon section of the Settings dialog box holds only two of the ribbon variables:



The Structure of Ribbons

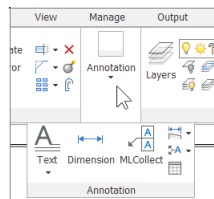
Along the top of the ribbon is a series of *tabs* with names like **Home** and **Insert**. Tabs are collections of *panels*, and panels collect similar commands. You can think of tabs as overlapping toolbars. (NEW IN v20) The **B** “tab” is not a tab, but the file menu; that’s why it’s colored blue.



The ribbon consists of tabs and panels

Panels are identified by names along the bottom of the ribbon, like **Draw** and **Modify**.

When the ribbon is too wide for the screen, panels are compacted with a slideout, as shown below.



Condensed **Annotations** panel showing all items in a slide-out panel

The purpose of subdividing a ribbon into tabs and panels is to present a logical collection of related commands. For example, many 2D drawing and editing commands are found in the Draw tab. All parametric commands are clustered in the Parametric tab.

The formal structure of a ribbon looks like the following:

Ribbon

Tab (one or more tabs)

Panel (one or more panels)

- Rows (rows are optional; multiple rows allow vertically-stacked buttons)
- Buttons and combo bars (drop lists)
- Ribbon breaks (separator lines)
- Split buttons (drop-downs, fly outs)
- Toggle buttons (change color to show on-off status)

Here it gets tricky: although tabs and panels are customized by the Customize dialog box, the *content* of ribbon you see on the screen can also be defined by the current workspace! So, when you customize the ribbon, you may have to work in two places:

Ribbon tab — defines all ribbon tabs and panels available to BricsCAD

Workspace tab — toggles the visibility tabs and panels to determine which ones are seen by users

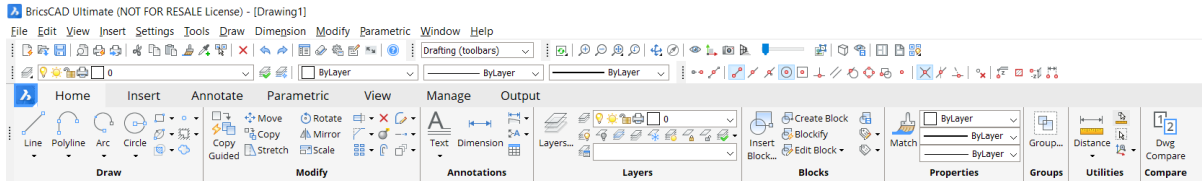
Technically, this is called “indirection.” It makes customizing ribbons more complex with the benefit of greater flexibility. It makes things easier for the you, the customizer: create one master set of tabs and panels, and then click them on and off for various workspaces.

TUTORIAL: HOW TO ADD PANELS TO RIBBON TABS

There are two ways to customize the ribbon: change the panels displayed by tabs, and change the content of panels. First, let's see how to add a panel to a ribbon tab.

1. Start BricsCAD.
2. If there is no ribbon visible, then turn it on. To display the ribbon, enter the **Ribbon** command:
: **ribbon**

Notice that the ribbon appears. If toolbars are on, then the ribbon appears below them.

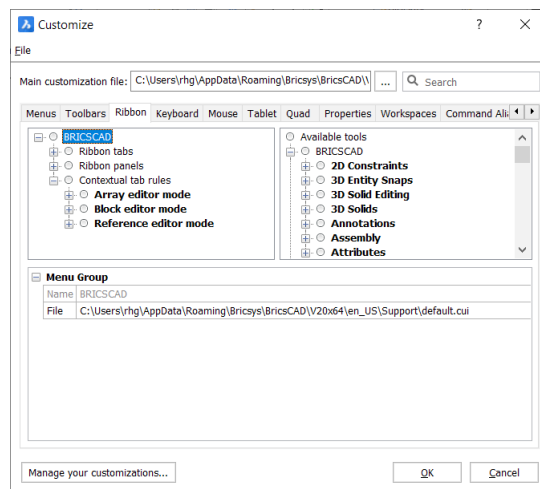


Ribbon added to the "Drafting (toolbars)" workspace


TIPS You turn off the ribbon with the **RibbonClose** command, or by clicking the small **x** at the ribbon's upper left corner.

When no drawings are open, all commands on the ribbon no longer work; they are colored gray. Use the **B** "tab" to open a drawing or start a new one.

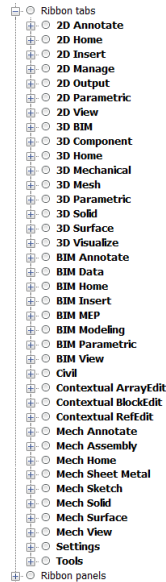
3. Open the Customize dialog box. I find typing the **cui** alias the fastest way to do this.
4. Choose the **Ribbon** tab. (It can get confusing: *tabs* in the dialog box, and *tabs* on the ribbon. To distinguish between them, I'll always write "ribbon tab" to refer to tabs in the ribbon.) Notice there are three nodes for customizing ribbons, **Ribbon Tabs**, **Ribbon Panels**, and **Contextual Tabs**:
 - ▶ **Ribbon Tabs** node — specifies which panels occupy a tab
 - ▶ **Ribbon Panels** node — customizes the content of panels with buttons
 - ▶ **(NEW TO V20) Contextual Tabs** node — tabs that display for the duration of a command



Customize dialog box open to the Ribbon tab

- Open the **Ribbon Tabs** node by clicking the  **Expand** button. Notice the long list of *tab* names, starting with “Home.” There are nearly 40 of ‘em, and you can make more.

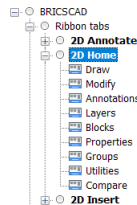
(NEW IN V20) Tabs are listed in alphabetical order; some tabs have been renamed, such as “Home 2D” to “2D Home”; the total number of tabs is nearly doubled; contextual tabs are added.



Ribbon Tabs node showing tabs provided with BricsCAD

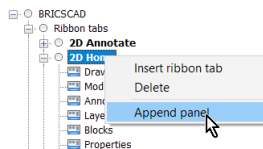
TIP Some tab names seem to have near-duplicate names, such as “2D Home” and “3D Home.” The difference is that the 3D Home tab contains commands suitable for 2D drafting, while 3D Home is meant for 3D modeling.

- Each tab on the ribbon holds one or more panels. Click the  **Expand** button next to **2D Home**. Notice the list of panel names, such as **Draw** and **Modify**.



The panels that reside in the 2D Home tab

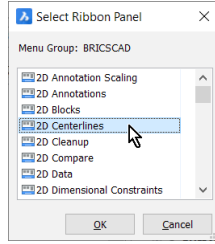
- To add a panel to a tab, follow these steps:
 - Right-click the name of a ribbon tab, such as **2D Home**.
 - From the shortcut menu, choose **Append Panel**.



Right-clicking a tab to insert a panel

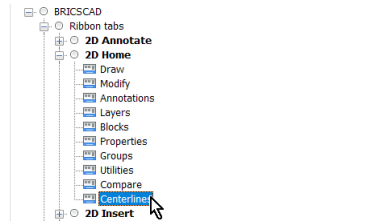
TIP You might think that you could drag an item from the **Available Tools** pane, but it doesn’t work with the tabs section. The Available Tools are meant for customizing panels.

Notice the Select Ribbon Panel dialog box. It lists the names all panels, and not just ones related to 2D.



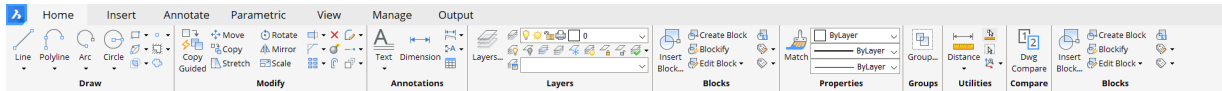
List of available panels to insert into tabs

- c. Select the name of a panel, such as “2D Centerlines,” and then click **OK** to close the dialog box. Notice that the panel is added to the end of the 2D Home list.



Centerlines panel added to the end of the “2D Home” tab

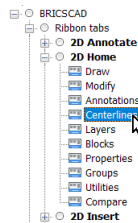
- d. To see the newly added panel in the ribbon, click **OK** to close the Customize dialog box. Notice that the new panel appears at the far end of the ribbon’s Home tab.



Centerlines panel added to the end of the Home tab

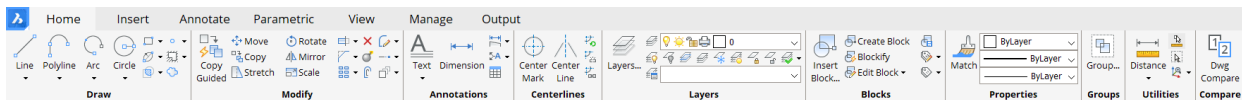
Moving Panels

You might not want to change the order in which panels appear in a tab, simply drag them around. In the figure below, I dragged the “**Centerlines**” panel up to **Annotations**. BricsCAD places it below (after) of the “Annotations” panel.



Rearranging the order by panels by dragging them around

The result of the move is shown in the ribbon illustrated below:



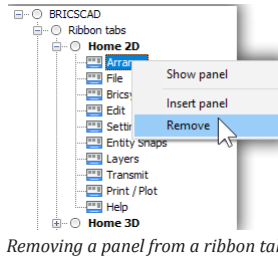
Centerlines panel moved next to Annotations

Copying Panels — Not

Making a duplicate of a panel is a great way to make a new one, without starting from scratch. In previous releases, we could make a duplicate by holding down the **Ctrl** key (**Cmd** on Mac) while dragging. In V20, this appears to no longer work.

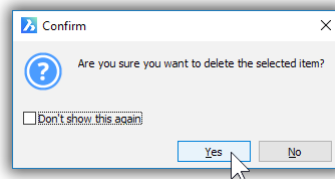
Removing Panels

To remove a panel, right-click its name, and then choose **Remove** from the shortcut menu.



Removing a panel from a ribbon tab

BricsCAD asks if you are sure; click **Yes**.



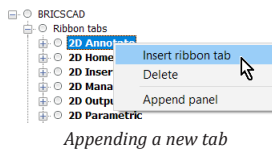
Confirming the removal

TIP To reset the UI to the fresh-out-of-the-box look, open the Customize dialog box, click the **Manage Your Customizations** button, and then click the **Reset to Defaults** button.

TUTORIAL: MAKING NEW TABS

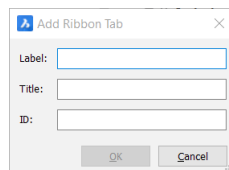
You have seen how to modify the look of a tab by adding, moving, and removing panels. In this tutorial you learn how to create a new tab from scratch.

1. Open the Customize dialog box, and then go to the **Ribbon** tab.
2. Open the **Ribbon Tabs** section, and then right-click any tab name. Notice the shortcut menu:



Appending a new tab

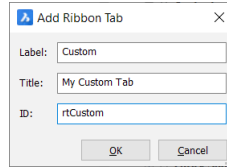
3. Choose **Insert Ribbon Tab**. Notice that BricsCAD opens the Add Ribbon Tab dialog box.



Naming the new tab

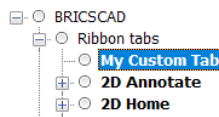
- Fill out the fields with something unique, such as with this data:

Field	Input	Meaning
Label	Custom	Label displayed by the tab on the ribbon
Title	My Custom Tab	Title shown in the Customize dialog box
ID	rtCustom	Identification used by BricsCAD to distinguish between elements; "rt" is short for ribbon tab and identifies the purpose of the ID



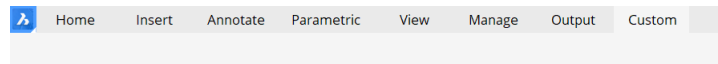
Dialog box filled out

- Click **OK** to close the dialog box. Notice that the new tab is added to the start of the list of tab names. You can drag it to another location, if you wish.



New tab added to the ribbon

- At this point, the tab is empty. Exit the Customize dialog box, and notice that BricsCAD display its empty.

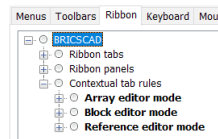


Empty tab

(NEW IN V20). You now perform these adding and moving tasks in the Ribbon tab. Prior to BricsCAD V20, these tasks had to be done in the Workspaces tab, including making the new tab visible.

QUICK SUMMARY OF CONTEXTUAL TABS

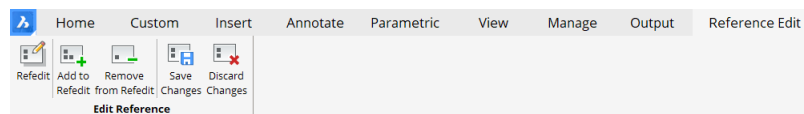
(NEW TO V20) Contextual tabs display automatically when a specific command is invoked, and then stay on the screen for the duration of a command. BricsCAD provides three of them, at time of writing:



Array editor – initiated by the ArrayEdit command

Block editor – initiated by the BEdit command

Reference editor – initiated by the RefEdit command (see below)



When the command ends, the tab disappears. Contextual tabs appear to be hard-coded into BricsCAD, and so you cannot create additional ones.

Adding Panels to A New Ribbon Tab

To add panels to the new, empty ribbon tab, review the earlier tutorial. In brief, right-click the tab's name, and then choose **Append Panel**.

Moving Tabs Along the Ribbon

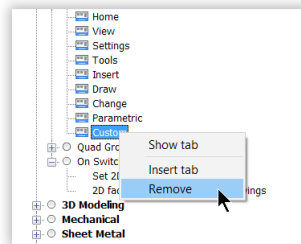
To move a ribbon tab to a different location on the ribbon, just drag its name to the new spot in the **Workspaces** tab.

Making Copies of Tabs

When you hold down the **Ctrl** key (Cmd on Mac) while dragging the name of a ribbon tab, BricsCAD makes a copy of the tab.

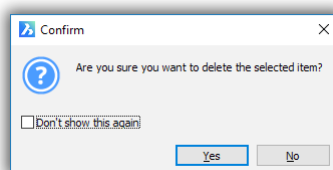
Hiding Tabs in a Workspace

To hide a ribbon tab, you need to switch to the Workspace tab. Right-click its name in, and then choose **Remove**.



Hiding a tab by removing it from the workspace

Ignore the warning message: the Remove action does not erase the tab, but merely removes it from view in this specific workspace. To actually erase a ribbon tab from BricsCAD, you need to remove it in the Ribbon tab of this dialog box.



False warning message

CUSTOMIZING RIBBON PANELS

Now that you know how to customize and create ribbon tabs, let's move on to a more complex task: customizing the content of panels. It is at the panel level where the real work of customizing ribbons takes place!

BricsCAD boasts the many panels, and you can make your own. You can change the content of existing panels, add new panels, or erase them. Panels hold many kinds of elements, such as sub-panels, rows, buttons, and other controls.

QUICK SUMMARY OF PANEL PARAMETERS

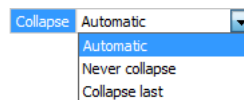
When you select the name of a panel in the Ribbon Tabs section of the Customize dialog box, BricsCAD displays the following parameters:

Ribbon Panel Reference	
ID	rpFile2D
Collapse	Automatic
Label	File
Title	Home - File 2D
Key Tip	FL

This is the meaning of the parameters:

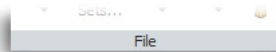
ID — identification used by BricsCAD for this user interface element. It must be unique, and should not be changed for elements that ship with the software. In this case, “rp” is short for *ribbon panel*.

Collapse — controls how to make panels smaller when the tab is wider than the BricsCAD window. Choose one of the options:



- ▶ **Automatic** lets BricsCAD decide when to collapse the panel; default setting for all panels
- ▶ **Never Collapse** keeps the panel full size, but cuts off buttons when BricsCAD window becomes too narrow
- ▶ **Collapse Last** causes other panels to collapse first

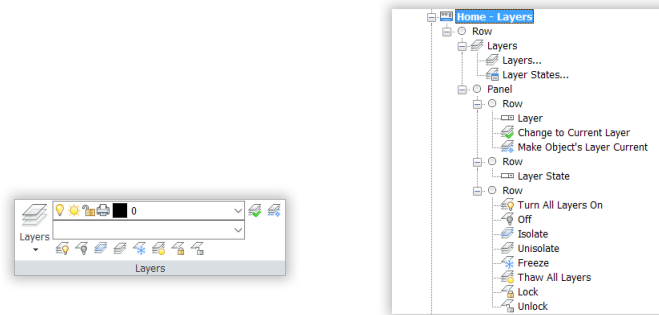
Label — name that appears as the panel name on the ribbon. In this case, “File” appears:



Title — title of the panel

Key Tip — shortcut that accesses the panel from the keyboard (not yet implemented in BricsCAD)

Below I show the **Layers** ribbon panel along side elements that make up the panel in the Customize dialog box.



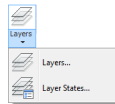
Left: Layers panel; **right:** elements of the panel listed in the Customize dialog box

TIP To easily see a panel's definition from its tab, right-click the panel name and then choose **Show Panel**. BricsCAD jumps to the panel's definition.

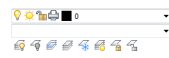
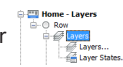
Notice that each panel definition begins with a **Row** element. It is followed by any other element, such as one or more buttons or more rows.

Panel Design Tips

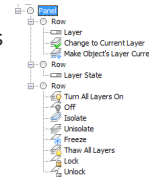
Here are a couple of tips for designing panels:



Flyouts are defined by a **Large** button assigned **Split** behavior



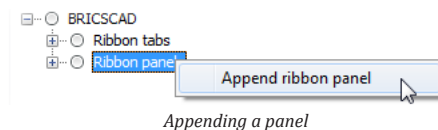
Three rows are define by a sub-panel so that their elements are positioned adjacent to the large Layers flyout button



TUTORIAL: POPULATING A NEW PANEL

You learned in an earlier tutorial how to create a new panel. Now it is time to fill it up (populate it) with buttons and other elements. Open the Customize dialog box, and click the **Ribbons** tab. You are working with the “Ribbon Panels” node, as follows:

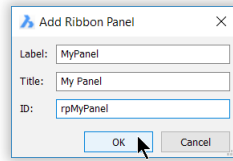
1. Right click **Ribbon Panels**, and choose **Append Ribbon Panel**.



Appending a panel

Recall that “Append” means the panel will be added to the end of the list.

2. Notice the dialog box, and that it looks like the one for making new ribbon tabs: this one is specific to panels.



Naming and ID'ing the new panel

Fill in the fields as shown below, and then click **OK**.

Field	Value	Meaning
Label	MyPanel	Identifies the panel inside the Customize dialog box
Title	My Panel	Labels the panel for the user on the ribbon
ID	rpMyPanel	Identifies the panel to BricsCAD; "rp" is short for ribbon panel

Notice that the new panel is added to the end of the list of panels. If you were to exit Customize now, you would see that it is blank, as illustrated below.

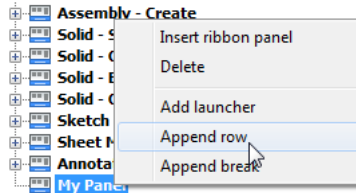


New panels are empty

3. The very first thing you do with a new panel is to add a row. Rows hold buttons and other UI elements in a horizontal row. (To create a vertical column of elements, you would append two or more rows; BricsCAD stacks them automatically.)

To add a row, follow these steps:

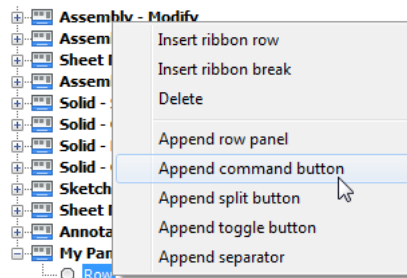
- a. Right-click the name of the new panel.
- b. Choose **Append Row** from the shortcut menu.



Adding a row to the new panel

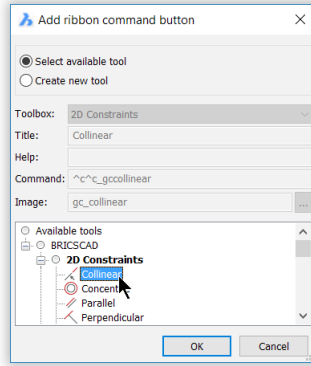
4. Now fill the row with one or more buttons. First add a regular button, which BricsCAD calls a "command button." (Later you tackle the other buttons.) This is how it works:

- a. Right-click **Row**, and then choose **Append Command Button** from the shortcut menu.



Adding a button to the row

- b. Notice the Add Ribbon Command Button dialog box. Ensure that **Select Available Tool** is selected.



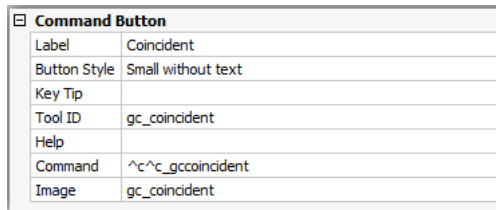
Selecting a tool from the available ones

- c. Choose a command, such as **Coincident**, and then click **OK**. Notice that it is added to the My Panel panel.



Tool (command button) added to the new row

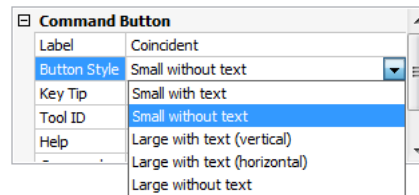
(If you were to check the ribbon back in BricsCAD, the panel would look like this the figure below.)



Left: Panel with single command button; right: Editing the parameters that define the command button

In the parameters pane at the bottom of the Customize dialog box, there are fields that define the button. Several of them should be already familiar to you from the chapters on customizing tool bars and menus, such as Help, Command, and Image. See figure above, at right.

Of specific interest to ribbon design is the **Button Style** field. It provides the following options:

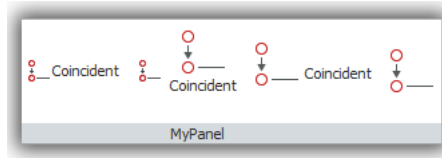


Styling the button

Here is what the options mean:

Button Style	Icon Size	Text Label	Illustration
Small With Text	16x16 pixels	Beside the icon	
Small Without Text	16x16 pixels	No label	
Large with Text (Vertical)	32x32 pixels	Below the icon	
Large with Text (Horizontal)	32x32 pixels	Beside the icon	
Large Without Text	32x32 pixels	No label	

This is what the panel looks like with buttons made from each setting:



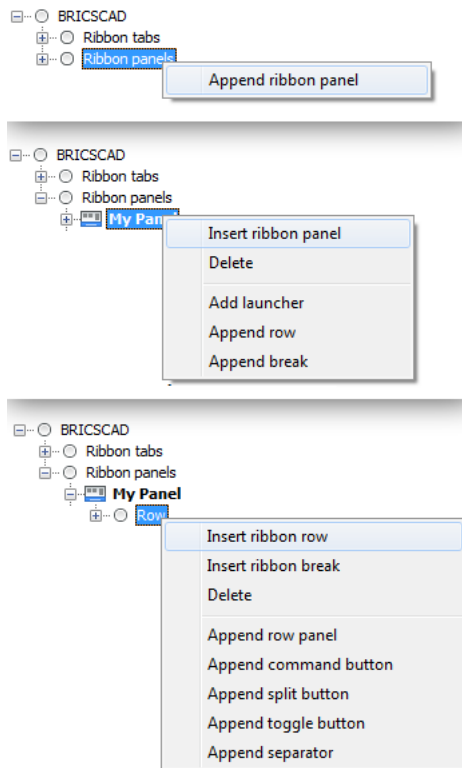
Same button displayed with different styles

With the basics of panel design accomplished, for the remainder of this chapter, I catalog all panel functions.

CATALOG OF PANEL ELEMENTS

Inserting and removing elements from panels is accomplished through shortcut menus, which are accessed by right-clicking existing elements. The sole exception is moving elements around, which is done through drag'n drop.

These are the three shortcut menus that contain the commands; the menu shown for "Row" is also the one accessed from all other elements, such as Panel and button.



Shortcut menus for editing panels

In the following sections, I describe the functions of each option grouped as follows:

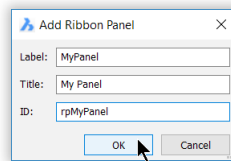
- › Append Ribbon Panel / Insert Ribbon Panel
- › Delete
- › Add Launcher
- › Append Row / Insert Ribbon Row / Insert Row Panel
- › Append Break / Insert Ribbon Break / Append Separator
- › Append Split Button
- › Append Toggle Button

Append Ribbon Panel / Insert Ribbon Panel

The **Append** and **Insert Ribbon Panel** options both add a new, blank panel to the list of Ribbon Panels. The difference between them is subtle:

- › **Append Ribbon Panel** adds the new panel to the top of the list
- › **Insert Ribbon Panel** adds the new panel to the end of the list

If it ends up in the wrong place, just drag the panel name to the proper location. Both options prompt you to fill out the fields in the same dialog box:



Labeling a panel

The ID should start with “rp” to identify it as a ribbon panel, and the name must contain no spaces.

PANEL PROPERTIES

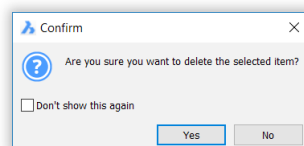
Should you need to, you can modify the names in the properties pane, except for the ID, which is fixed permanently — unless you erase the panel. (The Key Tip property does not function, yet.)

Ribbon Panel	
ID	rpidentification
Label	Name that appears in the ribbon
Title	Name that appears in the Customize dlg box
Key Tip	

Properties of a panel

Delete

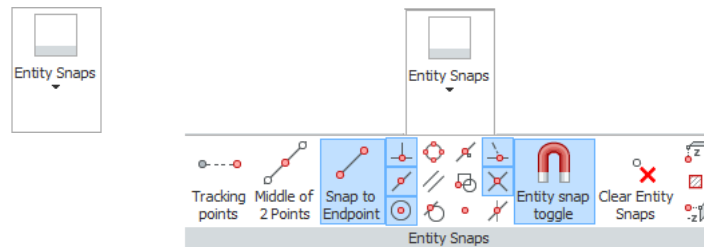
The **Delete** option erases the selected element. BricsCAD asks if you are sure:



Last chance before erasing it

Add Launcher

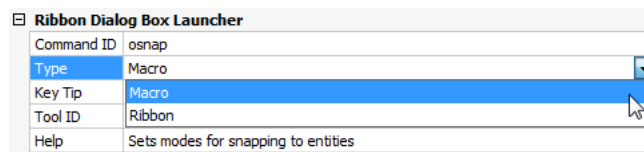
A launcher is a small panel with a ▾ flyout button. When you click the flyout button, the panel expands, as shown below. This is useful for tabs that are really wide (keeping their size in check) or for panels that contain rarely used commands.



Left: Closed launcher; right: opened launcher

TYPE PROPERTY

Launchers have just one unique property. **Type** toggles it between “Macro” and “Ribbon.”



Type options

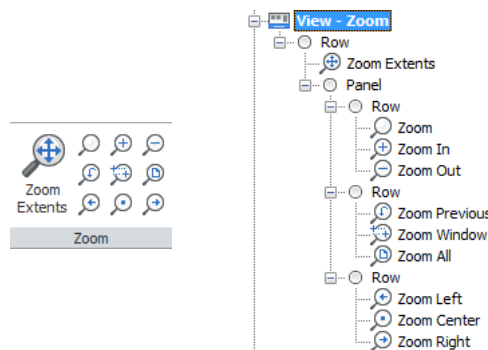
At time of writing, however, they have no effect on the launcher.

Append Row / Insert Ribbon Row / Insert Row Panel

Rows and row panels are meant to group elements within panels. The difference between the three options are as follows:

- ▶ **Append Row** — adds a row to the panel; a row holds one or more buttons horizontally
- ▶ **Insert Ribbon Row** — also adds a row to the panel; there seems to be no difference from Append Row option
- ▶ **Insert Row Panel** — adds a sub-panel to the panel; a panel holds one or more rows vertically

A common use of rows and row panels is to locate a group of smaller buttons adjacent to a large one, as illustrated below.



Left: Matrix of zoom buttons in panel; right: how they are defined in the Customize dialog box

To get the nine buttons adjacent to the one big Zoom Extents button, one Panel and three Row elements were used:

- ▶ (Row) **Panel** — segregates the three rows from the big button
- ▶ **Row** (x3) — creates three rows of horizontal buttons, stacked vertically

Rows have no properties; row panels have the following properties, none of which work at this time.

Ribbon Row Panel	
Resize Style	Automatic
Resize Priority	100
Justify Top	Yes

Properties of rows

ROW PANEL PROPERTIES

The **Resize Style** property determines what happens to the row panel when the ribbon is too small for the width of the BricsCAD window. However, none of these have an effect at the time of writing.

Row Panel	Options	Meaning
Resize Style	<ul style="list-style-type: none"> • Automatic • Never hide text • Never wrap • Never shrink • Do not resize 	<p>Lets BricsCAD handle the re-sizing on its own terms</p> <p>Eliminates icons before eliminating icons</p> <p>Prevents panel from wrapping, splitting into two or more rows</p> <p>Prevents panel from being made smaller</p> <p>Prevents panel from changing its size</p>
Resize Priority	100 (default)	Determines whether other panels should resize before this one Range is 1 (resizes first) to 1000 (resizes last)
Justify Top	<ul style="list-style-type: none"> • Yes • No 	<p>Justifies row panels to the top of the row</p> <p>Centers the row panels</p>

TIP BricsCAD normally stacks ribbon elements vertically, and the Row Panel element aligns them horizontally. (There is no “column” element.) You can use row panels to create rows within rows, or as columns (a stack of buttons) next to rows.

Append Break / Insert Ribbon Break / Append Separator

Breaks split a panel into two, so that the second half slides out when clicked. BricsCAD, however, does not support breaks. If you were to append a break, the contents of the panel would disappear, so avoid using this element until Bricsys implements it!

The figures below show before and after appending a break to the Home-File 2D panel.



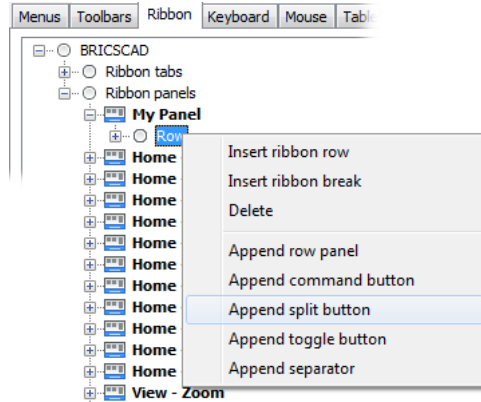
Left: Before....; right: ...and after applying the faulty Break parameter

Separators draw lines between elements in panels. BricsCAD, however, does not support breaks at time of writing.

Append Split Button

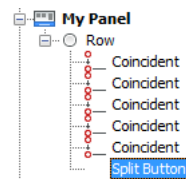
To create a flyout-like effect on the ribbon, you take two steps: (a) append a split button, and then (b) specify how it works with the Behavior property. To add a flyout to the panel, follow these steps:

1. In the panel you are designing, right-click a row and then choose **Append Split Button** from the shortcut menu. For this tutorial, the panel is named “My Panel,” as illustrated below.




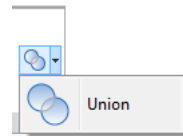
Adding a split button to a panel

2. Notice that Split Button is added to the row:



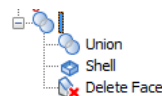
New split button

Assign a command by right-clicking **Split Button** and then choosing **Append Command Button** from the shortcut menu. From the Add Ribbon Command Button dialog box, select any command and then click **OK**. The result is a small button with the flyout icon to the right —  — the small black arrow. If you were to click it, you would see the button repeated on the flout.



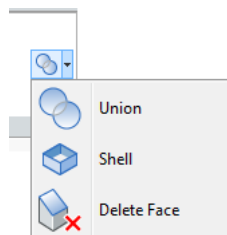
Single split button

3. Add one or two more buttons to the split so that it looks something like this:



More buttons added to the split

On the ribbon, the effect is as follows:



Multiple split buttons

4. With the split button in place, it's now time to adjust its look. Split buttons have several unique parameters of interest to you:

Split Button	
Label	
Button Style	Small without text
Behavior	Split with recent
List Style	Icon with text
Grouping	No

Parameters for split buttons

- ▶ **Behavior** — determines what happens with the topmost button
- ▶ **List Style** — specifies the look of buttons in the dropdown
- ▶ **Grouping** — gathers buttons into groups

Let's take a look at how they affect split buttons.

BEHAVIOR PROPERTY

The **Behavior** property of split buttons determines how the topmost button behaves when users click on it. Here are the options:

Split Button	
Label	
Button Style	Large with text (vertical)
Behavior	Drop down
List Style	Drop down
Grouping	Drop down with recent

Parameters for Behavior property

The options determine whether the button displays the default command (the first one in the list of buttons), or the *most recently used* one (abbreviated as “MRU” by the programming biz). As well, Behavior determines whether the element looks like a drop down (like a flyout) or like a split button (shows two buttons at once).

TIPS Split buttons let you click the upper half to execute the most-recently used command, or lower half to display the drop-list (flyout).

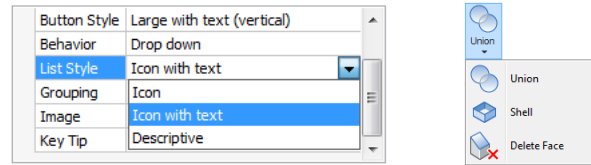
It is usual to use make split buttons large ones, so that they are easier for users to manipulate./

At time of writing, the Behavior parameter was not implemented; the only behavior that works is “Drop Down with Recent,” no matter which one you choose. If Behavior were to work, then these are the behavior options:

Behavior	Displays
Drop Down	Default command (first one in the list)
Drop Down with Recent	MRU (most recently used) command
Split	Default command
Split with Recent	MRU command (default option)
Split with Recent (static text)	Above line: Icon of MRU command Below line: default command

LIST STYLE PROPERTY

The List Style property determines the look of buttons in drop-downs.



Left: parameters for List Style property; right: how they appear in the ribbon

At time of writing, the List Style parameter was not implemented; the only style that works is “Icons with Text” no matter which one you choose. If it were to work, then these would be the behavior options:

List Style	Displays
Icon	Only icons
Icon with Text	Icons with text (default)
Descriptive	Icons with boldface text

TIP If you really need to cram in buttons, which I do not recommend, then use the **Icon** option, as this option takes up the least space.

GROUPING PROPERTY

The Grouping property gathers buttons in split lists into groups. Grouping works with the Group Name property, which defines the groups by name, but it was not implemented in BricsCAD at time of writing.

The Grouping two options are No (default) and Yes:

Grouping	Displays
Yes	Buttons in drop-downs are grouped by their assigned group name
No	Buttons are listed in the order in which they appear in the Customize dialog box

Append Toggle Button

Toggle buttons display a blue background when on, and a normal background when off. They are meant to provide a visual indication of the on-off status of a setting, as shown below with the Entity Snaps panel.



Toggle buttons appearing blue when turned on

The catch is that the Toggle button itself doesn't know how to handle the on-off status. It turns out that a toggle-style button adds a parameter for entering Diesel code, as highlighted below.

Toggle Button	
Label	Snap to Endpoint
Diesel	<code>\$(if,\$(=,\$(and,\$(getvar,OSMODE),0x0001),0),,!.)</code>
Button Style	Large with text (vertical)
Key Tip	
Tool ID	osnap_endpoint

Diesel code needed to toggle buttons

DIESEL PROPERTY

BricsCAD monitors the Diesel code to see whether to turn the blue background on. This is exactly the same situation as with menu macros, in which you use Diesel to turn check marks on and off. Here is the code for one of the entity snap toggles on the ribbon:

```
$(if,$(=,$(and,$(getvar,OSMODE),0x0001),0),,!.)
```

The good news is that you can copy and paste this code; all you need to do is replaced "OsMode" with the name of another variable.

Customizing Keystroke Shortcuts, Aliases, & Shell Commands

Power users know that the keyboard is the fastest way to enter commands. BricsCAD has several ways to use the keyboard efficiently, among them keyboard shortcuts and aliases. These let you carry out commands by simply pressing assigned keys on the keyboard — often just one or two. Both facilities are handled by the Customize dialog box.

CHAPTER SUMMARY

This chapter covers the following topics:

- Understanding, editing, and deleting keyboard shortcuts
- Learning how keystrokes differ in Mac from Windows/Linux
- Listing all keystroke shortcuts
- Assigning multiple commands
- Learning about command aliases
- Editing and deleting aliases
- Applying alias rules
- Writing shell commands

QUICK SUMMARY OF SHORTCUT KEYSTROKES

BricsCAD uses these shortcut keystrokes, most of which can be customized through the Customize dialog box. On Mac computers, use **Cmd** instead of **Ctrl**.

FUNCTION KEYS

Shortcut	Commands	Meaning
F1	Help	Displays the Help dialog box
F2	TextScr, GraphScr	Toggles between Text and Graphics windows
Shift+F2	CliState	Toggles the command bar
Ctrl+F2	Ribbonstate	Toggles the ribbon
F3	Osnap T	Toggles object snap mode
Shift+F3	StatBar	Toggles the status bar
F4	Tablet T	Toggles tablet mode
Shift+F4	ScrollBar	Toggles the scroll bars
Ctrl+F4	WClose	Closes the current drawing; function provided by Windows
Alt+F4	Quit	Closes drawings and BricsCAD; function provided by Windows
F5	Isoplane	Cycles through isoplanes
F6	Coordinate T	Cycles through coordinate display modes
Ctrl+F6	...	Switches to the next drawing; function provided by Windows
F7	Grid T	Toggles the display of the grid
F8	Orthogonal T	Toggles orthogonal mode
Shift+F8	VbaMan	Displays VBA Manager dialog box
Alt+F8	VbaRun	Displays Run BricsCAD VBA Macro dialog box
F9	Snap T	Toggles snap mode
F10	SnapType	Toggles polar tracking
F11	PolarMode	Toggles object snap tracking
Shift+F11	AddInMan	Displays the Add-in Manager dialog box
Alt+F11	VBA	Opens the Visual Basic Editor
F12	QuadDisplay	Toggles the Quad cursor; cannot be redefined with Customize
Ctrl_F12	...	Toggles sub-entity selection mode; cannot be redefined with Customize

CONTROL KEYS

On MacOS computers, press **Cmd** instead of **Ctrl**.

Shortcut	Command	Meaning
Ctrl+1	Properties PropertiesOff	Toggles Properties panel
Ctrl+2	Explorer	Displays Drawing Explorer
Ctrl+9	CommandLine CommandLineHide	Toggles command bar
Ctrl+0	CleanScreenOn CleanScreenOff	Toggles clean screen mode
Ctrl+A	SelGrips All	Selects all non-frozen objects
Ctrl+B	Snap T	Toggles snap mode
Ctrl+C	CopyClip	Copies selected objects to Clipboard

...continued

These are the differences between *keyboard shortcuts* and *aliases*:

- ▶ **Keystroke shortcuts** are like CTRL+C, ALT-TAB, and CTRL+V that to copy objects to the Clipboard, switch to another application, and then paste them, respectively. You hold down the **CTRL** key, and then press **C**. BricsCAD has many other keyboard shortcuts for its commands, and it lets you create your own. Once you've memorized even a few, they let you work at top speed. The **Keyboard** tab assigns shortcuts to function keys, **CTRL**, **ALT**, **SHIFT**, and/or arrow key combinations.
- ▶ **Aliases** are abbreviations for command names, such as L for the Line command or AA for Area. So that you don't have to type full command names each time, you can create more aliases in the Customize dialog box's Aliases tab.

...continued

Ctrl+Shift+C	CopyBase	Copies selected objects with base point
Ctrl+E	Isoplane	Switches to next isoplane
Ctrl+F	-Osnap T	Toggles entity snap mode
Ctrl+G	Grid T	Toggles display of the grid
Ctrl+H	PickStyle	Toggles pick style
Ctrl+I	Coords	Cycles through coordinate display modes
Ctrl+J	;	Repeats the last command
Ctrl+K	Hyperlink	Displays Hyperlink dialog box
Ctrl+Shift+L	LookFrom	Toggles look-from viewpoint gadget
Ctrl+L	Orthogonal T	Toggles orthographic mode
Ctrl+M	;	Repeats the last command
Ctrl+N	New	Displays the New Drawing dialog box
Ctrl+O	Open	Displays the Open Drawing dialog box
Ctrl+P	Print	Displays the Print dialog box
Ctrl+Shift+P	OpmState	Toggles the Properties panel
Ctrl+Q	Quit	Closes drawings and BricsCAD
Ctrl+R	^V	Cycles through viewports
Ctrl+S	QSave	Saves the current drawing
Ctrl+Shift+S	SaveAs	Displays the Save Drawing As dialog box
Ctrl+T	Tablet T	Toggles tablet mode
Ctrl+V	PasteClip	Pastes entities from Clipboard
Ctrl+Shift+V	PasteBlock	Pastes entities from Clipboard as a block
Ctrl+Alt+V	PasteSpec	Displays the Paste Special dialog box
Ctrl+X	CutClip	Cuts selected entities to Clipboard.
Ctrl+Y	Redo	Redoes the last undo
Ctrl+Z	U	Undoes the last command

OTHER KEYS

Del	Erase	Erases the selected entities
Enter		Executes command, repeats command, executes default option
Esc		Cancels the current command
Home		Resets the 3D view to home view

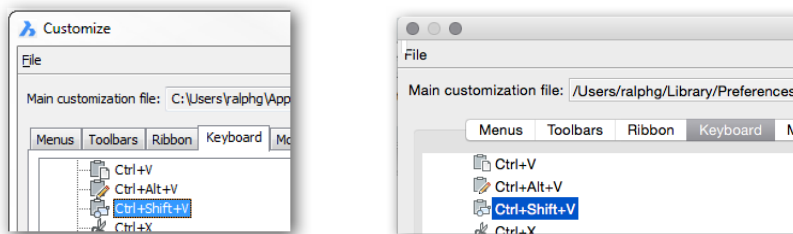
Out-of-the-box, BricsCAD defines the shortcut keystrokes listed in the boxed text on the previous pages. Regular keys (like A and 1) can be attached to the following special keys:

- ▶ **Function keys** — those marked with the **F** prefix, such as **F1** and **F2**
- ▶ **Shift keys** — hold down the **SHIFT** key, and then press a function, number, or alphabet key, such as **F2** or **B**
- ▶ **Alternate (Option) keys** — hold down the **ALT** key, and then press another key; on Macs, hold down **OPTIONS** key
- ▶ **Control (Command) keys** — hold down the **CTRL** key, and press another key; on Macs, hold down **CMD** key
- ▶ **Shift + Control keys** — hold down the **SHIFT** and **CTRL** (or **CMD**) keys, and press another key
- ▶ **Shift + Alternate keys** — hold down both the **SHIFT** and **ALT** (or **OPT**) keys, and then press another key
- ▶ **Control + Alternate keys** — hold down both the **CTRL** (or **CMD**) and **ALT** (or **OPT**) keys, and press another key
- ▶ **Control + Alternate + Shift keys** — hold down the **CTRL** (or **CMD**) and **ALT** (or **OPT**) and **SHIFT** keys all at the same time, and then press another key

TIP It does not matter if you press **SHIFT** first or **CTRL (CMD)** first — similarly for **ALT (OPT)**.

You can add and change definitions by assigning commands to as many as 188 key combinations.

While Macs use **Cmd/Options** instead of the **Ctrl/Alt** used by Windows and Linux, the Customize dialog box displays **Ctrl** and **Alt** for all three operating systems. For instance, “Paste as Block” is shown as Ctrl+Alt+V in the Windows, Linux, and Mac versions of the Customize dialog box, but is used in the Mac version by pressing Command+Options+V.



Left: Keyboard shortcuts in Windows; right: same shortcuts in Mac version of Customize dialog box

The **Control** key on Mac keyboards cannot be used with BricsCAD.

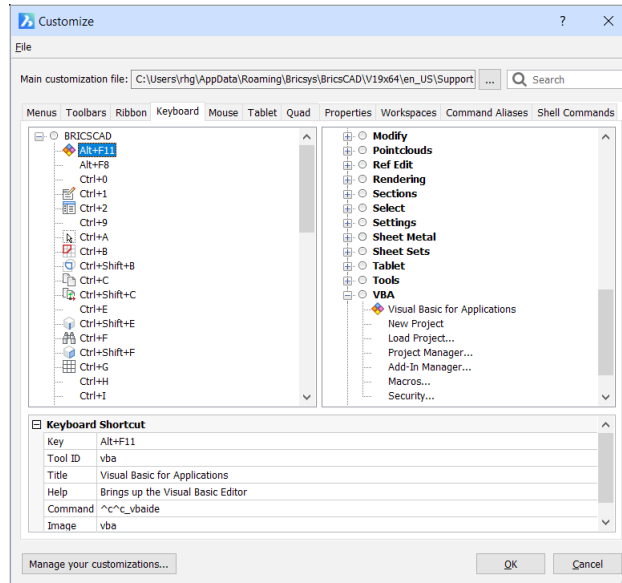
Here is a mapping table between the keys on the three operating systems:

<u>Windows, Linux</u>	<u>Mac Equivalent</u>	<u>Mac Symbol</u>
Alt (Alternative)	Option	⌥
Ctrl (Control)	Cmd (Command)	⌘
...	Control	⌞
F (Function)	F (Function)	...
Shift	Shift	⇧

TUTORIAL: DEFINING SHORTCUT KEYS

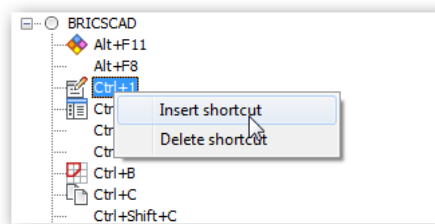
In this tutorial, you assign the Fillet command to **CTRL+SHIFT+F**. Here are the steps to defining this shortcut keystroke:

1. Enter the **Customize** command. (Alternatively, right-click any toolbar or ribbon, and then choose **Customize** from the shortcut menu, or enter the **CUI** alias.) Notice the Customize dialog box.
2. Choose the **Keyboard** tab. Notice that it consists of three panes:



Customize dialog box displaying the Keyboard tab

- ▶ **Shortcuts** pane (at the left) — lists the keyboard shortcuts that are currently assigned
 - ▶ **Available Tools** pane (at the right) — lists all of BricsCAD's commands, sorted by menu name
 - ▶ **Keyboard Shortcut** pane (at the bottom) — for editing shortcut settings
3. To define a new shortcut, right-click any item in the Shortcuts area, and then choose **Insert Shortcut**.



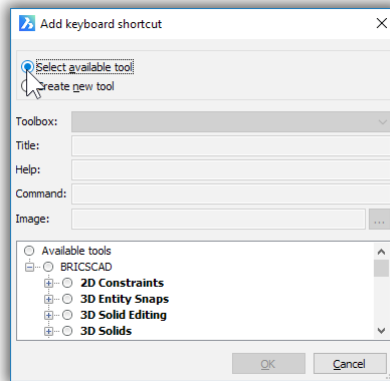
Right-clicking to access shortcut menu

TIP Do not change keystrokes reserved by Windows, such as these ones:

ALT+F4	Exits BricsCAD
CTRL+F4	Closes the current window
CTRL+F6	Changes focus to the next window
F1	Displays help

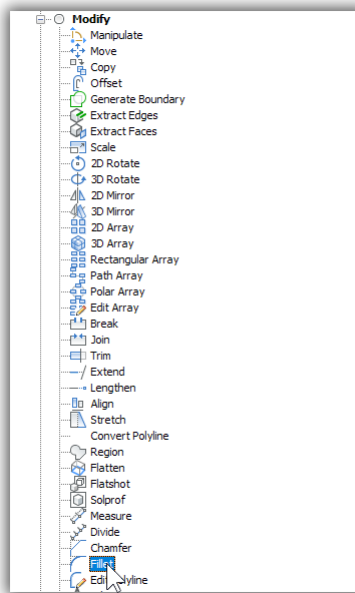
Notice that the Add Keyboard Shortcut dialog box appears, and that it offers these options:

- ▶ **Select available tool** — for choosing an existing tool (aka command) from BricsCAD’s list
- ▶ **Create a new tool** — for creating new tools (using macros) from scratch.



Dialog box for choosing the “tool” (command) to add to the shortcut keystroke

4. In this tutorial, you work with an existing tool, and in this case the “available tool” is the Fillet command. There is, unfortunately, no quick way to locate the command, as they are not listed alphabetically.



Arriving at the Fillet command

So, here is how to locate the command:

- a. Ensure the **Select Available Tool** option is selected.
- b. The Fillet command is a modification command, so you’ll find it under **Modify**. In the list of Available Tools, open **Modify** by clicking the + sign next to it.
- c. Scroll down until you come across **Fillet**.

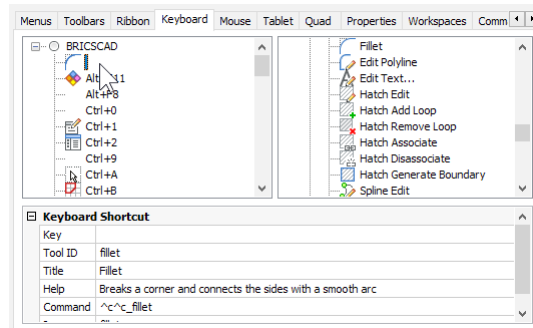
TIP As a faster alternative, press the ‘f’ key on the keyboard. This causes the highlight to jump to Flatten. Keeping pressing ‘f’ until the cursor jumps to Fillet.

- d. Click **OK** to accept Fillet.

- Back in the Customize dialog box, notice that much of the data is filled in for you, such as the help string and command macro. The macro looks like this:

`^c^c_fillet`

(Read chapter 8 to learn more about macros.)

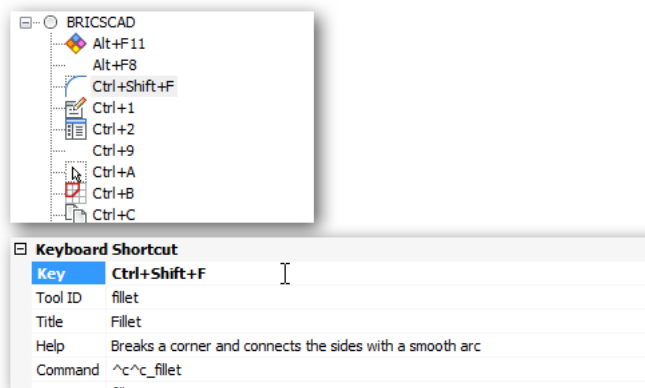


Fillet added, but not yet assigned a keystroke

All that is missing is the desired keystroke. You add it by pressing the desired keys on the keyboard, like this:

- Click the blank field next to **Key**.
- Press the key combination on the keyboard, whether Windows, Linux, or Mac: **CTRL+SHIFT+F**
 - ▶ In Windows and Linux: Hold down the **Ctrl** and **Shift** keys, and then press **F**.
 - ▶ In MacOS: Hold down the **Cmd** and **Shift** keys, and then press **F**.

Then let go of the three keys. Notice that the shortcut is added to the Shortcut list.



Pressing the keystrokes to assign to the tool

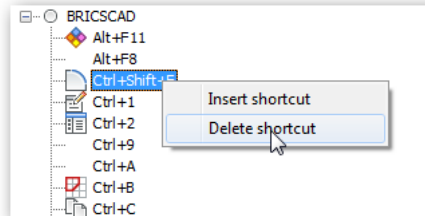
- Click **OK** to dismiss the Customize dialog box and save your work.
- Test the keystroke shortcut by holding down the **CTRL** (**CMD** on Macs) and **SHIFT** keys, and then pressing **F**. BricsCAD should execute the **Fillet** command.

TIP You can assign one or more keystroke shortcuts per command. But once a keystroke is assigned, it cannot be used for other commands.

TUTORIAL: EDITING & DELETING KEYBOARD SHORTCUTS

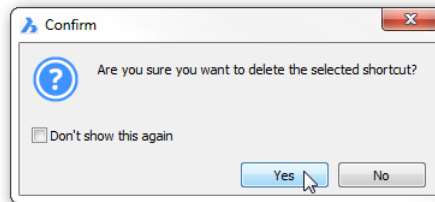
To edit or delete a keyboard shortcut, follow these steps:

1. Open the Customize dialog box with the **Cui** alias.
2. In the **Keyboard** tab, select a keystroke in the left hand column, such as the **CTRL+SHIFT+F** you defined earlier.
3. In the Keyboard Shortcut area, edit the command. You can, for example, backspace over the command text, and then enter another macro.
4. To remove a keyboard shortcut, right-click it, and then choose **Delete Shortcut** from the shortcut menu.



Deleting a shortcut definition

5. BricsCAD asks if you are sure; click **Yes**.



Getting one last chance before the definition is gone

6. Click **OK** to exit the dialog box.

Tutorial: How to Assign Multiple Commands

You can assign more than one command to keyboard shortcuts. When two or more commands are executed together; they are called *macros*. (Learn more about macros in Chapter 8.)

For example, to copy all objects in the drawing to the Clipboard takes two commands: **Select All**, followed by **CopyClip**. The macros for each are as follows:

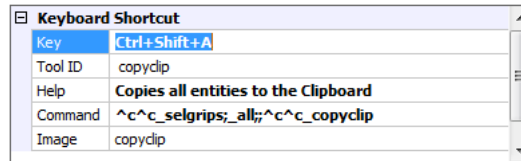
```
Select All    ^c^c_selgrips;_all;;  
CopyClip     ^c^c_copyclip
```

To combine these two commands into a single keystroke shortcut, Ctrl+Shift+A, follow these steps:

1. Insert a new keyboard shortcut.
2. In the Add Keyboard Shortcut dialog box, choose one command, such as **Copy** (CopyClip) found in the Edit section.
3. Click **OK** to return to the Customize dialog box.
4. Add the other command, Select All, by editing the **Command** section of the Keyboard Shortcut area. Add the text shown in boldface:

```
^c^c_selgrips;_all;;^c^c_copyclip
```

The semicolon (;) is a metacharacter that's equivalent to pressing Enter.



Adding a shortcut keystroke to the macro

5. Add the shortcut keystroke to the **Key** field:
ctrl+shift+a

You may wish to update the **Help** string to something like, “Copies all entities to the Clipboard.”

6. Click **OK**, and then test the macro by pressing **CTRL+SHIFT+A**. BricsCAD reports:
: _SELGRIPS
Select entities to display grips: _ALL
Select entities to display grips:
: _COPYCLIP

Try pasting the copied objects into another document using **CTRL+V**.

Customizing Command Aliases

As well as keystroke shortcuts, BricsCAD also allows you to define one- or more-letter command shortcuts, called *aliases*. An alias typically is an abbreviation of a command name, such as **L** for the **Line** command, and **OS** for **OSnap** (object snap).

You may wonder about longer aliases, such as **Colour**. Aliases can, in fact, be any length of characters, but when they are more than two or three characters in length, then they start to defeat the purpose of aliases, which is to be brief. Long alias names are, however, useful for making BricsCAD compatible with older versions with different command names, and with other CAD packages. For instance, Colour is another name for the Color command.

BricsCAD predefines around 300 aliases. There's a lot of them, because BricsCAD needs to have all of AutoCAD's aliases, plus more for BricsCAD commands that have changed names over previous releases.

With recent releases of BricsCAD, aliases are being deprecated. No new aliases are being added. This is because AutoComplete has taken over the task of entering a few letters to access an entire command name. In the figure below, I typed in l-a-y.



BricsCAD finding all commands that begin with LAY

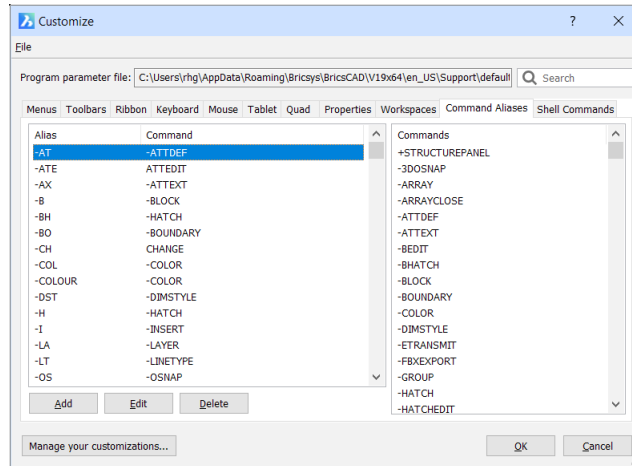
Instead of memorizing aliases (some of them obscure) that work for many — but not all commands — we now type the first one, two, or three letters of any command name to access all of them. Who would know that 'cui' is an alias for Customize; AutoComplete lets new users type 'cus' instead.

TIP Although they were designed to reduce keyboard typing, aliases can also be used in toolbar and menu macros. However, if the definition of the alias is changed, then the macro will no longer work.

TUTORIAL: CUSTOMIZING ALIASES

You access aliases, as follows:

1. Enter the **Customize** command. (Or, enter the **Cui** alias at the ‘:’ command prompt.)
2. Notice the Customize dialog box. Click the **Command Aliases** tab.



Customize dialog box showing the Command Aliases tab

This version of the dialog box looks different from other tabs, in that it has just two panes, plus some extra buttons along the bottom

Alias-Command pane (at the left) — lists all aliases already defined. Notice that an alias, such as **-AT** is linked with the **-AttDef** command.

Commands pane (on the right) — lists the names of all commands found in BricsCAD.

TIPS Aliases are stored in the the *default.pgp* file.

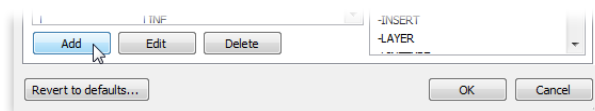
If you’ve created aliases with AutoCAD, you can import them into BricsCAD using Notepad to open the *default.pgp* file. Copy and paste aliases from AutoCAD’s *acad.pgp* file.

Unlike keyboard shortcuts, aliases *cannot* be macros. This means each alias supports a single command only.

TUTORIAL: CREATING NEW ALIASES

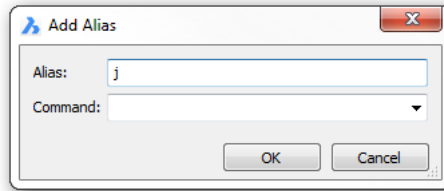
In this tutorial, you create an alias for one of BricsCAD’s commands lacking an alias: **J** for the **Join** command.

1. In the Customize dialog box’s **Command Aliases** tab, click the **Add** button.



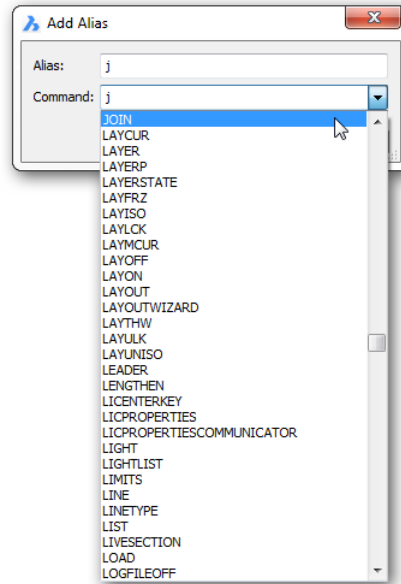
Adding an alias

2. Notice the **Add Alias** text entry box:
 - a. In the Alias field, enter **j**.



Specifying the new alias character

- b. From the **Command** droplist, choose **Join**.



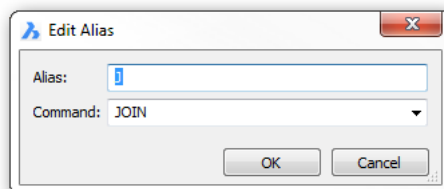
Choosing the command from a droplist

- c. Click **OK** to dismiss the dialog box.
3. Click **OK** to close the Customize dialog box.
4. Now test the alias by entering **J** and then pressing **Enter**. BricsCAD should execute the Join command.

Tutorial: Editing & Deleting Aliases

To edit an alias, follow these steps:

1. In the Customize dialog box's Aliases tab, choose the alias you wish to edit, such as the **J** you defined above.
2. Click **Edit**. (To erase an alias, click **Delete** instead.)



Editing an alias

BRICSCAD ALIASES SORTED BY COMMAND NAME

A

align	al
aperture	ap
apparent	planviewint
arc	a
area	aa
array	ar
attdef	at, ddattdef
-attdef	-at
attdisp	ad
attedit	-ate
atttext	ax, ddatttext
-atttext	-ax

B

background	backgrounds
base	ba
blipmode	bm
block	b
-block	-b
boundary	bo, bpoly
-boundary	-bo
break	br

C

centerline	cl
centermark	cm
chamfer	cha
change	-ch
circle	c
color	col
-color	-col, -colour
color	colour, ddcolor, ddcolour, setcolor
copy	co, cp
copylink	cl
customize	cui
cylinder	cyl

D

ddedit	ed
ddgrips	gr
ddselect	se
ddvpoint	setvpoint, viewctl, vp

dist	di
divide	div
donut	do, doughnut
draworder	dr
dsettings	ddrmodes, rm
dview	dv
dxfout	dx

Dimensions

dim	dimension
dimaligned	dal, dimali
dimangular	dan, dimang
dimbaseline	dba, dimbase
dimcenter	dce
dimcontinue	dco, dimcont
dimdiameter	ddi, dimdia
dimedit	ded, dimed
dimlinear	dimhorizontal, dimlin,
dimrotated, dimvertical, dli	
dimordinate	dimord, dor
dimoverride	dimover, dov
dimradius	dimrad, dra
dimstyle	d, ddim, dimsty, ds, dst, expdim-
styles, setdim	
-dimstyle	-dst
dimtedit	dimted

E

eattedit	ate
ellipse	el
erase	delete, e
expblocks	bx, xb
explode	x
export	dwfout, exp
expucs	dducs, uc
extend	ex
extrude	ext

F

fillet	f, fi
--------	-------

G

geographiclocation geo
grid g

H

hatch bh, h
-hatch -bh,-h
hatchedit he
hide hi

I

id idpoint
image expimages, im
imageadjust iad
imageattach iat
imageclip icl
import imp
insert ddinsert, i
-insert -i
insertaligned insal
insertobj io
interfere inf
intersect in
isolateobjects isolate
isoplane is

L

layer ddlmodes, explayers, la
-layer -la
layerstate las
laymcur setlayer
leader le, lead
lengthen editlen, len
light lighting
lightlist ll
line 3dline, l
linetype ddtype, exptypes, lt
-linetype -lt
list li, ls
ltscale lts

M

matbrowseropen matb
matchprop ma
materialmap setuv
materials finish, mat, rmat
mirror mi
mirror3d 3dmirror, 3m

move m
mslide msnapshot
mspace ms
mtext mt, t
mview mv

N

newwiz ddnew

O

offset o
oops undelete, unerase
open op
options cfg, config, preferences, prefs
orthogonal or, ortho
osnap ddesnap, ddosnap, os, setesnap
-osnap esnap,-os

P

pan p, -p
pastespec pa
pedit editpline, pe
pline pl, polyline
point po
polygon pol
preview ppreview, pre
properties ch, ddchprop, ddmodify, mo, pr, props
propertiesclose prc
pspace ps
purge pu
-purge -pu
pyramid pyr

Q

qnew n
qtext qt
quit exit

R

rectang rec, rect, rectangle
redraw r
redrawall ra
regen re
regenall rea
region reg
reinit ri
rename ddrename, ren
-rename -ren
render rr

renderenvironment	fog
renderpresets	roptions
renderwin	renderscr
revolve	rev
rotate	ro
rotate3d	3r, 3drotate
rpref	setrender

S

save	sa
scale	sc
script	scr
section	sec
selgrips	selgrip
setucs	dducsp, ucp
setvar	set
shade	sha
shademode	vscurrent
sketch	freehand
slice	sl
snap	sn
solid	plane, so
spell	sp
spline	spl
splinedit	spe
stretch	s
style	ddstyle, expfonts, expstyle, expstyles, st
-style	font
subtract	su
sunproperties	sun

T

tablet	ta
-text	-t
text	tx
thickness	th
time	ti
tolerance	tol

torus	tor
trim	tr

U

union	uni
units	ddunits, un
-units	-un

V

vbaide	vba
view	ddview, expviews, v
-view	-v
vplayer	vl
vpoint	viewpoint,-viewpoint,-vp,-vpoint
vports	viewports, vport, vw
vslide	vs, vsnapshot

W

wblock	w
wcloseall	closeall
wedge	we
wmfin	wi
wmfout	wo

X

xattach	xa
xbind	-xb
xclip	clip
xline	inffline, xl
xref	exprefs, xr
-xref	-xr

Z

zoom	z
------	---

#

3darray	3a, array3d
3dface	3f, face
3dmesh	mesh
3dpoly	3p

3. In the **Edit Alias** dialog box, select another command, and then click **OK**.
4. Click **OK** to exit the Customize dialog box.

RULES FOR WRITING ALIASES

Here are some suggestions Autodesk provides for creating command aliases:

- ▶ An alias should reduce a command to two characters at most.
- ▶ Commands with a control-key equivalent, status bar button, or function key do not require a command alias. Examples of commands to avoid include the **New** command (already assigned to **CTRL+N**), **Snap** (already on the status line), and **Help** (already assigned to function key **F1**).
- ▶ Try to assign the first character of a command. If it is already taken by another command, assign the first two, and so on. For example, **C** is assigned to the **Circle** command, while **CO** is assigned to the **Copy** command.
- ▶ For consistency, add suffixes for related aliases. For example, **H** is assigned to the **Hatch** command, so assign **HE** to **HatchEdit**.

Tutorial: Hand-Coding Aliases

If you wish to write your aliases directly, open the *default.pgp* file with a text editor. In Windows, open the file in Notepad (or Text Editor in Linux, or TextEdit in Mac). The *.pgp* file is found in the following locations:

Windows — C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\default.pgp

Linux — /home/<login>/Bricsys/BricsCAD/V20/en_US/Support

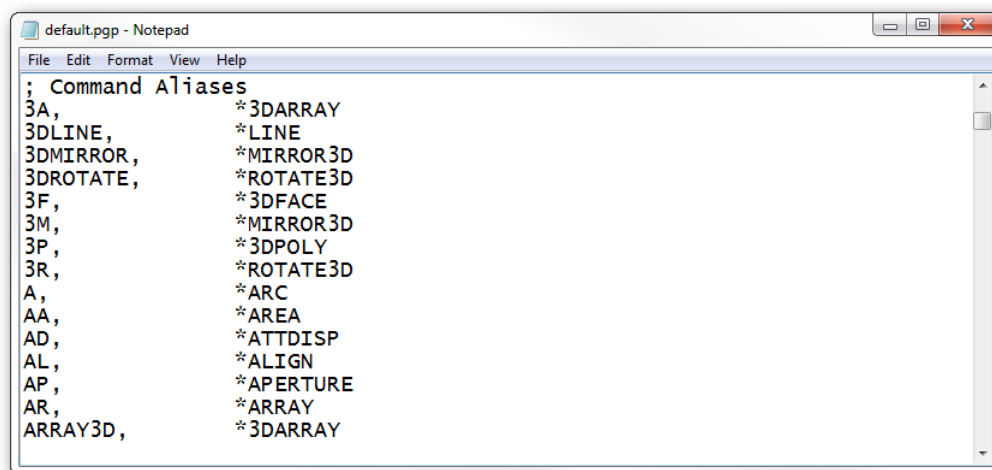
Mac — /Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/Support

In all cases, the beginning of the file's Command Aliases section looks like this:

To add a new alias, mimic the format shown above:

```
alias, *commandName
```

Enter the alias name followed by comma, space, asterisk, and the command name.



Text editor showing the content of the Default.pgp file

When done, press **Ctrl+S** (or **Cmd+S** on Macs) to save the file. Back in BricsCAD, use the **ReInit** command to reload the *.pgp* file.

Customizing Shell Commands

The final tab in the Customize dialog box is labelled **Shell Commands**. It is meant for customizing *shell* commands, which are almost never used anymore.

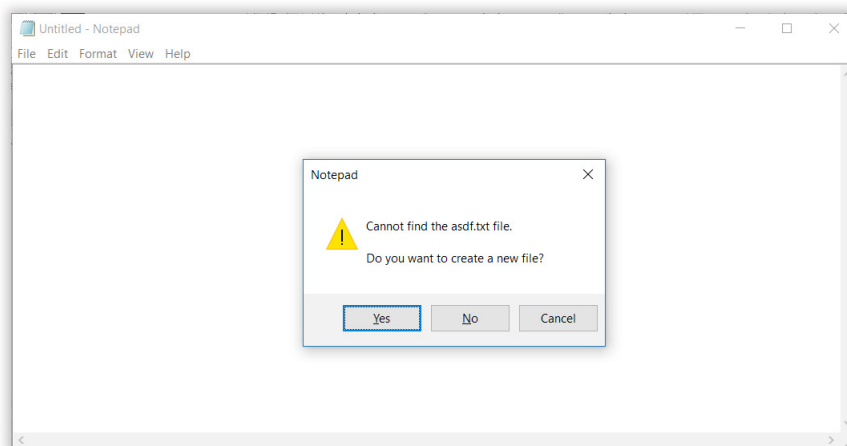
They are a holdover from the days of DOS (disk-based operating system), from before Windows allowed multiple programs to run on PCs. The DOS operating system limited computers to running one program at a time. (Additional programs, called “TSRs” [short for terminate but stay resident], could be loaded into memory and accessed, but they had to be very small and be specially coded.)

If you wanted access a text editor or another useful program while running CAD, then you could “shell out” of the program, run the text editor, and then return to the CAD program. The CAD program suspended operations while the text editor was running. This was done with the **Shell** command, sometimes known as running an “external command.”

Today, the Shell command is no longer necessary, because Windows lets us run many programs at the same time as we want, switching between them effortlessly. But Shell still works in BricsCAD and can be a handy way to access other software. For example, instead of starting a text editor by navigating through the Windows Start menu, you can type “notepad” at the BricsCAD command prompt:

```
: notepad  
File to edit: (Enter a file name)
```

After you enter a file name, Windows launches Notepad with the file. When you enter the name of a file that does not exist, Notepad offers to start a new file with that name, as shown below.



Starting a new text file

To load a file at the same time as executing the external command, include the file name with the command name, like this:

```
Shell Command: start notepad default.pgp
```

When the name of the command and/or file have spaces in them, you'll need to use quotation marks around them, like this:

```
Shell Command: start notepad "c:\program files\bricsys\bricscad V20\default.pgp"
```

Quotation marks let the operating system differentiate between spaces that separate commands from spaces that are part of a file or path name.

Shell commands are stored in the same *default.pgp* file as aliases, and so is found in the following folder: *C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support*.

The shell portion of file has not been adapted for use with Linux and Mac, because of shell commands' history stemming from DOS.

The Shell Commands section of the *default.pgp* file looks like this:

```
DEL,          DEL,          8,      File to delete: ,
SH,          ,          1,      *OS Command: ,
SHELL,       ,          1,      *OS Command: ,
START,       START,       1,      *Application to start: ,
NOTEPAD, START NOTEPAD, 1,      *File to edit: ,
```

There are four components that define a shell command:

Alias — specifies an alias for the shell command. It can be any word not already used by the *default.pgp* file.

Shell Command — specifies the name of the command sent to the operating system. Notice that for Notepad there are two words, Start and Notepad:

Start is the command that executes another command, Notepad in this case. Leave out the “.exe” extension of the command name. If the shell command needs to also specify a file name, it is entered here, as I'll show you later.

Prompt — specifies the wording of a prompt to display in the command bar. This can be blank or consist of some helpful words, such as “File to edit.”

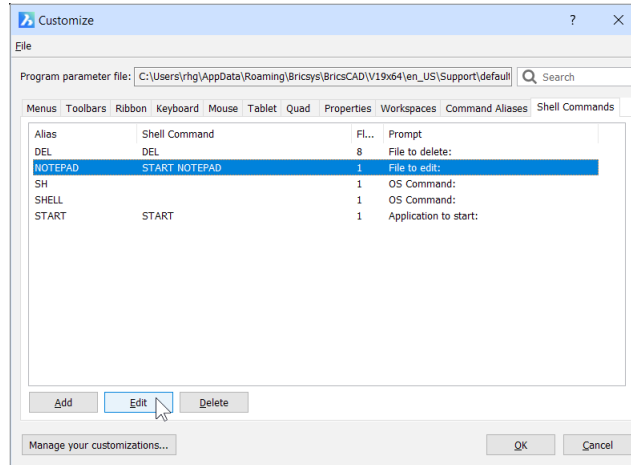
In addition, there is a flags field at the end of each line. It is left blank, because BricsCAD does not support flags. If it ever did, here is what they would mean:

Flag	Meaning
1	On: BricsCAD does not wait for the application to finish before returning to the command prompt. Off: BricsCAD waits for the application to finish.
2	On: The applications runs minimized; i.e., on the task bar. Off: The application runs normally, ie, displayed on the screen.
4	On: The application does not appear. Off: The application appears on the screen.
8	On: The shell command uses quotation marks; required when the name of the file contain spaces. Off: The shell command does not use quotation marks.

TUTORIAL: EDITING SHELL COMMANDS

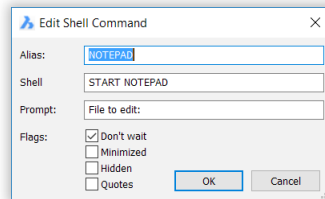
Shell commands are defined by the Shell Commands tab of the Customize dialog box. To see how they work, we'll look at one of the existing commands, Notepad. Follow these steps:

1. Enter the **Customize** command.
2. In the Customize dialog box, select the **Shell Command** tab. Notice that a few commands are already defined.



Adding a shell command

3. To edit a shell command, select one of them and click the **Edit** button. For this tutorial, select “Notepad,” and then click **Edit**. Notice the Edit Shell Command dialog box.



Editing a shell command

TIP It's not a good idea to use the names of existing BricsCAD commands or aliases; if you do, BricsCAD complains, “Cannot use the name for a shell command alias, because it already exist as a normal alias.”

4. Make changes to the parameters, such as the alias or prompt.
5. Click **OK** to exit the dialog box, and then click OK to exit the Customize dialog box.
6. Test the new customization.

The process for adding a new shell command is similar.

Customizing Mouse, Double-click, & Tablet Buttons

The mouse is your constant companion in BricsCAD, and you can customize its buttons, as well of those on a stylus or puck used with digitizing tablets. BricsCAD does not, however, explicitly support touch screens and styli used with Windows 8.x and 10. BricsCAD supports 3D mice, but its buttons are customized by the utility software provided with them

The Customize dialog box in BricsCAD is where you go to change the meanings of up to two buttons on mice, and up to 15 buttons on tablet input devices:

Mouse tab — assigns macros to the right and middle buttons, along with Shift, Ctrl (Cmd on Macs), and Shift+Ctrl keys; these are called “clicks.” You can also assign macros to double-click actions.

Tablet tab — assigns macros to buttons, along with Shift key

CHAPTER SUMMARY

This chapter covers the following topics:

- Understanding how commands are assigned to mouse and tablet buttons and double-click actions
- Assigning commands, macros, and shortcut menus to mouse buttons
- Assigning actions to double-click actions and tablet buttons

SUMMARY OF MOUSE COMMANDS & VARIABLES

These are the system variables that affect the use of mice with BricsCAD:

Variable	Meaning
CtrlMouse	Toggles specific key-button combinations; when on , the following work: Ctrl+Left button 3D view rotation Ctrl+Rigth button 3D viewing with fixed z axes Ctrl+Shift+Left button Real-time zooming Ctrl+Shift+Right button Real-time panning
Ctrl3DMouse	Toggles the use of 3D mice with BricsCAD (must restart the program after changing it): 0 (Off) Disables 3D mouse 1 (On) Enables 3D mouse
MButtonPan	Specifies the function of the middle button: 0 (Off) Carries out the action defined by the Customize command 1 (On) Pans drawing while dragging with middle button held down; default
ShortcutMenu	Specifies what happens when you press the right mouse button: 1 Enable default mode shortcut menus 2 (Default) Display shortcut menu for editing 4 Display shortcut menu for commands during any command 8 Display shortcut menu for commands when command options are active 16 (Default) Display the shortcut menu when right button is held down longer
ShortcutMenuDuration	Specifies how long to hold down the right button until a shortcut menu appears: 250 (Default) Time in milliseconds, or 1/4 second
ZoomFactor	Sets the zoom speed: 3 Slowest speed, helpful for very slow zooming 60 Default speed 100 Fastest speed, useful for very large drawings
ZoomWheel	Determines the scroll wheel's zoom direction: 0 (Off) Moving wheel forward zooms in; default 1 (On) Moving wheel forward zooms out; Mac-like zoom

The following command works with tablets:

Tablet — toggles use of the tablet, and configures tablet surfaces.

About Mice and Their Buttons

The very first computer mouse had three buttons. Since then, the button count on mice has strayed in different directions. Some mice have many more than three buttons, while others sport no buttons at all.



Left to right: An early mouse with three buttons; a modern mouse of many buttons; and an Apple mouse with no buttons

For instance, I use the Logitech MX mouse with my primary work computer. It has nine buttons (I think) that offer functions like moving forward-reverse through Web pages and side-to-side scrolling. Other mice, such as one model from Apple, have no buttons at all: you tap, slide, pinch, and otherwise move your fingers on the smooth surface, as if it were a touchpad.

Mouse Buttons. Whether the mouse has three, nine, or zero buttons, BricsCAD customizes only two of them — the ones traditionally named “middle” and “right.”

The left mouse button is never customized as it performs the all-important *picking* function.



BricsCAD's 1-3-2 numbering of left, right, and center buttons on mice

Modifier Keystrokes. You can define more than two actions for each button by adding Shift, Ctrl (on Macs, the Cmd button), and Shift+Ctrl keys to buttons.

So, when you hold down the **Shift** key while clicking the right button, BricsCAD executes a different command from when you click the right button alone.

Double-clicking. The other way to access more commands through the mouse is with double-clicking. When you double-click an entity in the drawing, BricsCAD runs a command suitable to editing the entity.

QUICK SUMMARY OF DEFAULT BUTTONS

Sometimes you might see buttons referred to by number. Here is what the numbers mean:

Button	Number
Left	1
Right	2
Center	3

DEFAULT ACTIONS

BricsCAD assigns these macros to the right and center mouse buttons through the Mouse tab of the Customize dialog box.:

Keystroke+Button	Action	Macro
Right button	Repeats last command	<i>unassigned</i>
Middle button	Displays object snap shortcut menu	<code>\$p0=SNAP \$p0=*</code>
Shift+		
Right button	Displays Entity Snap shortcut menu	<i>unassigned</i>
Middle button	Real-time rotation	<i>unassigned</i>
Ctrl+		
Right button	Real-time rotation about z axis	<i>unassigned</i>
Middle button	Displays object snap shortcut menu	<i>unassigned</i>
Shift+Ctrl		
Right button	Real-time pan	<i>unassigned</i>
Middle button	Real-time rotation	<i>unassigned</i>

WALKTHROUGH BUTTONS

Walkthrough navigation in perspective views uses the following mouse buttons and keystrokes:

Mouse Button	Command	Meaning
Alt + Left button	RtWalk	Walks forward, backwards, and sideways
Alt + Middle button	RtUpDown	Moves up, down, and sideways
Ctrl + Middle button	RtLook	Looks around
Ctrl + Home key	...	Resets view direction to the horizontal
Alt + Home key	...	Moves target point to the center of the scene
Alt + Plus key	RtWalkSpeedFactor	Increases walking speed
Alt+Minus key	RtWalkSpeedFactor	Decreases walking speed
Ctrl + Plus key	RtRotationSpeedFactor	Increases rotation speed
Ctrl+Minus key	RtRotationSpeedFactor	Decreases rotation speed

TABLET BUTTONS

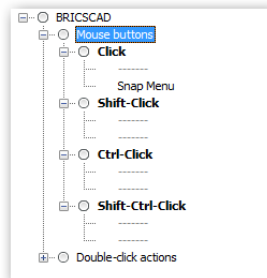
BricsCAD assigns no macros to stylus or puck buttons, by default. This can be done through the Tablet tab of the Customize dialog box.

Most double-click actions bring up the Properties panel, if it isn't already visible. The remainder are assigned to entity-specific commands, such as **HatchEdit** for hatch patterns or double-clicking a dimension starts the **DdEdit** command for editing its text. You can customize double-click actions.

Delayed Press. There is one more mouse action available for invoking commands. When you hold down the right mouse button longer than 250 milliseconds, then BricsCAD can carry out a different command. The command, however, cannot be customized by you.

(Some CAD packages execute commands through mouse *gestures*, where the mouse's dictional movements are interpreted as commands. This is not available in BricsCAD.)

The Customize dialog box's **Mouse** tab lists the possibilities, as shown below. The only button already assigned is Click (left button); it displays the Snap Menu. The other buttons are unassigned.



Customize dialog box's Mouse tab showing button definitions

Here is how to read the content of the Mouse tab: under **Click**, there are two entries: the first (-----) is for button #2, and the second (Snap Menu) is for button #3.

The remaining entries are for keyboard modifiers to buttons — Shift, Ctrl, and Shift+Ctrl — such as Shift-Click. At the end is the section for double-click actions

About the Pick Button

You cannot customize the pick button (left button) for good reason. You never want to lose the ability to pick things in the user interface or the drawing! BricsCAD doesn't let you customize the pick button in combination with keystrokes, either.

About the Right Button

The right button almost always has two specific functions:

- In the drawing area** — ends the current command or restarts the last command
- In most user interface elements** — accesses shortcut menus, just like in other programs

About the Middle Button

The middle button is often used as a quick way to pan and zoom around the drawing:

- Panning** — hold down the middle button, and then drag the mouse to pan
- Zooming** — roll the scroll wheel back and forth to zoom in and out at the cursor
- Orbiting** — hold down the **Shift** key, hold down the middle button, then drag the mouse around to orbit in 3D about the cursor

TIP When you zoom and orbit using the middle button, the action takes place where ever the cursor is in the drawing. For example, when you roll the scroll wheel to zoom in, you zoom into where the cursor is located. This is great for zooming into a specific place in the drawing: just move the cursor there, and then scroll.

If your mouse has a scroll wheel (most do, these days), then the wheel is the middle button: to drag, hold down the scroll wheel and then move the mouse.

Troubleshooting

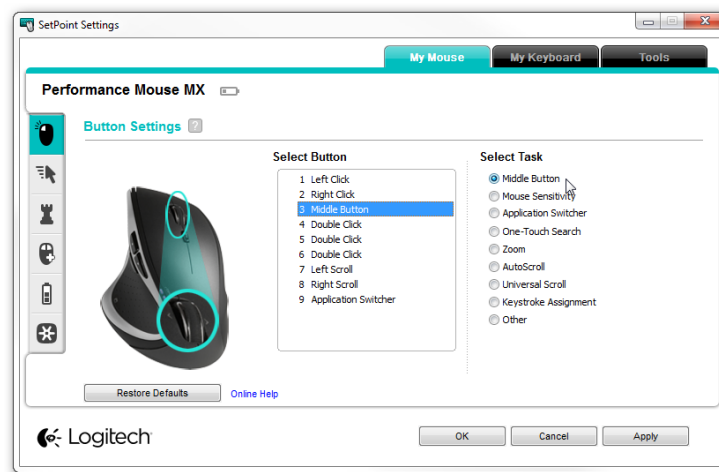
If panning and zooming do not work as you expect, then there are two settings to check out.

- ▶ Enter the **MButtonPan** variable to ensure it is set to **On**. When on, then the middle button pans the drawing. When off, the middle button carries out the action defined in the Customize dialog box.

: **mbuttonpan**

New current value for MBUTTONPAN (Off or On) <Off>: **on**

- ▶ If the middle button is still not panning the drawing, then you may need to change the button's definition in the mouse's utility software. For example, if you use the Logitech's SetPoint software to define the middle button as "Double Click" or anything else, then this overrides the definitions assigned by BricsCAD. To fix the no-panning problem, change the mouse driver's definition back to "Middle Button," as illustrated below.



Software for defining Logitech mouse button actions

Me, I used to always define the middle button for double-clicks, which is an especially efficient way of getting around Windows. But this interferes with the CAD program's ability to pan using the middle button. So I now define the double-click action to one of the side-mounted buttons on my nine-button mouse.

OTHER INPUT DEVICES

Many weird and wonderful input devices have been invented during the course of computing history, such as 100-button boxes, voice input, and virtual reality goggles. Here I will concentrate on three supported by BricsCAD: digitizing tablets, 3D mice, and touch screens.

Digitizing Tablets

Even before mice were invented, digitizing tablets were the most popular way to control CAD software. Other input devices in the 1970s and early 1980s included the light pen and of course the keyboard.

Tablets are dual purpose, allowing users to input commands and draw-edit the model. They ranged in size from a piece of paper to covering an entire desk.



*Left: Intergraph workstation with two monitors and desk-size digitizer in the 1970s
(Image source design.osu.edu/carlson/history/lesson10.html)*



*Right: Typical 12"x12" digitizing tablet used in the 1980s with four-button puck, stylus, and connecting cables
(Image source www.biocomp.net/o12691.htm)*

Pucks had a minimum of four buttons, and were available with 12 or 16 buttons. The pucks used for input are precise (unlike today's mice), because they use absolute positioning. (Mice use relative positioning — it doesn't matter where the mouse is positioned on the desk.) Also available for input was a stylus, like the styli used today with certain touch screen computers and mobile devices.

Some CAD users still employ the legendary digitizing tablet. It is, however, increasingly difficult to use older models as most computers no longer have the serial port needed to connect the tablet; you can buy serial ports as an add-on to for desktop computers.

Tablet support is included with BricsCAD, as described later.

3D Mice

The 3D mouse is designed to make work with 3D drawings easier. It features a puck that you move up and down in the z direction, as well as twist and rotate for 3D view rotations.

In practice, you use two mice:

- ▶ The regular "2D" mouse for choosing commands and picking objects
- ▶ The puck of the 3D mouse for moving the viewpoint

Users typically employ the regular mouse with the right hand, and the 3D mouse with the left.

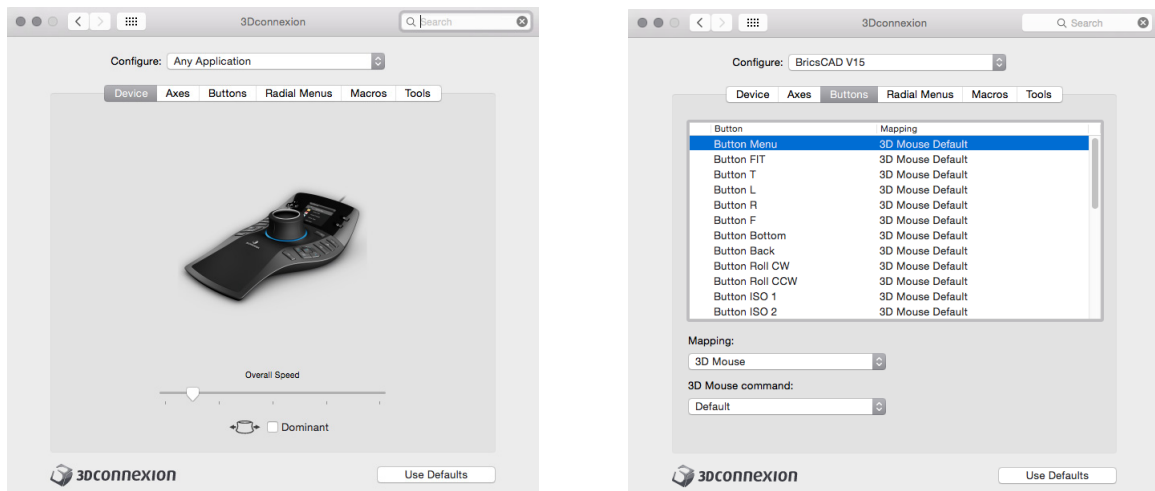
The 3D mice range in size from simple wireless puck with two buttons to multi-button behemoths sporting a customizable LCD display. The movement of the puck and the buttons can be customized.



Left: Two-button 3D mouse; right: multi-button 3D mouse with LCD screen (image source 3Dconnexion)

BricsCAD supports 3D mice from 3Dconnexion, but before it can recognize a 3D mouse, the 3Dconnexion device driver must be installed on your computer. Driver software is included for computers running recent releases of Windows, MacOS, and Linux. For support and downloads, see <http://www.3dconnexion.com/supported-software/mechanical-engineering/bricscad.html>. You may need to reboot your computer after installing the 3Dconnexion driver.

There are no controls in BricsCAD for 3D mice, with the exception of the **Ctrl3DMouse** variable; it enables and disables the 3D mouse. The actions of the 3D mouse's buttons and puck are defined by the 3Dconnexion Properties software, as illustrated below.



Settings for multi-button SpacePilot Pro mouse

Touch Screens

BricsCAD does not officially support touch screens. But they are common with computers running Windows 8.x and 10, and so I thought I should touch on the subject here. Pardon the pun.

The touch actions supported by BricsCAD are just those supported by Windows itself. Below I list the touch actions supported by Microsoft in Windows, and the reaction from BricsCAD.

Touch Action	Effect in Windows	Effect in BricsCAD
One-finger actions		
Tap	Equivalent to clicking (selecting item)	Picks points while drawing entities; Selects individual entities and grips; Selects UI elements, like ribbon buttons
Press and hold	Equivalent to right-clicking	Displays right-click menus; <i>Does not cancel or restart commands</i>
Flick up or down	Scrolls page down or down	<i>Has no effect</i>
Drag vertically	Scrolls page up or down	Moves drawing and dialog box scroll bars
Drag horizontally	Selects text	Selects text in command bar

Touch Action	Effect in Windows	Effect in BricsCAD
Two-finger actions		
Pinch	Zooms in or out	Zooms in and out
Rotate	Rotates clockwise or counter-clockwise	<i>Has no effect</i>
Hold and tap *	Equivalent to press-and-hold	Same as press-and-hold

*) Press the element with one finger, then quickly tap with another finger, while continuing to press the item with the first finger. Access shortcut menus like press and hold and right-clicking.

While many screens support up to ten fingers touching at the same time, Windows supports just one- or two -finger touches. Individual applications can support more simultaneous touches, if they wish

Touch Pads

Mac computers do not support touch screens. The MacOS operating system does, however, support touch pads, whose actions are listed below. (Image source Logitech.)



Touch Action	Effect in iOS	Effect in BricsCAD
One-finger Actions		
Tap	Equivalent to clicking (selecting item)	Picks points while drawing entities; Selects individual entities and grips; Selects UI elements, like toolbar buttons
Two-finger Actions		
Tap	Equivalent to right-clicking	Cancels and restarts commands
Double-tap	Smart zoom to element	<i>Has no effect</i>
Pinch	Zooms in or out	<i>Has no effect</i>
Rotate	Rotates clockwise or counter-clockwise	<i>Has no effect</i>
Three-finger Actions		
Drag	Moves elements, like dialog boxes	Zooms in and out

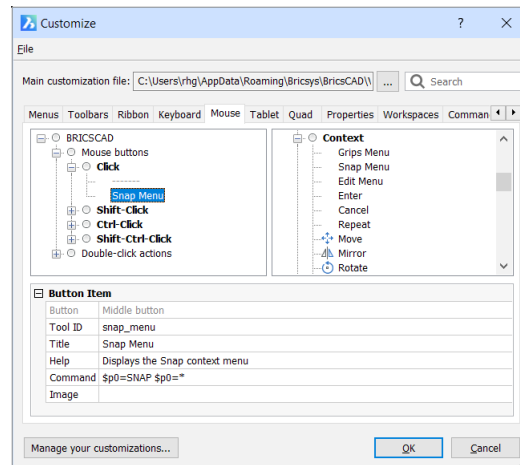
Note: Four-finger actions are not supported by BricsCAD.

Defining Actions for Mouse Buttons

In summary, you can assign the following types of actions to mouse buttons:

- ▶ Single-click actions — to right and center buttons only
- ▶ Shift, Ctrl, and Alt keystrokes — to assign additional actions to right and center buttons
- ▶ Double-click actions — to the left button only
- ▶ Commands, macros, and shortcut menus — can be assigned as actions to buttons

In BricsCAD, the Customize dialog box's **Mouse** tab consists of three panes:




Customize dialog box showing the **Mouse** tab

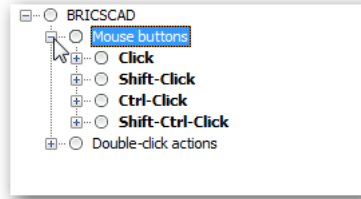
- ▶ **Click** pane (at the left) — lists the mouse buttons and key combinations that can be assigned actions
- ▶ **Available Tools** pane (at the right) — lists all of BricsCAD's commands sorted by menu name, and shortcut menus; to assign a command or menu to a button, just drag it from this pane into the appropriate button in the Clicks pane
- ▶ **Button Item** pane (at the bottom) — edits button settings

TUTORIAL: BUTTON ASSIGNMENT

In this tutorial, you assign the **Move** command to the Ctrl+right button by following these steps:

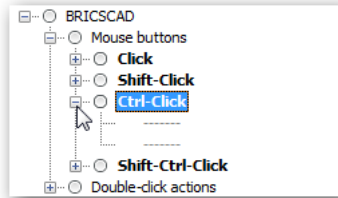
1. From the **Tools** menu, select **Customize**. Notice the Customize dialog box.
(Alternatively, enter the **Customize** command, or enter the **cui** alias. Or, else right-click any toolbar, and then choose **Customize** from the shortcut menu.)
2. Choose the **Mouse** tab. See figure above.
3. The mouse button you are defining will be pressed in conjunction with the **Ctrl** key, and so you need to access the correct part of the pane. Do it like this:

- a. Open the **Mouse Buttons** node by clicking the  button.



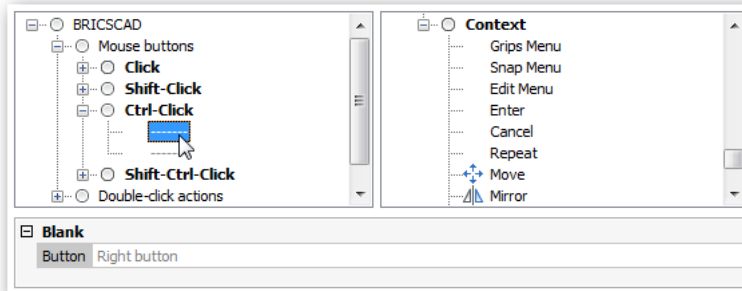
Opening the Mouse Buttons node

- b. Open the **Ctrl-Click** node.



Opening the Ctrl+Click node

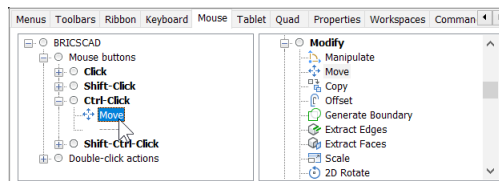
Notice that under Ctrl-Click there are two blank entries: each button is labeled with '-----'. The first '-----' refers to the right button and the second one to the middle button. Click and then look at the bottom pane, where the button name is identified.



Identifying the right button

I think it is a bug in BricsCAD that the default actions are not listed. These default actions are turned on and off with the **CtrlMouse** variable, as described by the boxed text earlier, “Summary of Mouse Variables.”

4. To attach the **Move** command to this button, drag the Move tool onto the button, like this:
- a. In the Available Tools pane, open the **Modify** node.

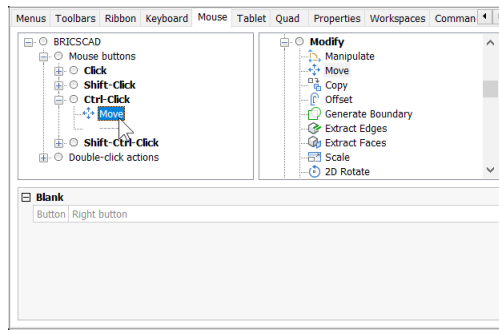


Choosing a command from the list of Available Tools

- b. Choose the **Move** tool.
- c. Drag it to the correct button position under **Ctrl+Click**, as shown above. Notice that the Move command now occupies the first mouse button position under Ctrl-Click (see figure above).

TIP If you drag the tool to the wrong button, then no worries. Just drag it to the correct one.

In the Button Item pane, notice that there are no properties for you to edit. BricsCAD changes the Button property for you when you move the tool to another button.



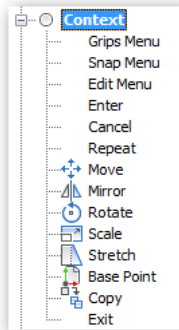
No properties for new buttons

5. With Move assigned to the Ctrl+Click right button, it's time to test it:
 - a. Click **OK** to save the changes and exit the Customize dialog box.
 - b. In the BricsCAD drawing window, hold down the **Ctrl** key, and then press the mouse's right button. The Move command should start up.

TUTORIAL: ASSIGNING SHORTCUT MENUS TO BUTTONS

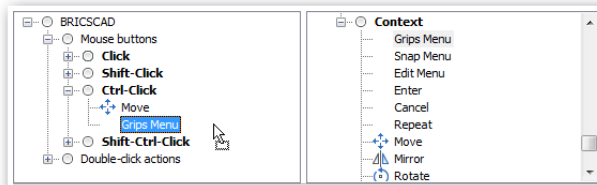
To attach a shortcut menu to a mouse button, follow the same steps as above. When it comes to step 4, however, you do things just a touch differently. Follow these steps:

- 1 - 3. Follow the steps listed above.
4. In the **Available Tools** pane, scroll down to the last tool, which is named "Context."
 - a. Open the node to see the list of shortcut menus (a.k.a. context menus).



Context (right-click) menus available in BricsCAD

- b. Choose a menu tool, and then drag it to the desired button position.

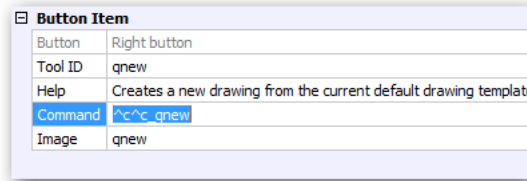


Dragging a context menu to a button, from left to right

Tutorial: Writing Macros for Buttons

In addition to commands and shortcut menus, you can attach macros to buttons in two steps:

1. Drag any tool onto a button, as described in the tutorial “Defining Actions for Buttons.” The tool doesn’t matter, just make sure it isn’t one taken from the Context node.
2. Edit the tool’s Command property:



A macro being written in the Command field

See Chapter 8 for information on writing macros.

CUSTOMIZING DOUBLE-CLICK ACTIONS

When you double-click an entity, BricsCAD performs an action that is related to the entity. For instance, double-click a hatch pattern, and the Edit Hatch dialog box appears; double-click some text and a text editor appears.

If an action is not defined for an entity, then BricsCAD displays the Properties panel. Not that there is anything wrong with it; the Properties panel in fact is sometimes more powerful than the designated editing command.

Here is a list of the default double-click actions that don’t launch the Properties panel:

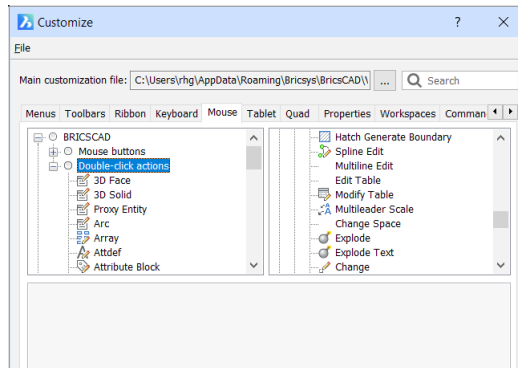
Entity	Double-click Command
Array (associative)	ArrayEdit
Attribute definition	DdEdit
Attribute in block	EAttEdit
Block	BEdit
Dimension	DdEdit (edits dimension text)
Hatch pattern	HatchEdit dialog box
Image	ImageAdjust
LwPolyline (modern polyline)	PEdit
Multiline leader	DdEdit (edits leader text)
Polyline	PEdit
Section plane object	ClipDisplay (toggles clipping on and off)
Spline	SplinEdit
Text (single-line text)	DdEdit
Tolerance	DdEdit (to edit the text in the tolerance)
Xref (external reference file)	RefEdit

Changing a Double-click Action

We are going to assign the Join command to the arc entity. Even better, we will turn it into a macro that closes any arc that you double-click. Closing an arc makes it a circle.

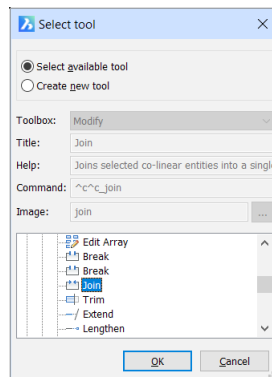
To change the action assigned to a double-click, follow these steps.

1. Open the Customize dialog box with the **cui** alias, and then click the **Mouse** tab.
2. Scroll down until you reach the Double-click section.
3. Open the node. Notice the list of entity names.



Double-click section of the Customize dialog box

4. To change the action associated with an entity, select the name of the entity. For this tutorial select **Arc**.
5. You cannot just edit the Command property, because then the double-click action will not work, I find. Instead, follow these steps:
 - a. Click the **Tool ID** property. Notice the Browse button at the far right end.
 - b. Click the **Browse** button. Notice the Select Tool dialog box.
 - c. Go to the **Modify** node, and then choose the **Join** tool.



Selecting Join from the list of tools

- d. Click **OK**.
6. Back in the Customize dialog box, notice that BricsCAD has filled in all the properties for the Join command. Now you can edit the command macro to make it close arcs, like this:
 - a. Click in the **Command** field.
 - b. Add the text shown in blue:
`^C^Carc;c1;`

c. And you're done.

Double-Click Action	
Name	Arc
DXF Name	ARC
Tool ID	Join
Title	Join
Help	Joins selected co-linear entities into a single entity
Command	^c^c_join;d;
Image	Join

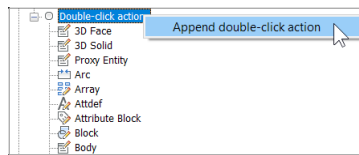
New command assigned

7. Test the change you made, like this:
 - a. Click the **OK** button to exit the Customize dialog box.
 - b. Use the **Arc** command to draw an arc.
 - c. Double-click the arc: it should close into a circle.

Making a New Double-click Action

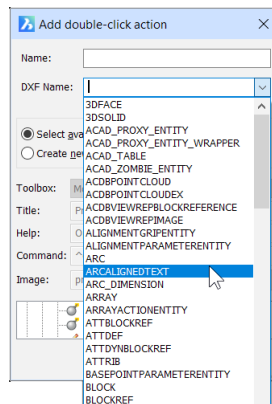
Not all entities are listed in the Mouse tab's Double-click section. To see the full list — and to create a new command that offsets xlines — follow these steps:

1. Right-click **Double-click action**.
2. Then choose **Append double-click action** from the shortcut menu.



Adding an entity to the list

Notice the Add Double-click Action dialog box.

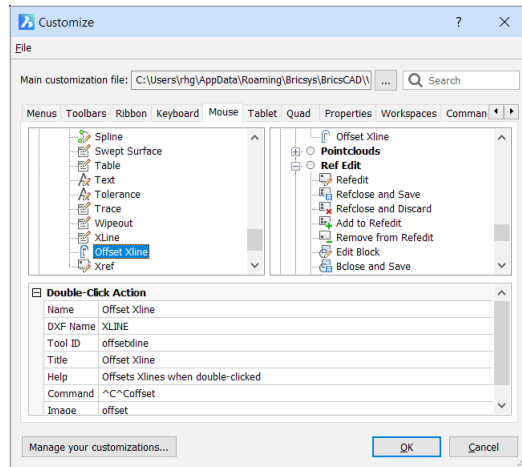


Selecting a DXF name

3. Click the **DXF Name** droplist, and then choose an entity type. For this tutorial, scroll all the way down to **XLine**.
4. With the entity chosen, now it's time to assign a command to the double-click action, and so choose the **Create New Tool** radio button.
5. From the **Toolbox** droplist, chose a BricsCAD command to apply. For this tutorial, we'll choose the **Offset** command.
6. Now fill out *all* the fields that are blank in the dialog box — Name, Title, Help, Command, and even Image — because if you leave one or more blank, then the OK button remains unavailable.

(To add an image, click the Browse button at the far end of the Image field, and then chose an icon from the hundreds provided by BricsCAD.)

7. Click **OK** to close the dialog box. Notice that the Double-click Action pane has all of its fields filled in, because you filled them out.

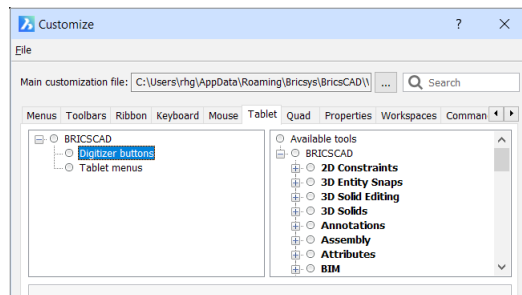


Completed macro

8. If you wish, edit the Command macro to provide the Offset command with a set offset distance, and so on.
9. Your final step is to test the change you made, like this:
 - a. Click the **OK** button to exit the Customize dialog box.
 - b. Use the **XLine** command to draw a construction line, and then press **Esc** to end the command.
 - c. Double-click the xline: the Offset command should launch, prompting you for the offset options.

Defining Actions for Tablet Buttons

Tablet digitizer buttons are customized in the **Tablet** tab, as illustrated below.



Customize dialog box showing the Tablet tab

Now, normally there are no entries under Digitizer Buttons and Tablet Menus, because BricsCAD includes nothing for tablets with the Default profile. If you use a tablet, then you need to download the partial CUI files for tablet buttons and drawings overlays from the Bricsys Web site.

At time of writing, the Web site no longer appeared to host the file, so you can download it from my cloud account:

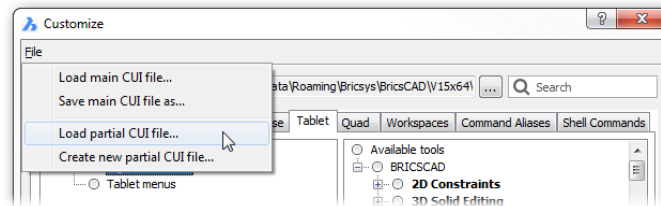
<https://my.pcloud.com/publink/show?code=kZ5k2hkZaJFXkmOS6S5zNw8TXAWWMm3bP8WV>.

After the *Tablet.zip* file is downloaded, unzip it. Notice that it contains two partial *.cui* files — *tablet.cui* and *tablet(acadLike).cui* — along with several support files that not necessary right now.

overlay(A3).pdf	10/20/2010 12:49 ...	Adobe Acrobat D...	741 KB
overlay(cm).dwg	5/4/2010 4:49 PM	AutoCAD Drawing	38 KB
overlay(inch).dwg	5/4/2010 4:50 PM	AutoCAD Drawing	38 KB
overlay.png	4/29/2010 12:10 PM	PNG File	1,222 KB
tablet(acadLike).cui	2/4/2010 3:24 PM	BricsCAD Customi...	88 KB
tablet.cui	4/29/2010 3:24 PM	BricsCAD Customi...	78 KB
Tablet.zip	5/8/2014 12:22 PM	zip Archive	1,639 KB

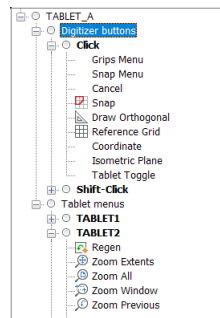
Listing of files inside the Tablet.zip archive

Back in BricsCAD, open the Customize dialog box, and then click **File | Load Partial CUI File** (or use the **MenuLoad** command).



Loading a partial .cui file

In the Choose A Customization File dialog box, select *tablet.cui* and then click **Open**. Notice that the Customization dialog box now lists two sections for tablet buttons and menus.



Button and menu definitions added to support digitizing tables

TIP Use the **Tablet** command to configure and calibrate the overlay on the surface of the tablet. This command works only after a tablet is attached to the computer, and its drivers have been installed, including *WinTab32.dll*. See “Digitizing Tablet” in the *BricsCAD User Guide* portion of the online help system.



1 2 3 4 5 6 7 8 9 10 11 12 13 14

A
B
C
D
E
F
G
H

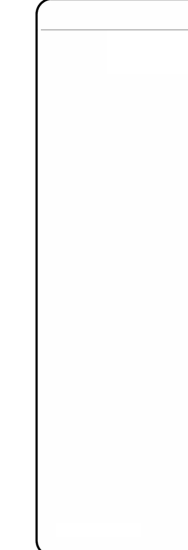
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z

REGEN	EXTENTS	ZOOMALL	WINDOW	PREVIOUS
REDRAW	ZOOMLEFT	ZOOMCNTR	ZOOMOUT	ZOOM IN
ATTDISP	UCSICON	REF GRID	SNAPGRID	PAN
TOOLBAR	PMSPACE	VIEWPORTS	MVIEW	EXPVIEWS
VIEWS	DEPVIEW	PLAN	SETVIEWPT	DVIEWCTRL
FULLREN	RENDER	LBACK	BACK	RBACK
SHADE	HIDE	LSIDE	TOP	RSIDE
MATERIAL	BKGROUND	LEFT	FRONT	RIGHT
LIGHTING	RENDERSET	LBOTTOM	BOTTOM	RBOTTOM

VIEWING / RENDERING

BOX	MULTITEXT	CIRCLE	LINE
SPHERE	TEXT	DONUT	RAY
CYLINDER	PYRAMID	SPLINE	INFINITE
CONE	3DFACE	ELLIPSE	ELLIP ARC
WEDGE	POLYFACE	BLOCK	POLYLINE
TORUS	DISH	POINT	3DPOLY
TABSURF	MESH	HATCH	POLYGON
REVSURF	RULESURF	3DMESH	RECTANG
PLANE	EDGESURF	THICKNESS	ARC

ENTITY CREATION



WORKSPACE

INSERTOBJ	PASTESPEC	INSBLOCK	XREF	EXPLAYER
SETDIM	LIMITS	SAVEBLK	UNITS	EXPBLOCK
SAVESTYLE	RESTORE	APPLYSTYL	LINEAR	EXPVIEWS
LEADER	ROTATED	OBLIQUE	ANGULAR	EXPCOORD
CONTINUE	BASELINE	ORDINATE	ALIGNED	EXPLINE
TOLERANCE	CENTER	DIAMETER	RADIUS	EXPSTYLE
EDITDTEXT	ROTATE	REPOSITION	RESTORE	EXPDIM

DIMENSIONING / DRAWING EXPLORER

AREA	DISTANCE	FILL	SPELLING
STATUS	LISTINFO	IDCOORD	SETVAR
RECORD	STOP REC	PLAY	APPLDAD
APERTURE	DIMSTYLE	COLOR	SETLAYER
ORTHO	SETTINGS	SET UCS	DSETTINGS
CUSTOMIZE	MAIL	DDESELECT	DDGRIPS
CMDBAR	BROWSE	MENU	OPTIONS

SETTINGS / INQUIRY

NEAREST	ENDPOINT
MIDPOINT	CENTER
PERP	TANGENT
QUADRANT	INSERTION
POINT	EXTENSION
INTERSECT	APPARENT
CLEAR	FROM

SNAPS

4 15 16 17 18 19 20 21 22 23 24 25 26



135°	90°	45°
180°		0°
225°	270°	315°
1/2	1/4	1/8
1/16	1/32	1/64
	PAN	
	ZOOM WINDOW	REDRAW ALL
	ZOOM EXTENTS	
	ZOOM PREVIOUS	REGEN ALL

QUICK ENTRY

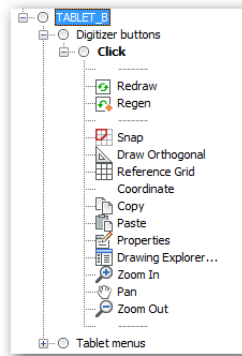
BASEPOINT	CUT	COPYCLIP	PASTE	COPYPROP	UNDO	REDO	PROPERTIES
ENTDATA	MEASURE	DIVIDE	DELETE	OLELINKS	DEL ALL	TIME	SEL ALL
COPYDATA	MIRROR	3D MIRROR	COPY	MOVE	ROTATE	SCALE	STRETCH
MOVEDATA	EXTEND	BREAK	CHAMFER	FILLET	3D ARRAY	ARRAY	ROTATE 3D
DELDATA	LENGTHEN	JOIN	TRIM	PARALLEL	BOUNDARY	FLATTEN	ELEVATION
REASSOC	VISRETAIN	VPLAYER	EDITPOLY	CHANGE	ORDER	EDITTEXT	EXPLODE
DDATDEF	DATTEdit	ATTEDIT	DDATTEXT	CASCADE	TILEVERT	TILEHORIZ	ARRANGE

MODIFICATION / TOOLS / WINDOWING

ESC	NEW	OPEN
SAVE	SAVE	
VBA	DRAWING EXPLORER	
HELP	SAVE AS	PAGESETUP
TABLTOFF	EXPORT	PRINT
CALIBRATE	PREVIEW	PURGE
CLOSEALL	CLOSE	EXIT

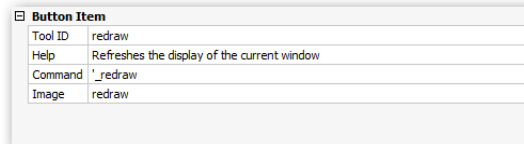
STANDARD

To attach commands, macros, or shortcut menus to buttons, follow the steps described in the for mouse buttons — just do your work in the **Click** section of Digitizer Buttons, as illustrated below.



Default puck button assignments

The properties for digitizer buttons are pretty similar to that of mouse buttons:



Puck button properties

The tablet overlay drawing provided by Bricsys is illustrated on the other pages. Use *overlay(cm).dwg* for metric drawings and *overlay(inch).dwg* for Imperial drawings.

Customizing the Quad

The trend in CAD user interface design is to move more of the action to the cursor, and so Bricsys developed the Quad interface to do just that: it allows us to select commands very near to the cursor.

The Quad is somewhat customizable, though sadly not as fully customizable as are menus or toolbars. BricsCAD uses workspaces to determine which groups of commands are displayed by the Quad, just as with the ribbon. This means that commands specific to sheet metal that appear in the Sheet Metal workspace, BIM commands for the BIM workspace, and so on.

Quad also uses entity recognition to determine which commands are suitable at the moment, especially when it comes to editing. Select 3D solid, and solid editing commands appear.

The content of the Quad's interface is changed through the Customize dialog box, and this is the subject of this chapter.

CHAPTER SUMMARY

This chapter covers the following topics:

- Understanding how the quad works
- Customizing the commands and groups displayed by the quad

QUICK SUMMARY OF QUAD VARIABLES

To turn the Quad cursor on and off, press the **F12** function key; there is no “Quad” command. The following variables control the look of the Quad.

QuadAperture — specifies the area to search around the cursor for entities, sized in pixels

QuadCommandLaunch — determines if Quad launches with the application

QuadCommandSort — specifies sort order of commands

QuadDisplay — toggles display of the Quad cursor

QuadExpandDelay — specifies the delay before expanding the Quad, in milliseconds

QuadExpandTabDelay — specifies the delay before expanding underlying buttons

QuadExpandGroup — specifies how groups expand

QuadGoTransparent — toggles whether Quad go transparency as cursor moves away

QuadHideDelay — specifies the delay to display the Quad following mouse movement, in msec

QuadHideMargin — specifies the area in which the cursor keeps the Quad alive

QuadIconSize — toggles the Quad between displaying small, large, or extra large icons

QuadIconSpace — specifies spacing between icons

QuadMostRecentItems — determines the number of most-recent items on the Quad

QuadPopupCorner — locates the Quad relative to cursor

QuadPropertyUnits — determines the automatic formatting of units when InsBase is not zero

QuadShowDelay — specifies the Quad’s display delay after an entity is highlighted

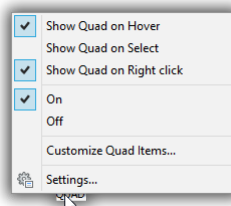
_QuadTabFlags — determines style of Quad

(NEW IN V20) TooltipDelay — specifies the delay before tooltips appear, in msec, in Quad, ribbon, and Properties

QuadWidth — specifies the width of the Quad, in columns

ON THE STATUS BAR

On the status bar, click **QUAD** to turn it on and off. Right-click the button to reveal this shortcut menu:



Show Quad on Hover — QuadDisplay = 1

Show Quad on Select — QuadDisplay = 2

Show Quad on Right-click — QuadDisplay = 4

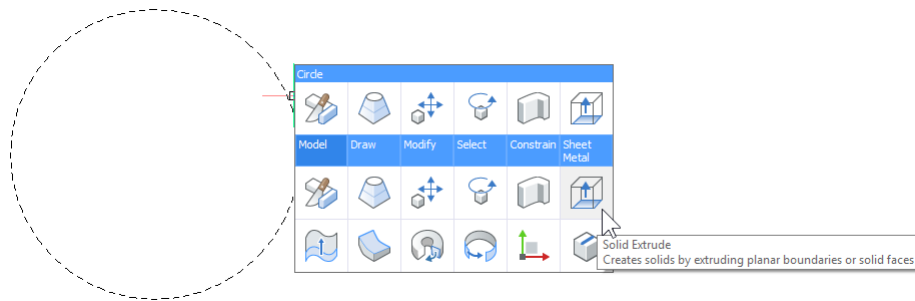
On / Off — toggles display of Quad, like clicking QUAD on the status bar (QuadDisplay = -1)

Customize Quad Items — displays Workspaces tab of Customize dialog box for changing the commands and groups displayed by the quad

Settings — goes to the Quad section of the Settings dialog box

ABOUT THE QUAD

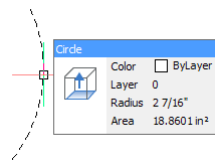
The Quad interface is unique to BricsCAD. It incorporates information about entities, along with drawing and editing functions into a rectangle that's right next to the cursor.



The Quad cursor, fully expanded

STEP 1: MOVE CURSOR ONTO AN ENTITY


Most of the time, the Quad is not visible; most of the time, you see the usual crosshair cursor. When, however, you pass the cursor over an entity, the entity highlights and the Quad appears as a single button. See figure below. (If you do not see the Quad, then turn it on clicking **QUAD** on status bar or pressing the **F12** function key.)




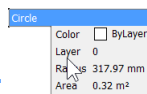
Initial appearance of the Quad

The initial button shows two things:

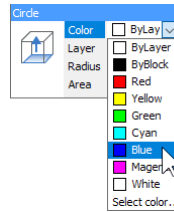
- ▶ **Last-used Command** — on the left is the command (shown by the icon) that you last accessed on the Quad
- ▶ **Rollover Tooltip** — to the right of the icon are several properties of the highlighted entity (if **RT** is turned on at the status bar)

Last-Used Command.  The single button is the icon of the last-used command. By clicking it, you can quickly repeat the last command multiple times.

Rollover Tooltip.  The name and the list of the entity's properties are together called a "rollover tooltip," because it appears when the cursor "rolls over" an entity. (If the rollover tooltip does not appear in the Quad, then click the **RT** button on the status bar to turn it on.)



By moving the cursor into the rollover tooltip, you can change the properties of the highlighted entity. Click on a property values, such as **Color**, to see a droplist of options.

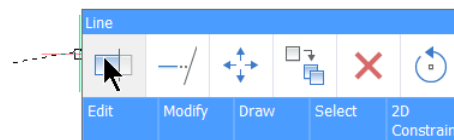


Changing the properties of the highlighted entity

You can change the properties listed by the Quad on a property-by-property basis. See Chapter 13 for more on using and customizing rollover tooltips (a.k.a. quick properties). I don't find rollover tooltips useful, and so tend to keep RT turned off.

Step 2: Expand the Quad

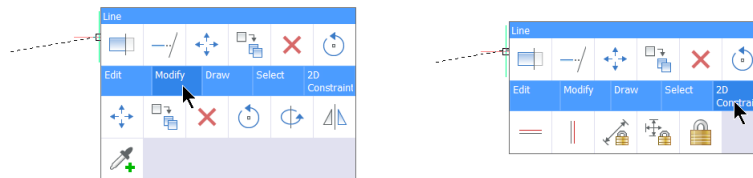
As you move the arrow cursor into the lonely last-command-used button, the Quad expands to show additional buttons. Usually, these are the commands that are commonly used with the highlighted entity. Click a button to execute its command.



Quad expanding when the cursor passes over it

Step 3: Move Into Groupings

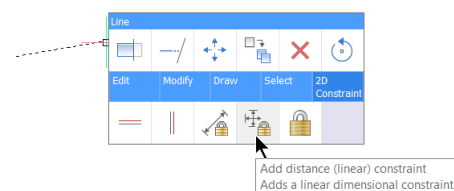
Below the first row of commands is a row of blue boxes. These are headings, and they hold groups of related commands. When you move the arrow cursor down into a blue box, such as **Modify**, more commands appear, in this case related to modifying entities. Some groups of commands are for operations common to *any* entity, while others might be specific to the entity that is highlighted.



Quad expanding further when the cursor across the blue headings

The commands you see in the Quad vary according to the workspace. See Chapter 14 on workspaces.

Identifying Icons. To see the meaning of an icon, pause the cursor over a button, and then read the description in the tooltip.

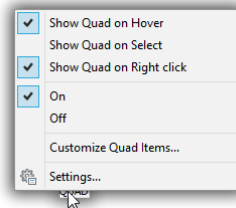


Viewing the meaning of a button

TUTORIAL: DRAWING WITH QUAD

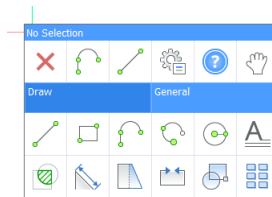
The Quad can start up drawing commands, as well as editing ones. The process, however, is different from using the Quad for editing, because there is no existing entity over which to hover! To draw with the Quad, follow these steps:

1. Turn on the **Show Quad on Right-click** option by right-clicking QUAD on the status bar, and then selecting the option.



Turning on Quad's ability to draw entities

2. Right-click in the drawing area. Notice that the Quad appears with drawing commands.



Quad displaying drawing commands

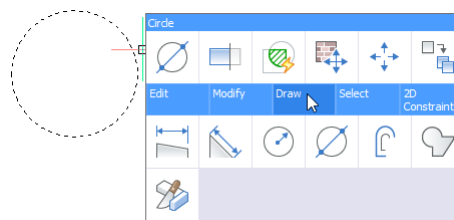
3. Select one, and then begin drawing the entity.

Tutorial: Dimensioning with Quad

The Quad is especially handy for dimensioning entities quickly. The speed is due to two presets: dimensioning commands use the Entity option by default, and the Quad knows which entity is selected.

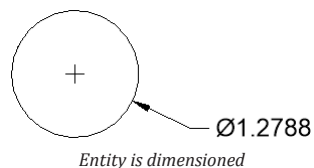
To dimension an entity, follow these steps:

1. Pause the cursor over the entity.
2. Move the cursor into the Quad's **Draw** section.



Selecting a dimensioning command from the Quad

3. Click a dimensioning command. Notice that it was dimensioned without you having to select the entity.

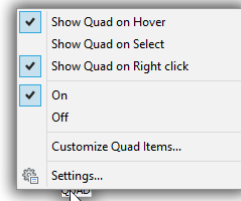


Entity is dimensioned

The same thing happens when dimensioning a line with two extension lines: all it takes is a single click.

MODIFYING THE QUAD'S BEHAVIOR

To make the Quad work differently from its default settings, take a look at the **Quad** section of the Settings dialog box. The fastest way to get there is to right-click **QUAD** on the status bar, and then choose **Settings** in the shortcut menu.



Accessing Quad settings quickly

In the Settings dialog box, you can change the look of the Quad through the following variables:

- ▶ Display the Quad only when an entity is selected (**QuadDisplay** variable = 2)
- ▶ Have the Quad pop up at a different location (**QuadPopUpCorner**)
- ▶ Make the Quad icons larger or smaller, tighter or wider (**QuadIconSize** and **QuadIconSpace**)
- ▶ Control the appearance of rollover properties (**RolloverTips** and **RollOverOpacity**)

See boxed text for the full list.

Customizing the Quad

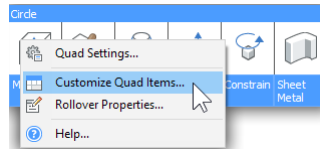
The Quad has been under a great deal of development from when it was first introduced. It seems to get new functions with every release. For instance, the Quad Reactors section was added to BricsCAD V17. But in terms of customization, V17 removed the Quad tab from the Customize dialog box, and moved its content to the Workspace tab, under Quad Groups. But with V18 the Quad tab returned!

Until V18, all parts of the Quad were hard-coded, meaning we could not change them. Despite its presence in the Customize dialog box, the only customization available was whether (or not) to display predefined groups.

With V18, full customization became available:

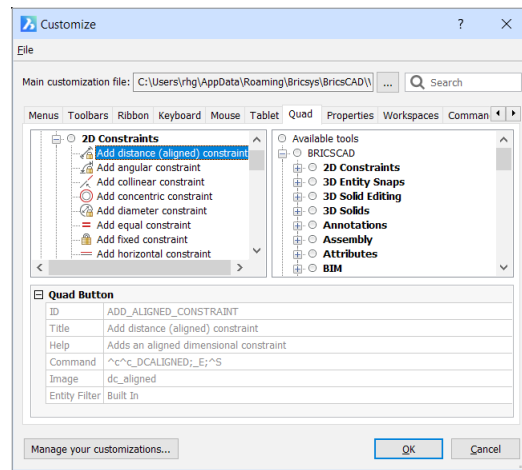
- ▶ You can add and remove buttons from groups
- ▶ Create and destroy groups
- ▶ You cannot, however, edit the properties of existing buttons, such as the **Title** or **Command** fields

To customize the Quad, enter the **Customize** command and then choose the **Quad** tab. Alternatively, right-click the Quad itself (or right-click QUAD on the status bar), and then choose **Customize Quad Items**.



Accessing the Customize dialog box from the Quad

When you do, the Customize dialog box appears at the Quad tab. Notice that the Quad Button properties are grayed out, indicating that they cannot be edited by you.

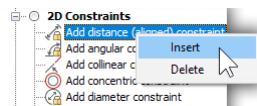


Quad tab showing uneditable properties for buttons

Tutorial: Customizing Quad Buttons

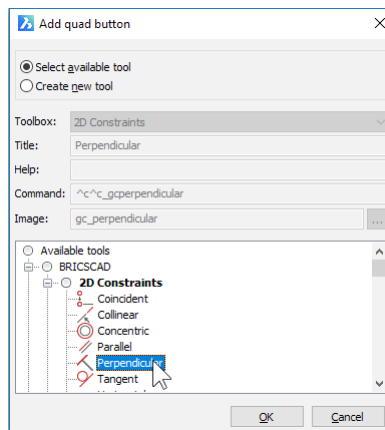
To add buttons to the Quad, follow these steps:

1. Right-click the name of a button, and then choose **Insert**.



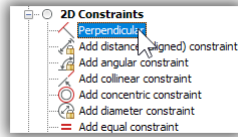
Right-clicking a Quad button

2. Notice the Add Quad Button dialog box. Choose a command (a.k.a. “tool”) from the list, and then click **OK**.



Dialog box for adding buttons to the Quad

- Back in the Customize dialog box, the added tool appears above the one you selected.



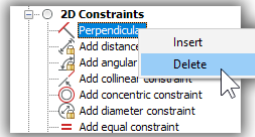
Tool added to the Quad

- Click **OK** to exit the Customize dialog box, and then check the Quad to make sure the new button works.

Deleting Buttons.

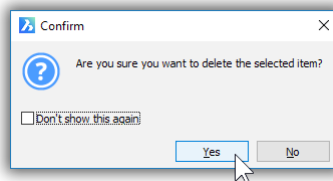
To remove buttons from the Quad, follow these steps:

- Right-click the name of a button, and then choose **Delete**.



Choosing a Quad button to remove

- Notice the warning dialog box:

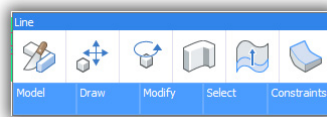


Agreeing to the question

Click **Yes** to finalize the removal.

CUSTOMIZING QUAD TABS

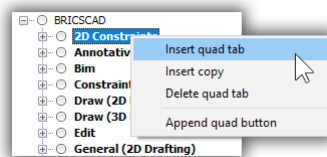
A *tab* is a group of buttons. Think of tabs as toolbars or ribbon panels. Typically, similar commands inhabit a tab, such as for editing or for constraints. The figure below shows tabs named **Model**, **Draw**, and so on.



Tabs in the Quad

To edit tabs in the Quad, follow these steps:

- In the Customize dialog box's Quad tab, right-click the name of a tab. Notice the options available:



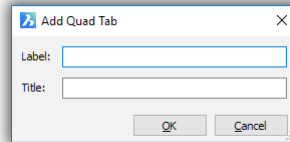
Right-clicking a Quad button

The options have the following meaning:

- ▶ **Insert Quad Tab** — adds a new tab, empty of commands
- ▶ **Insert Copy** — makes a copy of the selected tab, and then prompts for a new name
- ▶ **Delete Quad Tab** — removes the tab from the Quad
- ▶ **Append Quad Button** — adds a button to the current tab; see the previous tutorial

Tutorial: Adding Tabs. To add a tab to the Quad, following these steps:

1. Right-click a tab name, and then choose **Insert Quad Tab** from the shortcut menu.
2. Notice the Add Quad Tab dialog box. Enter a label and a title.

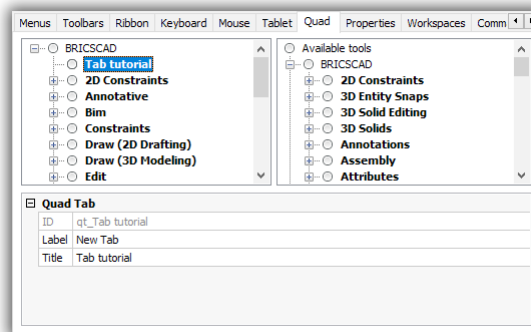


Naming the new tab

For this tutorial, enter the following:

Label **New Tab**
Title **Tab tutorial**

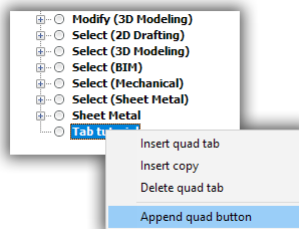
3. Click **OK** to dismiss the dialog box. Back in the Customize dialog box, notice that the new tab name is added above the one you selected.



New tab added to the list

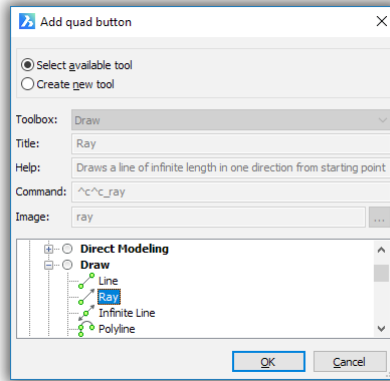
In the Quad Tab properties pane, the Label and Title fields appear with the names you gave them.

4. The tab is unpopulated, so add a button by following these steps:
 - a. Right-click the new tab, and then select **Append Quad Button**.



Adding a button to the new tab

- b. Notice the Add Quad Button dialog box. For this tutorial, choose a drawing command, such as **Draw > Ray**.



Selecting a drawing command

- c. Click **OK** to close the dialog box. Back in the Customize dialog box, notice that the new tab sports the new button.

At this point in the tutorial, we pause.

Where's My New Tab?

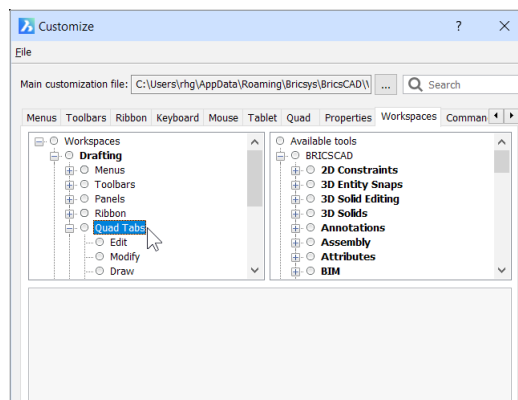
You might think that the new tab will now appear in the Quad, like a new menu or a new ribbon panel — but not so fast. Probably, it won't. That's because the appearance of a tab depends on how the *workspace* is set up.

Is the tab part of a workspace? The appearance of tabs is controlled by the workspace, as described next.

Tutorial: Turning On Quad Groups (Tabs)

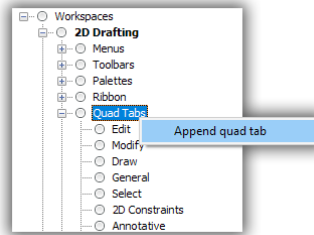
The **Workspace** tab of the Customize dialog box specifies which Quad tabs are allowed to be displayed. (Quad tabs used to be called “groups.”) No tab is seen until its name is added to the Quad Tabs section of a specific workspace. If you want the tab in every workspace, then you have to add it over again to each one.

1. In the Customize dialog box, click the **Workspace** tab.
2. Open the **Drafting** workspace by clicking the + (node) next to it.



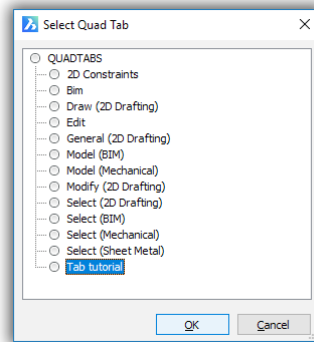
Quad Tabs section of the Workspaces tab

- Open the **Quad Tabs** section. Notice that the new tab you created is not listed. Similarly, when you scroll through the Available Tools listing, it is not there, either.
- Right-click **Quad Tabs**, and then choose **Append Quad Tab** from the shortcut menu.



Getting ready to append a tab

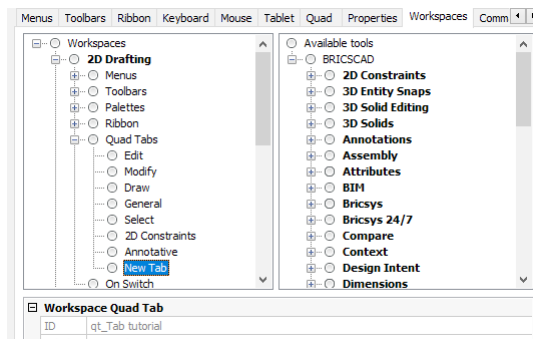
- Notice the Select Quad Tab dialog box, which lists the names of tabs you can add to the workspace. The tabs are listed in alphabetical order by their labels (such as “Tab tutorial”), rather than by their titles (such as “New Tab”), because labels are more descriptive.



Selecting the tab to add

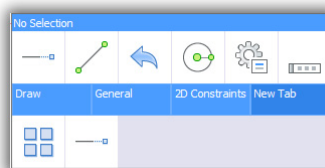
Select “Tab tutorial” and then click **OK**.

- Notice that the tab is added to the workspace, under the name of “New Tab.”



“New Tab” added to the 2D Drafting workspace

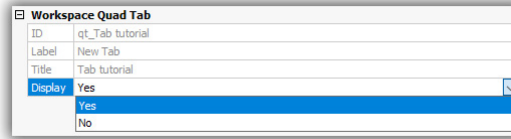
- Let’s see if the new tab appears in the Quad. Click **OK** to close the Customize dialog box.
- In the BricsCAD drawing area, right-click to display the Quad in drawing mode. Move the cursor to display the tabs, and then the content of the “New Tab” tab.



New Tab successfully added to the Quad

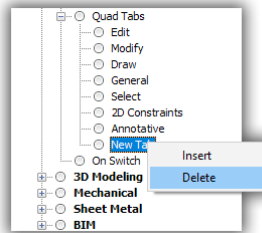
Toggle Quad Tabs

You have to add a tab to the Quad Tabs section to make it appear in the Quad. But you don't need to remove it to hide it. Instead, change the value of its **Display** parameter, as shown in the figure below:



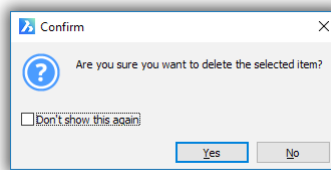
Toggle the visibility of a Quad tab

When you remove a tab from the Quad Tabs section in the Workspace tab...



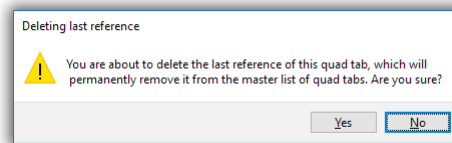
Deleting a Quad Tab from a workspace

...BricsCAD asks if you are sure:



Confirming the deletion

However, if you are removing the last reference of a Quad tab in all workspaces, then the situation gets really serious:



Are you r-e-a-l-ly sure?

For some reason, BricsCAD erases the tab (group) definition from the Quad tab, as well as from the workspace. This seems rather extreme to me, and I don't know why BricsCAD is programmed to do this.

A workaround is to create a dummy workspace, and store your custom Quad tabs in it.

ABOUT QUAD ENTITY FILTERS

Entity filters are what BricsCAD uses to determine which commands to display. An entity filter is a piece of programming code that reacts to (filters) the entity found under the cursor. For example, when the selected object is text, then the Quad shows commands specific to text — in addition to the All Entities section.

(Entity filters were named “Quad Reactors” in BricsCAD V17, and prior to V17 were known as “Custom.” In those releases, reactors could only be toggled on or off. Even today, in one shortcut menu they are called “Custom Alias,” and in one dialog box, they are called “Entity Alias.”)

You can customise filters for Quad commands (buttons) that you create. Quad items that are built by Bricsys cannot be changed, and so are shown in gray, as illustrated below.

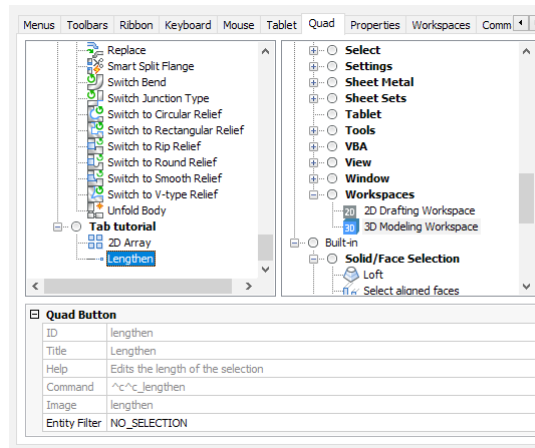
Quad Button	
ID	UNFOLD_BODY
Title	Unfold Body
Help	Unfolds a sheet metal body
Command	^c^c_smUnfold
Image	sm_unfold
Entity Filter	Built In

Filters cannot be edited for pre-made Quad buttons

Tutorial: Changing Entity Filters

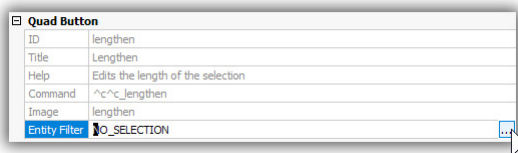
To specify the entities that your custom Quad button should recognize, follow these steps:

1. In the **Quad** tab of the Customize dialog box, select a command in your custom Quad Tab. For this tutorial, the Quad Tab is named “Tab tutorial.”

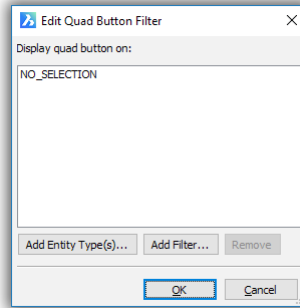


Select a Quad Button item

2. Look down to the Quad Button pane. Notice that you cannot change any property, except **Entity Filter**. (The other properties are grayed out.)
3. Select **Entity Filter**, and then click the **Browse** button (found at the right end of the field).

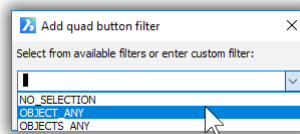


4. Notice the Edit Quad Button Filter dialog box, and that it lists the generic “No_Selection” filter.



Current state of custom filter

5. You can add a generic filter for all entities, and/or add specific entities to be filtered.
 - a. To add a generic filter, click the **Add Filter** button.
 - b. Notice that the new dialog box also sports the title, Add Quad Button Filter.



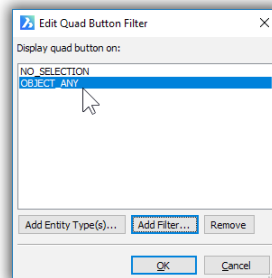
Choice of filters

The content of this dialog box is not documented, and so I am only guessing at what the options mean:

Available Filter	Meaning
Objects_Any	For common editing commands, such as Move and Erase; (this filter was formerly named ALL_ENTITIES)
Object_Any	For entity-specific editing commands, such as PLine and AttEdit
No_Selection	For drawing commands, where no entity is selected initially; (this filter was formerly named NO_SELECTION)

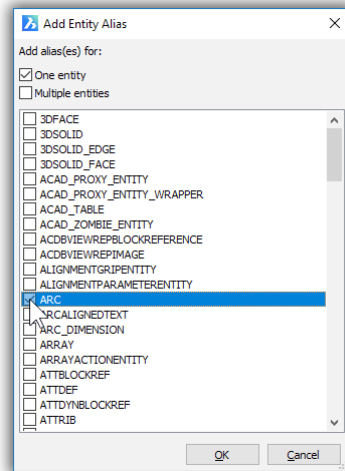
You can also type in the name of your own “custom filter,” which I think is meant for third-party developers.

- c. Click **OK** to close the dialog box. Notice that your selection is added to the list.



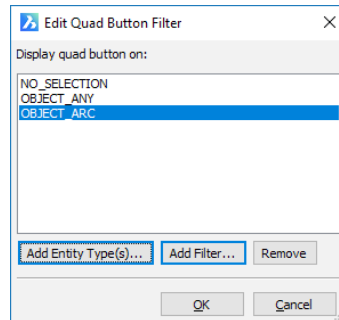
Filter added to the list

- d. Now click the **Add Entity Type(s)** button. Notice the Add Entity Alias dialog box.



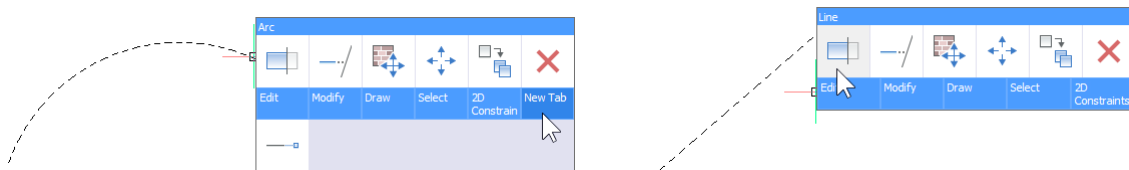
Choosing an entity

- e. Choose one or more entities to which your Quad button should react, such as **Arc**.
 f. Click **OK**. Notice that again the item is added to the filter list.



Arcs added to the filter list

- g. To remove a filter from the list, select it and then click the **Remove** button. Remove the No_Selection and Object_Any filters, leaving on the Object_Arc filter.
6. Click **OK** to close the Edit Quad Button Filter dialog box, and then click **OK** to close the Customize dialog box.
7. Test the change you make to the button's property by drawing a line and an arc. Your custom Quad tab appears when you pause the cursor over the arc, but not over the line.



Left: New Tab tab appearing for an arc; right: ...but not for the line

How the Quad Works. Or, How Does It Know What Entity Is There?

Pieter Clarysse of Bricsys explains: “The cursor makes use of C++ reactors to determine which entities are nearby. The icons that appear are appropriate to the entity.

“For example, if the cursor is near an intersection, it will display the Chamfer and Fillet commands. When the cursor is over a gap between two entities, it will have the Trim and Extend commands; the size of the gap (aperture) it recognizes can be adjusted in the Settings dialog box.”

To select the entity that is under the cursor in macros, use the **^S** metacharacter, which is unique to BricsCAD. It is unique, because ^S selects the entity without you needing to pick it. This allows for actions like dimensioning entities with a single pick: the pick consists of selecting the dimensioning command from the Quad.

Customizing Rollover Properties

All entities in CAD drawings carry properties, such as color and linetype. BricsCAD provides several ways to view the values of properties:

DbList command — lists names and properties of all entities in the drawing in the Prompt History window

List command — lists names and properties of selected entities in the Prompt History window

Properties panel — displays and changes most properties interactively in a panel

Rollover Tooltips (a.k.a. “quick properties”) — displays selected properties in the Quad

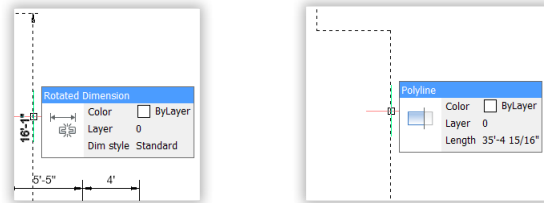
In this chapter, we concern ourselves with customizing rollover properties through the Customize dialog box.

CHAPTER SUMMARY

This chapter covers the following topics:

- Understanding how rollover tooltips work with the Quad
- Customizing the information displayed by rollover tooltips

Rollover properties are handy for seeing the properties of a single entity over which the cursor is hovering (rolling over). Rollover properties are rolled into the Quad: when you hover the cursor over an entity, a short list of its properties are displayed for you. Two examples are shown below.



Left: Properties of a linear dimension; right: ...and a polyline

While the DbList, List, and Properties can display properties of more than one entity, rollover properties are limited to a single entity. Despite the single-entity limitation, rollover properties are handy, because we can change the properties that are displayed right there in the Quad.

QUICK SUMMARY OF ROLLOVER PROPERTY SETTINGS

To turn on Rollover Tooltips, click the **RT** button on the status bar. There are no commands for rollover tooltips. The variables that control the operation and look of rollover tooltips are available through the Settings dialog box:

The **RolloverTooltips** variable determines when the properties appear:

RolloverTooltips	Properties are...
0	Properties not displayed by the Quad
1	Properties displayed while the cursor hovers over an entity
2	Properties displayed when the cursor enter the Quad's title bar

The **RolloverOpacity** variable determines the see-thru-ness of the tooltip:

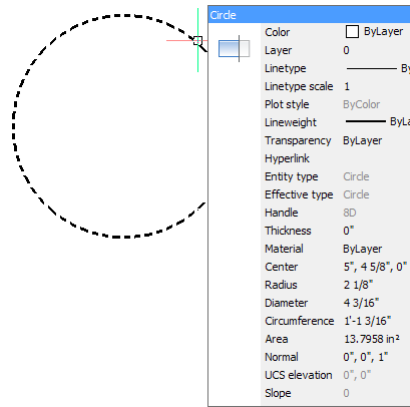
RolloverOpacity	Tooltip is...
100	Opaque (default)
10	Translucent (minimum value)

The **RolloverSelectionSet** variable determines the types of properties displayed::

RolloverSelectionSet	Properties displayed...
0	None
1	Only General properties (default)
2	All properties shared by the selected entities *

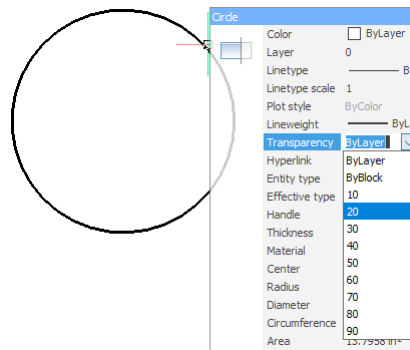
*) Bricsys notes that large selection sets may cause slow reaction

You can have the rollover display as many properties as you want; the limit is the height of your computer's screen.



Extreme example of a rollover displaying many properties of a circle

The properties displayed by the rollover tooltip can be edited. (Those shown in gray cannot; they are read-only.)



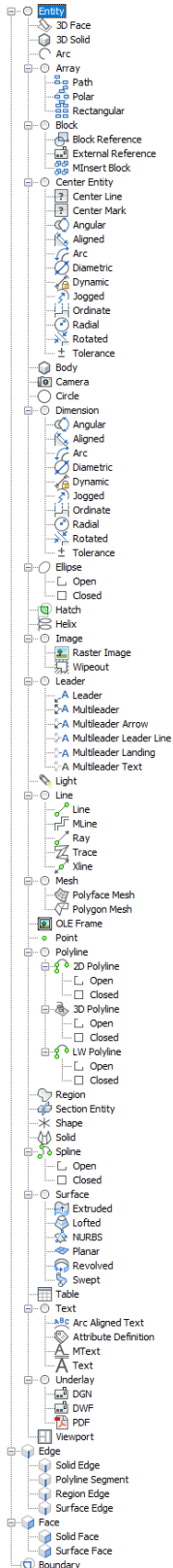
Changing the transparency of a circle

The display of rollover properties are turned on and off in these ways:

- ▶ Click the **RT** button on the status bar
- ▶ Change the value of the **RolloverTooltips** variable

There is no command that toggles rollovers. The `QuadIconSize` variable has no effect on the size of rollover properties. The properties content of the rollovers' are customized with the **Customize** command.

QUICK SUMMARY OF ROLLOVER PROPERTIES

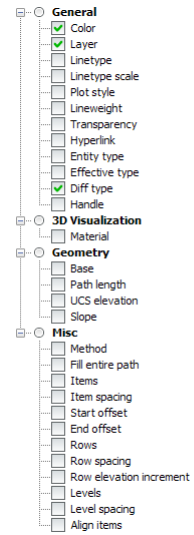


The properties displayed by rollover tooltips are changed through the Customize dialog box's **Properties** tab.

Shown at right are the entities and sub-entities available in BricsCAD. The properties of one entity, the path array, are shown to the far right.

There are a couple of things to be aware of:

- A green checkmark indicates the property is displayed by rollover tooltips.
- Some entities are “hidden” inside others. For example, multiline leaders are found inside the Leaders section, and mtext is found in the Text section.
- Parts of 3D entities, like edges and faces, have their own categories at the end of the list.



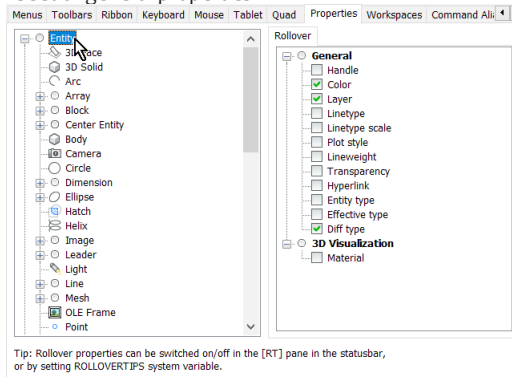
Customizing Rollover Properties

You use the **Properties** tab of the Customize dialog box to change the properties displayed by rollover tooltips. The customization is done on an entity-by-entity basis (with the one exception noted in the Tip below), so in most cases you need to take two steps:

Step 1. Select the entity of which you want to change the properties displayed; and then...

Step 2. Select the properties you want displayed for the entity by the Quad

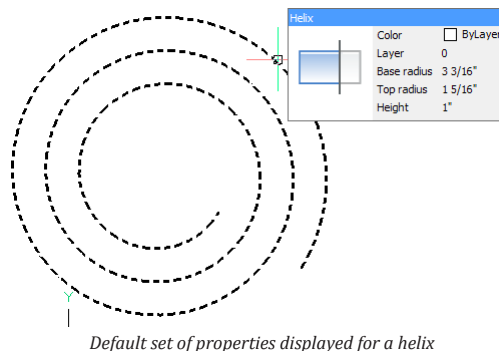
TIP The **Entity** “entity” is at the very top of the list, and has a special function. When you turn its properties on or off, it affects all entities. For example, choose **Entity**, and then click **Linetype** to turn it on. The Linetype property is now turned on for all entities in BricsCAD. This trick is handy when you want all entities to display the same set of general properties.



By default, all entities have most properties are turned off due to the overwhelming number of properties that are available! How overwhelming? The facing page lists all of the *entities* available in BricsCAD, while adjacent to the list are the *properties* for just one of them, a path array. Now, the number of properties varies wildly between entity types, from a just few to many: dimension entities, for instance, have nearly 100 properties each.

TUTORIAL: HOW TO CHANGE PROPERTIES DISPLAYED BY ROLLOVERS

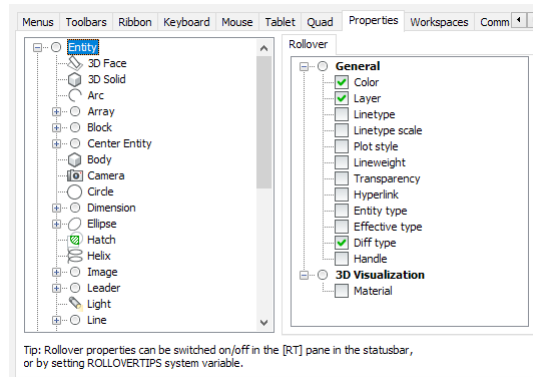
For this tutorial, you will change the properties of helices. The default properties for the helix entity are shown below: color, layer, base radius, top radius, and height. To customize the properties listed by the rollover tooltip, follow these steps.



Default set of properties displayed for a helix

In this tutorial, you turn off the Color property, and turn on the Total Length property.

1. Enter the **Customize** command to open the Customize dialog box.
2. Choose the **Properties** tab. Notice that there are two panes:

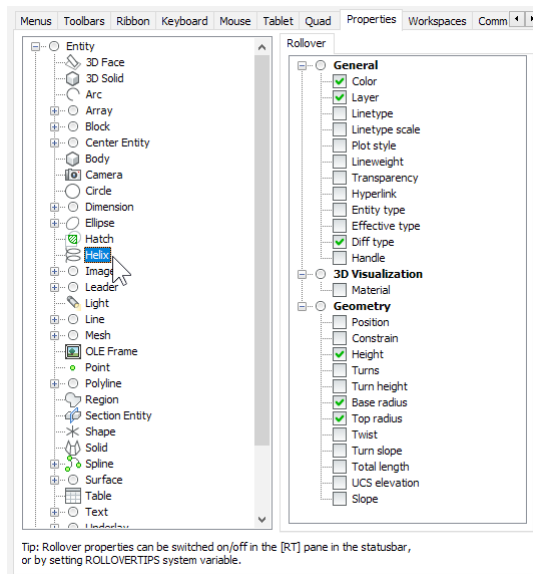


The Properties tab of the Customize dialog box

- ▶ **Entity** pane (at left) lists the names of all entities handled by BricsCAD
- ▶ **Rollover** pane (to the right) lists all properties that apply to the selected entity

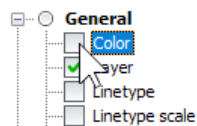
When no entity is selected, the properties pane displays only the General properties, those that are common to all entities.

3. In the entities pane, select **Helix**. Next door in the Rollover pane, notice that certain properties that are turned on by default for helixes.



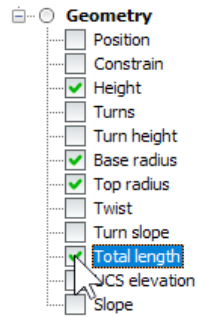
Properties for the helix

4. In the **General** section of the properties pane, click **Color** to turn it off (so that no check mark shows).



Turning off the Color property

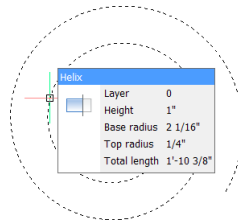
- In the **Geometry** section, click **Total Length** to turn it on (so that a check mark shows).



*Turning on the **Total Length** property*

- Click **OK** to exit the dialog box.
- Test your customization!
 - Draw a helix.
 - Pause the cursor over it.

Notice that the Quad no longer reports the color, but now shows the total length of the helix instead. (If the Quad does not appear, then click **QUAD** on the toolbar, or else press **F12** on the keyboard.)



Properties for a helix, as customized by you

Customizing Multiple UIs with Workspaces

BricsCAD is a general CAD engine designed to work with many different disciplines. It can be used for architectural modeling, for mechanical design, for mapping — all of which employ different sets of commands. When it comes to the user interface itself, you might prefer to work with toolbars or with the ribbon; you might want the drawing area shown by a gradient of colors or in a solid color.

Workspaces let you customize the user interface to your liking, and switched between them quickly. BricsCAD comes with several workspaces already defined for general drafting, 2D mechanical design, 3D general modeling, and 3D BIM modeling.

In this chapter, you learn how to customize workspaces to your liking.

CHAPTER SUMMARY

This chapter covers the following topics:

- Understanding how workspaces change the user interface
- Customizing workspaces

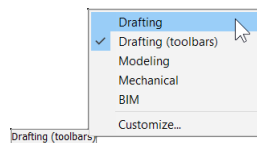
QUICK SUMMARY OF WORKSPACE COMMANDS & VARIABLES

The names of workspaces changed with BricsCAD V20:

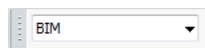
Old Workspace Name	New Workspace Name	Default Display
2D Drafting	Drafting	Ribbon
...	Drafting (Toolbars)	Menu bar and toolbars, no ribbon
...	Modeling	Ribbon
3D Modeling	Modeling (Toobars)	Menu bar and toolbars, no ribbon
BIM	BIM	Ribbon
Mechanical	Mechanical	Ribbon
Sheetmetal		(removed from V19)

The Sheet Metal workspace was removed from V15, returned with V16, and then removed again from V19. BricsCAD V17 disabled the **On Switch** parameter; is still present but is inoperative.

You can switch between workspaces with the status bar and a toolbar:



On the status bar, right-click the name of a workspace, and then choose another one



In the Workspaces toolbar, click the droplist and then choose the name of a workspace

There is no way to change the workspace from the ribbon or the menu bar. The U (undo) command does not reverse workspace changes.

COMMANDS

The following commands work with workspaces:

Workspace — saves, renames, deletes, and sets the current named workspace.

WsSave — the current user interface as a named workspace.

WsSettings — opens the Customize dialog box at the Workspaces tab (and not the Settings dialog box, oddly enough)

VARIABLES

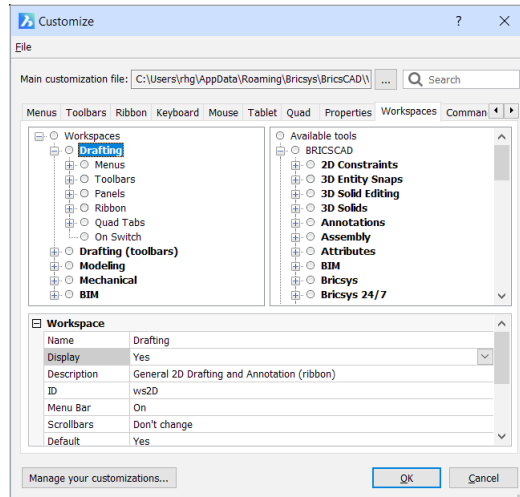
The following variables work with workspaces:

WsCurrent variable — specifies the name of the current workspace.

WsAutoSave variable — determines if changes to the user interface are save to the workspace.

Workspace Customization Elements

Workspaces are created and modified in the **Workspaces** tab of the Customize dialog box. It lists the user interface elements that can be displayed, and additional options determine if the elements actually are displayed by the workspace. In this chapter, we learn how this works.



Elements of workspaces displayed by the Customize dialog box

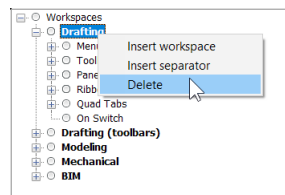
ADDING AND REMOVING WORKSPACES

You can add and remove workspaces by right-clicking a workspace name in the Customize dialog box, and then choosing an option from a shortcut menu.

Removing Workspaces

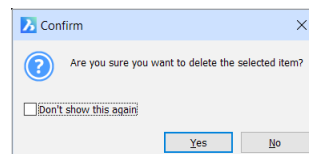
To remove a workspace, follow these steps:

1. Right-click the workspace name and then choose **Delete**.



Removing a workspace

2. BricsCAD displays the “Are You Sure?” dialog box:



Getting a second chance

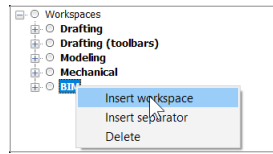
Answer **Yes** or **No**.

You recover the removed element with the Manage Your Customizations button, as described in Chapter 9.

Adding Workspaces

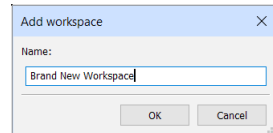
To add a workspace, follow these steps:

1. Right-click the name an existing workspace name, and then choose **Insert Workspace**.



Starting to insert a new workspace

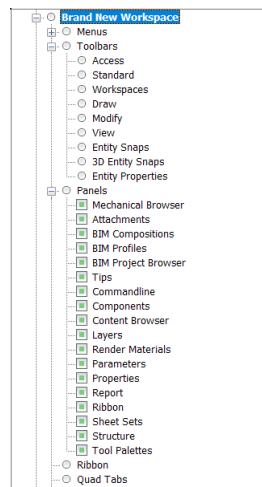
2. BricsCAD displays the Add Workspace dialog box for you to give it a name. If you click **OK** without entering a name, the workspace is not created.



Naming the new workspace

Notice that it is inserted *above* the workspace that you selected. The new workspace contains the following elements:

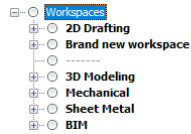
- ▶ All menus
- ▶ The set of toolbars shown below
- ▶ The panels shown below, all set to “Don’t Change”
- ▶ No ribbon elements
- ▶ No Quad items



Default elements of a new workspace

About Insert Separator

The **Insert Separator** option adds a line as a row of dashes. It is visible only in the Customize dialog box; the line does not appear in the Workspace toolbar or on the status bar.



Separator added to the Workspaces tab

You use the separator as a guide to visually separate groups of workspaces.

TOGGING THE DISPLAY OF UI ELEMENTS

The Workspaces tab's primary purpose is to toggle the display of user interface elements on and off independently in each workspace. You turn elements on and off by these methods:

Method 1. Include the UI element in the workspace

Method 2. Toggle the **Display** parameter to On or Off (not all elements offer this parameter)

In the following section, we look at both methods.

Workspace Property Toggles

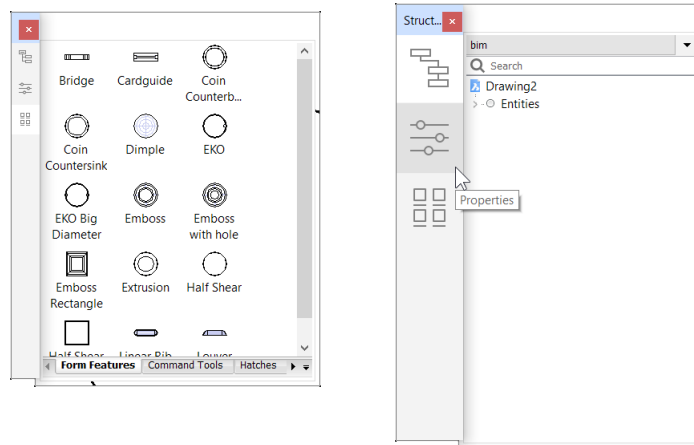
When you choose the name of a workspace in the Customize dialog box, the Workspace pane displays a long list of properties ([greatly expanded in V20](#)). You can think of these properties as master toggles.

Workspace	
Name	new
Display	Yes
Description	
ID	wsnew
Menu Bar	Don't change
Scrollbars	Don't change
Default	No
Stack Type	Fixed resizeable panelset
Panel Button Size	Small
Tool Button Size	Small
Toolbar Margin	
Tool Padding	
Delete Tool	On
DMAUDIT Detail Level	
Extrude Mode	
DMPUSHPULL Subtract	On
Generate Associative Drawings	On
Report Panel Mode	Classic Theme
Selection Modes	
Sheet Set Manager Auto Open	On
Structure Tree Config	
Components Config	
Warning Message Level	

Default properties of a new workspace displayed by the Customize dialog box

Workspace properties have the following meaning:

Workspace Property	Meaning
Name	Name of the workspace displayed in the workspace list in the status bar and by the Workspace toolbar (reported by the WsCurrent variable)
Display	Determines if the workspace name is displayed by the status bar and toolbar droplists: <ul style="list-style-type: none"> • Yes • No <p>(Note: When this property is turned off, the name of the workspace is still recognized by the Workspace command.)</p>
Description	Help-like description displayed in the status bar
ID	Identifies the element in the CUI file, which stores all of these customizations; the “ws” prefix marks this element as a workspace
Menu Bar	Toggles the display state of the menu bar (also toggled by the MenuBar variable): <ul style="list-style-type: none"> • On shows the menu bar when entering this workspace • Off hides the menu bar when entering this workspace • Don't Change the display state when entering this workspace
Scrollbars	Toggles the display of scroll bars (also toggled by the ScrollBars variable): <ul style="list-style-type: none"> • On shows the scroll bar when entering the workspace • Off hides the scroll bar • Don't Change the display state when entering this workspace
Default	Toggles whether this workspace is the default one when BricsCAD starts: <ul style="list-style-type: none"> • Yes shows this workspace when BricsCAD starts • No does not show this workspace
Stack Type	Determines how panels are displayed (reported by read-only StackPanelType variable): <ul style="list-style-type: none"> • Fixed resizable panelset prevents the panel from collapsing (0) • Collapsible panelset allows the panel to collapse into a set (1) • Flyout panelset places the panel in a flyout; see figure below (2)



Left: Small panel icons; right: extra-large panel icons

Panel Button Size	Specifies initial size of buttons on panels (reported by read-only PanelButtonSize variable): <ul style="list-style-type: none"> • Small displays 16x16 icons (0); see figure above • Large displays 24x24 icons (1) • Very Large displays 32x32 icons (2)
--------------------------	---

Tool Button Size	Specifies initial size of buttons on toolbars (reported by read-only ToolButtonSize variable): <ul style="list-style-type: none"> • Small displays 16x16 icons (0) • Large displays 24x24 icons (1) • Very Large displays 32x32 icons (2)
Toolbar Margin	Specifies margin above and below each toolbar; measured in pixels (reported by read-only ToolbarMargin variable)
Tool Padding	Specifies margin between icons on each toolbar; measured in pixels; (reported by read-only ToolIconPadding variable)



Above: Toolbar with normal spacing
Below: Toolbar with padding = 5 and margin = 5



TIP Use the **Toolbar Margin** and **Tool Padding** values to increase the space around toolbar buttons, which can make them easier to touch with a finger on a touchscreen monitor. For this, you could create a workspace named “Drafting (Touchscreen).”

(NEW IN V20) The following workspace properties are specific to variables used by the Mechanical edition of BricsCAD. These settings are not documented by Bricsys.

Delete Tool	Specifies what to do with tool entities following the Subtract command: <ul style="list-style-type: none"> • 0 does not delete tool entities • 1 deletes tool entities
DMAUDIT Level	Specifies messages to display: <ul style="list-style-type: none"> • 0 reports errors • 1 ignores dynamic range errors • 2 ignores sliver faces • 3 ignores both
Extrude Mode	Determines what happens during the Extrude command: <ul style="list-style-type: none"> • 0 unites newly created solids with existing ones • 1 creates new solids when extruding a contour laying on a solid • 2 subtracts newly created portions from intersected solids • 4 do not modify intersecting solids
DMPUSHPULL Subtract	Specifies whether to enable subtract mode in DmPushPull command: <ul style="list-style-type: none"> • 0 does not subtract • 1 subtracts
Generate Associative Drawings	Toggles associativity of generated drawings with the 3D model.
Report Panel Mode	Determines the look of the Report panel: <ul style="list-style-type: none"> • 0 Classic mode (dockable window) • 1 Modern mode (transparent panel) • 2 Hidden mode (hidden in status bar)
Selection Modes	Specifies the subentities to highlight during entity selection: <ul style="list-style-type: none"> • 0 selects the entire 3D model • 1 select edges • 2 select faces • 4 detect boundaries • 8 select vertices

Sheet Set Manager Open Toggles whether the Sheet Set Manager panel is opened automatically when a drawing from a sheet set is opened.

Structure Tree Config Names the .cst structure tree configuration file to use

Components Config Names the .ccf components configuration file to use

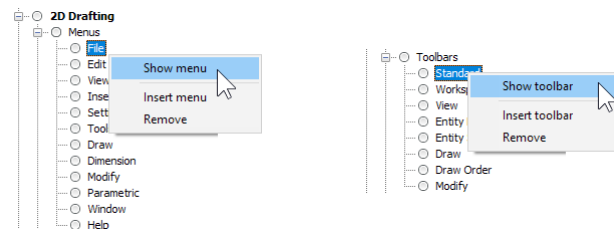
Warning Message Level Determine which warning messages to display; see figure below:

Warning messages	0xFFFFFF (1048575)
1	<input checked="" type="checkbox"/> Selecting 3D context with hardware rendering off
2	<input checked="" type="checkbox"/> Modifying tool property in Customize dialog
4	<input checked="" type="checkbox"/> Deleting sheet custom property
8	<input checked="" type="checkbox"/> Moving entities to frozen or off layer
16	<input checked="" type="checkbox"/> Saving to previous version not supporting some entities
32	<input checked="" type="checkbox"/> Detecting modified attachments when opening the parent drawing
64	<input checked="" type="checkbox"/> Creating new layer not matching the current layer filter
128	<input checked="" type="checkbox"/> Render: Tile sizes between 4 and 127 are processed as 128
256	<input checked="" type="checkbox"/> Expanding category mass in properties panel
512	<input checked="" type="checkbox"/> Deleting an item in Customize dialog
1024	<input checked="" type="checkbox"/> Publish: Save sheet list
2048	<input checked="" type="checkbox"/> Delete layouts in Page Setup Explorer
4096	<input checked="" type="checkbox"/> Mass properties calculation takes long time
8192	<input checked="" type="checkbox"/> Array editing state
16384	<input checked="" type="checkbox"/> Incompatible units
32768	<input checked="" type="checkbox"/> Modified block definition will cause all related block references update
65536	<input checked="" type="checkbox"/> A Data Link has changed, Any tables using this link may need to be updated
131072	<input checked="" type="checkbox"/> VIEWBASE usage for architectural drawings

Options of the warning messages variable

Show Menus

Several shortcut menus in the Workspaces tab have an option called “Show,” such as **Show Menu** and **Show Toolbar**.



Shortcut menu in Workspaces tab to access a specific menu or toolbar

These options have nothing to do with the visibility of menus, toolbars, or other UI elements; instead, it switches (in this case) to the **Menu** tab of the Customization dialog box and to the specific menu (“File,” in the figure above), so that you can customize the content of the File menu.

TOGGLING VISIBILITY OF UI ELEMENTS

The whole purpose of workspaces is to determine what UI elements are displayed, and how. When these elements are listed in the Workspaces tab, then they will (probably) appear in the workspace — unless the **Display** property is set to “No.” But there are some subtleties for each type of elements. Let’s go through them.

Toggling Menus

The **Menus** node determines which menu drop-downs are displayed, such as File and Edit. When a menu name appears in this list, it is displayed by the workspace. Toggle the display of the menu bar through the Properties pane of the workspace

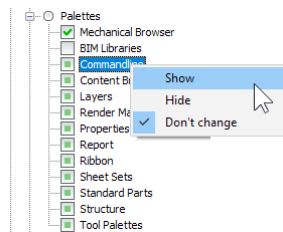
To make changes to a menu, use the Show Menu shortcut (as described above) to get to the Menu tab; see Chapter 6.

Toggling Toolbars

The **Toolbars** node determines which toolbars are displayed by the workspace. When a toolbar name appears in this list, it is displayed by the workspace. You can change the content of toolbars with the Toolbars tab; see Chapter 7.

Toggling Panels

The **Palettes** node determines which panels (a.k.a. palettes or bars) are displayed through the use of a three-way toggle:



Shortcut menu for palette items

- means the panel will be displayed when switching to this workspace
- means the panel is not displayed
- means the panel display state is not changed (i.e., if off when entering the workspace, then it stays off)

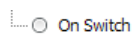
Toggling Ribbons

The **Ribbon** node determines which ribbon tabs are displayed by the workspace. When the name of a ribbon's tab appears in this list, then it is displayed by the workspace. You change the content of tabs and panels through the Ribbon tab; see Chapter 9

Toggle the Quad

The **Quad Tabs** node determines which quad groups and commands are displayed. When a Quad tab (a.k.a. Quad group) name appears in this list, it is displayed by the workspace, unless the Display property = No. You change the content of quad groups through the Quad tab; see Chapter 12.

The **On Switch** node was disabled with BricsCAD V17, and acts now only as a placeholder.



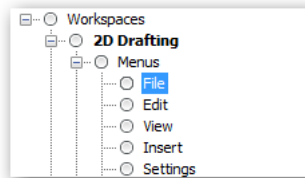
On Switch acts like the appendix in the human body

FINE-TUNING UI ELEMENTS

Earlier you saw the properties that are available for workspaces. Some user interface elements have additional properties that you can adjust, specifically menu items, toolbars, palettes, ribbon tabs, and Quad tabs.

Workspace Properties for Menus

When you choose the name of a menu, such as “File” ...



File menu selected in Workspaces tab

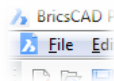
...then BricsCAD plays these properties:

Workspace Menu	
Menu Group	BRICSCAD
ID	mnFile
Display	Yes
Title	&File
Diesel	

Properties of a dropdown menu

Here is the meaning of the properties:

Property	Meaning
Menu Group	Name of the menu group
ID	Identifies the element in the CUIX file; “mn” prefix marks this element as a menu.
Display	Determines if the menu is displayed by the menu bar: <ul style="list-style-type: none">• Yes• No
Title	Name of the menu displayed on the menu bar; the & indicates the underlined letter for Alt -key shortcut access. To access menus with a keyboard, hold down the Alt key; notice the underlined characters on the menu bar, and then press the underlined letter to access the menu, such as f for File menu, as illustrated below.
Diesel	Executes Diesel code when the user selects the men



Properties of Toolbars

When you choose the name of a toolbar, such as “Standard,” BricsCAD displays these properties:

Workspace Toolbar	
Menu Group	BRICSCAD
ID	tbStandard
Display	Yes
Position	Top
Row	0
Column	0
X	
Y	
Title	Standard

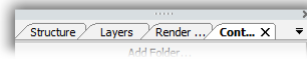
Properties of a toolbar

Property	Meaning
Menu Group	Name of the menu group
ID	Identifies the element in the CUIX file; “tb” prefix marks this element as a toolbar
Display	Determines if the toolbar is displayed when the workspace is opened: <ul style="list-style-type: none"> • Yes • No
Position	Determines the toolbar’s initial location: <ul style="list-style-type: none"> • Floating — toolbar is located using the X and Y properties • Top — toolbar is attached to the top of the drawing area • Left — toolbar is attached at the left of the drawing area • Bottom — toolbar is attached to the bottom of the drawing area • Right — toolbar is attached at the right of the drawing area
Row	Specifies the number of toolbar rows
Column	Specifies the number of toolbar columns
X	Locates the top of floating toolbars; the horizontal measurement is in pixels from the top of the computer screen. Floating toolbars can float outside of the BricsCAD window
Y	Locates left edge of toolbar, measured in pixels from the left edge of the computer’s screen
Title	Name of the toolbar displayed on the title bar of floating toolbars

Properties of Panels

When you choose the name of a panel such as “Command Line,” BricsCAD plays the properties listed below. The same properties are available to every panel. Palettes and bars are renamed “panels,” but the words ‘palette’ and ‘bar’ still appear sometimes in the BricsCAD program and documentation.

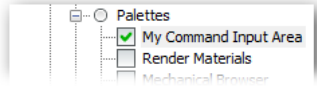
TIP When panels are CDOCK’ed (center docked), they overlap. Because the topmost panel hides the others, BricsCAD automatically display tabs so that the user can switch between them.



The **Stack Z Order** parameter determines the order in which the tabs appear.

Workspace Palette	
ID	MATERIALSBROWSER
Title	Render Materials
Display	Hide
State	Dock Left
Stack ID	LDOCK
Stack Z Order	3
Dock Column	4
Dock Row	0
Dock Width	300
Dock Height	916
Float Left	
Float Top	
Float Width	
Float Height	

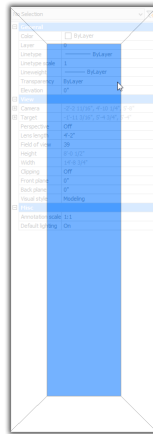
Properties of a panel, the Materials Browser panel in this case

Property	Meaning
ID	Identifies the name of the element in the CUIX file; cannot be edited by users
Title	Name of the panel displayed in the Customize dialog box; a changed title has no effect on the name displayed by the panel's title bar.
	 <p><i>Changing the panel Title from "Command Line" to "My Command Input Area"</i></p>
Display	<p>Determines if the panel is displayed by the workspace:</p> <ul style="list-style-type: none"> • Show — shows the panel when users switches to workspace • Hide — does not show the panel when users switches to workspace • Don't Change — show if visible in previous workspace; keep hidden, if not
State	<p>Determines default location of the panel when user switches to this workspace:</p> <ul style="list-style-type: none"> • Floating — floats anywhere at coordinates defined by Float parameters • Dock Top — docks to top of drawing area as defined by Dock parameters • Dock Left — docks to left of drawing area • Dock Bottom — docks to bottom of drawing area • Dock Right — docks to right of drawing area • Don't Change — show if visible in previous workspace; keep hidden, if not
Stack ID	<p>Locates the panel when stacked:</p> <ul style="list-style-type: none"> • LDOCK — docked to the left of the stack • RDOCK — docked to the right of the stack • TDOCK — docked at the top of the stack • BDOCK — docked at the bottom of the stack • CDOCK — docked on center of the stack, overlapping other panels
Stack Z Order	<p>Determines which panel is on top of other panels when center stacked (CDOCK):</p> <ul style="list-style-type: none"> • 0 — Highest priority (panel appears on top of other, higher-numbered panels)
Dock Column	Position of the panel (relative to other panels) when docked to the left or right
Dock Row	Position of the panel (relative to other panels) when docked to the top of bottom
Dock Width	Width of the panel when docked at left or right; measured in pixels.
Dock Height	Height of the panel when docked at top or bottom; measured in pixels
Float Left	Left edge's starting location of a floating panel; 0 = left edge of the main monitor
Float Top	Top edge's starting location of a floating panel; 0 = top edge of the main monitor
Float Width	Width of the panel when floating; measured in pixels
Float Height	Height of the floating panel; measured in pixels
Transparency	<p>Determines the level of translucency of the panel:</p> <ul style="list-style-type: none"> • 0 Fully opaque (default) • 90 Maximum translucency (nearly transparent)

TIPS When you assign the same **Stack Z Order** number to two or more panels, BricsCAD changes one of the duplicated numbers automatically. For instance, assign Stack Z Order = 2 to Layers and Rendering, and one of them will be changed to 3.

The **Float Left** and **Top** parameters apply to the position of the panels when using a multi-monitor setup.

When you move a panel to the left edge of the drawing area, BricsCAD previews the five possible dock locations, as illustrated below: left, right, top, bottom, or center.



Panel preview

Properties of Ribbon Tabs

When you choose the name of a ribbon tab, such as “Home,” BricsCAD displays these properties for the tab:

Ribbon Tab Reference	
Menu Group	BRICSCAD
ID	rtHome2D
Label	Home
Title	Home 2D
Key Tip	H

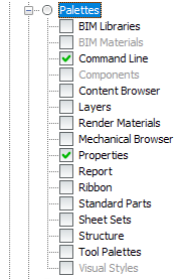
Properties of a ribbon tab

Property	Meaning
Menu Group	Name of the menu group.
ID	Identifies element in the CUIX file; “rt” prefix marks this element as a ribbon tab
Label	Identifies the element in the Customize dialog box
Title	Names the tab as displayed on the ribbon
Key Tip	Specifies letter to use for Alt -key shortcut access; not implemented.

TIP If key tips were implemented, it would work like this: hold down the **Alt** key; notice the characters in the tooltips, then press the letter on the keyboard to access the tab, such as **h** for Home tab.

Quad Reactors were removed from BricsCAD V18, as were the properties of Quad Tabs (Groups) from the Customize dialog box.

TIP When items are shown in gray and are turned off, such as **BIM Libraries** in the figure below, it means that an extra license needs to be purchased, or that the panel has not yet been implemented, such as Visual Styles.



Properties of Quad Items

The only aspect of Quad items that can be customized is whether they appear in the current workspace. When you choose the name of a Quad item (under a Quad Tab name), BricsCAD displays these properties:



Properties of a Quad item

Property	Meaning
Title	Name of the group displayed by the Quad; read-only
Label	Name of the Quad tab; read-only
Title	Named displayed by the Quad tab; read-only
Display	Determines if the tab is displayed by the quad in this workspace: <ul style="list-style-type: none"> • Yes • No

The job of creating Quad groups was moved from this tab to the new Quad tab in BricsCAD V18; see Chapter 12.

PART III

Other Customizations in BricsCAD

Designing Tool & Structure Panels

The purpose of the Tool Palettes panel is to provide access to collections of commands and drawing elements you use often, such as specific hatch patterns. Instead of rummaging through a variety of commands and dialog boxes, you drag commonly-used items from this centralized palette into the drawing.

The purpose of the Structure panel is to show you the *structure* of the drawing: which entities are connected to which. You can customize the way it displays the structure.

In this chapter, you learn how to customize the Tool Palettes and Structure panels.

CHAPTER SUMMARY

This chapter covers the following topics:

- Customizing icons and commands in Tool Palettes panel
- Importing palette files from AutoCAD
- Organizing palettes through groups
- Customizing the content of Structure panels

PANEL COMMANDS AND VARIABLES

RELATED COMMANDS

StructurePanel and **StructurePanelClose** — open and close the Structure panel displaying tree structure of the drawing content

StructurePanel opens the StructureTree Configuration File dialog box, prompting you to select a .cst (Configure Structure Tree) file; when you click **Open**, the Structure panel is opened and displays the configuration defined by the .cst file.

ToolPalettes and **ToolPalettesClose** — opens and closes the Tool Palettes panel

-ToolPanel — shows, hides, and toggles the visibility of specified panels through the command line

TpNavigate — loads tool palettes and groups through the command line

RELATED SYSTEM VARIABLES

ToolPalettePath — specifies the path to the folder holding .xtp files

TpState — reports whether the tool palettes panel is open (read-only)

StructureTreeConfig — specifies the CST file to use to configure the Structure panel

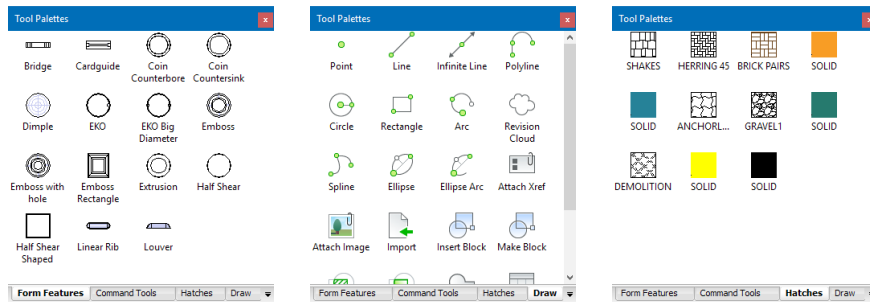
RELATED FILE TYPES

CST — files that configure structure trees

XTP — files that stored Tool Palette data, short for “xml tool palette”

About the Tool Palettes Panel

A most useful tool to CAD operators and managers is that the Tools Palettes panel. To show you what the Tool Palettes panel looks like, the four default ones are illustrated below. From left to right, we have one filled with 3D form features, the next with commands, and the last with hatch and fill patterns.



Left to right: Palettes provided by default in BricsCAD

Here is how you work with the icons shown by each palette:

- ▶ **Form Features** tab — drag a feature onto a sheet metal part
- ▶ **Command Tools** tab — click an icon to run the associated command, such as the Line command
- ▶ **Hatches** tab — drag one of the hatch or fill pattern icons into a closed object in the drawing

The Tool Palette panel is displayed by pressing **Ctrl+3** (**Cmd+3** on the Mac) or by entering the **ToolPalettes** command. It shows the same group of palettes in every drawing that is opened. Indeed, BricsCAD has the ability to provide one set standardized of Tool Palettes for an entire office.

ToolPalettes does not, unfortunately, support blocks. Instead, use the Components panel with the **ComponentsPanelOpen** command.

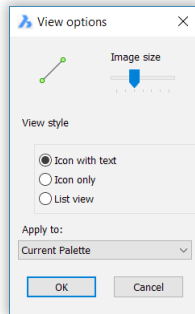
(*Form features* are sheet metal features that mimic applying a forming tool to the sheet metal, such as bridges, louver, and embosses. They inserted from built-in or user-defined libraries; BricsCAD recognizes form features in imported geometry. Form features are listed in the Mechanical Browser pane with their parameters; they can be edited directly or parametrically through Properties panel. Each feature is represented by a .dwg file in the *C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\DesignLibrary\SheetMetal\FormFeatures* folder.)

The user interface of the Tool Palettes panel is subtle, with numerous options “hidden” in shortcut menus all over the palette. So, a large part of this chapter exposes these shortcut menus to you.

TIP You can use the right-click menu to add components from files listed in the **Folders** tab of the BricsCAD Drawing Explorer to the current tool palette.

QUICK SUMMARY OF VIEW OPTIONS

The View Options dialog box controls the size and style of icons. To access the dialog box, right-click the current tab in the Tool Palettes, and then choose **View Options** from the shortcut menu.



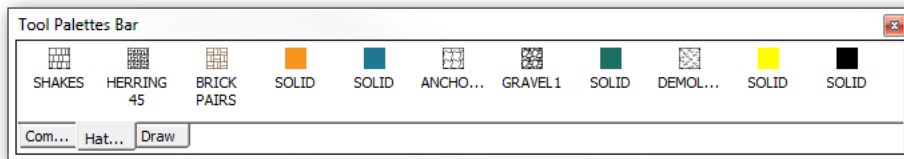
Adjusting the look of palette icons

Image Size — slider changes the size of the icons, from smaller to larger

View Style — switches between showing icons with labels (text), or showing icons only, or or text-only

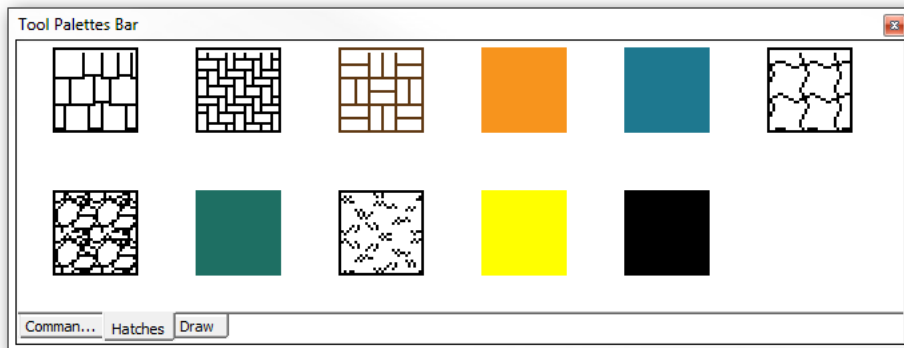
Apply to — applies the changes to the current tab or to all tabs

The figures below shows the effects of the Image Size and View Style options. Here is the smallest icon size, with text labels:



Small icons with labels

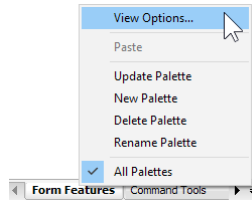
And here you see large icons with no labels. Removing labels squeezes in more icons, but may make it harder to know the purpose of them.



Large icons with no labels

Navigating Tools Palettes

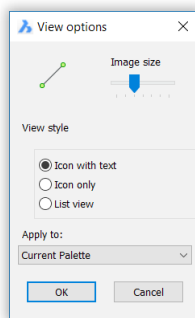
Different shortcut menus appear in the Tool Palettes panel, depending on which tab you right-click. I recommend that first off you right-click the tab of the current palette, because it displays the largest number of options, and so is the most useful:



Right-clicking the current (or top-most) tab

Clicking one of the other tabs gets you an abbreviated version. Here is the meaning of the options:

View Options displays a dialog box for setting the size and look of icons, along with descriptions; see the figure below and the boxed text on the next page for the meaning of the functions



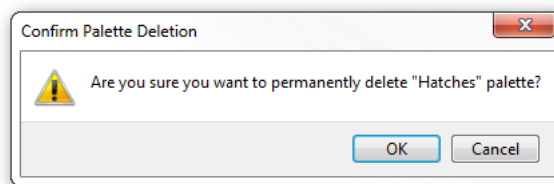
View options for all palettes

Paste pastes data from the Clipboard into the palette; available only when the Clipboard contains data appropriate to the Tool Palette, such as a block definition

Update Palette updates the image of the icons, should a source (like a block definition) have changed

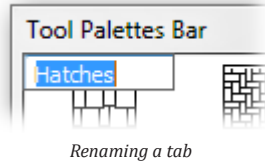
New Palette creates a new blank tab, and then prompts you to name it

Delete Palette warns against deleting the tab, and then removes it after you answer in the affirmative.



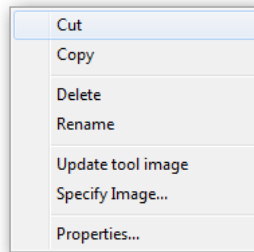
Deleting a tab

Rename Palette renames the selected tab; see figure below:



ICON CUSTOMIZATION

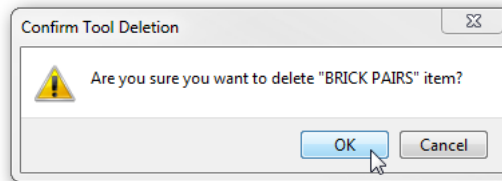
When you right-click a tool's icon, BricsCAD shows the following shortcut menu. Depending on which icon you right-click, the shortcut menu may show fewer options:



Shortcut menu for individual tools

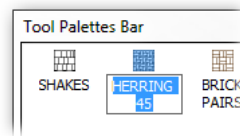
Cut and **Copy** — place the tool on the Clipboard (Cut also deletes it from the palette); you can then right-click the palette and select Paste

Delete — removes the tool after you affirm the questioning dialog box:



Affirming the deletion of a tool

Rename — allows you to rename the tool, as shown below



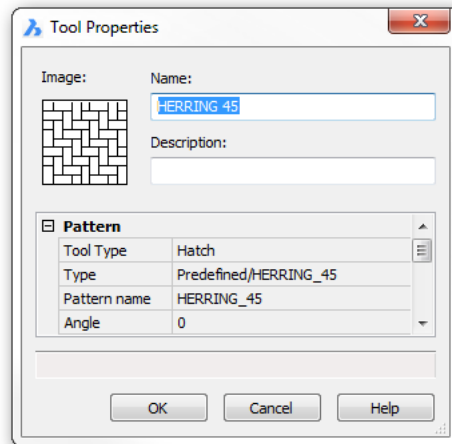
Renaming a tool

Update tool image — refreshes the icon (applies to blocks and hatch patterns only)

Specify Image — opens the Select Image File dialog box, from which you select an image in BMP bitmap, GIF, JPEG, PNG, or TIF format; large images are resized automatically to fit the icon area

Remove Image — returns the icon to its default, removing the image you applied with the Specify Image option; I suppose a better name for this option could be “Reset Image”

Properties — displays a dialog box for changing the item’s properties (see figure below); the content of the dialog box varies according to the type of tool selected, and is discussed later in this chapter

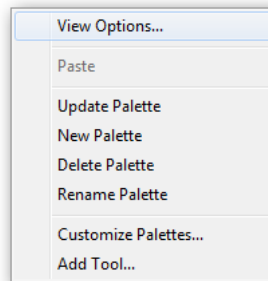


Changing the properties (and functions) of a tool

Shown above are the properties of hatch patterns. To edit a property, click on it in the dialog box, and then make the change.

PALETTE CUSTOMIZATION

The first step in customizing the Tool Palettes panel is to right-click on an unused area of the panel — not on an icon or a tab — to get the following shortcut menu:



Right-click menu for controlling palettes

View Options — displays the same dialog box as does the tab’s shortcut menu

Paste — pastes a tool, if one has been copied or cut to the Clipboard

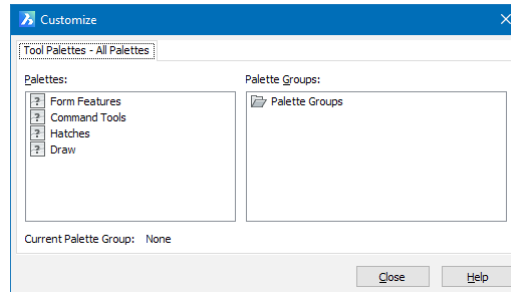
Update tool image — refreshes the icon (applies to blocks and hatch patterns only)

New Palette — adds a new blank palette

Delete Palette — removes a palette

Rename Palette — changes the name on the tab

Customize Palettes — displays the Customize dialog box for creating palette groups (see figure below); the dialog box is described later



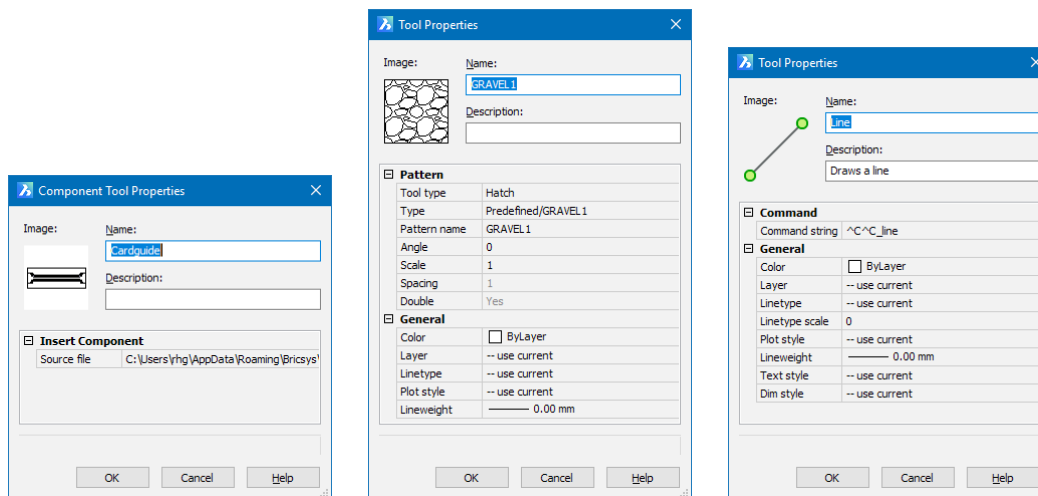
Customize dialog box for creating palette groups

Add Tool — displays the Customize dialog box, as described in earlier in this book

Customizing Tools

The Tool Properties dialog box lets you customize the actions of commands and hatch patterns. For instance, you can specify that clicking an icon (that looks like a cloud) draws revisions clouds that are red in color, and placed on a specific layer or with a certain linetype.

To access this very important dialog box, right-click the icon you want to customize, and then choose **Properties**. There are three versions of the dialog box — one each for form features (components), hatches, and commands, as illustrated below:



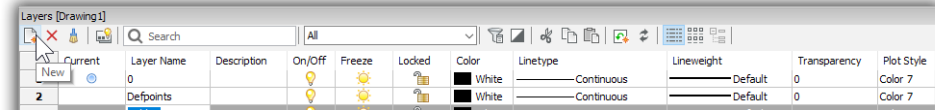
Left to right: Tool properties for form features, patterns, and all other commands

Customizing Tools Properties

In this tutorial, you make a copy of an existing tool, and then customize it by changing its properties. You modify the Line tool to draw lines with the “Hidden” linetype on layer “Hidden.”

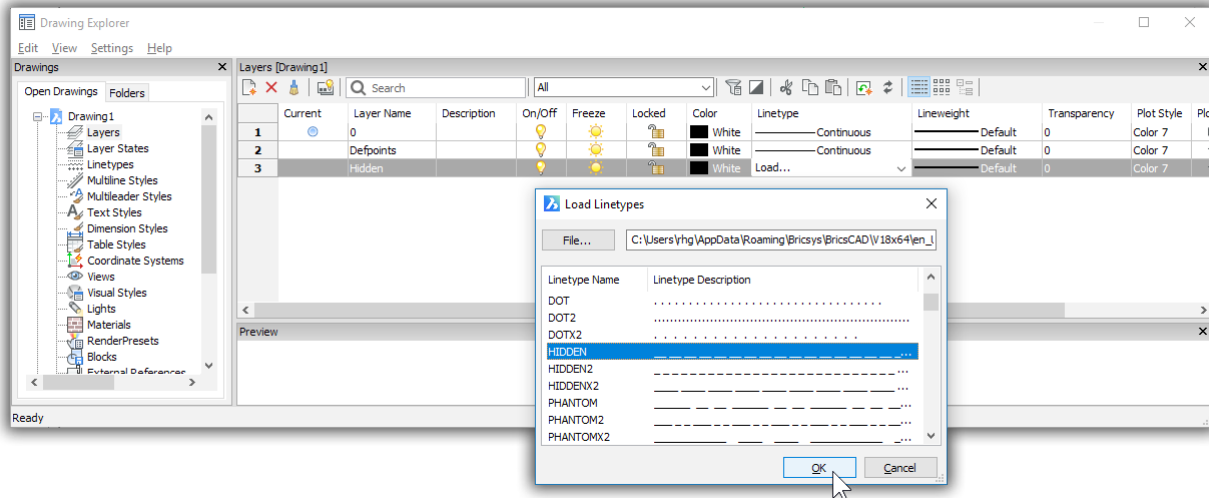
Follow these steps to customize it:

1. Create a new layer named “Hidden.” Use the **Layer** command to do this in the Drawing Explorer dialog box.



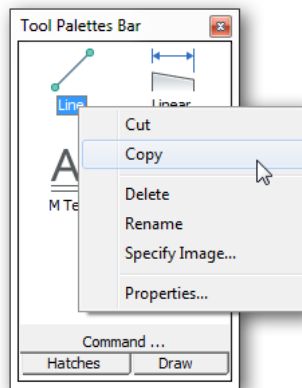
Creating a new layer named “Hidden”

2. Load the “Hidden” linetype into the drawing. Use the **Linetype > Load** option for this.



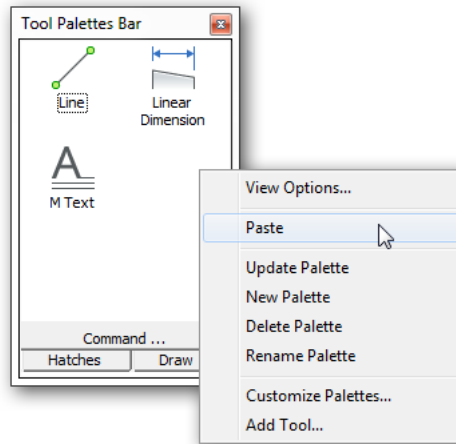
Drawing Explorer listing available linetypes

3. Close the Drawing Explorer.
4. Open the Tool Palettes panel with the **ToolPalettes** command, and then choose the **Command Tools** tab.
5. Make a copy of the Line tool by copying it and then pasting it, like this:
 - a. Right-click the **Line** tool, and then choose **Copy** from the shortcut menu.



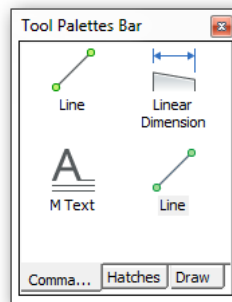
Copying the Line tool

- b. Right-click a blank area of the palette, and then choose **Paste**.



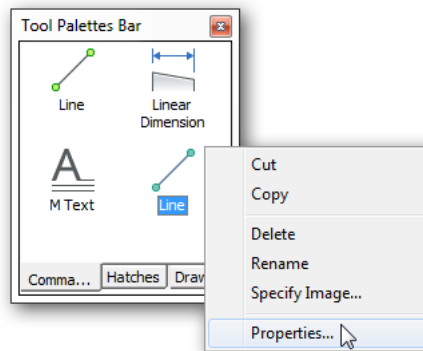
Pasting the tool as a new one

Notice that BricsCAD creates a second Line tool.



The second Line tool in place

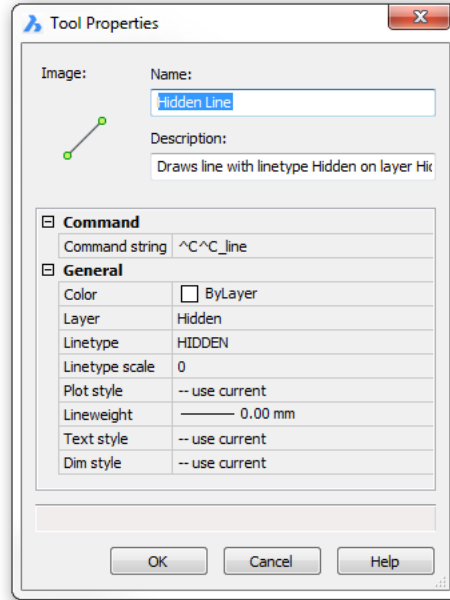
6. With the copy made, you can now edit its properties. Right-click the new item, and then choose **Properties**.



Opening the properties of the new Line tool

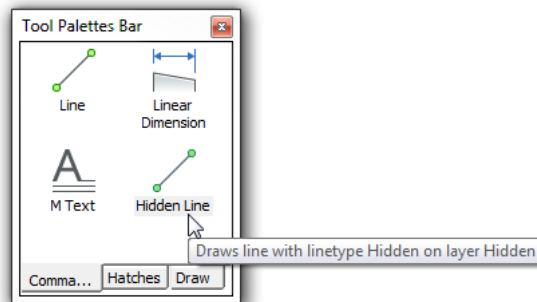
7. In the Properties dialog box, change the following properties for this tutorial:

Properties	Value
Name	Hidden Line
Description	Draws line with linetype Hidden on layer Hidden
Linetype	Hidden
Layer	Hidden



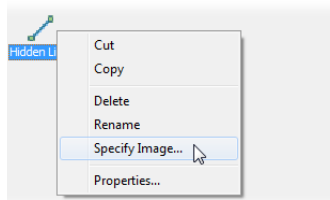
Changing the properties

8. Click **OK**. Notice that BricsCAD changes the label of the icon.



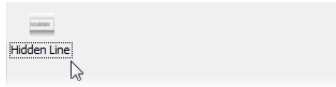
Renamed icon

9. If you want to change the icon associated with the button, follow these steps:
- Right-click the icon, and then from the shortcut menu, choose **Specify Image**.



Starting to assign an image to the button

- b. In the Select Image Position File dialog box, select an image file (in BMP, JPEG, PNG, GIF, or TIFF format), and then click **Open**. Notice that the icon changes. BricsCAD automatically resizes the image to fit the area of the icon.



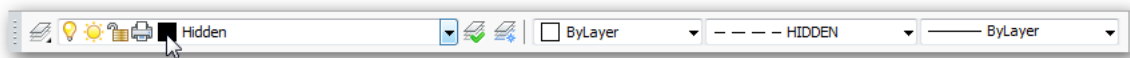
Icon changes to the newly selected image

10. To test the tool that it actually works, click it and draw some line segments. If you do not see the hidden linetype, change the value of the linetype scale with the **LtScale** command.



Drawing line segments with hidden linetype automatically applied

After drawing the lines, click them to see that they are being drawn on the Hidden layer.

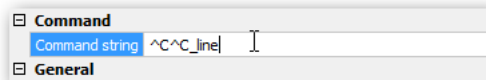


"Hidden" layer automatically set

Adding Programs and Macros to Tools

You can make tools carry out simple programs (known as "macros"), but the process is less direct than with just commands. It takes a workaround shown by these steps:

1. Place any geometric entity (tool) on a palette. It doesn't matter what the entity is, because it is used only as a placeholder.
2. Right-click the newly-added tool, and then select **Properties**.
3. In the **Command String** area, replace the command with a piece of programming code or a macro.



Editing the macro assigned to a button

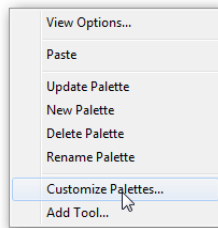
TIP You can either type the new code, or copy'n paste it from another source, such as from the earlier chapters of this book that deal with writing macros.

4. Click **OK** and then test to macro to make sure it works.

Organizing Tools with Groups

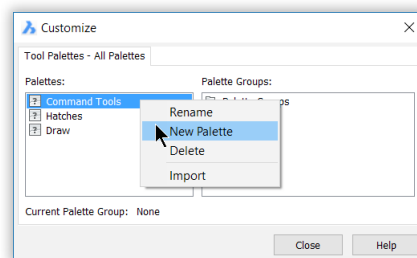
BricsCAD allows you to create many palettes, but too many palettes can become unwieldy, and so BricsCAD allows you to create groups of palettes. Groups let you show only those palettes you need currently.

To create groups, right-click a blank area of the Tool Palettes panel, and then choose **Customize Palettes** from the shortcut menu.



Accessing that other Customize dialog box

Notice that the Customize dialog box appears, and that it is different from the Customize dialog box displayed by the Customize command! (There is no independent command to display this dialog box.)



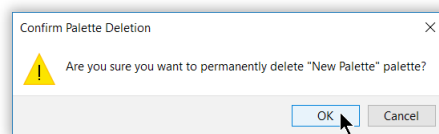
Dialog box for creating palette groups

The commands and options for this dialog box are accessed solely by right-click menus. For instance, to create a new palette, right-click in the Palettes area and then select **New Palette**, as shown above.

Rename — renames the palette

New Palette — creates a new palette named “New Palette,” which you then give a new name

Delete — erases the palette after you click OK to this dialog box:



Import — imports palette files in XTP and BTC formats; opens the Import Palette dialog box.

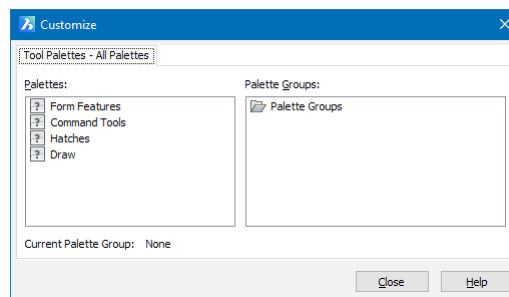
XTP is short for “XML Tool Palette,” the file format used by AutoCAD. BTC is short for “BricsCAD Tool Collection.”

CREATING PALETTE GROUPS

You can create as many palettes as you want; I am unsure if there is an upper theoretical limit. I can see a design firm creating dozens of palettes, some for electrical engineers, some for landscape designers, and so on. You can create grouping of palettes so that the electrical engineer doesn't have to see the palettes for landscaping. A *group* is a smaller set of palettes.

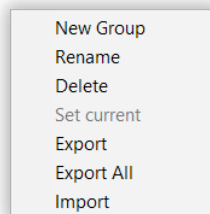
BricsCAD does not come with any palette groups, so you get to create your own through the oddly-named Customize dialog box — oddly named, because this is *not* the dialog box of the same name that is opened with the Customize command.

There is no command to access this dialog box; instead, you (as always) right-click a blank area of a palette, and then choose **Customize Palettes** from the shortcut menu. Notice the other Customize dialog box:



Accessing that other Customize dialog box

(While this dialog box existed prior to V17, the groups function did not work.) As elsewhere with Tool Palettes, all commands in this dialog box are executed through shortcut menu. When you right-click the name of a group, the following menu appears:



Shortcut menu for palette groups

New Group — creates a new but empty palette group

Rename — renames the palette group

Delete — erases the group with no warning

Set Current — sets the selected group as the current one, meaning it will be displayed by the Tool Palette panel; when the group is empty, then this command is grayed out

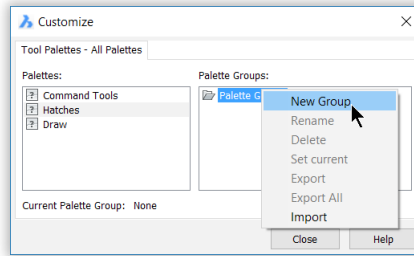
Export — exports the current group to a XPG file, as described later

Export All — exports all groups in a single XPG file

Import — imports XTP and BTC files, as described later

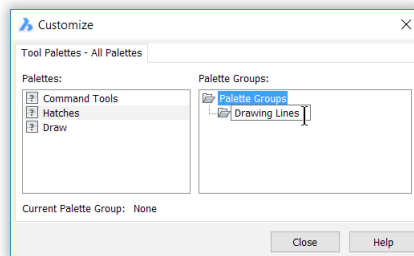
To create a group of palettes, follow these steps:

1. In the Customize dialog box, right-click **Palette Groups**, and then from the shortcut menu choose **New Group**.



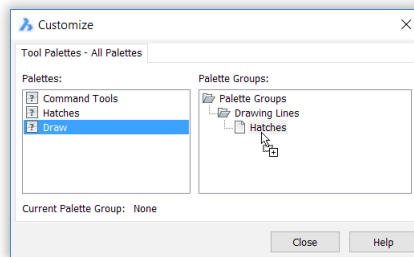
Creating a new group of palettes

2. Notice that BricsCAD creates a new group name named “New Group.” For this tutorial, change then name to **Drawing Lines**.



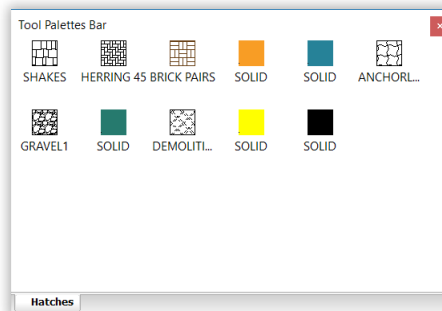
Naming the new group

3. Now drag palette names from the Palettes list over into the newly formed group. For this tutorial, drag over “Hatches.”



Add palettes to a group

4. Right-click the group “Drawing Lines” and then from the dialog box choose **Set Current**.
5. When done, click **Close**. Notice that the Tool Palette panel now shows just one palette, Hatches.

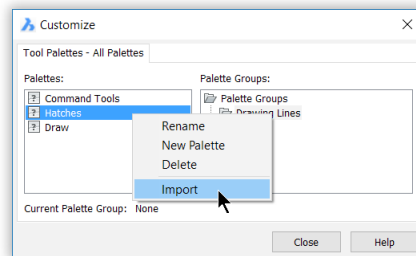


Tool palettes with a custom group that shows just one palette

IMPORTING TOOL PALETTES FROM AUTOCAD

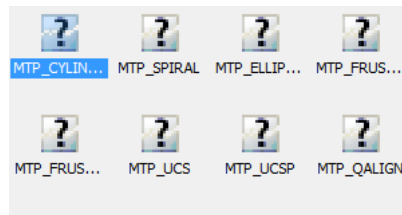
BricsCAD reads files saved the XTP format; this is the format in which AutoCAD saves tool palettes. “XTP” is an XML-based file format; the name is short for “xml tool palette.” To import them into BricsCAD, follow these steps:

1. In this Customize dialog box, right click and then choose **Import** from the shortcut menu.



Choosing the Import option

2. In the Import Palette dialog box, navigate to where you have XTP files.
3. Choose the file, and then click **Open**. Notice that BricsCAD adds the palette to its collection.
4. Click **Close** to close the dialog box. Notice the new tab with its icons.



Tools lacking icons

BricsCAD does not have access to the icons used for tools, because AutoCAD stores them internally. This is why question-mark icons are displayed.

Sharing Tool Palette Groups by Exporting Them

Once you customize a tool palette, you might want to share it with others. To do so, you export groups. Groups are exported in XTG files (short for “XML tool group”).

Now, individual palettes cannot be exported (I am not sure why!), but you can do the same thing by putting a single palette into a group, and then exporting it.

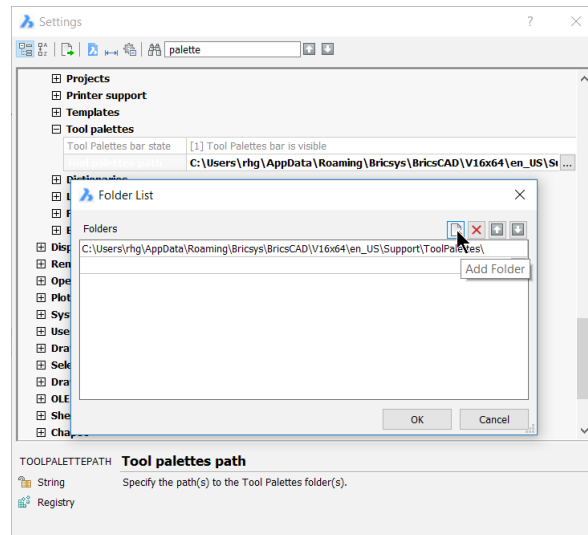
Here is how to do this:

1. In this Customize dialog box, right click a group, and then choose **Export** from the shortcut menu.
2. In the Export Palette dialog box, navigate to where you store XTG files.
3. Name the file, and then click **Save**.

Alternative Sharing Method

Another way to share palettes is to edit the Settings dialog box in BricsCAD to point to where BTC and XTP files are stored, such as for AutoCAD.

1. Enter the **Settings** command.
2. In the search field, enter “palette.”
3. Open the Tool Palettes section, and then go to the Tool Palettes Path setting.
4. Click the **...** **Browse** button.



Adding folders to BricsCAD's folder searches

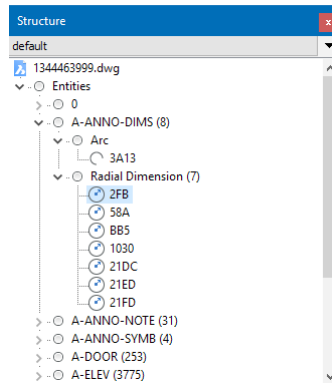
5. In the Folder List dialog box, click the **Add Folder** button.
6. Enter the path to the folder holding the XTP or BTC files. If necessary, click the **Browse** button and then use the Choose a Folder dialog box to locate the folder.
7. Click **OK** sufficient times to back out of all the folders!

This dialog box can be used to point to other AutoCAD support folders, such as for hatch pattern collections, linetype files, printer setups, and so on.

Customizing the Structure Panel

The Structure panel displays a structured tree view of the drawing's content. This includes the names of entities, blocks, and nearly any other entity. You can customize the elements that are listed and in which order.

When you select the name of an entity in the structure tree, the entity is highlighted in the drawing — and vice versa. Notice that entities are identified by hexadecimal (base 16) numbers.



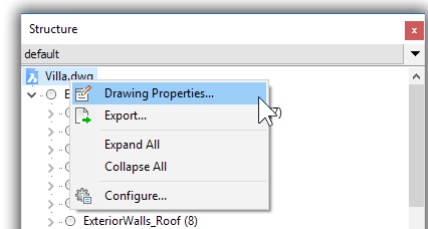
Structure panel showing the structure of a floor plan by entities

There are two commands to open the Structure panel:

- ▶ **StructurePanel** opens the Structure panel; **StructurePanelClose** closes it. Alternatively, right-click a toolbar or other UI element in BricsCAD, and then choose **Structure** from the shortcut menu.
- ▶ **+StructurePanel** opens the Structure Tree Configuration File dialog box, prompting you to select a **.cst** (Configure Structure Tree) file; when you click **Open**, the Structure panel is opened and displays the configuration defined by the **.cst** file.

A limitation: The panel operates in model space only.

When the panel opens, right-click the name of the drawing to see the following shortcut menu:



Accessing options

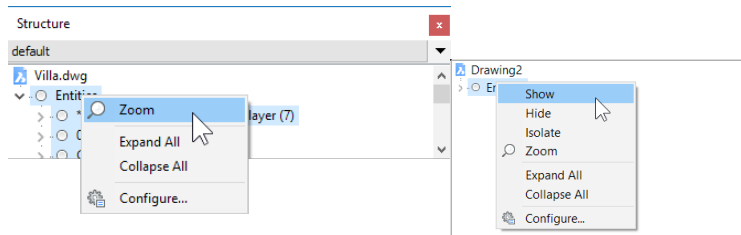
Drawing Properties — displays the Drawing properties dialog box, the same as entering the **DwgProps** command

Export — exports the drawing as an XML file

Expand / Collapse All — expands and collapses all nodes

Configure — displays the Configure Structure Tree (more later)

Right-clicking any other item in the panel displays the following shortcut menu, with a few different options:



Accessing another shortcut menu

Show — Show the entity in the drawing

Hide — Hides the entity from the drawing

Isolate — Hides all other entities in the drawing

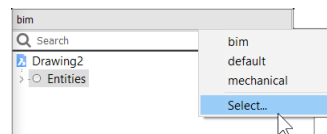
Zoom — zooms into the selected entity(ies)

Structure Configurations

The data displayed by the panel can be customized. BricsCAD provides several pre-made configurations suitable for various kinds of drawings. The `C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support` folder holds these `.cst` files:

```
default.cst
bim.cst
mechanical.cst
```

To load another configuration, click the down arrow (upper-right corner of the panel), and then choose one from the list or else navigate to another folder with the **Select** option.



Selecting a Structure panel configuration

Alternatively, enter the **StructureTreeConfig** variable, which prompts you at the command line to specify the name of a `.cst` file:

```
: structuretreeconfig
New value for StructureTreeConfig, or . for none/< C:\Users\<Login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\default.cst>:
```

CUSTOMIZING THE STRUCTURE PANEL

The format of the panel is customized through the Configure Structure Tree dialog box. You can create many configurations, depending on your needs. Customization of a configuration takes two steps:

Step 1 — Create a rule

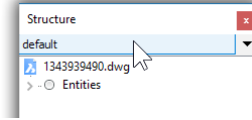
Step 2 — Specify the properties of the rule

STRUCTURE OF .cst FILES

The C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support folder holds several .cst files. The *bim.cst* file is shown below, organizing the building spatially, first by Building, then by Story, BIM type, and then composition. Sections are grouped by type: Section, Plan, Elevation, and Detail.

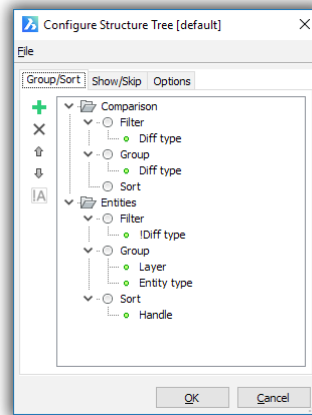
```
{
  "created": {
    "by": "BricsCAD",
    "on": "2016-10-28 10:50:09"
  },
  "rules": [
    {
      "name": "Building Elements",
      "group": [
        "BIM/Building",
        "BIM/Story",
        "BIM/Type",
        "BIM/Composition"
      ],
      "sort": [
        "BIM/Name"
      ]
    },
    {
      "name": "Sections",
      "group": [
        "BIM/Section Type"
      ],
      "sort": [
        "Name"
      ]
    },
    {
      "name": "Entities",
      "group": [
        "EntityType"
      ],
      "sort": [
        "Name",
        "EffectiveName",
        "Handle"
      ]
    }
  ],
  "mode": "showAll",
  "some": [],
  "options": {
    "treeSelect": "select",
    "entitySelect": true,
    "autoCollapse": false,
    "displayMode": "byType"
  }
}
```

To customize the data displayed by the panel, click the bar (the one with the word “default” in the figure below).



Opening the Configure Structure Tree dialog box

This handy shortcut opens the Configure Structure Tree dialog box.



Dialog box for customizing the display of the Structure panel

The dialog box sports three tabs — Group/Sort, Show/Skip, and Options. Let’s take a look at what they offer.

Group/Sort Tab

The Group/Sort tab determines how drawing data is displayed. Here you can add rules and property filters, move items around, and remove them.

To add a rule:

1. Select an existing rule name.
2. Click the green **+** (Add) button. Notice that the rule is duplicated.
3. Use the arrows to move the selected rule up and down the list.
4. Edit the content of the rule, as described later.

To rename a rule:

1. Select a rule name.
2. Click on the name a second time.
3. Give the rule a different name, and then press **Enter**.

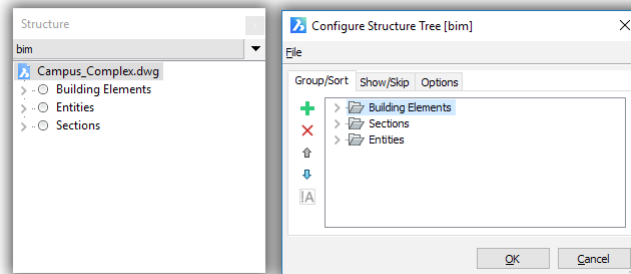
To remove a rule:

1. Select a rule name.
2. Click the red **X** button; there is no warning.

Examining Rules

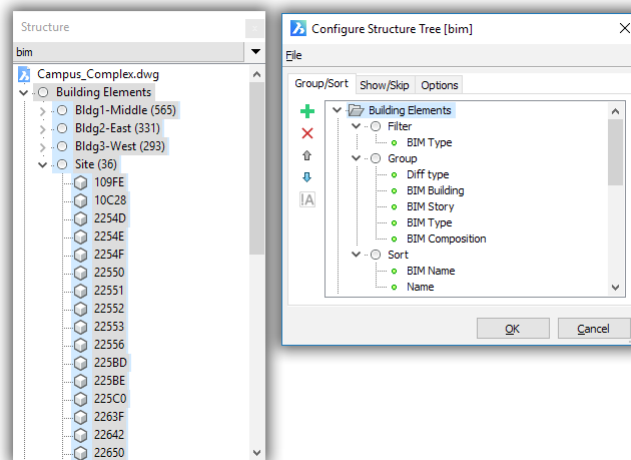
The way that element information is displayed in the Structure panel is determined by *rules*. The rules specify which entities are displayed, and the order in which their properties are listed. By listing one property ahead of another one, the second property becomes a subset of the first. More on this later.

Let's take the example of the *bim.cst* configuration.



Left: Structure panel's display controlled by; right: ...the bim.cst file

The names of the rules — such as **Building Elements** and **Sections** — are for use by humans only, and describe the content of the rules. Think of them as sections. Click a gray > (angle bracket) to open the section, displaying the elements of the rule.



Expanded rules

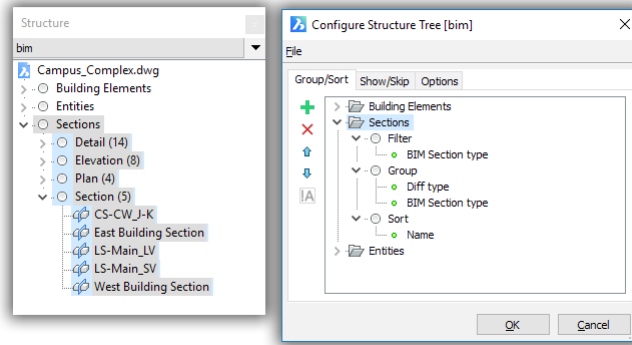
There are three elements in each rule:

- Filter** — determines which elements are displayed by the rule; in this rule, it's the names of BIM Types.
- Group** — determines how the BIM Types are grouped together; in this case, BIM Buildings, Stories, etc.
- Sort** — specifies how the elements are sorted within each group; here, it's by BIM Name.

Let's look at another rule to assist with understanding how configurations work: information about sections. The Sections rule arrives at this by specifying the following:

Rule	Property	Meaning
Filter	BIM Section type	List details, elevation, plan, and section names
Group	BIM Section type	Group together BIM section types
Sort	Name	Sort section types by name alphabetically

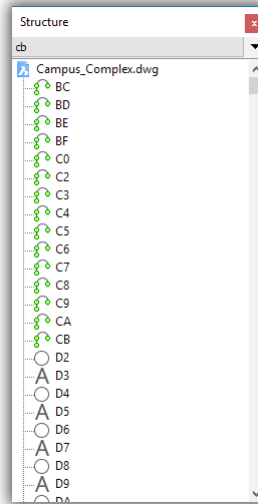
You can see why I used the word “elements” instead of “entities.”



BIM rules for displaying sections

TIPS The **Diff Type** property is used during the **3dCompare** command; it reports differences, if any, between the two drawings being compared.

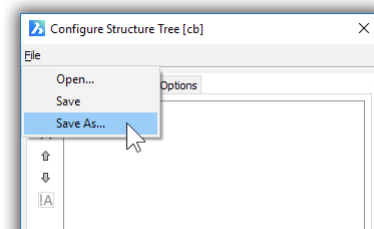
When there is no configuration file loaded, the Structure panel displays drawing entities in alphabetical order. The icon in front of each name identifies the entity type, such as polyline, circle, and text:



Constructing Rules

To construct a rule, you select properties from a long list that I will show you shortly.

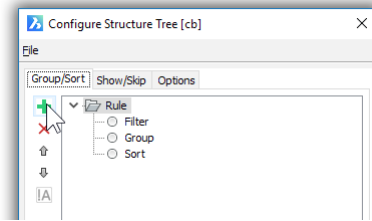
For this tutorial, I created a blank configuration file. I did this by deleting all rules from an existing .cst file, and then used the **File | Save As** command to save it (with the name *cb.cst*).



Saving a blank .cst file

In this tutorial, we create a rule that causes the Structure panel to list only lines in the current drawing, sorted by layer name, and then by length.

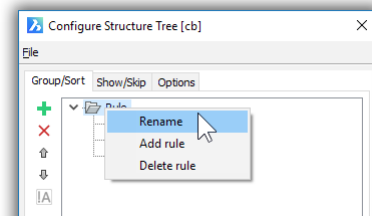
1. To start constructing a new set of configuration rules, click the green **+** icon. Notice that a generic rule is added, named “Rule.”



A generic rule added

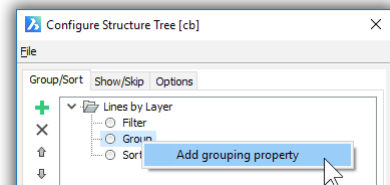
The generic rule has three (fixed) sections that define the rule: Filter, Group, and Sort. You cannot add or delete these three sections. If the section is empty, then it does nothing.

2. Rename the rule as “Lines by Layer,” as follows:
 - a. Right-click “Rule,” and then choose **Rename**.



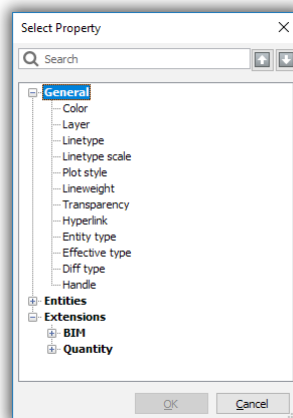
Renaming a rule

- b. Change “Rule” to **Lines by Layer**, and then press **Enter**.
3. We want the Structure tree to display only lines, so right-click **Group**, and then choose **Add Filter Property**.



Adding properties to a rule

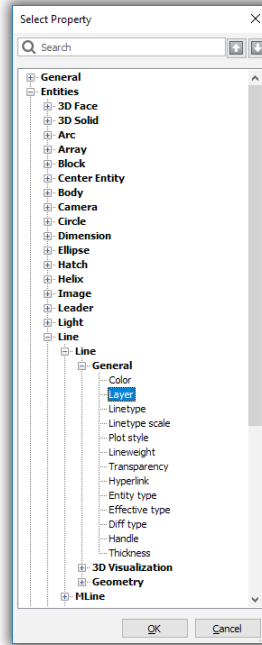
Notice the Select Property dialog box. It lists every property available in BricsCAD — a couple hundred of them.



Properties dialog box

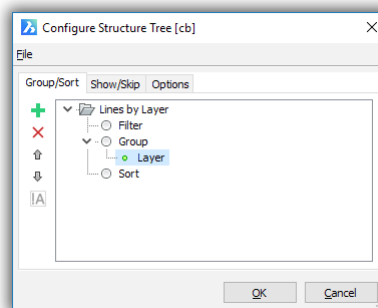
The properties are grouped into these sections:

- ▶ **General** — lists properties common to all entities; these should be known to you from the Properties panel
 - ▶ **Entities** — lists the names of all entities and their specific properties
 - ▶ **Extensions** — lists properties that are specific to add-ons to BricsCAD, such as the BIM module
4. For this tutorial, we want lines listed by layer name. Click the + next to **Entities**, and then work your way down: **Entities > Line > Line > General > Layer**:



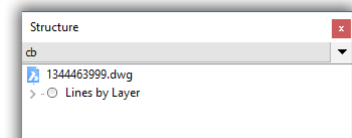
Properties

5. Close the Properties dialog box by clicking **OK**. Notice that “Layer” is added under **Group**.



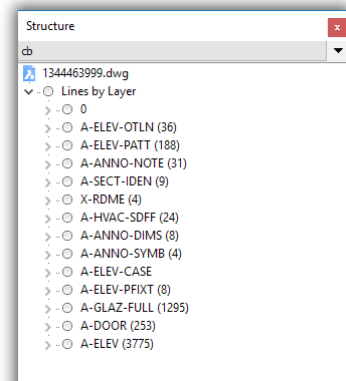
Entities grouped by layer

6. Let's see what this has done to the Structure panel. Click **OK** to close the dialog box. Notice that the panel lists the “Lines by Layer” rule.



New rule displayed in Structure panel

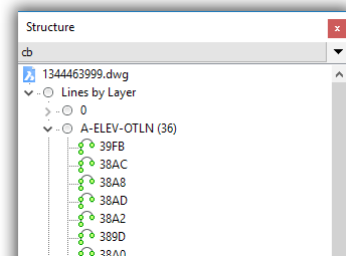
7. Now let's see what is listed under that rule. Click the > next to "Lines by Layer."



Layer names in the drawing

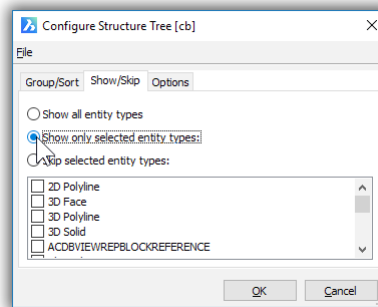
So far it looks good. You see the list of all layer names in the drawing. (I don't know the order in which they are listed — perhaps in order they were created?) The number in parentheses next to each layer name is the number of entities on that layer, such as (36) for A-ELEV-OTLN.

8. Click the > next to layer "A-ELEV-OTLN." Oops! What's this? The icon shows that those are polylines listed.



Polylines listed by layer

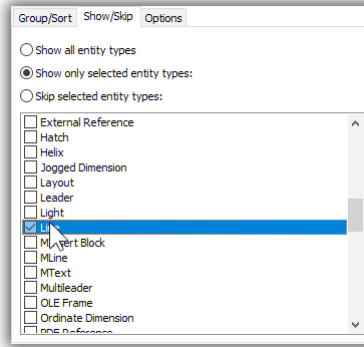
9. To restrict the list of entities to lines only, follow these steps:
- Return to the Configure Structure Tree dialog box by clicking the "cb" bar.
 - Click the **Show/Skip** tab.



Restricting the entities listed by Structure

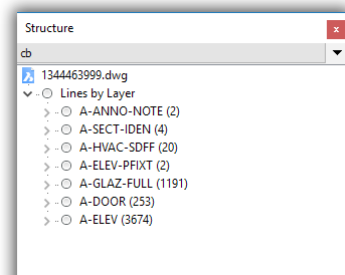
- c. Choose the radio button next to **Show Only Selected Entity Types**.

- d. Scroll down until you find **Line**, and then click the box next to it.



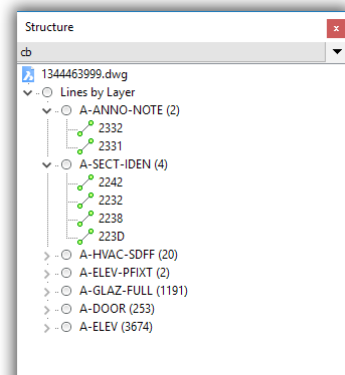
Choosing the Line entity

- e. Click **OK** to dismiss the dialog box.
10. Now open the “Line by Layer” rule in the Structure panel. Notice that fewer layers are displayed, because now only layers with lines are listed.



Only layers with lines listed

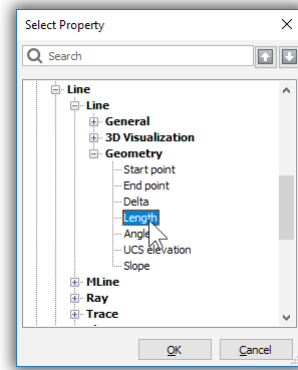
11. Open a layer name. Notice that just line entities are listed.



Lines on each layer

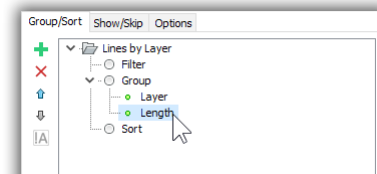
12. Add a subcategory, such as the length of each line. Follow these steps:
- Return to the Configure Structure Tree dialog box by clicking the “cb” bar.
 - Right-click **Group**, and then choose **Add Grouping Property** from the shortcut menu.

- c. In the Select Properties dialog box, navigate to **Entities > Line > Line > Geometry > Length**.



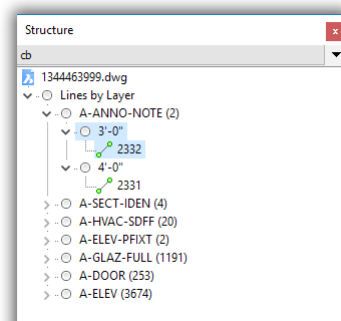
Adding Length to the group

- d. Click **OK**. Notice that Length is added under Group.



Length added to the Group section

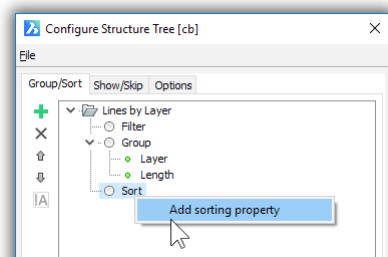
- e. Click **OK** to close the dialog box. Open up the nodes to find that the layers and lengths of lines are listed, such as layer “A-ANNO-NOTE” and then 3'0”.



Lines listed by layer name, and then by length

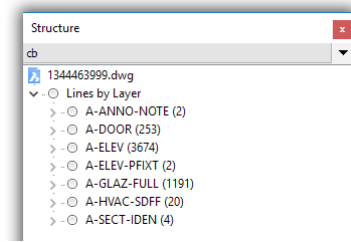
13. As I mentioned earlier, the elements seem to be sorted in a random order. Let's now sort them alphabetically. Follow these steps:

- Return to the Configure Structure Tree dialog box by clicking the “cb” bar.
- Right-click **Sort**, and then choose **Add Sorting Property** from the shortcut menu.



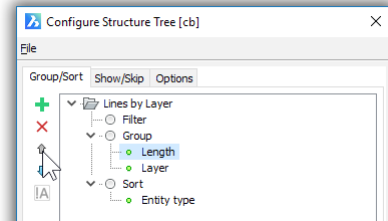
Adding a sorting property

- c. Choose **Entity Type**, and then click **OK** twice to dismiss the dialog boxes.
- d. Take a look at how the added rule affected the listing: notice that the layer names are now alphabetical.



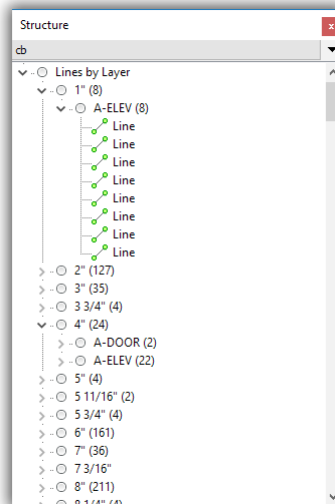
Layer names listed alphabetically

14. As a final step in this tutorial, let's switch the order of the Layer and Length rules, like this.
 - a. Return to the Configure Structure Tree dialog box by clicking the "cb" bar.
 - b. Select "Length," and then click the up arrow button to move it above the "Layer" rule.



Changing the order of rules

- c. Click **OK** to dismiss the dialog box.
- d. Notice that lines are now listed by length, and then by layer name.



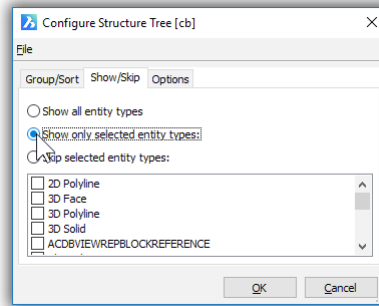
Lines listed by length and layer

Show/Skip Tab

The Show/Skip tab gives you options in how to display items. This lets you filter out entities you are not interested seeing in the Structure panel, such as ordinary lines.

- ▶ **Show all entity types** in the drawing (default)
- ▶ **Show only selected entities types** chosen in the list below (those with the check mark)

- ▶ **Skip selected entity types** as chosen in the list below



Options found in the Show/Skip tab

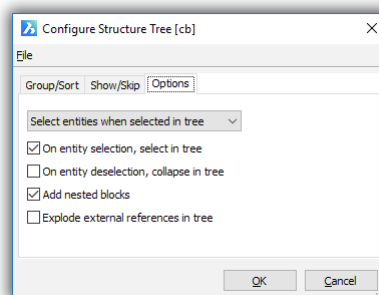
Options Tab

The Options tab provides options that control what happens with the structure tree:

- ▶ **Ignore tree selection** — nothing happens when you select an item in the Structure panel
- ▶ **Highlight entities when selected in tree** — (default) when you select an item in the Structure panel, it gets highlighted in the drawing
- ▶ **Select entities when selected in tree** — when you select an item in the panel, it is also selected in the drawing, following which you can immediately edit it

On entity selection, select in tree — when on, the entity you select in the drawing is highlighted in the Structure panel

On entity deselection, collapse in tree — when on, the tree in the Structure panel collapses when the entity is no longer selected in the drawing. This is useful, because the content in the panel can get very long.



Specifying options

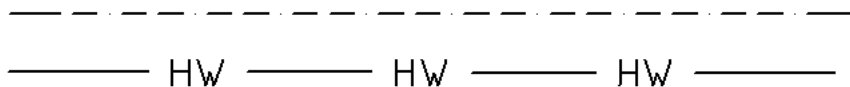
Add nested blocks — when on, includes blocks that are nested inside other blocks.

Explode external references in tree — when on, lists all elements in an xref separately; when off, lists the xref as a single element.

Creating Simple & Complex Linetypes

BricsCAD supports two styles of linetypes, simple and complex:

- **Simple linetypes** — consist of lines, gaps, and dots strung together in a variety of patterns.
- **Complex linetypes** — add text and shapes to simple linetypes.



Top: Simple linetype consisting of dashes, dots, and gaps.

Above: Complex linetype for hot water pipes.

The BricsCAD package includes many simple and complex linetypes, and you can create your own, as described in this chapter.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Discovering commands and system variables that affect linetypes
- Understanding the special case of polylines
- Checking compatibility with AutoCAD
- Customizing linetypes
- Editing linetype definitions
- Testing new linetypes
- Creating linetypes with text editors
- Understanding the linetype format
- Creating complex linetypes

QUICK SUMMARY OF LINETYPE DEFINITIONS

Linetypes are stored in .lin files and loaded with the Linetype command. Each linetype definition consists of two lines of text, a header that labels the linetype, followed by a line of data that describes the linetype format.

LINE 1: HEADER

Example: *Name, . _ . _ . _

* (asterisk) — indicates the start of the linetype definition.

Name — names the linetype.

, (comma) — separates the name from the description.

. _ . _ — illustrates the linetype pattern, to a maximum of 47 characters.

LINE 2: SIMPLE LINETYPE DATA

Example: A, .25, -.1, 0, -.1

A — specifies the alignment flag to force the linetype to begin and end a line segment adjusted to the overall length of the object.

.25 — specifies the length of the dash, when LtScale = 1.0.

-.1 — specifies the length of the space, using a negative value.

0 (zero) — specifies a dot.

LINE 2: COMPLEX LINETYPE DATA

Complex linetypes provide additional parameters within square brackets, as shown in boldface below.

Example: A, 1.0, -.25, [**“HW”**, **STANDARD**, **S=.2**, **R=0.0**, **X=-0.1**, **Y=-0.1**], -.40

“HW” — specifies the letters to be displayed by the linetype.

STANDARD — specifies the text style. Optional; when missing, current style is used.

S=.2 — specifies the height of the text, or its scale factor, depending on the following:

- When style's height = 0, then S specifies the height (0.2 in this case).
- When style's height is not 0, then S multiplies the style's height (0.2x).

R=0.0 — rotates the text relative to the direction of the linetype. Optional; when missing, angle = 0. Default is in degrees; can use r and g to specify radians or grads.

A=0.0 — rotates text relative to the x-axis to ensure that the text is always oriented in the same direction. Optional.

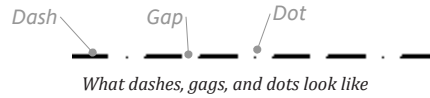
X=-0.1, Y=-0.1 — offsets the text in the x and y directions.

NOTES

Every data line must begin with a dash; every dash and dot must be separated with a space. To include comments in the .lin file, prefix lines with a semi-colon (;).

About Simple and Complex Linetypes

Simple linetypes consist of lines, gaps, and dots ordered in a variety of patterns. This is the most common type of linetype, and its components are shown by the figure below.



And here are some of the simple linetypes included with BricsCAD. Notice that they all consist of the gaps, dashes, and/or dots in a variety of patterns:

Linetype Name	Linetype Description
BORDER	---.---.---.---.---.---.---.---.---.---
BORDER.2	---.---.---.---.---.---.---.---.---.---
BORDERX2	---.---.---.---.---.---.---.---.---.---
CENTER	---.---.---.---.---.---.---.---.---.---
CENTER.2	---.---.---.---.---.---.---.---.---.---
CENTERX2	---.---.---.---.---.---.---.---.---.---
DASHDOT	---.---.---.---.---.---.---.---.---.---
DASHDOT.2	---.---.---.---.---.---.---.---.---.---
DASHDOTX2	---.---.---.---.---.---.---.---.---.---

Simple linetypes consisting of lines, gaps, and dots in a variety of patterns

Also included with BricsCAD are standardized linetypes defined by ISO, the International Organization of Standards. Complex linetypes are like simple linetypes, but include text, such as ones that indicate fence and gas lines, as illustrated below:

Linetype Name	Linetype Description
FENCELINE1	Fenceline_circle ---0---0---0---0---0---0---
HOT_WATER_SU...	Hot water supply --- HW --- HW --- HW ---
GAS_LINE	Gas line ---GAS---GAS---GAS---GAS---GAS---G...
SMILEYS	Smiley line --- :^) --- 8) --- :^) ---

Complex linetypes adds characters to simple linetypes

As with colors, the convention is to assign linetypes to objects in drawings through layers — not with the Linetype command! Using the Layer command, you assign different linetypes to various layers.

You can, however, apply linetypes to objects directly, like colors, through the Entity Properties toolbar or Properties pane.

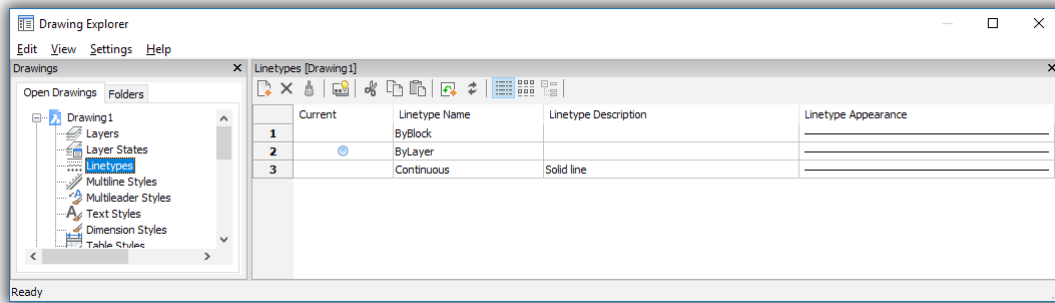
COMMANDS AFFECTING LINETYPES

Linetypes are not stored in drawings; instead, they have to be loaded from .lin files. It's always been a source of irritation to me that I gotta load the file into drawings before I can use any linetype. There is one workaround, and just one: add all linetypes to all template drawings.

BricsCAD provides two commands loading linetypes into drawings, Linetype and -Linetype.

Loading Linetypes

The **Linetype** command opens Drawing Explorer for loading, listing, renaming, and deleting linetypes. (In older days, this command was known as ExpLTypes, short for “explore linetypes.”) As an alternative, you can access Drawing Explorer from the **Tools** menu: choose **Drawing Explorer | Linetypes**.



Drawing Explorer handles all aspects of loading and assigning linetypes

The other command is **-Linetype**, and it operates at the command prompt. It loads linetypes, lists the names of those already loaded, and can define new ones. It is meant mainly for use with scripts and LISP programs.

The two commands load linetypes from these *.lin* files:

- ▶ *default.lin* — definitions for imperial linetypes
- ▶ *iso.lin* — definitions for metric (ISO) linetypes
- ▶ *standard.shx* — source for characters used by complex linetypes

BricsCAD stores linetypes in the following folders:

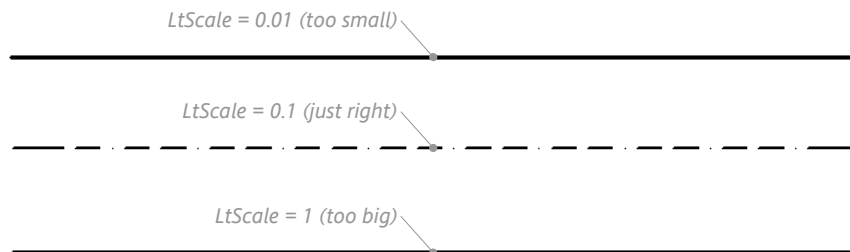
Window — C:\Users\

Linux — /home/<login>/Bricsys/BricsCAD/V20x64/en_US/Support

Mac — /Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/Support

Scaling Linetypes

Like text, linetypes can be tricky to size. You have to scale the gaps and dashes in just the right way. Too small a scale, and linetypes look solid — but takes a suspiciously long time to redraw. Too large, and the linetype also looks solid. Here’s what the problem looks like:



The effects of scale on linetypes

And to solve the problem, BricsCAD has the **LtScale** system variable, short for “linetype scale.” It sets the scale of linetypes. Typically, the scale factor you use for text, dimensions, and hatch patterns also applies to linetypes. Nice, eh?

SYSTEM VARIABLES AFFECTING LINETYPES

There are many system variables that control the look and size of linetypes. They happen to be scattered all around the Settings dialog box, and so I provide this complete list of them. This first set of variables determine the linetypes used for drawings:

MeasureInit — sets the initial unit of measurement for new drawings (metric or Imperial), and so determines which .LIN files are used (ANSI or ISO)

Measurement — changes the units for the current drawing between metric and Imperial, and so determines which ANSI or ISO.LIN files are used

SrchPath — specifies the path to LIN definition files

These are the system variables that relate to linetypes applied to entities in drawings:

CeLType — holds the name of the linetype currently in effect; short for “current entity linetype”

CeLtScale — specifies the current linetype scale

LtScale — stores the current linetype scale factor; short for “linetype scale” (default = 1.0)

PLineGen — determines how linetype cross polyline vertices; short for “polyline generation”

VisRetain — determines whether changes made to xref layers, such as linetypes, are saved with the drawing

In addition to regular entities, linetypes can also be specified for parts of regular and dynamic dimensions, and for visual styles:

DimLType — specifies the linetype for dimension lines

DimLtEx1 and **DimLtEx2** — specifies the linetype for the first and second extension lines

DynDimLineType — specifies the linetype displayed by dynamic dimensions as they are being moved

ObscuredLType — specifies the linetype of obscured line; independent of zoom scale

A final set of variables specifies the linetype scale factor in outside of traditional model space:

MsLtScale — annotatively scales linetypes in model space

PsLtScale — scales linetypes in paper space; short for “paper space linetype scale”

The **CeLType** system variable reports the name of the current linetype. You can use it as a keyboard shortcut to change the name of the current linetype, like this:

```
: ceLtype  
New value for CELTYPE, or . for none/<"ByLayer">: continuous
```

The Special Case of Paper Space

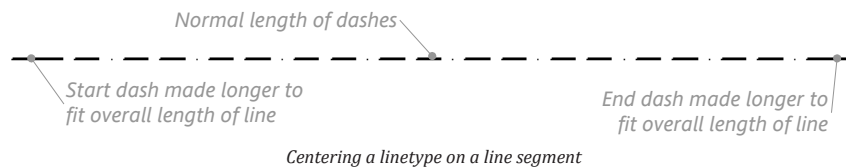
Because linetypes are affected by scale, their scale becomes a problem in paper space. A linetype scale that looks fine in model space will look wrong in paper space, because paper space almost always has its own scale factor. By default, the scale of linetypes in paper space is 1.0 — no matter what it may be in model space.

The solution comes with the **PsLtScale** system variable. Its job is to scale all linetypes relative to paper space. Say, for example, the paper space scale is $1/4" = 1'$ (that's 1:48). By setting PsLtScale to 48, BricsCAD automatically displays linetypes 48 times larger in paper space than in model space.

The Special Case of Polylines

Then there's a trick to employ when it comes to polylines. To understand the problem, it helps to know how BricsCAD generates linetypes. In an attempt to apply linetypes as nicely as it can, the software generates the linetype based on (a) the length of the object and (b) the linetype scale factor.

Essentially, BricsCAD starts at one end of the object, and then works its way to the other end. The program then centers the linetype pattern so that it looks nice and even at both ends. You'll never see the linetype pattern abruptly ending midway at one end of the object. Here is how a line looks with a linetype applied centered:



Consider, then, the polyline. While it looks like one long connected line-arc-spline, it contains many vertices, even when you do not see them. Each vertex signals the start and end of a line or arc segment. BricsCAD faithfully restarts the linetype pattern each time it encounters a vertex.

When the vertices are close together, BricsCAD never gets around to restarting the pattern, resulting in a polyline that looks solid, or continuous. This drives some people nuts, like cartographers who use polylines for drawing contours. The solution is to use the **PlineGen** system variable. When turned off (the default), BricsCAD works as before, generating the linetype from vertex to vertex. When changed to on, BricsCAD generates the linetype from one end of the polyline to the other end — ah, instant relief! (This problem does not affect splines.)

- Describe the linetype with any words you want up to 47 characters long.

Linetype description: . _ . _ . _ . _

A good descriptive text would be the pattern you plan to create, using dots, underlines, and spaces.

- Finally! You get to define the linetype pattern.

Linetype definition (positive numbers for lines, negative for spaces):

A,

But, what's this A? The letter **A** forces the linetype to *align* between two endpoints. That's what causes the linytypes start and stop with a dash, adjusted to fit. The **A** could also stand for "actually" because, actually, I don't have a choice when I create a linetype on-the-fly: BricsCAD forces the letter A on me.

Type the codes after the A, as follows:

A, .25,-.1,0,-.1

I could go on for a total of 78 characters but I won't.

- I press **Enter** to end linetype definition, and I'm done.

Linetype "dit-dah" was defined in C:\Users\...\support\default.lin.

Linetype: ? to list/Create/Load/Set: (*Press ENTER.*)

Well, not quite done. I still need to test the pattern. By the way, new linytypes are added to the *end* of the *default.lin* file.

Testing the New Linetype

It is important to always test a new customization creation. As simple as they are, linytypes are no exception. Test the Dit-Dah pattern, as follows:

- Use the **Linetype Load** command to load the pattern into drawing:

: **-linetype**

Linetype: ? to list/Create/Load/Set: **L**

Enter linetype to load: **dit-dah**

- Up pops the Select Linetype File dialog box. Select *default.lin*, and then click **Open**. BricsCAD confirms:

Linetype DIT-DAH loaded.

- Use the **Set** option to set the linetype, as follows:

?/Create/Load/Set: **s**

New entity linetype (or ?) <BYLAYER>:

- Here you can type either the name of a loaded linetype (such as "dit-dah") or type ? to see which linytypes are already loaded.

- This time, get serious and set the current linetype to "dit-dah":

?/Create/Load/Set: **s**

New entity linetype (or ?) <BYLAYER>: **dit-dah**

?/Create/Load/Set: (*Press ENTER.*)

- Now, draw a line, and appreciate the linetype it is drawn with. Your debugging session is over.

CREATING LINETYPES WITH TEXT EDITORS

You can edit the *default.lin* linetype file directly to create custom linetypes. Here's how:

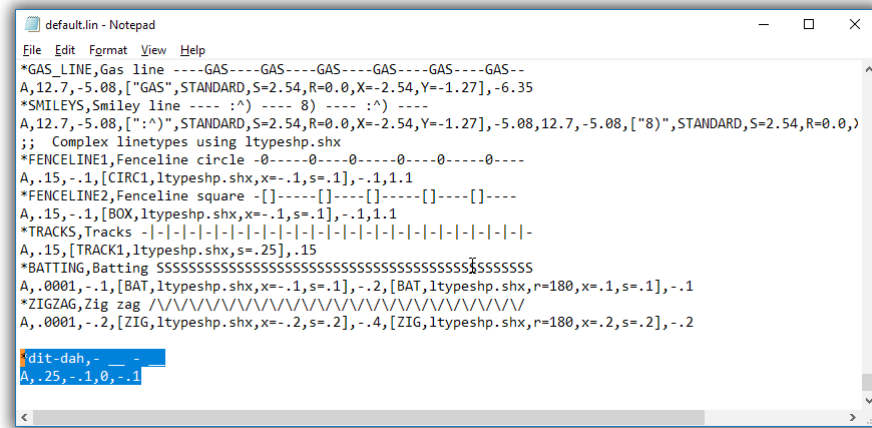
1. Start a text editor (not a word processor), such as NotePad in Windows, Text Editor in Linux, or TextEdit in MacOS.

2. Open the *default.lin* file. You find it in one of the following locations:

Windows — C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20\en_US\support

Linux — /home/<login>/Bricsys/BricsCAD/V20/en_US/Support

MacOS — /Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/Support



Adding linetype definitions with a text editor

3. When you scroll down to the end of the file, you see the Dit-Dah pattern you defined as per the earlier tutorial.
4. You can modify an existing linetype, or add a new linetype. The process is exactly the same as when you did it within BricsCAD, with two exceptions: (1) BricsCAD isn't there to prompt you; and (2) you don't need to use the "A" prefix.
5. Save the *.lin* file with the same name (*default.lin*) or a new name, then test it within BricsCAD.

TIP If you can't be bothered burrowing all the way down to the `C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20\en_US\support` folder, use the following trick:

1. Start with the **-Linetype** command's **Create** option.
2. Enter a nonsense name when prompted for "Name for new linetype," like ASDF.
3. When you press **Enter**, the Create or Append Linetype File dialog box appears.
4. Right-click *default.lin* and then choose **Open With | Notepad**.

Notepad opens with the *default.lin* file, ready for editing.

Linetype Format (.lin)

The linetype definition consists of two lines of text:

LINE 1: HEADER

Line one is the header, such as `*dit-dah, . _ . _ . _` where:

- `*` Asterisk indicates the start of a new linetype definition. DIT-DAH Name of the linetype.
- `,` Comma separates the name from the description.
- `. _ . _` Dot-space-spline pattern describes the linetype (to a maximum of 47 characters), which is displayed by the `Linetype ?` command.

LINE 2: DATA

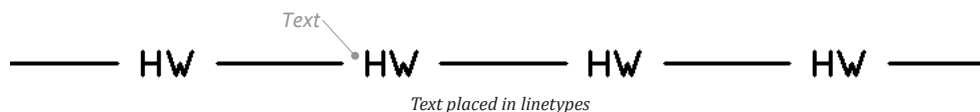
Line two is the data, such as `A, .25,-.1,0,-.1`, where:

- `A` “A” is the alignment flag, which forces BricsCAD to start and end the linetype with a line segment adjusted to the overall length of the object.
- `.25` First number specifies the length of dashes when `LtScale = 1.0`. Every linetype data line must begin with a dash.
- `-.1` Numbers with negative signs specify the length of gaps when `LtScale = 1.0`; every linetype data line must follow the initial dash with a gap.
- `0` Zeros draw dots.

You can use a semicolon (;) to prefix any line as a comment line. Anything after the semicolon is ignored by BricsCAD.

COMPLEX (2D) LINETYPES

“Complex” linetypes include text characters. Truth be told, that’s all they are: text — or, more accurately, *shapes*. See Chapter 17 for full information on shapes.



The complex linetype is a mixture of text and simple linetype codes — the dash, gap, and dot you learned of earlier. The text are characters that can come from any `.shx font` file.

You could make complex patterns using ASCII art. For example, a square can be made from a pair of square brackets to create the box effect: [and]. A zig-zag linetype can use the slash and backslash characters, / and \.

Here, ASCII characters created smiley faces:

_____ :^) _____ B) _____ :^) _____ B) _____

Symbols created from punctuation and other characters

The text used in complex linetypes come from `.shx` file. The shape file format is arcane, written to be a highly efficient form of symbol for the slow running personal computers of the 1980s. Shapes were quickly superseded by blocks, but remain on the scene due to their use in linetypes and so on.

If you want to write a custom shape definitions, see chapter 18. Be warned, however, that coding shapes requires a knowledge of trigonometry. Jason Bourhill recommends that you use the MkShape (make shape) utility provided by Martin Drese from

<https://www.bricsys.com/applications/a/?express-tools-a589-a11002>.

The Express Tools collection also contains MklType utility that makes linetypes, without needing to code them.

EMBEDDING TEXT IN LINETYPES

The hot water linetype combines a dash and a gap with the letters HW using the Standard text style (which uses the `arial.ttf` font file).

Here is the code for hot water:

```
*HOT_WATER, Hot Water ----HW----HW----HW----HW----HW----HW--  
A,1.0,-.25,["HW",STANDARD,S=.2,R=0.0,X=-0.1,Y=-0.1],-.40
```

Much of this looks familiar, with the exception of the colored text between the square brackets, shown in boldface. That is how text is embedded in linetypes, and here's what it means:

Text

"HW" Prints the letters between the dashes.

Text Style

STANDARD Applies this text style to the text. This is optional; when missing, BricsCAD uses the current text style, whose name is stored in system variable **TextStyle**.

Text Scale

S=.2 S specifies the text size or scale factor. It can mean one of two things:

- When the **height** defined by the text style is 0 (as is often the case), then **S** defines the height; in this case, the text is drawn 0.2 units tall).
- When the text style height is not 0, then this number *multiplies* the text style's height; in this case, the text is drawn at 0.2 times (or 20%) of the height defined in the text style.

Text Rotation

- R** Rotates the text relative to the direction of the line; $R=0.0$ means no rotation. The default measurement is degrees; other forms of angular measurement are:
- **r** for radian, such as $R=1.2r$ (there are 2π radian in a circle).
 - **g** for grad, such as $R=150g$ (there are 400g in a circle).

The R parameter is optional and so can be left out. In this case, BricsCAD assumes zero degrees.

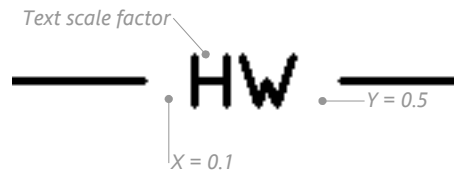
Absolute

- A** Rotates the text relative to the x-axis (the “A” is short for *absolute*). This ensures the linetype text is drawn so that it is always oriented in the same direction, no matter the angle of the line. Rotation is always performed within the text baseline and capital height. That’s so the text isn’t rotated way off near the orbit of Pluto.

The A parameter is optional and can be left out.

X and Y Offset

- X** Shifts the text in the x-direction from the linetype definition vertex, which helps center the text in the line. For example, $X=-0.1$ shifts it to the right by 0.1 units.
- Y** Shifts the text in the y-direction from the linetype definition vertex. $Y=-0.1$ shifts text down by 0.1 units. In both cases, the units are in the linetype scale factor, which is stored in system variable LtScale.



Parameters for positioning text in a linetype

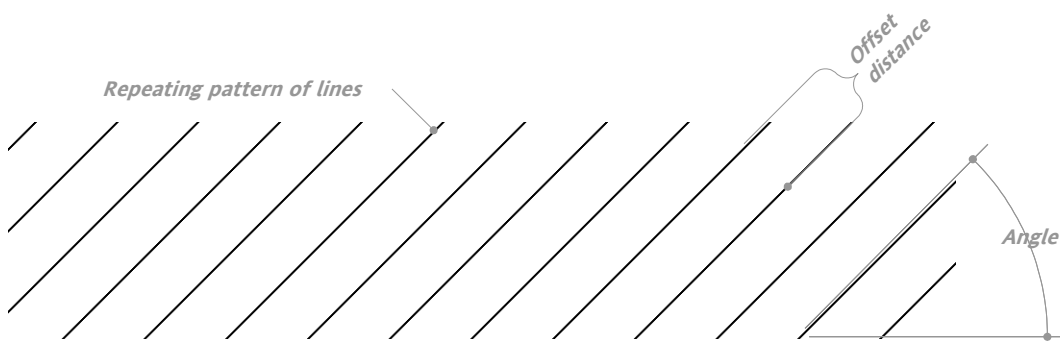
Summing up, you can create a text-based linetype with a single parameter, such as [“HW”], or you can exercise fine control over the font, size, rotation, and position with the six parameters listed above. BricsCAD can work with any *.shx* font file you have on your computer.

Parameter	Meaning	Optional?	Example
""	Text	Required	“HW”
<i>filename</i>	Name of text style	Default style	STANDARD
S=	Text size or scale factor	Style height	S=0.5
R=	Rotation angle	Angle = 0	R=45
A=	Absolute rotation angle	Angle = 0	A=0
X=	Horizontal offset	Offset = 0	X=0.1
Y=	Vertical offset	Offset = 0	Y=-0.1

BricsCAD does not recognize the U parameter, used by recent releases of AutoCAD to keep shapes upright.

Patterning Hatches

Despite seemingly complex, hatch patterns consist of the same three basic elements as do linetypes: dashes, gaps, and dots. The pattern repeats itself, which is done by specifying an offset distance and an angle, as illustrated below:



CHAPTER SUMMARY

The following topics are covered in this chapter:

- Finding the source of hatch patterns
- Creating custom hatch patterns
- Understanding the *default.pat* file

QUICK SUMMARY OF PATTERN DEFINITIONS

Hatch patterns are stored in .pat files, and are applied with the Hatch command. Each hatch pattern definition consists of at least two lines of text, a header that labels the pattern, followed by one or more lines of data that describe the pattern. (Gradients are hard coded, and cannot be customized.)

LINE 1: HEADER

Example: *Name,Description

* (asterisk) — indicates the start of the hatch pattern

Name — names the pattern

, (comma) — separates the name from the description

Description — describes the pattern

LINE 2: HATCH PATTERN DATA

Example: 45, 0,0, 0,0.125

45 — specifies the angle of the line segment

0,0 — specifies x,y coordinates of the start of the line segment.

0,.125 — specifies ending coordinates of the line segment.

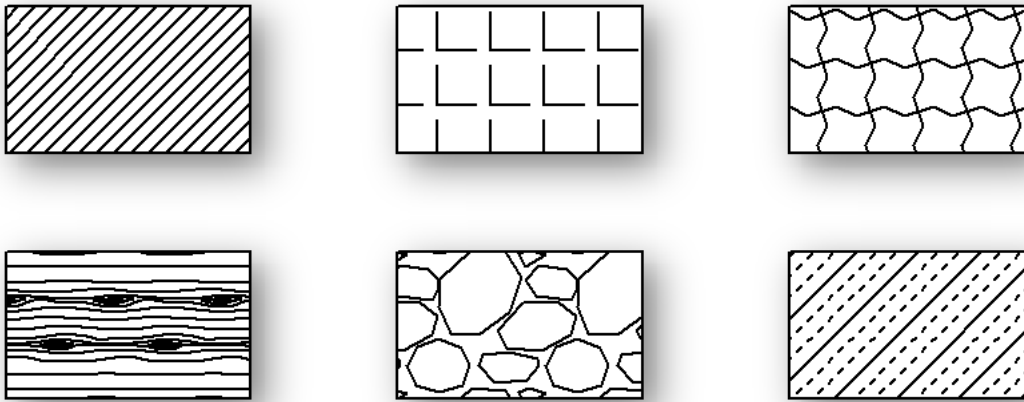
dash definition — defines dashes in the line segment using the same code as in linetypes:

- Positive number draws a dash, such as 0.25
- Zero (0) draws a dot
- Negative number draws a gap, such as -0.25.

NOTE

To include comments in the .pat file, prefix lines with a semi-colon (;).

The result is like the samples illustrated below.



Examples of hatching patterns provided with BricsCAD

BricsCAD cannot create hatch patterns made of circles and other nonlinear objects. BricsCAD also can solid-fill and gradient-fill areas in any color.

BricsCAD comes with 87 hatch patterns, plus solid fill and nine gradient fill patterns. Even so, your office drafting standard may well require additional patterns. In this chapter, we look at how to create hatch patterns, and edit existing ones.

Where Do Hatch Patterns Come From?

The **-Hatch** command creates hatch patterns at the command line; **Hatch** displays a dialog box to do the same thing. Unlike linetypes, the pattern file is loaded automatically the first time you use the Hatch or -Hatch commands (formerly the BHatch command). Hatch patterns are defined in files external to BricsCAD:

- ▶ *default.pat* contains the hatch patterns you use most commonly
- ▶ *iso.pat* contains hatch patterns as defined by the ISO
- ▶ Other *.pat* files can also contain hatch patterns, but it is easier to keep all patterns in a single file

These are the folders in which BricsCAD stores its pattern files (replace <login> with your log in name):

Windows: C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20\en_US\support
Linux: /home/<login>/Bricsys/BricsCAD/V20/en_US/Support
MacOS: /Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/Support

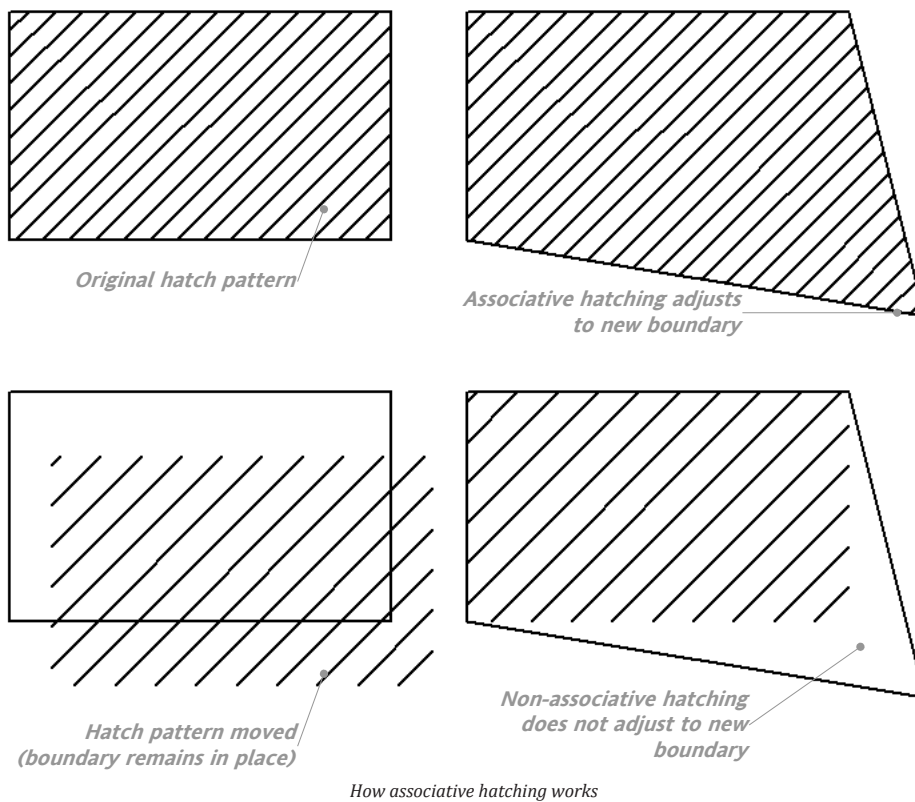
HOW HATCH PATTERNS WORK

When you apply hatching to an area, BricsCAD generates a repeating pattern of parallel lines and gaps based on the definition in the *.pat* file. The pattern comes to a stop when it reaches a boundary; if BricsCAD cannot detect a boundary, it refuses to place the pattern.

Once the hatch is in place, you can use the Move command to move the hatch pattern elsewhere in the drawing, if you so chose.

BricsCAD can create *non-associative* and *associative* hatch patterns; the **Associative** toggle is found in the Options area of the Hatch and Gradient dialog box.

- ▶ *Non-associative* means the area of the pattern is fixed. When you change the boundary, the pattern remains in place, as illustrated below. This property is useful when you want the pattern to remain fixed.
- ▶ *Associative* hatching means the pattern's shape updates as you change the boundary.

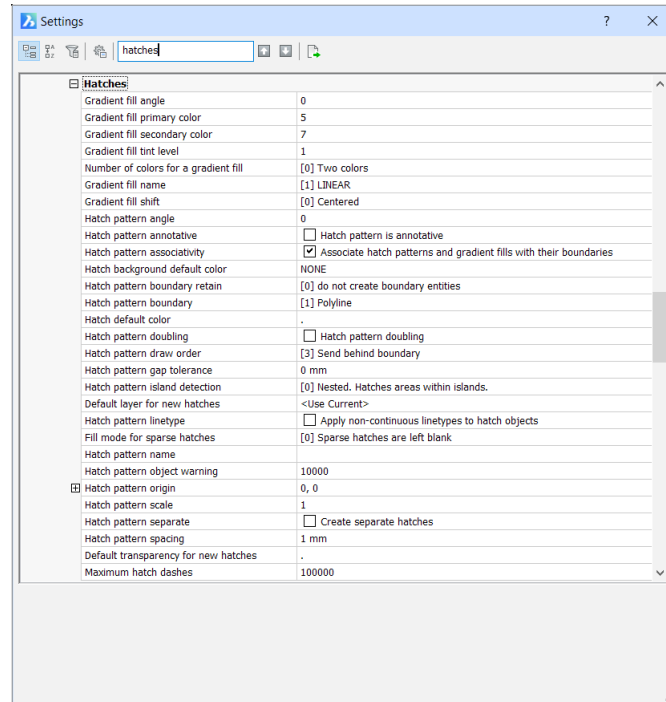


In either case, you can move the pattern independent of its boundary. This is because BricsCAD treats both kinds of hatches as blocks. You can use the **Explode** command to explode blocks into their constituent lines.

You can use the **HatchEdit** command or the **Properties** command to edit parameters of hatch, solid, and gradient patterns.

System Variables that Control Hatches

BricsCAD has system variables that control how hatches are created, and report their most recent settings. In the Settings dialog box (**Settings** command), enter “hatches” in the search field:



Settings that affect hatches

Creating Custom Hatch Patterns

BricsCAD provides you with two ways to create custom hatch patterns: (a) simple patterns defined with the Hatch and -Hatch commands; and (b) edit the *default.pat* file or write new *.pat* files with a text editor. We look at both methods in this chapter. Unlike linetypes, you cannot create hatch patterns in Drawing Explorer.

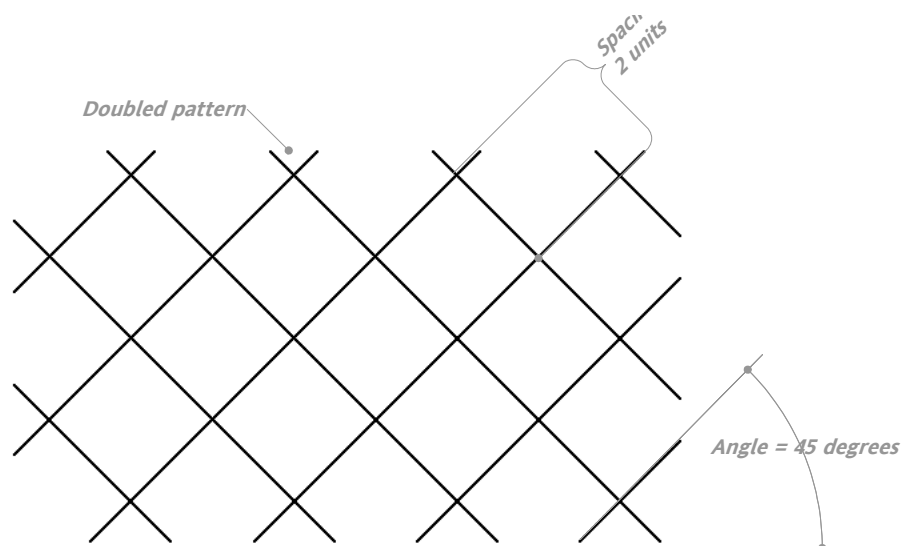
When you create simple hatch patterns with the Hatch command, BricsCAD does not, unfortunately, save the fruit of your labor (unlike when you create a custom linetype with Linetype.) For this reason, think of the first method of creating custom hatch patterns on-the-fly.

-HATCH COMMAND

Your options for creating a hatch on-the-fly are limited to simple patterns. Using the **-Hatch** command, you access the Properties option, followed by User defined, as follows:

1. Enter the **-Hatch** command (formerly the Hatch command, with no hyphen prefix):
: **-hatch**
Current hatch pattern: ANSI31
2. Select the **P** (properties) option, and then the **U** (user defined) option:
Specify internal point or: Properties/Select/Remove islands/Advanced/Draw order/
Origin : **p**

Enter a pattern name or: ? to list patterns/Solid/User defined/<ANSI31> : **u**
3. Specify three parameters for custom hatch patterns: Angle, Spacing, and Crosshatching. First, the angle:
Proceed/Angle for lines <0>: **45**
4. Second, the spacing between parallel lines.
Space between standard pattern lines <1.0000>: **2**
5. Third, decide if you want the pattern *crosshatched*. That means the pattern is repeated at 90 degrees to the first one.
Cross-hatch area? Yes/No/<No>: **y**
6. Finally, you select the object or boundary to hatch:
Specify internal point or [Properties/Select/Remove islands/Advanced/
Draw order/Origin]: *(Pick a point in the drawing to apply the pattern.)*



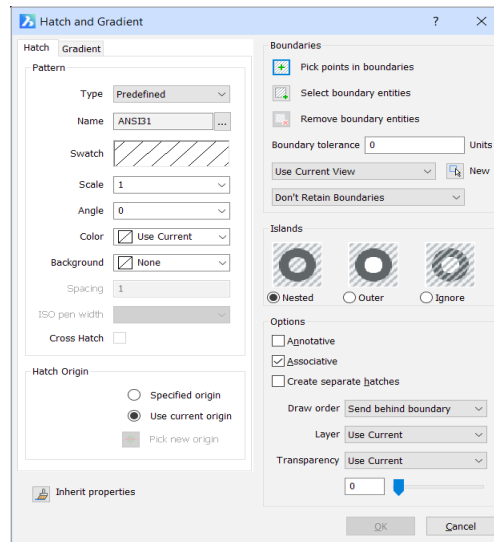
Defining a custom hatch pattern inside BricsCAD

BricsCAD draws the pattern, but — as mentioned earlier — the custom hatch is not saved to a *.pat* file.

HATCH COMMAND

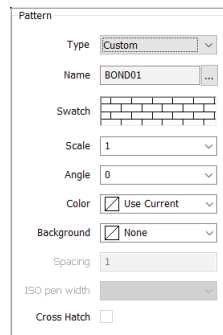
Creating custom hatch patterns with the **Hatch** command is more like filling out a form:

1. From the **Draw** menu, select **Hatch**. (Alternatively, type **Hatch** at the command prompt). Notice the Hatch and Gradient dialog box.



Hatch and Gradient dialog box


2. From the **Type** drop list, select **Custom**.



Location of parameters for user-defined hatch patterns in the dialog box

3. BricsCAD allows you to enter values for **Angle**, **Spacing**, and **Cross Hatch**, as well as color of the cross-hatching and the background color. Enter values such as these:

Angle	45
Spacing	2
Cross Hatch	Yes

4. Click the  **Pick points in boundaries** button, and then select the area you want hatched.
5. Press **Enter** to return to the dialog box, and then click **OK**. BricsCAD applies the custom pattern.

Understanding the .pat Format

Let's dig into the contents of the *default.pat* file to get a better understanding of how a pattern is constructed.

1. Start a text editor (not a word processor), such as Notepad on Windows, Text Edit on Linux, or TextEdit on Mac.
2. In Windows, open the *default.pat* file from the `|Users|<login>|AppData|Roaming|Bricsys|BricsCAD|V14x64|en_US|support` folder.
In Linux, open the *default.pat* file from the `/home/<login>/Bricsys/BricsCAD/V20/en_US/Support` folder.
In MacOS, open the *default.pat* file from the `/Users/<login>/Library/Preferences/Bricsys/BricsCAD/V20x64/en_US/Support/default.lin` folder.
3. Scroll down a bit, and take a look at the seemingly-incomprehensible series of numbers and punctuation contained by this file. I've reproduced the first dozen lines here:

```
; Note: Dummy pattern description used for 'Solid fill'.
*SOLID, Solid fill
45,0,0,0,0.1

*ANSI31,ANSI Iron
45, 0,0, 0,0.125

*ANSI32,ANSI Steel
45, 0,0, 0,0.375
45, .176776695,0, 0,.375
```

COMMENT AND HEADER LINES

The definition of a hatch pattern consists of two or more lines of text. The first line is called the *header*, such as `*SOLID, Solid fill`.

Comment

The semicolon (;) indicates a comment line, such as

```
; Note: Dummy pattern description used for 'Solid fill'.
```

That lets you include notes to yourself that are ignored by BricsCAD.

Start of Definition

The asterisk (*) is important, because it signals to BricsCAD the start of a new hatch pattern definition.

Pattern Name

Following the asterisk comes the name for the hatch pattern, such as `SOLID`. The name must be unique in the file. If it isn't, BricsCAD uses the first pattern it finds by that name.

The comma following the name merely separates the name from the description. The comma is optional; it doesn't have to be there: a space works just as well.

Description

The text following the pattern name is the description displayed by the **-Hatch ?** command, such as “Solid fill.” This description is also optional, but highly recommended. You are limited to a maximum of 80 characters for the name, comma, and the description. If you need more room for the description, use comment lines, such as:

```
; Note: Dummy pattern description used for 'Solid fill'.  
*SOLID, Solid fill
```

THE HATCH DATA

With the comment lines and the header line out of the way, let’s get down to the nitty-gritty hatch pattern data and how it is coded. Lines 2 and following are the data, such as:

```
0, 0,0, 0,.275, .2,-.075 90, 0,0, 0,.275, .2,-.075
```

Every line of data uses the same format:

```
angle, xOrigin, yOrigin, xOffset, yOffset [, dash1, ...]
```

Angle

Angle is the angle at which this line of hatch pattern data is displayed. The “0” means the hatch line is drawn horizontally; a “90” means the line is drawn vertically, and so on. A comma (,) separates the numbers.

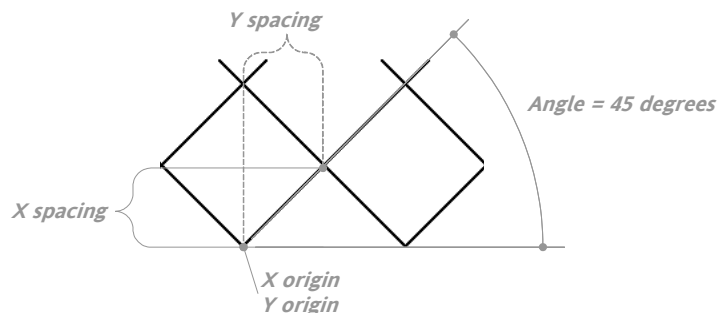
xOrigin and yOrigin

The *xOrigin* specifies that the first line of the hatch pattern passes through this x-coordinate. The value of the *yOrigin* means that the first line of the hatch pattern passes through this y-coordinate.

xOffset and yOffset

The *xOffset* specifies the distance between line segments, aka the *gap* distance. You use this parameter only to specify the offset for vertical or diagonal lines (To specify the distance between dashes, use the **dash1** parameter.) In most hatch patterns, *xOffset* has a value of 0.0. Even though this parameter is rarely used, it is not optional.

The *yOffset* is the vertical distance between repeating lines, and is used by every hatch pattern.



Defining hatch patterns through spacing, angle, and origin

Dash1,...

dash1 defines the dashes in the hatch pattern line (the code is the same as for linetypes):

- ▶ A positive number, such as 0.25, is the length of the dash.
- ▶ A 0 draws a dot.
- ▶ A negative number, such as -0.25, draws a gap.

TIP The dot drawn by the hatch pattern may create a problem when it comes time to plot. If you find that the dots in a hatch pattern are not printed, use a very short line segment, such as 0.01, instead of a 0.

When you are finished editing a pattern, save the *.pat* file.

ADDING SAMPLES TO THE HATCH PALETTE

BricsCAD adds visual samples of your custom hatch patterns to the palette automatically. You can have more than one *.pat* file; however, the additional ones are limited to one pattern definition per file, and the definition's name must match the file name.

TIPS ON CREATING PATTERN CODES

Some miscellaneous comments on hatch pattern coding:

Tip 1: Hatch pattern lines are drawn infinitely long. What this means is that BricsCAD draws the line as long as necessary, as long as it reaches a boundary. BricsCAD will not draw the hatch pattern unless it does find a boundary.

Tip 2: At the very least, each line of pattern code must include the **angle**, **x-** and **y-origin**, and the **x-** and **y-offset**. This draws a continuous line.

Tip 3: The **dash1** parameter(s) is optional but when used draws a line with the dash-gap-dot pattern.

Tip 4: It's a lot easier for someone else (or you, six months from now) to read your hatch pattern code if you use tabs and spaces to format the code into nice columns.

Tip 5: To change the angle of a hatch pattern upon placing it in the drawing, you've got a couple of options:

- Specify the angle during the **Hatch** command.
- Set the angle in system variable **SnapAng**. The effect of **SnapAng** on the hatch pattern angle is additive: if the hatch pattern defines the lines drawn at 45 degrees and **SnapAng** is 20 degrees, then BricsCAD draws the hatch lines at 65 degrees. For example:

```
: snapang  
New current angle for SNAPANG <0>: 20
```

The **x-offset** and **y-offset** parameters are unaffected by the angle parameter, because **x-offset** is always in the direction of the line and **y-offset** is always perpendicular (90 degrees) to the line.

If you are uncomfortable using system variables, then the **Snap** command provides the same opportunity via the **Rotate** option:

```
: snap
Snap is off (x and y = 0.5000): ON/Rotate/Style/Aspect/<Snap spacing>: r
Base point for snap grid <0.0000,0.0000>: 1,1
Rotation angle <0>: 45
```

Tip 6: You can specify a weight (or line width) for hatch patterns line. If you wish, you can also make thick-looking patterns by using closely spaced lines, like this:

```
*Thick_Line, Closely spaced lines
0, 0,0, 0,.25 0, 0,.01, 0,.25 0, 0,.02, 0,.25
```

Tip 7: To draw dash and gap segments at an angle, use the sine of the angle in degrees, like this:

<u>Angle</u>	<u>Dash length (sine)</u>
0	0
30	0.433
45	0.707
60	0.866
90	1.0

Tip 8: You cannot specify arcs, circles, and other round elements in a hatch pattern file. Everything consists of straight lines and dots. To simulate circular elements, use a series of very short dashes.

Decoding Shapes & Fonts

BricsCAD uses .shx files for fonts, shapes, GD&T symbols, and complex linetypes. You can create source .shp files, the subject of this chapter. BricsCAD, unfortunately, lacks the compiler needed to convert .shp to the compiled .shx files that BricsCAD works with.

BricsCAD displays fonts from both TrueType (.ttf) and AutoCAD shape (.shx) files.

In addition to using .shx files for displaying fonts, BricsCAD use a second type of .shx file for simple blocks-type entities known as “shapes,” and so includes the **Load** and **Shape** commands for loading and placing them.

(Explanations updated by Jason Bourhill of CAD Concepts, www.cadconcepts.co.nz.)

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Understanding shapes with fonts, complex linetypes, shapes, and GD&T symbols
- Learning about shape files
- Detailing the shape file format

QUICK SUMMARY OF SHAPE DEFINITIONS

Shapes and fonts are defined by .shp files, which need to be compiled into .shx files. A shape definition consists of at least two lines of text, a header that labels the shape or font, followed by one or more lines of data that describe the shape. The end of the data section is signified with a zero.

LINE 1: HEADER

*130,6,NAME

* (asterisk) — indicates the start of the shape definition.

130 — numbers the shape; fonts use the character's ASCII number. Range is 1-255.

6 — specifies the total number of data bytes.

NAME — names the shape; must be in uppercase, and a maximum of 16 characters

LINE 2: DATA

014,002,01C,001,01C,0

Shape data consists of vector and instruction codes. Vector codes define movement and drawing in 16 directions:

0 — first digit (always 0) indicates the number is hexadecimal

1 — second character specifies the vector length, and ranges from 1 through F (15 units).

4 — third character specifies the direction of the vector

INSTRUCTION CODES

Hexadecimal	Decimal	Description
000	0	End of shape definition.
Basic Draw and Move		
001	1	Begin draw mode (pen down).
002	2	End draw mode (pen up).
Scaling		
003	3	Divide vector lengths by next byte.
004	4	Multiply vector lengths by next byte.
Memory		
005	5	Push current location onto stack.
006	6	Pop current location from stack.
Draw Subshape		
007	7	Draw subshape number given by next byte.
Advanced Draw and Move		
008	8	X,y displacement given by next two bytes.
009	9	Multiple x,y displacements; terminated with (0,0) code.
Arcs		
00A	10	Octant arc defined by next two bytes.
00B	11	Fractional arc defined by next five bytes.
00C	12	Arc defined by x,y displacement and bulge.
00D	13	Multiple bulge-specified arcs.

Fonts, Complex Linetypes, and Shapes

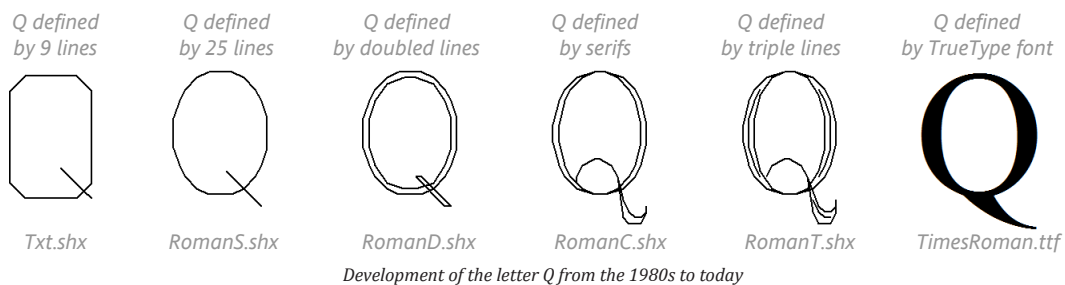
BricsCAD uses shapes for fonts, the text in complex linetypes, and shapes.

SHX FONTS

In the early days of CAD, fonts were coded to be highly efficient. Computers didn't have much horsepower, and text was one of the slowest parts of the drawing display. To solve the problem of vector fonts taking a long time to display on the slow computers of the 1980s, Autodesk invented the SHX format: the simpler the font, the fewer the lines, the faster the display.

The simplest font of all, *Txt.shx*, looked ugly, but was able to draw most characters with just eight lines. The drawback to SHX-based shapes, however, is that they are not well-suited to defining the complex curves that truly represent fonts, nor can they be properly filled.

As computers became faster over time, the number of lines used to draw characters increased. Eventually, Apple's TrueType font technology allowed for truly smooth looking and fully filled fonts, even in CAD drawings and on plots. The gallery below illustrates the development of the letter Q, from the original *Txt.shx* to the more recent *TimesRoman.ttf*.



These are the file names of the SHX files included with BricsCAD:

SHX Font File	Meaning
complex.shx	Serif font
hangul.shx	Korean font
isocp.shx	ISO standard font (European)
italic.shx	Single-stroke italic font
italicc.shx	Double-stroke italic font
italict.shx	Triple-stroke italic font
japanese.shx	Japanese font
monotxt.shx	Mono-spaced font (every character takes up the same width)
romanc.shx	Triple-stroke serif font
romand.shx	Double-stroke serif font
romans.shx	Single-stroke serif font
romant.shx	Triple-stroke serif font (same as RomanC)
simplex.shx	Non-serif font
trad_chin.shx	Chinese font
txt.shx	Minimal non-serif font

About Fonts in BricsCAD

To be compatible with old drawings, BricsCAD supports the use of original *.shx* fonts as well as today's *.ttf* TrueType fonts. TTF files are included by default with the Windows and MacOS operating systems, and with some Linux systems. BricsCAD does not support PostScript fonts, such as those provided as PFA and PFB files.

To load a font file into a drawing, use the **Style** command (a.k.a. Drawing Explorer), and then place text with the **Text**, **MText**, and other text-related commands. BricsCAD accesses TrueType fonts from each operating system's default font folder:

Windows: *C:\Windows\Fonts*
Linux: */usr/share/fonts/truetype*
MacOS: */Users/Library/FontCollections*

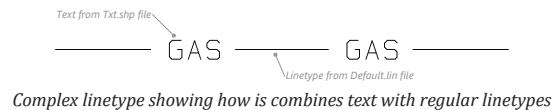
If a drawing displays fonts incorrectly, then the problem lies with BricsCAD not finding the location of the source font file. Here you have two solutions:

- ▶ Use **Settings | Files | Support file search path** to add paths
- ▶ Add the missing font files to the existing paths

TIP To obtain a list of all fonts used by a drawing, run the **eTransmit** command.

USING SHX IN COMPLEX LINETYPES

Complex linetypes use shapes for the text portion. The position and size of the text is defined in the *default.lin* and *iso.lin* linetype files, while the characters themselves are defined by the *arial.ttf* font, by default. The font used is determined by the Standard text style; change the style, and the linetype font changes.



They are loaded and placed with the **Linetype** command. For details, see the chapter on linetypes

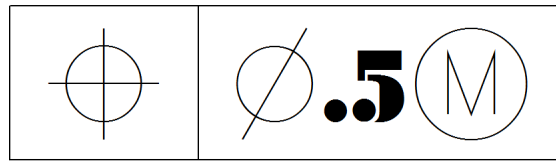
SHX IN SHAPES

Shapes are an early form of block (symbol). Like fonts, they displayed very quickly on the slow computers of the 1980s. Unfortunately, they were very hard to code; today it is so much easier to use blocks, and so shapes are no longer used for symbols. Nowadays, shapes are used only by the Tolerance command.

Shapes use a format of the SHX file that is nearly identical to that of fonts. Shapes must first be loaded into the drawing with the **Load** command, and then placed in the drawing with the **Shape** command.

SHX IN GD&T

GD&T (geometric dimensioning and tolerances) symbols are used for machining parts precisely.



Tolerance symbols

The symbols are placed by the **Tolerance** command, and are based on shapes from the *gdt.shx* file.

SHAPE COMPATIBILITY WITH AUTOCAD

For fonts, complex linetypes, shapes, and tolerances, BricsCAD can use any *.shx* file that AutoCAD can use.

Both CAD packages use a *fmp* file (short for “font map”) to substitute similar looking *.shx* fonts for those found in each other’s drawings. At time of writing, BricsCAD specifies *default.fmp* but has not implemented it.

About Shape Files

The shape file format is arcane, written to be a highly efficient form of symbol for the slow running personal computers of the 1980s. Shapes were quickly superseded by blocks, but remain on the scene due to their use in linetypes and so on.

Coding custom shape definitions requires a knowledge of trigonometry. Jason Bourhill recommends that you use the MkShape (make shape) utility provided by Martin Drese from <https://www.bricsys.com/applications/a/?express-tools-a589-al1002>.

There are two kinds of files used for shapes: *.shp* and *.shx*. The differences between them are as follows:

- ▶ **.shp** are shape *source* files. When you write or edit a shape or font, you work with the *.shp* file. A portion of a typical *.shp* file looks like this:

```
*130,6,TRACK1
014,002,01C,001,01C,0
```

- ▶ **.shx** are *compiled* shape files. These are the files that are loaded into BricsCAD for use with fonts, tolerances, and so on.

TIPS You can use AutoCAD or the Compile utility from Bricsys to compile shape files you create using information in this chapter. The Compile utility is part of the free BCadTools collection by Torsten Moses from <https://www.bricsys.com/applications/a/?bcadtools-freeware-a335-a1528>

You cannot edit .shx files, unless you have access to a shape decompiler program. Autodesk offers the DumpShx.Exe utility in AutoCAD's Express folder, or else search online for decompilers.

THE SHAPE FILE FORMAT

Autodesk defined two formats for the shape file: one for *shapes* (simple blocks), and one for *fonts*. The difference between the two types is subtle: the font version of the file includes a code 0 to alert the CAD system to treat the file as a font definition. When the 0 is missing, the file is treated as a shape definition.

BricsCAD can load both forms of shape file, as shapes with the Load command, and as fonts with the Style command. You cannot, unfortunately, distinguish between the two easily. One way is to guess by the file or folder names. For instance, *italic.shx* is clearly a font file, while *ltypeshap.shx* is probably a shape file. Other file names can be vague: *symusic.shx* seems like a shape file, but in fact is a font file (musical symbols). BricsCAD does not warn you if you load the wrong kind of shape file with the Load command; in contrast, the Style command lists only font-related SHX files.

Here are some aspects about the shape file format:

- ▶ Shape files typically define one or more shapes, up to 258 in total.
- ▶ Font files typically defines all the characters for a single font, such as A-Z, a-z, 0-9, and punctuation.
- ▶ Unicode font files can have up to 32,768 definitions.

Like many other customization files, shape definitions consist of two or more lines. The first line is the header, which labels the shape, while the second (and following) lines define the shape through codes. The final code in each definition is 0, which is called the *terminator*.

The general format of a shape definition consists of a header line, followed by one or more definition lines:

```
*shapeNumber, totalBytes, shapeName  
byte1, byte2, byte3, . . . , 0
```

Each line can be up to 128 characters in length; shape files with longer lines will not be compiled. Each definition is limited to a total of 2,000 bytes.

You can use blank lines to separate shape definitions and the semicolon (;) to include comments in the file.

HEADER FIELDS

The following describes the fields of the shape's header description:

Definition Start

*130,6,TRACK1

The asterisk signals AutoCAD that the next shape definition is starting.

shapeNumber

*130,6,TRACK1

Each shape requires a unique number by which it is identified. For fonts, the number is the equivalent ASCII code, such as 65 for the letter A.

TIP The *shapeNumbers* 256, 257, and 258 are reserved for the degree, plus-or-minus, and diameter symbols.

totalBytes

*130,6,TRACK1

After defining the shape, you have to add up the number of bytes that describe the shape, including the terminator, 0. Makes no sense to me. There is a limit of 2,000 bytes per shape definition. Unicode shape numbers use two bytes each.

shapeName

*130,6,TRACK1

Shape names can be mixed case. Maximum length of the name is 16 characters; excess characters are truncated.

DEFINITION LINES

The header line is followed by one or more lines that define the shape or font. This is the nitty-gritty part of shape files, and you will now see why they are rarely used anymore.

bytes

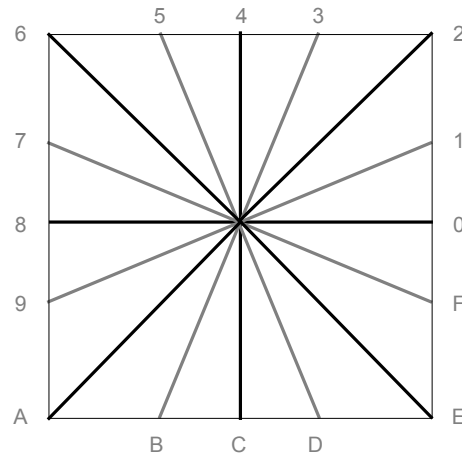
014,002,01C,001,01C,0

The shape is defined by "bytes," called that because each code is a single byte (the computer measurement) in size. Bytes define vector lengths and directions, or instruction codes. They can be in decimal (base 10) or hexadecimal (base 16) format. Definition lines are a maximum of 128 characters long (including commas), and a maximum of 2,000 bytes overall (not including commas). The last definition line ends with a 0.

TIP When the first character of a byte is a 0, the two characters following are in hexadecimal, such as 00C (12, in decimal).

VECTOR CODES

Vector codes describe how the shape is drawn. They define movement (pen up) and drawing (pen down). Vector codes are limited to 16 directions, increments of 22.5 degrees, as shown by the figure:



Vectors defining direction and distance

Notice that the lengths are not radial. Diagonal vectors such as 2 and E are 1.414 times longer than the orthogonal vectors, such as 4 and 0. (Recall that 1.414 is the square root of 2.)

Vector codes are always in hexadecimal notation, such as 02C:

- First character is always 0 to indicate that the number is in hexadecimal.
- Second character is the vector length, ranging from 1 through F (15 units).
- Third character is the direction, as noted by the figure above.

Thus, 02C would draw a line 2 units long in the -y direction (downward). By now, you can see that you need to understand hexadecimal notation.

Hexadecimal Conversion

Autodesk used hexadecimal (base 16) notation, because it was more efficient for use by computers in the days when CPUs were slow. Back then, programmers did a lot of work to minimize the load on the computer. Here is a conversion table between decimal and hexadecimal numbers:

<u>Decimal</u>	<u>Hexadecimal</u>
0 ... 9	0 ... 9
10	A
11	B
12	C
13	D
14	E
15	F

INSTRUCTION CODES

In addition to describing direction and length, shapes use codes to provide instructions. Code numbers can be in decimal (*dec*) or hexadecimal (*hex*). Hex codes always have three digits, the first being a 0 (zero). Notice that some codes rely on additional codes following. And, note that shapes are limited to lines, arcs, and spaces.

Hex	Dec	Description
000	0	End of shape definition.
<i>Basic Draw and Move</i>		
001	1	Begin draw mode (pen down).
002	2	End draw mode (pen up).
<i>Scaling</i>		
003	3	Divide vector lengths by next byte.
004	4	Multiply vector lengths by next byte.
<i>Memory</i>		
005	5	Push current location onto stack.
006	6	Pop current location from stack.
<i>Draw Subshape</i>		
007	7	Draw subshape number given by next byte.
<i>Advanced Draw and Move</i>		
008	8	X,y displacement given by next two bytes.
009	9	Multiple x,y displacements; terminated with (0,0) code.
<i>Arcs</i>		
00A	10	Octant arc defined by next two bytes.
00B	11	Fractional arc defined by next five bytes.
00C	12	Arc defined by x,y displacement and bulge.
00D	13	Multiple bulge-specified arcs.
<i>Fonts Only</i>		
00E	14	Process next command only if vertical text code exists.

A *stack* is a specific type of memory called FILO, short for “first in, last out.” When two numbers are stored in the stack memory, the last number stored is the first one out. Think of an elevator, where the first person in is usually the last one out.

End of Shape - 0/000

Code 0 must mark the end of every shape definition. It appears at the end of the last line.

```
00C, (2,0, -127), 0
```

In hex notation, 0 appears as 000.

Draw Mode - 1/001

Code 1 starts drawing mode (“pen” is down). By default, every shape definition starts with draw mode turned on.

2/002: Move Mode -

Code 2 starts move mode (“pen” is up). In the sample below, the pen is raised before moving to a new location.

```
2,8,(-36,-63),1,0
```

Reduced Scale - 3/003

Code 3 specifies the relative size of each vector. Each shape starts off at the height of one of the orthogonal vectors, such as 4. To make the shape smaller, use code 3 followed by a byte specifying the scale factor, 1 through 255. For example, the following code draws the shape half as large:

```
3,2
```

TIP Within a shape definition, the scale factor is cumulative. Using the same scale code twice multiplies the effect. For example, 3,2 followed by another 3,2 makes part of the shape four times smaller.
At the end of the shape definition, return the scale to unity so that other shapes are not affected.

Enlarged Scale - 4/004

To make the shape larger, use code 4 followed by a byte specifying the scale factor, 1 through 255. For example, the following code draws the shape twice as large:

```
4,2
```

Note that you can use the 3 and 4 codes within a shape definition to make parts of the shape larger and smaller.

Save (Push) - 5/005

Code 5 saves (*pushes*) the current *x,y*-coordinates to the stack memory. You then use code 6 to recall (*pop*) the coordinates for later use. The stack memory is limited to four coordinates. By the end of the shape definition, you must recall all coordinates that you saved; i.e., there must be an equal number of code 5s and 6s, as shown below:

```
2,14,8,(-8,-25),14,5,8,(6,24),1,01A,016,012,01E,02C,02B,01A,2,  
8,(8,5),1,01A,016,012,01E,02C,02B,01A,2,8,(4,-19),14,6,  
14,8,(8,-9),0
```

Recall (Pop) - 6/006

Code 6 recalls (*pops*) the most-recently saved coordinates from the stack memory.

Subshape - 7/007

Code 7 calls a subshape, which is simply another shape. Shapes can be used within other shapes, which helps reduce the tedium of coding shapes. Code 7 is followed by reference to another shape number, between 1 to 255. (Recall that all shapes within a *.shp* file are identified by number.) For example:

```
7,2
```

calls shape 2 as a subshape.

X,y Distance - 8/008

Codes 8 and 9 overcome the restriction that the vector codes (just 16 directions) place on drawing. Code 8 defines a distance using two bytes that range from -128 to 127:

```
8,xDistance,yDistance
```

The example below shows code 8 being used often:

```
2,14,3,2,14,8,(-21,-50),14,4,2,14,5,8,(11,25),1,8,(-7,-32),2,
8,(13,32),1,8,(-7,-32),2,8,(-6,19),1,0E0,2,8,(-15,-6),1,0E0,2,
8,(4,-6),14,6,14,3,2,14,8,(21,-32),14,4,2,0
```

In the first line of code above, 8,(-21,-50) draws 21 units left (-x), and 50 units down (-y).

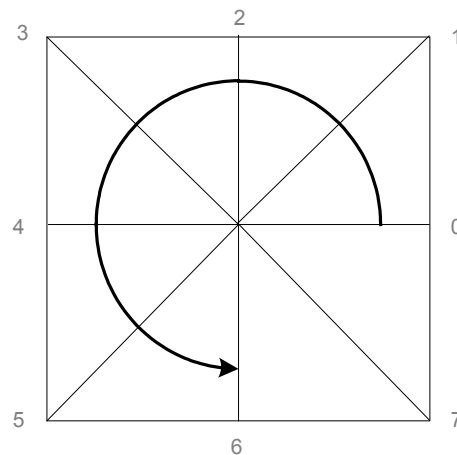
X,y Distances - 9/009

Whereas code 8 specifies a single coordinate, code 9 specifies a series of coordinates, terminated by (0,0). For example:

```
9,(1,2),(-3,4),(5,-6),(0,0)
```

Octant Arc - 10/00A

Code 10 defines an *octant* arc, which is an arc whose angle is limited to multiples of 45 degrees, as shown in the following figure. The arc always starts at position 0, and then moves counterclockwise.



Defining the length of an arc through octants

The arc is specified by the following bytes:

```
10,radius,- 0 startingOctant octantSpan
```

- **10** specifies an octant arc.
- *radius* is a value between 1 and 255.
- Negative sign changes the direction of the arc to clockwise; leave it out for counterclockwise direction.
- **0** specifies the following characters are hexadecimal.
- *startingOctant* specifies where the arc starts; the value ranges between 0 and 7).
- *octantSpan* specifies how hard the arc travels, again a number between 0 through 7.

TIPS When *octantSpan* is **0**, the shape draws a circle.

The octant arc code usually uses parentheses to make itself clearer, such as:
10,(25,-040)

Fractional Arc - 11/ 00B

Code 11 is more useful because it draws arcs that don't end and start at octant angles. Its specification requires, however, five bytes:

```
11,startOffset,endOffset,highRadius,radius,- 0 startingOctant octantSpan
```

- ▶ **11** defines the fractional arc.
- ▶ *startOffset* specifies how far (in degrees) from the octant angle the arc begins.
- ▶ *endOffset* specifies how far from an octant angle the arc ends.
- ▶ *highRadius* specifies a radius larger than 255 units; when the arc has a radius of 255 units or smaller, then this parameter is 0. The *highRadius* is multiplied by 256, then added to the *radius* value to find the radius of the arc.
- ▶ *radius* is a value between 1 and 255.
- ▶ Negative sign changes the direction of the arc to clockwise; leave it out for counterclockwise direction.
- ▶ **0** specifies the following characters are hexadecimal.
- ▶ *startingOctant* specifies where the arc starts; the value ranges between 0 and 7.
- ▶ *octantSpan* specifies how far the arc travels, again a number between 0 through 7.

TIP Here is how Autodesk suggests determining the value of *startOffset* and *endOffset*:

1. Determine the offsets by calculating the difference in degrees between the starting octant's boundary (which is always a multiple of 45 degrees) and the start of the arc.
2. Multiply the difference by 256.
3. Divide the result by 45.

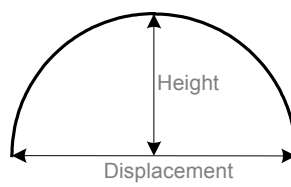
Bulge Arc - 12/00C

Code 12 draws a single-segment arc by applying a bulge factor to the displacement vector.

```
0C,xDisplacement,yDisplacement,bulge
```

- ▶ *xDisplacement* and *yDisplacement* specify the starting x,y-coordinates of the arc.
- ▶ *bulge* specifies the curvature of the arc. All three values range from -127 to 127.

This is how Autodesk says the bulge is calculated: "If the line segment specified by the displacement has length D, and the perpendicular distance from the midpoint of that segment has height H, the magnitude of the bulge is $((2 * H / D) * 127)$."



Calculating the size of a bulge

A semicircle (180 degrees) would have a bulge value of 127 (drawn counterclockwise) or -127 (drawn clockwise), while a line has a value of 0. For an arc of greater than 180 degrees, use two arcs in a row.

Polyarc - 13/00D

Code 13 draws a *polyarc*, an arc with two or more parts. It is terminated by (0,0).

```
13,(0,2,127),(0,2,-127),(0,0)
```

TIP To draw a straight line between two arcs, it is more efficient to use a zero-bulge arc, than to switch between arcs and lines.

Flag Vertical Text Flag - 14/00E

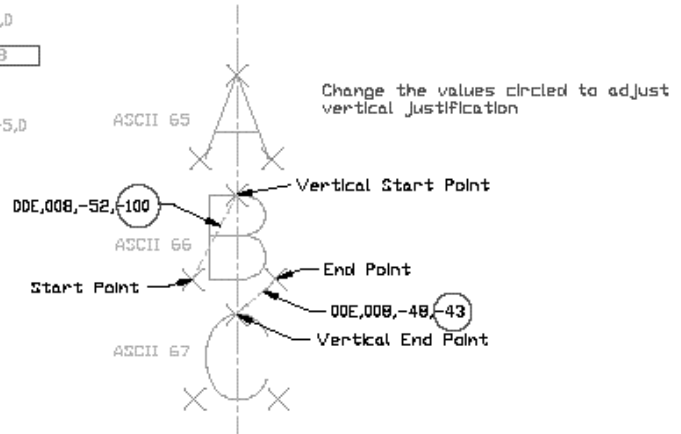
Code 14 is for fonts only, and only fonts that are designed to be placed horizontally and vertically. When the orientation is vertical, the code following is processed; if horizontal, the code is skipped.

R O M A N S

```
*65,38,65
002,00E,00B,-43,-100,008,67,33,001,00D,-48,0,0,0,0,002,00B
62,-33,001,00D,-38,100,0,-38,-100,0,0,0,002,008,61,0,00E,00B
-43,-43,00D

*66,75,66
002,00E,00B,-52,-100,008,19,52,001,00D,43,0,0,14,5,0,5,5,0,5,9,0
0,10,0,-5,9,0,-5,5,0,-14,5,0,-43,0,0,0,-100,0,43,0,0,14,5,0,5,5
0,5,9,0,0,14,0,-5,10,0,-5,5,0,-14,4,0,0,002,008,38,-52,00E,00B
-48,-43,00D

*67,72,67
002,00E,00B,-50,-100,008,86,24,001,00D,-5,-10,0,-10,-9,0,-9,-5,0
-19,0,0,-10,5,0,-9,9,0,-5,10,0,-5,14,0,0,24,0,5,14,0,5,10,0,9,9
0,10,5,0,19,0,0,9,-5,0,10,-9,0,5,-10,0,0,0,002,008,14,-76,00E
008,-50,-43,00D
```



Coding with Field Text

Fields are a special form of text that update automatically. Fields look like text with a gray background, and show values provided by BricsCAD or the operating system, such as the diameter of a circle or the date and time. To show new values, the text can updated manually or automatically.

In this chapter, you learn how to place fields in mtext, regular text, and in attributes, as well as how to customise the look of fields.

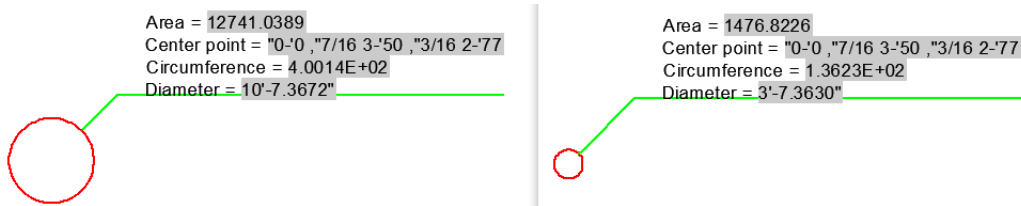
CHAPTER SUMMARY

The following topics are covered by this chapter:

- Placing field text with Field, MText, AttDef, and other commands
- Changing field text
- Exhaustive references of all field codes

For instance, the circle illustrated below at left has its area, center point, and so on described by regular text (white background) and by field text with the gray background. The gray background is only displayed and not plotted, and can be turned off with the **FieldDisplay** variable.

When the size of the circle is reduced and the field text updated, new values show up for the ones that changed — the area, circumference, and radius. See figure at right.



*Left: Circle with field text in gray, and regular text.
Right: Changed circle with updated field text.*

FIELD COMMANDS & VARIABLES

COMMANDS

Field — displays the Field dialog box for constructing field text; also accessed through text commands such as MText and AttDef

UpdateField — forces an update of field values, should they have changed

VARIABLES

DbMod (read-only) — reports if the drawing has been modified by changes to fields

FieldDisplay — toggles the gray background to field text

FieldEval — specifies when fields should be updated; default = 31 (all turned on):

- 0 Not updated automatically
- 1 Updated when the drawing is opened
- 2 Updated when the drawing is saved
- 4 Updated when the drawing is plotted
- 8 Updated when the eTransmit command is used
- 16 Updated when the drawing is regeneration

NOTE

The FieldEval variable does not update the Date field; it is updated only by the UpdateField command.

Placing Field Text

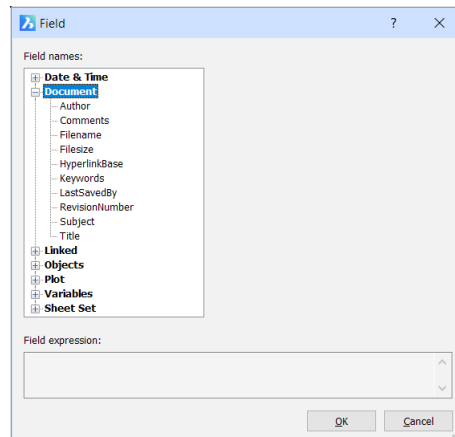
Field text is added to drawings through the **Field**, **Text**, **MText**, **AttDef**, and **Table** commands, and can be placed in dimensions and leaders (as mtext). Let's take a look at each one.

FIELD COMMAND

The **Field** command places field text in drawings in a manner similar to the Text command, placing single lines of text. It operates identically in Linux, Mac, and Windows. For this tutorial, the drawing's creation date is inserted as a field.

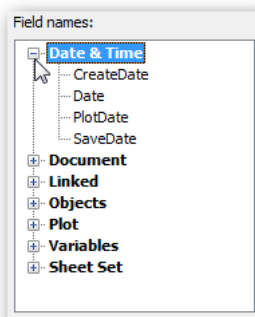
1. Enter the **Field** command.
: field

Notice that BricsCAD displays the Field dialog box.



Field dialog box

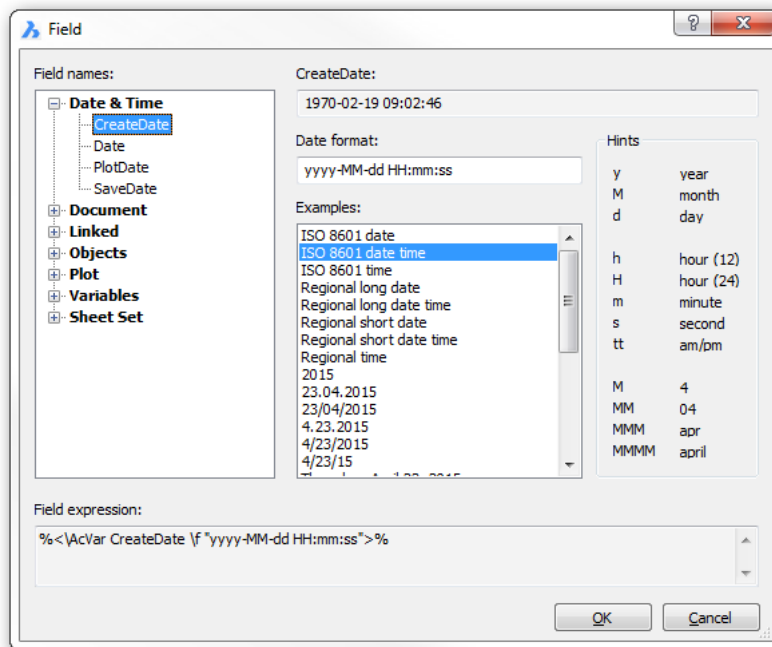
2. Select a field that you want by selecting a group (such as Date & Time or Document), choosing a field name from the group (like CreateDate or Author), and then applying formatting, if available.
For this tutorial, the task is to insert the creation date as a field: CreateDate is in the Date & Time group. Follow these steps:
 - a. Under Field Names, open the **Date & Time** node by clicking the + button.



Fields available for Date & Time

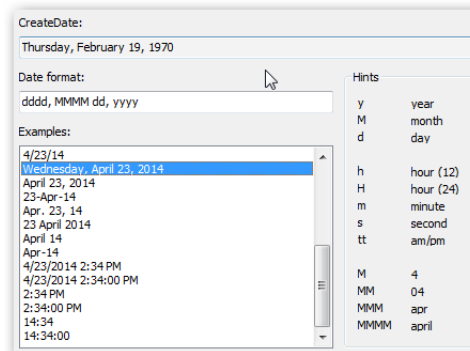
- b. Notice the fields that are available for specifying dates. Choose **CreateDate**.

- c. Notice that the empty part of the dialog box suddenly fills up with all kinds of options related to formatting dates and times.



Formatting options for the CreateDate field

You can format the date field by selecting a format from the **Examples** column, or else construct your own format. For this tutorial, scroll down and then choose the one that looks like “Thursday, April 23, 2014.” (The exact date displayed will differ.)



Selecting a format for the date

Notice that the **Date Format** area shows the date’s format code: dddd, MMMM dd, yyyy. This is where you can edit the formatting code, something that I describe later.

3. Click **OK**. In the command panel, notice that BricsCAD prompts you with a set of Text-like options. (Incorrectly, it shows “MTEXT”; it should say “TEXT.”)
MTEXT Current text style: "Standard" Text height: 2.5
4. Pick a point in the drawing to place the field text:
Specify start point or [Height/Justify]: (Enter an option, or else pick a point)

BricsCAD places the field in the drawing with today's date, using the current text style. (The date you see will differ from the one shown below.)

Friday, February 15, 2013

Date generated by field code

If, however, the drawing is a new one and has not been saved, then all you'll see are four dashes, like this:



Field code that lacks meaning

This is the way that BricsCAD tells you a field lacks a valid value. To give the drawing a creation date, use the **Save** command.

TIP When a field displays four hash marks, like `####`, it means the field value is invalid for some reason.

FIELDS IN MTEXT

If you want to embed field text with regular text, then use the MText or AttDef commands; the Text command and dimensioning commands can't do this. *Embedding* lets you mix regular text and field text in useful ways, such as the combination of "Date: " with a dynamic date.

Date: May 14

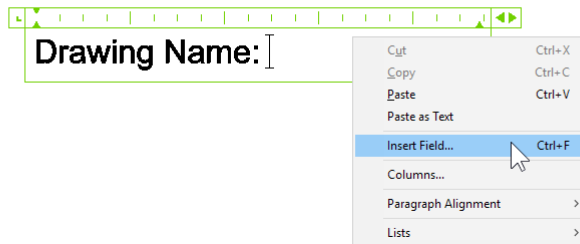
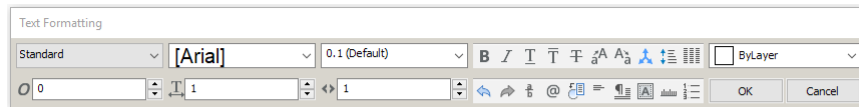
Combining text with field code

In this tutorial, you place a field that reports the file name of the drawing.

1. Start the **MText** command, and then answer its prompts:
: **mtext**
Multiline Text: First corner for block of text: (*Pick a point*)
Select Opposite corner for block of text or [Justification/Rotation angle/text Style/text Height/Direction/Width]: (*Pick another point*)
2. Type some text, such as "Drawing Name:".
3. To insert the field text, press **Ctrl+F**.

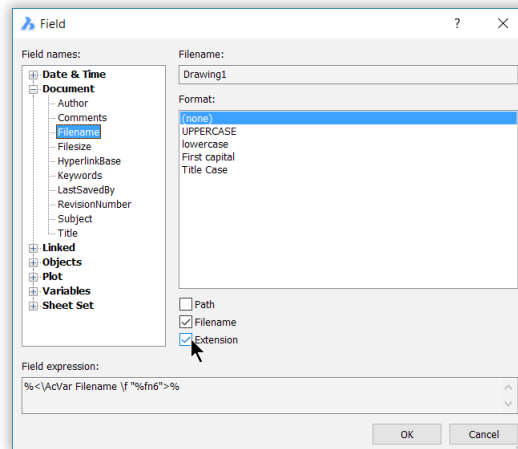
Alternatively, in the Text Formatting toolbar, click the  **Field** button.

Or else right-click the Text Formatting toolbar, and then from the shortcut menu select **Insert Field**.



Accessing fields in the MText command

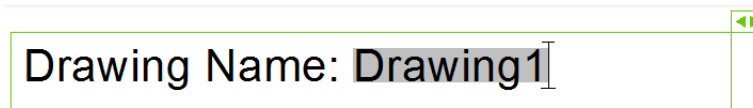
4. Notice that the Field dialog box opens. To choose the field for displaying the name of the drawing file, follow these steps:



Choosing formatting options for the file name

- a. In the Field Names list, open the **Document** node.
 - b. Under Document, choose **Filename**.
 - c. In the Format area, set these options:

Format	(none)
Path	No
Filename	Yes
File extension	Yes
5. Click **OK**. Notice that BricsCAD adds the file name field. It may appear as “Drawing1,” or whatever the file name of the your drawing is. You can tell that it is field text, because of the gray background.



Field text with filename

6. Exit the mtext editor by clicking **OK**.
7. To see field text in action, now save the drawing to change its name:
 - a. Enter the **SaveAs** command.
 - b. In the Save Drawing As dialog box, enter a file name like “Field text example.”
 - c. Click **Save**.

Notice that the field text changes to reflect the new file name.

Drawing Name: Field text example.dwg

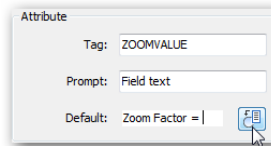
Field text with filename and extension


FIELDS IN ATTRIBUTES

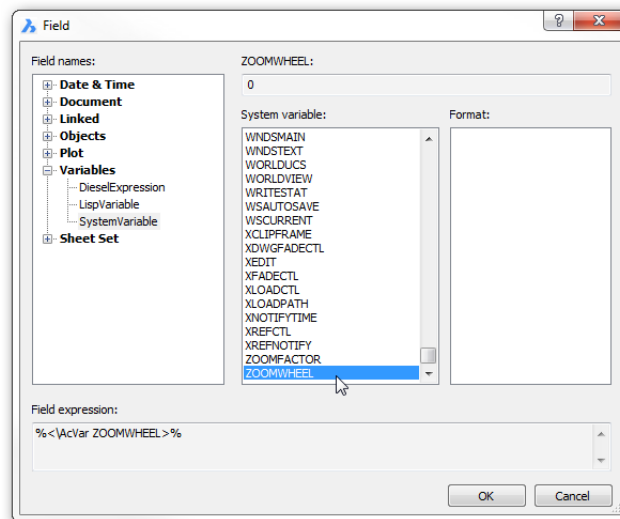
Field text can be made part of an attribute definition. Recall that attributes are used to add custom data to blocks. As shown by the following tutorial, this is done with the **AttDef** command, which is normally used to define attributes.

In this tutorial, you create an attribute that reports the current zoom level of the drawing.

1. Start the **AttDef** command. Notice the Attribute Definition dialog box.
2. In the Attribute section, fill in the attribute text fields — Tag, Prompt, and Default fields. YT; you can use the figure below as an example of the text to use:

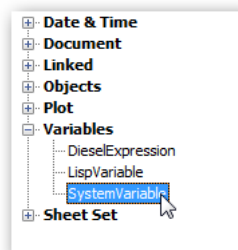


3. To add a field to the attribute, click the  **Insert Field** button next to the Default field. Notice that BricsCAD displays the usual Field dialog box, as shown below.



Field dialog box

4. Select a system variable like this:
 - a. From the Fields Name list, open the **Variables** node.

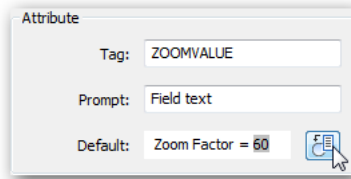


Choosing the SystemVariable field name

- b. Select **SystemVariable**.
- c. Under the list of System Variables, scroll right to the end and then pick **ZoomFactor**.
- d. It has no format options, so click **OK**.

- Click **OK** to close the Field dialog box.

Back in the Attribute Definition dialog box, notice that the field text shows in the Default box with a gray background.



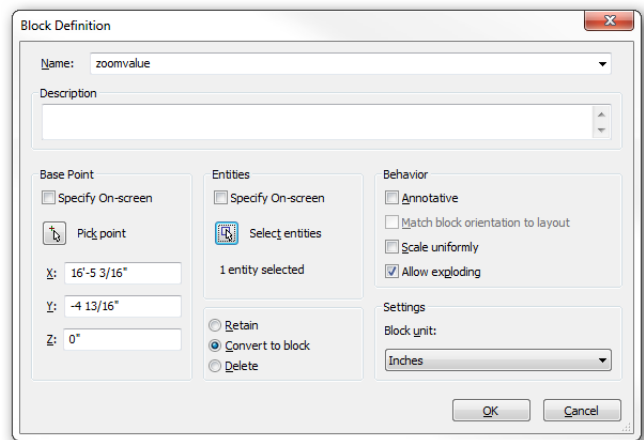
Entering field code as an attribute definition

- Click **OK** to close the dialog box. All you see in the drawing is ZOOMVALUE.

ZOOMVALUE

No field text yet!

- To see the field text, you need to turn the attribute into a block and then insert the block. Follow these steps:
 - Start the Block command.

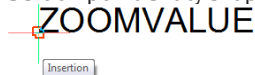


Placing a block as field text

- Enter the following parameters:

Name zoomvalue

Base Point Click **Pick Point**, and then pick the lower right corner to ZOOMVALUE; use the **INS**ertion point entity snap to assist you



Using Insertion entity snap to locate the insertion point of the text

Entities Click **Select Entities**, and then choose the text

Convert to Block Yes

TIP Users should not be allowed to modify fields, so turn on the **Constant** setting found in the Mode section in the AttDef dialog box.

- Click **OK**.
- When the Edit Attributes dialog box appears, click **Cancel**; you don't need its services.

8. Zoom in and out. The zoom factor value does not change. To update the field takes two steps this time.
 - a. First, change the value of the zoom factor using the related variable:
 : zoomfactor
 New current value for ZOOMFACTOR (3 to 100) <60>: 5
 - b. The field text still does not change (I'll explain later why this is), so enter the **Regen** command to see the value of the field updated.
 : regen

Zoom Factor = 5

Field text visible after Regen command

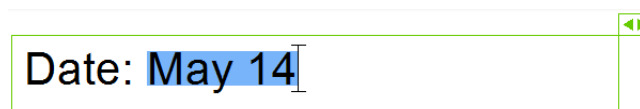
Changing Field Text

To change field text, simply double-click it; BricsCAD displays the Field dialog box. Use it to make changes. Alternatively, you can change field values by using the Properties palette. Here, we look at both approaches.

DOUBLE-CLICKING FIELDS IN MTEXT

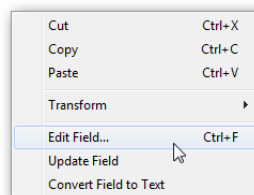
To edit fields placed by the MText command, you use this procedure.

1. Double-click the field text. Notice that BricsCAD displays the mtext editor.
2. Click the field text. Notice that its background color changes to blue.



Field text in mtext

3. Now double-click the blue, and notice that the Field dialog box appears. (Alternatively, you can right-click field text to access the following shortcut menu.)



Accessing field text editor

Here's what the the three field-related options mean on the shortcut menu:

Edit Field — displays the Field dialog box.

Update Field — forces an update the field's value.

Convert Field to Text — turns the field into normal text, freezing the value of the field.

4. Select a different field type, or change the field formatting.
5. Click **OK** to close the dialog box, and then click **OK** to exit the mtext editor.

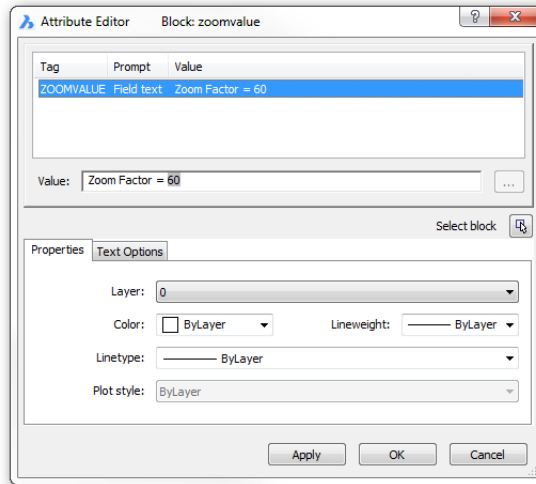
So the background to field text changes color, depending on its editing state:

- May** Gray = field text in unedited state
- May** Blue = field text ready for editing

EDITING FIELDS IN ATTRIBUTE DEFINITIONS

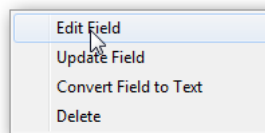
When field text is in an attribute *definition*, you can edit it, as follows:

1. Double-click the attribute text. BricsCAD displays the Attribute Editor dialog box. (The EAttEdit command was executed.)



Field text in attribute editor

2. In the dialog box's **Default** field, double-click the gray field text. Alternatively, you can right-click the field text itself to access this shortcut menu:



Accessing field text editor

3. Notice the Field dialog box. Make your changes, and then click **OK** to exit the dialog boxes.

Controlling the Way Fields Update

The point to using field text is that it can update values — manually or automatically. You force an update with the **UpdateField** command, or else specify when automatic updates take place with the **FieldEval** system variable.

UPDATEFIELD COMMAND

To update selected fields manually, use the **UpdateField** command. It asks you to select one or more fields, and then update their values.

```
: updatefield
Select field(s) to update: (Press Ctrl+A, or select individual fields)
Select field(s) to update: (Press Enter to end object selection)
<n field(s) found, n field(s) updated>
```

To update all fields in the drawing, press **Ctrl+A** at the ‘Select fields’ prompt.

FIELDEVAL COMMAND

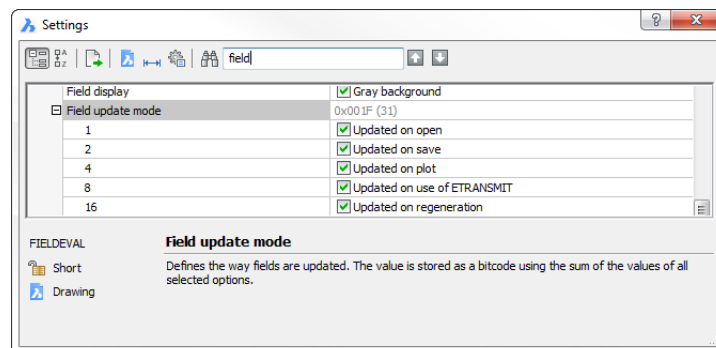
Earlier, you used the Regen command to force the value of a field to update. This was an application of an automatic update that was dictated by the **FieldEval** sysvar. It wasn’t the Regen command itself that did the updating; rather the command triggered BricsCAD to also update all fields in the drawing.

Fields are updated by BricsCAD when any of the following trigger events occur:

- Open** — when the drawing file is opened
- Save** — after the drawing file is saved, as you saw above with the CreateDate field
- Plot** — before the drawing is plotted
- eTransmit** — as the drawing is being prepared for packaging with the eTransmit command
- Regen** — when the drawing is regenerated (reloaded from the .dwg file)

Note that the settings in **FieldEval** variable do not update Date fields; dates can be updated only with the **UpdateField** command.

These events are controlled with the **FieldEval** system variable, which is best accessed through the Settings dialog box:



Settings for field text

If you choose the access the system variable at the command line, it looks like this:

```
: fieldval
New current value for FIELDEVAL (0 to 31) <31>: (Enter a number)
```

The value takes one or more of these values:

FieldEval	Comments
0	Fields are not updated automatically; use the UpdateField command
1	Open command
2	Save command
4	Plot command
8	eTransmit command
16	Regen command

The default is 31, the sum of 1+2+4+8+16 — all options are turned on, except for 0.

FIELDDISPLAY COMMAND

The **FieldDisplay** system variable determines whether field text displays that gray rectangular background or not:

```
: fielddisplay
New current value for FIELDDISPLAY [1 for on (ON)/0 for off (OF): (Enter OFF or On)
```

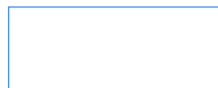
I say, leave it on all the time, because (a) its purpose is to lets you distinguish between field and regular text, and (b) the gray background is never plotted, anyhow.

FieldDisplay	Comments
0	Fields don't display the gray rectangular background
1	They do

Another Field Text Example


In the following tutorial, you get field text to report the area of a rectangle.

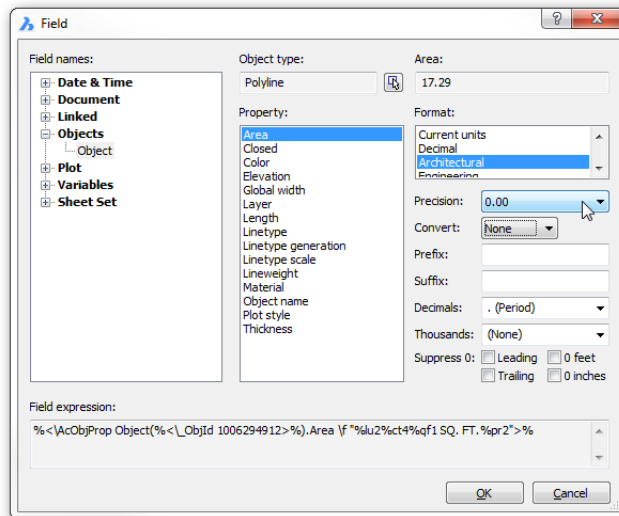
1. Start BricsCAD, and then use the **Rectang** command to draw a rectangle of any size.



A rectangle

2. Use the **Field** command to create the field code for the area of the rectangle. Select the following options:

Field Option	Value
Field Category	Objects
Field Names	Object
Object Type	(Click  Select Object button, and then select the rectangle.) Polyline
Property	Area
Format	Architectural
Precision	0.00

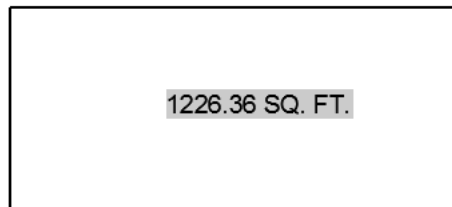


Field dialog box

TIP The field text is linked automatically to the rectangle through the **Select Object** button.

3. Click **OK** to exit the dialog box.
4. BricsCAD switches to mtext-like prompts:

MTEXT Current style: "Standard" Text height: 2.5
Specify start point or [Height/Justify]: (Pick a point inside the rectangle)



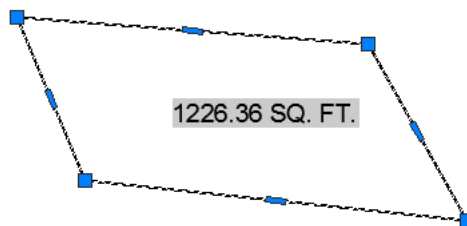
Field text inside rectangle

The field text is placed in the drawing. Notice that the units are shown as "SQ. FT." — square feet.

Updating the Field Text

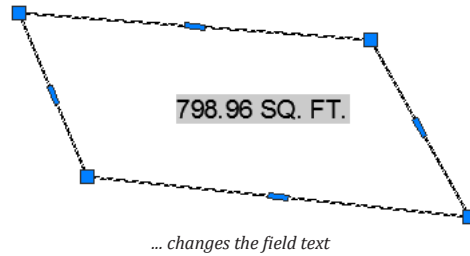
With the field text reporting the area of the rectangle, you can go ahead and change the size of the rectangle to see how the field updates.

1. Select the rectangle, and then use the grips to change the size of the rectangle.



Changing the rectangle...

2. Enter the **Regen** command to update the field text.



Notice that the field text changes to reflect the new area.

TIP It is important to remember that field text is tied to specific objects. If you erase the rectangle and then use the **UpdateField** command, the field text will read ##### because it no longer has a valid meaning, because its related object is gone.

COMPATIBILITY WITH AUTOCAD FIELD CODES

In general, field codes generated by BricsCAD are compatible with those from AutoCAD. The primary difference is that BricsCAD tends to have fewer entities and fewer codes for some entities. In addition, when you paste literal field codes into a drawing, BricsCAD interprets them as a script; in AutoCAD, they are pasted as field text.

When you open a drawing from AutoCAD in BricsCAD containing unsupported field codes (such as for mlines), BricsCAD displays them correctly as field text, but cannot edit them.

Understanding Field Codes

Field text uses a coding system that is not documented by neither Autodesk nor Bricsys. A typical field code looks like this:

```
%<\AcVar Filename \f "%tc4%fn7">%
```

(Parts of the code that never change are shown in blue.) Generally there are two pairs in a code, one set before the `\f`, and another after:

- ▶ Before the `\f` is the *type* of field
- ▶ After the `\f` is the *format* of the field

I figure that `\f` is short for “format.” In the drawing, the code listed above gives the file name in a field, like Drawing1.dwg.

Let’s parse the field code to see what it means:

<code>%<</code>	Start of field code
<code>\AcVar</code>	AutoCAD variable
Filename	Name of the variable
<code>\f "</code>	Start of format code(s)
<code>%tc4</code>	Text capitalization using format #4
<code>%fn7</code>	File name using format #7
<code>"</code>	End of format codes
<code>>%</code>	End of field code.

`%<` Signals the start of a field code, just as ‘(’ tells BricsCAD that LISP code is starting, and ‘\$(’ indicates the start of Diesel code.

`\` Backslash Indicates that a metaword follows. In this case, `\AcVar` refers to “AutoCAD variable,” and that the following word will be the name of a variable — **Filename**, in this case.

`\f` Specifies that one or more format codes are to follow.

`" "` Quotation marks *delimit* format codes; format codes are always held within the quotation marks.

`%` Percent Indicates the start of a format code.

- The first code, `%tc`, specifies the **text** capitalization. The value of **4** means that the text of the file name is shown in *title case*, meaning the first letter of each word is capitalized.
- The second format code, `%fn7`, specifies how much of the **file** name is displayed; a value of **7** means that the path, file name, and file extension are all displayed.

`>%` Signals the end of the field code.

Here is another example of a field code. This one shows the properties of an object, a circle:

```
%<\AcObjProp Object(%<\_ObjId 2126544536>%).Center \f "%lu2%pt3">%
```

let’s pick it apart

<code>%<</code>	Start of field code
<code>\AcObjProp</code>	Metatext for AutoCAD object properties
Object(%<_ObjId 2126544536>%)	Object identified by number
.	(Dot) Start of property
Center	Property: center coordinates of circle
<code>\f</code>	Metatext for format code
<code>"</code>	Start of format code
<code>%lu2</code>	Linear units style #2, decimal
<code>%pt3</code>	Points style #3, x,y coordinates
<code>"</code>	End of format code
<code>>%</code>	End of field code

Complete Field Code Reference

GROUPS

Fields belong to groups. All of them are found in one of the following group names:

Field Group	Group Name
Objects and named objects	AcObjProp <i>objectId</i>
System variables	AcVar <i>varName</i>
Diesel code	AcDiesel <i>code</i>

METAWORDS

Fields use meta-words to allow additional information, such as a hyperlink or units. Meta-words are identified by the backslash (\) prefix. The backslash is followed by text that is surrounded by quotation marks ("). Here are a few examples:

Meta-word Meaning	MetaWord
Hyperlink address follows \	\href " <i>hyperlinkReference</i> "
Formatting codes follow \	\f " <i>formatCodes</i> "
Inches units follows \	\"

FORMATTING

The text displayed by fields is formatted using the following format codes:

Formatting	Format Code
Decimal (.) places	%.
Angular Units	%au
Bytes (filesize)	%by
Convert	%ct
Decimal Separator	%ds
File Name, path, and extension	%fn
Linear Units	%lu
Line Weight units	%lw
Precision	%pr
Points (x,y,z)	%pt
Scale Factor	%qf
Text Case	%tc
Hexadecimal conversion	%X

Some notes on format codes:

%X forces numbers to be displayed in hexadecimal notation (base 16)

%ld is a code used by file sizes; I haven't figured out, but it seems to have no effect

%qf is used by scale factors, but employs values I haven't figured out yet

Some codes use the same naming system as related variables. For example, **%lu** (linear units) uses the same values as the **LUnits** system variable, such as 1 = scientific units and 2 = decimal units.

COMPLETE FORMAT CODE REFERENCE

Here is a summary of all of the format codes employed by fields.

%tcn — Text Case

Specifies how to display the case of text.

Meaning	Code
No formatting	<i>bLank</i>
UPPERCASE	%tc1
lowercase	%tc2
First capital	%tc3
Title Case	%tc4

%lun — Linear Units

Specifies how to display linear units. Values match those of the LUnits system variable. Decimal units can use decimal separators; see %ds below.

Meaning	Code
No formatting	<i>bLank</i>
Scientific	%lu1
Decimal *	%lu2
Engineering units	%lu3
Architectural units	%lu4
Fractional units	%lu5
Current Units	%lu6

%dsn — Decimal Separator

Decimal separators specify the character that separates thousands in decimal units (%lu2). BricsCAD uses standard ASCII codes between 31 and 127 for specifying decimal separators. For the meaning of ASCII codes, consult an ASCII table such as at <http://www.asciitable.com>. These are just a few examples.

Meaning	Code	Notes
Space separator	%ds32	
Comma (,) separator	%ds44	Used in North America
Decimal (.) separator	%ds46	Used in European countries
Angle (<) separator	%ds60	
Letter A separator	%ds65	

%aun — Angular Units

Specifies how angular units are displayed, and matches the values used by the AUnits variable.

Meaning	Code
No formatting	<i>bLank</i>
Decimal degrees	%au0
Deg/min/sec	%au1
Grads	%au2
Radians	%au3
Surveyor's Units	%au4
Current units	%au5

%lwn — Line Weight units

Specifies the units by which to display lineweights; similar to LwUnits system variable.

Meaning	Code
Millimeters	%lw1
Inches	%lw2

%qfn — scale Factor

Specifies scale factor for plot and viewport scales.

Meaning	Code
Viewport custom scale	%qf1
Plot scale	%qf2816

%ctn — ConverT

Specifies how plot scales and areas are displayed:

Plot and Viewport Scales	Code	Notes
No scale	<i>blank</i>	
#:1	%ct0	#
1:#	%ct1	1/#
#"=1'0"	%ct2	12*#
Area Scales	Code	Notes
Square feet	%ct3	#
Square inches	%ct4	12*#

%ptn — PointTs (xyz coordinates)

Specifies which coordinates to display; default displays all three (x, y, and z).

Meaning	Code
X, Y and Z	<i>none</i>
X only	%pt1
Y only	%pt2
Z only	%pt4
X and Y	%pt3
X and Z	%pt5
Y and Z	%pt6

%.n — decimal places

Specifies number of decimal places displayed by real numbers:

Meaning	Code
8	%.8
7	%.7
6	%.6
5	%.5
4	%.4
3	%.3
2	%.2
1	%.1
0	%.0

%prn — *display Precision*

Specifies fractional precision or number of decimal places displayed, in a manner similar to the LuPrec system variable. Note that under some conditions, %pr7 and %pr8 display at most 1/64 precision.

Fractions	Decimal Places	Code
1/256	8	%pr8 *
1/128	7	%pr7 *
1/64	6	%pr6
1/32	5	%pr5
1/16	4	%pr4
1/8	3	%pr3
1/4	2	%pr2
1/2	1	%pr1
1	0	%pr0

%FNN — *FILE NAMES*

Specifies how much of the file name to display.

Meaning	Code
No file name, path only	%fn1
File name only, without extension	%fn2
File name and path, without extension	%fn3
File name with extension	%fn6
File name with extension and path	%fn7

%BYN — *BYTES (FILE SIZE)*

Specifies the format in which to display file sizes.

Meaning	Code
Bytes	%by1
Kilobytes	%by2
Megabytes	%by3

HREF - HYPERLINKS

Specifies the format of hyperlinks.

Meaning	Code
\href	Indicates that a hyperlink address follows
#,	(Optional) Target
#	(Optional) Text to display
#0	Indicates end of hyperlink address

QUICK SUMMARY OF FIELD DATE AND TIME CODES

The date and time are formatted by the codes listed below; this list is more complete than the one provided by Bricsys.

Letters not used for codes are treated literally, such as c and Q. You can use characters as separators, such as / - and ,. The number of characters sometimes affects the date and time displayed: one or two “d”s display the date of the month, while three or four display the day of the week. Some codes are case-sensitive: uppercase M means month, while lowercase m means minute. “System Time” means the date and time as formatted specified by Windows.

Format	Comment	Example
Months (<i>must use uppercase M</i>)		
M	Number of month.	3 (March)
MM	Number with zero prefix.	03
MMM	Three-letter abbreviation.	Mar
MMMM	Full month name.	March
Dates		
d	Date of the month.	6
dd	Date, with zero prefix.	06
Days		
ddd	Abbreviated day of the week.	Fri
dddd	Full day name.	Friday
Years (<i>must use lowercase y</i>)		
y	Single digit year.	6 (2006)
yy	Two-digit year.	06
yyy or yyyy	Four-digit year.	2006
Hours		
h	12-hour clock.	5
hh	Hour with zero prefix.	05
t	Single-character AM or PM.	A
tt	Placeholder for AM or PM.	AM
H	24-hour clock.	17
HH	24-hour with zero prefix.	07
Minutes (<i>must use lowercase m</i>)		
m	Minutes.	9
mm	Minutes with zero prefix.	09
Seconds		
s	Seconds.	8
ss	Seconds with zero prefix.	08
Examples of System Time (<i>case sensitive</i>)		
%C	Date and time in short format.	6/21/05 4:18:06 PM
%#C	Date and time.	Friday, June 21, 2005 4:18:06 PM
%X	Time.	4:18:06 PM
%x	Date in short format.	6/21/05
%#x	Date in long format.	Friday, June 21, 2005

DATE & TIME FORMAT CODES

Format	Field Codes	Examples
<i>Year, Month, Day, Hour, Minute Seconds, and AM/PM</i>		
Month	M	8 (also 10, 11, 12)
	MM	08
	MMM	Aug
	MMMM	August
Day	d	3 (also 31)
	dd	03 (also 31)
	dddd	Sunday
Year	yy	04
	yyyy	2004
Hour	h	4 (also 12)
	hh	04 (also 12)
Minute	m	5 (also 59)
	mm	05 (also 59)
Second	s	2 (also 59)
	ss	02 (also 59)
am or pm	tt	AM, PM (leave out for 24-hour clock)
Regional long date	%#x	Saturday, July 31, 2004
Regional long date and time	%#c	Saturday, July 31, 2004 7:45:19 PM
Regional short date	%x	7/31/2004
Regional date and time	%c	7/31/2004 7:45:19 PM
Regional time	%X	7:45:19 PM

Format	Field Codes	Examples
<i>Alternative Day, Date, Month, Year, time, and AM/PM</i>		
Date	D	1
Date with zero prefix	DD	01
Abbreviated day name	DDD	Sat.
Full day name	DDDD	Saturday
Month	M	3
Month with zero prefix	MO	03
Abbreviated month name	MON	Mar.
Full month name	MONTH	March
Abbreviated year	YY	04
Full year	YYYY	2004
Hour	H	9
Hour with zero prefix	HH	09
Minutes with zero prefix	MM	03
Seconds with zero prefix	SS	08
Millisecond (1/1000 of a sec)	MSEC	257
Displays AM or PM	AM/PM	AM
Displays am or pm	am/pm	am
Displays A or P	A/P	A
Displays a or p	a/p	a

Objects and Property Names

In general, objects employ the following field text coding:

```
%<\AcObjProp Object(%<\_ObjId idNumber>%).property [\f "format"]>%
```

idNumber identifies the object

property describes the object's property; named objects are similar, but are restricted to the **.property** property

format is optional, and formats the property

Here is an example with formatting:

```
%<\AcObjProp Object(%<\_ObjId 2130015880>%).LinetypeScale \f "%tc1">%=
```

And without formatting:

```
%<\AcObjProp Object(%<\_ObjId 2130015880>%).LinetypeScale>%
```

TIP Thanks to www.cadforum.cz for identifying that %X is the code for hex format.

PROPERTIES IN COMMON

Here are the properties common to all entities:

Property Name	Field Code	Formatting
Color	TrueColor	Text
Layer	Layer	Text
Linetype	Linetype	Text
Linetype Scale	LinetypeScale	Linear units
Lineweight	Lineweight	Measurement
Material	Material	Text
Object Name	ObjectName	Text
Plot Style	PlotStyleName	Text
Position	Coordinates	Linear units
Slope	Slope	Angular units
Thickness	Thickness	Linear units
Transparency	EntityTransparency	Text
UCS Elevation	UCSElevation	Text

OBJECT PROPERTIES

Most entities have a few (or many!) properties, but some have no additional properties. Here is a list of entities and field properties unique to them (i.e. excluding the common properties listed above).

Arcs

Property Name	Field Code	Formatting
Arc Length	ArcLength	Linear units
Area	Area	Linear units
Center	Center	Linear units & XYZ
End	EndPoint	Linear units & XYZ
End Angle	EndAngle	Angular units
Normal	Normal	Linear units & XYZ
Radius	Radius	Linear units
Start	StartPoint	Linear units & XYZ
Start Angle	StartAngle	Angular units
Total Angle	TotalAngle	Angular units

Attribute Definition

Single-line text properties, plus these:

Property Name	Field Code	Formatting
Constant	Constant	Text
Invisible	Invisible	Text
Lock Position	LockPosition	Text
Preset	Preset	Text
Prompt	PromptString	Text
Tag	TagString	Text
Verify	Verify	Text

Associative Dimensions

Field Name	Field Code	Formatting
Associative	Associative	Text

Blocks, Block Placeholders, and External References

Property Name	Field Code	Formatting
Block Unit	InsUnits	Units
Name	Name	Text
Position	InsertionPoint	Linear units & XYZ
Prompt ¹	TextString	Text
Rotation	Rotation	Angular units
Scale X	XScaleFactor	Linear units
Scale Y	YScaleFactor	Linear units
Scale Z	ZScaleFactor	Linear units
Unit Factor	Unit Factor	Linear units

¹ Found in blocks with attributes.

Circles

Property Name	Field Code	Formatting
Area	Area	Linear units
Center	Center	Linear units and XYZ
Circumference	Circumference	Linear units
Diameter	Diameter	Linear units
Normal	Normal	Linear units and XYZ
Radius	Radius	Linear units

Ellipses

Property Name	Field Code	Formatting
Area	Area	Linear units
Center	Center	Linear units & XYZ
End	EndPoint	Linear units & XYZ
End Angle	EndAngle	Angular units
End Point	EndPoint	Linear units
Major Axis Vector	MajorAxis	Linear units & XYZ
Major Radius	MajorRadius	Linear units
Minor Axis Vector	MinorAxis	Linear units & XYZ
Minor Radius	MinorRadius	Linear units
Radius Ratio	RadiusRatio	Linear units
Start Point	StartPoint	Linear units & XYZ
Start Angle	StartAngle	Angular units

Hatches

Property Name	Field Code	Formatting
Angle	Angle	Angular units
Area	Area	Linear units
Associative	AssociativeHatch	Text
Double	PatternDouble	Text
Elevation	Elevation	Linear units
Island Detection Style	HatchStyle	Text
Origin Point	Origin	Linear units
Pattern Name	PatternName	Text
Scale	PatternScale	Linear units
Spacing	PatternSpace	Linear units
Type	PatternType	Text

Leaders

No additional properties

Lines

Property Name	Field Code	Formatting
Angle	Angle	Angular units
Delta	Delta	Angular units & XYZ
End Point	EndPoint	Angular units & XYZ
Length	Length	Linear units
Start Point	StartPoint	Angular units & XYZ

Mtext

Property Name	Field Code	Formatting
Contents	TextString	Text
Direction	DrawingDirection	none
Height	Height	Linear units
Line Space Factor	LineSpacingFactor	Linear units
Line Space Style	LineSpacingStyle	Text
Position	InsertionPoint	Linear units & XYZ
Rotation	Rotation	Angular units
Style	StyleName	Text
Width	Width	Linear units

OLE (object linking and embedding) objects

No additional properties

Polylines

Polylines include donuts, rectangles, polygons, revclouds, and certain ellipses.

Property Name	Field Code	Formatting
Area	Area	Area units
Closed	Closed	none
Elevation	Elevation	Linear units
Global Width	ConstantWidth	Linear units
Length	Length	Linear units
Linetype Generation	LinetypeGeneration	none

Polygon Meshes

Property Name	Field Code	Formatting
M Closed	MClose	Text
M Density	MDensity	none
M Vertex Count	MVertexCount	none
N Closed	NClose	Text
N Density	NDensity	none
N Vertex Count	NVertexCount	none

Polyface Meshes

No additional properties

Raster Images

Property Name	Field Code	Formatting
Name	Image name	Text
Position	Insertion Point	Linear units
Rotation	Rotation	Angular units
Width	ImageWidth	Linear units

Regions

Property Name	Field Code	Formatting
Area	Area	Area units
Perimeter	Perimeter	Linear units

Rays and Xlines

Property Name	Field Code	Formatting
Basepoint	BasePoint	Linear units & XYZ
Direction Vector	DirectionVector	Linear units & XYZ
Second Point	SecondPoint	Linear units & XYZ

Shapes

Property Name	Field Code	Formatting
Name	Name	Text
Obliquing	ObliqueAngle	Angular units
Position	InsertionPoint	Linear units & XYZ
Rotation	Rotation	Angular units
Size	Height	Linear units
Width Factor	ScaleFactor	Linear units

Single-line Text

Property Name	Field Code	Formatting
Backward	Backward	Text
Contents	TextString	Text
Height	Height	Linear units
Justify	Alignment	Text
Obliquing	ObliqueAngle	Angular units
Position	InsertionPoint	Linear units & XYZ
Rotation	Rotation	Angular units
Style	StyleName	Text
Text Alignment	TextAlignmentPoint	Linear units & XYZ
Upside Down	UpsideDown	Text
Width Factor	ScaleFactor	Linear units

Splines

Property Name	Field Code	Formatting
Area	Area	Area units
Closed	Closed	none
Control Points	NumberOfControlPoints	none
Degree	Degree	none
End Tangent	EndTangent	Linear units & XYZ
Fit Tolerance	FitTolerance	Linear units
Fit Points	NumberOfFitPoints	none
Planar	Planar	none
Start Tangent	StartTangent	Linear units

Tables

Property Name	Field Code	Formatting
Columns	Columns	none
Direction	FlowDirection	none
Height	Height	Linear units
Horizontal cell margin	HorzCellMargin	Linear units
Position	InsertionPoint	Linear units & XYZ
Rows	Rows	none
Style	StyleName	Text
Width	Width	Linear units

Tolerances

Property Name	Field Code	Formatting
Position	InsertionPoint	Linear units & XYZ
Text Height	Text Height	Linear units

Viewports

Property Name	Field Code	Formatting
Center	Center	Linear units & XYZ
Height	Height	Linear units
Width	Width	Linear units

3D Faces

No additional properties

3D Polylines

Property Name	Field Code	Formatting
Closed	Closed	Text
Fit/Smooth	Type	Text
Length	Length	Linear units

3D Solids

Property Name	Field Code	Formatting
Area	Area	Linear units
Centroid	Centroid	Linear units, XYZ
Gyration Radii	GyrationRadii	Linear units, XYZ
Moments of Inertia	MomentsOfInertia	Linear units, XYZ
Product of Inertia XY	ProductOfInertiaXY	Linear units
Product of Inertia XZ	ProductOfInertiaXZ	Linear units
Product of Inertia YZ	ProductOfInertiaYZ	Linear units
Volume	Volume	Linear units

Sheet Sets

Property Name	Field Code	Formatting
CurrentSheetCategory	Sheet.Category	Text
CurrentSheetCustom	Sheet.	Text
CurrentSheetDescription	Sheet.Description	Text
CurrentSheetIssuePurpose	Sheet.IssuePurpose	Text
CurrentSheetNumber	Sheet.Number	Text
CurrentSheetNumberAndTitle	Sheet.NumberAndTitle	Text
CurrentSheetRevisionDate	Sheet.RevisionDate	Text
CurrentSheetRevisionNumber	Sheet.RevisionNumber	Text
CurrentSheetSet	SheetSet.Name	Text
CurrentSheetSetCustom	SheetSet.	Text
CurrentSheetSetDescription	SheetSet.Description	Text
CurrentSheetProjectMilestone	SheetSet.ProjectMilestone	Text
CurrentSheetProjectName	SheetSet.ProjectName	Text
CurrentSheetProjectNumber	SheetSet.ProjectNumber	Text
CurrentSheetProjectPhase	SheetSet.ProjectPhase	Text
CurrentSheetSetCount	SheetSet.SheetCount	Text
CurrentSheetSubSet	Subset.Name	Text
CurrentSheetSetSubSheetCount	Subset.SheetCount	Text
CurrentSheetTitle	Sheet.Title	Text

NAMED OBJECT PROPERTIES

Named objects are entities that have names or style names: you access them by name. The entities that fall into this category are as follows:

- Layer names
- Linetype names
- View names
- Dimension styles
- Text styles
- Table styles

As of writing, you can only access the names related to each named object.

PART III

Programming BricsCAD

Writing Scripts

BricsCAD's clearest programming possibility is the script. In this chapter, you learn how to write scripts, and how to use its built-in script recording feature.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Understanding scripts
- Learning about drawbacks to scripts
- Employing script commands and modifiers
- Using special characters.
- Recording scripts

What are Scripts?

Scripts mimic what you type at the keyboard. Anything you type at the ':' command prompt can be put in a script file. That includes BricsCAD commands, their options, your responses, and — significantly— LISP code.

Mouse actions, however, cannot be included in script files, such as selecting dialog box and toolbar buttons. Scripts are strictly keyboard-oriented.

The purpose of scripts is to reduce the number of keystrokes you type. By placing the keystrokes and coordinate picks in a file, the file reruns your previously-entered commands. (Think of scripts as a predecessor to macros.)

A script file that draws a line and a circle might look like this:

```
line 1,1 2,2
circle 2,2 1
```

In this script, the Line command starts, and then is given two sets of x,y coordinates, (1,1) and (2,2). The Circle command starts, and is given a center point (2,2) and a radius (1). Hidden are the extra space at the end of each line, which are like pressing the Spacebar to end a command. In this chapter, I show hidden spaces with this character: ■ .

Scripts are stored in files that have the .scr extension. Script files consist of plain ASCII text format. For this reason, do not use a word processor, such as Libre Office. Instead, to write scripts use a text editor, such as Notepad in Windows, Text Edit in Linux, or TextEdit in Mac.

You can use the BricsCAD script creation command **RecScript** (short for “record script”) to record your scripts. Or you can enter the command text directly into an .scr file: when I feel like a DOS *power user*, I'll write the script in the Windows command prompt (press **Windows+R**, and then enter the **Cmd** command):

```
C:\> copy con filename.scr
;This is the script file
line 1,1 2,2
circle 2,2 1
```

When I'm done, I press **Ctrl+Z** to tell the operating system that I've finished editing, and to close the file.

DRAWBACKS TO SCRIPTS

A limitation to scripts is that just one script file can be loaded into BricsCAD at a time. A script file can, however, call another script file. Or, you can use some other customization facility to load additional script files, such as with toolboxes, menu macros, and LISP routines.

Another limitation is that scripts stall when they encounter invalid command syntax. I sometimes have to go through the code-debug cycle a few times to get the script correct.

It is useful to have an BricsCAD reference text on hand that lists all command names and their options.

Strictly Command-Line Oriented

Another limitation is significant in this age of GUIs (graphical user interfaces): scripts cannot control mouse movements nor actions in dialog boxes. This is a reason that nearly all commands that display dialog boxes also have a command-line equivalent. But different commands handle this differently:

- ▶ Some commands have different names. For example, to control layers, there is the **Layer** for the dialog box and **-Layer** for the command line. If the script needs to create or change a layer, use the **-Layer** command, or better yet the **CLayer** system variable, as follows:

```
; Change layer:  
clayer=layername
```

- ▶ Some commands need system variable **FileDia** turned off. This forces commands that display the **Open File** and **Save File** dialog boxes — such as **Open**, **Script**, and **VSlide** — to prompt for filenames at the command line. Thus, script files should include the following lines to turn off file dialog boxes:

```
; Turn off dialog boxes:  
filedia=0
```

```
; Load slide file:  
vslide=filename
```

- ▶ When **FileDia** is turned off, use the ~ (tilde) as a filename prefix to force the display of the dialog box. For example:

```
: script  
Script to run: ~ (BricsCAD displays Run Script dialog box.)
```

- ▶ Some commands have no command-line equivalent, such as the **Plot** command. Instead, when this command is used in a script, the command-line version appears automatically.
- ▶ While BricsCAD accepts command aliases with - (hyphen) prefixes to force the command-line version of commands, it lacks the hyphen-commands found in AutoCAD.

Recording with RecScript

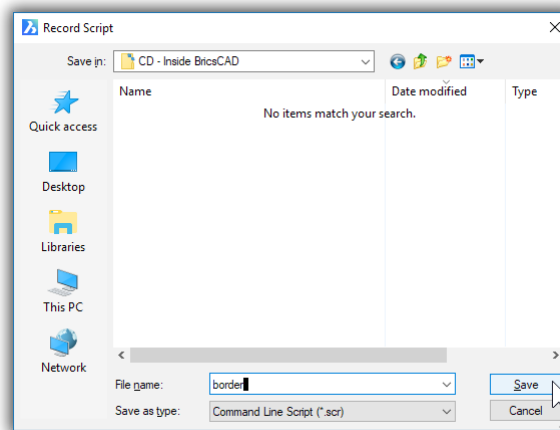
The **RecScript** command records keystrokes, and then saves them to an *.scr* script file.

The **StopScript** command tells BricsCAD to stop recording.

The **Script** command plays back the script.

Let's see how this works. Record a script for drawing a rectangular border sized 24x26 units:

1. In a new drawing, start the **RecScript** command. (Alternatively, from the **Tools** menu select **Record Script**.) Notice the Record Script dialog box.



Starting to record a script by giving it a file name

2. Enter a file name for the script. It can be any name that will remind you of the script's function, and can be up to 255 characters long. For this tutorial, enter **border**, and then click **Save**.
3. Notice that the dialog box goes away, and that BricsCAD appears to be doing nothing. In fact, it is waiting for you to enter commands. Enter the commands and options shown in boldface:
: **rectang**
Chamfer/Elevation/Fillet/Rotated/Square/Thickness/Width/Area/Dimensions/<Select first corner of rectangle>: **0,0**
Other corner of rectangle: **36,24**

: **zoom**
Zoom: In/Out/All/Center/Dynamic/Extents/Left/Previous/Right/Scale/Window/<Scale (nX/nXP)>: **e**
4. When done, enter the StopScript command to signal BricsCAD that you are done:
: **stopscript**
5. Now run the script with the Script command, as follows:
 - a. Start a new drawing with the **New** command, so that you can see the effect of the script.
 - b. Enter the **Script** command.
 - c. Notice the Run Script dialog box. Choose **border.scr**, and then click **Open**.

Notice that the script instantly draws the rectangle, and then zooms the drawing to the extents of the newly-drawn border. Indeed, it may occur so fast that you don't notice it!

TIP You can use the mouse to pick points in the drawing during commands that are being recorded by the RecScript command. BricsCAD records the pick points as x,y coordinates.

Writing Scripts by Hand

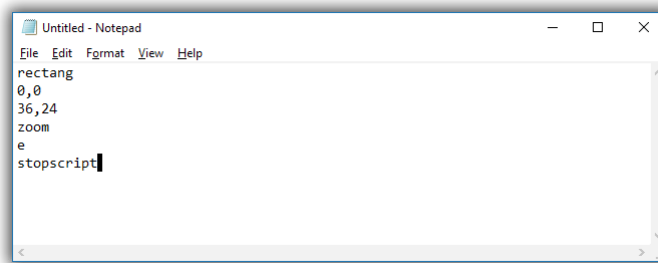
While BricsCAD has commands for creating and running scripts, it has not command for editing them. If you want to change the coordinates used by the `Rectang` command, you have to edit the script file with Notepad in Windows, `Text Edit` in Linux, or `TextEdit` in Mac.

Here is how it works:

1. Open the `border.scr` file in the text editor.

TIP If you are not sure where the `border.scr` file is located on your computer, here is a quick way to find and open it in Windows: start the **Script** command, and then in the dialog box right-click the `.scr` file. From the shortcut menu, select **Open**. Notice that the file opens in Notepad.

Notice the commands and options that you entered during the script recording session:



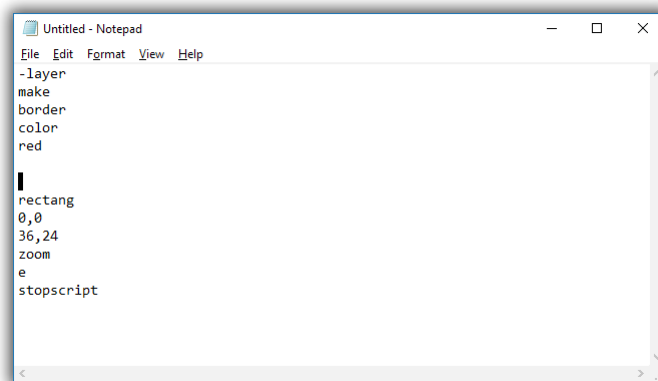
```
Untitled - Notepad
File Edit Format View Help
rectang
0,0
36,24
zoom
e
stopscript
```

Entering a script in a text editor

2. Let's change the size of the border to 18x24. Edit the "36,24" text, replacing it with...
18,24
3. Let's also add the command for placing the rectangle on a layer named "Border" and colored blue:
 - a. Place the cursor in front of "rectang," and then press **Enter** to make an empty line.
 - b. Enter the following text:

```
-layer
make
border
color
red
```

 - <-- One blank line
 - <-- A second blank line
 - c. Make sure you include two blank lines; these act like pressing **Enter** during commands. The file should look like this now:

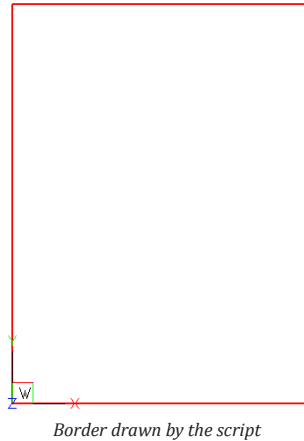


```
Untitled - Notepad
File Edit Format View Help
-layer
make
border
color
red

rectang
0,0
36,24
zoom
e
stopscript
```

Adding Enters to the script

4. Save the file with the **File | Save** command.
5. Return to BricsCAD, and then start a new drawing.
6. Use the **Script** command to test that the *border.scr* file is operating correctly. You should see a red rectangle.



Script Commands and Modifiers

There are a grand total of four commands that relate specifically to scripts. In fact, these commands are of absolutely no use for any other purpose. In addition, BricsCAD has the **RecScript** command for recording scripts, as described earlier in this chapter.

In rough order of importance, the four basic commands are:

SCRIPT

The **Script** command performs double-duty: (1) it loads a script file; and (2) immediately begins running it. Use it like this:

```
: script  
Script to run: filename
```

Remember to turn off (set to 0) the **FileDia** system variable so that the prompts appear at the command line, instead of the dialog box.

RSCRIPT

Short for “repeat script,” this command reruns whatever script is currently loaded in BricsCAD. A great way to create infinite loops. There are no options:

```
: rscript
```

RESUME

This command resumes a paused script file. Pause a script file by pressing the **Backspace** key. Again, no options:

```
: resume
```


DELAY

To create a pause in a script file without human intervention, use the **Delay** command along with a number. The number specifies the pause in milliseconds, where 1,000 milliseconds equal one second. The minimum delay is 1 millisecond; the maximum is 32767 milliseconds, which is just under 33 seconds.

While you could use **Delay** at the ':' prompt, that makes little sense; instead, **Delay** is used in a script file to wait while a slide file is displayed or to slow down the script file enough for humans to watch the process, like this:

```
; Pause script for ten seconds:  
delay 10000
```

SPECIAL CHARACTERS

In addition to the script-specific commands, there are some special characters and keys.

Enter - (space)

The most important special characters are invisible: both the space and the carriage return (or end-of-line) are the equivalent to when you press the spacebar or Enter keys. In fact, both are interchangeable. But the tricky part is that they are invisible. Sometimes, I'll write a script that requires a bunch of blank space because the command requires that I press the **ENTER** key several times in a row. **AttEdit** is an excellent example:

```
; Edit the attributes one at a time:  
attedit 1,2
```

How many spaces are there between **attedit** and the coordinates **1,2**? I'll wait while you count them...

For this reason, it is better to place one script item per line, like this:

```
; Edit the attributes one at a time:  
attedit
```

```
1,2
```

Now it's easier to count the four spaces, since there is one per blank line.

Comment - ;

You probably have already noticed that the semicolon lets you insert comments in a script file. BricsCAD ignores anything following the semicolon.

Transparent - '

Scripts can be run *transparently* during a command. Simply prefix the **Script** command with an apostrophe to run a script while another command is active, like this:

```
: line  
Start of line: 'script  
Script to run: filename
```

All four of BricsCAD's script-specific commands are transparent, even **'Delay**. That lets you create a delay during the operation of a command — as if I needed an excuse to run BricsCAD slowly!

Pause - Backspace

...is the key I mentioned earlier for pausing a script file.

Stop - ESC

...stops a script file dead in its tracks; use the **RScript** command to start it up again from the beginning

Programming with LISP

While toolbar and menu macros are easy to write and edit, they limit your ability to control BricsCAD. In this chapter, we look at the most powerful method available to “non-programmers” for customizing BricsCAD — the LISP programming language — at the cost of being somewhat more difficult to create than macros or scripts.

This chapter shows you how to write programs using LISP, while appendix C provides a concise reference to LISP functions.

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Learning the history of LISP in BricsCAD
- Checking the compatibility between LISP and AutoLISP
- Introducing the LISP programming language
- Employing simple LISP to add two numbers
- Using LISP in commands
- Overviewing LISP functions and external command functions
- Accessing system variables.
- Using advanced LISP functions
- Writing a simple LISP program
- Saving data to files

The History of LISP in BricsCAD

LISP is one of the earliest programming languages, developed in the late 1950s to assist artificial intelligence research. Its name is short for “list processing,” and it was designed to handle lists of words, numbers, and symbols.

LISP first appeared in CAD when, back in 1985, Autodesk added an undocumented feature to AutoCAD v2.15 called “Variables and Expressions.” Programmers at Autodesk had taken XLISP, a public domain dialect written by David Betz, and adapted it for AutoCAD. The initial release of Variables and Expressions was weak, because it lacked *conditional statements* -- needed by programming languages to make decisions.

With additional releases, Autodesk added the missing programming statements, the powerful GETxxx, SSxxx, and EntMod routines (that provide direct access to entities in the drawing database), and they renamed the programming language “AutoLISP.” This allowed third-party developers to write routines that manipulated the entire drawing, and non-programmers to write simple routines that automated everyday drafting activities.

When SoftDesk developed IntelliCAD, they included a programming language very similar to AutoLISP, calling it simply “LISP.” (I think it would have been better to call it IntelliLISP to prevent confusion with the real LISP programming language. Better yet, they could have given it the trendy moniker of iLISP.)

BLADE ENVIRONMENT

BricsCAD includes LISP, and supports VisualLISP (not covered by this book). With V18, Bricsys includes an advanced LISP authoring environment called Blade: Bricsys LISP advanced development environment (not covered by this book). To start it, enter the **blade** command from within BricsCAD. More information about Blade search online for “bricsys blade,” such as <https://blog.bricsys.com/inside-bricsys-blade/>.

COMPATIBILITY BETWEEN LISP AND AUTOLISP

LISP in BricsCAD is, for the most part, compatible with AutoCAD’s AutoLISP. If you know AutoLISP, you can program immediately in LISP, including controlling dialog boxes. LISP has, however, some differences of which you should be aware.

Additional LISP Functions

LISP in BricsCAD contains additional functions not found in AutoLISP. These include the following:

LISP Function	Meaning
acos	Arc cosine
asin	Arc sine
atanh	Hyperbolic arc tangent
ceiling	smallest integer that is not smaller than x.
cosh	Hyperbolic cosine
find	Finds an item in a list
floor	Greatest integer less than or equal to x
get_diskserialid	Unique nine-digit id string
getpid	Process ID of the current process
grarc	Draws a temporary arc or circle, with specified radius and color; optionally highlighted
grfill	Draws temporary filled polygon area, with specified color; optionally in highlighted mode
log10	Log 10
position	Index number of an item in a list
remove	Removes an item from a list
round	Rounds to the nearest integer
search	Searches for an item, and returns its list number
sinh	Hyperbolic sine
sleep	Pause execution
string-split	Splits a string based on a delimiter
tan	Tangent
tanh	Hyperbolic tangent
until	Tests the expression until it is true
vla-collection->list	Returns a collection as a LISP list

Different LISP Functions

LISP has several functions that operate differently from AutoLISP, by providing additional support.

These include:

LISP Function	Comment
osnap	Supports PLA (planview) entity snap for snapping to 2D intersections.
ssget <i>and</i> ssadd	Supports additional selection modes: CC - Crossing Circle O - Outside OC - Outside Circle OP - Outside Polygon PO -POint

Missing AutoLISP Functions

LISP lacks some functions found in AutoLISP. Because of the dynamic nature of LISP, it's difficult to create a definitive list. Here are some of the functions I have found missing:

- All **dict**-related functions.
- All SQL-related functions, which link between objects in the AutoCAD drawing with records in an external database file. In AutoCAD, these functions start with "ase_", as in ase_lsunit and ase_docmp.

The LISP Programming Language

LISP is capable of many tasks, from adding together two numbers — during the middle of a command — to drawing parametrically a staircase in 3D, to generating a new user interface for BricsCAD, to manipulating data in the drawing database.

The most important aspect of LISP, in my opinion, is that it lets you toss off a few lines of code to help automate your work. In this chapter's tutorials, I show you how to write simple LISP code that makes your BricsCAD drafting day easier.

In contrast, BricsCAD's most powerful programming facility — known as SDS (solutions development system) — is merely an interface: you have to buy additional the programming tools (read: \$\$\$) and have an in-depth knowledge of advanced programming methodology. The primary advantage to using SDS is speed: these programs run compute-intensive code as much as 100 times faster than LISP.

SIMPLE LISP: ADDING TWO NUMBERS

With that bit of background, let's dive right into using LISP. Let's start with something easy, something everyone knows about, adding together two numbers, like 9 plus 7.

1. Start BricsCAD, any version; there is no need to open a drawing.
2. When the ':' command prompt appears, type the boldface text, shown below, on the keyboard:
: (+ 9 7) (Press ENTER.)
16
:

BricsCAD instantly replies with the answer, 16. (In this chapter, I show the function I'm talking about in [cyan](#).) Getting to this answer through (+ 9 7) may, however, seem convoluted to you. That's because LISP uses *prefix notation*:

The operator + appears before the operands, 9 and 7.

Think of it in terms of "add 9 and 7." This is similar to how BricsCAD itself works: type in the command name first (such as **Circle**), and then enter the coordinates of the circle.

3. Notice the parentheses that surround the LISP statement. Every opening parenthesis, (, requires a closing parenthesis,). I can tell you right now that balancing parentheses is the most frustrating aspect to LISP. Here's what happens when you leave out the closing parentheses:
: (+ 9 7 (Press ENTER.)
Missing: 1) >

BricsCAD displays the "Missing: 1)" prompt to tell you that one closing parenthesis is missing. If two closing parentheses were missing, the prompt would read "Missing: 2)".

4. Type the missing) and BricsCAD is satisfied:
Missing: 1) >) (Press ENTER.)
16
:
5. The parentheses serve a second purpose: they alert BricsCAD that you are using LISP. If you were to enter

the same LISP function '+ 7 9' without parentheses, BricsCAD would react unfavorably to each character typed, interpreting each space as the end of a command name:

```
: + (Press the spacebar.)
Unable to recognize command. Please try again.

: 9 (Press the spacebar.)
Unable to recognize command. Please try again.

: 7 (Press the spacebar.)
Unable to recognize command. Please try again.
:
```

6. As you might suspect, LISP provides all the basic arithmetic functions: addition, subtraction, multiplication, and division. Try each of the functions, subtraction first:

```
: (- 9 7)
2
:
```

7. Multiplication is done using the familiar * (asterisk) symbol, as follows:

```
: (* 9 7)
63
:
```

8. Finally, division is performed with the / (slash) symbol:

```
: (/ 9 7)
1
:
```

Oops, that's not correct! Dividing 9 by 7 is 1.28571, not 1. What happened? Up until now, you have been working with *integer* numbers (also known as *whole* numbers). For that reason, LISP has been returning the results as integer numbers, although this was not apparent until you performed the division.

To work with real numbers, add a decimal suffix, which can be as simple as .0 — this converts integers to real numbers, and forces LISP to perform real-number division, as follows:

```
: (/ 9.0 7)
1.28571
:
```

And LISP returns the answer correct to five decimal places.

9. Let's see how LISP lets you *nest* calculations. "Nest" means to perform more than one calculation at a time.

```
: (+ (- (* (/ 9.0 7.0) 4) 3) 2)
4.14286
:
```

Note how the parentheses aid in separating the nesting of the terms.

LISP IN COMMANDS

Okay, so we've learned how BricsCAD works as a \$495 four-function calculator. This overpriced calculator pays its way when you employ LISP to perform calculations within commands. For example, you may need to draw a linear array of seven circles to fit in a 9" space.

1. Start the **Circle** command, as follows:

```
: circle
2Point/3Point/RadTanTan/Arc/Multiple/<Center of circle>: (Pick a point.)
```

2. Instead of typing the value for the diameter, enter the LISP equation, as follows:

```
Diameter/<Radius>: (/ 9.0 7)

Diameter/<Radius>: 1.28571
```

BricsCAD draws a circle with a diameter of 1.28571 inches. You can use an appropriate LISP function anytime BricsCAD expects user input.

3. Now go on to the **Array** command, and draw the other six circles, as follows:

```
: array
Select entities to array: L
Entities in set: 1 Select entities to array: (Press ENTER.)
Type of array: Polar/<Rectangular>: r
Number of rows in the array <1>: (Press ENTER.)
Number of columns <1>: 7
Horizontal distance between columns: (/ 0.9 7)
Horizontal distance between columns: 0.128571
```

Once again, you use LISP to specify the array spacing, which happens to equal the circle diameter.

REMEMBERING THE RESULT: SETQ

In the above example, you used the $(/ 9.0 7)$ equation twice: once in the **Circle** command and again in **Array**. Just as the **M**-key on a calculator lets it remember the result of your calculation, LISP can be made to remember the results of *all* your calculations.

To do this, employ the most common LISP function, known as **setq**. This curiously named function is short for *SET equal to*.

1. To save the result of a calculation, use the **setq** function together with a *variable*, as follows:

```
: (setq x (/ 9.0 7))
1.28571
:
```

Here, **x** remembers the result of the $(/ 9.0 7.0)$ calculation. Notice the extra set of parentheses.

From algebra class, you probably recall equations like ' $x = 7 + 9$ ' and ' $x = 7 / 9$ '. The **x** is known as a *variable* because it can have any value.

2. To prove to yourself that **x** contains the value of 1.28571, use BricsCAD's **!** (exclamation) prefix, as follows:

```
: !x
1.28571
:
```

The **!** prefix (sometimes called "bang") is useful for reminding yourself of the value contained by a variable, in case you've forgotten, or are wondering what happened during the calculation.

LISP isn't limited to just one variable. You can make up any combination of characters to create variable names, such as **pt1**, **diameter**, and **yvalue**. The only limitation is that you cannot use LISP function names, such as **setq**, **T**, and **getint**. In fact, it is good to create variable names that reflect the content, such as the circle diameter calculated above. But you also want to balance a descriptive name, such as **diameter**, with minimized typing, such as **x**. A good compromise is **dia**.

3. You make one variable equal another, as follows:

```
: (setq dia x)
1.28571
```

```
: !dia
1.28571
:
```

4. Redo the **Circle** and **Array** commands, this time using variable **dia**, as follows:

```
: circle
2Point/3Point/RadTanTan/Arc/Multiple/<Center of circle>: (Pick a point.)
Diameter/<Radius>: !dia
Diameter/<Radius>: 1.28571
```



```
: array
Select entities to array: L
Entities in set: 1 Select entities to array: (Press ENTER.)
Type of array: Polar/<Rectangular>: r
Number of rows in the array <1>: (Press ENTER.)
Number of columns <1>: 7
Horizontal distance between columns: !dia
Horizontal distance between columns: 0.128571
```

BricsCAD draws precisely the same seven circles, using the value 1.28571 stored in **dia**.

LISP Function Overview

LISP is so powerful that it can manipulate almost any aspect of the BricsCAD drawing. In the following tutorial, you get a taste of the many different kinds of functions LISP offers you for manipulating numbers and words. As we start on our whirlwind tour of several groups of functions, start BricsCAD, and then type the examples in the **Prompt History** window (press **F2**) at the ':' command prompt.

MATH FUNCTIONS

In addition to the four basic arithmetic functions, LISP has many of the mathematical functions you might expect in a programming language. The list includes trigonometric, logarithmic, logical, and bit manipulation functions; one type of function missing is matrix manipulation.

For example, the **min** function returns the smallest (minimum) of a list of numbers:

```
: (min 7 3 5 11)
3
```

To remember the result of this function, add **setq** with variable **minnbr**, as follows:

```
: (setq minnbr (min 7 3 5 11))
3
```

Now each time you want to refer to the minimum value of that series of numbers, you can refer to variable **minnbr**. Here's an example of a trig function, sine:

```
: (sin minnbr)
0.14112
```

Returns the sine of the angle of 3 radians.

TIPS You must provide the angle in radians, not degrees. This is many times an inconvenience, because often you work with degrees, but must convert them to radians.

Fortunately, LISP can do this for you, as long as you code it correctly. Recall that there are 2π (approximately 6.282) radians in 360 degrees. For example, to get the sine of 45 degrees, you have to indulge in some fancy footwork:

```
: (sin (* (/ 45 180.0) pi))  
0.707107
```

Here I divided the degrees (45) by 180, then multiplied by **pi**. Either the 45 or the 180 needs a decimal (.0) to force division by real numbers, rather than by integers.

By the way, **pi** is the only constant predefined in LISP, and is equal to 3.1415926. That means you just type **pi**, instead of 3.1415926 each time you need the value of pi in a function. To see this for yourself, use the exclamation mark at the command prompt:

```
!:pi  
3.14159
```

LISP displays the result to six decimal places, even though it performs calculations to 32-bit accuracy.

GEOMETRIC FUNCTIONS

Since CAD deals with geometry, LISP has a number of functions for dealing with geometry.

Distance Between Two Points

The LISP **distance** function is similar to BricsCAD's **Dist** command: it returns the 3D distance between two points. To see how it works, first assign x,y-coordinates to a pair of points, **p1** and **p2**, as follows:

```
: (setq p1 '(1.3 5.7))  
(1.3 5.7)  
  
: (setq p2 '(7.5 3.1 11))  
(7.5 3.1 11)  
  
: (distance p1 p2)  
6.72309
```

You may have missed that single quote mark in front of the list of x,y-coordinates, as in: '(1.3 5.7). That tells LISP you are creating a *pair* (or *triple* in the case of x,y,z) of coordinates, and that it should not evaluate the numbers. Technically, the ' mark creates a *list* of numbers.

To separate the coordinates use spaces, not commas. Note that when you leave out the z-coordinate, LISP assumes it equals 0.0000.

The Angle from 0 Degrees

Other geometric functions of interest include finding the angle from 0 degrees (usually pointing east) to the line defined by **p1** and **p2**:

```
: (angle p1 p2)  
5.88611
```

The result is returned in radians: 5.88611.

The Intersection of Two Lines

The intersection of two lines is determined by the **inters** function:

```
: (inters pt1 pt2 pt3 pt4)
```

Entity Snaps

In the following function, you are finding the midpoint of the line that starts at **p1**. You apply the **osnap** function and specify the type of osnap; LISP returns the x,y,z-coordinates of the entity snap point. The entity must actually exist.

```
: line
From point: !p1
To point: !p2
To point: (Press ENTER.)

: (osnap p1 "mid")
(4.4 4.4 5.5)
```

Here “**mid**” refers to the midpoint entity snap mode.

The other geometric functions include **textbox** (for finding the rectangular outline of a line of text) and **Polar**, which returns a 3D point of a specified distance and angle.

CONDITIONAL FUNCTIONS

You could say that *conditional* functions are most important, because they define the existence of a programming language. It is conditionals that allow a computer program to “think” and make decisions. Conditional functions check if one value is less than, equal to, or greater than another value. They check if something is true; or they repeat an action until something is false.

If you’re not sure if it’s a programming language or merely a macro language, check for conditionals. Toolbar macros, for example, have no conditionals; they are not a programming language.

Here is an example of how conditional functions operate: *if the floor-to-ceiling distance is greater than eight feet, then draw 14 steps; else, draw 13 steps.* Notice that there are two parts to the statement: the *if* part is the true part; the *else* part is the false part. Do something if it is true; otherwise, so something else if it is false.

Similar wording is used in LISP’s condition functions. Enter the following at the ‘:’ prompt:

```
: (if (> height 96) (setq steps 14) (setq steps 13))
13
```

Let’s break down this code to see how the **if** function compares with our statement:

(if	If
(>	greater than
height	floor-to-ceiling distance is
96)	8 feet;
(setq steps 14)	Then
	use 14 steps.
(setq steps 13)	Else
)	use 13 steps.

Other Conditionals

The **if** function is limited to evaluating just one conditional. The **cond** functions evaluate many conditions. The **repeat** function executes a specific number of times, while the **while** function executes code for as long as it is true.

STRING AND CONVERSION FUNCTIONS

You can manipulate *strings* (text consisting of one or more characters) in LISP, but to a lesser extent than numbers. For example, you can find the length of a string as follows:

```
: (strlen "BricsCAD World")  
16
```

The **strlen** (short for *STRing LENgth*) function tells you that “BricsCAD World” has 16 characters in it, counting the space. Notice how “BricsCAD World” is surrounded by quotation marks. That tells LISP you are working with a string, not a variable.

If you were to type **(strlen BricsCAD World)**, LISP tries to find the length of the strings held by variables BricsCAD and World. For example:

```
: (setq BricsCAD "A software package")  
"A software package"  
  
: (setq world "the planet earth")  
"the planet earth"  
  
: (strlen BricsCAD world)  
34
```

Joining Strings of Text

Other string functions change all characters to upper or lower case (**strcase**), returns part of a string (**substr**), searches and replaces text in a string (**subst**), and join two strings together (**strcat**), as follows:

```
: (strcat BricsCAD " used all over " world)  
"A software package used all over the planet earth"
```

That’s how you create reports, such as “13 steps drawn”, by mixing variables and text.

Converting Between Text and Numbers

Related to string functions are the conversion functions, because some of them convert to and from strings. For example, earlier I showed how to convert degrees to radians. That’s fine for decimal degrees, like 45.3711 degrees. But how do you convert 45 degrees, 37 minutes and 11 seconds, which BricsCAD represents as 45d37’11”? That’s where a conversion function like **angtof** (short for *ANGLE TO Floating-point*) comes in. It converts an angle string to real-number radians:

```
: (angtof "45d37'11\" 1)  
0.796214
```

Here we’ve supplied **angtof** with the angle in degrees-minutes-seconds format. However, LISP isn’t smart enough to know, so we tell it by means of the *mode* number, 1 in this case.

This (and some other functions) use the following as mode codes:

Mode	Meaning	Example
0	Decimal degrees	45.3711
1	Degrees-minutes-seconds	45d 37' 11"
2	Grad	100.1234
3	Radian	0.3964
4	Surveyor units	N 45d37'11" E

Notice the similarity between the mode numbers and the values of system variable **AUnits** — and the modes used by Diesel. The coincidence is not accident. When you don't know ahead of time the current setting of units, you make use of this fact by specifying the mode number as a variable, as follows:

```
: (angtof "45d37'11\" (getvar "aunits"))
0.796214
```

Here we use **getvar** (short for *GET VARIABLE*), the LISP function that gets the value of a system variable. We used **getvar** to get **aunits**, which holds the state of angular display as set by the **Units** command.

Notice how the seconds indicator (") is handled: \". That's so it doesn't get confused with the closing quote mark (") that indicates the end of the string.

Other Conversion Functions

Other conversion functions convert one unit of measurement into another (via the **cvunit** function and the *default.unt* file), an integer number into a string (**itos**), a character into its ASCII value (**ascii**: for example, letter A into ASCII value 65), and translates (moves) a point from one coordinate system to another (**trans**).

The *default.unt* file is found in the *C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20\en_US\Support* folder.

EXTERNAL COMMAND FUNCTIONS

“Powerful” often equates to “complicated,” yet one of LISP's most powerful functions is its simplest to understand: the **command** function. As its name suggests, **command** executes BricsCAD commands from within LISP.

Think about it: this means that it is trivial to get LISP to draw a circle, place text, zoom a viewport, whatever. Anything you *type* at the ':' command prompt is available with the **command** function. Let's see how **command** works by drawing a circle. First, though, let's recall how the **Circle** command operates:

```
: circle
2Point/3Point/RadTanTan/Arc/Multiple/<Center of circle>: 2,2
Diameter/<Radius>: D
Diameter of circle: 1.5
```

Switching to the **command** function, you mimic what you type at the ‘:’ prompt, as follows. (This is where Chapter 16’s practice in creating script files is handy.)

```
: (command "circle" "2,2" "D" "1.5")
```

Notice how all typed text is in quotation marks. After you enter that line of code, BricsCAD responds by drawing the circle:

```
: circle
2Point/3Point/RadTanTan/Arc/Multiple/<Center of circle>: 2,2
Diameter/<Radius> <1.2857>: D
Diameter of circle <2.5714>: 1.5
```

Let’s look at one of the more complex commands to use with the **command** function, **Text**. When we use the **Text** command, BricsCAD presents these prompts:

```
: text
Text: Style/Align/Fit/Center/Middle/Right/Justify/<Start point>: 5,10
Height of text <0.2000>: 1.5
Rotation angle of text <0>: (Press ENTER.)
Text: Tailoring BricsCAD
```

Converted to LISP-ese, this becomes:

```
: (command "text" "5,10" "1.5" "" "Tailoring BricsCAD")
```

And BricsCAD responds with:

```
: text
Text: Style/Align/Fit/Center/Middle/Right/Justify/<Start point>: 5,10
Height of text <1.5000>: 1.5
Rotation angle of text <0>:
Text: Tailoring BricsCAD
```

and then draws the text.

For the ‘Rotation angle:’ prompt, we simply pressed the **ENTER** key. Notice how that is dealt with in the LISP function: “” — a pair of empty quotation marks.

You use the same “” to end commands that automatically repeat themselves, such as the **Line** command:

```
: (command "line" "1,2" "3,4" "")
```

When you don’t include that final “”, then you leave BricsCAD hanging with a ‘End point:’ prompt and your LISP routine fails.

By now it should be clear to you that you have to really know the prompt sequence of BricsCAD’s more than 300 commands to work effectively with the **command** function. The easiest way to get a handle on those is to purchase one of the “quick reference” books on the market, which list commands in alphabetical order, along with the complete prompt sequence. And, as we see in a minute, check that the quick reference book has a listing of all system variables, their default value, and the range of permissible values.

Command Function Limitation

But the **command** function has a failing. Earlier, I said, “Anything you *type* at the ‘:’ command prompt is available with the **command** function.” I now place emphasis on the word “type.” The **command** function breaks down completely when it comes to dialog boxes. That’s right: any command that uses a dialog box won’t work with the command function — nor, for that matter, with the macros we looked at in previous chapters. It is for this reason that BricsCAD includes command-line versions of almost every (but not all) command.

Accessing System Variables

While you can use the **command** function to access system variables, LISP has a pair of more direct functions: **getvar** and **setvar**.

Getvar gets the value of a system variable, while **setvar** changes (sets) the value.

For example, system variable **SplFrame** determines whether the frame of a spline polyline is displayed; by default, the value of **SplFrame** is 0: the frame is not displayed, as confirmed by **getvar**:

```
: (getvar "splframe")  
0
```

To display the frame, change the value of **SplFrame** to 1 with **setvar** as follows:

```
: (setvar "splframe" 1)  
1
```

We have, however, made a crass assumption: that the initial value of **SplFrame** is 0. Zero is the default value, but not necessarily the value at the time that you run the LISP routine. How do we know what the value of **SplFrame** is before we change it? We’ll answer that question later in this chapter. Stay tuned.

GETXXX FUNCTIONS

It’s one thing to execute a command that draws a new entity, such as the circle and text we drew above with the **command** function. It is trickier working with entities that already exist, such as moving that circle or editing the text. That’s where the a group of functions known collectively as **Getxxx** come into play. These functions get data from the screen. Some of the more useful ones include:

getpoint	Returns the x,y,z-coordinate of a picked point.
getangle	Returns the angle in radians.
getstring	Returns the text typed by the user.
getreal	Returns the value of a real number typed by the user.

Here’s how to use some of these with the **Text** command. Let’s redo the code with **getstring** so that LISP prompts us for everything first, then executes the **Text** command. Here is the first line of code, which prompts the user to input some text:

```
: (setq TxtStr (getstring T "What do you want to write? "))  
What do you want to write?
```

Notice that extra “T”; that’s a workaround that lets **getstring** accept a string of text with spaces. When you leave out the **T**, then **getstring** accepts text up to the first space only. If you were to enter “Tailoring BricsCAD”, you would end up with just “Tailoring” and no “BricsCAD.”

Also in the line of code above, the **setq** function stores the phrase, such as “Tailoring BricsCAD,” in the variable **TxtStr**.

In the next line of code, we use the **getreal** function to ask for the height of text, which is a real number (decimal) entered by the user:

```
: (setq TxtHt (getreal "How big do you want the letters? "))
How big do you want the letters? 2
2.0
```

Notice how **getreal** converts the 2 (an integer) to a real number, 2.0. The value is stored in variable **TxtHt**.

Next, we use the **getangle** function to ask for the rotation angle of the text:

```
: (setq TxtAng (getangle "Tilt the text by how much? "))
Tilt the text by how much? 30
0.523599
```

Notice how **getangle** converts the 30 (a decimal degree) into radians, 0.523599. The value is stored in variable **TxtAng**.

Then, we use the **getpoint** function to ask the user for the insertion point of the text:

```
: (setq TxtIns (getpoint "Where do you want the text to start? "))
Where do you want the text to start? (Pick a point.)
(27.8068 4.9825 0.0)
```

Notice how **getpoint** returns the x, y, and z values of the coordinate, even though z is zero. The user can pick a point on the screen, or enter a coordinate pair (x,y) or triple (x,y,z).

Finally, we execute the **Text** command with the four variables:

```
: (command "text" TxtIns TxtHt TxtAng TxtStr)
text Justify/Style:
Height <1.5000>: 2.0000000000000000
Rotation angle <0>: 0.523598775598299
Text: Tailoring BricsCAD
: nil
```

There! We’ve just customized the **Text** command to our liking. Not only did we change the prompts that the user sees, but we used LISP to change the order of the prompts.

SELECTION SET FUNCTIONS

To work with more than one entity at a time, LISP has a group of functions for creating selection sets. These all begin with “SS”, as in:

SsAdd	Adds entities to selection sets.
SsDel	Deletes entities from selection sets.
SsGetFirst	Reports the number of selected entities.
SsLength	Reports the number of entities in the selection set.
SsMemb	Checks if entities are part of a selection set.
SsName	Identifies the nth entity in a selection set.
SsSetFirst	Highlights objects in a selection set.

BricsCAD’s **Select** command can deal only with one selection set at a time; in contrast, the LISP SSxxx commands can work with up to 128 selection sets.

ENTITY MANIPULATION FUNCTIONS

The really powerful LISP functions are the ones that go right in and manipulate the drawing database. Unlike the **command** function, which is powerful but simple, the entity manipulation functions are powerful and complicated. Here’s a summary of what some of these are:

EntMake	Creates new entities.
EntGet	Gets the data that describes entities in drawings.
EntMod	Changes entities.
EntDel	Erases entities from the database.
TblObjName	Gets the names of entities in symbol tables.

The “Ent” prefix is short for entity. The “symbol table” refers to the part of the drawing database that stores the names of layers, text styles, and other named entities in the drawing.

To create and manipulate entities, these LISP functions work with a variant on the DXF format, known as “dotted pairs.” For example, to work with a layer named RightOfWay, you employ the following format:

```
"2 . RightOfWay"
```

The quotation marks indicate the start and end of the data, while the dot in the middle separates the two values: The **2** is the DXF code for layer names, and **RightOfWay** is the name of the layer. You can see that to work with these entity manipulation functions, you need a good grasp of the DXF format.

ADVANCED LISP FUNCTIONS

There is a whole host of LISP functions that you may never use in your BricsCAD programming career. For example, there are LISP functions for controlling the memory, such as **as gc** (garbage collection) and **mem** (memory status). Another set of LISP functions are strictly for loading and displaying dialog boxes, such as **load_dialog** and **new_dialog**.

Writing a Simple LISP Program

In this section, you learn the first steps in writing a LISP routine of your own.

WHY WRITE A PROGRAM?

If you are like many CAD users, you are busy creating drawings, and you have no time to learn how to write software programs. No doubt, you may be wondering, “Why bother learning a programming language?” In some ways, it’s like being back again in school. Sitting in the classroom sometimes seems like a waste of time.

But the things you learn now make life easier later. Learning some LISP programming now means you’ll feel really good whipping off a few lines of code to let LISP perform tedious tasks for you. The nice thing about LISP is that you can program it on the fly. And you can use it for really simple but tedious tasks.

Here’s the example we’ll use for this tutorial:

The Id Command

BricsCAD has the **Id** command. When you pick a point on the screen, **Id** reports the 3D x,y,z- coordinates of the point. Problem is, **Id** reports the value in the command prompt area, like this:

```
: id  
Select a point to identify coordinates: (Pick a point.)  
X = 8.9227 Y = 6.5907 Z = 0.0000
```

Wouldn’t it be great if you could change **Id** so that it places the coordinates in the drawing, next to the pick point? That would let you label x,y-coordinates and z-elevations over a site plan. With LISP, you can.

THE PLAN OF ATTACK

Before you write any LISP code, you need to figure out how you’re going to get those x,y,z-coordinates off the command prompt area, and into the drawing. Recognize that there are two parts to solving the problem:

- Part 1.** Obtain the coordinates from the drawing, probably by picking a point.
- Part 2.** Place the coordinates as text in the drawing.

Obtaining the Coordinates

LISP provides several ways to get the coordinates of a picked point. Browsing through the *LISP Programming Language Reference*, you learn you could:

- ▶ Use the **Id** command with the **command** function, as in **(command “ID”)**.
- ▶ Use the **LastPoint** system variable with the **getvar** function, as in **(getvar “lastpoint”)**.
- ▶ Use the **getpoint** function, as in **(getpoint “Pick a point: ”)**

It would be a useful lesson to use each of the three, and then see what happens. By experimenting, you make mistakes, and then learn from the mistakes.

1. Start BricsCAD, load a drawing, and switch to the Prompt History window with **F2**. At the ‘:’ prompt, enter:
: (command "ID")

Here you are executing an BricsCAD command (**Id**) from within a LISP routine. The **command** function lets you use any BricsCAD command in LISP. The BricsCAD command is in quotation marks “**ID**” because the command is a *string* (programmer-talk for “text”). Just as before, BricsCAD prompts you for the point.

2. In response to the LISP routine’s prompt, pick a point:
Select a point to identify coordinates: (*Pick a point.*)
X = 8.9227 Y = 6.5907 Z = 0.0000

3. Unknown to you, BricsCAD always stores the x,y,z-coordinates of the last-picked point in a system variable called **LastPoint**. So, you should copy the coordinates from **LastPoint** to a variable of your own making. You need to do this because the coordinates in **LastPoint** are overwritten with the next use of a command that makes use of a picked point.

Recall from in this chapter that the **setq** function stores values in variables. Make use of it now. At the ‘:’ prompt, enter:

```
: (setq xyz (getvar "LastPoint"))  
(8.9227 6.5907 0.0000)
```

- **Xyz** is the name of the variable in which you store the x,y,z-coordinate.
- **Getvar** is the name of the LISP function that retrieves the value stored in a system variable.
- And “**LastPoint**” is the name of the system variable; it is surrounded by quotation marks because it is a system variable name (a string).

After entering the LISP function, BricsCAD returns the value it stored in variable **xyz**, such as (8.9227 6.5907 0.0000) — your result will be different. Notice how the coordinates are surrounded by parenthesis. This is called a *list*, for which LISP is famous (indeed, LISP is short for “list processing”). Spaces separate the numbers, which are the x, y, and z-coordinates, respectively:

```
x      8.9227  
y      6.5907  
z      0.0000
```

BricsCAD always stores the values in the order of x, y, and z. You will never find the z- coordinate first or the x-coordinate last.

So, we’ve now solved the first problem in one manner. We obtained the coordinates from the drawing, and then stored them in a variable. We did mention a third LISP function we could use, **getpoint**. Programmers prefer **getpoint** because it is more efficient than the **Id-LastPoint** combo we used above.

Type the following to see that it works exactly the same, the difference being that we provide the prompt text (“Point: ”):

```
: (setq xyz (getpoint "Point: "))  
Point: (Pick a point.)  
(8.9227 6.5907 0.0000)
```

As before, we use the **setq** function to store the value of the coordinates in variable **xyz**. The **getpoint** function waits for you to pick a point on the screen. The “**Point:**” is called a prompt, which tells the user what the program is expecting the user to do. We could just as easily have written anything, like:

```
: (setq xyz (getpoint "Press the mouse button: "))
Press the mouse button: (Pick a point.)
(8.9227 6.5907 0.0000)
```

Or, we could have no prompt at all, as follows:

```
: (setq xyz (getpoint))
(Pick a point.)
(8.9227 6.5907 0.0000)
```

That’s right. No prompt. Just a silent BricsCAD waiting patiently for the right thing to happen ... and the user puzzled at why nothing is happening. A lack of communication, you might say. That’s why prompts are important.

We’ve now seen a couple of approaches that solve the same problem in different ways. With the x,y,z-coordinates safely stored in a variable, let’s tackle the second problem

PLACING THE TEXT

To place text in the drawing, we can use only the **command** function in conjunction with the **Text** command. I suppose the **MText** command might work, but you want to place one line of text, and the **Text** command is excellent for that. The **Text** command is, however, trickier than the **Id** command. It has a minimum of four prompts that your LISP routine must answer:

```
: text
Text: Style/Align/Fit/Center/Middle/Right/Justify/<Start point>:
Height of text <2>:
Rotation angle of text <0>:
Text:
```

Start point: a pair of numbers, specifically an x,y-coordinate.

Height of text: a number to makes the text legible.

Rotation angle of text: a number, probably 0 degrees.

Text: the string, in our case the x,y,z-coordinates.

Let’s construct a LISP function for placing the x,y,z-coordinates as text:

```
(command "text" xyz 200 0 xyz)
```

(**command** is the **command** function.

“**text**” is the BricsCAD **Text** command being executed.

xyz variable stores the starting point for the text.

200 is the height of the text. Change this number to something convenient for your drawings.

0 is the rotation angle of the text.

xyz means you’re lucky: the **Text** command accepts numbers as text.

) and remember: one closing parenthesis for every opening parenthesis.

Try this out at the ‘:’ prompt:

```
: (command "text" xyz 200 0 xyz)
Text: Style/Align/Fit/Center/Middle/Right/Justify/<Start point>:
Height of text: 200
Rotation angle of text: 0
Text: 2958.348773815669,5740.821183398367
:
```

BricsCAD runs through the **Text** command, inserting the responses for its prompts, then placing the coordinates as text. We’ve solved the second part of the problem.

PUTTING IT TOGETHER

Let’s put together the two solutions to your problem:

```
(setq xyz (getpoint "Pick point: "))
(command "text" xyz 200 0 xyz)
```

There you have it: a full-fledged LISP program. Well, not quite. It’s a pain to retype those two lines each time you want to label a point. In the next section, you find out how to save the code as a *.lsp* file on disk. You’ll also dress up the code.

Adding to the Simple LISP Program

There you have it: a full-fledged LISP program. Well, not quite. What you have is the *algorithm* — the core of every computer program that performs the actual work. What is lacking is most of a *user interface* — the part that makes it easier for any user to employ the program.

All you have for a user interface is part of the first line that prompts, “Select point to identify coordinates: ”. There’s a lot of user interface problems with this little program. How many can you think of? Here’s a list of problems I came up with:

- ▶ It’s a pain to retype those two lines each time you want to label a point — you need to give the program a name ...
- ▶ ... and you need to save it on disk so that you don’t need to retype the code with each new BricsCAD session...
- ▶ ... and, if you use this LISP program a lot, then you should have a way of having it load automatically.
- ▶ The x,y,z-coordinates are printed to eight decimal places; for most users, that’s w-a-y too many.
- ▶ You may want to control the layer that the text is placed on.
- ▶ You may want a specific text style.
- ▶ Certainly, you would like some control over the size and orientation of the text.
- ▶ Here’s an orthogonal idea: store the x,y,z-coordinates to a file on disk — just in case you ever want to reuse the data.

CONQUERING FEATURE BLOAT

“Okay,” you may be thinking, “I can agree that these are mostly desirable improvements. Go right ahead, Mr. Grabowski: Show me how to add them in.”

But, wait a minute! When you’re not familiar with LISP, you may not realize how a user interface adds a tremendous amount of code, which mean more bugs and more debugging. (If you are familiar with programming, then you know how quickly a simple program fills up with feature-bloat.) While all those added features sound desirable, they may make the program less desirable. Can you imagine how irritated you’d get if you had to answer the questions about decimal places, text font, text size, text orientation, layer name, filename — each time you wanted to label a single point?

Take a second look at the wish list above. Check off features important to you, and then cross out those you could live without.

Wishlist Item #1: Naming the Program

To give the program a name, surround the code with the **defun** function, and give it a name, as follows:

```
(defun c:label ( / xyz)
  (setq xyz (getpoint "Pick point: "))
  (command "text" xyz 200 0 xyz)
)
```

Let’s take a look at what’s been added, piece by piece:

Defining the Function - defun

(defun defines the name of the function. In LISP, the terms function, program, and routine are used interchangeably (**defun** is short for “define function.”)

Naming the Function - C:

c:label is the name of the function. I decided to call this program “Label”; you can call it anything you like, so long as the name does not conflict with that of any built-in LISP function or other user-defined function. The **c:** prefix make this LISP routine appear like an BricsCAD command.

To run the Label program, all you need do is type “label” at the ‘:’ prompt, like this:

```
: label
Select a point to identify coordinates: (Pick a point.)
```

When the **c:** prefix is missing, however, then you have to run the program like a LISP function, complete with the parentheses, as follows:

```
: (label)
Select a point to identify coordinates: (Pick a point.)
```

Local and Global Variables - /

(/ xyz) declares the names of *input* and *local* variables; the slash separates the two:

Input variables — feed data to LISP routines; the names of input variables appear before the slash.

Local variables — used only within programs; the names of local variables appear after the slash.

In this program, **xyz** is the name of the variable that is used strictly within the program. If variables are not declared local, they become *global*. The value of a global variable can be accessed by *any* LISP function loaded into BricsCAD.

The benefit to declaring variables as local is that BricsCAD automatically frees up the memory used by the variable when the LISP program ends; the drawback is that the value is lost, making debugging harder. For this reason, otherwise-local variables are kept global until the program is debugged.

And the) closing parenthesis balances the opening parenthesis at the beginning of the program.

Wishlist Item #2: Saving the Program

By saving the program to a file on disk, you avoid retyping the code with each new BricsCAD session. You do this, as follows:

1. Start a text editor (the Notepad supplied with Windows or Text Edit with Linux and Mac are good).
2. Type the code shown:

```
(defun c:label ( / xyz)
  (setq xyz (getpoint "Pick point: "))
  (command "text" xyz 200 0 xyz)
)
```

I indented the code in the middle to make it stand out from the **defun** line and the closing parenthesis. This is standard among programmers; the indents make it easier to read code. You can use a pair of spaces or the tab key because LISP doesn't care.

3. Save the file with the name *label.lsp* in BricsCAD's folder.

Wishlist Item #3: Automatically Loading the Program

To load the program into BricsCAD, type the following:

```
: (load "label")
```

If BricsCAD cannot find the LISP program, then you have to specify the path. Assuming you saved *label.lsp* in the *\cad\support* folder, you would enter:

```
: (load "\\cad\\support\\label")
```

Now try using the point labelling routine, as follows:

```
: label
Select a point to identify coordinates: (Pick a point.)
```

TIP BricsCAD provides a way to automatically load LISP programs. When BricsCAD starts up, it looks for a file called *icad.lsp*. BricsCAD automatically loads the names of LISP programs listed in the file.

Adding *label.lsp* to *icad.lsp* is easy. Open the *icad.lsp* file with a text editor (if the file does not exist, then start a new file called *acad.lsp* and store it in the `\BricsCAD` folder). Add the name of the program:

```
(load "label.lsp")
```

Save the *icad.lsp* file. Start BricsCAD and it should load *label* automatically.

Wishlist #4: Using Car and Cdr

The x,y,z-coordinates are printed to eight decimal places — that’s too many. There are two solutions. One is to ask the user the number of decimal places, as shown by the following code fragment:

```
: (setq uprec (getint "Label precision: "))
Label precision: 1
1
```

Or steal the value stored in system variable **LUPrec** — the precision specified by the user through the **Units** command — under the (not necessarily true) assumption that the user want consistent units. The code to do this is as follows:

```
(setq uprec (getvar "LUPREC"))
```

That was the easy part. The tough part is applying the precision to the x,y,z-coordinates, which takes three steps: (1) pick apart the coordinate triplet; (2) apply the precision factor; and (3) join together the coordinates. Here’s how:

1. Open *label.lsp* in NotePad or other text editor. Remove `/xyz` from the code. This makes the variable “global,” so that you can check its value at BricsCAD’s ‘:’ prompt. The code should look like this:

```
(defun c:label ( )
  (setq xyz (getpoint "Pick point: "))
  (command "text" xyz 200 0 xyz)
)
```

2. Save, and then load *label.lsp* into BricsCAD.
3. Run *label.lsp*, picking any point on the screen. If you don’t see the coordinates printed on the screen, use the **Zoom Extents** command.
4. At the ‘:’ prompt, enter the following:
: !xyz
(6.10049 8.14595 10.0)

The exclamation mark forces BricsCAD to print the value of variable **xyz**, which holds the x,y,z-coordinates. Your results will differ, depending on where you picked.

5. LISP has several functions for picking apart a list. Here you use the **car** and **cdr** functions, and combinations thereof. The **car** function extracts the first item (the x-coordinate) from a list. Try it now:

```
: (car xyz)
6.10049
```

6. The **cdr** function is the compliment to **car**. It removes the first item from the list, and then gives you what’s left over:
: (cdr xyz)
(8.14595 10.0)

7. In addition to **car** and **cdr**, LISP allows me to combine the “a” and “d” in several ways to extract other items in the list. To extract the y-coordinate, use **cadr**, as follows:

```
: (cadr xyz)
8.14595
```


8. And to extract the z-coordinate, use **caddr**, as follows:

```
: (caddr xyz)
8.14595
```

9. I now have a way to extract the x-coordinate, the y-coordinate, and the z-coordinate from variable **xyz**. I'll store them in their own variables, as follows:

```
: (setq ptx (car xyz)
Missing: 1) > pty (cadr xyz)

Missing: 1) > ptz (caddr xyz)
Missing: 1) > )
```

You use variable **PtX** to store the x-coordinate, **PtY** for the y-coordinate, and so on. In addition, a form of LISP shorthand was used in the code above that allows you apply the **setq** function to several variables. Recall the reason for BricsCAD's 'Missing: 1)' prompt: it reminds you that a closing parenthesis is missing.

10. Now that the three coordinates are separated, you can finally reduce the number of decimal places. There are a couple of ways to do this. Use the **rtos** function, because it does two things at once: (1) changes the number of decimal places to any number between 0 and 8; and (2) converts the real number into a string. Why a string? You'll see later. For now, here is the **rtos** function at work:

```
: (rtos ptx 2 uprec)
"6.1"
```

The **rtos** function uses three parameters: **ptx**, **2**, and **uprec**.

PtX Name of the variable holding the real number.

- 2** Type of conversion, decimal in this case. The number **2** is based on system variable **LUnits**, which defines five modes of units:

Mode	Units
1	Scientific
2	Decimal
3	Engineering
4	Architectural
5	Fractional

UPrec Name of the variable holding the precision (the code for that is at the beginning of this section). This varies, depending on the type of units. For example, a value of 2 for decimal means two decimal places; a 2 for architectural means quarter-inch.

Assuming, then, that the precision in **UPrec** is 1, the **rtos** function in the code fragment above reduces 6.10049 to 6.1.

11. Truncate, and preserve the values of x, y, and z three times, as follows:

```
: (setq ptx (rtos ptx 2 uprec)
1> pty (rtos pty 2 uprec)
1> ptz (rtos ptz 2 uprec)
1> )
```

Notice that you can set a variable equal to itself: **PtX** holds the new value of the x-coordinate after **rtos** gets finished processing the earlier value stored in **PtX**. Reusing a variable name like this helps conserve memory.

12. With the coordinates truncated, you now have to string (pardon the pun) them together with the **strcat** function, short for string concatenation. Try it now:

```
: (strcat ptx pty ptz)
"6.18.110.0"
```

13. Oops! Not quite the look you may have been hoping for. Since LISP can't know when you want spaces, it provides none. You have to insert them yourself using **strcat**, one of the most useful LISP functions. It lets you create a string that contains text and variables, like this:

```
(setq xyz (strcat ptx " , " pty " , " ptz))
"6.1, 8.1, 10.0"
```

That's more like it!

14. Back to the text editor. Add in the code you developed here, shown in **boldface**, and with LISP functions in **cyan**:

```
(defun c:label ( / xyz xyz1 uprec ptx pty ptz)
  (setq uprec (getint "Label precision: "))
  (setq xyz (getpoint "Pick point: "))
  (setq ptx (car xyz)
        pty (cadr xyz)
        ptz (caddr xyz)
  )

  (setq ptx (rtos ptx 2 uprec)
        pty (rtos pty 2 uprec)
        ptz (rtos ptz 2 uprec)
  )

  (setq xyz1 (strcat ptx " , " pty " , " ptz))
  (command "text" xyz 200 0 xyz1)
)
```

Notice that all variables are local. Notice, too, the change to variable **xyz** in the last couple of lines: you don't want the text placed at the rounded-off coordinates, so use **xyz1** as the variable holding the text string.

15. Finally, you should add comments to your code to remind you what it does when you look at the code several months from now. Semicolons indicate the start of comments:

```
; Label.Lsp labels a picked point with its x,y,z-coordinates.
; by Ralph Grabowski, 25 February, 1996.
(defun c:label ( / xyz xyz1 uprec ptx pty ptz)

  ; Ask user for the number of decimal places:
  (setq uprec (getint "Label precision: "))

  ; Ask the user to pick a point in the drawing:
  (setq xyz (getpoint "Pick point: "))

  ; Separate 3D point into individual x,y,z-values:
  (setq ptx (car xyz)
        pty (cadr xyz)
        ptz (caddr xyz)
  )

  ; Truncate values:
  (setq ptx (rtos ptx 2 uprec)
        pty (rtos pty 2 uprec)
        ptz (rtos ptz 2 uprec)
  )

  ; Recombine individual values into a 3D point:
  (setq xyz1 (strcat ptx " , " pty " , " ptz))

  ; Place text:
  (command "text" xyz 200 0 xyz1)
)
```

16. Save the file as *label.lsp*, then load the LISP routine into BricsCAD with:

```
: (load "label")
"C:LABEL"
```

```
17. Run the routine, and respond to the prompts:
    : label
    Label precision: 1
    Pick point: (Pick a point.)
    text Justify.../⟨Start point⟩:
    Height of text <200.0000>: 200
    Rotation angle of text <0>: 0
    Text: 5012.3, 773.2, 0.0
    :
```

Saving Data to Files

In the previous tutorial, we begin to worry about user interface enhancements. What started out as two lines of code has now bulged out into 23. In this tutorial, we learn how to fight feature bloat (more later), and add the ability to save data to a file.

A reader wrote me with this wish list item: “The LISP file comes in very handy with some of the programs I use, but I would like to be able to save the data collected on the x,y,z-coordinates in a text file.”

Saving the data to file is easily done with the **open**, **write-line**, and **close** functions. Let’s take a look at how to do this. Dealing with files in LISP is simpler than for most programming languages because LISP has very weak file access functions. All it can do is read and write ASCII files in sequential order; LISP cannot deal with binary files nor can it access data in random order.

THE THREE STEPS

There are three steps in writing data to a file:

- Step 1. **Open** the file.
- Step 2. **Write** the data to the file.
- Step 3. **Close** the file.

Step 1: Open the File

LISP has the **open** function for opening files. The function lets you open files for *one* of three purposes: (1) read data from the file; (2) write data to the file; or (3) append data to the file. You must choose one of these at a time; LISP cannot do all three at once.

In all cases, LISP takes care of creating the file if it does not already exist. Reading data is easy enough to understand, but what’s the difference between “writing” and “appending” data?

- ▶ When I ask BricsCAD to open a file to **write**, all existing data in that file is *erased*, and then the new data is added.
- ▶ When I ask BricsCAD to open a file to **append**, the new data is *added* to the end of the existing data.

For our purpose, we want to keep adding data to the file, so choose *append* mode. The LISP code looks like this:

```
(setq FIL (open "xyzdata.txt" "a"))
```

Here you are setting something (through **setq**) equal to a variable named **FIL**. What is it? In pretty much all programming languages, we don't deal with the filename directly, but instead deal with a *file descriptor*. This is a name (some sequence of letters and numbers) to which the operating system assigns the filename. Now that you have the file descriptor stored in variable **FIL**, you work with **FIL**, not the filename, which I have decided to call *xyzdata.txt*.

The final "a" tells LISP you want to open *xyzdata.txt* for appending data. The options for the **open** function are:

Option	Comment
"a"	Appends data to end of file.
"w"	Writes data to file (erase existing data).
"r"	Reads data from file.

Step 2: Write Data to the File

To write data to files, use the **write-line** function. This function writes one line of data at a time. (Another function, the **write** function, writes single *characters* to files.) The code looks like this:

```
(write-line xyz1 fil)
```

You cannot, however, just write raw data to the file because it would look like three decimal points and a lot of digits, like this:

```
8.15483.27520.0000
```

Most software is able to read data with commas separating numbers, like this:

```
8.1548, 3.2752, 0.0000
```

That includes spreadsheets, database programs, and even some word processing software. I tell these programs that when they read the data, they should consider the comma to be a *separator* and not a comma. In that way, the spreadsheet program places each number in its own cell. With each number in its own cell, I can manipulate the data. For this reason, you need code that formats the data.

Fortunately, you've done that already. Last tutorial, you used the **strcat** function along with the **cdr**, **cadr**, and **caddr** functions to separate the x, y, and z components of the coordinate triplet. So you can reuse the code, which looks like this:

```
(setq ptx (car xyz)
      pty (cadr xyz)
      ptz (caddr xyz)
)
(setq xyz1 (strcat ptx " " pty " " ptz))
```

The **strcat** function places the commas between the coordinate values.

Step 3: Close the File

Finally, for good housekeeping purposes, close the file. BricsCAD will automatically close the file for you if you forget, but a good programmer cleans up after themselves. Closing the file is as simple as:

```
(close fil)
```

PUTTING IT TOGETHER

Add the code for opening, formatting, writing, and closing to the *lable.lsp* program:

```
(defun c:label ( / xyz xyz1 uprec ptx pty ptz)
  (setq uprec (getint "Label precision: "))
  (setq xyz (getpoint "Pick point: "))
  (setq ptx (car xyz)
        pty (cadr xyz)
        ptz (caddr xyz)
  )

  ; Format the x,y,z coordinates:
  (setq ptx (rtos ptx 2 uprec)
        pty (rtos pty 2 uprec)
        ptz (rtos ptz 2 uprec)
  )

  ; Add commas between the three coordinates:
  (setq xyz1 (strcat ptx ", " pty ", " ptz))

  ; Write coordinates to the drawing:
  (command "text" xyz 200 0 xyz1)

  ; Open the data file for appending:
  (setq fl (open "xyzdata.txt" "a"))

  ; Write the line of data to the file:
  (write-line xyz1 fl)

  ; Close the file:
  (close fl)
)
```

Using a text editor, such as Notepad, make the additions (shown in **boldface** above) to your copy of *lable.lsp*. Load it into BricsCAD with the **load** function:

```
: (load "label")
```

And run the program by entering **Label** at the ':' prompt:

```
: label
Label precision: 4
Pick point: (Pick a point.)
```

As you pick points on the screen, the routine labels the picked points, but also writes the 3D point data to file. After a while, this is what the data file looks something like this:

```
8.1548, 3.2752, 0.0000
7.0856, 4.4883, 0.0000
6.4295, 5.6528, 0.0000
5.5303, 6.7688, 0.0000
5.4331, 8.3215, 0.0000
```

Wishlist #5: Layers

Let's take a moment to revisit the wishlist. One wishlist item is to control the layer on which the text is placed. There are two ways to approach this wishlist item:

- The no-code method is to set the layer before starting the LISP function.
- The LISP-code version is to ask the user for the name of the layer, then use the **setvar** function to set system variable **CLayer** (much easier than using the **Layer** command), as follows:

```
(setq lname (getstring "Label layer: "))
(setvar "CLAYER" lname)
```

Add those two line before the line with the “Pick point” prompt.

Wishlist #6: Text Style

To specify the text style, there are the same two methods. The no-code method is to simply set the text style before starting the routine. Otherwise, you can write LISP code similar to set the style with the **setvar** command, as follows:

```
(setq tsname (getstring "Label text style: "))  
(setvar "TEXTSTYLE" tsname)
```

Once again, add those two line before the line with the “Pick point” prompt.

By now, you may be noticing that your program is starting to look big. This is called “feature bloat.” More features, especially in the area of user interface, makes software grow far beyond the size of its basic algorithm.

TIPS IN USING LISP

To conclude this chapter, here are tips for helping out when you write your LISP functions.

Tip #1. Use an ASCII Text Editor.

LISP code must be written in plain ASCII text — no special characters and no formatting (like **bold** or **color**) of the sort that word processors add to the file. When you write LISP code with, say, Word, then save as a *.doc*-format file (the default), BricsCAD will simply refuse to load the LISP file, even when the file extension is *.lsp*.

In an increasingly Window-ized world, it is harder to find a true ASCII text editor. There is one, however, supplied free by Microsoft with Windows called Notepad, which you’ll find in the *\windows* folder. Do not use Write or WordPad supplied with Windows. While both of these have an option to save in ASCII, you’re bound to forget sometimes and end up frustrated. Linux provides the excellent Text Edit (aka gedit) text editor, while Mac has TextEdit.

Almost any other word processor has an option to save text in plain ASCII, but not by default. Word processors have a number of different terms for what I mean by “pure ASCII format.” Word calls it “Text Only”; WordPerfect calls it “DOS Text”; WordPad calls it “Text Document”; and Atlantis calls it “Text Files.” You get the idea.

Tip #2: Loading LSP Code into BricsCAD

To load the LISP code into BricsCAD, you use the **load** function. Here’s an example where *points*.*lsp* is the name of the LISP routine:

```
: (load "points")
```

You don’t need to type the *.lsp* extension.

When BricsCAD cannot find *points.lsp*, you need to specify the folder name by using either a forward slash or double backslashes — your choice:

```
: (load "\\BricsCAD\\points")
```

After you've typed this a few times, you'll find it gets tedious. To solve the problem, write a one-line LISP routine that reduces the keystrokes, like this:

```
: (defun c:x () (load "points"))
```

Now anytime you need to load the *points.lsp* routine, you just type **X** and press **Enter**, as follows:

```
: x
```

Under Windows, you could also just drag the *.lsp* file from the File Manager into BricsCAD. Note that the code moves one way: from the text editor to BricsCAD; you cannot drag the code from BricsCAD back to the text editor.

Tip #3: Toggling System Variables

One problem in programming is: How to change a value when you don't know what the value is? In BricsCAD, you come across this problem with system variables, many of which are toggles. A *toggle* system variable has a value of 0 or 1, indicating that the value is either off (0) or on (1). For example, system variable **SplFrame** is by default 0: when turned off, splined polylines do not display their frame.

No programmer ever assumes that the value of **SplFrame** is going to be zero just because that's its default value. In the case of toggle system variables, there two solutions:

- (1) Employ the **if** function to see if the value is 0 or 1.
- (2) Subtract 1, and take the absolute value.

Tip #4: Be Neat and Tidy.

Remember, your mother told you to always pick up your things. This problem of setting system variables applies universally. When your LISP routine changes values of system variables, it must always set them back to the way they were before the routine began running.

Many programmers write a set of generic functions that save the current settings at the beginning of the routine, carries out the changes, and then restores the saved values at the end of the routine. Here's a code fragment that shows this, where the original value of **SplFrame** is stored in variable **SplVar** using **getvar**, and then restored with **setvar**:

```
(setq splvar (getvar "splframe"))  
...  
(setvar "splframe" splvar)
```

Tip #5: UPPER vs. lowercase

In (almost) all cases, LISP doesn't care if you use UPPERCASE or lowercase for writing the code. For legibility, there are some conventions:

- ▶ LISP function names in all lowercase.
- ▶ Your function names in Mixed Case.
- ▶ BricsCAD variables and command names in all UPPERCASE.

As I said, LISP doesn't care, and converts everything into uppercase in any case. It also strips out all comments, excess white space, tabs, and return characters. The exception is text in quote marks, such as prompts, which are left as is.

There are two exception where LISP does care: when you are working with escape codes and the letter T.

Escape codes are used in text strings, and must remain lowercase. For example, `\e` is the escape character (equivalent to ASCII 27) and `\t` is the tab character. Note that they use backslashes; it is for this reason that you cannot use the backslash for separating folders names back in Tip #2. LISP would think you were typing an escape code.

And some functions use the letter T as a *flag*. It must remain uppercase.

Tip # 6: Quotation Marks as Quotation Marks

As we have seen, LISP uses quotation marks (") for strings. Thus, you cannot use a quotation mark as for displaying quotation marks and inches, such as displaying 25 inches as 25".

The workaround is to use the escape codes mentioned above in Tip #5, specifically the octal code equivalent for the ASCII character for the quotation mark. Sound complicated? It is. But all you need to know is 042. Here's how it works:

First, assign the strings to variables, as follows:

```
(setq disttxt "The length is ")
(setq distval 25)
(setq qumark "\042")
```

Notice how I assigned octal 042 to variable **qumark**. The backslash tells LISP the numbers following are in octal. Octal, by the way, is half of hexadecimal: 0 1 2 3 4 5 6 7 10 11 12 ... 16 17 20 21 ...

Then concatenate the three strings together with the **strcat** function:

```
(strcat disttxt distval qumark)
```

To produce the prompt:

```
The length is 25"
```


Tip #7: Tabs and Quotation Marks

Vijay Katkar is writing code for a dialog box with a list box. He told me, “I want to display strings in it — just like the dialog box displayed by the **Layer** command. I am able to concatenate the values and print the strings but there is no vertical alignment, since the strings are of different lengths. I tried using the tab metacharacter (`\t`) in the string but it prints the literal `\t` in the list box. Is there any way I can get around this problem?”

I recall a similar problem: How to display quotation marks or the inches symbol within a text string? For example, I have a line of LISP code that I want to print out as:

```
The diameter is 2.54"
```

Normally, I cannot use the quotation (`"`) character in a string. LISP uses the quotation as its string delimiter to mark the beginning and ending of the string. In the following line of code:

```
(prompt "The diameter is 2.54")
```

LISP sees the first quotation mark as the start of the string, the second quotation as the end of the string, and the third quotation mark as an error. The solution is the `\nnn` metacharacter. This lets me insert any ASCII character, including special characters, such as tab, escape, and quotation marks. The workaround here is to use the ASCII code for the quotation mark, `\042`, like this:

```
(prompt "The diameter is 2.54\042")
```

Similarly, Vijay needs to use the `\009` metacharacter to space the text in his dialog box. And, in fact, that worked: “According to what you had told me, I used the same and it worked.”

Designing Dialog Boxes with DCL

DCL allows programmers to create custom dialog boxes for LISP routines. Short for "dialog control language," DCL was added to BricsCAD in V8 for compatibility with AutoCAD.

DCL is a structured language used to describe the elements (called "tiles") that make up dialog boxes. Tiles includes edit boxes, list boxes, radio buttons, image tiles, and title bars. Each of these has one or more attributes, such as its position, background color, and the action it performs.

CHAPTER SUMMARY

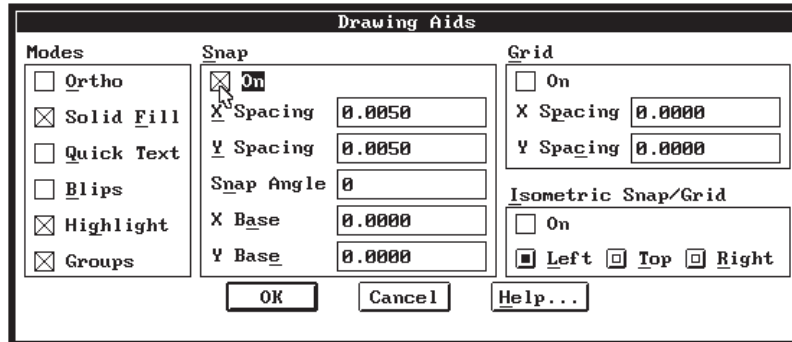
The following topics are covered in this chapter:

- Learning the history of DCL
- Finding out the makeup of dialog boxes
- Coding your first dialog box
- Using LISP code to load and run dialog boxes
- Finding examples of DCL coding
- Debugging DCL
- Discovering additional DCL learning resources

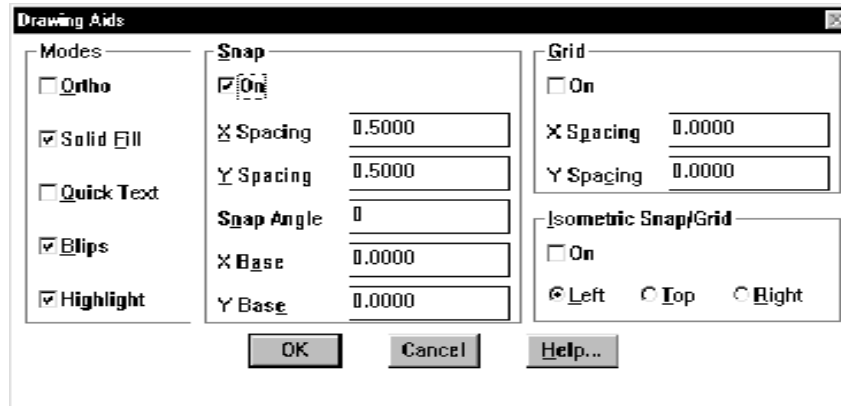
A QUICK HISTORY OF DCL

Autodesk first added DCL (short for "dialog control language") as an undocumented feature to AutoCAD Release 11 for Windows. It was designed for creating platform-independent dialog boxes. At that time, Autodesk produced versions of AutoCAD for "every viable engineering platform," which included DOS, Windows, Unix, Macintosh, and OS/2, and DCL was part of a project code-named "Proteus," whose aim was to make AutoCAD work and look identical on every operating system.

As the figures below show, the project was a success. First, here is AutoCAD Release 11's Drawing Aids dialog box running on DOS:



And here is the same dialog box in the Windows version of AutoCAD Release 11:



Notice how similar the DOS and Windows dialog boxes look. (The Drawing Aids dialog box is now known as the Options dialog box.)

By Release 14, however, Proteus became meaningless, because Autodesk chose to support only the Windows operating system. But DCL continues to hang around as the only way to create dialog boxes with LISP, and Bricsys makes good use of DCL for its support of Linux, MacOS, and Windows.

Applications written in LISP, SDS, and DRx can make use of DCL for dialog boxes. Menu and toolbar macros can too, when they link to LISP routines that call the DCL code. (VBA does not use DCL, because it has its own dialog construction environment.)

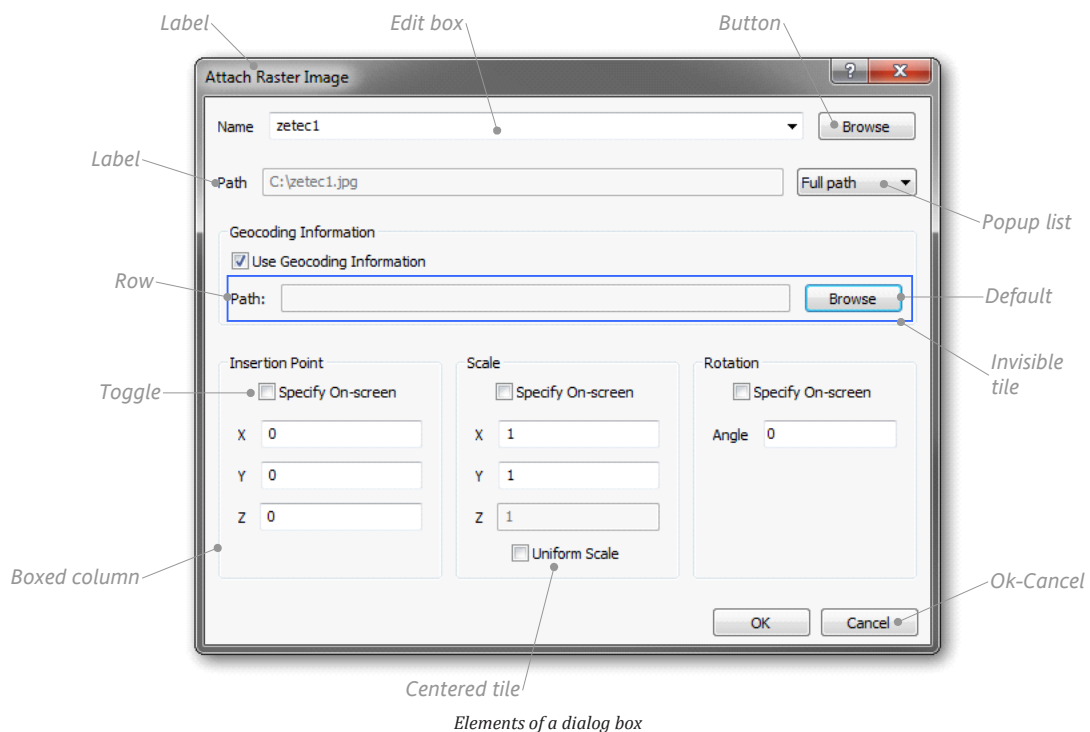
Bricsys provides no programming environment to help you create DCL files — it's hand coding all the way. That means a text editor such as NotePad in Windows and Text Edit in Linux or Mac will be your DCL programming environment. Some third-party developers have created DCL development tools.

When you want a LISP routine to display a dialog box, you need to write two pieces of code:

- ▶ Code in a *.dcl* file that defines the dialog box and the functions of its tiles.
- ▶ Code in the *.lsp* file that loads the *.dcl* file, and then activates the tiles.

Working with dialog boxes always involves a pair of files, *.dcl* and *.lsp*, with the LISP code controlling the dialog box code.

A drawback to DCL is that it cannot create self-modify dialog boxes, such as ones that add or remove buttons. It can, however, dynamically change the contents of droplists and such.



What Dialog Boxes Are Made Of

Dialog boxes can consist of many elements, such as radio buttons, sliders, images, tabs, and check boxes. These elements are called "tiles." DCL allows you to create many different types of elements, but it does not have tiles for every element found in today's dialog boxes. That's because DCL hasn't been upgraded since it was introduced some 20 years ago. (Those elements not possible with DCL can be created through VBA.)

The figure below illustrates many of the dialog box elements that are possible with DCL, along with some names of specific DCL tiles.

Most tiles are visible, but some are invisible, such as the row and column tiles highlighted in the figure above by blue rectangles:

HOW DCL OPERATES

The two pieces of code that are required to make dialog boxes operate are (a) DCL code that specifies the layout of tiles and their attributes in the dialog box, and (b) LISP code that activates and controls the dialog box.

You do not need to specify the overall size of the dialog box; BricsCAD takes care of that by automatically sizing it. The default is that tiles are stacked in columns; you only need to specify when tiles should be aligned in a row.

Some back and forth is permitted while running DCL and LISP; this is known as "callbacks." Callbacks are used to provide names to file dialog boxes, to gray out certain buttons, to change the content of popup lists (droplists), and so on.

This chapter shows you how to write DCL with LISP code. Appendix B provides you with a comprehensive reference to all DCL tiles, their attributes, and related LISP functions.

Your First DCL File

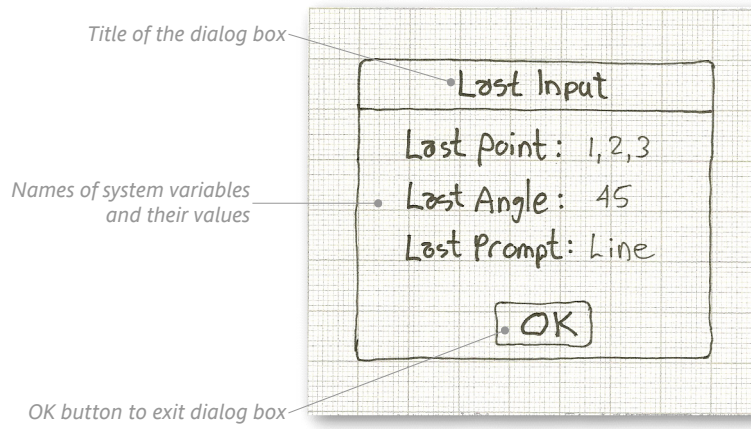
Before writing any code for a dialog box, it is helpful to plan out the tiles. Where will the buttons, droplists, and text entry boxes go in the dialog box? It's a good thing to get your pencil, and then sketch your ideas on paper.

For this tutorial, you will create a dialog box that displays the values stored in these system variables:

- LastPoint** — stores the last 3D point entered in the drawing.
- LastAngle** — stores the angle defined by the last two points entered.
- LastPrompt** — stores the last text entered at the command line.

Take a moment to think about the design of the dialog box. It would have a title that explains the purpose of the dialog box. It probably should have three lines of text that report the name and value of each system variable. And it should have an **OK** button to exit the dialog box.

It might look like this:



Sketching the new dialog box

DCL PROGRAMMING STRUCTURE

The programming structure of this dialog box looks like this:

Start the dialog box definition:

- Specify the dialog box's title
- Specify a column:
 - System variable LastPoint and its 3D coordinates
 - System variable LastAngle and its angle
 - System variable LastPrompt and its text
- Locate the OK button

End the dialog box definition.

In this first tutorial, you will write just enough code to display the dialog box and its OK button. In the tutorials that come later, you add the bells and whistles.

Start Dialog Box Definition

The content of every *.dcl* file begins with a name *attribute*. This is the name by which the dialog code is called later by the associated LISP routine. The name function looks like this:

```
name: dialog {
```

Like LISP, an open brace needs a closing brace to signal the end of a dialog box definition:

```
}
```

Between the two braces you write all the code that defines the look of the dialog box.

For this tutorial, name the dialog box "lastInput," as follows:

```
lastInput: dialog {  
}
```

DCL names are case-sensitive, so "lastInput" is not the same as "LastINPUT" or "lastinput."

Dialog Box Title

The text for the dialog box's title bar is specified by the **label** property, as follows:

```
name: dialog {  
    label = "Dialog box title";  
}
```

Label this dialog box "Last Input" like this:

```
lastInput: dialog {  
    label = "Last Input";  
}
```

The title text needs to be surrounded by quotation marks ("). The label property must be terminated with a semicolon (;). And it's helpful to indent the code to make it readable.

OK Button

Every dialog box needs an exit button, at least an OK. (Windows places a default **X** button in the upper-right corner of every dialog box, which also works to exit dialog boxes made with DCL!)

Buttons are defined with the **button** property, followed by the properties of the button enclosed in braces:

```
: button {  
}
```

Because dialog boxes can have multiple buttons, every button must be identified by a property called the "key." The *key* is how LISP gives instructions to buttons. Use the **key** attribute to identify

QUICK SUMMARY OF DCL METACHARACTERS

DCL Metacharacter	Meaning
//	(slash-slash) Indicates a comment line.
/*	(slash-asterisk) Starts comment section.
*/	(asterisk-slash) Ends comment section.
:	(colon) Starts a tile definition. Predefined tiles, like spacer, do not use the colon.
{	(brace) Starts dialog and tile attributes.
	(space) Separates symbols.
=	(equals) Defines attribute values.
"	(straight quotation) Encloses text attributes.
;	(semi-colon) Ends attribute definition. Every attribute must end with a semi-colon.
}	(brace) Ends tile and dialog attributes.

this OK button as "okButton," as follows:

```
key = "okButton";
```

The button needs to display a label for users to read. This is an OK button, so label it " OK " with the **label** attribute, as follows:

```
label = " OK ";
```

Let's put all of these together. The code added for the OK button is shown here in **color**, with the key, label, and is_default attributes. (See below for info about the default attribute.) We have a button identified as "okButton," sporting the label "OK," and is set at the default tile.

```
lastInput: dialog {
  label = "Last Input";
  : button {
    key = "okButton";
    label = " OK ";
    is_default = true;
  }
}
```

TIP The DCL code for the **OK** button is like a *subroutine*. The same code can be reused any time a dialog box needs an **OK** button, which is pretty much all the time. Later, you will see how to create subroutines with DCL code.

The Default Tile

To make life easier for users, one tile of a dialog box is always made the *default* tile. Users need only press **Enter** to activate the default tile. Dialog boxes highlight the default tile in some way, such as with a dashed or colored outline. User can press **Tab** to move the default *focus* (currently highlighted tile) to other areas of the dialog box.

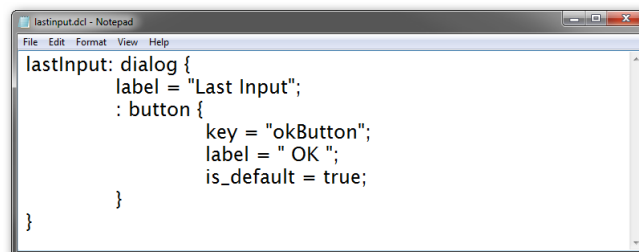
A tile is made the default with the **is_default** attribute, as follows:

```
is_default = true;
```

Testing DCL Code

You have enough DCL code to test it now, which lets you see how the dialog box is developing. To test the code, take these steps:

1. Open Notepad, Text Edit, or another ASCII text editor.
2. Enter the DCL code we developed earlier:



Entering DCL code into a text editor

LISP CODE TO LOAD AND RUN DIALOG BOXES

The following LISP code is what you use to load, run, and exit the lastInput.dcl dialog box definition file:

```
(defun C:xx ()
  (setq dlg-id (load_dialog "c:\\lastInput"))
  (new_dialog "lastInput" dlg-id)
  (action_tile "accept" "(done_dialog)")
  (start_dialog)
  (unload_dialog dlg-id)
)
```

To see what the LISP code means, let's take it apart.

The function is defined as "xx" with LISP's defun function. Programming = debugging, so I like to use an easy-to-enter name for the LISP routine, like "xx."

```
(defun C:xx ()
```

The lastInput.dcl file is loaded with the load_dialog function. There is no need to specify the ".dcl" extension, because this is the sole purpose of this function: to load DCL files.

- In Windows, include the name of the drive, C:\\. Recall that LISP requires you to use \\ instead of \ for separating folder names.

```
(setq dlg-id (load_dialog "c:\\lastInput"))
```

- In Linux, leave out the name of the drive:

```
(setq dlg-id (load_dialog "lastInput"))
```

DCL files can contain more than one dialog box definition, and so the next step is to use the new_dialog function to tell BricsCAD which one you want to access. In this case, there is just the one, "lastInput."

```
(new_dialog "lastInput" dlg-id)
```

The dialog box contains a button named "okButton," and its purpose is defined by LISP — not DCL! Here you use the action_tile function to assign an action to the "okButton" tile. The button's purpose in life is to execute the done_dialog function that exits the dialog box. In short, click OK to exit the dialog box. You can read this as follows: "the action for the tile named okButton is ...".

```
(action_tile "okButton" "(done_dialog)")
```

After all these preliminaries, the big moment arrives. The start_dialog function launches the dialog box, and waits then for you to click its button.

```
(start_dialog)
```

As neat programmers, we unload the dialog box from memory with the unload_dialog function.

```
(unload_dialog dlg-id)
```

And a final parenthesis ends the xx function.

```
)
```

Important: Ensure that this DCL file and the LSP file use straight quotation marks that look like this: ". If they contain curly quotation marks (" or "), the routines will fail. LISP will complain, "error: bad argument type <NIL>; expected <STRING>" while DCL will put up a dialog box complaining, *syntax error: unexpected ""*.

- Save the file as *lastinput.dcl*. So that the LISP *xx.lsp* routine can find it easily, save the file in a top level folder:
 - In Windows, save the DCL file to the C:\ drive.
 - In Linux and Mac, save the DCL file to your home folder. For example, I log in to Linux with the user name of "ralphg," so I saved the file in the *ralphg* folder.
- Now, open a new file, and then enter the LISP code described in the boxed text on the following page: "LISP Code to Load and Run Dialog Boxes."

```

xx.lsp - Notepad
File Edit Format View Help
(defun C:xx ()
  (setq dlg-id (load_dialog "c:\\lastInput"))
  (new_dialog "lastInput" dlg-id)
  (action_tile "accept" "(done_dialog)")
  (start_dialog)
  (unload_dialog dlg-id)
)

```

Entering LISP code into a text editor

- Save the file as *xx.lsp*, also in the same folder as the DCL file.
- Switch to BricsCAD, and then open the *xx.lsp* file in BricsCAD:
 - In Windows, use Explorer to drag the *xx.lsp* file into the BricsCAD drawing window. (Dragging the file is a lot faster than entering the **AppLoad** command or using the LISP **load** function!)
 - In Linux, you have to use the **load** function, because files cannot be dragged into the Linux version of BricsCAD. Enter the following at the ':' prompt:
: (load "xx")
- Type **xx** to run the routine, which then loads the dialog box:
: **xx**

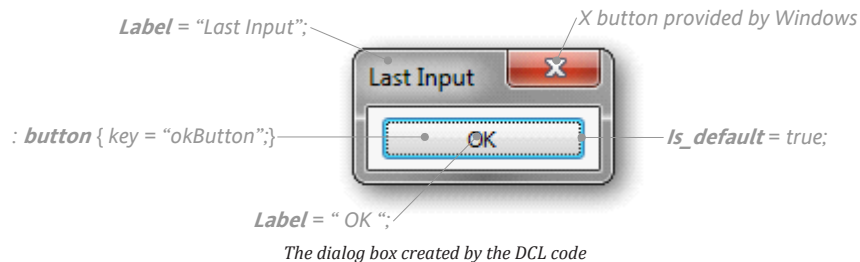
Notice that the dialog box appears! Well, it should, if you haven't made any coding errors.



Left: Dialog box in Windows 7.
Right: Dialog box in Linux Mint.

- Click **OK** to exit the dialog box.

Here is a map of how the DCL code created the dialog box:



DISPLAYING DATA FROM SYSTEM VARIABLES

The basic structure of the dialog box is in place: the label and the OK button. Now it is time to add the data we want displayed by the system variables.

The data from the sysvars will look like this in the dialog box:

```
Last angle:    45
Last point:    1,2,3
Last prompt:   Line
```

I show the *static* text in **color**. It never changes. This text acts like a prompt to tell users what the numbers mean.

The black text is *variable*; its display changes, and depends on the value of the associated sysvar.

The **Text** tile is the one that displays text in dialog boxes, and its code will look like this:

```
: text {
    label = "Last angle: ";
    key = "lastAngle";
}
```

Are you able to recognize the attributes of this text tile?

Begin the text with this tile:

```
: text {
```

Next, the **label** attribute provides the prompt, 'Last angle: '.

```
label = "Last angle: ";
```

The **key** attribute identifies the text tile as "lastAngle."

```
key = "lastAngle";
```

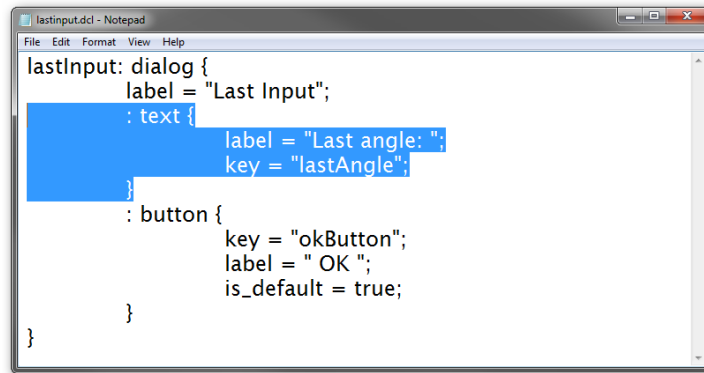
Finally, the text tile is closed with the brace.

```
}
```

TIP Text tiles can have the following attributes, as described fully by the DCL reference later in this ebook:

- alignment
 - fixed_height
 - fixed_width
 - height
 - is_bold
 - key
 - label
 - value
 - width
-

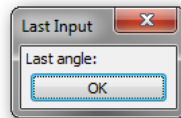
Add the highlighted code to the DCL file...



```
lastInput: dialog {
  label = "Last Input";
  : text {
    label = "Last angle: ";
    key = "lastAngle";
  }
  : button {
    key = "okButton";
    label = " OK ";
    is_default = true;
  }
}
```

Adding DCL code in the text editor

...and then run the `xx.lsp` routine again. Notice that the dialog box now displays the 'Last angle:' text:



Dialog box that results from the DCL added code

The next step is to display the value stored by the `LastAngle` system variable. Add a second text tile:

```
: text {
  value = "";
  key = "lastAngleData";
}
```

The **value** of this tile is initially blank, because it has no label and no value. To complete the text tile, we need to use a LISP function that extracts the value from the **LastAngle** system variable, and then shoves it into the dialog box.

The link between the LISP code and the DCL file is with the **key**, which is named here "lastAngle-Data." (I'll show you the LISP code a bit later on.) Now the DCL file looks like this, with the new code shown in **color**. You can copy this code and paste it into the text editor.

```
lastInput: dialog {
  label = "Last Input";
  : text {
    label = "Last angle: ";
    key = "lastAngle";
  }
  : text {
    value = "";
    key = "lastAngleData";
  }
  : button {
    key = "okButton";
    label = "OK";
    is_default = true;
  }
}
```

(If you were to run this DCL code now, the dialog box would look no different. It still needs LISP to tell it the value of the last angle. This is coming up next.)

ADDING THE COMPLIMENTARY LISP CODE

Writing DCL code is always only half the job. The other half is to write the complementary code in LISP. Extracting the value from LastAngle take these two steps:

Step 1: Use the **getvar** function to access the value of sysvar LastAngle, and then store the gotten value in variable *lang* (short for "last angle") with the **setq** function, as follows:

```
(setq lang (getvar "LastAngle"))
```

Step 2: Use the **set_tile** function to set the value of *lang* to the "lastAngleData" tile:

```
(set_tile "lastAngleData" (rtos lang 2 2))
```

TIP Tiles work only with text, no numbers. However, the value of **LastAngle** is a number, so you have to convert it to text. This is done with the **rtos** function:

```
(rtos lang 2 2))
```

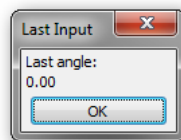
Here, I am converting the real number to a string (a.k.a. text) using mode 2 (decimal) and precision 2 (two decimal places).

With the new lines of code shown in **color**, the LSP file now looks like this:

```
(defun C:xx ()
  (setq dlg-id (load_dialog "c:\\lastInput"))
  (new_dialog "lastInput" dlg-id)
  (setq lang (getvar "lastangle"))
  (set_tile "lastAngleData" (rtos lang 2 2))
  (action_tile "okButton" "(done_dialog)")
  (start_dialog)
  (unload_dialog dlg-id)
)
```

Save the *.dcl* and *.lsp* files, and then reload and run *xx.lsp* in BricsCAD.

The dialog box now looks like this:



Dialog box reporting the angle value

CLUSTERING TEXT

Hmmm... the two pieces of text are stacked on top of one another, and that is a problem. They should be horizontal. The text is stacked vertically, because DCL places tiles in *columns* by default.

The solution is to force the two text tiles to appear next to each other with the **Row** tile:

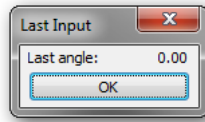
```
: row {
  : text {
    label = "Last angle: ";
    key = "lastAngle";
  }
}
```

```

: text {
    value = "";
    key = "lastAngleData";
}
}

```

Modify the DCL file by adding the row tile, and then rerun the LISP file. The result should look better, like this:



Angle text formatted into a single line

Now that the last-angle text looks proper, you can copy and paste its code for use by the other two lines, and then make suitable modifications. The changes you need to make are shown below in color:

```

: row {
  : text {
    label = "Last point: ";
    key = "lastPoint";
  }
  : text {
    value = "";
    key = "lastPointData";
  }
}

```

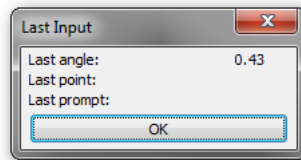
And for final prompt:

```

: row {
  : text {
    label = "Last prompt: ";
    key = "lastPrompt";
  }
  : text {
    value = "";
    key = "lastPromptData";
  }
}

```

Running `xx.lsp` gives the dialog box all three prompts, but data is missing from the two new ones:



Added lines of info

Supplying the Variable Text

The data is supplied by LISP code. Here we look at how to handle 3D coordinates and text.

Recall that LISP returns the value of points as list of three numbers, like this:

```
(1.0000 2.0000 3.0000)
```

The numbers represent the x, y, and z coordinates, respectively. We need to convert the list of three numbers to a string — why does it have to be so hard?! Use the following code, which assumes that variable *lpt* contains (1.0000 2.0000 3.0000):

```
(car lpt)
```

The **car** function extracts the x-coordinate from the list as a real number, such as 1.0000. Similarly:

```
(cadr lpt)
```

```
(caddr lpt)
```

The **cadr** and **caddr** functions extract the y (2.0000) and z (3.0000) coordinates, respectively. To convert the real numbers to strings, use the **rtos** function, as follows:

```
(rtos (car lpt))
```

```
(rtos (cadr lpt))
```

```
(rtos (caddr lpt))
```

And then to combine the three individual strings into one string, use the **strcat** (string concatenation) function, as follows:

```
(strcat  
  (rtos (car lpt))  
  (rtos (cadr lpt))  
  (rtos (caddr lpt))  
)
```

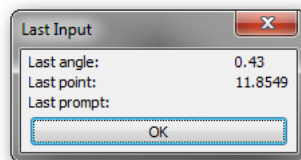
This code displays 1.000 2.000 3.000. It would be a nice touch to put commas between the numbers:

```
(strcat  
  (rtos (car lpt)) ","  
  (rtos (cadr lpt)) ","  
  (rtos (caddr lpt))  
)
```

Put both lines of code together, and we arrive at the LISP needed to implant the value of the Last-Point system variable in the dialog box:

```
(setq lpt (getvar "lastpoint"))  
(set_tile "lastPointData" (strcat (rtos (car lpt)) "," (rtos (cadr lpt)) "," (rtos  
  (caddr lpt))))
```

Add the code to the *xx.lsp*, and then run it in BricsCAD to see the result.



Adding the last point data

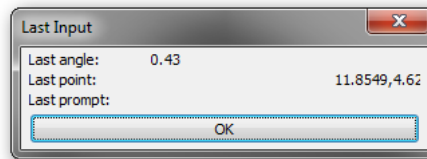
Leaving Room for Variable Text

Oops, the numbers are cut off. BricsCAD sizes the dialog box is sized before the LISP code inserts the data, so it doesn't know that the dialog box needs to be bigger to accommodate the x,y,z coordinates — which can run to many characters in length.

The solution is to use the **width** attribute for each text tiles, like this:

```
: text {  
    value = "";  
    key = "lastAngleData";  
    width = 33;  
}
```

When added to the DCL file, the result looks like this:



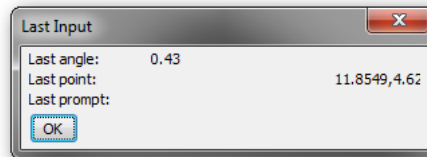
Adding last x,y point data

FIXING THE BUTTON WIDTH

Oops. Now the OK button is too wide. To make it narrower (i.e., fix its width), use the **fixed_width** attribute in the DCL file:

```
fixed_width = true;
```

By setting it to true, the button is made only as wide as the label.



More changes

Centering the Button

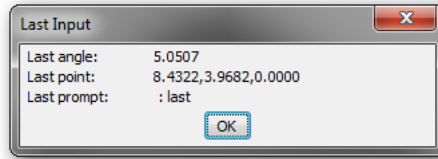
Oops! Now the button is no longer centered. By default, the button is left-justified. To center it, use the **alignment** attribute:

```
alignment = centered;
```

Add the new code to the button portion of the DCL file...

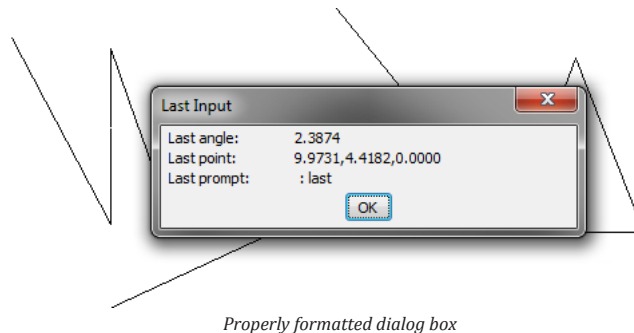
```
: button {  
    key = "okButton";  
    label = "OK";  
    is_default = true;  
    alignment = centered;  
    fixed_width = true;  
}
```

...and then rerun the `xx.lsp` file to see that the properly-sized OK button is centered.



TESTING THE DIALOG BOX

It's always a good idea to test the dialog box under a number of situations. Use the **Line** command to draw a few lines. This action changes the values of the three sysvars. Re-run the `xx.lsp` routine. The values displayed by the dialog box should be different.



Defining the Command

So far, you've been running `xx.lsp` to develop and test the dialog box. Now that it's running properly, you should change the "xx" name to one that is more descriptive. Rename the LISP file as `last.lsp`, and change the function name inside to **C:last**, and make the variables local, as follows:

```
(defun c:last (/ dlg-id lang lpt lcmd)
  (setq dlg-id (load_dialog "c:\\lastInput"))
  (new_dialog "lastInput" dlg-id)
  (setq lang (getvar "lastangle"))
  (set_tile "lastAngleData" (rtos lang))
  (setq lpt (getvar "lastpoint"))
  (set_tile "lastPointData" (strcat (rtos (car lpt)) "," (rtos (cadr lpt)) ","
  (rtos (caddr lpt))))
  (setq lcmd (getvar "lastprompt"))
  (set_tile "lastPromptData" lcmd)
  (action_tile "okButton" "(done_dialog)")
  (start_dialog)
  (unload_dialog dlg-id)
)
```

The DCL file looks like this, in its entirety:

```
lastInput: dialog {
  label = "Last Input";
  : row {
    : text {
      label = "Last angle: ";
      key = "lastAngle";
    }
    : text {
      value = "";
    }
  }
}
```

```

        key = "lastAngleData";
        width = 33;
    }
}
: row {
    : text {
        label = "Last point: ";
        key = "lastPoint";
    }
    : text {
        value = "";
        key = "lastPointData";
        width = 33;
    }
}
: row {
    : text {
        label = "Last prompt: ";
        key = "lastPrompt";
    }
    : text {
        value = "";
        key = "lastPromptData";
        width = 33;
    }
}
: button {
    key = "okButton";
    label = "OK";
    is_default = true;
    alignment = centered;
    fixed_width = true;
}
}

```

If you would like to have this command loaded automatically each time you start BricsCAD, add *last.lsp* to the **AppLoad** command's startup list.

Examples of DCL Tiles

With the basic tutorial behind you, let's take a look at how to code other types of dialog box features. In this last part of the chapter, we look at how to code the following tiles:

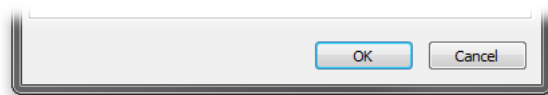
- Buttons
- Check boxes (toggles)
- Radio buttons
- Clusters (columns and rows)

Recall that two pieces of code are always required: (1) the DCL code that specifies the layout of the dialog box, and (2) the LISP code that activates the dialog box.

Appendix B provides you with a comprehensive reference to all DCL tiles, their attributes, and related LISP functions.

BUTTONS

In the preceding tutorial, you coded an OK button that allowed you to exit the dialog box. It turns out that you don't need to do the coding, because BricsCAD codes a number of buttons and other dialog box elements for you. These are found in a file called *base.dcl* that is normally loaded into BricsCAD automatically. (The full list is provided in the appendix).



The names of the pre-built tiles are:

Prebuilt Tile	Button(s) Displayed
ok_only	OK
ok_cancel	OK Cancel
ok_cancel_help	OK Cancel Help
ok_cancel_help_info	OK Cancel Help Info...
Ok_Cancel_Help_Errtile	OK Cancel Help, <i>plus space for error messages.</i>

Use these prebuilt tiles to ensure a consistent look for your dialog boxes. Here is an example of how to use these buttons in DCL files:

```
ok_only;
```

It's just that easy!

Notice that the tile name lacks the traditional colon (:) prefix, but does require the semicolon (;) terminator.

DCL allows you to create buttons that have labels made of text (**button** tiles) or images (**image_button** tiles).

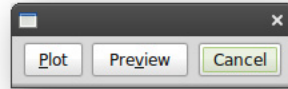
To indicate that the button opens another dialog box, use an ellipsis (...), such as **Info...**

In addition to text and image buttons, settings can be changed with check boxes (**toggle tiles**) and radio buttons (**radio_button tiles**), as described next.

Making Buttons Work

OK and **Cancel** are easy, because their functions are already defined. It's one thing to populate a dialog box with buttons; it's another to have them execute commands.

Let's see how to make buttons execute commands. In the tutorial, you create a dialog box with **Plot** and **Preview** buttons. The figure below shows how it will look in Linux; it looks similar in Windows.



The purpose of the Plot button is to execute the **Plot** command, and of Preview button to execute the **Preview** command.

The easy solution would be to add an **action** attribute to each button to execute a LISP function, such as (**command "plot"**). But we cannot, because DCL does not allow the highly-useful **command** function to be used in the action attribute!

The key to solving the problem is the **key** attribute. It gives buttons identifying names by which LISP functions can reference them, such as:

```
key = "plot";
```

Then, over in the LISP file, you use the **action_tile** function to execute the Plot command. Well, not quite. It has the same restriction against use of the **command** function, so you must approach this indirectly by getting **action_tile** to refer to a *second* LISP routine, such as (**action_tile "plot" "(cmd-plot)"**).

But even this will not work, because you need your custom dialog box to disappear from the screen, and be replaced by the Plot dialog box. The solution is to become even more indirect:

```
(action_tile "plot" "(setq nextDlg 1) (done_dialog)")
```

"plot" — identifies the Plot button through its key, "plot".

(setq nextDlg 1) — records that the user clicked the Plot button for further processing later on.

(done_dialog) — closes the dialog box.

This is done twice, once each when the user clicks the Plot button or the Preview button. The Preview button's code is similar; changes are shown in boldface:

```
(action_tile "preview" "(setq nextDlg 2) (done_dialog)")
```

Then, you need some code that decides what to do when **nextDlg** is set to 1 or 2:

```
(if (= nextDlg 1) (cmd-plot))
```

```
(if (= nextDlg 2) (cmd-preview))
```

When nextDlg = 1, then the following subroutine is run:

```
(defun cmd-plot ()
  (command "print")
)
```

The purpose of the **Print** command is to force BricsCAD to display the dialog box of the Plot command; otherwise, the prompts are displayed at the command line.

If you prefer the prompts at the command line, then change the code:

```
(defun cmd-plot ()
  (command "plot")
)
```

When nextDlg = 2, then the following subroutine is run instead:

```
(defun cmd-preview ()
  (command "preview")
)
```

With the planning behind us, let's look at *all* the code. First, in the *x.dcl* file, you add the key attributes to each button. The code that relates to the Plot button is shown boldface, while Preview-related code is shown in color:

```
x: dialog { label = "Plot";
: row {
  : button { label = "Plot"; mnemonic = "P"; key = plot; }
  : button { label = "Preview"; mnemonic = "v"; key = "preview"; }
  cancel_button;
} }
```

Second, in the **xx.lsp** file, you add the code that executes the Plot and Preview commands.

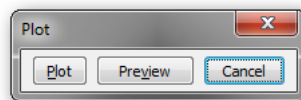
```
(defun c:xx (/)
  (setq dlg-id (load_dialog "c:\\x"))
  (new_dialog "x" dlg-id)
  (action_tile "plot" "(setq nextDlg 1) (done_dialog)")
  (action_tile "preview" "(setq nextDlg 2) (done_dialog)")
  (start_dialog)
  (unload_dialog dlg-id)
  (if (= nextDlg 1) (cmd-plot))
  (if (= nextDlg 2) (cmd-preview))
)

(defun cmd-plot ()
  (command "print")
)

(defun cmd-preview ()
  (command "preview")
)
```

In Linux, remember to remove the "c:\\\" so that the load_dialog line reads as follows:

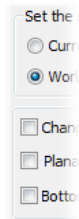
```
(setq dlg-id (load_dialog "x"))
```



When the dialog box appears, click each button to ensure it executes the related command.

Check Boxes

Check boxes allow you to have one or more options turned on. They contrast to *radio buttons*, which limit you to a single choice. Check boxes are created by the **toggle** tile.



Top: Radio buttons made with the radio tile.
Above: Check boxes made with the toggle tile

In this tutorial, you create a check box that changes the shape of point objects. This is accomplished by changing the value of the **PdMode** system variable. Yes, there is the **DdPType** command that does the same thing, but this is a difference approach, as you will see.

The PdMode system variable can take these values:

PdMode	Meaning
0	Dot (.)
1	Nothing
2	Plus (+)
3	Cross (x)
4	Short vertical line ()
32	Circle
64	Square

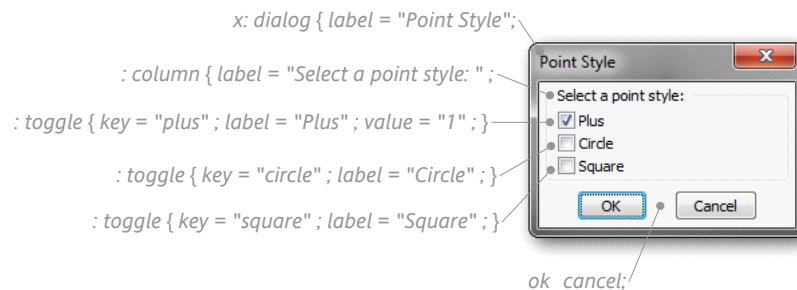
In addition, these numbers can be combined through addition. For example, 34 ($32 + 2$) adds a circle (32) to the plus symbol (2).



Left to right: PdMode = 32, 33, and 34.

Here is a peculiarity to points to be aware of: 32 actually a circle with dot ($32 + 0$), because 0 draws a dot. In comparison, 33 ($32 + 1$) is the circle alone, because the 1 displays nothing!

Let's see how to create a dialog box that lets us select combinations of the plus, circle, and square point symbols. How about a dialog box that looks something like this...



Here is the code needed to generate the dialog box:

```
x: dialog { label = "Point Style";
  : column { label = "Select a point style: " ;
    : toggle { key = "plus" ; label = "Plus" ; value = "1" ; }
    : toggle { key = "circle" ; label = "Circle" ; }
    : toggle { key = "square" ; label = "Square" ; }
  }
  ok_cancel;
}
```

Notice that **value = "1"** turns on the Plus option (to show the check mark), making it the default value.

Now let's write the LISP file to make the dialog box work. Something as simple as (action_tile "plus" "(setvar "pdmode" 2)") doesn't work, because the user might want to select more than one option — which is the whole point to toggles. You need the code to go through three steps:

Step 1: Read which option(s) users have checked.

Step 2: Add up the setting(s).

Step 3: Set **PdMode** to show the desired point style.

Let's implement it:

1. To read user input from dialog boxes, employ LISP's **\$value** variable for the Plus toggle:
(action_tile "plus" "(setq plusVar \$value)")

Repeat the code for the other two toggles, Circle and Square:

```
(action_tile "circle" "(setq circleVar $value)")
(action_tile "square" (setq squareVar $value))
```

2. The **\$value** variable contains just 1s and 0s. Later, we will use a lookup table to convert the 1s and 0s into the values expected by **PdMode**. For instance, if Plus is selected ("1"), then PdMode expects a value of 2. The lookup table uses the **if** function to correct the numbers, as follows:

```
(if (= plusVar "1") (setq plusNum 2) (setq plusNum 0))
```

This can be read as:

```
If plusVar = 1, then set plusNum = 2;
otherwise, set plusNum = 0.
```

Repeat the lookup code for the other two toggles, Circle and Square:

```
(if (= squareVar "1") (setq squareNum 64) (setq squareNum 0))
(if (= circleVar "1") (setq circleNum 32) (setq circleNum 0))
```

TIP The \$value retrieved by **get_tile** is actually a string, like "1". The **PdMode** system variable, however, expects an integer. Thus, the lookup table performs a secondary function of converting strings to integers.

With the values set to what PdMode expects, add them up with the **+** function:

```
(setq vars (+ plusNum circleNum squareNum))
```

3. To change the value of **PdMode**, you employ LISP's **setvar** function, like this:
(setvar "pdmode" vars)

Here is all of the LISP code:

```
(defun c:xx (/)
  (setq dlg-id (load_dialog "c:\\x"))
  (new_dialog "x" dlg-id)

  ;; Get the current values from each toggle tile:

  (setq plusVar (get_tile "plus"))
  (setq circleVar (get_tile "circle"))
  (setq squareVar (get_tile "square"))

  ;; See which toggles the user clicks:

  (action_tile "plus" "(setq plusVar $value)")
  (action_tile "circle" "(setq circleVar $value)")
  (action_tile "square" "(setq squareVar $value)")

  (start_dialog)
  (unload_dialog dlg-id)

  ;; Lookup table converts "0"/"1" strings to the correct integers:

  (if (= plusVar "1") (setq plusNum 2) (setq plusNum 0))
  (if (= circleVar "1") (setq circleNum 32) (setq circleNum 0))
  (if (= squareVar "1") (setq squareNum 64) (setq squareNum 0))

  ;; Add up the integers, and then change system variable:

  (setq vars (+ plusNum circleNum squareNum))
  (setvar "pdmode" vars)
)
```

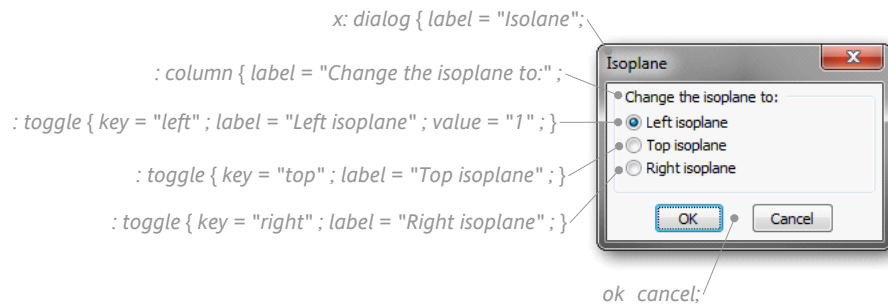
Radio Buttons

Radio buttons are easier to code than toggles, because only one can be active at a time.

In this tutorial, you create a dialog box that uses radio buttons to change the isoplane. The dialog box changes the value of the **SnapIsoPair** system variable, which takes the following values:

SnapIsoPair	Meaning
0	Left isoplane (default).
1	Top isoplane.
2	Right isoplane.

To make a dialog box that looks like this...



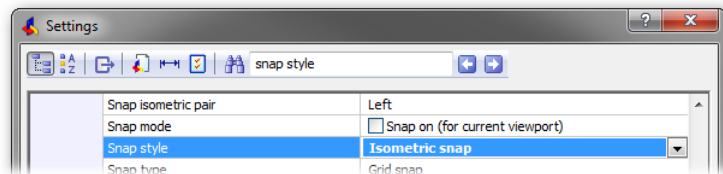
... takes this code:

```
x: dialog { label = "Isoplane";
  : column { label = "Change the isoplane to: " ;
    : radio_button { key = "left" ; label = "Left isoplane" ; value = "1" ; }
    : radio_button { key = "top" ; label = "Top isoplane" ; }
    : radio_button { key = "right" ; label = "Right isoplane" ; }
    spacer;
  }
  ok_cancel;
}
```

Notice that **value = "1"** turns turns on the X for the check box next to Left.

Before going on to the accompanying LISP file, first set up BricsCAD to display isometric mode:

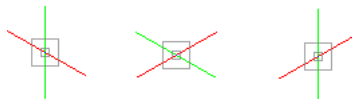
1. Enter the **Settings** command.
2. In the Search field, enter "Snap Style."



3. In the Snap Type droplist, select **Isometric snap**.
4. Click **X** to dismiss the dialog box.

BricsCAD is now in isometric mode.

As you use the dialog box described below, the cursor changes its orientation:



Left to right: Cursor for the left, top, and right isoplanes.

Let's now turn to the LISP file that will make this dialog box work. It is similar to the code used for toggles; the primary difference is that values are not added together:

```
(defun c:xx (/)
  (setq dlg-id (load_dialog "c:\\x"))
  (new_dialog "x" dlg-id)

  ;; See which radio button the user clicks:

  (action_tile "left" "(setq leftVar $value)")
  (action_tile "top" "(setq topVar $value)")
  (action_tile "right" "(setq rightVar $value)")

  (start_dialog)
  (unload_dialog dlg-id)

  ;; Lookup table:

  (if (= leftVar "1") (setq vars 0))
  (if (= topVar "1") (setq vars 1))
  (if (= rightVar "1") (setq vars 2))

  ;; Change system variable:

  (setvar "snapisopair" vars)
)
```

We have been cheating a bit, because we are forcing the dialog box to show the Left isoplane as the default. This is not necessarily true. You really should modify the DCL and LISP code to make the dialog box initially show the default isoplane — whether left, top, or right.

Setting the default is done with LISP's **set_tile** function. First, change the DCL code so that it no longer makes the Left isoplane the default: change *value = "1"* to:

```
value = ""
```

In the LISP code, you need to do the following: (a) extract the value of SnapIsoPair with **getvar**, and then (b) use **set_tile** as a callback.

1. Extract the current value of **SnapIsoPair** with the **getvar** function:

```
(setq vars (getvar "snapisopair"))
```
2. Set the default button with the **set_tile** function:

```
(if (= vars 0) (set_tile "left" "1"))
```

This reads, as follows:

```
If the value of SnapIsoPair is 0 (= vars 0),  
then turn on the Left isoplane radio button (set_tile "left" "1").
```

Write similar code for the other two radio buttons:

```
(if (= vars 1) (set_tile "top" "1"))  
(if (= vars 2) (set_tile "right" "1"))
```

The other change you need to make is to change some of the variables to local:

```
(defun c:xx (/ leftVar topVar rightVar)
```

This forces the three variables to lose their value when the LISP routine ends. Otherwise, **rightVar** keeps its value (it's the last one) and makes Right isoplane the default each time the dialog box is opened.

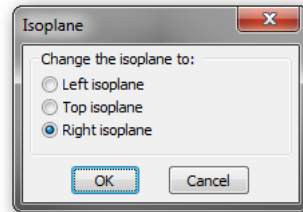
With these changes in place, the improved code looks like this — with changes highlighted in boldface:

```
(defun c:xx (/ leftVar topVar rightVar)  
  (setq vars (getvar "snapisopair"))  
  (setq dlg-id (load_dialog "c:\\x"))  
  (new_dialog "x" dlg-id)  
  
  ;; Set the default button:  
  
  (if (= vars 0) (set_tile "left" "1"))  
  (if (= vars 1) (set_tile "top" "1"))  
  (if (= vars 2) (set_tile "right" "1"))  
  
  ;; See which radio button the user clicks:  
  
  (action_tile "left" "(setq leftVar $value)")  
  (action_tile "top" "(setq topVar $value)")  
  (action_tile "right" "(setq rightVar $value)")  
  
  (start_dialog)  
  (unload_dialog dlg-id)
```

```
;; Lookup table:
      (if (= leftVar "1") (setq vars 0))
      (if (= topVar "1") (setq vars 1))
      (if (= rightVar "1") (setq vars 2))

;; Change system variable:
      (setvar "snapisopair" vars)
    )
```

Now each time the dialog box starts, it correctly displays the default isoplane, such as “Right,” as illustrated below:



CLUSTERS

Clusters help you combine related groups of controls. DCL lets you specify vertical, horizontal, boxed, and unboxed clusters. In addition, *radio clusters* are required when you want to have two radio buttons on at the same time. In all other cases, clusters are needed only for visual and organizational purposes.

BricsCAD makes it look as if there are *eight* tiles for making clusters:

Column	Row
Boxed_Column	Boxed_Row
Radio_Column	Radio_Row
Boxed_Radio_Column	Boxed_Radio_Row

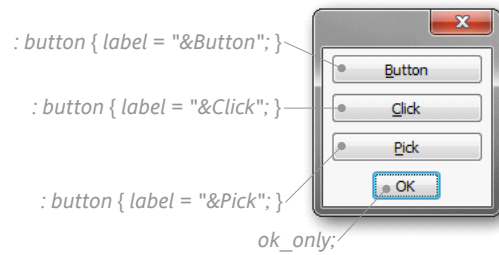
But these eight can be reduced to three, when you take the following into account:

- The **column** tile is usually not needed, because BricsCAD automatically stacks tiles vertically into columns.
- The **column** and **row** tiles display a box as soon as you include a label for them.
- Tiles with **radio** in their names are only for clustering radio buttons.

Columns and Rows

BricsCAD normally stacks tiles, so no **column** tile is needed, as illustrated by this DCL code:

```
x: dialog {
      : button { label = "&Button"; }
      : button { label = "&Click"; }
      : button { label = "&Pick"; }
      ok_only;
}
```

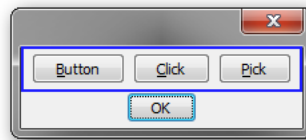


(The ampersand — **&** — specifies the shortcut keystroke that accesses the button from the keyboard with the **Alt** key, such as pressing **Alt+B**.)

To create a horizontal row of tiles, use the **row {}** tile, as shown in boldface below:

```
x: dialog {
    : row {
        : button { label = "&Button" ;}
        : button { label = "&Click"; }
        : button{ label = "&Pick" ;}
    }
    ok_only;
}
```

The boxing of the horizontal row is invisible, so I highlighted it with a blue rectangle.



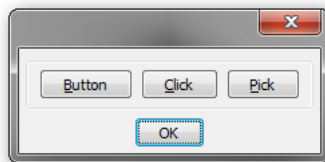
Because the **ok_only** tile is outside of the **row {}** tile, it is located outside of the cluster, stacked vertically below the row of three buttons.

Boxed Row

To actually show a rectangle (box) around the three buttons, change "row" to **boxed_row**, as follows:

```
x: dialog {
    : boxed_row {
        // et cetera
    }
    ok_only;
}
```

The box is made of white and gray lines to give it a chiseled 3D look.

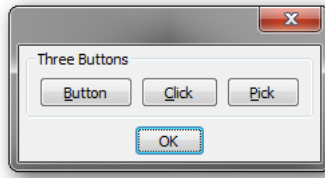


Boxed Row with Label

You can add text to describe the purpose of the boxed buttons with the **label** attribute, as shown in boldface below:

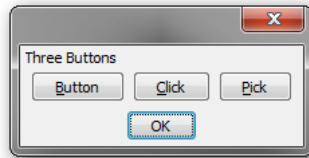
```
x: dialog {
  : boxed_row { label = "Three Buttons";
    // et cetera
  }
  ok_only;
}
```

The curious thing is that you get the same effect whether using the **boxed_row** or **row** tile. That's right: when you add a label to the **row** tile, BCL automatically adds a box around the cluster.



To eliminate the box, precede the row with the **text** tile for the title, as follows:

```
x: dialog {
  : text { label = "Three Buttons";}
  : row {
    // et cetera
  }
  ok_only;
}
```



Special Tiles for Radio Buttons

You can use the regular row and column tiles with radio buttons, except in one case: when more than one radio button needs to be turned on. Recall that only one radio button can be on (shown the black dot) at a time; BricsCAD automatically turns off all other radio buttons that might be set to on (**value = "1"**).

The solution is to use two or more **radio_column** tiles, each holding one of the radio button sets that need to be on.

It is not recommended to use rows for radio buttons, because this horizontal configuration is psychologically more difficult for users.

Debugging DCL

The most common DCL coding errors are due to errors in punctuation, such as leaving out a closing semi-colon or quotation mark. These problems are announced by error-message dialog boxes, which I illustrate later in this section.

DCL_SETTINGS

DCL contains a debugger for finding certain coding errors. To activate the debugger, add the **audit_level** parameter to the beginning of the DCL file, before the **dialog** tile:

```
dc1_settings : defalut_dc1_settings { audit_level = 3 ; }
x : dialog { // et cetera
```

The debugger operates at four levels:

Audit Level	Meaning
0	No debugging performed.
1	(Default) Checks for DCL errors that may terminate BricsCAD, such as undefined tiles or circular prototype definitions.
2	Checks for undesirable layouts and behaviors such as missing attributes or wrong attribute values.
3	Checks for redundant attribute definitions.

DCL ERROR MESSAGES

BricsCAD displays DCL-related error messages in dialog boxes. You may encounter some of the following:

Semantic error(s) is DCL file

Sometimes an error dialog box suggests that you look at the *acad.dce* file — the DCL error file. The problem is that the dialog box doesn't tell you *where* this file is located. After running Windows Search on my computer's C: and D: drives, I finally found the file in the *D:\documents and settings\administrator\my documents* folder.

The file contains information about errors, such as:

```
===== DCL semantic audit of c:\x =====
Error. Widget named "asdfasfads" is undefined.
```

It's not clear to me why some errors are displayed directly in the message dialog boxes, while others are stored in the *acad.dce* file.

Dialog has neither an OK nor a CANCEL button

Dialog boxes need to exit through an **OK** or **Cancel** button. At the very least, add the **ok_only** tile to the DCL file. DCL was written before Windows automatically added the **x** (cancel) button to all dialog boxes, and Autodesk has failed to update DCL to take this innovation into account.

Error in dialog file "filename.dcl", line n

Your DCL file contains the name of a tile unknown to BricsCAD. Check its spelling. In this example, **ok_only** was prefixed by a colon (:), which is incorrect for prebuilt tiles.

```
Incorrect:      : ok_only ;
Correct:       ok_only ;
```


Dialog too large to fit on screen

A tile in the DCL file is creating a dialog box that would not fit your computer's screen. This can happen when the **edit_edith**, **width**, or **height** attributes are too large.

Additional Resources

There is more to learn about writing dialog boxes with DCL, such as through these DCL tutorials:

- ▶ AfraLisp at <https://www.afralisp.net/dialog-control-language/> sports tutorials on AutoCAD, including these topics:



The screenshot shows the AfraLisp website. The header features the AfraLisp logo and a search bar. The navigation menu includes links for AutoLISP, Visual LISP, DCL, VBA, AutoCAD, Reference, Download, and Forum. The main content area is titled 'Dialog Control Language (DCL)' and includes a section 'What is Dialog Control Language?' with a brief description. A sidebar on the right contains links for 'Dialog Control Language (DCL)', 'DCL Beginners' Tutorials', 'DCL Intermediate Tutorials', and 'AfraLISP Archive'.

- Getting Started
- DCL Primer - Download
- Dialog Box Layout
- Dialogue Boxes Step by Step
- Dialogue Boxes in Action
- Nesting and Hiding Dialogues
- Hiding Dialogues Revisited
- LISP Message Box
- LISP Input Box
- Referencing DCL Files
- LISP Functions for Dialog Control Language (DCL)
- Functional Synopsis of DCL Files
- DCL Attributes
- Dialogue Box Default Data
- DCL Model
- DCL Progress Bar
- Attributes and Dialogue Boxes
- DCL without the DCL File
- The AfraLisp DCL Tutorials
- Entering Edit Boxes

- ▶ Jeffery Sanders has at http://www.jefferypsanders.com/autolisp_DCL.html his “DCL: Dialog Control Language” tutorial:



The screenshot shows a web page with a blue border. At the top, the text "(JefferyPSanders.com)" is displayed in a large, bold, blue font. Below this, on the left side, there are three blue links: "Back to AutoLisp Tutorial Homepage", "Back to AutoLisp Homepage", and "Back to JefferyPSanders.com". In the center, the title "The AutoLisp Tutorial - DCL" is written in bold black text, followed by "Dialog Control Language" in a smaller black font. A horizontal blue line separates the title from the "Contents" section. The "Contents" section lists several items with blue bullet points: "Getting Started", "Rows and Columns", "Controls", "Image", and "Action".

Getting Started
Rows and Columns
Controls
Image
Action
Set_Tile and Mode_Tile
List and how to handle them.
Saving data from the dialog box
Part 1 - Buttons
Part 2 - Edit_Box
Part 3 - List_Box
Part 4 - PopUp_List
Part 5 - Radio_Buttons
Part 6 - Text and Toggle
Part 7 - Putting it all together

Dabbling in VBA

The Pro and Platinum versions of BricsCAD for Widows include one of Microsoft's programming languages, VBA — Visual Basic for Applications. This is a version of Visual Basic designed to work inside of software programs. BricsCAD runs VBA programs from menus and toolbars, and at the command prompt.

VBA is completely different from LISP, just as LISP is completely different from Diesel and macros. If you learned the BASIC programming language, then that knowledge will be of no help, unfortunately, because Visual Basic has nothing in common with BASIC except for the name.

This chapter introduces you to the concepts of VBA programming and show you how to use it in BricsCAD. (The Classic, MacOS, Linux, and demo versions of BricsCAD do not include VBA.)

CHAPTER SUMMARY

The following topics are covered in this chapter:

- Introducing Visual Basic for Applications
- Learning about VBA-related commands
- Sending commands through VBA
- Using the VBA programming environment
- Designing dialog boxes
- Examining VBA code

QUICK SUMMARY OF VBA PROGRAM COMPONENTS

Projects store macros. (LISP calls these “programs.”)

Macros refer to chunks of VBA programming code. (LISP calls these “functions.”) VBA macros can be embedded (stored in drawings) or saved to .dwb files on disk. See the boxed text for the pros and cons of each.

Reactors are pieces of macro code that react to events in the drawing, such as the drawing being saved, an object added to the drawing, or the user clicking a mouse button.

Forms refer to the location where VBA code is constructed. Often, forms look like dialog boxes.

Controls refer elements in forms, such as check boxes and droplists.

Classes are definitions of objects. For example, AcadLine is the class that defines the line entity.

Objects refer to classes put into forms. The objects can have the following attributes:

- Properties that describe the object, such as its color, height, and width.

- Methods that modify objects, such as copying and rotating them.

- Events that report when objects are modified.

QUICK SUMMARY OF VBA COMMANDS IN BRICSCAD

The VBA-related command names are shown below in boldface, while equivalent menu names are shown in parentheses. You access the menu items from the Tools | VBA menu.

Vba (Visual Basic for Applications) opens the VB Editor for writing and debugging macros.

VbaRun (Macros) loads and runs VBA macros; displays the Macros dialog box and lists the names of VBA macros stored in the current drawing.

VbaNew (New Project) specifies the name of a new VBA project file.

VbaLoad (Load Project) loads .dwb VBA project files; displays the Open dialog box.

-VbaLoad command load .dwb project files at the command prompt.

VbaMan (Project Manager) displays the VBA Manager dialog box.

AddInMan (Add-in Manager) lists programs that can be loaded into BricsCAD, and controls how they are loaded; displays the Add-In Manager dialog box.

(**VbaStmt** is not supported by BricsCAD; its purpose in other programs is to load and run macros at the command prompt.)

Introduction to VBA

VBA is the second most-important programming language in BricsCAD. While LISP is the easier of the two to learn and use, it becomes cumbersome and slow for large programs and large sets of data. Furthermore, to create dialog boxes, LISP requires that you employ the difficult-to-understand DCL system.

At the other end of the programming spectrum is DRX or BRX, the DWG or BricsCAD runtime extensions. These programming interfaces are the fastest of all, because they are intimately tied into BricsCAD. You use D/BRX application programming interfaces with programs written in C or one of its offshoots. They are not simple to learn, and they present a drawback: you must pay for a compiler that works with D/BRX. In contrast, LISP is free with all versions of BricsCAD, and VBA is free with BricsCAD Pro. DRX is not covered by this book. Note that as of BricsCAD V8, B/DRX replaces SDS, the Softdesk Development System. Also as of V8, BricsCAD switched the format of its VBA source files from IntelliCAD's VBI to AutoCAD's DVB format.

In contrast, VBA is fast and is designed with today's user interfaces in mind. An advantage to learning VBA is that you can use the same programming language in many other Windows programs; learn once, program in many. Perhaps the toughest part of learning VBA is getting to know its jargon. Let's begin!

ACCESSING VBA PROGRAMS

You write VBA code in a separate programming environment called the "VB Editor." The editor provides assistance in writing the code, as well as constructing the user interface, which usually consists of dialog boxes.

You can run VBA programs at the BricsCAD command line or through its Add In Manager dialog box. Programs can also be launched from menu and toolbar macros, as well as from VisualLISP functions, topics not covered by this ebook.

The code can be *embedded* in a BricsCAD drawing or kept outside of BricsCAD for access by all drawings:

- ▶ To run embedded macros, use the **VbaRun** command.
- ▶ To run the macros stored in a *.dvh* project file, first load them through the **VbaMan** dialog box or the **VbaLoad** command prompt. Once loaded, the macros can be run with the either the **VbaRun** or **VbaMan** commands.

Sending Commands

VBA has a command that works just like the LISP (**command**) function: **SendCommand** executes any BricsCAD command, such as Line, Erase, and Zoom. The function also handles command options, such as "1,1" and "All."

Let's take a look at it. Here is the VBA code for drawing a line between several pairs of x,y coordinates.

```
Sub Using_the_SendCommand()  
    ThisDrawing.SendCommand "line 1,1 1,8 11,8 11,1 c "  
End Sub
```

TIP This VBA code isn't too different from the equivalent code in LISP, which looks like this:
(defun using_the_sendcommand ()
 (command "line 1,1 2,2 c ")
)

The words used in the snippet of VBA code have the following meaning:

Sub starts a new subroutine (or function).

Using_the_SendCommand() names the subroutine. The parentheses () indicate that no variables are used. Unlike LISP, VBA needs to know the *names* of variables and their type ahead of time. I'll cover variables and types later in this chapter, but for now it's enough to know that *type* refers to the type of data the variable holds, such as text (strings), whole numbers (integers), decimal numbers (reals), and other kinds of data.

ThisDrawing.SendCommand operates in the current drawing, identified generically by "ThisDrawing." You do not specify the drawing's name, you just need to use "ThisDrawing," and VBA knows what you're talking about.

"line 1,1 1,8 11,8 11,1 c " executes the Line command, drawing four lines that make up rectangle between 1,1 to 11,8. The command and its prompts are read as a string, and then sent to BricsCAD's command processor — just as if you had typed this at the command prompt.

TIP To end the command correctly, ensure that the string has a space at the end, just before the closing quotation mark. In the code above, you can see the space between the c and the ".

End Sub signals the end of the subroutine.

EMBEDDED OR EXTERNAL

BricsCAD stores VBA macros in drawings (embedded) or in .dwb files (external). There are pros and cons to each method, as listed by the following table:

	Embedded	External
Storage:	in drawings	in .dwb files
Loading:	loaded with drawings	loaded with VbaLoad command
Distribution:	with the .dwg file	with the .dwb file
Reactors:	yes	no

An embedded macro cannot be used by other drawings, unless you specifically embed it into other drawing files.

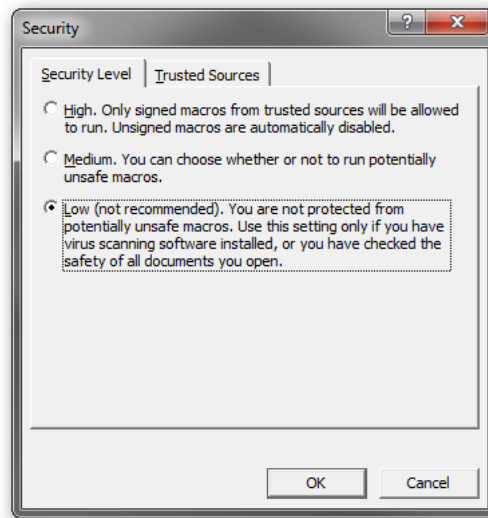
Use the VbaMan command's Embed button to convert projects to embedded projects. A serious problem with embedded macros is that they can contain viruses. Hence, BricsCAD displays a warning dialog box that gives you the option of disabling or enabling macros, or preventing them from loading at all.

WRITING AND RUNNING VBA ROUTINES

Whereas Notepad or other text editor can be used to write LISP routines, a programming environment included with BricsCAD must be used for VBA. You use this environment to write and run all code.

To access the VBA editor, follow these steps:

1. The first step is to ensure that BricsCAD can run VBA macros. Because VBA is a source of viruses, the ability to run VBA programs is normally turned off. Here is how to enable VBA macros:
 - a. From the BricsCAD **Tools** menu, choose **Security**. Notice the Security dialog box.



- b. If necessary, choose the **Security Level** tab.

TIP This dialog box is necessary because of a poor decision by Microsoft programmers. When they created VBA: they allowed documents to store VBA code. This convenient feature turned into a major security problem, because it made it easy for hackers to distribute benign-looking Word and Excel files that contained malicious VBA code.

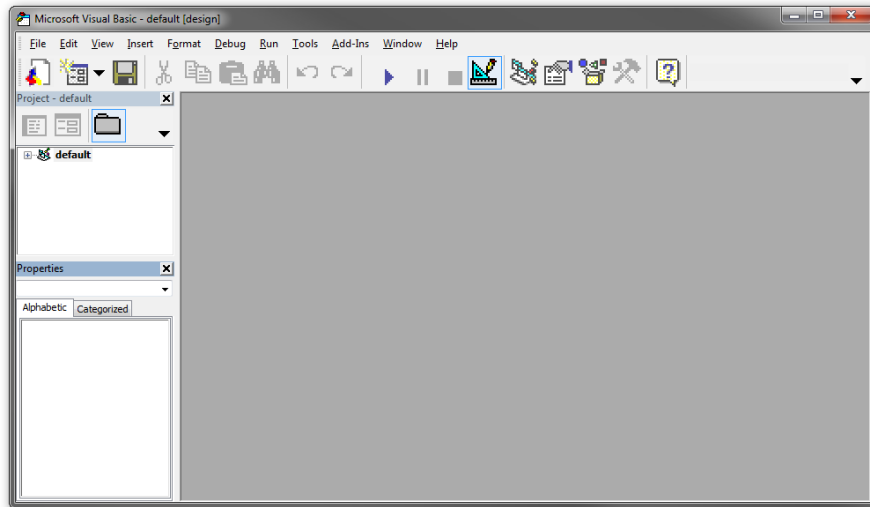
After too many people and companies suffered from having precious files erased from their computers, Microsoft finally added this dialog box to overcome the VBA exploit. Today, Windows-based documents cannot run VBA code by default (security level = high), and so you must lower the security level for VBA to work.

- c. If the security level is set to High, change it to **Medium** or **Low**. The differences between the settings are listed by the table below:

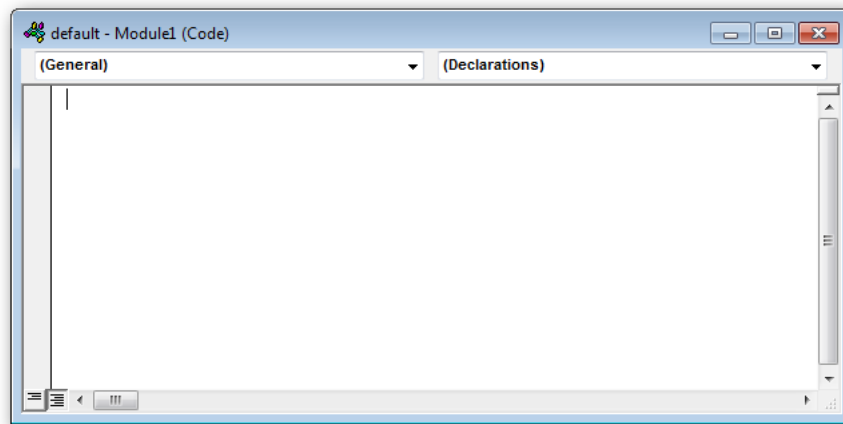
VBA Setting	Meaning
High	All VBA routines are prevented from operating; default.
Medium	BricsCAD asks if you wish to run each VBA routine.
Low	All VBA routines are run, without question

- d. After you change the security level in this dialog box, you must restart BricsCAD with the **Quit** command or by using **File | Exit**.
2. With VBA enabled, you can now open its programming environment. From the **Tools** menu, choose **VBA**, and

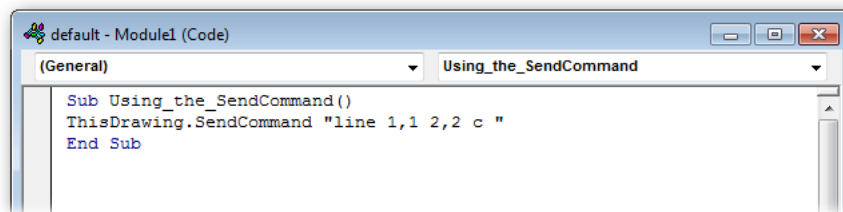
then choose **Visual Basic for Applications**. Notice the VBA programming environment.



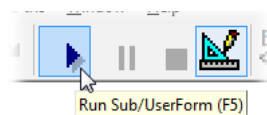
3. Code is written in *modules* — a form that is initially blank, into which you type in the code. To start a new module, from the **Insert** menu, choose **Module**. Notice that a blank window appears, as shown below.



4. Enter the following code into the module. (This is the same line drawing code you saw earlier.)
`Sub Using_the_SendCommand()
ThisDrawing.SendCommand "line 1,1 2,2 1,2 c "
End Sub`

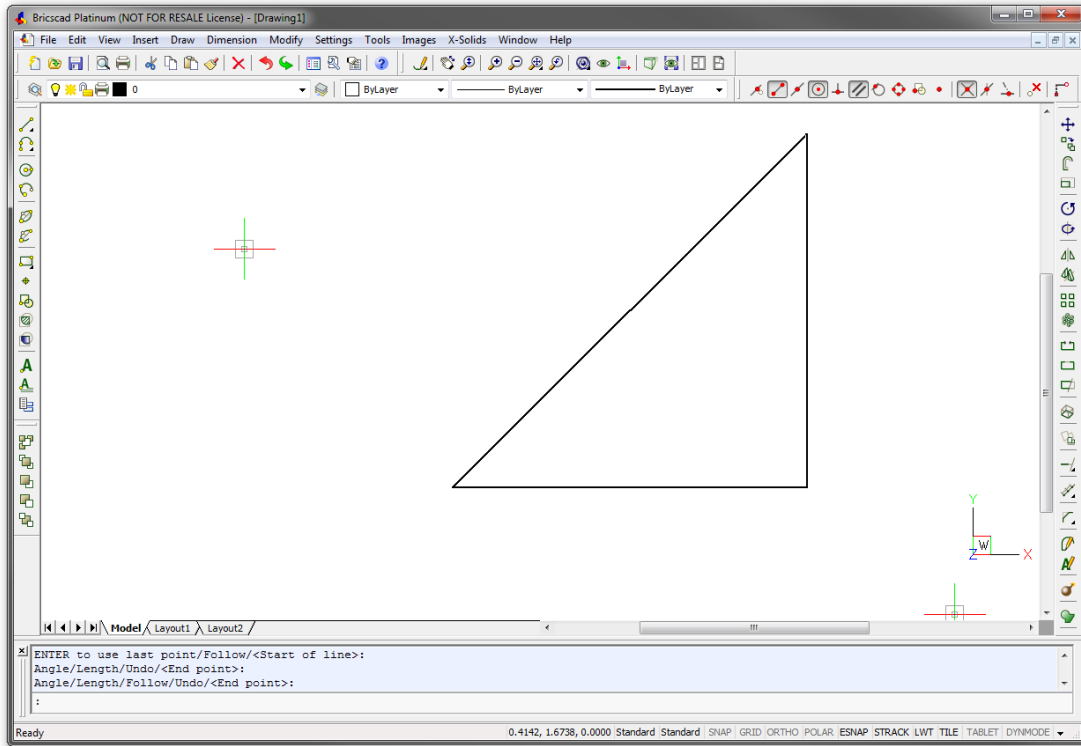


5. You now have enough code to execute a program. Click the **Run** button, which you find on the toolbar.



Success! Notice that BricsCAD draws a triangle. If the routine fails to run, check for these problems:

- Is security is set to High or Medium?
- Does the code contain spelling errors?



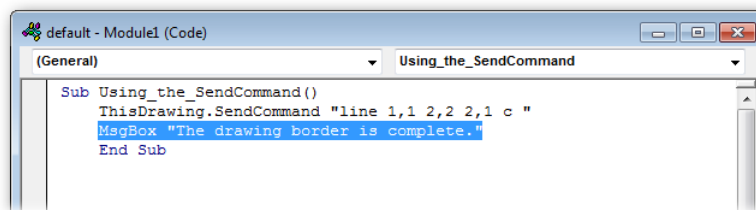
You can use `SendCommand` to draw and edit entities, and to change viewpoints with commands like `Zoom` and `Pan`. You can use it to insert blocks, change properties, and plot drawings. Just be careful that you enter the options of commands correctly; coordinates and the names of options are particularly fussy.

DISPLAYING MESSAGES

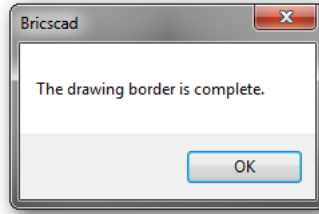
Displaying messages in dialog boxes is as easy as using this code with the **MsgBox** function:

```
MsgBox "The drawing border is complete."
```

1. Add the line to the code in the VBA editor:



2. And then click the **Run** button again. You should see a dialog box in BricsCAD that looks like this:



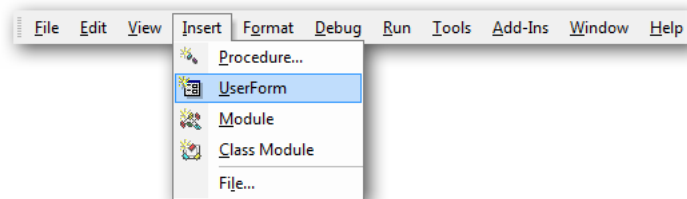
Now that is much, *much* easier than coding a dialog box in LISP with DCL!

Constructing Dialog Boxes

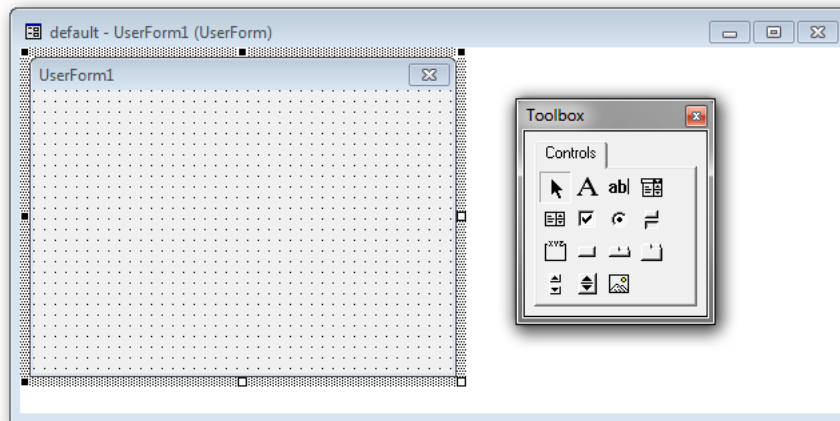
Speaking of dialog boxes, VBA includes an interactive dialog box construction kit called the “user form.” Let’s take a look at how it works.

To start a new user form, follow these steps:

1. In the Project palette, right-click **Module1**.
2. From the shortcut menu, choose **Insert**, and then choose **Userform**.



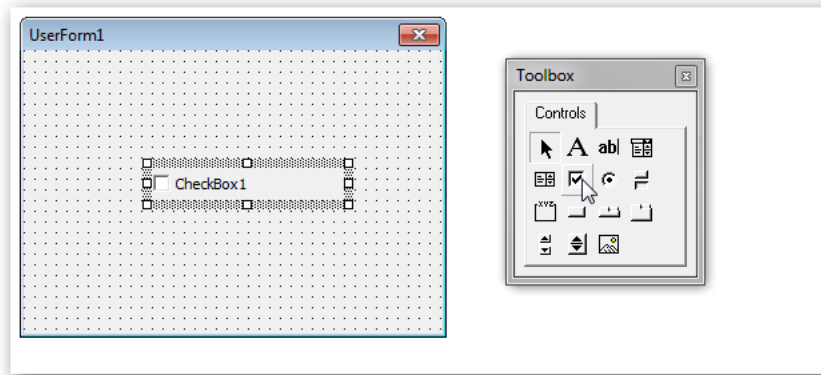
Notice the gray window filled with a grid of dots. This is where you design dialog boxes.



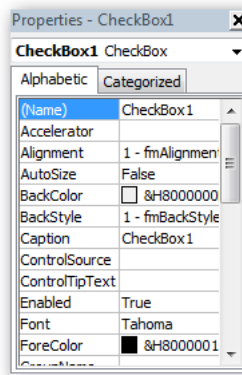
Adjacent to the form is the *Toolbox*. It contains the elements that make up dialog boxes — known as “controls” in the correct VBA jargon. You will probably recognize many of the controls, such as text entry box, check box, and radio button.

3. To place a control, choose one from the Toolbox, and then position it in the user form. For example, to add a check box, follow these steps:
 - a. In the Toolbox, click the check box item.

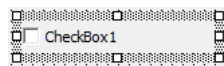
- b. In UserForm1, click anywhere. Notice that the check box is placed with generic text that reads "Check-box1."



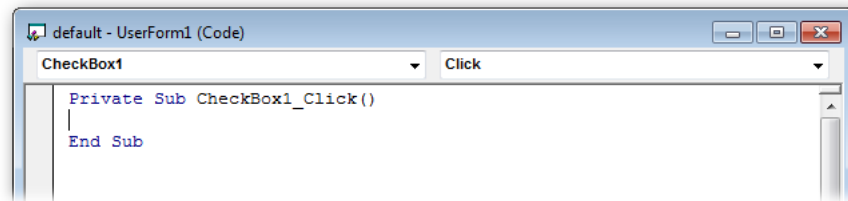
- c. To edit the text or any other property of the check box, glance over at the Properties palette. (If it is not visible, choose **Properties Window** from the **View** menu; if necessary, click the **Categorized** tab.) Notice the name of the control, plus a ton of other properties — the choices can become quite overwhelming.



- d. You have written some code and you have drawn a simple dialog box. To connect the check box with the VBA code, choose the **Select Objects** tool from the Toolbox, and then double-click the check box control.



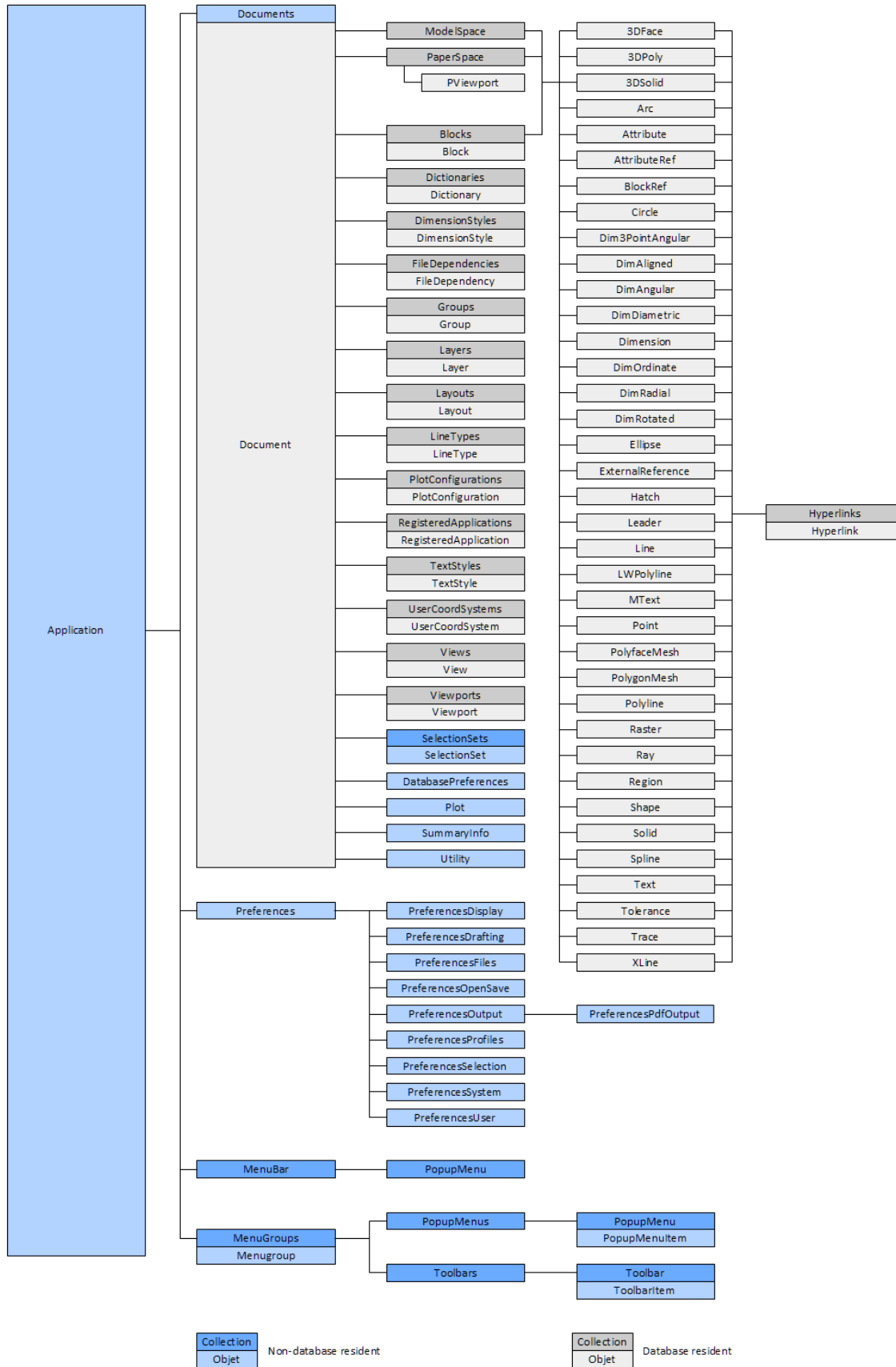
Notice that another module window opens, into which you can enter code — something I won't detail at this point.



We will look at the code-dialog box link in greater detail later in this chapter. First, though, an introduction to how VBA really works.

BricsCAD V20 Automation Object Model

Source: https://www.bricsys.com/bricscad/help/en_US/CurVer/DevRef/index.html?page=source%2FCOM_ComponentObjectModel.htm



Object-Oriented Programming

In programming, is more efficient to work with *objects*. No, not geometric objects, but programming objects. To keep clear the distinction, I refer to geometric objects as *entities*.

You had a hint of the object-oriented nature of VBA with the **ThisDrawing.SendCommand** piece of code: the SendCommand function is instructed to operate on the ThisDrawing object, which is the current drawing. You can add objects to ThisDrawing, such as **ModelSpace** to ensure the commands are executed in model space instead of paper space:

```
ThisDrawing.ModelSpace.SendCommand()
```

Notice the dots (.) that connects them, much like the dot in the dotted pairs used by LISP to access entity data. VBA is premised upon object orientation, where everything in BricsCAD is organized as objects and according to a strict hierarchy. Technically, this is known as "exposing the BricsCAD database" through Microsoft's Common Object Model (COM).

COMMON OBJECT MODEL

On the facing page is a very important figure: it is a diagram of the object model in BricsCAD. (It changes from release to release as new objects are introduced.) The chart shows how entities relate to objects:

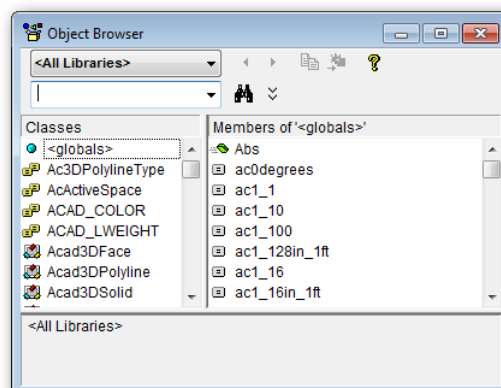
- ▶ Entities are in found model or paper space, or in blocks.
- ▶ Model/paper space and blocks are found in *documents*.
- ▶ Documents (drawings) are found in the *application* (BricsCAD).

As an alternative to the diagram, you can use the Object Browser found in BricsCAD's VBA programming environment, described next.

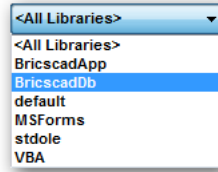
OBJECT BROWSER

The object browser lists all of the objects that VBA can access in BricsCAD. To use the object browser, follow these steps:

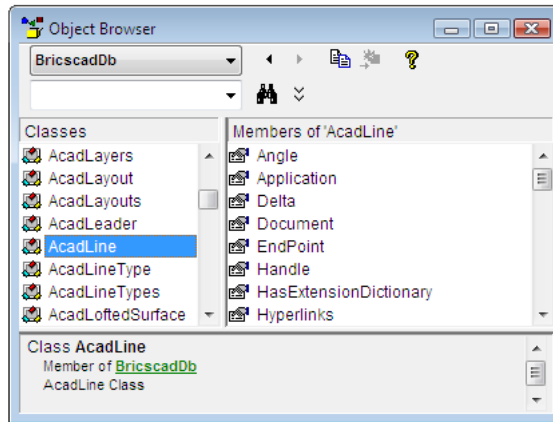
1. From the **View** menu, choose **Object Browser**. Notice the Object Browser palette.



- In the All Libraries droplist, choose **BricsadDb**. (Db is short for database.)

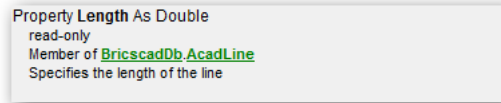


- Scroll down to **AcadLine**. This is the BricsCAD line entity, but it is named *acadline* to maintain compatibility with VBA applications programmed in AutoCAD.



- On the right, notice all the properties, methods, and events that are available for line entities. I've detailed them in the following section.
- At the bottom is helpful info. As the cursor rests on a *member*, a brief description is provided, along with a link to the *parent*.

The figure below shows the information provided for **Length**:



- ▶ Length is a *property* that specifies the length of the current line.
- ▶ It is a double variable (double accuracy floating point).
- ▶ It is read-only, which means programmers and users cannot edit the value.
- ▶ It is a member of BricsCADDb.AcadLine.

By now, you might realize that the Property palette reports the values stored by BricsCADDb for all entities in the drawing. Let's take a closer look at all that a line object entails.

TIPS Rename the buttons and text boxes so that the names describe what they do. For instance, rename the OK button as **btnOK**; rename the Last Point text box as **txtLastPoint**, and so on.

If the font size and style in the Code window are too small, you can change them. From the **Tools** menu, choose **Options**. Select **Editor Format**, and then choose a different font size and/or font name.

The VBA code editor uses color to highlight different type of code:

Green text	Comments
Black text	Normal code
Blue text	VBA keywords
Red text	Errors in the syntax
Yellow highlight	Execution points
	Breakpoints

LINE ENTITY

The line entity is created with the **AcadLine** method in model or paper space, and in a block:

ModelSpace.AddLine adds a line to model space.

PaperSpace.AddLine adds a line to the current layout tab.

Block.AddLine adds a line to the specified block, dynamic block, or xref block.

Lines have properties, methods, and events:

- ▶ *Properties* affect the geometry and look of the line.
- ▶ *Method* refers to the ways in which lines can be edited.
- ▶ *Events* refers to the manner in which entities report that they've been changed.

Below, I've listed all properties, methods, and events for line entities. The list gives you an idea of the richness (or, complexity) of the access you have to the internals of BricsCAD, the richness provided by the object model through VBA.

Properties

Lines can have the following properties. Some of these will be familiar to you; others will be new.

Properties	Meaning
Geometric Properties	
Angle	Angle in radians from the x-axis measured counterclockwise.
Delta	Delta-x, -y, and -z values, from one endpoint to the other.
Length	Length of the line.
Normal	Normal to the line.
EndPoint	X,y,z-coordinate of the line's end point.
StartPoint	X,y,z-coordinate of the start point.
Entity Properties	
Hyperlinks	Embedded hyperlink.
Layer	Layer name.
Linetype	Linetype name.
LinetypeScale	Linetype scale.
Lineweight	Lineweight width.
Material	Material name (used for rendering).
PlotStyleName	Plot style name, if enabled.
Thickness	Thickness, in the z-direction.
TrueColor	Color.
Visible	Visibility, independent of layer setting.
Other Properties	
Application	Specifies the BricsCAD application.
Document	Specifies the drawing.
Handle	Specifies the entity identification number.
HasExtensionDictionary	Reports whether the line has an extension dictionary.
ObjectID	Alternative method of obtaining the entity id number.
OwnerID	Reports the ObjectID of the parent object.

Methods

The line can be edited with the following methods:

Method	Meaning
Entity Editing	
ArrayPolar	Creates a polar array of the line.
ArrayRectangular	Creates a rectangular array.
Copy	Copies the line.
Delete	Erases the line.
Mirror	Mirrors the line.
Mirror3D	Mirrors the line in 3D.
Move	Moves the line.
Offset	Creates an offset copy of the line.
Rotate	Rotates the line.
Rotate3D	Rotates the line in 3D.
ScaleEntity	Resizes the line.
TransformBy	Moves, scales, and/or rotates the line.
Other Method:	
GetBoundingBox	Reports the coordinates of the rectangle that encompasses the line.
GetExtensionDictionary	Returns the line's extension dictionary.
GetXData	Returns the line's extended entity data.
SetXData	Stores extended entity data in the line.
IntersectWith	Returns coordinates where line intersects other objects.
Highlight	Highlights the line.
Update	Regenerates the line.

Events

When entities are changed, they trigger events. For lines, there is just one event. The **Modified** event is triggered whenever a property is set, even when the new value equals the current one.

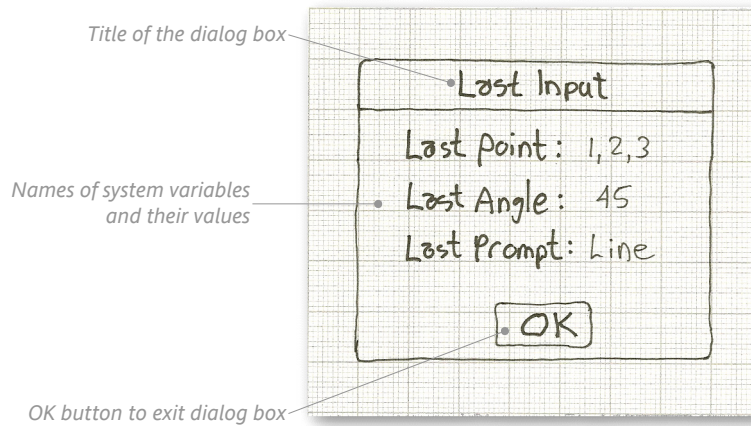
Events are prevented from triggering while modal dialog boxes are open. (A *modal* dialog box is one that must be dismissed before you can continue working in BricsCAD; ie, most dialog boxes.)

Dialog Box with Code

In an earlier chapter, I showed you how to construct a dialog box using DCL, and then add the LISP code to make it work. The dialog box looked like this:

The dialog box displays the current value of three system variables:

- ▶ **Last point** reports the current value of the LastPoint system variable.
- ▶ **Last angle** reports the value of LastAngle (read-only).
- ▶ **Last prompt** reports the value of LastPrompt (read-only).

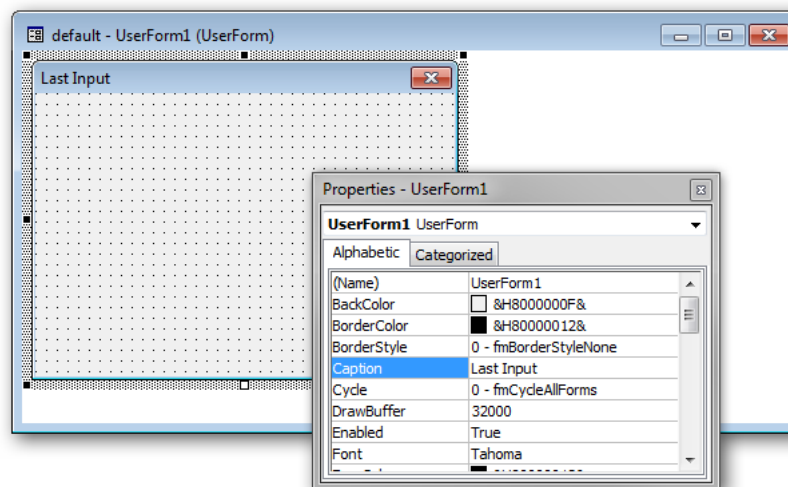


Let's repeat the tutorial, this time using VBA to do both jobs done separately by DCL and LISP before – designing the dialog box and writing-running the code.

DESIGNING THE DIALOG BOX

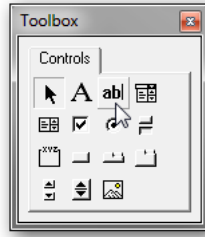
Dialog boxes are designed with the VBA programming environment, as follows:

1. Start BricsCAD, and then use the **Tools | VBA | Visual Basic for Applications** command to open the VBA programming environment.
2. Start a new Userform. (From the **Insert** menu, choose **UserForm**.) Notice that VBA creates a generic dialog box named UserForm1.
3. Change the name on the title bar by following these steps:
 - a. Open the Properties palette. (From the **View** menu, choose **Properties Window**.)
 - b. Scroll down to **Caption**, and then change "UserForm1" to **Last Input**. As you type, notice that the title bar of the dialog box updates at the same time.

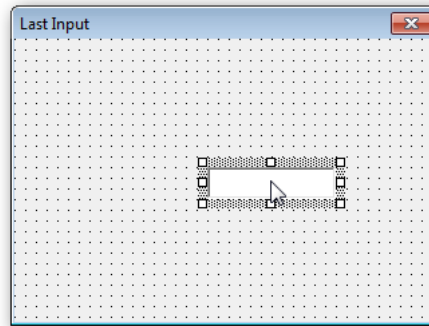


4. The bulk of our new dialog box consists of three text input boxes. The first one is constructed like this:

- a. In the Toolbox, choose the **TextBox** control.



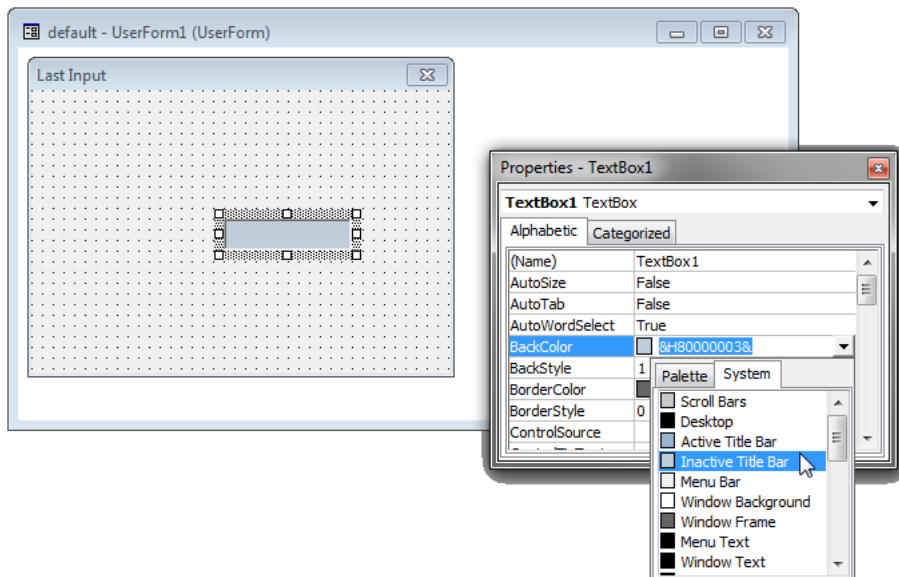
- b. Click anywhere in the center of the form. Notice the text entry box appears, but it lacks a text prompt for the user, such as "Last Angle:" You add the text a little later on.



5. In BricsCAD, the LastAngle system variable is *read-only*. This means that users can view the value but not change it. Text boxes that cannot be edited by users are traditionally colored gray. Here is how to make the text box read-only and gray:

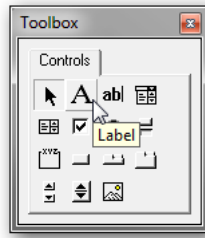
- a. Ensure the text box is selected (has grips, as illustrated above).
- b. In the Properties palette, change the value of **BackColor** (found in the Appearance section) to **Inactive Title Bar**.

“Inactive Title Bar” is an *enum*, a preset value in VBA, kind of like **pi** in LISP. (Enum is short for “enumerated.”) Should the user change the colors of Windows’ user interface, the background color of this text box will also change.

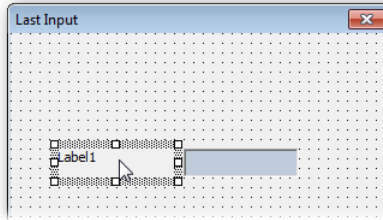


6. To add the prompt, use the Label tool, as follows:

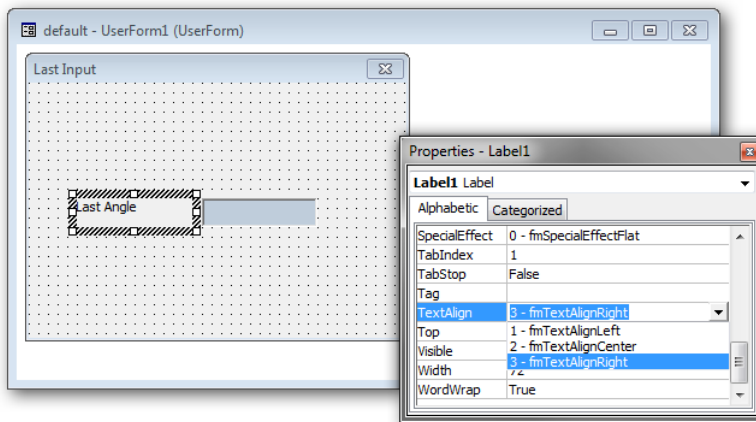
- a. Choose the **Label** tool from the Toolbox.



- b. Click and drag a rectangle in front of the text box. If necessary, drag the label into position.



- c. Backspace over the generic "Label1" text, replacing it with **Last Angle**:



- d. To right justify the text, change the value of **TextAlign** property to **3** (fmTextAlignRight).

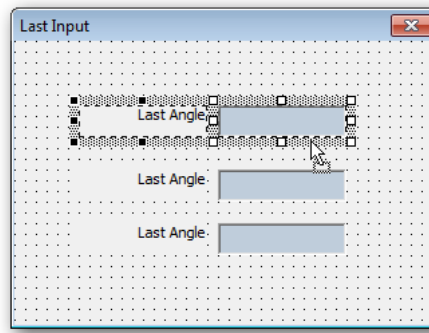
TIP You can drag dialog box elements with the cursor, but it tends to jump to the grid dot spacing. To fine tune the location of an element, use the **Position** section of the Properties palette.

Change the value of **Top** to move the element up and down, **Left** to move horizontally. The values represent the number of pixels from the upper left corner of the dialog box.

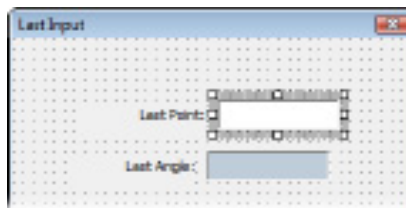
7. You can create the other two text input boxes through copy and paste:

- a. Use the cursor to select both elements. Here are two ways to do it:
 - You can drag a rectangle around both of them.
 - Alternatively, you can choose one, hold down the **Ctrl** key, and then choose the other.
- b. Press **Ctrl+C** to make a copy (stored in the Clipboard).

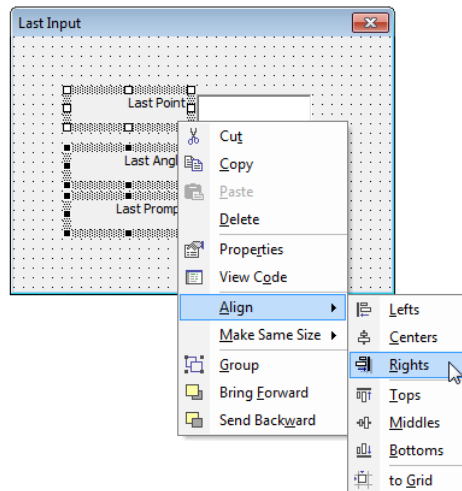
- c. Press **Ctrl+V** to paste the copy in the dialog box. The copies are pasted right on top of the originals, unfortunately. This means you need to move one of them, following the pasting.



- d. Separate the overlapping elements by dragging the copies above the originals.
8. Change the properties of the new pair of text input elements:
 - Change text label to **Last Point**.
 - Change **BackColor** of the text box **Window Background** (white), because the value of the LastPoint system variable can be changed by the user.

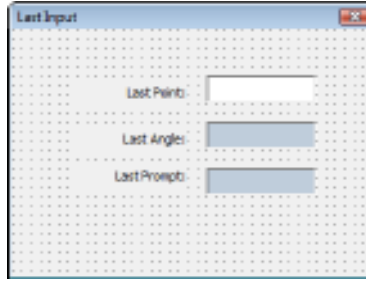


9. Edit the wording for the **Last Prompt** field. Keep the background color of the Last Prompt text box gray, because the LastPrompt system variable cannot be edited by users.
10. It is quite likely that the elements don't line up perfectly. VBA can align them for you, as follows:
 - a. Select the three text elements, and then right-click.

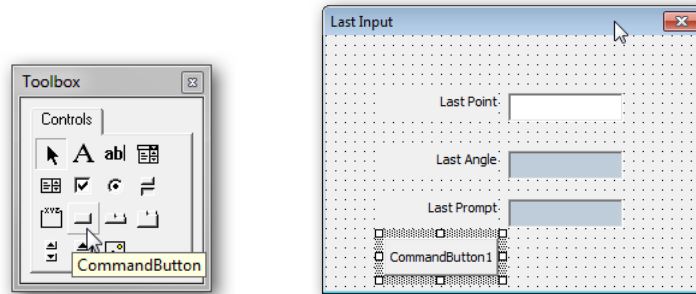


- b. From the shortcut menu, choose **Right**. Notice that they now line up perfectly.

- c. Repeat for the three input boxes.

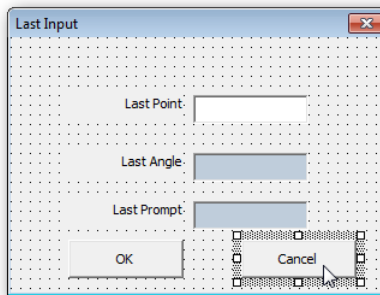


11. The final elements are the OK and Cancel buttons. From the Toolbox, drag the **Command Button** element onto the user form.



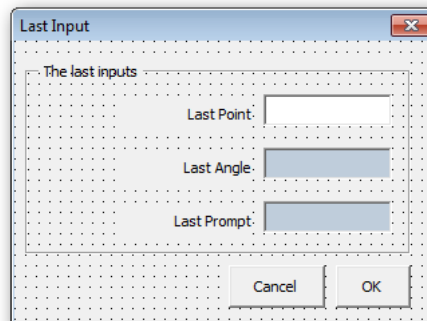
Left: Selecting the CommandButton tool, and then... right: ...dragging it onto the form.

12. Change its Caption property to **OK**.
13. Repeat to add the **Cancel** button.



The *design* of the dialog box is complete. The next stage adds code that makes the dialog box operate.

If you wish, you can fine tune the look of the dialog box by making the OK button narrower, adding a frame around the text input boxes, changing colors of elements, and so on. I find it interesting that I prefer working with DCL, because BricsCAD takes care of lining up dialog box elements so that it looks good — without all the manual tweaking required by VBA.



ADDING THE CODE

With the dialog box design in place, let's start working on the code. In LISP, a single routine handles everything that happens in the dialog box; in contrast, VBA uses many snippets of code. One snippet handles the Cancel button, another the OK button, another handles the value displayed by the Last Point text box, and so on.

You don't need to worry about linking code snippets to dialog box elements. VBA does that for you. When the user clicks on a text box or the OK button, VBA runs the correct snippet of code automatically.

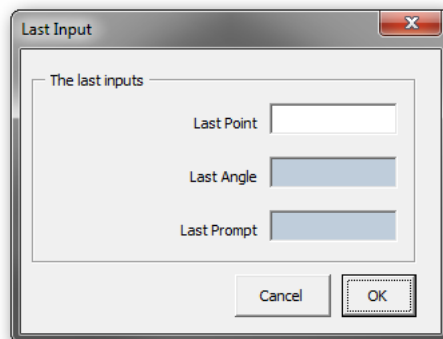
Clicking Cancel

To link the Cancel button to a VBA code snippet, follow these steps:

1. Double-click the **Cancel** button. Notice that a module-like form appears, and that it is partially filled out.
2. Add the command for closing the dialog box:
`End`
3. You're done!



4. Well, that's not quite all. You still need to test that Cancel button actually works. Here's how:
 - a. On the VBA toolbar, click the **Run** button. Notice that the dialog box appears in BricsCAD.



- b. Try clicking an element other than the Cancel button, such as the OK button. Nothing happens, because there is no code tied to it.
- c. Click **Cancel**. The dialog box should disappear. Yay, it works!

QUICK SUMMARY OF VBA DATA TYPES

Data Type	Comment	Range From	To
Byte	...	0	255
Boolean	...	True	False
Integer	...	-32,768	32,767
Long	Long integer	-2,147,483,648	2,147,483,647
Single	Single-precision floating-point	-3.402823E38 1.401298E-45	-1.401298E-45 (negative Values) 3.402823E38 (positive values)
Double	Double-precision floating-point	-1.79769313486231E308 4.94065645841247E-324	-4.94065645841247E-324 (negative values) 1.79769313486232E308 (positive values)
Decimal	...	+/-79,228,162,514,264,337,593,543,950,335 (no decimal point) +/-7.9228162514264337593543950335 (28 decimal places) +/-0.00000000000000000000000000000001 (smallest nonzero number)	
Date	...	January 1, 100	December 31, 9999
Currency	Scaled integer	-922,337,203,685,477.5808	922,337,203,685,477.5807
String	Variable-length Fixed-length	0 1	approximately 2 billion characters approximately 65,400 characters
Variant	Numbers Characters	Any numeric value up to a double Same as variable-length string	
Object	...	Any object reference	
User-defined	...	Same range as its data type.	

QUICK SUMMARY OF VBA DATA TYPE RETURN VALUES

Constant	Value	Description
vbEmpty	0	Empty or uninitialized
vbNull	1	Null or no valid data
vbInteger	2	Integer
vbLong	3	Long integer
vbSingle	4	Single-precision floating-point number
vbDouble	5	Double-precision floating-point number
vbCurrency	6	Currency value
vbDate	7	Date value
vbString	8	String
vbObject	9	Object
vbError	10	Error value
vbBoolean	11	Boolean value
vbVariant	12	Variant (array)
vbDataObject	13	Data access object
vbDecimal	14	Decimal value
vbByte	17	Byte value
vbUserDefinedType	36	Variant (user-defined types)
vbArray	8192	Array

QUICK SUMMARY OF VBA STRING MANIPULATION

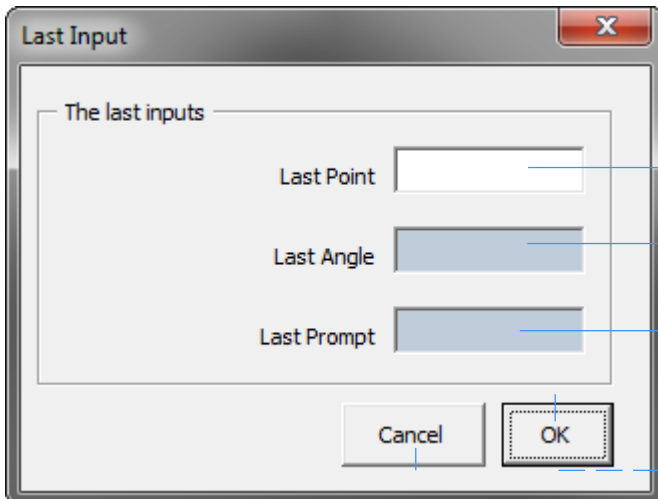
<u>Keyword, Operator</u>	<u>Comment</u>
Asc, Chr	Accesses ASCII and ANSI values.
Format, Lcase, Ucase	Converts to lower- or uppercase.
Format	Formats strings.
InStr, Left, LTrim, Mid, Len	Finds lengths of strings.
LSet, Rset	Justifies string left or right.
Option Compare	Sets string comparison rules.
Right, RTrim,	Trim Manipulates strings.
Space, String	Creates strings of repeating characters.
StrComp	Compares two strings.
StrConv	Converts strings.
&	Concatenates strings.

LastInput.Dvb

With the introduction to VBA programming behind you, let's carry on and examine a fully-coded program. Below is the Last Input dialog box, and on the facing page is the VBA code for *LastInput.Dvb*. In the following pages, I comment on parts of the code.

The main part of the project is shown in **cyan**; other modules are like subroutines that support the main module. I've added lines to visually separate modules, and I have color coded module names so that you can cross-reference them.

The following VBA code was developed by Ferdinand Janssens, programmer at Bricsys.



```
Option Explicit
```

```
Private Sub txtLastAngle_Change()  
End Sub
```

```
Private Sub txtLastPrompt_Change()  
End Sub
```

```
Private Sub UserForm_Initialize()  
    UpdateForm  
End Sub
```

```
Private Sub btnUpdate_Click()  
    UpdateForm  
End Sub
```

```
Private Sub btnOK_Click()  
    Unload Me  
End Sub
```

```
Private Sub UpdateForm()  
    Dim vLastpoint As Variant  
    vLastpoint = ThisDrawing.GetVariable("LASTPOINT")  
    Me.txtLastPoint.Text = PointToString(vLastpoint)  
    Me.txtLastAngle.Text = AngleToString(ThisDrawing.GetVariable("LASTANGLE"))  
    Me.txtLastPrompt.Text = TrimLF(ThisDrawing.GetVariable("LASTPROMPT"))  
End Sub
```

```
Private Function PointToString(vIn As Variant) As String  
    Dim sPt As String: sPt = vbNullString  
    Dim iPrecision As Integer  
    iPrecision = ThisDrawing.GetVariable("LUPREC") ' LUPREC holds the current Linear  
    Unit precision (see Setting Dialog)  
    If VarType(vIn) > vbArray Then  
        sPt = StringFromValueFixedDecimal(vIn(0), iPrecision) & ", "  
        sPt = sPt & StringFromValueFixedDecimal(vIn(1), iPrecision) & ", "  
        sPt = sPt & StringFromValueFixedDecimal(vIn(2), iPrecision)  
    End If  
    PointToString = sPt  
End Function
```

```
Private Function StringToPoint(sIn As String) As Variant  
    Dim sCoords() As String: sCoords = Strings.Split(sIn, ",")  
    Dim tmpPt(0 To 2) As Double  
    If UBound(sCoords) = 0 Then  
        tmpPt(0) = Val(sCoords(0))  
    ElseIf UBound(sCoords) = 1 Then  
        tmpPt(0) = Val(sCoords(0))  
        tmpPt(1) = Val(sCoords(1))  
    ElseIf UBound(sCoords) = 2 Then  
        tmpPt(0) = Val(sCoords(0))  
        tmpPt(1) = Val(sCoords(1))  
        tmpPt(2) = Val(sCoords(2))  
    End If  
    StringToPoint = tmpPt  
End Function
```

```
Private Sub txtLastPoint_BeforeUpdate(ByVal Cancel As MSForms.ReturnBoolean)  
    Dim ptModif As Variant  
    ptModif = StringToPoint(Me.txtLastPoint.Text)  
    ThisDrawing.SetVariable "LASTPOINT", ptModif
```

```

    Me.txtLastPoint.Text = PointToString(ptModif)
End Sub

```

```

Private Function StringFromValueFixedDecimal(ByVal dVal As Double, ByVal iDecimals As Integer) As String
    StringFromValueFixedDecimal = VBA.FormatNumber(VBA.Round(dVal, iDecimals), iDecimals)
End Function

```

```

Private Function TrimLF(ByVal sVal As String) As String
    TrimLF = VBA.Replace(sVal, vbLf, vbNullString)
End Function

```

```

Private Function AngleToString(dRadians As Double) As String
    Dim iAnglePrecision As Integer
    iAnglePrecision = ThisDrawing.GetVariable("AUPREC") ' AUPREC holds the current Angular Unit precision (see Setting Dialog)
    AngleToString = ThisDrawing.Utility.AngleToString(dRadians, acDegrees, iAnglePrecision)
End Function

```

Conversion Routines

VBA was not designed with CAD in mind, and so it does not easily handle concepts unique to vector drawings, such as the processing of 2D and 3D points. Just as in LISP, VBA must separate coordinate triplets, and then recombine them as strings.

Two of the conversion routines in Mr Janssens's program are useful for any VBA programming with BricsCAD. These are as follows:

- **PointToString** converts 3D points (x, y, z coordinates) to strings, such as 3,2,1 to "3","2","1".
- **StringToPoint** converts strings back to 1D, 2D, or 3D coordinate points, such as "3","2","1" to 3,2,1.

Frankly, I am surprised at the amount of code VBA needs for adding and removing quotation marks from the single, most common, type of CAD data. The good news is that once you write these two routines, you can use them over again in your other VBA programs.

Here are descriptions of how they work.

POINTTOSTRING CONVERSION FUNCTION

The PointToString routine adds quotations marks to each coordinate to convert them from real numbers to strings. For example, 3.4,2,0 becomes "3.4","2","1". It looks like this:

```

Private Function PointToString(vIn As Variant) As String
    Dim sPt As String: sPt = vbNullString
    Dim iPrecision As Integer
    iPrecision = ThisDrawing.GetVariable("LUPREC")
    If VarType(vIn) > vbArray Then
        sPt = StringFromValueFixedDecimal(vIn(0), iPrecision) & ", "
        sPt = sPt & StringFromValueFixedDecimal(vIn(1), iPrecision) & ", "
        sPt = sPt & StringFromValueFixedDecimal(vIn(2), iPrecision)
    End If
    PointToString = sPt
End Function

```

(VBA keywords are shown in **boldface**.)

Let's examine how this code works, line by line.

Private Function PointToString(vIn As Variant) As String

Private means that the function can be accessed only within this module. This is roughly analogous to the practice in LISP where variables names are placed after the slash character to make them local, such as (defun function (/ vaname)).

Function specifies the name, arguments, and code. It is like the **defun** function in LISP.

PointToString is the name of the function.

vIn is the name of the argument's variable (*vIn* is short for "variant input"). The purpose of this variable is to receive the argument passed to this function when it is processed.

As declares the data type of the argument.

Variant is the data type, meaning the function is completely flexible when it comes to data types, working with numbers, text, and arrays.

As String means that the output of the function is a variable length string.

In summary, this line of code defines a local function named "PointToString" that expects numbers or text as input, and then returns text.

QUICK SUMMARY OF VBA PREDEFINED CONSTANTS

Constant	Value	Comments
vbCrLf	Chr(13) + Chr(10)	Carriage-return, linefeed
vbCr	Chr(13)	Carriage-return
vbLf	Chr(10)	Linefeed
vbNewLine	Chr(13) + Chr(10)	New line character (\n)
vbNullChar	Chr(0)	Character with value 0
vbNullString	0	String with value 0; used for external procedures
vbObjectError	-2147221504	Values greater are user-defined error numbers
vbTab	Chr(9)	Tab (\t)
vbBack	Chr(8)	Backspace

Dim sPt As String: sPt = vbNullString

Dim is the most common way of declaring variable names. Unlike LISP, VBA needs to know ahead of time the names of variables and their data types. While to experienced LISP programmers declarations seems like unnecessary extra work, this ahead-of-time declaration is one of the ways that VBA routines run faster than ones written in LISP.

sPt is the name of the variable (*sPt* is short for “string point”).

As is the keyword for declaring data types.

String is the data type.

: (colon) indicates the end of a line label. **sPt** is given its initial value:

vbNullString is one of VBA’s predefined constants — just like **pi** is predefined as 3.1431... in LISP. The value of **vbNullString** is 0 (not the same as a zero-length string, ""). This is done so that the dialog box initially displays 0 when the **LastPoint** contains nothing.

In summary, this line of code defines a variable named "sPt" and assigns it the value of 0.

Dim iPrecision As Integer

iPrecision is the name of another variable (short for “integer precision”). Its purpose is to specify the number of decimal places used by this function.

As Integer defines its data type as an integer, because an integer is large enough to hold the value of decimal places, which in BricsCAD ranges from 0 to 8.

In summary, this line of code defines a variable named "iPrecision."

iPrecision = ThisDrawing.GetVariable("LUPREC")

ThisDrawing is VBA’s way of accessing data from the current drawing — without needing to know its name.

GetVariable gets the value of system variables, and it gets the value of the current drawing. This is like using the (**getvar**) function in LISP.

"LuPrec" is the name of the system variable that stores the value of the current linear units precision (as set by the Setting dialog box). **LuPrec** is a BricsCAD name and has nothing to do with VBA; this means that you can use the same line of code to access the value of any system variable, including those unique to BricsCAD.

In summary, this line of code gets the value of system variable **LuPrec**, and then stores it in **iPrecision**.

If VarType(vIn) > vbArray Then

If starts the usual if-then decision-making construct found in all programming languages. (If they have no “if-then” construct, then they are not programming languages.) In this case, **if** checks the value of **vIn**.

VarType is the function that determines the data type of variables. It returns an integer that reports the data type. Once you know the data type, you can perform other work on it. In this case, it checks the data type of `vIn`.

`>` is the greater than function.

vbArray is another VBA constant; this one carries the value of 8192. However, array types always return a value larger than 8192 in order to report the type of the array. An array can consist of numbers, text, Booleans, and so on. In our program, the array is the coordinate triplet, such as 1,2,3.

In summary, this line of code checks to see if the data type of `vIn` is an array. More specifically, it asks, "Is the value of `vIn` greater than 8192? If so, then it is an array, and processing can continue."

sPt = StringFromValueFixedDecimal(vIn(0), iPrecision) & ", "

StringFromValueFixedDecimal is a user-defined function that converts a decimal into a string, and then simulates the number of decimal points. (It is listed a little later in the *LastPoint.Dvb* program.) It expects two arguments: a decimal number and the precision (ie, the number of decimal points to display).

vIn(0) extracts the first value of array `vIn`. Yup, VBA considers 0 as #1, just as in LISP. If `vIn` is 3.2,2,0, then 3.2 is extracted.

iPrecision specifies the number of decimal points. For example, if `vIn(0)` is 3.2, then this function changes it to "3.2000" (when `iPrecision` is 4) or to "3", when `iPrecision` is 0.

& is VBA's function for *concatenating* (linking together) strings — same as the `StrCat` function in LISP.

`", "` is concatenated to the string, resulting in `sPt` holding the value of "3.2000, ".

In summary, this line of code converts the first element of the coordinate array into a string with a fixed number of decimal points, and then adds a comma and space.

sPt = sPt & StringFromValueFixedDecimal(vIn(1), iPrecision) & ", "

This line of code is identical to the one above, but with two differences:

sPt = sPt & concatenates the existing value of `sPt` ("3.2000, ") with the second value extracted from the array.

StringFromValueFixedDecimal(vIn(1)) extracts the second element from the array.

In summary, this line of code converts the second element of the coordinate array into a string, and then concatenates it to the first element. `sPt` now holds "3.2000, 2.0000, ". You can start to see how the numerical array is being converted, piece by piece, to a string array.

sPt = sPt & StringFromValueFixedDecimal(vIn(2), iPrecision)

The process repeats, with sPt now holding the string "3.2000, 2.0000, 0.000".

End If

End indicates the end of a section.

If indicates the end of the if-then statement. If vIn hadn't been an array, then the routine would have skipped the previous three lines of code, and jumped to here. Can you guess the value sPt would hold in this case?

PointToString = sPt

The value of sPt is assigned to PointToString, where it can be accessed by any other line of code. (If vIn had not been an array, the value of sPt would be 0.)

End Function

End specifies the end of the module.

Function indicates that the function has come to an end. Because this is a subroutine, the value of PointToString is now returned to the main part of the code, where it is used by this statement:

```
Me.txtLastPoint.Text = PointToString(vLastpoint)
```

STRINGTOPOINT CONVERSION FUNCTION

The StringToPoint routine removes quotations marks from each string to convert it to a real number. For example, "3.4, 2, 0" becomes 3.42,0. Some of the code will be familiar to you from above.

```
Private Function StringToPoint(sIn As String) As Variant
    Dim sCoords() As String: sCoords = Strings.Split(sIn, ",")
    Dim tmpPt(0 To 2) As Double

    If UBound(sCoords) = 0 Then
        tmpPt(0) = Val(sCoords(0))

    ElseIf UBound(sCoords) = 1 Then
        tmpPt(0) = Val(sCoords(0))
        tmpPt(1) = Val(sCoords(1))

    ElseIf UBound(sCoords) = 2 Then
        tmpPt(0) = Val(sCoords(0))
        tmpPt(1) = Val(sCoords(1))
        tmpPt(2) = Val(sCoords(2))

    End If

    StringToPoint = tmpPt
End Function
```

Let's examine what some of this code does:

Dim sCoords() As String: sCoords = Strings.Split(sIn, ",")

Dim sCoords() As String defines variable sCoords (sort for "string coordinates"), and assigns a data type of String.

Split splits the string into a one-dimensional array with the specified number of substrings.

"," specifies the delimiter, which tells Split where to make the split. In this case, a string like "3.4, 2, 0" becomes "3.4", "2", and "0".

If UBound(sCoords) = 0 Then

UBound reports the size of an array. It is useful in determining whether the function is dealing with 2D coordinates (a 2-element array) or 3D, a three-element array.

tmpPt(o) = Val(sCoords(o))

Val converts numbers in strings as a numeric value. In short, the "3.4" becomes 3.4.

This subroutine is used by the txtLastPoint_BeforeUpdate function.

LOADING AND RUNNING LASTINPUT.DVB

You can download the *LastInput.Dvb* file from my Dropbox account at <https://www.dropbox.com/s/l3maokh191wke1h/lastinput.dcl?dl=0>.

Follow these steps to load the program:

1. Start BricsCAD.
2. From the **Tools** menu, select **VBA**, and then choose **Load Project**.
3. In the Open dialog box, choose "LastPoint.Dvb", and then click **Open**. The program is now loaded into BricsCAD.
(If the Security dialog box appears, choose **Low**, and then click **OK**.)

To run the program, follow these steps:

1. From the **Tools** menu, select **VBA**, and then choose **Macros**.
2. In the Run BricsCAD VBA Macro dialog box, choose "Module1.main".
3. Click **Run**.
Notice that the Last Input dialog box appears. If the drawing is brand-new (no objects drawing), then the fields report 0.
4. To see the dialog box at work, start the Line command, and then draw a few lines. (This dialog box is non-model, meaning it can stay open even as you execute other commands in BricsCAD.)
5. Click **Update** to see the dialog box report the values of the last point, angle, and prompt.
6. To change the value of Last Point, highlight the coordinates, and then enter different values for x, y, and z.
7. When done, click **OK**. The dialog box disappears.
8. Press **Esc** to cancel the Line command.

TIP To include a VBA project in a toolbar or menu macro, use the **-VbaRun** command, and then provide the macro name as the argument.

QUICK SUMMARY OF VBA VARIABLE DECLARATIONS

Declaration	Comments
Dim	Default method of declaring variables: <ul style="list-style-type: none">• When Dim appears within the procedure, the variable is available only within the procedure.• When Dim appears in the declarations section of the module, the variable is available to all procedures within the module but not to other modules in the project.
Public	Makes variables available to all procedures in all modules in the project.
Private	Restricts variables for use only by procedures in the same module.
Static	Variables retain their values between calls.
Option Explicit	All variables must be explicitly declared within the module.

QUICK SUMMARY OF VBA SHORTCUT REFERENCES

Term	Comments
This	Refers to the current or active BricsCAD document.
Me	Makes variables available to every procedure in a class module. Used when a class has more than one instance, because Me refers to the instance of the class in the code currently being executed.

PART IV

Appendices

Command Summary

THIS APPENDIX, YOU CAN REFERENCE THE NAMES OF OVER 900 COMMANDS IN BRICSCAD.

They are listed alphabetically by name, as well as in groupings of common commands, as follows:

- ai-** commands
- bim-** (building information modeling) commands
- bm-** (BricsCAD mechanical) commands
- Civil** commands ([NEW TO V20](#))
- Cloud-** commands (ex-Chapoo commands)
- Dim-** (dimension) commands
- dc-** (dimensional constraint) commands
- dm-** (direct modeling) commands
- gc-** (geometric constraint) commands
- Layer** Commands
- sm-** (sheet metal) Commands
- VBA** (Visual Basic for Applications) commands
- ViewBase** commands

Commands with a hyphen prefix, such as -Color, are ones that run at the command prompt, and have a complimentary command, such as Color, that displays a dialog box.

Command names new in V20 are shown in [blue](#).

A Commands

About displays information about the program.

AcisIn imports 3D solids in SAT format (SAT is short for “save as text”).

AcisOut exports 3D solids and surface entities in SAT format.

AddInMan displays the VBA COM Add-In Manager dialog box.

AddSelected creates a new entity of the same type as an existing entity.

Align aligns entities with other entities in 2D and 3D space.

AlignSpace adjusts viewport angle, zoom factor, and pan position based on alignment points specified in model space and paper space; operates in paper space only.

AniPath makes movies from views generated by a camera moving through 3D scenes.

AnnReset resets all scale representations to the entity’s original positions

AnnUpdate updates annotative scale factors to match updates made with Style and DimStyle commands.

Aperture sets selection area for snapping to entities.

Apparent toggles Apparent intersection entity snap; snaps to the intersections of entities, even when they only appear to intersect in 3D space.

AppLoad loads DRX, LISP, and SDS applications to run inside BricsCAD; Mac and Linux load only LISP and SDS.

Arc draws arcs.

Area determines the area and perimeter of closed 2D objects; the area and length of open polylines and splines as if they were closed; the lengths only of lines, sketches, arcs, and elliptical arcs; and the areas of faces of 3D objects.

Array and **-Array** creates dynamic polar, path, and rectangular arrays of entities.

ArrayClassic runs the dialog box-based version of the Array command.

ArrayClose and **-ArrayClose** end the array editing session.

ArrayEdit edits entities and source entities of arrays.

ArrayEditExt edits entities in arrays.

ArrayPath distributes entity copies evenly along a path into multiple rows and levels.

ArrayPolar distributes entity copies evenly in a circular pattern about a center point or axis of rotation, using multiple rows and levels.

ArrayRect distributes entity copies into any number of rows, columns, and levels.

AttachmentsPanelOpen opens the Attachments panel for managing Xref, Raster Image, PDF, and Pointcloud attachments.

AttachmentsPanelClose closes the Attachments panel.

AttDef and **-AttDef** defines attributes for blocks.

AttDisp toggles the display of attributes through all, none, or those normally visible.

AttEdit edits the values and properties of attributes.

AttExt and **-AttExt** exports data from attributes to text files.

AttRedef redefines blocks and updates associated attributes.

AttSync synchronizes attribute definitions in all references to a specified block definition.

Audit repairs open drawings in case of data corruption.

AutoComplete sets the options for autocomplete mode on the command line.

Ai Commands

- Ai_Box** draws 3D boxes as mesh surfaces.
- Ai_CircTan** draws a circle tangent to three entities.
- Ai_Cone** draws 3D cones as mesh surfaces.
- Ai_Cylinder** draws 3D cylinders as mesh surfaces.
- Ai_DeSelect** unselects all selected entities.
- Ai_Dish** draws 3D dishes as mesh surfaces.
- Ai_Dome** draws 3D domes (half-spheres) as mesh surfaces.
- Ai_DrawOrder** changes the display order of overlapping entities.
- Ai_Fms** switches to the first layout tab and enters model space of the first viewport.
- AiMleaderEditAdd** adds leader lines to multi-leaders.
- AiMleaderEditRemove** removes leader lines from multi-leaders.
- Ai_Molc** makes the layer current of the selected entity (short for “make object layer current”).
- Ai_MSpace** switches to model tab.
- Ai_PSpace** switches to the first layout tab.
- Ai_Pyramid** draws 3D pyramids as mesh surfaces.
- Ai_SelAll** selects all non-frozen entities in the current space, like Ctrl+A.
- Ai_Sphere** draws 3D spheres as mesh surfaces.
- Ai_TileModel** sets TileMode variable to 1 and then switches to model tab.
- Ai_Torus** draws 3D tori as mesh surfaces.
- Ai_Wedge** draws 3D wedges as mesh surfaces.

B Commands

- Background** displays the Background dialog box for creating background colors and images in shaded modes and renderings.
- Base** changes the drawing's insertion point when it is inserted into other drawings.
- BAttMan** manages the attributes of block definitions (short for Block Attribute Manager).
- BClose** closes the Block Editor
- BEdit** and **-BEdit** open the Block Editor environment
- BHatch** and **-BHatch** fills closed areas with repeating patterns, solid colors, or gradients.
- Blade** opens the LISP editing environment
- BlipMode** enables and disables display of marker blips.
- Block** and **-Block** groups entities into blocks (symbols).
- Blockify** converts entities to blocks to save space and increase speed
- BmpOut** exports the current viewport as a BMP (bitmap) file.
- Boundary** and **-Boundary** draws a polyline that forms a boundary around the inside closed areas.
- Box** draws three-dimensional solid boxes.
- Break** removes portions of entities.
- Browser** opens the default Web browser.

BIM Commands

(Available as an add-on to the Platinum edition only; *bim* = building information modeling)

bimAddEccentricity controls relative positions of the axes in linear solids.

bimApplyProfile applies profiles to linear entities and linear solids.

bimAttachComposition attaches BIM compositions to solids.

bimAttachSpatialLocation locates the drawing in mapping references.

bimClassify classifies an entity as a building element with a name and an internal 'guid' (globally unique identifier).

bimCopy copies entities *normal* (at 90 degrees) to the selected face.

bimCurtainWall creates curtain walls made of planar quadrilateral panels from free-form surfaces.

bimDecompose decomposes composition-based solids into separate plies.

bimDisplayComposition toggles the display of compositions on and off.

bimDrag drags faces of solids; when dragging major faces, it preserves connections with minor faces; when dragging minor faces, it optionally connects minor faces to major faces of other solids.

bimFlip flips the starting face from which the layers of a composition are set out.

bimFlowConnect connects linear solids.

bimGrid creates rectangular and radial grids with automatically-applied labels.

bimIfy automatically classifies and spatial locates the entire bim model.

bimInsert and **-bimInsert** insert windows and doors in solids.

bimLinearSolid creates chains of linear solids.

bimList list names and properties of BIM entities in the current drawing.

bimMultiSelect selects one or more linear solids with coplanar and/or parallel axes based on the initial solid or face selected.

bimPatch reserves an of a BIM model for editing with the RefEdit command.

bimProfiles displays the Profiles dialog box for creating and editing profiles.

bimProjectInfo displays the BIM Project Info dialog box for specifying project library databases.

bimPropagate (replaces *bimSuggest*) maps details from selected solids to all similar solids, as well as on grids.

bimPropagateEdges propagates along the edges of planar solids, such as railings.

bimPropagateLinear propagates connections to linear elements, such as connections to walls and slabs.

bimPropagatePattern propagates a single element (such as a switch) to multiple locations and grids.

bimPropagatePlanar propagates connections to planar elements, such as walls, slabs, and roofs.

bimProperties displays the BIM Properties dialog box for specifying and editing properties of bim projects.

bimQuickDraw draws rooms and stories from rectangles and L-shapes.

bimRecalculateAxis recalculates the axes of structural elements back to their centroids.

bimRoom defines room areas with markers.

bimRoomBoundingElements determines which elements (walls, floors, etc) determine bounds of rooms.

bimSchedule generates linked schedule tables after analyze building elements in BIM models.

bimSection creates BIM section entities.

bimSectionOpen opens the drawing file related to a BIM section entity; or the 3D BIM model related to a BIM section drawing.

bimSectionUpdate updates and exports BIM sections.

bimSetReferenceFace controls the layout of plies through reference and opposing faces.

bimSplit splits segmented solids into separated solids automatically; splits solids using cutting faces.

bimStair creates stairs between two floors (as a rectangular parametric array).

bimStretch stretches BIM entities.

bimStructuralConnect connects linear solids.

bimTag tags BIM sections.

bimUpdateRoom updates data about the selected room.

bimUpdateThickness re-applies the overall thickness of a composition to the solid.

bimWindowCreate replaces closed entities with parametric window entities; displays the choose window style dialog box.

bimWindowPrint prints a specified area of the BIM model.

bimWindowUpdate updates openings made by windows or doors in solids in case the opening did not updated correctly automatically.

ClipDisplay toggles the clipped display property of a section plane or a BIM section entity.

BricsCAD Mechanical Commands

(Available in Platinum edition only; bm = BricsCAD mechanical)

bmBalloon associates balloon with assembly components in Model space and in generated views in layouts.

bmBom inserts bill of material (BOM) tables in the current drawing.

-bmCreateComponent creates a component from a selection set; add it to the library.

bmDependencies lists all files, containing component definitions inserted in the assembly, in the command window.

bmDissolve dissolves a mechanical component inserted in the current drawing.

bmExplode creates a block of an exploded representation of an assembly.

bmExplodeMove allows users to created exploded representations of assemblies.

bmExternalize converts local components to external components.

bmForm creates a new mechanical component and inserts it into the current drawing; if necessary, run bmMech to initialize the mechanical structure in the current drawing.

bmHardware and **-bmHardware** insert standard hardware parts as a mechanical component in the current drawing.

bmHide hides the visibility of mechanical components; hidden inserts are taken into account by commands such as bmBom and bmMassProp.

bmInsert and **-bmInsert** insert an existing mechanical component as a virtual component into the current drawing.

bmLispGet retrieve variables for blocks and parameters of components.

bmLocalize converts external components to local components.

bmMassProp computes mass properties for the current model using densities assigned to the components (defined by the density property of the components and subcomponents).

bmMech converts the current drawing into a mechanical component.

bmNew creates a mechanical component as a new drawing file.

bmOpen opens the source drawing of external mechanical components.

bmOpenCopy opens a copy of a component insert as a new drawing.

-bmParameters lists and edits parameters of inserted components.

bmRecover recovers broken mechanical structures.

bmReplace replaces a component insert.

bmShow shows previously hidden mechanical components.

bmUnlink breaks links between components.

bmUnmech converts the current mechanical component into a plain drawing.

bmUpdate reloads all referenced components from external files and updates BOM tables.

bmVStyle applies visual styles to mechanical component inserts.

bmXConvert converts X-Hardware solids in the current drawing to mechanical components.

C Commands

Cal displays the operating system's Calculator program.

Callout places callouts; can be used only from the SheetSet panel.

Camera changes the viewpoint to perspective.

Center toggles Center entity snap; snaps to the center of circles, arcs, and other circular entities.

CenterDisassociate disassociates center lines and marks from circles and arcs

Centerline places associative center lines on circles and arcs

Centermark places associative center marks on circles and arcs

CenterReassociate reassociates centerlines/marks with circles and arcs

CenterResets resets centerline and mark entities, if moved

Chamfer bevels entities.

Change changes the position and properties of entities: endpoint, color, elevation, layer, linetype, linetype scale, lineweight, and thickness.

ChProp changes just the properties of entities.

ChSpace moves entities from paper space to model space and vice versa.

Circle draws circles.

CleanScreenOn hides most user interface elements to maximize the drawing area.

CleanScreenOff restores the user interface to its default configuration.

CleanUnusedVariables clears unused variables from memory.

Close exits the current drawing, but not the program.

Color and **-Color** specifies the color for entities.

CommandLine and **CommandLineHide** open and close the command bar.

Commands reports the names of all commands supported by the program.

CommunicatorInfo reports the status of the Communicator add-on

ComponentsPanelOpen opens the Components panel for accessing symbols

ComponentsPanelClose closes the Components panel

Cone draws three-dimensional solid cones.

ContentBrowserClose and **ContentBrowserOpen** close and open the Content Browser panel.

ConvertCtb converts older CBT (color-based plot tables) files to newer STB (style-based plot tables) files.

ConvertOldLights converts old light definitions to the current format.

ConvertOldMaterials converts old material definitions to the current format.

ConvertPoly converts lightweight polylines to classic polylines (2D polylines) and vice versa.

ConvertPStyles converts drawings to from CTB (color-based plotting) to STB (plot styles).

ConvToMesh converts 3D solids and surfaces to mesh objects

ConvToSolid converts watertight meshes, circles, and closed polylines to 3D solids

ConvToSurface converts 3D solids, open polylines and other entities to 3D surfaces

Copy duplicates entities.

CopyBase copies entities with a specified reference point to the Clipboard.

CopyClip copies entities to the Clipboard.

CopyEData Copies extended entity data from one entity to others.

CopyGuided copies entities along guidelines.

CopyHist copies the command history to the Clipboard.

CopyToLayer copies selected entities to another layer.

CPageSetup edits the page setup of the current layout or model space.

CuiLoad and **CuiUnload** load and unload CUI and CUIX (user interface customization), MNU (menu), MNS (LISP code), and ICM (IntelliCAD menu) files.

Customize customizes user interface elements, such as menus, toolbars, and shortcuts.

CutClip copies entities to the Clipboard and deletes the entities.

Cylinder draws three-dimensional solid cylinders.

Civil Commands

(Civil engineering commands are part of Platinum)

Alignment creates horizontal and vertical alignments typically used to design roads.

AlignmentEdit edits horizontal and vertical alignments.

AlignmentView views alignment along TIN surfaces.

AlignmentVInitial creates vertical alignments.

Grading interactively modifies TIN surfaces to create grading effects, such as for roads and foundations.

LandXmlEexport exports the drawing in LandXML format

LandXmlImport imports LandXML files into the current drawing

Tin (triangulated irregular network) imports data from TIN files to create land surfaces, and converts Civil 3D surfaces to the BricsCAD format

TinEdit adds and removes points, break lines, and boundaries in TIN surfaces.

TinExtract creates a mesh or 3D solid between TIN surfaces or between a TIN surface and elevation.

TinMerge combines two or more TIN surfaces into one.

TinModify deforms and smooths selected TIN surfaces.

TinVolume creates a TIN volume surface between a base and comparison TIN surfaces or an elevation.

Cloud Commands

All *Chapoo*- commands were renamed *Cloud*- in V18

CloudAccount reports the status of the 24/7 account at the command bar.

CloudDownload downloads drawings from the 24/7 project to a local folder.

CloudLogoff logs off from the 24/7 project.

CloudLogon logs on to 24/7.

CloudOpen opens a drawing after downloading it from 24/7.

CloudProject opens the 24/7 project in the default browser.

CloudUpload uploads the current drawing to 24/7.

CloudWeb connects to the 24/7 website at <https://www.bricsys.com/en-intl/247/>.

D Commands

DataExtraction exports entity properties, block attributes and drawing information to CSV (comma separated values) file.

DataLink imports Excel spreadsheets and CSV files as linked table entities

DataLinkUpdate updates the data linked between a table and an external file

DbList lists information about all entities in the drawing (short for “database listing”).

DdAttE edits the values of attributes through a dialog box (short for “dynamic dialog attribute editor”).

DdEdit edits single-line text, multi-line text, attribute definitions, and attribute text (short for “dynamic dialog editor”).

DdEModes sets default values for creating entities (short for “dynamic dialog entity modes”).

DdFilter creates a selection set of the entities selected.

DdGrips specifies the properties of grips through the Settings dialog box.

DdPType specifies the look and size of point entities, through the Settings dialog box (short for “dynamic dialog point type”).

DdSelect specifies the properties for selecting entities, through the Settings dialog box.

DdSetVar displays the Settings dialog box to change the values of variables.

DdSTrack Sets the properties for snap tracking, through the Settings dialog box (short for “snap tracking”).

DdVPoint sets 3D viewpoints or plan view.

DefaultScaleList displays the Scale List Edit dialog box to edit the default scale factors

DesignTable creates new design tables for the Mechanical Browser.

-DesignTableEdit configures, replaces, exports, and deletes design tables at the command line.

Delay delays execution of the next command; for use with scripts only.

DeIEdData deletes extended entity data from the selected entity (short for “delete entity data”).

DgnImport imports Microstation design files and converts them to entities.

DgnImportOptions opens the Settings dialog box at the DgnImport section.

Dish draws dishes (bottom half-spheres) from polygon meshes.

Dist reports the distance and angle between two points.

Distantlight places distant lights.

Divide places points or blocks along entities.

Dome draws domes (top half-sphere) from polygon meshes.

Donut draws circular polylines with width.

Drag moves faces.

DragMode controls the appearance of objects while being dragged.

DrawOrder changes the display order of overlapping entities.

DrawOrderByLayer controls the draw order of overlapping objects through layer names.

DSettings displays the Settings dialog box for drafting settings (short for “drafting settings”).

DstConvert converts sheetset DST files to XML format.

DView changes the 3D viewpoint interactively, and turns on perspective mode (short for “dynamic view”).

DwgCodePage changes the code page for text in drawings.

DwgCompare compares differences between two drawings, and visually merges drawings.

DwgProps opens the Drawing Properties dialog box, showing the general information and user defined properties stored with a drawing.

Dxfln and **DxfOut** imports DXF files (short for “drawing exchange format”) and exports drawings in ASCII or binary DXF format.

Dimensioning Commands

(Dim = dimension)

Ai_Dim_TextAbove moves text above the dimension line.

Ai_Dim_TextCenter centers text on the dimension line.

Ai_Dim_TextHome moves text to its home position, as defined by the dimension style.

AiDimFlipArrow mirrors arrowheads on dimension lines.

AiDimPrec changes the precision of dimension text.

AiDimStyle creates dimension styles from a selected dimension.

Dim places and edits dimensions at the ‘Dimensioning command:’ prompt.

DimI executes a single dimension command at the ‘Dimensioning command:’ prompt.

DimAligned draws dimensions parallel to (aligned with) selected entities; works with lines, polylines, arcs, and circles.

DimAngular dimensions angles.

DimArc places arc length dimensions.

DimBaseline places multiple linear or angular dimensions starting at the same base point; command can only be used when at least one other dimension is already in the drawing.

DimCenter places center marks at the center points of circles and arcs.

DimContinue continues linear and angular dimensions from the endpoint of the previous dimension.

DimDiameter dimensions the diameter of circles and arcs, and places a center mark.

DimDisassociate removes associativity from selected dimension entities.

DimEdit changes wording and angle of dimension text; changes the angle of extension lines.

DimLeader draws leaders.

DimLinear places linear dimensions horizontally, vertically, or rotated.

DimOrdinate measures x and y ordinate distances from a common origin, specified by the current UCS origin.

DimOverride overrides the values of the current dimension style.

DimRadius dimensions the radii of arcs and circles.

DimReassociate reassociates or associates dimensions to entities or points on entities.

DimRegen updates associative dimensions (short for “dimension regeneration”).

DimStyle and **-DimStyle** creates and modifies dimension styles through the Drawing Explorer.

DimStyleSet reports the current dimension style in the command bar.

DimTEdit changes the position of dimension text.

Dimensional Constraint Commands

(*dc = dimensional constraint*)

CleanUnusedVariables purges variables not used by constraint expressions and not linked to dimensions.

dcAligned constrains the distance between two defining points on entities.

dcAngular constrains the angle between three constraint points on entities; or between two lines; or between two polyline segments; or constrains the angles of arcs or polyline arcs.

dcConvert converts an associative dimension to a dimensional constraint.

dcDiameter constrains the diameters of circles, arc, or polyline arcs.

dcDisplay shows and hides dimensional constraints.

dcHorizontal constrains the horizontal distance between two defining points on entities.

dcLinear constrains horizontal or vertical distance between two defining points on entities.

dcRadial constrains the radius of circles, arcs, or polyline arcs.

dcVertical constrains the vertical distance between two defining points on entities.

DelConstraint removes all dimensional (and geometrical) constraints from an entity.

DimConstraint applies a dimensional constraint to an entity or between constraint points on entities; converts associative dimensions to dynamic dimensions.

Direct Modeling Commands

(*Available for Pro or Platinum editions only; dm = direct modeling*)

dmAngle3D applies angle constraints between the faces of a solid or of different solids.

dmAudit checks and fixes 3D models.

dmAuditAll also checks and fixes 3D ACIS models in externally-referenced drawings

dmChamfer creates an equal distance chamfer between adjacent faces.

dmCoincident3D applies coincident constraints between two edges, two faces, or an edge and a face of two different solids.

dmConcentric3D applies concentric constraints between two cylindrical, spherical, or conical surfaces.

dmConstraint3D applies geometric relationships and dimensional constraints between sub-entities (such as faces, surfaces, and edges) of 3D entities.

dmCopyFaces copies features like holes and ribs to the same or other 3D solids

dmDeformCurve deforms one or more connected faces of a 3D solid/surface by replacing their edges with given curves.

dmDeformMove deforms one or more connected faces of a 3D solid/surface by moving and rotating their edges.

dmDeformPoint deforms as smoothly as possible (using G1 or G2 continuity) a region, one or more connected faces of a 3D solid or a surface by moving a point lying on one of them in arbitrary 3D direction.

dmDelete deletes faces and solids.

dmDistance3D applies a distance constraint between two sub-entities of a solid or of different solids.

dmExtrude creates 3D solids by extruding closed 2D entities, regions or closed boundaries.

dmFillet creates a smooth fillet between adjacent faces sharing a sharp edge.

dmFix3D applies a fixed constraint to a solid or to an edge or a face of a solid.

dmGroup creates new groups, edits them, and dissolves groups.

dmMove moves the selected solids, or faces or edges of a solid using a vector.

dmParallel3D applies a parallel constraint between two faces of a solid or of different solids.

dmPerpendicular3D applies a perpendicular constraint between two faces of a solid or of different solids.

dmPushPull adds or removes volume from a solid by moving a face.

dmRadius3D applies a radius constraint to cylindrical surfaces or circular edges.

dmRepair fixes inconsistencies in 3D geometry supported by ACIS kernel (3D solids, surfaces).

dmRevolve creates 3D solids by revolution of closed 2D entities or regions about an axis.

dmRigidSet3D defines a set of entities or sub-entities as a rigid body.

dmRotate rotates faces of a solid around an axis.

dmSelect selects edges and faces of 3D solids or surfaces based on their geometric properties.

dmSelectEdges selects faces and edges of 3D solids.

dmSimplify simplifies the geometry and topology of 3D solid entities by removing unnecessary edges and vertices, merges seam edges, and replaces the geometry of faces and edges by analytic surfaces and curves, if possible within the user-specified tolerance. Run this command on imported 3D solid geometry.

dmSimplifyAll also unnecessary elements in externally referenced drawings

dmStitch converts a set of region and surface entities that bound a watertight area to a 3D solid.

dmTangent3D applies a tangent constraint between a face and a curved surface of different solids.

dmThicken creates 3D solids by thickening (i.e. adding thickness to) surfaces, their faces, and faces of 3D solids.

dmTwist twists 3D solids, surfaces, and regions by an angle.

dmUpdate forces 3D constraints to update.

E Commands

EAttEdit edits the value and most properties of attributes (short for “enhanced attribute editor”).

EdgeSurf creates a 3D Coons mesh surface patch between four lines, forming a closed shape (short for “edge surface”).

EditEData creates and edits extended entity data (short for “edit entity data”).

Elev changes the default elevation and thickness.

Ellipse draws ellipses and elliptical arcs.

EndCompare ends the drawing compare session

Endpoint toggles endpoint entity snap; snaps to the ends of open entities, such as line, arcs, and open polylines.

Erase erases selected entities from drawings; alternatively, press the Del key.

eTransmit creates a package of a drawing file and all its dependencies, such as external references, images, font files, plot configuration files, plot style tables and font map files.

ExpBlocks opens the Blocks section of the Drawing Explorer dialog box (short for “explorer blocks”).

ExpFolders opens the Drawing Explorer on the Folders tab.

ExpImages opens the Drawing Explorer at the Images section.

ExpLayers opens the Drawing Explorer at the Layers section.

Explode breaks complex objects into their component entities.

Explorer opens the Drawing Explorer dialog box, which controls Layers, Layer States, Linetypes, Multiline Styles, Multileader Styles, Text Styles, Dimension Styles, Table Styles, Coordinate Systems, Views, Visual Styles, Lights, Materials, Render Presets, Blocks, External References, Images, PDF Underlays, Dependencies, Page Setups, and Section Planes.

Export saves entities in other file formats.

ExportLayout exports visible objects from the current layout to model space of new drawings.

ExportPDF exports the current layout to a PDF file.

ExpPdfs opens the Drawing Explorer at the PDF section.

ExpUcs creates, modifies, deletes named UCSes through the Drawing Explorer (short for “explore user-defined coordinate systems”).

ExpXrefs opens the Drawing Explorer at the XRefs section.

Extend extends entities to bounding edges defined by other entities.

Extension toggles extension entity snap, which snaps to the point where a line extended would intersect another entity.

Extrude extrudes closed entities as 3D solids and open ones as 3D surfaces.

F Commands

FbxExport and **-FbxExport** export 3D models in FBX format for rendering programs

Field inserts text that is updated automatically when system variables change.

FileOpen opens drawing (DWG), template (DWT), and interchange (DXF) files from the command line.

Files opens the operating system’s file manager, such as Windows Explorer or Finder.

Fill fills areas with a solid color or color gradient

Fillet rounds entities.

Find finds and replaces text in notes, annotations, and dimension text.

Flatshot creates a hidden line representation of all 3D solids in model space as a block or a new drawing.

Flatten flattens 2D objects with thickness and allows to convert splines to polylines.

G Commands

GCE snaps the the geometric center of entities.

GenerateBoundary creates closed polylines from faces of 3D solids, as well as from boundaries detected when the Enable Boundary Detection of SelectionModes is activated.

GeographicLocation sets the geographic location of the drawing.

GoToStart displays the Start tab.

Gradient and **-Gradient** fill closed areas with gradient fills of one or two colors.

GradientBkgOff and **GradientBkgOn** turn off and on the gradient displayed in the working area.

GraphScr switches from the text windows to the graphics windows (short for “graphics screen”).

Grid turns the grid display on or off and sets other grid options.

Group and **-Group** creates and modifies named groups of entities.

Geometric Constraint Commands

(For 3D constraints, see *Direct Modeling Commands* section; gc = geometric constraints)

ConstraintBar shows, hides, and resets the display of geometric constraint icons.

DelConstraint removes all geometrical (and dimensional) constraints from an entity.

gcCenter snaps to the centroid of closed entities.

gcCoincident constrains points on entities coincidentally; or constrains a point on an entity to another entity.

gcCollinear constrains lines collinearly.

gcConcentric constrains the center points of arcs, circles, ellipses, and/or elliptical arcs to be coincident.

gcEqual constrains lines to have the same length, or arcs and circles to have the same radius.

gcFix constrains points on entities to fixed positions.

gcHorizontal constrains lines or linear polyline segments, or pairs of points on entities to be parallel to the x axis in the current coordinate system.

gcParallel constrains two lines or linear polyline segments to be parallel to each other.

gcPerpendicular constrains two lines or linear polyline segments to be perpendicular to each other.

gcSmooth constrains a spline to be fluidly continuous to another spline, or arc, or line, or polyline.

gcSymmetric constrains two entities, or two points on entities, to be symmetric about a line of symmetry.

gcTangent constrains one entity tangent to another.

gcVertical constrains lines or linear polyline segments, or pairs of points on entities to be parallel to the y axis in the current coordinate system.

GeomConstraint acts as a universal command that applies all available geometric constraint points.

H Commands

Hatch and **-Hatch** fills a selected boundary with a pattern.

HatchEdit and **-HatchEdit** edits hatch patterns and gradient fills.

HatchGenerateBoundary generates a boundary around a hatch or gradient fill.

HatchGripEdit adds and removes grips from hatches and gradients.

HatchToBack sets the draw order of all hatch entities in the drawing to display behind all other entities.

Helix draws 2D spirals or 3D helices.

Help displays online help.

HelpSearch prompts for searching through the help files at the command prompt.

Hide removes hidden lines from 3D entities until the **UnisolateObjects** command is used.

HideObjects temporarily hides selected entities.

Hyperlink and **-Hyperlink** adds hyperlinks to entities or modifies existing hyperlinks.

HyperlinkOptions controls the display of the hyperlink cursor, shortcut menu, and tooltips.

I Commands

Id reports the x,y,z coordinates of a picked point.

Image inserts raster images in drawings through the Drawing Explorer.

ImageAdjust adjusts the properties of images through the Properties palette.

ImageAttach and **-ImageAttach** attaches raster images to the drawing like xrefs.

ImageClip clips images.

ImageFrame toggles the frame around images.

ImageQuality determines the display quality of images attached to the drawing.

Import displays a dialog box for importing files into the drawing: DWG, DXF, DWT, and DAE (Collada) files. Platinum edition also imports IFC and SKP (SketchUp) files. Additional formats are imported with the optional Communicator module.

Imprint imprints 2D entities onto planar faces of 3D solids and surfaces; allows to create additional edges on planar faces.

Insert and **-Insert** inserts blocks or another drawing into the current drawing.

InsertAligned inserts blocks repeatedly, and inserts mirrored blocks.

InsertGuided and **-InsertGuided** inserts blocks along guide curves.

Insertion toggles Insertion entity snap; snaps to the insertion point of text and blocks.

InsertObj displays data from other programs in drawings, such as text documents, spreadsheets, and images.

Interfere checks interferences between solid models.

Intersect creates regions or 3D solids from the intersection of regions or 3D solids.

Intersection toggles Intersection entity snap; snaps to the intersections of entities.

IsolateObjects hides all other entities from view.

Isoplane controls the isometric plane (left, right, or top) when isometric snap is used.

J Command

Join joins lines, lwpolylines, 2D polylines, 3D polylines, circular arcs, elliptical arcs, splines and helices at common endpoints.

K Command

KeepMe visually merges drawings during the DrawingCompare command

L Commands

Layer: see Layer Commands below.

Layout creates, copies, renames, and deletes layouts.

LayoutManager displays the Layout Manager dialog box for creating, naming, and reordering sets of layouts

LConnect creates connections between faces of two solids

Leader draws leader lines that connect annotations to drawing entities.

Lengthen changes the length of open objects, such as lines and arcs.

LicenseManager provides access to all Bricsys software licenses, as shown below.

LicEnterKey enters the license key number (short for “licence enter key”).

LicProperties reports the BricsCAD license information; modifies and deactivates single user and volume license keys.

LicPropertiesCommunicator reports license information for the optional extra-cost Communicator add-on.

Light places lights in drawings.

LightList displays the lighting palette.

Limits sets the extents of the drawing and the grid.

Line draws straight line segments.

LineStyle and **-LineStyle** creates, loads, and sets linestyles.

List lists the properties of selected entities at the command line.

LiveSection toggles the Live Section property of a section plane.

Load loads compiled SHX shape files into the drawing.

Loft creates 3D solids passing through two or more cross sections.

LogFileOff and **LogFileOn** turn off and on log file recording.

LWeight sets lineweight options.

Layer Commands

LayCur moves the selected entities to the current layer.

Layer and **-Layer** controls layers and layer properties.

LayerP undoes previously applied changes to layer settings when LayerPMode is on (short for “layer previous”).

LayerPMode controls the tracking of changes made to layer settings.

LayersPanelClose and **LayersPanelOpen** closes and open the Layers panel.

LayerState saves and restores the properties of layers.

LayFrz and **LayThw** freeze and thaw the layers associated with entities selected in the drawing.

LayIso and **LayUnIso** isolate and restore layers associated with entities selected in the drawing; locks or turns off all other layers (short for “layer isolate”).

LayLck and **LayUlck** lock and unlock the layers of selected entities.

LayMCur changes the working layer to that of a selected entity (short for “layer make current”).

LayOff and **LayOn** turn off and on layers associated with entities selected in the drawing; off layers cannot be seen.

M Commands

Mail attaches the current drawing to a new message with your computer’s default email client.

Manipulate launches the widget for rotating, copying, moving, mirroring, and scaling entities.

MapConnect sets up a connection with a Web Map Service, after the GeographicLocation command defines the geographic location in the drawing.

MassProp reports the area, perimeter, and other mathematical properties of 3D solids and 2D regions (short for “mass properties”).

MatBrowserClose and **MatBrowserOpen** close and open the materials browser.

MatchPerspective changes the viewpoint in perspective mode to match a background image.

MatchProp assigns the properties of one entity to one or more other entities (short for “match properties”).

MaterialAssign assigns materials from the Material Browser onto 3D objects

MaterialMap maps material definitions onto the surfaces of objects, with presets for boxes, planes, spheres, and cylinders.

Materials creates materials and edits their properties through the Drawing Explorer.

MatLib displays the Rendering Materials panel.

Measure places points or blocks along entities.

MechanicalBrowserClose closes the Mechanical Browser panel.

MechanicalBrowserOpen displays the Mechanical Browser panel.

Menu loads menu files to modify the user interface.

MenuLoad and **MenuUnload** load and unload CUIX and CUI (user interface customization), MNU (menu), MNS (LISP code), and ICM (IntelliCAD menu) files.

Midpoint toggles Midpoint entity snap; snaps to the middle of lines, arcs, and other open entities.

MInsert inserts a block as a rectangular array; combines the -Insert and Array commands (short for “multiple insertion”).

Mirror draws mirror image copies of entities.

Mirror3D draws mirror images of entities about a plane in 3D space.

MLeader creates multileader entities using the current multileader style.

MLeaderAlign aligns multiple leaders

MLeaderCollect collects multiple leader blocks

MLeaderEdit adds leader lines to and removes leader lines from a multileader entity.

MLeaderEditExt adds and removes leader lines, adds and removes vertices from a multileader entity.

MLeaderStyle creates and manages multileader styles through the Drawing Explorer.

MLine draws multilines.

MLineStyle creates and edits multiline styles.

ModelerProperties and **-ModelerProperties** controls the various settings of the ACIS modeler through the Settings dialog box.

Move displaces entities a specified distance in a specified direction.

MoveEData moves extended entity data from one entity to another.

MSlide makes SLD (slide) files from the current view.

MSpace switches to model space inside a viewport of layout tab.

MText and **-MText** opens the multi-line text editor interface for placing paragraph text.

Multiple command prefix forces commands to repeat themselves automatically.

MView creates viewports in layout tab.

MvSetup prepares sets of paper space viewports; superseded by the ViewBase command.

MTP snaps to the midpoint between two points.

N Commands

Navigate walks and flies through 3D models.

Nearest toggles Nearest entity snap mode; snaps to the nearest geometry on entities.

NetLoad loads .NET applications.

New starts new drawing files.

NewSheetSet creates a new sheet set.

NewWiz starts new drawings with the New Drawing Wizard.

Node toggles Node entity snap mode; snaps to point entities.

None turns off all entity snap modes.

Number adds incremented number tags for BIM entities

O Commands

ObjectScale and **-ObjectScale** adds or removes supported scales for annotative entities.

Offset offsets linear entities in parallel orientation.

OleLinks adjusts links of OLE entities embedded in or linked to drawings (short for “object linking and embedding”).

OleOpen opens OLE objects for modification.

OnWeb opens the Bricsys home page in your computer’s default Web browser.

Oops un-erases the last erased entity, including those erased by the Block command.

Open opens an existing drawing file.

OpenSheetSet and **-OpenSheetSet** open an existing sheet set.

Options configures program operating parameters.

Orthogonal constrains the pointer so it moves parallel to the axes of the current coordinate system.

OSnap and **-OSnap** sets entity snaps through the Settings dialog box or the command line (short for “object snap”).

Overkill and **-Overkill** deletes duplicate entities and overlapping lines, arcs or polylines and unifies partly overlapping or contiguous ones.

P Commands

PageSetup creates and edits page setups for plotting drawings in the Drawing Explorer.

Pan and **-Pan** moves the drawing display in the active view tile.

Panelize command draws freeform surfaces as subdivision meshes, optionally planarizing the panels.

Parallel turns on parallel entity snap.

-Parameters create and edit constraint expressions and values.

ParametersPanelOpen opens the Parameters panel.

ParametersPanelClose closes the Parameters panel.

Parameterize adds constraints and parameters to models automatically.

ParametricBlock creates a parametric block from entities in the drawing; useful for BEdit.

PasteBlock inserts data from the Clipboard as block.

PasteClip inserts data from the Clipboard.

PasteOrig pastes entities from the clipboard at the coordinates from the source drawing.

PasteSpec pastes entities from the clipboard, after the user specifies the format.

PdfAdjust adjust the fade, contrast and monochrome settings of PDF underlays.

Pdfattach and **-PdfAttach** attaches PDF files as underlays into the drawing.

PdfClip clips PDF underlays.

PdfImport And **-PdfImport** imports PDF files and converts them to drawing entities.

PdfLayers controls the display of layers in PDF underlays.

PdfOptions controls the exporting of drawings in PDF format through the Settings dialog box.

PEdit edits polylines, 3D polylines, and 3D meshes (short for “polyline edit”).

PEditExt edits vertices and segments of a polyline.

Perpendicular toggles perpendicular entity snap mode.

PFace draws 3D multi-sided meshes; meant for use by programs (short for “polyface mesh”).

Plan sets plan view to construction plane.

PLine draws polyline lines, arcs, and splines with optional width (short for “polyline”).

Plot and **-Plot** both execute the plot command at the command line.

PlotStamp specifies a header and footer for plotted output.

PlotStyle sets the current plot style; works only when plot styles are enabled in drawings.

PlotterManager creates customized parameter PC3 files for printers and other output devices; executes the PlotConfig.exe utility program.

Point draws point entities.

PointCloud displays the Point Cloud section of the Drawing Explorer

PointCloudAttach and **-PointCloudAttach** attach Bricsys-format point cloud files to the current drawing

PointCloudColorMap changes the colors of point based on their elevation.

PointCloudCrop and **PointCloudUncrop** crop the extents of the current point cloud, and undo the cropping.

PointCloudPointSize specifies the size of points in a point cloud.

PointCloudPointSize_Minus decreases the size of points in a point cloud.

PointCloudPointSize_Plus increases the size of points in a point cloud.

PointCloudPreprocess and **-PointCloudPreprocess** convert ASCII PTS, PTX, LAS, and other cloud files into the compressed binary Bricsys file format.

PointLight places point lights in drawings.

Polygon draws equi-sided polygons from polylines of 3 to 1,024 sides.

PolySolid creates 3D wall-like solids.

Preview shows a preview before printing the drawing.

Print plots the drawing to a plotter, printer, or file.

ProfileManager sets current, create, copy, delete, import and export user profiles.

ProjectGeometry projects geometry like curves, and edges onto regions, surfaces, and 3D solids.

Properties displays the Properties palette to change drawing entity properties.

PropertiesClose closes the Properties palette.

PSetupIn and **-PSetupIn** imports page setup definitions from another drawing.

PSpace switches from model to paper space (short for “paper space”).

Publish and **-Publish** prints sheet lists of model space or paper space layouts; saves a sheet list to a file.

Purge and **-Purge** remove unused named entities from drawings, such as unused layers and linetypes.

Pyramid draws three-dimensional solid pyramids.

Q Commands

QLeader draws leaders; specifies properties through a dialog box.

QNew opens new drawings in BricsCAD (short for “quick new”).

QPrint prints the drawing with the default plot configuration, without displaying the Print dialog box (short for “quick print”).

QSave saves the drawing without displaying the Save dialog box (short for “quick save”).

QSelect composes a selection set using filters.

QText toggles the display of text as rectangles (short for “quick text”).

Quadrant toggles snaps to quadrant points of circles, arcs, and polyarcs.

Quick toggles snaps to the first entity geometry found; used together with at least one other entity snap mode.

Quit ends BricsCAD; optionally saves unsaved drawings.

R Commands

Ray draws semi-infinite construction lines

ReAssocApp associates extended entity data with applications (short for “reassociate application”).

Recover repairs damaged drawings.

RecScript records keystrokes to an SCR file for playback with the Script command (short for “record script”).

Rectang draws a rectangular polyline.

Redefine restores built-in commands that have been undefined using the Undefine command.

Redo reverses the effects of a previous U command.

Redraw refreshes the display of the active view tile.

RedrawAll refreshes the display of all currently-open view tiles.

RedSdkInfo reports on rendering related hardware and driver specifications (short for “Red software development kit information”).

RefClose closes the in-situ block and xref editor.

RefEdit and **-RefEdit** edits blocks and externally-referenced drawings (short for “reference editor”).

RefSet adds and removes entities from the block or external reference being edited.

Regen regenerates the current viewport.

RegenAll regenerates all viewports.

RegenAuto determines when BricsCAD regenerates the drawing automatically.

Region converts an entity enclosing an area into a region.

ReInit reloads the PGP alias file (short for “re-initialize”).

Rename and **-Rename** changes the names of objects.

Render and **-Render** generates photorealistic renderings of 3D models using materials and lights.

RenderPresets creates and edits rendering presets, and to set the current render preset.

ReportPanelClose and **ReportPanelOpen** close and open the Report panel.

ResetAssocViews removes associative data from blocks

ResetBlock resets dynamic blocks to their default values.

Resume resumes an interrupted script.

RevCloud draws revision clouds commonly used for red-lining drawings.

Revolve draws 3D solids or surfaces by revolving 2D objects about an axis.

RevSurf creates 3D mesh surfaces by revolving open entities around a axis (usually a line).

Ribbon displays the ribbon user interface.

RibbonClose closes the ribbon.

Rotate rotates entities about a base point.

Rotate3D moves entities about a 3D axis.

RScript reruns the currently loaded SCR script file (short for “repeat script”).

RtLook moves the viewpoint through a 3D scene (short for “real time looking”).

RtPan pans the view in real time.

RtRot, **RtRotCtr**, or **RtRotF** rotate the viewpoint in real time.

RtRotX, **RtRotY**, or **RtRotZ** rotates the 3D viewpoint about the x, y, or z axis in real time.

RtUpDown tilts the viewpoint up, down, left, or right in real time.

RtWalk walk lefts, right, forward or backward through 3D scenes in real time.

RtZoom zooms into the drawing in real time.

RuleSurf draws ruled surfaces between two curves.

S Commands

Save saves the drawing under the current file name or a specified name.

SaveAll saves all open drawings.

SaveAs saves an unnamed drawing with a file name or renames the current drawing.

SaveAsR12 saves drawings in DWG R12 format.

SaveFileFolder opens the File Explorer to the folder in which the current drawing is saved.

Scale enlarges or reduces specified entities equally in the X,Y, and Z directions.

ScaleListEdit and **-ScaleListEdit** edits the list of scale factors used by annotative scaling, sheet scales, and plot scales.

Script loads and runs SCR script files.

Scrollbar toggles the display of the horizontal and vertical scroll bars.

Section creates a cross section based on the intersection of a plane and 3D solids.

SectionPlane creates a section entity that creates sections of 3D solids.

SectionPlaneSettings defines the properties of section plane entities in the Drawing Explorer.

SectionPlaneToBlock saves the selected section plane as a 2D cross section / elevation block or a 3D cutaway section block.

Security determines whether VBA macros can run automatically; not available in the 64-bit version.

SecurityOptions sets a password to protect the drawing.

Select places selected entities in the 'Previous' selection set.

SelectAlignedFaces selects all faces in a model which are coplanar with a selected face.

SelectAlignedSolids selects all solids in a model of which a face is coplanar with a selected face.

SelectConnectedFaces selects all faces in a model which are connected to a selected face.

SelectConnectedSolids selects all solids in a model which are connected to a selected face.

SelectSimilar selects entities of the same type and properties.

SelGrips prompts to select entities and then displays grips.

Settings displays the Settings dialog box for changing the values of variables.

SettingsSearch opens the Settings dialog box at the specified category, variable name, or user preference.

SetUCS sets the UCS to a viewpoint specified through a dialog box.

SetVar displays and changes the values of system variables (short for "set variables").

Sh and **Shell** open the Windows command prompt window; runs other applications (short for "shell").

Shade shades the drawing mode.

ShadeMode sets the current visual style at the command line, such as Realistic, Conceptual, Edges, and X-ray.

-ShadeMode sets the old type of shade modes: 2D, 3D, Hidden, Flat, Flat with Edges, Gouraud, and Gouraud with edges.

Shape places shapes from SHX files in drawings.

SheetSet and **SheetsetHide** manage sheet sets, and closes the Sheet Set pane.

Singleton toggles whether multiple copies of BricsCAD can run at the same time.

Site imports terrain models from points and Civil 3D surfaces, or creates them from entities.

SiteEdit edits terrain sites.

Sketch draws freehand lines.

Slice slices 3D solids with a plane or surface.

Snap restricts pointer movements and pointing in the drawing to specified intervals.

Solid draws solid-filled 2D faces.

SolidEdit edits 3D solids and 2D regions.

SolProf creates hidden line representations of 3D solids in a layout viewport.

Spell checks the spelling of text in the drawing.

Sphere draws three-dimensional solid spheres.

Spline draws quadratic or cubic non-uniform rational Bezier spline (NURBS) curves.

SpotLight inserts spot lights into drawings.

Screenshot takes a screen grab of the current space, excluding all UI elements.

Start runs operating system applications.

StatBar toggles the display of the status bar.

Status reports status of the drawing's settings in the Text window.

StlOut export 3D models in STL format for 3D printing (short for "stereolithography").

StopScript stops recording of scripts begun with the RunScript command.

Stretch moves or stretches entities.

StandardPartsPanelClose and **StandardPartsPanelOpen** close and open the Standard Parts panel.

StructurePanel and **StructurePanelClose** open and close the Structure panel displaying tree structure of the drawing content.

+StructurePanel opens a CST structure tree configuration file.

Style and **-Style** creates and edits text styles through the Drawing Explorer.

StylesManager creates and attaches plot style files.

Subtract creates a composite region or a 3D solid by subtraction.

SunProperties edits sun properties through the Drawing Explorer.

SupportFolder opens the `C:\Users\<login>\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support` folder.

SvgOptions controls the output as SVG files.

Sweep creates solid primitives or surfaces by sweeping two dimensional entities along a path.

SysWindows arranges windows.

Sheet Metal Commands

(Available for Mechanical edition; requires an additional license; sm = sheet metal)

LicPropertiesSheetmetal reports the license state of the sheet metal module.

smAssemblyExport converts 3D solid sheet metal parts to DXF files with unfolding information.

smBendCreate converts hard edges (sharp edges between flange faces) into bends.

smBendSwitch converts bends to lofted bends.

smConvert automatically recognizes flanges and bends in a 3D solid.

smDelete removes a bend or a junction by restoring the hard edge between two flanges; removes a flange with all the bends adjacent to it.

smDissolve removes sheet metal data from the selected features.

smExport2D exports unfolded representations of sheet metal bodies as 2D profiles in DXF or DWG files.

smExportOSM exports sheet metal solids to OSM files (short for “Open Sheet Metal”) used by CADMAN-B CAM systems.

smExtrude extrudes polylines to sheet metal parts.

smFlangeBase creates base (initial) flanges of sheet metal parts from closed 2D entities.

smFlangeBend bends existing flanges along a line, taking into account the k-factor.

smFlangeConnect closes gaps between two arbitrarily oriented flanges.

smFlangeContour creates flange from a closed contour.

smFlangeEdge creates one or more flanges to a sheet metal part by pulling one or more edges of an existing flange.

smFlangeRotate rotates a selected flange of a sheet metal part with automatic selection of the rotation axis depending on the design intent.

smFlip switches flange sides to reverse reference faces.

smForm adds forms to sheet metal.

smHemCreate creates a variety of hems on sheet metal models.

smImprint uses imprinted edges to split thickness faces of sheet metal parts.

smJunctionCreate converts hard edges (sharp edges between flange faces) and bends into junctions.

smJunctionSwitch changes symmetrical junction features to overlapping faces.

smLispGet returns values related to sheet metal variables.

smLispSet changes values related to sheet metal variables.

smLoft creates sheet metal part with lofted bends and flanges from two non-coplanar curves.

smParametrize generates consistent sets of 3D constraints for sheet metal parts

smReliefCreate creates proper corner (three or more adjacent flanges) and bend reliefs (at the start and end of a flange edge).

smRepair restores the 3D solid model of a sheet metal part by thickening one of its sides: all thickness faces become perpendicular to flange faces.

smReplace replacing form features with ones from libraries.

smRibCreate adds associative rib (form) features on sheet metal parts based on 2D profiles.

smSelect selects hard edges and form features of sheet metal parts.

smSplit splits flanges and lofted bend; replaces the old **smFlangeSplit** command.

smTabCreate creates a tab between two flanges.

smUnfold generates unfolded 2D or 3D representations of sheet metal parts.

T Commands

Table and **-Table** draws tables in drawings.

Tableedit edits text in table cells.

TableExport exports the contents of a table entity to CSV (command separated values) files.

TableMod modifies the properties of table cells.

TableStyle creates and manages table styles through the Drawing Explorer.

Tablet configures and calibrates tablets, and toggles tablet mode (WINDOWS MODE).

TabSurf draws tabulated surfaces from a path curve and a direction vector.

Tangent toggles tangent entity snap; snaps to the tangency of circles, arcs, ellipses and elliptical arcs.

TConnect connects solids by their faces

TemplateFolder opens the `C:\Users\<login>\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\Templates` folder.

Text and **-Text** places lines of text in the drawing.

TextScr displays the text window showing command history (short for “text screen”).

TextToFront sets the draw order of all texts and dimensions in the drawing to display in front of all other entities.

TfLoad and **TfSave** open and save handle, xsd, and strip data from DWT template files.

Time reports on the time spent in the drawing.

TInsert inserts blocks in the cells of tables.

Tolerance draws tolerances (datum indicators and basic dimension notation).

Toolbar and **-Toolbar** displays and hides toolbars.

ToolPalettes opens the Tool Palettes bar.

ToolPalettesClose closes the Tool Palettes bar.

-ToolPanel opens tool panels by name at the command bar.

Torus draws three-dimensional torrid solids.

TpNavigate opens tool palettes or group at the command bar.

Trace draws traces.

Transparency toggles the transparency of monotone images; has nothing do with the transparency property.

Trim trims entities at a cutting edge defined by other entities.

TxtExp explodes text into polyline segments.

U Commands

U reverses the most recent command.

Ucs creates and displays named UCSes through the command bar (short for “user-defined coordinate system”).

UcsIcon toggles the display of the UCS icon.

Undefine disables built-in commands.

Undo restores deleted entities.

UndoEnt undoes property changes to selected entities.

Union creates composite regions or solids by addition.

UnisolateObjects makes entities visible again following the IsolateObjects and HideObjects commands.

Units and **-Units** sets coordinate and angle display formats and precision.

UpdateField forces the values of field text to update.

Url opens the default Web browser (short for “uniform resource locator”).

V Commands

View and **-View** saves, restores, and manages user-defined model and sheet views, and presets views.

ViewHorizontal rotates the viewpoint to make z=0 (horizontal)

ViewLabel adds labels to views; available through the Sheet Set manager only.

ViewRes sets the view resolution and toggles fast-zoom mode (short for “view resolution”).

VisualStyles and **-VisualStyles** creates and edits visual style definitions in the Drawing Explorer or at the command line.

VmlOut exports drawings in VML format embedded in Web pages (short for “vector markup language”).

VpClip clips viewports in layouts (short for “view port clipping”).

VpLayer changes the properties of layers in the current paper space viewport (short for “view port layer”).

VpMax and **VpMin** maximize and minimize the current viewport in paper space.

VPoint Changes the 3D viewpoint through a dialog box.

VPorts and **-VPorts** create one or more viewports in model space (short for “viewports”).

VSlide displays images saved as SLD or WMF files (short for “view slide”).

VBA Commands

(Available in Pro and Platinum editions only; vba = Visual Basic for Applications)

Vbalde opens the **BLADE** editing window; short for “integrated development environment”.

VbaLoad and **-VbaLoad** loads VBA projects.

VbaMan manages VBA projects; short for “manager”.

VbaRun and **-VbaRun** runs, creates, edits, and deletes VBA macros.

VbaSecurity sets the security level for running VBA macros.

VbaUnload unloads VBA projects.

ViewBase Commands

(Available in Pro and Platinum editions only)

ViewBase generates associative orthographic and standard isometric views of a 3D solid model in a paper space layout.

ViewDetail creates a detail view of a portion of a standard generated drawing at a larger scale.

ViewDetailStyle specifies the visual format of detail views and detail symbols.

ViewEdit changes the scale and hidden line visibility of drawing views; works in paper space only.

ViewExport exports the content of drawing views to Model space or to a new drawing; operates in paper space only.

ViewProj generates additional projected views from an existing drawing view.

ViewSection creates cross section views based on standard drawing views generated by the ViewBase command in paper space layouts.

ViewSectionStyle specifies the visual format of section views and section lines.

ViewUpdate updates drawing views.

W Commands

WBlock and **-WBlock** export blocks, selected entities, or the entire drawing as a DWG file.

WCascade, **WClose**, **WCloseAll**, **WNext**, and **WPrev** cascade the windows, close the current window, close all windows, and switch to the next or previous windows.

Weblight places Web lights.

Wedge draws three-dimensional solids with a sloped face tapering along the X axis.

WhoHas reports the ownership of a drawing file.

WhTile, **WiArrange**, and **WvTile** tiles windows horizontally, in an overlapping manner, or vertically.

WipeOut creates blank areas in drawings.

WmfOut exports the drawing in WMF (WIndows meta file), EMF (enhanced meta file), or SLD (slide) format.

WorkSets creates and loads named sets of drawing files.

Workspace sets the current workspace; creates, modifies, and saves workspaces.

WsSaveAs saves the current user interface by name.

WsSettings opens the Customize dialog box at the Workspace tab.

X Commands

XAttach attaches externally-referenced drawings.

XClip clips externally-referenced drawings.

XEdges extracts edges from 3D solids as lines.

XLine draws infinitely long lines.

XmlSave prompts for handles to save in an XML file.

XOpen opens externally-referenced drawings in a new window.

Xplode explodes entities, and provides control over the resulting entities.

XRef and **-XRef** attaches DWG files to the current drawing through the Drawing Explorer or the command line.

Z Commands

Zcenter toggles the 3D center entity snap; snaps to the center of planar or curved 3D faces.

Zknot toggles the 3D knot entity snap; snaps to a knot on a spline.

Zmidpoint toggles the 3D midpoint snap; snaps to the midpoint of a face edge.

Znearest toggles the 3D nearest entity snap; snaps to a point on the face of a 3D entity that is nearest to the cursor.

Znone disables all 3D snap modes.

Zoom increases or decreases the visible part of the drawing.

Zperpendicular toggles the 3D perpendicular entity snap; snaps to a point perpendicular to a face.

Zvertex toggles the 3D vertex entity snap; snaps to the closest vertex of a 3D entity.

Commands

? displays the Help window.

2dIntersection toggles apparent intersection entity snap; snaps to the intersections of entities, even when they only appear to intersect in 3D space.

3D draws 3D polygon mesh objects: boxes, cones, cylinders, dishes, domes, pyramids, spheres, tori, wedges, or meshes.

3DArray constructs 3D rectangular arrays and rotated polar arrays.

3DCompare compares the 3D content of two drawing files.

3DConvert converts 3D solids to polyface meshes.

3DFace draws 3D 4-edged faces with optional invisible edges.

3DIntersection toggles Intersection entity snap; snaps to the intersections of entities.

3DMesh draws 3D surface meshes.

3DOsnap and **-3DOsnap** sets the entity snap modes for 3D entities through the Settings dialog box.

3DPoly draws 3D polylines.

Summary of Variables & Settings

BRICSCAD USES VARIABLES TO STORE AND REPORT SETTINGS AFFECTING THE PROGRAM and drawings. There are two types of variables: *system* variables that mimic the names and values from AutoCAD, and *preference* variables unique to BricsCAD. You access and change variables through a dialog box (**Settings** command) or directly on the command line (**SetVar** command).

This appendix lists over 1,000 variable names in alphabetical order.

UPPERCASE text indicates the name is also found in AutoCAD as a system variable

MixedCase text means the variable is a *preference*, and so is unique to BricsCAD

Blue text indicates that the variable is new in V20

ikeThrough text indicates the variable was removed from BricsCAD

userid or *login* refers to your computer login name

When you see **Read-only** (r/o), it means that you cannot change the variable's value; the value has been set by BricsCAD or by the operating system.

System Variable Name	Read-Only	Default Value
A Variables		
ACADLSPASDOC		0
ACADPREFIX	r/o	"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Support\,; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Fonts\; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Help\en_US\"
ACADVER	r/o	"20.0 BricsCAD"
AcisHlrResolution		-1
ACISOUTVER		70
AcisSaveAsMode		0
AdaptiveGridStepSize		4.0000
AFLAGS		0
ALLOWBREAKLINECROSSINGS		"1"
ALLOWEDBENDANGLES		"1"
AllowTabExternalMove		1
AllowTabMove		1
AllowTabSplit		1
ANGBASE		0
ANGDIR		0
Anglesamplinginterval		"5"
ANNOALLVISIBLE		1
ANNOAUTOSCALE		-4
AnnoSelected	r/o	0
ANNOTATEDDWG		0
AntiAliasRender		2
AntiAliasScreen		1
APBOX		0
ArcTessellation		"0.01"
APERTURE		10
AREA	r/o	0
AREAPEC		-1
AREAUNITS		"in ft mi µm mm cm m km"
ARRAYASSOCIATIVITY		1
ARRAYEDITSTATE	r/o	0
ARRAYTYPE		0
Associativity		"3"
ATTDIA		0
ATTMODE		1
AttractionDistance		3
ATTREQ		1
AUDITCTL		0
AuditErrorCount	r/o	0
AUNITS		0
AUPREC		0
AutoAdoptSizes		"1"
AUTOCOMPLETEDELAY		0.3
AUTOCOMPLETEMODE		47
AutoFlipQuarterTurn		"1"
AutomaticConnection		"1"
AutomaticTees		"0"
AUTOMENULOAD		1
AutoResetScales		0
AutosaveChecksOnlyFirstBitDBMOD	1	
AUTOSNAP		119
AutoTrackingVecColor		171

System Variable Name	Ready-Only	Default Value
AutoUpdateRooms		"1"
AutoVpFitting		1
AXISMODE		0
AXISUNIT		X= 0 Y= 0 Z= 0

B Variables

BACKGROUNDPLOT		2
BActionColor		"7"
BACKZ	r/o	0
BASEFILE		"Default-mm.dwt"
BDependencyHighlight		1
bimConnectCutType		"0"
bimMatchProp		"1"
BIMOSMODE		0
BINDTYPE		0
BKGCOLOR		7
BKGCOLORPS		7
BLIPMODE		0
BLOCKEDITLOCK		0
BLOCKEDITOR		0
BlocksPath		"C:\Users\userid\Documents\"
bmAutoUpdate		1
bmForceUpdateMode		0
bmFormTemplatePath		""
bmReportPanel		0
BMUPDATEMODE		0
BndLimit		1000
BoundaryColor		95
BParameterSize		12
BpTextHorizontal		1
BSysLibCopyOverwrite		0
BtMarkDisplay		1
BVMODE		0

C Variables

CACHELAYOUT		1
CAMERADISPLAY		0
CAMERAHEIGHT		0
CANNOSCALE		"1:1"
CANNOSCALEVALUE	r/o	1
CDATE	r/o	20160211.15522
CECOLOR		"ByLayer"
CELTSSCALE		1
CELTYPE		"ByLayer"
CELWEIGHT		-1
CenterCrossGap		"0.05x"
CenterCrossSize		"0.1x"
CenterExe		0.1200
CenterLayer		""
CenterLtscale		1.0000
CenterLtype		"Center2"
CENTERLTYPEFILE		"Default.Lin"
CenterMarkExe		1
CETRANSARECNY		"ByLayer"
CGEOCS	r/o	""

System Variable Name	Read-Only	Default Value
CHAMFERA		0
CHAMFERB		0
CHAMFERC		0
CHAMFERD		0
CHAMMODE		0
<i>Chapoolog</i>		
<i>ChapoologVerbose</i>		
<i>ChapoolOnModified</i>		
<i>ChapoolServer</i>		
<i>ChapoolTempFolder</i>		
<i>ChapoolUploadDependencies</i>		
<i>ChapoolWebsite</i>		
CheckDwlPresence		0
CIRCLERAD		0
CLAYER		"0"
CLEANSCREENOPTIONS		15
CLEANSCREENSTATE	r/o	0
ClipboardFormat		1
CLIPBOARDFORMATS		127
CliPromptLines		4
CLISTATE	r/o	1
CloseChecksOnlyFirstBitDBMOD		0
CloudDownloadPath		"c:\users\userid\documents\bricsys247\"
CloudLog		0
CloudLogVerbose		0
CloudOnModified		1
CloudServer		"https://my.bricsys247.com/"
CloudSessionPath		"c:\users\userid\AppData\Local\bricsys\bricscad\"
CloudTempFolder		"C:\Users\userid\AppData\Local\Temp\Bricsys_24_7\"
CloudUploadDependencies		1
CMATERIAL		"ByLayer"
CMDACTIVE	r/o	1
CMDDIA		1
CMDECHO		1
CmdLineEditBgColor		"#fefefe"
CmdLineEditFgColor		"#202020"
CmdLineFontName		"Consolas"
CmdLineFontSize		10
CmdLineListBgColor		"#ecf1ff"
CmdLineListFgColor		"#000000"
CMDLNTXT		","
CMDNAMES	r/o	"SETTINGS"
CMLEADERSTYLE		"Standard"
CMLJUST		0
CMLSCALE		1
CMLSTYLE		"Standard"
CMPLRMIS		1
CMPLRMOD1		253
CMPLRMOD2		2
CMPLRNEW		3
CMPDIFFLIMIT		1000
CmpFadeCtl		80
CmpLog		0
ColorTheme		0
COLORX		11
COLORY		112

System Variable Name	Ready-Only	Default Value
COLORZ		150
COMAcadCompatibility		0
COMPASS		0
Componentspath		"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\Bim\Components\"
CONSTRAINTBARDISPLAY		3
ContinuousMotion		0
ConvertToArrays		1
COORDS		1
COPYMODE		0
CPLOTSTYLE		"ByColor"
CPROFILE	r/o	"Default"
CreateThumbnailOnTheFly		1
CREATEVIEWPORTS		1
CrosshairDrawMode		2
CROSSINGAREACOLOR		91
CTAB		"Model"
CTABLESTYLE		"Standard"
Ctrl3DMouse		1
CtrlMButton		1
CTRLMOUSE		1
CURSORSIZE		3
CVPORT		2

D Variables

DataLinkNotify		2
DATE	r/o	2456335.6613464
DBCSTATE	r/o	0
DBLCLKEDIT		1
DBMOD	r/o	0
DCTCUST		""
DCTMAIN		"en_US.dic"
ddBetweenKnots		2
ddFastMode		0
ddGridAspectRatio		0
ddMaxFacetEdgeLength		0
ddMaxNumGridLines		10000
ddNormalTol		15
ddPointsPerEdge		0
ddSurfaceTol		0
ddUseFacetRES		1
DefaultBlockName		0
DefaultInsPoint		0
DEFAULTLIGHTING		0
DefaultLightShadowBlur		8
DefaultNewSheetTemplate		""
DefaultRoomHeight replaced by RoomHeight		
DEFLPSTYLE		"Normal"
DEFPLSTYLE		"ByColor"
DeleteInterference		1
DeleteTool		1
DELOBJ		1
DEMANDLOAD		3
DGNFRAME		2
dgnImp2dClosedBSplineCurveImportMode		0
dgnImp2dEllipsImportMode		0

System Variable Name	Read-Only	Default Value
dgnImp2dShapelImportMode		0
dgnImp3dClosedBSplineCurveImportMode		1
dgnImp3dEllipseImportMode		0
dgnImp3dObjectImportMode		0
dgnImp3dShapelImportMode		1
dgnImpBreakDimensionAssociation		0
dgnImpConvertDgnColorIndicesToTrueColors		0
dgnImpConvertEmptyDataFieldsToSpaces		1
dgnImpEraseUnusedResources		0
dgnImpExplodeTextNodes		0
dgnImpImportActiveModelToModelSpace		1
dgnImpImportInvisibleElements		1
dgnImpImportPaperSpaceModels		1
dgnImpImportViewIndex		-1
dgnImpRecomputeDimensionsAfterImport		0
dgnImpSymbolResourceFiles		""
dgnImpXRefImportMode		2
DGNOSNAP		1
DIASAT	r/o	0
DisplayAxes		"0"
DisplayAxesForMep		"0"
DisplayScaling	r/o	125
DisplaySidesAndEnds		"1"
DisplaySnapMarkerInAllViews		0
DisplayTooltips		1
DISPPAPERBKG		1
DISPPAPERMARGINS		1
DISPSILH		0
DISTANCE	r/o	0
dmAuditLevel		1
DMAUTOUPDATE		1
dmConnectionCutType		0
dmExtrudeMode		0
dmPushPullSubtract		1
DMRECOGNIZE		0
DockPriority		1
DocTabPosition		0
DONUTID		0.5
DONUTOD		1
DRAGMODE		2
DragModeHide		0
DRAGMODEINTERRUPT		1
DRAGOPEN		1
DRAGP1		10
DRAGP2		25
DRAGSNAP		0
DrawingPath		"C:\Users\userid\Documents\"
DrawingViewPreset		"none"
DrawingViewPresetHidden		0
DrawingViewPresetScale		""
DrawingViewPresetTangent		0
DrawingViewPresetScale		""
DrawingViewPresetTangent		0
DRAWORDERCTL		3
DxTextAdjustAlignment		0

System Variable Name	Ready-Only	Default Value
DWFFRAME		2
DWFOSNAP		1
DwfVersion		2
DWGCHECK		0
DWGCODEPAGE	r/o	"ANSI_1252"
DWGNAME	r/o	"Drawing1.dwg"
DWGPREFIX	r/o	"C:\Program Files (x86)\Bricsys\BricsCAD V20x64\"
DWGTITLED	r/o	0
DXEVAL		12
DxTextAdjustAlignment		0
DYNCONSTRAINTMODE		1
DYNDIGRIP		31
DynDimAperture		20
DynDimColorHot		142
DynDimColorHover		142
DynDimDistance		1
DynDimLineType		1
DYNDIVIS		1
DynInputTransparency		65
DYNMODE		3
DynPicCoords		0

Dimensions Variables

DIMADEC		0
DIMALT		0
DIMALTD		2
DIMALTF		25.4
DIMALTRND		0
DIMALTTD		2
DIMALTTZ		0
DIMALTU		2
DIMALTZ		0
DIMANNO	r/o	0
DIMAPOST		""
DIMARCSYM		0
DIMASO		1
DIMASSOC		2
DIMASZ		0.18
DIMATFIT		3
DIMAUNIT		0
DIMAZIN		0
DIMBLK		""
DIMBLK1		""
DIMBLK2		""
DIMCEN		0.09
DIMCLRD		0
DIMCLRE		0
DIMCLRT		0
DIMDEC		4
DIMDLE		0
DIMDLI		0.38
DIMDSEP		"0"
DIMEXE		0.18
DIMEXO		0.0625
DIMFIT		3

System Variable Name	Read-Only	Default Value
DIMFRAC		0
DIMFXL		1
DIMFXLON		0
DIMGAP		0.09
DIMJOGANG		0.7853981634
DIMJUST		0
Dimlayer		" "
DIMLDRBLK		""
DIMLFAC		1
DIMLIM		0
DIMLTEX1		""
DIMLTEX2		""
DIMLTYPE		""
DIMLUNIT		2
DIMLWD		-2
DIMLWE		-2
DIMPOST		""
DIMRND		0
DIMSAH		0
DIMSCALE		1
DIMSD1		0
DIMSD2		0
DIMSE1		0
DIMSE2		0
DIMSHO		1
DIMSOXD		0
DIMSTYLE	r/o	"Standard"
DIMTAD		0
DIMTDEC		4
DIMTFAC		1
DIMTFILL		0
DIMTFILLCLR		"BYBLOCK"
DIMTIH		1
DIMTIX		0
DIMTM		0
DIMTMOVE		0
DIMTOFL		0
DIMTOH		1
DIMTOL		0
DIMTOLJ		1
DIMTP		0
DIMTSZ		0
DIMTVP		0
DIMTXSTY		"Standard"
DIMTXT		0.18
DIMTXTDIRECTION		0
DIMTZIN		0
DIMUNIT		2
DIMUPT		0
DIMZIN		0

E Variables

EDGEMODE	0
ELEVATION	0

System Variable Name	Ready-Only	Default Value
ElevationAtBreaklineCrossings		"0"
EnableAttraction		1
EnableBimBkUpdate		"0"
EnableHyperlinkMenu		1
EnableHyperlinkTooltip		0
ERRNO		0
EXPERT		0
ExpInsAlign		0
ExpInsAngle		0
ExpInsFixAngle		1
ExpInsFixScale		1
ExpInsScale		1
EXPLMODE		1
ExportAcisFormatVersion		"0"
ExportCatiav4FormatVersion		"0"
ExportCatiav5FormatVersion		"0"
ExportHiddenParts		"0"
EXPORTMODELSPACE		0
EXPORTPAGESETUP		0
EXPORTPAPERSPACE		0
ExportParasolidFormatVersion		"0"
ExportProductStructure		"1"
ExportStepFormatVersion		"1"
EXTMAX	r/o	-1.0000E+20,-1.0000E+20,-1.0000E+20
EXTMIN	r/o	1.0000E+20,1.0000E+20,1.0000E+20
EXTNAMES		1

F Variables

FACETRATIO		0
FACETRES		0.5
fbxExportCameras		1
fbxExportEntities		1
fbxExportEntitiesSelType		0
fbxExportLights		1
fbxExportMaterials		1
fbxExportTextures		0
fbxExportTexturesPath		"c:\program files\bricsys\bricscad v20 en_us\"
FEATURECOLORS		1
FIELDDISPLAY		1
FIELDEVAL		31
FILEDIA		1
FILLETRAD		0.5
FILLMODE		1
FittingRadiusType		"0"
FittingRadiusValue		"1.5"
FLATLAND		Off
FONALT		"simplex.shx"
FONTMAP		"default.fmp"
FRAME		3
FRAMESELECTION		0
FRONTZ	r/o	0
FULLOPEN	r/o	1

G Variables

GDIOBJECTS	r/o	3768
GearTeethNumber		"1"
GENERATEASSOCVIEWS		0
GEOLATLONGFORMAT		1
GEOMARKERVISIBILITY		1
GeomRelations		0
GetStarted		1
GfAng		0.0000
GfClr1		"5"
GfClr2		"7"
GfClrLum		1.0000
GfClrState		0
GfName		1
GfShift		0
GLSWAPMODE		2
GradientColorBottom		"#d2d2d2"
GradientColorMiddle		"#fafafa"
GradientColorTop		"#ffffff"
GradientMode		"0"
GRIDAXISCOLOR		252
GRIDDISPLAY		3
GRIDMAJOR		5
GRIDMAJORCOLOR		253
GRIDMINORCOLOR		254
GRIDMODE		0
GRIDSTYLE		0
GRIDUNIT		1/2",1/2"
GRIDXYZT		1
GRIPBLOCK		0
GRIPCOLOR		72
GRIPDYNCOLOR		140
GRIPHOT		240
GRIPHOVER		150
GRIPOBJLIMIT		100
GRIPS		1
GRIPSIZE		4
GRIPTIPS		1
GsDeviceType		
GsDeviceType2D		0
GsDeviceType3D		1

H Variables

HALOGAP		0
HANDLES	r/o	1
HANDSEED		"64"
Headroom		"2000"
HIDEPRECISION		0
HideSystemPrinters		0
HIDETEXT		1
HIDEXREFSCALES		1
HIGHLIGHT		1
HIGHLIGHTCOLOR		142
HIGHLIGHTEFFECT		0

System Variable Name	Ready-Only	Default Value
<i>HomeGradientColorBottom</i>	replaced by	HorizonBkg_SkyLow
<i>HomeGradientColorMiddle</i>	replaced by	HorizonBkg_SkyHorizon
<i>HomeGradientColorTop</i>	replaced by	HorizonBkg_SkyHigh
<i>HomeGradientMode</i>	replaced by	HorizonBkg_Enable
HorizonBkg_Enable		1
HorizonBkg_GroundHorizon		"#878787"
HorizonBkg_GroundOrigin		"#5F5F5F"
HorizonBkg_SkyHigh		"#239BFF"
HorizonBkg_SkyHorizon		"#FFFFFF"
HorizonBkg_SkyLow		"#FAFAFF"
HotkeyAssistant		1
HPANG		0
HPANNOTATIVE		0
HPASSOC		1
HpBackgroundColor		" "
HpColor		" "
HPBOUND		1
HPBOUNDRETAIN		0
HPDOUBLE		0
HPDRAWORDER		3
HPGAPTOL		0
HPLAYER		" "
HPLINETYPE		0
HplandDetection		0
HPMAXAREAS		0
HPNAME		""
HPOBJWARNING		10000
HPORIGIN		0",0"
HPSCALE		1
HPSEPARATE		0
HPSPACE		1
HPSTYLE	replaced by	HplandDetection
HPTRANSPARENCY		" "
HYPERLINKBASE		""

I Variables

IfcExplodeExternalReferences	"0"
IfcExportBaseQuantities	"0"
IfcExportElementsOnOffAndFrozenLayer	"1"
IfcExportMultiplyElementsAsAggregated	"0"
IfcExportTesselation	"0"
IfcImportBimData	"1"
IfcImportBrepGeometryAsMeshes	"0"
IfcImportModelOrigin	"0"
IfcImportParametricComponents	"0"
ImportIfcProjectStructurAsXrefs	"0"
IfcImportSpaces	"0"
ImageCacheFolder	"C:\Users\userid\AppData\Local\Temp\ImageCache\"
ImageCacheMaxMemory	160
ImageDiskCache	1
IMAGEFRAME	1
IMAGEHLT	0
ImageNotify	0
ImportColors	"1"
ImportCreoAlternateSearchPaths	""

System Variable Name	Read-Only	Default Value
ImportCuiFileExists		0
ImportHiddenparts		"0"
ImportIfcProjectStructureAsXrefs		"0"
ImportIgesSimplify		"1"
ImportIgesStitch		"1"
ImportInventorAlternateSearchPaths		""
ImportNxAlternateSearchPaths		""
ImportPmi		"1"
ImportProductStructure		"2"
ImportRepair		"0"
ImportSimplify		"0"
ImportSolidedgeAlternateSearchPaths		""
ImportSolidworksAlternateSearchPaths		""
ImportSolidworksRotateYz		"1"
ImportStepRotateYz		"0"
ImportStitch		"0"
IncludePlotStamp		1
INDEXCTL		0
INETLOCATION		"http://www.bricsys.com"
INSBASE		0",0",0"
INSNAME		""
INSUNITS		1
INSUNITSDEFSOURCE		0
INSUNITSDEFTARGET		0
InsUnitsScaling		1
INTERFERECOLOR		"BYLAYER"
InterfereLayer		"Interference"
INTERFEREOBJS		""
INTERFEREVPVS		""
InteriorElevationMinLength		"20"
InteriorElevationOffset		"2"
INTERSECTIONCOLOR		257
INTERSECTIONDISPLAY		0
SAVEBAK		1
SAVEPERCENT		50
ISOLINES		4

L Variables

LASTANGLE		0
LASTPO		0",0",0"
LASTPROMPT	r/o	": SETTINGS"
LATITUDE		37.795
LayerFilterExcess		250
LAYERPMODE		1
LAYLOCKFADECTL		50
LAYOUTREGENCTL		2
LayoutTab		1
LegacyCodeSearch	r/o	0
LengthSamplingInterval		"40"
LENGHUNITS		""
LENSLENGTH		50
LevelOfDetail		"0"
LicExpDays		31
LICFLAGS		7
LICKEY	r/o	"7897-9999-0000-99999-0000"

System Variable Name	Ready-Only	Default Value
LightGlyphColor		30
LIGHTGLYPHDISPLAY		1
LIGHTINGUNITS		0
LightWebGlyphColor		1
LIMCHECK		0
LIMMAX		1',9"
LIMMIN		0",0"
LINEARBRIGHTNESS		0
LINEARCONTRAST		0
LISPINIT		1
LoadMechanical2d		0
LOCALE		"en_US"
LocalRootFolder		"C:\Users\userid\AppData\Local\Bricsys\BricsCAD\V20x64\en_US"
LOCALROOTPREFIX	r/o	"C:\Users\userid\AppData\Local\Bricsys\BricsCAD\V20x64\en_US"
LOCKUI		0
LOFTANG1		1.5707963268
LOFTANG2		1.5707963268
LOFTMAG1		0
LOFTMAG2		0
LOFTNORMALS		1
LOFTPARAM		7
LOGFILEMODE		0
LOGFILENAME	r/o	""
LOGFILEPATH	r/o	"C:\Users\userid\AppData\Local\Bricsys\BricsCAD\V20x64\en_US"
LOGINNAME	r/o	"userid"
LONGITUDE		-122.394
LookFromDirectionMode		1
LookFromFeedback		1
LookFromZoomExtents		1
LTSCALE		1
LUNITS		4
LUPREC		4
LWDEFAULT		25
LWDISPLAY		0
LWDISPSCALE		0.55
LWUNITS		1

M Variables

M_axisDiameter		6
M_totalAxisLength		130
MACROREC		0
MACROTRACE		0
MANIPULATOR		0
ManipulatorColorTheme		0
MANIPULATORDURATION		250
ManipulatorSize		1
MassPrec		-1
MassPropAccuracy		0.01
MASSUNITS		"oz lbs stone mg g kg tonne"
MaterialsPath		""
MAXACTVP		64
MAXHATCH		100000
MAXSORT		200
MAXTHREADS		0
MBSTATE	r/o	0

System Variable Name	Read-Only	Default Value
MBUTTONPAN		1
MEASUREINIT		0
MEASUREMENT		0
Mech2dSaveFormat		2013
MENUBAR		1
MENUCTL		1
MENUECHO		0
MENUNAME	r/o	"C:\Users\userid\AppData\Roaming\Bricsys\Bricscad\V20x64\en_US\Support\default.cui"
MESHTYPE		1
MiddleClickClose		1
MILLISECS	r/o	436750804
MIRRTEXT		1
MLEADERSCALE		1
MODEMACRO		""
MSLTSCALE		1
MSOLESCALE		1
MTEXTCOLUMN		0
MTEXTDETECTSPACE		1
MTEXTED		""
MTEXTFIXED		2
MTFLAGS		0
MultiSelectAngularTolerance		"3"
MyDocumentsFolder		"C:\Users\userid\Documents\"
MYDOCUMENTSPREFIX	r/o	"C:\Users\userid\Documents\"

N Variables

NEARESTDISTANCE		1
NAVVCUBEDISPLAY		3
NAVVCUBELOCATION		0
NAVVCUBEOPACITY		50
NAVVCUBEORIENT		1
NavVCubeSize		4
NFILELIST		10
NOMUTT		0
NORTHDIRECTION		0

O Variables

OBJECTISOLATIONMODE		0
OBSCUREDOLOR		257
OBSCUREDLTTYPE		0
OFFSETDIST		-1
OFFSETERASE		0
OFFSETGAPTYP		0
OLEFRAME		2
OLEHIDE		0
OLEQUALITY		0
OLESTARTUP		0
OPMSTATE	r/o	1
OrbitAutoTarget		1
ORTHOMODE		0
OSMODE		4133
OSNAPCOORD		2
OSNAPZ		0
OSOPTIONS		1

P Variables

PanBuffer		1
PanelButtonSize	r/o	1
PAPERUPDATE		0
PARAMETERCOPYMODE		1
PdfCache		2
PdfEmbeddedTtf		1
PdfExportSolidHatchType		2
PDFFRAME		1
PdfHatchToBmpDpi		300
PdfImageAntiAlias		1
PdfImageCompression		1
PdfImageDPI		300
PdfImportApplyLineweight		1
PdfImportAsBlock		0
PdfImportConvertSolidsToHatches		0
PdfImportImagePath		"pdf images"
PdfImportJoinLineAndArcSegments		1
PdfImportLayersUseType		0
PdfImportRasterImages		0
PdfImportSolidFills		1
PdfImportTrueTypeText		1
PdfImportTrueTypeTextAsGeometry		0
PdfImportUseGeometryOptimization		1
PdfImportVectorGeometry		1
PdfLayersSetting		1
PdfLayoutsToExport		0
PdfMergeControl		0
PdfNotify		0
PDFOSNAP		1
PdfPaperHeight		297
PdfPaperSizeOverride		0
PdfPaperWidth		210
<i>PdfPRCCompression</i>		
<i>PdfPRCExport</i>		
<i>PdfPRCSingleViewMode</i>		
PdfRenderDPI		300
PdfShxTextAsGeometry		0
PdfSimpleGeomOptimization		1
<i>PdfTextIsSearchable</i>		
PdfTtfTextAsGeometry		0
PdfUsePlotStyles		1
PdfVectorResolutionDpi		2400
PdfZoomToExtentsMode		1
PDMODE		0
PDSIZE		0
PEDITACCEPT		0
PELLIPSE		0
PERIMETER		0
PERSPECTIVE		0
PFACEVMAX		4
PICKADD		1
PICKAUTO		3
PICKBOX		4
PICKDRAG		0

System Variable Name	Read-Only	Default Value
PICKFIRST		1
PICKSTYLE		1
PictureExportScale		1
_PKSER	r/o	""
PlacesBarFolder1		0
PlacesBarFolder2		1
PlacesBarFolder3		3
PlacesBarFolder4		5
PLATFORM	r/o	"Microsoft Windows NT Version 6.2"
PLINECACHE		0
PLINECONVERTMODE		0
PLINEGEN		0
PLINETYPE		2
PLINEWID		0
PlotCfgPath		"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\PlotConfig"
PLOTID		""
PlotOutputPath		""
PLOTROTMODE		2
PlotStylePath		"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\PlotStyles"
PLOTTER		0
PLOTTRANSPARENCYOVERRIDE		1
PLQUIET		0
Pointcloud2dVsDisplay		1
PointcloudBoundary		1
PointcloudCacheFolder		"c:\users\userid\AppData\local\temp\pointcloudcache"
PointcloudPointMax		4000000
PointcloudPointSize		2
POLARADDANG		""
POLARANG		90
POLARDIST		0
POLARMODE		0
POLYSIDES		4
POPUPS	r/o	1
PpState	r/o	0
PreviewDelay		30
PREVIEWEFFECT		2
PREVIEWFILTER		5
<i>PreviewTopdown</i>		
PREVIEWTYPE		0
PreviewWndInOpenDlg		1
PrintFile		""
PrintPdfPreview		1
PRODUCT	r/o	"Bricscad"
PROGBAR		1
PROGRAM	r/o	"BRICSCAD"
PROJECTIONTYPE		0
PROJECTNAME		""
ProjectSearchPaths		""
PROJMODE		1
PROMPTMENU		3
PromptMenuFlags		1
PromptOptionFormat		2
PromptOptionTranslateKeywords		1
PropertyPreview		1
PropertyPreviewDelay		500
PropertyPreviewObjLimit		500

System Variable Name	Ready-Only	Default Value
PropPrevTimeout		1
PROPUNITS		103
PropUnitsVersion		1
PROXYGRAPHICS		1
PROXYNOTICE		1
PROXYSHOW		1
PROXYWEBSEARCH		1
PSLTSCALE		1
PSOLHEIGHT		4
PSOLWIDTH		0.25
PSTYLEMODE	r/o	1
PSTYLEPOLICY		1
PSVPSCALE		0
PUBLISHALLSHEETS		1
PUCSBASE		""

Q Variables

QAFLAGS		0
QTEXTMODE		0
<i>QuadAperture</i>		
QuadCommandLaunch		1
<i>QuadCommandSort</i>		
QuadDisplay		0
QuadExpandDelay		110
<i>QuadExpandGroup</i>		
QuadExpandTabDelay		50
QuadGoTransparent		0
QuadHideDelay		1000
QuadHideMargin		40
QuadIconSize		32
QuadIconSpace		1
QuadMostRecentItems		2
QuadPopupCorner		1
QuadShowDelay		150
_QuadTabFlags		12
QuadToolipDelay		1200
<i>QuadWarpPointer</i>		
QuadWidth		5

R Variables

R12SaveAccuracy		8
R12SaveDeviation		0
RASTERPREVIEW		1
RE_INIT	r/o	0
RealTimeSpeedUp		5
REALWORLDSCALE		1
RecentPath		"C:\Users\userid\Documents\"
RedHilite_ducslocked_face_alpha		25
RedHilite_ducslocked_face_color		"#007aff"
RedHiliteFull_Edge_Alpha		100
RedHiliteFull_Edge_Color		"#007AFF"
RedHiliteFull_Edge_ShowHidden		0
RedHiliteFull_Edge_Smoothing		1

System Variable Name	Read-Only	Default Value
RedHiliteFull_Edge_Thickness		2
RedHiliteFull_Face_Alpha		10
RedHiliteFull_Face_Color		"#007AFF"
Redhilite_hiddenedge_smoothing		1
Redhilite_hiddenedge_thickness		1
RedHilitePartial_SelectedEdgeGlow_Alpha		75
RedHilitePartial_SelectedEdgeGlow_Color		"#FFFFFF"
RedHilitePartial_SelectedEdgeGlow_Smoothing		1
RedHilitePartial_SelectedEdgeGlow_Thickness		3
RedHilitePartial_SelectedEdge_Alpha		100
RedHilitePartial_SelectedEdge_Color		"#007AFF"
RedHilitePartial_SelectedEdge_ShowGlow		1
RedHilitePartial_SelectedEdge_Smoothing		1
RedHilitePartial_SelectedEdge_Thickness		2
RedHilitePartial_SelectedFace_Alpha		10
RedHilitePartial_SelectedFace_Color		"#007AFF"
<i>RedHilitePartial_UnselectedEdge_Alpha</i>		
<i>RedHilitePartial_UnselectedEdge_Color</i>		
RedHilitePartial_UnselectedEdge_ShowHidden		1
<i>RedHilitePartial_UnselectedEdge_Smoothing</i>		
<i>RedHilitePartial_UnselectedEdge_Thickness</i>		
RedHilite_HiddenEdge_Alpha		50
RedHilite_HiddenEdge_Color		"#FFFFFF"
RedHilite_HiddenEdge_Smoothing		1
RedHilite_HiddenEdge_Thickness		1
RedSdkLineSmoothing		0
ReduceLengthType		"0"
ReduceLengthValue		"0.5"
RefeditLockNotInWorkset		0
REFEDITNAME	r/o	""
REGENMODE		1
RegExpand		1
REMEMBERFOLDERS		1
RenderMaterialDownload		1
RenderMaterialPath		"C:\ProgramData\..."
RenderMaterialStaticPath		"C:\Program Files\..."
RenderUsingHardware		1
ReportPanel		0
ReportPanelMode		0
RestoreConnections		"1"
RevCloudArcStyle		0
REVCLOUDCREATEMODE		1
REVCLOUDGRIPS		1
RevCloudMaxArcLength		0.375
RevCloudMinArcLength		0.375
RhinoVersion		60
RIBBONDOCKEDHEIGHT		120
RIBBONSTATE	r/o	0
Riserheight		"170"
RoamableRootFolder	r/o	"c:\users..."
ROAMABLEROOTPREFIX	r/o	"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US"
ROLLOVEROPACITY		100
ROLLOVERTIPS		1
RolloverSelectionSet		1
Roomheight		"3000"
RTDISPLAY		1

System Variable Name	Ready-Only	Default Value
RTRotationSpeedFactor		1
RTWalkSpeedFactor replaced by the StepSize variable		
RubberbandColor		40
RubberbandStyle		1
RunAsLevel		2

S Variables

Safemode	r/o	0
SaveChangeToLayout		1
SAVEFIDELITY		1
SAVEFILE	r/o	""
SAVEFILEPATH		"C:\Users\userid\AppData\Local\Temp\"
SaveFormat		1
SaveLayerSnapshot		1
SAVENAME	r/o	""
SaveOnDocSwitch		0
SAVEROUNDTRIP		1
SAVETIME		60
SCREENBOXES	r/o	26
SCREENMODE	r/o	1
SCREENSIZE	r/o	145'-8",73'-3"
SCRLHIST		256
SDI		0
SearchAll		0
SectionScale		"0.02"
SectionSettingsSearchPath		""
SectionSheetsTemplateImperial		""
SectionSheetsTemplateMetric		""
SecureLoad	r/o	0
SELECTIONANNODISPLAY		1
SELECTIONAREA		1
SELECTIONAREAOPACITY		25
SelectionModes		0
SELECTIONPREVIEW		3
SELECTSIMILARMODE		130
SHADEDGE		3
SHADEDIF		70
SheetNumberLeadingZeroes		1
SheetSetAutoBackup		1
SheetSetTemplatePath		"C:\Users\userid\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\Templates\Sheet Sets\"
SHORTCUTMENU		18
SHORTCUTMENDURATION		250
ShowDocTabs		1
ShowFullPathInTitle		0
SHOWLAYERUSAGE		0
ShowScrollButtons		1
ShowTabCloseButton		0
ShowTabCloseButtonActive		0
ShowTabCloseButtonAll		1
ShowTabControls		1
ShowWindowListButton		1
SHPNAME		""
SingletonMode		0
SKETCHINC		0.1
SKPOLY		0

System Variable Name	Read-Only	Default Value
SkpStitch		1
SKYSTATUS		0
SlabThickness		"250"
smAssemblyExportReportPathType		"0"
smAssemblyExportSolidTypesInReports		"1"
smAttributesLayerColor		"7"
smAttributesLayerTextHeight		"0.01"
smAttributesLayerTextHeightType		"0"
smBendAnnotationsLayerColor		"5"
smBendAnnotationsLayerTextHeight		"0.01"
smBendAnnotationsLayerTextHeightType		"0"
smBendlinesDownlayerColor		"1"
smBendlinesDownlayerLinetype		"Continuous"
smBendlinesDownlayerLineweight		"-3"
smBendlinesUplayerColor		"1"
smBendlinesUplayerLinetype		"Continuous"
smBendlinesUplayerLineweight		"-3"
SMCOLORBEND		"#FFDC50"
SMCOLORBENDRELIEF		"#64D296"
SMCOLORCORNERRELIEF		"#64D296"
SMCOLORFLANGE		"#90A4AE"
smcolorflangereferenceside		"#68a4ae"
Smcolorform		"#8791e1"
SMCOLORJUNCTION		"#FF6E40"
SMCOLORLOFTEDBEND		"#A0DCFA"
smColorMiter		"#af46d8"
smColorTab		""
smColorWrongBend		"#ff3300"
smContoursLayerColor		"7"
smContoursLayerLinetype		"continuous"
smContoursLayerLineweight		"30"
smConvertPreferFormFeatures		"0"
smConvertRecognizeHoles		"0"
smConvertRecognizeRibControlCurves		"0"
smConvertWrongFeatureThicknessDeviationType		"0"
smConvertWrongFeatureThicknessDeviationValue		"0.2"
smdefaultbendlineextenttype		"0"
smdefaultbendlineextentvalue		"0.25"
smdefaultbendradiustype		"2"
smdefaultbendradiusvalue		"1"
smdefaultbendreliefwidthtype		"0"
smdefaultbendreliefwidthvalue		"0.5"
smdefaultcornerreliefdiametertype		"-1"
smdefaultflangesplitextensiontype		"0"
smdefaultflangesplitextensionvalue		"0.1"
smdefaultflangesplitgaptype		"0"
smdefaultflangesplitgapvalue		"0.1"
smdefaultformfeatureunfoldmode		"4"
smDefaultHemGapType		"1"
smDefaultHemGapValue		"0.005"
smdefaultjunctionalignmenttorelief		"0"
smdefaultjunctiongaptype		"0"
smdefaultjunctiongapvalue		"0.001"
smdefaultkfactor		"0.27324"
smDefaultLoftedBendNumberSamples		"10"
smdefaultreliefextensiontype		"0"

System Variable Name	Ready-Only	Default Value
smdefaultreliefextensionvalue		"0.1"
smdefaulttribfilletradiustype		"0"
smdefaulttribfilletradiusvalue		"5"
smdefaulttribprofileradiustype		"0"
smdefaulttribprofileradiusvalue		"2"
smdefaulttribroundradiustype		"0"
smdefaulttribroundradiusvalue		"1"
smdefaultsharpbendradiuslimitratio		"5"
smDefaultTabChamferDistanceType		"0"
smDefaultTabChamferDistanceValue		"0.1"
smDefaultTabClearanceType		"0"
smDefaultTabClearanceCalue		"0.1"
smDefaultTabDistanceType		"0"
smDefaultTabDistanceValue		"20"
smDefaultTabEdgeType		"0"
smDefaultTabFilletRadiusType		"0"
smDefaultTabFilletRadiusValue		"0.1"
smDefaultTabHeightType		"0"
smDefaultTabHeightValue		"1"
smDefaultTabLengthType		"0"
smDefaultTabLengthValue		"4"
smDefaultTabSlotNumber		"3"
smdefaultthickness		"0.078740157480315"
smexportsmapproximationaccuracy		"0.000393701"
smexportsmminimaledgelengeth		"0.001968505"
smformfeaturesdowncolor		"6"
smformfeaturesdownlayerlinetype		"continuous"
smformfeaturesdownlayerlineweight		"-3"
smformfeaturesupcolor		"6"
smformfeaturesuplayerlinetype		"continuous"
smformfeaturesuplayerlineweight		"-3"
smjunctioncreatehealcoincident		"0"
smoverallannotationslayercolor		"3"
smoverallannotationslayerlinetype		"continuous"
smoverallannotationslayerlineweight		"-3"
smparametrizeholesparametrization		"3"
smrepairloftedbendmerge		"0"
smsmartfeatures		"3"
smSplitAmbiguousInput		"0"
smSplitconvertbendtojunction		"1"
smSplithealcoincident		"0"
smSplitorthogonalbendsplit		"0"
SMTARGETCAM		""
SNAPANG		0
SNAPBASE		0",0"
SNAPISOPAIR		0
SnapMarkerColor		20
SnapMarkerSize		6
SnapMarkerThickness		2
SNAPMODE		0
SNAPSTYL		0
SNAPTYPE		0
SNAPUNIT		1/2",1/2"
SOLIDCHECK		1
SORTENTS		127

System Variable Name	Read-Only	Default Value
spaAdjustMode		0
spaCheckLevel		10
spaGridAspectRatio		0
spaGridMode		1
spaMaxFacetEdgeLength		0
spaMaxNumGridLines		512
spaMinUGridLines		0
spaMinVGridLines		0
spaNormalTol		15
spaSurfaceTol		-1
spaTriangMode		1
spaUseFacetRES		1
SPLFRAME		0
SPLINESEGS		8
SPLINETYPE		6
SRCHPATH		"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Support\; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Fonts\; C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Help\en_US\"
SSFOUND		""
SSLOCATE		1
SSMAUTOOPEN		1
SSMPOLLTIME		15
SSMSHEETSTATUS		2
SSMSTATE		0
StackPanelType	r/o	0
Stairwidth		"1000"
StampFontSize		0.2
StampFontStyle		"Arial"
StampFooter		""
StampFooterOffsetX		0
StampFooterOffsetY		0
StampHeader		""
StampHeaderOffsetX		0
StampHeaderOffsetY		0
StampUnits		0
Statusbar		1
STARTUP		1
STEPSIZE		6
StlPositiveQuadrant		1
STEPSPERSEC		2
StructureTreeConfig		"mechanical.cst"
SURFTAB1		6
SURFTAB2		6
SURFTYPE		6
SURFU		6
SURFV		6
SvgBlendedGradients		0
SvgDefaultImageExtension		".png"
SvgGenericFontFamily		0
SvgHiddenLineRemoving		
SvgImageBase		""
SvgImageUrl		""
SvgLineWeightScale		1
SvgOutputHeight		768
SvgOutputWidth		1024

System Variable Name	Ready-Only	Default Value
SvgPrecision		6
SvgScaleFactor		0
SYSCODEPAGE	r/o	"ANSI_1252"

T Variables

TabControlHeight		25
TABMODE		0
TabsFixedWidth		0
TangentLengthType		"0"
TangentLengthValue		"0"
TARGET		0",0",0"
TDCREATE	r/o	2456335.5399919
TDINDWG	r/o	0.121354456
TDUCREATE	r/o	2456335.8733252
TDUPDATE	r/o	2456335.5399919
TDUSRTIMER	r/o	0.121354456
TDUPDATE	r/o	2456335.8733252
TeeTangentLengthType		"0"
TeeTangentLengthValue		"0.5"
TemplatePath		"C:\Users\userid\AppData\Local\Bricsys\BricsCAD\V20x64\en_US\Templates\"
TEMPPREFIX		""
TestFlags		0
TestGsFlags		0
TEXTANGLE		0
TEXTED		2
TEXTEDITMODE		0
TEXTEVAL		0
TEXTFILL		1
TEXTQLTY		50
TEXTSIZE		0.2
TEXTSTYLE		"Standard"
TextureMapPath		"C:\Program Files (x86)\Bricsys\BricsCAD V20x64\Textures\1\"
THICKNESS		0
ThreadDisplay		"0"
THUMBSIZE		1
TILEMODE		1
TILEMODELIGHTSYNCH		1
TIMEZONE		-8000
ToolbarMargin	r/o	0
ToolbuttonSize	r/o	0
TooliconPadding	r/o	0
Tips		1
ToolbarIconSize		
TOOLPALETTEPATH		"C:\Users\userid\AppData\Roaming\Bricsys\BricsCAD\V20x64\en_US\Support\ToolPalettes\"
TOOLTIPS		1
TPSTATE	r/o	0
TRACEWID		0.05
TRACKPATH		0
TRANSPARENCYDISPLAY		1
TrayIcons		1
TrayNotify		1
TrayTimeout		0
TreadLength		"290"
TREEDEPTH		3020
TREEMAX		10000000

System Variable Name	Read-Only	Default Value
TRIMMODE		1
TrustedPaths	r/o	""
TSPACEFAC		1
TSPACETYPE		1
TSTACKALIGN		1
TSTACKSIZE		70
TTFASSTEXT		3

U Variables

UCSAXISANG		90
UCSBASE		""
UCSDETECT		0
UCSFOLLOW		0
UCSICON		3
UCSICONPOS		0
UCSNAME	r/o	""
UCSORG	r/o	0",0",0"
UCSORTHO		1
UCSVIEW		1
UCSVP		1
UCSXDIR	r/o	1",0",0"
UCSYDIR	r/o	0",1",0"
UNDOCTL	r/o	5
UNDOMARKS	r/o	0
UNITMODE		0
UseBIM		
UseCommunicator		1
UseMechanical		
USER1 thru USER5		0
USERR1 thru USERR5		0
USERS1 thru USERS5		""
UseSheetMetal		
UseStandardOpenFileDialog		0

V Variables

VbaMacros		1
VENDORNAME	r/o	"Bricsys"
VerboseBimSectionUpdate		"1"
_VERNUM	r/o	"19.1.06 (UNICODE)"
VersionCustomizableFiles	r/o	"344"
VIEWCTR	r/o	10 7/16",4 1/2",0"
VIEWDIR	r/o	0",0",1"
VIEWMODE	r/o	0
VIEWSIZE	r/o	297
VIEWTWIST	r/o	0
VIEWUPDATEAUTO		1
VISRETAIN		1
VOLUMEPREC		-1
VOLUMEUNITS		"in ft mi μm mm cm m km"
VPROTATEASSOC		1
VSMAX	r/o	-1.0000E+20,-1.0000E+20,-1.0000E+20
VpMaximizedState	r/o	0
VSMIN	r/o	1.0000E+20,1.0000E+20,1.0000E+20
VTDURATION		750

System Variable Name	Ready-Only	Default Value
VTENABLE		3
VTFPS		7

W Variables

Wallwidth		"250"
WarningMessages		65535
WHIPARC		1
WHIPTHREAD		0
WINDOWAREACOLOR		150
WIPEOUTFRAME		1
WMFBKGND		0
WMFFOREGND		0
WNDLMAIN		2
WNDLSCRL		0
WDLSTAT		1
WDLTABS		
WDLTEXT		1
WNDPMAIN		0",0"
WDPTEXT		3'-4",3'-4"
WNSMAIN		101'-2",66'-11"
WNSTEXT		118'-4",86'
<i>WorkspaceSecurity</i>		
WORLDUCS		1
WORLDVIEW		1
WRITESTAT	r/o	1
WSAUTOSAVE		1
WSCURRENT		"2D Drafting"

X Variables

XCLIPFRAME		2
XDwgFadeCtl		70
XEDIT		1
XFADECTL		50
XLOADCTL		1
XLOADPATH		"C:\Users\userid\Documents\"
XNotifyTime		5
XREFCTL		0
XRefNotify		1
XREFOVERRIDE		0

Z Variables

ZOOMFACTOR		60
ZOOMWHEEL		0

Variables

3DCOMPAREMODE		3
3DOSMODE		11
3dSnapMarkerColor		5

Concise DCL Reference

DCL allows you to create the these elements in dialog boxes: buttons, popup lists, text edit boxes, radio buttons, image buttons, sliders, list boxes, and toggles.

These elements are called *tiles*, and can be clustered together as dialog boxes, boxed columns, boxed radio columns, radio columns, boxed radio rows, radio rows, boxed rows, rows, and columns. To make dialog boxes prettier and show graphical information, you can add these elements: images, spacer 0, text, spacer 1, and spacer.

The *base.dcl* file defines numerous basic tiles, such as the OK button, so that you don't need to write them from scratch.

Each tile works with one or more attributes. Attributes specify the look of the tile and how it works. For instance, the label tile specifies the text that appears on buttons. A special attribute, called "key," allows LISP code to communicate back to the dialog box and make changes, such as changing the text displayed by the dialog box's title bar.

This appendix describes every tile and its associated attributes, as well as the LISP functions that are specific to dialog boxes.

In this reference, the default value of attributes is shown in **boldface**. For example,

```
alignment = left | right | centered;
```

shows that "left" is the default value for the Alignment attribute.

The information in this reference applies equally to BricsCAD running on Linux, Mac, and Windows. When there are differences, the Linux and Mac portions are highlighted in gray.

QUICK REFERENCE OF DCL TILE NAMES

<code>boxed_column</code>	Draws a rectangle around a vertical column of tiles.
<code>boxed_radio_column</code>	Draws a rectangle around a vertical column of radio tiles.
<code>boxed_radio_row</code>	Draws a rectangle around a horizontal row of radio tiles.
<code>boxed_row</code>	Draws a rectangle around a horizontal row of tiles.
<code>button</code>	Displays a button with text.
<code>column</code>	Creates a column of tiles.
<code>dialog</code>	Creates a dialog box.
<code>default_dcl_settings</code>	Sets the level of debugging.
<code>edit_box</code>	Displays a text edit box.
<code>image</code>	Displays a static image.
<code>image_button</code>	Displays a button with an image.
<code>list_box</code>	Displays a list.
<code>paragraph</code>	Concatenates text tiles into vertical paragraphs.
<code>popup_list</code>	Displays a droplist.
<code>radio_button</code>	Displays a round radio button.
<code>radio_column</code>	Creates a column of radio buttons.
<code>radio_row</code>	Creates a row of radio buttons.
<code>row</code>	Creates a row of tiles.
<code>slider</code>	Displays a vertical or horizontal slider bar.
<code>spacer</code>	Inserts a rectangular space.
<code>spacer_0</code>	Inserts variable-width space.
<code>spacer_1</code>	Inserts narrow space.
<code>text</code>	Displays static text.
<code>text_part</code>	Contains a piece of text.
<code>toggle</code>	Displays a square checkbox.

DIALOG

The **dialog** tile defines dialog boxes.

```
name : dialog {  
    label = "text";  
    value = "text";  
    initial_focus = "key";  
}
```

NAME

The **name** attribute identifies dialog boxes by name. This allows you to have define all dialog boxes in a single DCL file. The LISP routine that accompanies the DCL file uses the **load_dialog** function to load the *filename.dcl* file, and then uses **new_dialog** to locate the specific dialog box, as follows:

```
(setq dlg-id (load_dialog "c:\\filename"))  
(new_dialog "name" dlg-id)
```

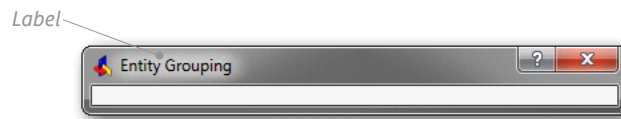
TIPS The *dlg-id* variable holds the system-assigned identifier for the DCL file. This is typically a number, such as 30.

If the number has a negative value, such as -1, then the DCL file failed to load correctly. You can use this number to generate error reports.

LABEL

The **label** attribute displays text on the dialog box's title bar, such as:

```
label = "Dialog Box";
```



The **value** attribute is nearly the same, because it also displays text on the title bar. The difference is that you can use the LISP **set_value** function to later change the title.

TIP You can change the dialog box's title when the accompanying LISP routine is run. This is useful, for example, with a file dialog box whose title should reflect the extension of the file extension being accessed. To change the text, use the (**set_tile key value**) function, which changes the value of the tile specified by *key*.

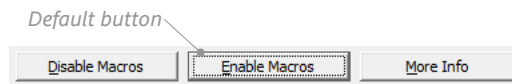
The problem is that BricsCAD cannot widen the dialog box to accommodate long titles. To avoid cutting off some of the **value** text, specify a long title with **label**.

QUICK REFERENCE OF DCL ATTRIBUTES

action	LISP action expression.
alignment	Horizontal or vertical position in a cluster.
allow_accept	Activates the is_default attribute when tile is selected.
aspect_ratio	Aspect ratio of an image.
audit_level	Specifies the level of debugging.
big_increment	Incremental distance to move.
children_alignment	Alignment of a cluster's children.
children_fixed_height	Height of a cluster's children doesn't grow during layout.
children_fixed_width	Width of a cluster's children doesn't grow during layout.
color	Background (fill) color of an image.
edit_limit	Maximum number of characters that can be entered.
edit_width	Width of the input field of the tile.
fixed_height	Prevents height from shrinking.
fixed_width	Prevents width from shrinking.
fixed_width_font	Displays text in a fixed pitch font.
height	Height of the tile.
initial_focus	Key of the tile with initial focus.
is_bold	Displays as bold.
is_cancel	Reacts to the cancel key (Esc).
is_default	Reacts to the accept key (Enter).
is_enabled	Tile is initially enabled.
is_tab_stop	Tile is a tab stop.
key	Tile name used by the application.
label	Displayed label of the tile.
layout	Whether the slider is horizontal or vertical.
list	Initial values to display in list.
max_value	Maximum value of a slider.
min_value	Minimum value of a slider.
mnemonic	Mnemonic character for the tile.
multiple_select	List box allows multiple items to be selected.
password_char	Masks characters entered in edit_box.
small_increment	Incremental distance to move.
tab_truncate	Truncates text longer larger than the associated tab stop.
tabs	Tab stops for list display.
value	Tile's initial value.
width	Width of the tile.

INITIAL FOCUS

The **initial_focus** attribute indicates which button or other tile gets the *focus*. "Focus" refers to the tile that is highlighted, the one that would be activated when you press the **Enter** key.



Usually, the focus is set to the OK button, or to the tile users are likely to assess the most.

KEY

The value of focus is the name of the **key** tied to the tile. For example, when the key of the OK button is "okButton," then you enter the following:

```
initial_focus = "okButton";
```

Exiting Dialog Boxes

Every dialog box must have at least an **OK** button, so that users can exit the dialog box. You can use predefined buttons to give your dialog boxes the same look as those of Bricsys-designed dialog boxes. These are called "subassemblies," and are found in *base.dcl*.

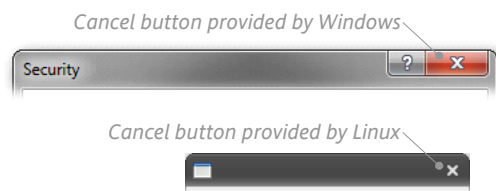
For instance, to include the standardized  button, use the **ok_only** subassembly like this:

```
name : dialog {  
    label = "Dialog Box";  
    ok_only;  
}
```

Notice that subassemblies are not prefixed by the ' : ' character. (If BricCAD complains about the *ok_only* subassembly, then you need to load the *base.dcl* file with the following bit of code:

```
Command: (load_dialog "base.dcl")
```

Both OK and Cancel exit the dialog box, but they have different meanings:



- **OK** records changes made by users.
- **Cancel** discards the changes.

After the dialog box is exited, BricCAD sets the read-only **DiaStat** system variable (short for "dialog box status") to one of the following values:

DiaStat	Meaning
0	User clicked Cancel to exit dialog box.
1	User clicked OK to exit dialog box.

QUICK REFERENCE OF LISP FUNCTIONS FOR DIALOG BOXES

.....

(action_tile)	Assigns action to be evaluated when user selects dialog box tile.
(add_list)	Adds and modifies strings in current dialog box listbox.
(client_data_tile)	Associates data from an application with a tile in the dialog box.
(dimX_tile)	Returns the x-dimension of the dialog box image tile.
(dimY_tile)	Returns the y-dimension of the dialog box image tile.
(done_dialog)	Terminates the dialog box.
(end_image)	Ends creation of dialog box image tile.
(end_list)	Ends processing of dialog box list box.
(fill_image)	Draws filled rectangles in dialog boxes.
(get_attr)	Retrieves the DCL value of the tile's attribute.
(get_tile)	Retrieves the value of tile.
(load_dialog)	Loads .dcl files that define dialog boxes.
(mode_tile)	Sets the mode of dialog box tiles.
(new_dialog)	Activates dialog boxes.
(set_tile)	Sets the value of dialog box tiles.
(slide_image)	Displays slides in dialog box image tiles.
(start_dialog)	Displays the current dialog box.
(start_image)	Starts creating images in dialog boxes.
(start_list)	Starts processing lists in dialog boxes.
(term_dialog)	Terminates dialog boxes.
(unload_dialog)	Unloads .dcl files from memory.
(vector_image)	Draws vectors in dialog box image tiles.

QUICK REFERENCE OF DIALOG BOXES DISPLAYED BY LISP FUNCTIONS

.....

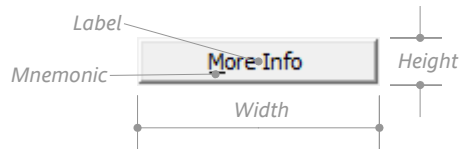
(acad_colordlg)	Displays the Select Color dialog box with only the Index Color tab.
(acad_helpdlg)	[obsolete] Displays the Help dialog box.
(acad_truecolordlg)	Displays the Select Color dialog box with all tabs.
(alert)	Displays the alert dialog box with customized warning.
(help)	Displays the Help window.
(initdia)	Forces display of the next command's dialog box.

At the right end of the dialog box's title bar is an **x** button. It is equivalent to **Cancel**, and users can use it in place of Cancel.

If the dialog box is tight on space, you can leave out the Cancel button, and let users use the x button on the dialog box's title bar; just remember to include the OK button.

BUTTON

The **button** tile defines buttons with text labels.



```
: button {  
    label = "text";  
    mnemonic = "char";  
  
    action = "(LISP function)";  
    key = "text";  
  
    is_cancel = false | true;  
    is_default = false | true;  
    is_enabled= true | false;  
    is_tab_stop= true | false;  
  
    width = number;  
    height = number;  
    fixed_height = false | true;  
    fixed_width = false | true;  
  
    alignment = centered | left | right;  
}
```

LABEL

The **label** attribute places text on the button tile, such as:

```
label = "More Info";
```

MNEMONIC

The **mnemonic** attribute underlines a character. Users can then access the button by pressing **Alt** and the letter. For example, the following code underlines the letter "M" in More Info:

```
mnemonic = "M";
```

SUMMARY OF TILE REFERENCES

This appendix lists DCL's tiles and attributes in order of importance, as follows:

Dialog

Button

- Ok_Only
- Ok_Cancel
- Ok_Cancel_Help
- Ok_Cancel_Help_Errtile
- Ok_Cancel_Help_Info

Radio_Button

Toggle

Image_Button

Edit_Box

List_Box

Popup_list

Slider

Text

- Text_Part
- Concatenation
- Paragraph
- Errtile

Spacer

- Spacer_0
- Spacer_1
- Image

Column

Row

- Boxed_Column
- Boxed_Row
- Radio_Column
- Radio_Row
- Boxed_Radio_Column
- Boxed_Radio_Row

TIPS As an alternative to the mnemonic attribute, you can prefix characters with **&** in the label, like this:
label = "&More Info";
Make sure that mnemonic characters are not used more than once in each dialog box. For instance, don't use M twice in the same dialog box.

ACTION

The **action** attribute contains LISP code that gets executed when users click the button. (This is called a "callback.") For example, the code could set the value of system variables, like this:

```
action = "(setvar "highlight" 0)"
```

TIPS You cannot, unfortunately, use the LISP **command** function to execute BricsCAD commands with the **action** attribute.

You can use the LISP **action_tile** function to override the action specified by the **action** attribute.

KEY

The **key** attribute gives an identifying tag to the button.

IS_CANCEL

The **is_cancel=true** attribute specifies that this button is selected when users press the **Esc** key.

```
is_cancel = true;
```

Usually, the dialog box is exited right away when users press **Esc**. In addition, BricsCAD sets the value of the **DiaStat** system variable to **0**. However, if the button has an **action** attribute, then the associated LISP expression is executed before the dialog box is exited.

TIPS Only one button in the dialog box can be assigned **is_cancel=true**. There is no point in having **is_cancel=false**, except for debugging perhaps.

IS_DEFAULT

The **is_default=true** attribute specifies that this button is selected when users press the **Enter** key — unless another button has the focus.

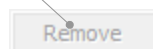
```
is_default true;
```

IS_ENABLE

The **is_enable=false** attribute allows you to gray-out buttons. This tells users that the buttons are unavailable, usually because some other condition is not satisfied, such as the drawing is in paper space instead of model space.

```
is_enabled= false;
```

Grayed-out text



When set to true, the buttons become available. To change the status from false to true, use the **mode_tile** function in LISP.

IS_TAB_STOP

The **is_tab_stop** attribute allows the button to receive focus when users press the **Tab** key. Pressing Tab is a popular way for power users to quickly move through the controls of dialog boxes; if the mouse is busted, then that's the only way to navigate a dialog box.

```
is_tab_stop= false;
```

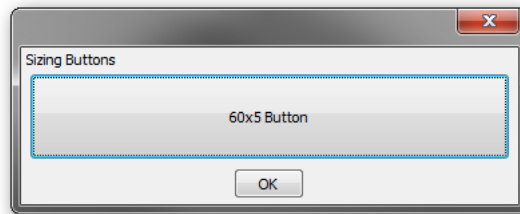
Normally, there is no reason **not** to allow a button to be a tab stop, and since the default is true, there's not much need for this attribute.

WIDTH & HEIGHT

The **width** and **height** attributes specify the minimum size of buttons. You can use integers (such as 5) or real numbers (such as 5.5).

Usually BricsCAD determines the correct size on its own, so you don't need to specify these attributes. But if you need to create extra large buttons, such as the one illustrated below, then go right ahead!

```
width = 60;  
height = 5;
```



The units are in characters, such as 60 characters wide and 5 lines tall. BricsCAD determines the size of character based on an average calculated of all letters in the 8-pt "MS San Serif" font used by Windows for text in dialog boxes. The font cannot be changed.

In the table below, the black areas indicate size of tiles based on a variety of values for the **Width** and **Height** attributes:

Size	Example
Height = 1	
Height = 2	
Height = 3	



FIXED_HEIGHT & FIXED_WIDTH

The **fixed_height** and **fixed_width** attributes prevent BricsCAD from expanding buttons to fill the available space. Recall that the **height** and **width** attributes specify only the minimum size, and so adding these two attributes also specifies the maximum size.

```
fixed_height = true;  
fixed_width = true;
```

TIP Use the **image_button** tile for buttons with colors and images.

ALIGNMENT

The **alignment** attribute is supposed to shift text left or right on the button. In practice, however, I find this attribute has no effect; the text is always centered.

```
alignment = left | right | centered;
```

Bricsys notes that alignment cannot be specified along the long axis of a cluster of tiles. The first and last tiles align with either end of the row (or column), while the inner tiles are distributed evenly between them. You can change the distribution with the **spacer_0** tile.

PREFABRICATED BUTTON ASSEMBLIES

Bricsys provides the following pre-fabricated button assemblies in *base.dcl*. This file is described in detail later in this chapter.

OK_ONLY

The **ok_only** tile defines the OK button.

```
ok_only;
```



OK_CANCEL

The **ok_cancel** tile defines a horizontal row of the OK and Cancel buttons.

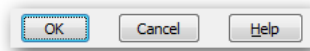
```
ok_cancel;
```



OK_CANCEL_HELP

The **ok_cancel** tile defines a horizontal row of OK, Cancel, and Help buttons.

```
ok_cancel_help;
```



OK_CANCEL_HELP_ERRTILE

The **ok_cancel_help_errtile** tile defines a horizontal row of OK, Cancel, and Help buttons, and space below for an error message.

```
ok_cancel_help_errtile;
```



OK_CANCEL_HELP_INFO

The **ok_cancel_help_info** tile defines a horizontal row of OK, Cancel, Help, and Info buttons. The **Info** button can be used to display a second dialog box with additional information.

```
ok_cancel_help_info;
```



RADIO_BUTTON

The **radio_button** tile defines radio buttons. These buttons are used when only one choice can be made from a selection, such as the top, left, or right isoplane. When selected, the radio button shows a black dot; when off, the round button is blank.

If the dialog box has more than one radio button in a cluster, only one can be on at a time. When users select a radio button, the other one turns off automatically. To have more than one radio button on at a time, segregate them into clusters with the **radio_row** or **radio_column** tiles.



```

: radio_button {
  action = "(LISP function)";
  key = "text";

  label = "text";
  mnemonic = "char";
  value = "0" | "1";

  is_enabled= true | false;
  is_tab_stop= true | false;

  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;

  alignment = left | right | centered;
}

```

LABEL

The **label** attribute describes the purpose of the radio button to users. The text is always to the right of the button.

VALUE

The **value** attribute determines whether the radio button is initially on or off:

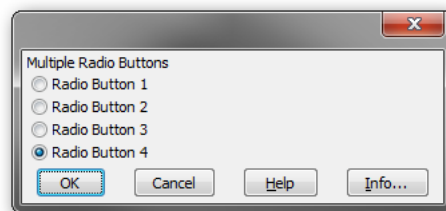
```
value = "1";
```

Value	Meaning	Example
0	Off	<input type="radio"/> Radio Button Off
1	On	<input checked="" type="radio"/> Radio Button On

When you leave out the **value** attribute, then BricsCAD turns on the first radio button. The other radio button attributes have the same meanings as for the **button** tile.

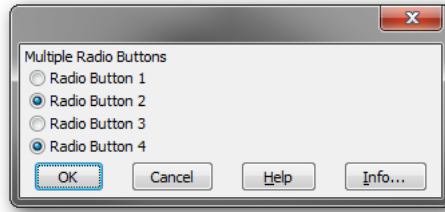
Multiple Radio Buttons

When more than one radio button has **value** set to **1**, then BricsCAD turns on only the last one. as illustrated below.



If you need more than one radio button to be turned on, then use check boxes instead. Use the **toggle** tile to make check boxes.

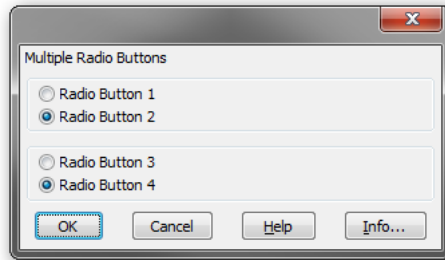
I mean, if you really want a dialog box to show two or more radio buttons turned on, then the workaround is to segregate radio buttons with the **radio_column** tile, as illustrated below.



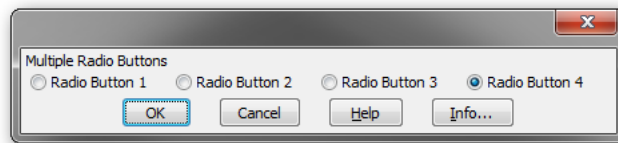
Here's the DCL code:

```
: radio_column {
  : radio_button { label = "Radio Button 1"; value = "1"; }
  : radio_button { label = "Radio Button 2"; value = "1"; }
}
: radio_column {
  : radio_button { label = "Radio Button 3"; value = "1"; }
  : radio_button { label = "Radio Button 4"; value = "1"; }
}
```

It's not clear that the four buttons are segregated into two sections, so it makes sense to replace **radio_column** with the **boxed_radio_column** tile to separate them visually.



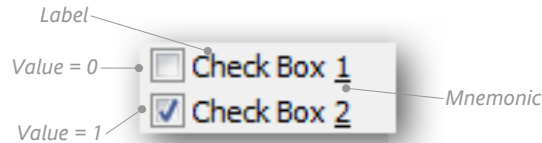
By default, BricsCAD stacks radio buttons vertically, as shown above. You can use the **radio_row** tile to force the radio buttons in a horizontal line — although this format is more difficult for users to navigate visually.



More on the **boxed_** and **radio_** tiles later in this appendix.

TOGGLE

The **toggle** tile defines check boxes. Check boxes are employed so users can select more than one choice at a time. (Use radio buttons to limit options to a single choice.)



```
: toggle {  
  action = "(LISP function)";  
  key = "text";  
  label = "text";  
  mnemonic = "char";  
  value = "0" | "1";  
  is_enabled= true | false;  
  is_tab_stop= true | false;  
  height = number;  
  width = number;  
  fixed_height = false | true;  
  fixed_width = false | true;  
  alignment = left | right | centered;  
}
```

LABEL

The **label** attribute describes to users the purpose of the check box. The text is always to the right of the button.

VALUE

The **value** attribute determines whether the toggle is initially on or off:

```
value = "1";
```

Value	Meaning	Example
0	Off	<input type="checkbox"/> Check Box 1
1	On	<input checked="" type="checkbox"/> Check Box 2

When check boxes lack the **value** attribute, then they are all turned off by default.

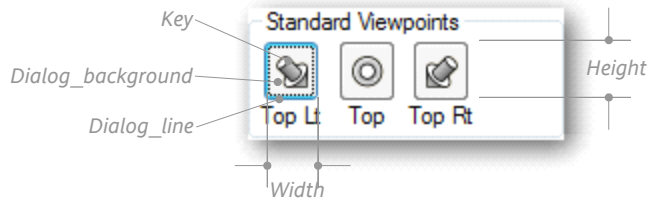
OTHER ATTRIBUTES

The other attributes have the same meaning as for the **radio** tile.

You can use the **boxed_row** and **boxed_column** tiles to segregate toggles into groups.

IMAGE_BUTTON

The **image_button** tile defines a button tile with an image. This can be the difficult to program, because some situations require you to correlate x,y coordinates from users' picks with LISP code.



```
: image_button {
  action = "(LISP function)";
  key = "text";

  aspect_ratio = number;
  mnemonic = "char";
  color = colornumber;

  allow_accept = false | true;
  is_enabled= true | false;
  is_tab_stop= true | false;

  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;
  alignment = left | right | centered;
}
```

KEY

The **key** attribute identifies the **image** tile to the accompanying LISP code, so that the slide image can be placed in the dialog box.

```
key = "image1";
```

Images of hatch patterns, fonts, and so on are placed on the image tile through the accompanying LISP code's callback function (**set_tile**) and the **key** attribute. There are two sources of image you can use:

- SLD slide files, which are created ahead of time with BricsCAD's **MSlide** command, and then placed with LISP's **slide_image** function.
- Vector lines, which are drawn on-the-fly by LISP's **vector_image** function.

ASPECT_RATIO, HEIGHT, & WIDTH

You use any two of these three attributes. The **aspect_ratio** attribute specifies the ratio between the height and width of the image tile, and must be used with either the **height** or the **width** attribute — but not both. Similarly, if you use the **height** and **width** attributes, you cannot use the **aspect_ratio** attribute.

Examples:

```
aspect_ratio = 1.333;  
height = 3;
```

Or..

```
aspect_ratio = 1.333;  
width = 4;
```

Or..

```
height = 3;  
width = 4;
```

COLOR

The **color** attributes specifies the background color of image tiles. You can use a color name or number; default = 7 (white or black).

#	Color Name	Meaning
0	black	ACI color 0 (black or white) ¹
1	red	ACI ² color 1
2	yellow	ACI color 2
3	green	ACI color 3
4	cyan	ACI color 4
5	blue	ACI color 5
6	magenta	ACI color 6
7	white	ACI color 7 (white or black) ¹
-1	graphics_foreground	Current default color of entities (usually ACI 7). ¹
-2	graphics_background	Current background color of BricsCAD's graphics screen.
-3	blue	
-4	black	
-5	gray	
-6	black	
-7	red	
-15	dialog_background	Current color of dialog box background (usually gray).
-16	dialog_foreground	Current color of dialog box text (usually black).
-18	dialog_line	Current color of dialog box lines (usually black).

Notes:

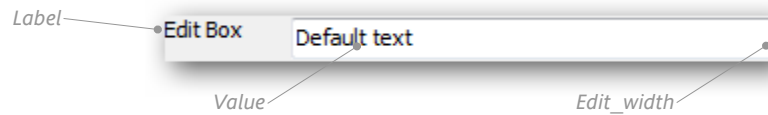
¹ The color is white when the background color is dark, but black when the background is light.

² ACI is short for "BricsCAD Color Index," and refers to the 256 color numbers.

Autodesk notes that "if your image tile is blank when you first display it, try changing its color to graphics_background or graphics_foreground."

EDIT_BOX

The **edit_box** tile defines a horizontal tile for entering text.



```
: edit_box {
  label = "text";
  mnemonic = "char";
  action = "(LISP function)";
  key = "text";
  value = "text";
  fixed_width_font = false | true;
  password_char = "char";
  edit_limit = 1-256;
  edit_width = 1-256;

  allow_accept = false | true;
  is_enabled= true | false;
  is_tab_stop= true | false;

  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;

  alignment = left | right | centered;
}
```

LABEL

The **label** attribute displays text that prompts users as to the text or numbers to enter. The label is always positioned to the left of the text entry box.

```
label = "Edit Box";
```

MNEMONIC

The **mnemonic** attribute provides the **Alt**+shortcut keystroke for the label. Alternatively, prefix a letter in the label with **&**.

```
mnemonic = "E";
```

or

```
label = "&Edit Box";
```

VALUE

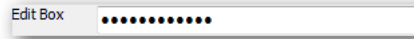
The **value** attribute displays default text in the edit box, such as "Default text" in the figure above. For a blank, leave it out, or use **value = ""**.

```
value = "Default text";
```


PASSWORD_CHAR

When the edit box is used for entering passwords, then you can specify a character with the **password_char** attribute that substitutes for user-entered text, such as "*".

```
password_char = "*";
```



FIXED_WIDTH_FONT

The **fixed_width_font** attribute determines whether the edit box uses a fixed width font; more precisely, the monospaced FixedSys font included with Windows. (This attribute is not documented.) Only the user text is affected by this attribute; the dialog box text keeps its font.

```
fixed_width_font = true;
```



EDIT_LIMIT

The **edit_limit** attribute limits the maximum number of characters users can type in. For text, the limit usually doesn't matter; the default is 132. You might want to expand the limit to its maximum of 256, or reduce it. For example, you may want to limit entry to a single character or digit.

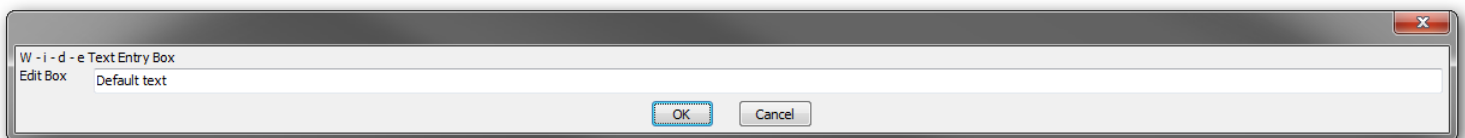
```
edit_limit = 256;
```

EDIT_WIDTH

The **edit_width** attribute determines the size of the edit box; it can be an integer or a real number. Users can enter more characters than this number, up to the maximum determined by **edit_limit**. The default width is whatever fits in the dialog box; specifying **edit_width = 0** has the same effect. In many cases, the default width is about 16 characters.

```
edit_width = 186;
```

I have found that the maximum value of 256 can overwhelm:

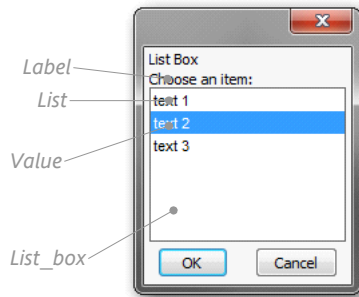


OTHER ATTRIBUTES

The remaining attributes have the same meaning as for other tiles.

LIST_BOX

The **list_box** tile defines tiles that list text items; users can select one or more of them.



```
: list_box {
  action = "(LISP function)";
  key = "text";

  label = "text";
  mnemonic = "char";

  list = "text 1\n\text 2\n\text 3";
  value = "0";
  multiple_select = false | true;;
  tabs = "number number number";
  tab_truncate = false | true;
  fixed_width_font = false | true;

  allow_accept = false | true;
  is_enabled = true | false;
  is_tab_stop = true | false;

  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;

  alignment = left | right | centered;
}
```

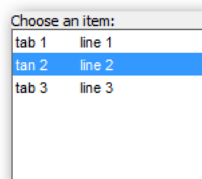
LIST

The **list** attribute specifies the text in the list box tile. Each item is separated by the **\n** metacharacter, which means "new line." When the list becomes too long for the list box, BricsCAD automatically adds a scroll bar, as illustrated later.

```
list = "text 1\n\text 2\n\text 3";
```

TABS

You can use the **tabs** attribute to line up text in list boxes. The tabs are specified in characters, such as at the 5th, 10th, 15th, and 20th character.



```
tabs = "5 10 15 20";
```

To specify tabs in the text of the **list** attribute, use the `\t` metacharacter (short for "tab"). The following DCL code and figure illustrate the use of `\n` and `\t`:

```
list = "tab 1\tline 1\n\tan 2\tline 2\n\tab 3\tline 3";
```

TAB_TRUNCATE

The **tab_truncate** attribute determines whether text is truncated when longer than the associated tab stop. Default is false, which means text is not truncated.

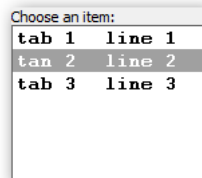
```
tab_truncate = true;
```

FIXED_WIDTH_FONT

The **fixed_width_font** attribute lets the list use the Windows FixedSys font, a *monospace* font (a.k.a. fixed width font), where each character takes up the same width. This can be useful when you need columns of text to line up; otherwise, fixed width text is not useful, because it makes the dialog box wider. (This attribute is undocumented.)

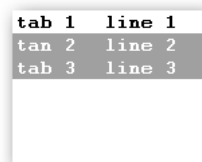
```
fixed_width_font = true;
```

In the figure below, both dialog boxes have **width = 30**. The fixed width font takes up more space.



VALUE

The **value** attribute specifies which item is initially highlighted. The default, **0**, means the first item is highlighted. If you want more than one item highlighted, then separate the digits with spaces.

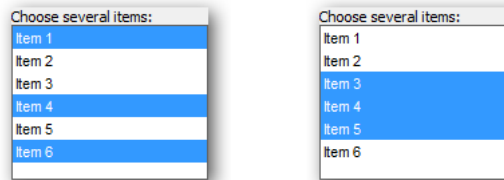


The following examples highlights items #2 and #3:

```
value = "1 2";
```

Multiple_Select

The **multiple_select** attribute determines whether users can select more than one item from the list. Users need to hold down the **Ctrl** key to select more than one item, or the **Shift** key to select a sequential range of items.



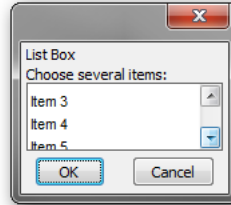
*Left: Selecting random items with the **Ctrl** key held down.
Right: Selecting sequential items with the **Shift** key held down.*

```
multiple_select = false;
```

When this attribute is set to false (the default setting), then the **value** attribute is restricted to the first digit. For example, **value = "1 2"** becomes **"1"**.

HEIGHT

The **height** attribute determines the height of the list box in lines. For example, **height = 7** means that the list box is seven lines tall, but has room for only six items, because the seventh line is used for the label.



When **height** is set to **0** or is not included, then the list box is stretched to accommodate all items in the list, if possible.

WIDTH

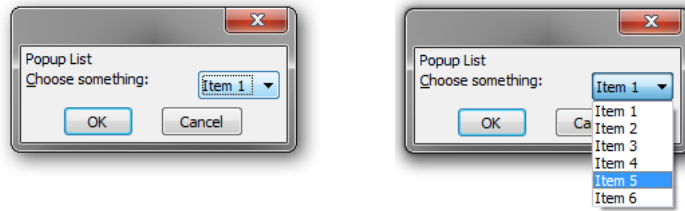
The **width** attribute determines the width of the list box. Width is measured in characters.

OTHER ATTRIBUTES

The remaining attributes operate identically to those in other tiles.

POPUP_LIST

The **popup_list** tile displays a droplist. Despite the name, this tile drops *down*, not up.



*Left: Popup list before...
Right: ...and after being selected by the user.*

```
: popup_list {  
  action = "(LISP function)";  
  key = "text";  
  
  label = "text";  
  mnemonic = "char";  
  
  list = "text 1\n\ntext 2\n\ntext 3";  
  tabs = "number number number";  
  tab_truncate = false | true;  
  value = "text";  
  fixed_width_font = false | true;  
  edit_width = 1-256;  
  
  is_enabled= true | false;  
  is_tab_stop= true | false;  
  
  height = number;  
  width = number;  
  fixed_height = false | true;  
  fixed_width = false | true;  
  
  alignment = left | right | centered;  
}
```

LABEL

The **label** attribute provides the prompt text for the droplist.

```
label = "Popup list: ";
```

MNEMONIC

As with other tiles, you can specify the **Alt**+shortcut with the **&** prefix, or else use the **mnemonic** attribute to indicate the shortcut keystroke.

```
label = "&Popup list: ";  
mnemonic = "P";
```

LIST

The **list** attribute specifies the text in the droplist tile. Each item is separated by the **\n** metacharacter. When the list becomes too long for the droplist, BricsCAD automatically adds a scroll bar.

```
list = "text 1\n\ntext 2\n\ntext 3";
```

TABS

If you need text to line up in columns, use the **tabs** attribute to specify the tab spacing.

```
tabs = "10 20 30";
```

Then use the **\t** metacharacter to specify where the tabs occur in the **list** attribute.

```
list = "text 1\ttext 2\ttext 3";
```

TAB_TRUNCATE

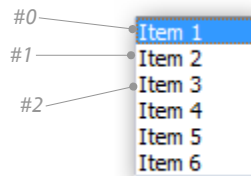
The **tab_truncate** attribute determines whether text is truncated when longer than the associated tab stop. Default is false, which means it is not truncated.

```
tab_truncate = true;
```

VALUE

The value attribute specifies which item is initially selected. The first item is #0 (the default). Use **value = ""** for no initial selection.

```
value = "1";
```

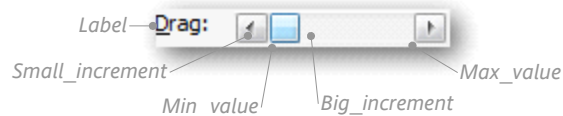


OTHER ATTRIBUTES

The other attributes are identical to those described for other tiles.

SLIDER

The **slider** tile defines vertical and horizontal sliders.



```
: slider {  
  action = "(LISP function)";  
  key = "text";  
  label = "text";  
  mnemonic = "char";  
  layout = horizontal | vertical;  
  max_value = integer;  
  min_value = integer;  
  big_increment = integer;  
  small_increment = integer;
```

```

    value = "text";
    height = number;
    width = number;
    fixed_height = false | true;
    fixed_width = false | true;
    alignment = left | right | centered;
}

```

LABEL & MNEMONIC

The **label** and **mnemonic** attributes name the slider. Alternatively, you could use the **boxed_row** attribute to give the slider its label, as illustrated below:

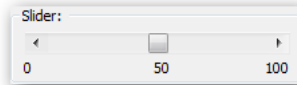


```

: boxed_row {
    label = "Slider: "; mnemonic = "S";
    : slider {
        max_value = 100;
        min_value = -100;
        big_increment = 10;
        small_increment = 1;
        value = "0";
    }
}

```

In addition to not labeling the slider, this tile provides no way to indicate to users the meaning of the minimum and maximum values. The workaround is to add a row of **text** underneath the slider, as illustrated here.



Notice that you need to use the **spacer** tile to position the three pieces of text appropriately:

```

: row {
    : text { value = "-100"; alignment = left; }
    : spacer {width = 11; }
    : text { value = "0"; alignment = centered; }
    : spacer {width = 8; }
    : text { value = "100"; alignment = right; }
}

```

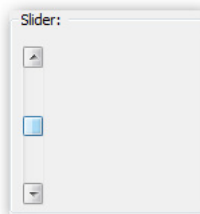
LAYOUT

The **layout** attribute determines if the slider is horizontal (default) or vertical, as illustrated below.

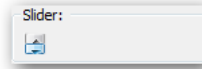
```

layout = vertical;

```



Horizontal sliders don't need to have a height or width attribute, because the default values are just fine. Vertical sliders, need the height specified, otherwise they end up with no height, as illustrated below. I suggest setting **height = 10**.



Using both a horizontal and vertical slider lets you create scroll bars for panning images.

MAX_VALUE

The **max_value** attribute specifies the upper limit of the scroll bar; default = 10000. It limits the maximum value when the slider is at the right (or top) end of the bar. You can use any integer between -32768 and 32767. If you need larger values, then use LISP code to multiply them.

```
max_value = 32767;
```



MIN_VALUE

The **min_value** attribute specifies the lower limit of the scroll bar; default = 0. It limits the minimum value when the slider is at the left (or bottom) end. You can use any integer between -32768 and 32767. To reverse the action of the scroll bar, use a larger value for **min_value** and a smaller one for **max_value**.

```
min_value = -32768;
```

BIG_INCREMENT

The **big_increment** attribute specifies the value of clicking the bar on either side of the slider. The default is 0.1 of the range between **max_value** and **min_value**. You can use any integer between the values of those two attributes.

```
big_increment = 100;
```



SMALL_INCREMENT

The **small_increment** attribute specifies the value of clicking the arrows. The default is 0.01 of the range between **max_value** and **min_value**.

```
small_increment = 1;
```


VALUE

The **value** attribute specifies the slider's initial position. Even though the value is an integer, it must be enclosed in quotation marks. The default is the same as **min_value**.

```
value = "1000";
```

HEIGHT

The **height** attribute specifies the size of vertical sliders; it has no effect on horizontal sliders. Height is measured in lines (of text). You have to specify a height for vertical sliders to avoid the problem described on the previous page.

```
height = 10;
```

WIDTH

The **width** attribute specifies the size of horizontal sliders; it has no effect on vertical sliders. Width is measured in character (of text). You don't have to specify a width for horizontal sliders, because the default is satisfactory.

```
width = 40;
```

FIXED_HEIGHT & FIXED WIDTH

The **fixed_height** and **fixed_width** attributes prevent DCL from expanding the slider to fit available space in the dialog box. Default in both cases is false, which means the height and width are not fixed. I suspect these attributes actually have no effect.

```
fixed_height = true;  
fixed_width = true;
```

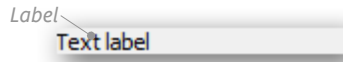
ALIGNMENT

The **alignment** attribute is supposed to shift the slider bar left or right, but I don't see that this attribute has any effect. The default is centered.

```
alignment = right;
```

TEXT

The **text** tile displays text in the dialog box. The text is static when specified in the DCL file, or dynamic when specified in the LSP file.



```
: text {
  label = "text";
  is_bold = false | true;
  value = "text";
  key = "text";

  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;

  alignment = left | right | centered;
}
```

LABEL

The **label** attribute specifies the text displayed by the dialog box. It is recommended that use this attribute for *static* text — text that doesn't change.

```
label = "Text label";
```

VALUE

The **value** attribute also specifies text displayed by dialog box. Bricsys recommends you use this attribute for *dynamic* text — text that's specified by the accompanying LISP code. For dynamic text, **value** is set to *null*, as shown here:

```
value = "";
```

Make sure you include the **width** attribute so that there is sufficient space for the text message. BricsCAD does not wrap text that is too long for the dialog box; text is truncated. And include the key attribute so that the LISP code can identify the text tile.

```
: text {
  value = "";
  key = "textField1";
  width = 40;
}
```

To display error messages or feedback on users' choices, use the **set_tile** function to assign text to the tile in the LISP code, like this:

```
(set_tile "textField1" "Error: Cannot set that value.")
```

The combination of DCL and LSP code results in the following display by the dialog box:

Error: Cannot set that value.

TIP If both **label** and **value** are used in the **text** tile code, however, then **value**'s text is displayed by the dialog box.

IS_BOLD

The **is_bold** attribute should boldface the text, but appears to not work.

```
is_bold = true;
```

HEIGHT & WIDTH

The height and width attributes size the text tile. Height starts measuring from the top of the text, and is measured in lines. Width starts from the left end of the text, and is measured in characters.

```
height = 5;  
width = 40;
```



FIXED_HEIGHT & FIXED WIDTH

The **fixed_height** and **fixed_width** attributes prevent DCL from expanding the text area to fit available space in the dialog box. Default in both cases is false, which means the height and width are not fixed.

```
fixed_height = true;  
fixed_width = true;
```

ALIGNMENT

The **alignment** attribute shifts the text to the left, right, or center of its width.

```
alignment = right;
```

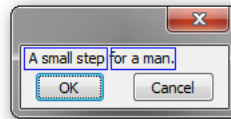
TEXT_PART

The **text_part** tile displays text without *margins*, the blank space around text. It is meant to combine several pieces of text into one line, when used with the **concatenation** tile.

```
: text_part {  
    label = "text";  
}
```

CONCATENATION

The **concatenation** tile strings together two or more **text** and/or **text_part** tiles. In the figure below, I have outlined in blue the two sections of text.



```
: concatenation {  
    : text_part { label = "A small step"; }  
    : text_part { label = "for a man."; }  
}
```

PARAGRAPH

The **paragraph** tile stacks lines of text, as illustrated below.



```
: paragraph {  
    : text_part { label = "A small step"; }  
    : text_part { label = "for a man."; }  
}
```

Errtile

The **errtile** tile defines a horizontal space for reporting error messages. It appears at the bottom of dialog boxes, and its key is "error."

```
errtile;
```

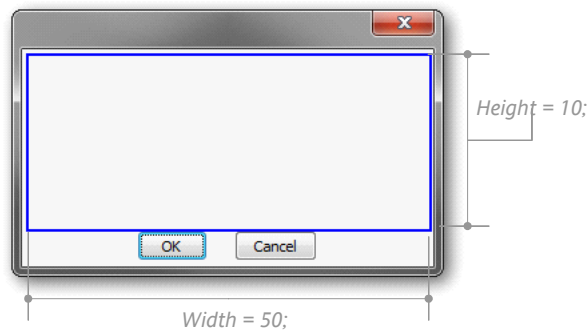
You use it in conjunction with the **set_tile** function in the accompanying LISP code.

```
(set_tile "error" "Error: Cannot set that value.")
```

Error: Cannot set that value.

SPACER

The **spacer** tile defines a vertical and/or horizontal space. The figure shows a 10x50 spacer outlined in blue, below. The spacer is measured in characters.



```
: spacer {  
  height = number;  
  width = number;  
  fixed_height = false | true;  
  fixed_width = false | true;  
  alignment = left | right | centered;  
}
```

SPACER_0

The **spacer_0** tile defines a variable-width space that spaces itself automatically.

```
spacer_0;
```

SPACER_1

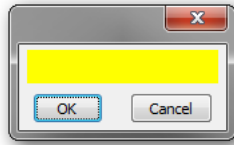
The **spacer_1** tile defines a very narrow space.

```
spacer_1;
```

IMAGE

The **image** tile defines a rectangular area for displaying an image — it's kind of like a colored spacer. It can display things like text font samples, hatch pattern samples, color samples, or icons that represent drawing and editing commands.

The easiest form is the color sample, as illustrated by the yellow rectangle below, because it is merely specified by the **color** attribute.



Images of hatch patterns, fonts, and so on can be placed on the image tile through the accompanying LISP code's callback function (**set_tile**) and the **key** attribute.

There are two sources of image you can use:

- SLD slide files, which are created ahead of time with the command that makes slides in BricsCAD, and then placed with LISP's **slide_image** function.
- Vector lines, which are drawn on-the-fly by LISP's **vector_image** function.

```
: image {  
  action = "(LISP function)";  
  key = "text";  
  value = "text";  
  mnemonic = "char";  
  color = colornumber;  
  aspect_ratio = number;  
  height = number;  
  width = number;  
  is_enabled= true | false;  
  is_tab_stop= true | false;  
  fixed_height = false | true;  
  fixed_width = false | true;  
  alignment = left | right | centered;  
}
```

KEY

The **key** attribute identifies the **image** tile to the accompanying LISP code, so that the slide image can be placed in the dialog box.

```
key = "image1";
```

Value and Mnemonic

The **value** and **mnemonic** attributes appear to have no effect.

COLOR

The **color** attribute defines the color of the image tile. When you leave out this attribute, the color is black (default).

Use the same color numbers as for the **image_button** tile. A popular number is **-15**, which displays the same color as that of the dialog box's background — usually gray.

```
color = -15;
```

Number	Color Name	Meaning
...	...	Default color is black
0	black	ACI color 0 (black or white) ¹
1	red	ACI ² color 1
2	yellow	ACI color 2
3	green	ACI color 3
4	cyan	ACI color 4
5	blue	ACI color 5
6	magenta	ACI color 6
7	white	ACI color 7 (white or black) ¹
-1	graphics_foreground	Current default color of entities (usually ACI 7). ¹
-2	graphics_background	Current background color of BricsCAD's graphics screen.
-3	blue	
-4	black	
-5	gray	
-6	black	
-7	red	
-15	dialog_background	Current color of dialog box background (usually gray).
-16	dialog_foreground	Current color of dialog box text (usually black).
-18	dialog_line	Current color of dialog box lines (usually black).

Notes:

¹ The color is white when the background color is dark, but black when the background is light.

² ACI is short for "BricsCAD Color Index," and refers to the 256 color numbers.

ASPECT_RATIO, HEIGHT, & WIDTH

You use any two of these three attributes. The **aspect_ratio** attribute specifies the ratio between the height and width of the image tile, and must be used with either the **height** or the **width** attribute — but not both. Similarly, if you use the **height** and **width** attributes, you cannot use the **aspect_ratio** attribute. Examples:

```
aspect_ratio = 1.333;  
height = 3;
```

Or...

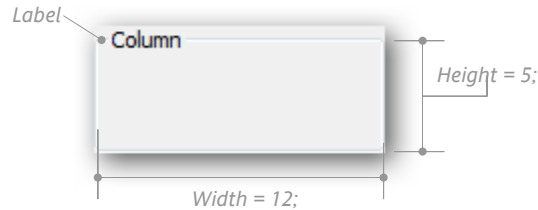
```
aspect_ratio = 1.333;  
width = 4;
```

Or...

```
height = 3;  
width = 4;
```

COLUMN

The **column** tile defines a vertical column of tiles. This tile is not normally needed, because tiles are stacked vertically by default. You would use it when you want two columns of tiles in the dialog box.



```
: column {  
    label = "text";  
    height = number;  
    width = number;  
    fixed_height = false | true;  
    fixed_width = false | true;  
    children_fixed_height = false | true;  
    children_fixed_width = false | true;  
    alignment = left | right | centered;  
    children_alignment = left | right | centered;  
}
```

LABEL

The **label** attribute provides a title for the column. Curiously, when the label is not used, then the column is unboxed; when a label is used, the column is boxed automatically — jsut as if it were the **boxed_column** tile.

```
    label = "Column";
```

HEIGHT & WIDTH

BricsCAD normally sizes the column automatically. You can use the **height** and **width** attributes to specify a larger size; height is measured in lines of text, width in characters.

```
    height = 10;  
    width = 40;
```

CHILDREN_FIXED_HEIGHT, CHILDREN_FIXED_WIDTH, & CHILDREN_ALIGNMENT

The **children_fixed_height** and **children_fixed_width** attributes fix the height and width of clustered tiles; these attributes can be overridden by the children's attributes.

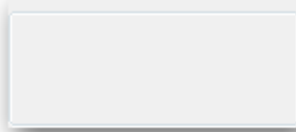
```
    children_fixed_height = true;  
    children_fixed_width = true;
```

The **children_alignment** attribute sets the alignment of clustered tiles; this attribute can be overridden by the children's alignment attributes.

```
    children_alignment = centered;
```


BOXED_COLUMN

The **boxed_column** tile places a box around a column of tiles. It is identical to the **column** tile, except that the box appears whether or not the tile has a label. The figure below illustrates the box without a label.



```
: boxed_column {
  label = "text";
  height = number;
  width = number;
  children_fixed_width = false | true;
  fixed_height = false | true;
  fixed_width = false | true;
  children_fixed_height = false | true;
  alignment = left | right | centered;
  children_alignment = left | right | centered;
}
```

RADIO-COLUMN & BOXED_RADIO_COLUMN

The **radio_column** and **boxed_radio_column** tiles define vertical columns for radio buttons. The only difference from the **boxed_column** and **column** tiles is the addition of the **value** attribute, which specifies which radio button is turned on.

```
: radio_column {                               // or : boxed_radio_column {
  label = "text";
  value = "number";
  height = number;
  width = number;
  fixed_height = false | true;
  fixed_width = false | true;
  children_fixed_height = false | true;
  children_fixed_width = false | true;
  alignment = left | right | centered;
  children_alignment = left | right | centered;
}
```

VALUE

The **value** attribute specifies which radio button is turned on, the first button being #0.

```
value = "2";
```

LISP Functions for Dialog Boxes

Dialog boxes are designed by DCL files and displayed by LISP routines. The most basic LISP routine to load, display and unload dialog boxes looks like this:

```
(defun c:functionName (/ dlg-id)
  (setq dlg-id (load_dialog "fileName"))
  (new_dialog "dialogName" dlg-id)
  ; Insert get_tile, set_tile, action_tile,
  ; and other functions here.
  (start_dialog)
  (unload_dialog dlg-id)
)
```

A *fileName.dcl* file specifies the layout of the dialog box. The most basic file looks like this:

```
dialogName : dialog {
  // Insert tiles here.
  ok_only;
}
```

This section of the chapter describes the LISP functions that interact with dialog boxes in the following order:

```
load_dialog
  new_dialog
  start_dialog
  done_dialog
  term_dialog

get_tile
  set_tile
  get_attr
  mode_tile
  action_tile
  client_data_tile

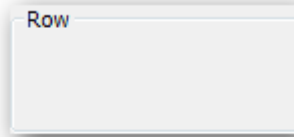
start_list
  add_list
  end_list

start_image
  slide_image
  fill_image
  vector_image
  dimx_tile
  dimy_tile
  end_image

alert
  help
  acad_helpdlg
  acad_colordlg
  acad_truecolordlg
  initdia
```

ROW & BOXED_ROW

The **row** and **boxed_row** tiles define a horizontal row of other tiles, called "children" or "clustered tiles." Like columns, including a label to the row tile adds the box; no label, no box. Otherwise, the two tiles are identical.



```
: row {                                     // or : boxed_row {
    label = "text";
    height = number;
    width = number;
    fixed_height = false | true;
    fixed_width = false | true;
    children_fixed_height = false | true;
    children_fixed_width = false | true;
    alignment = left | right | centered;
    children_alignment = centered | top | bottom;
}
```

OTHER ATTRIBUTES

Attributes are identical to those of the column tile, except that the **children_alignment** attribute is vertically oriented: top, bottom, or centered.

RADIO_ROW & BOXED_RADIO_ROW

The **radio_row** tile defines a horizontal row of radio buttons.

```
: radio_row {                               // or : boxed_radio_row {
    label = "text";
    value = "number";

    height = number;
    width = number;
    fixed_height = false | true;
    fixed_width = false | true;
    children_fixed_height = false | true;
    children_fixed_width = false | true;

    alignment = left | right | centered;
    children_alignment = centered | top | bottom;
}
```

VALUE

The **value** attribute specifies which radio button is turned on, the first button being #0.

```
value = "2";
```

LOAD_DIALOG

The **load_dialog** function loads *.dcl* files that define dialog boxes, and returns a *fileid* (the identifying number assigned by the operating system to open files) handle.

```
(load_dialog "dclFile")
```

dclFile — name of the *.dcl* file. It is in quotation marks. Remember to use double-slash path separators, as shown below. The ".dcl" extension is not required.

```
(load_dialog "c:\\filename"))
```

This function is usually used with **setq** to store the value of the handle, as follows:

```
(setq dclId (load_dialog "c:\\filename"))
```

This function returns a fileid handle such as 30, when successful, or **-1** if not.

NEW_DIALOG

The **new_dialog** function activates a named dialog box. This function is needed because *.dcl* files can contain more than one dialog box definition. Thus, **load_dialog** is used to load the *.dcl* file, and then **new_dialog** is used to access the specific dialog box.

```
(new_dialog dlgName dclId action screenPt)
```

dlgName — string identifying the dialog box in the *.dcl* file.

dclId — DCL fileid handle retrieved earlier by the **load_dialog** function.

action — [optional] string containing the LISP expression that executes as default action when users picks tiles that don't have a DCL action or LSP callback assigned by the **action_tile** function.

screenPt — [optional] 2D point list specifying the x,y-location of the dialog box's upper left corner of the BricsCAD window. Use **'(-1 -1)** to open the dialog box in the center of the BricsCAD window. To use this argument without the action argument, enter **""**, as follows:

```
(new_dialog dlgName dcl_id "" '(10,10))
```

This function returns **T** when successful, or **nil** if not.

START_DIALOG

The **start_dialog** function displays the dialog box. Before this function is executed, you should set up callbacks and other functions. This function has no arguments.

```
(start_dialog)
```

This function returns **1** when users exit the dialog box by clicking OK, or **0** if they click the Cancel button. A **-1** is returned when the dialog box is closed by the **term_dialog** function.

DONE_DIALOG

The **done_dialog** function closes the dialog box.

(**done_dialog** status)

status — positive integer returned by **start_dialog**, the meaning of which the application determines. For this to work, **done_dialog** must be called from a callback function such as **action_tile**.

This function returns a 2D point list in the form of '(x,y). It identifies the position of the upper-left corner of the dialog box at the time the user exited it. This allows you to reopen the dialog box in the same location.

TERM_DIALOG

The **term_dialog** function terminates dialog boxes. It is called by BricsCAD when applications (LISP routines) terminate while *.dcl* files are still open. This function has no arguments.

(**term_dialog**)

This function always returns **nil**.

UNLOAD_DIALOG

The **unload_dialog** function unloads *.dcl* files from memory.

(**unload_dialog** dclId)

dclId — specifies the file-id handle first acquired by the **load_id** function.

This function always returns **nil**.

GET_TILE

The **get_tile** function retrieves the values of tiles.

(**get_tile** key)

key — identifies the tile to be accessed.

This function returns a string containing the value of the tile's **value** attribute.

SET_TILE

The **set_tile** function sets the value of dialog box tiles.

(**set_tile** key value)

key — identifies the tile to be processed.

value — specifies a string that contains the new value to be assigned to the tile's **value** attribute.

This function returns the new value of the tile.

GET_ATTR

The **get_attr** function retrieves the DCL value of the tile's attribute.

(**get_attr** key attribute)

key — identifies the tile to be processed.

attribute — identifies the attribute whose value is to be retrieved.

This function returns a string with the attribute's as found in the DCL file.

MODE_TILE

The **mode_tile** function sets the mode of dialog box tiles. This allows you to change, for example, buttons from active (normal) to inactive (grayed out).

(**mode_tile** key mode)

key — identifies the tile to be processed.

mode — specifies the action to be applied to the tile:

Mode	Meaning
-------------	----------------

- | | |
|---|---------------------------------------|
| 0 | Enables the tile. |
| 1 | Disables the tile (grays it out). |
| 2 | Sets focus to the tile. |
| 3 | Selects the contents of the edit box. |
| 4 | Toggles image highlighting. |

This function returns **nil**.

ACTION_TILE

The **action_tile** function assigns action to be evaluated when users select the dialog box's tile.

(**action_tile** key action)

TIP The action assigned by this function overrules the action defined by the tile's **action** attribute, as well as the action specified by the **new_dialog** function.

key — identifies the tile to be processed.

action — a string that specifies the action, usually an LISP function. (LISP's **command** function cannot be used, unfortunately.) You can use the following metacharacters:

Metacharacter	Meaning
\$value	Current value of the tile.
\$key	Name of the tile.
\$data	Application-specific data set by client_data_tile .
\$reason	Callback reason.
\$x and \$y	Image's x,y coordinates.

This function returns **T**.

CLIENT_DATA_TILE

The **client_data_tile** function associates data from a function with a tile in the dialog box.

(**client_data_tile** key data)

key — identifies the tile to be processed.

data — specifies the string containing the data.

TIP Functions can refer to this data as \$data.

This function returns **nil**.

START_LIST

The **start_list** function starts processing list boxes and popup boxes.

(**start_list** key operation index)

key — identifies the list box or popup box being processed.

operation — [*optional*] specifies the operation being performed; default is to delete the existing list, and replace it with a new one specified by the **add_list** function. The operations are:

Operation	Meaning
1	Change selected list contents
2	Append new list entry
3	Delete old list and create new list (the default)

index — [*optional*] specifies which list item to modify; default is #0, the first item.

This function returns the name of the list.

TIPS In all cases, you use the list-related functions in this order:

```
(start_list)
(add_list)
(end_list)
```

You are warned against using the `set_tile` function between `start_list` and `end_list`, because that would change the nature of the list.

All actions by the `add_list` function apply only to the list specified by `start_list`; to switch to a different list, use `end_list` and then `start_list`.

ADD_LIST

The `add_list` function adds or modifies strings in list and popup boxes, depending on the operation specified by `start_list`.

```
(add_list strings)
```

strings — specifies the list of items to add or replace in the list. The string uses quotation marks to separate items in the list, as follows:

```
(add_list "firstItem" "secondItem" "thirdItem")
```

This function returns the string, if successful; otherwise `nil`, if not.

END_LIST

The `end_list` function ends processing of list and popup boxes.

```
(end_list)
```

This function returns `nil`.

START_IMAGE

The `start_image` function starts creating vector or slide images in dialog boxes.

```
(start_image key)
```

key — specifies the key name of the image tile.

This function returns the value of *key*; otherwise `nil`, if unsuccessful.

TIP Typically, you use the image-related functions in this order:

```
(start_image)
(fill_image)
(slide_image) or (vector_image)
(end_image)
```

SLIDE_IMAGE

The **slide_image** function displays slides in dialog box image tiles.

`(slide_image x y width height sldName)`

x — specifies the number of pixels to offset the image from the upper-left corner of the tile, in the x direction.

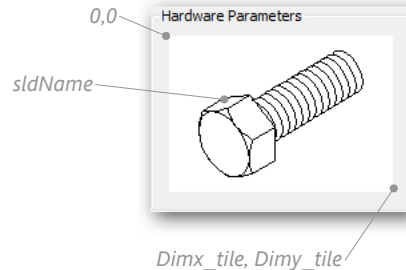
y — specifies the number of pixels to offset the image from the upper-left corner of the tile, in the y direction.

width — specifies the width of the image in pixels.

height — specifies the height of the image in pixels.

sldName — specifies the name of the slide image to display, which can be in an SLD slide file or SLB slide library file. When in a library, use this format:

`(slide_image 0 0 40 30 sldlibName(sldName))`



This function returns the name of the *sldName* as a string.

TIPS X and Y are always positive.

The coordinates of the upper left corner are 0,0.

You can get the coordinates of the lower-right corner through **dimx_tile** and **dimy_tile**, like this:

`(slide_image 0 0 (dimx_tile "slide_tile") (dimy_tile "slide_tile") "sldName")`

FILL_IMAGE

The **fill_image** function draws filled rectangles in dialog boxes.

`(fill_image x y width height color)`

x — specifies the number of pixels to offset the image from the upper-left corner of the tile, in the x direction.

y — specifies the number of pixels to offset the image from the upper-left corner of the tile, in the y direction.

width — specifies the width of the image in pixels.

height — specifies the height of the image in pixels.

color — specifies the color using ACI, or one of the following special numbers:

Number	Meaning
-2	Current background color of BricsCAD's drawing area.
-15	Current background color of the dialog box.
-16	Current text color of the dialog box.
-18	Current color of dialog box lines.

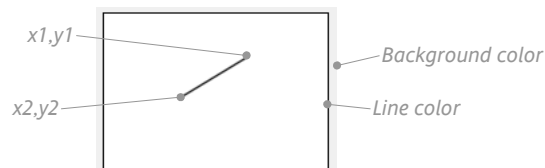
This function returns an integer representing the ACI fill color.

TIP This function must be used between the **start_image** and **end_image** functions.

VECTOR_IMAGE

The **vector_image** function draws vectors in dialog box image tiles.

(**vector_image** x1 y1 x2 y2 color)



x1 — specifies the x coordinate of the starting point.

y1 — specifies the y coordinate of the starting point.

x2 — specifies the x coordinate of the starting point.

y2 — specifies the y coordinate of the starting point.

color — specifies the color using ACI, or one of the special numbers listed above.

TIP One vector (line) is drawn with each call of this function. The line is drawn from $x1, y1$ to $x2, y2$.

DIMX_TILE & DIMY_TILE

The **dimx_tile** function returns the x-dimension of the image tile's lower right corner; the **dimy_tile** function does the same for the y-dimension.

(**dimx_tile** key)

(**dimy_tile** key)

key — specifies the key name of the image tile.



These functions return the "x-1" width and "y-1" height of the tile.

TIPS *Caution:* These functions return x,y coordinates are **one less** than the total x and y dimensions of the tile, because the upper-right corner is 0,0.

These functions are meant for use with the **slide_image**, **fill_image**, and **vector_image** functions.

END_IMAGE

.....

The **end_image** function signals the end of the image tile's definition.

(end_image)

This function returns **nil**.

DIALOG BOXES DISPLAYED BY LISP FUNCTIONS

The following LISP functions display BricsCAD dialog boxes.

Alert

The **alert** function displays the alert dialog box with customized warning. You can use the `\n` metacharacter to include line breaks.

```
(alert "Help me!\nI've fallen and I can't get up!")
```

Help)

The **help** function displays the Help window.

Acad_HelpDlg

The **acad_helpdlg** function displays the old-style Help dialog box with *.hlp* files.

```
(acad_helpdlg "acadctxt.hlp" "topic")
```

AcadColorDlg

The **acad_colordlg** function displays the Select Color dialog box with just the Index Color tab.

```
(acad_colordlg colorNum flag)
```

colorNum — specifies the default color number; ranges from 0 to 256. This integer is a required argument, even when you don't want to specify a default.

0 = ByBlock color.

256 = ByLayer color.

flag — [optional] disables the **ByLayer** and **ByBlock** buttons when set to **nil**.

For example, to open the Select Color dialog box, set red (1) as the default color, and gray out the By-buttons, use this form of the function:

```
(acad_colordlg 1 nil)
```

This function returns the number of the color selected by the user, or **nil** when the user clicks Cancel.

Acad_TrueColorDlg

The **acad_truecolordlg** function displays the Select Color dialog box with all tabs.

```
(acad_truecolordlg color flag byColor)
```

color — specifies the the default color as a dotted pair, where the first value is the DXF code for the type of color specification:

62 = ACI (index color).

420 = TrueColor spec in RGB (red-green-blue) format.

430 = color book name (not supported by BricsCAD).

Use the following formats:

Color Format	Dotted Pair Format	Example for Red
ACI	(62 . ColorIndex)	(62 . 1)
TrueColor	(420 . "red,green,blue")	(420 . "255,0,0")
Color Book	(430 . "colorbook\$colorname")	(430 . "RAL CLASSIC\$RAL 3026")

flag — [optional] disables the **ByLayer** and **ByBlock** buttons when set to **nil**.

byColor — [optional] sets the value of **ByLayer** and **ByBlock** color; use the same format as for *color*.

This function returns the color selected by the user in dotted-pair format. The list may contain more than one dotted-pair; the last one is the one selected by the user. For example, if the user selects from a color book, then the list contains the 430 pair (specifying the color book), as well as a 420 pair containing the TrueColor value and a 62 pair describing the closest ACI value. Color books are not supported by Bricscad.

Nil is returned when the user clicks **Cancel**.

InitDia

The **initdia** function forces the display of the dialog box of the following command, such as:

```
(initdia flag)
```

```
(command "image")
```

flag — [optional] when 0, resets command to display prompts at the command line.

This function is meant for commands that normally display their prompts at the command line during LISP routines. These include **AttDef**, **AttExt**, **Hatch** (**BHatch** in older versions of BricsCAD), **Block**, **Color**, **Image** (**ClassicImage** in BricsCAD 2007), **ImageAdjust**, **Insert**, **Layer**, **Linetype**, **MText**, **Plot**, **Rename**, **Style**, **Toolbar**, and **View**.

This function always returns *nil*.

Concise LISP Reference

This appendix offers an alphabetical list of functions for the LISP programming language:

Blue text indicates functions unique to BricsCAD (not found in AutoLISP).

Green text indicates functions specific to controlling DCL dialog boxes.

Italicized text indicates parameter(s).

[*square brackets*] indicate optional parameters.

ellipsis ... indicate that additional parameters are permitted

Pi is defined in LISP as a constant, 3.141...

LISP Function Summary

A

- (abs) Returns absolute value of number
(abs *number*)
- (acad_colorDlg) Displays the Select Color dialog box
(acad_colorDlg *color-number [flag]*)
- (acad_strlSort) Sorts list of strings in alphabetical order
(acad_strlSort *list*)
- (action_tile) Responds when the user clicks a dialog box tile
(action_tile *key expression*)
- (acos) Returns the arc cosine of x.
(acos *x*)
- (add_list) Adds text to an existing dialog box string
(add_list *string*)
- (ads) Reports which applications are loaded into BricsCAD
(ads)
- (alert) Displays a message box
(alert *string*)
- (alloc) Sets the memory segment size for LISP
(alloc *integer*)
- (and) Returns logical AND of the supplied arguments
(and *expression*)
- (angle) Returns the angle (in radians) of the line defined by two points
(angle *point1 point2*)
- (angtof) Converts string representation of angle to radians
(angtof *string [mode]*)
- (angtos) Converts angle (radians) to string representation
(angtos *angle [mode [precision]]*)
- (append) Appends list arguments to one list
(append *list1 list2*)
- (apply) Applies the specified function for each argument supplied in the list
(apply *function list*)
- (arx) Returns a list of the currently loaded ObjectARX applications
(arx)
- (arxload) Loads an ObjectARX application
(arxload *application [onfailure]*)
- (arxunload) Unloads an ObjectARX application
(arxunload *application [onfailure]*)

(ascii) Converts the first character of string to its ASCII char code (integer)
(*ascii string*)

(asin) Returns the arc sine of *x*.
(*asin x*)

(assoc) Finds the first matching item in the list
(*assoc item list*)

(atan) Returns arctangent
(*atan number1 [number2]*)

(atanh) Returns the hyperbolic arc tangent of *x*.
(*atanh x*)

(atof) Converts a string to a real number
(*atof string*)

(atoi) Converts a string to an integer
(*atoi string*)

(atom) Confirms that an item is an atom
(*atom item*)

(atoms-family) Returns a list of the currently defined symbols
(*atoms-family format [symbol1 symbol2]*)

(autoload) Loads the LISP application automatically when one of its commands is used
(*autoload application list*)

B

(boole) Applies the bitwise Boolean function
(*boole function integer1 integer2*)

(boundp) Confirms that this item has a value bound to it
(*boundp item*)

C

(caddr) Returns the third item of the list
(*caddr list*)

(cadr) Returns the second item of the list
(*cadr list*)

(car) Returns the first item of the list
(*car list*)

(cdr) Returns everything in the list except the first item
(*cdr list*)

(ceiling) Returns the smallest integer that is not smaller than *x*.
(*ceiling x*)

(chr) Converts the integer (ASCII char code) to a single-character string
(*chr integer*)

- (client_data_tile)** Associates data with a dialog box tile
(client_data_table *key data*)
- (close)** Close an open file
(close *file-descriptor*)
- (command)** Launches the BricsCAD command
(command *cmd [arguments]*)
- (cond)** Compares conditional statements
(cond (*statement1 result1*))
- (cons)** Adds this item to the beginning of the list
(cons *item list*)
- (cos)** Calculates the cosine
(cos *angle*)
- (cosh)** Returns the hyperbolic cosine of x.
(cosh *x*)
- (cvunit)** Converts a value from one unit of measurement to another
(cvunit *value from to*)

D

- (defun)** Defines a LISP function
(defun [*c:*] *name* ([*arg1 arg2*] / [*local-var1 local-var2*]) *expression*)
- (dictadd)** Adds a nongraphical object to a dictionary
(dictadd *ename symbol newobj*)
- (dictnext)** Finds the next item in a dictionary
(dictnext *ename [rewind]*)
- (dictremove)** Removes an item from a dictionary
(dictremove *ename symbol*)
- (dictrename)** Renames a dictionary entry
(dictrename *ename oldsym newsym*)
- (dictsearch)** Searches a dictionary for an item
(dictsearch *ename symbol [setnext]*)
- (distance)** Determines the distance between two points
(distance *point1 point2*)
- (distof)** Converts a string to a real number
(distof *string [mode]*)
- (done_dialog)** Terminates the dialog box
(done_dialog [*flag*])

E

- (end_image) Ends the creation of a dialog box image
(end_image)
- (end_list) Ends the processing of a dialog box list
(end_list)
- (entdel) Deletes the entity
(entdel *entity-name*)
- (entget) Retrieves the entity's definition data
(entget *entity-name* [*application-list*])
- (entlast) Gets the last entity in the drawing
(entlast)
- (entmake) Adds an entity to the drawing
(entmake [*entity-list*])
- (entmakex) Makes a new entity, give it a handle and return it's new entity name
(entmakex [*entity-list*])
- (entmod) Modifies the entity
(entmod *entity-list*)
- (entnext) Returns the next entity in the drawing
(entnext [*entity-name*])
- (entsel) Prompts the user to select an entity
(entsel [*prompt*])
- (entupd) Redraws the entity
(entupd *entity-name*)
- (eq) Determines whether two expressions are bound to the same symbol
(eq *statement1 statement2*)
- (equal) Determines whether two statements are the same within an optional tolerance value
(equal *statement1 statement2* [*tolerance*])
- (*error*) Displays an error message
(*error* *string*)
- (eval) Evaluates the LISP expression
(eval *statement*)
- (exit) Terminates
(exit)
- (exp) Calculates the natural exponent
(exp *number*)
- (expand) Allocates additional memory for LISP
(expand *integer*)
- (expt) Raises the number to the specified power
(expt *base power*)

F

- (fill_image) Fills a dialog box's rectangle with color
(fill_image *x y width height color*)
- (find) Returns item if item is found in list, otherwise it returns nil.
(find *item List*)
- (findfile) Searches for the specified file or directory
(findfile *filename*)
- (fix) Converts a real number to the nearest integer
(fix *number*)
- (float) Converts an integer to a real
(float *number*)
- (floor) Returns the greatest integer less than or equal to x.
(floor *x*)
- (foreach) Evaluates the expression to every item in the list
(foreach *variable List expression*)

G

- (gc) Performs garbage collection
(gc)
- (gcd) Calculates the greatest common denominator
(gcd *integer1 integer2*)
- (get_attr) Determines the attribute of a dialog box's key
(get_attr *key attribute*)
- (get_diskserialid) Returns a 9-digit unique id string, based on the first hard disk serial number. If the hard disk serial number can not be obtained in very rare cases, the 9-digit unique id string is based on the serial number of the first partition. This id string provides a licensing/hardlocking feature for LISP applications.
(get_diskserialid)
- (get_tile) Determines the value of a dialog box's tile
(get_tile *key*)
- (getangle) Prompts the user to specify an angle
(getangle [*point*] [*prompt*])
- (getcfig) Determines the value of the parameter
(getcfig *parameter*)
- (getcname) Determines the localized command name
(getcname [*_*] *command-name*)
- (getcorner) Prompts the user to specify the second corner of a rectangle
(getcorner *point* [*prompt*])
- (getdist) Prompts the user to specify two points
(getdist [*point*] [*prompt*])
- (getenv) Determines the value of the operating system variable
(getenv *variable*)

(getfiled) Displays the Open File dialog
(getfiled *title filename ext flags*)

(getint) Prompts the user to enter an integer
(getint [*prompt*])

(getkeyword) Prompts the user to select a keyword
(getkeyword [*prompt*])

(getorient) Prompts the user to specify an angle
(getorient [*pt*] [*prompt*])

(getpid) Returns the process ID of the current process.
(getpid)

(getpoint) Prompts the user to select a point
(getpoint [*point*] [*prompt*])

(getreal) Prompts the user to select a real number
(getreal [*prompt*])

(getstring) Prompts the user to enter a string
(getstring [*flag*] [*prompt*])

(getvar) Returns the value of a system variable
(getvar *sysvar*)

(graphscr) Switches to the graphics window
(graphscr)

(grarc) Draws a temporary arc or circle, with specified radius and color; optionally highlighted
(grarc *ptCenter radius startAng endAng color [minsegments] [highlight]*)

(grclear) Clears the viewport
(grclear)

(grdraw) Draws a line
(grdraw *point1 point2 color [highlight]*)

(grfill) Draws temporary filled polygon area, with specified color; optionally in highlighted mode
(grfill *ptlist color [highlight]*)

(grread) Reads the data coming in from the input devices
(grread [*flag*] [*bits*] [*cursor*])

(grtext) Writes text on the status line
(grtext [*flag text*])

(grvecs) Draws one or more lines
(grvecs *vector-lists [trans]*)

H

(handent) Returns the entity name based on its handle
(handent *handle*)

(help) Launches help
(help [*filename*] [*topic*] [*flag*])

I

- (if) Evaluates expressions conditionally
(if *test statement1 [statement2]*)
- (initdia) Forces the dialog box version of a command
(initdia *[flag]*)
- (initget) Initializes the keywords for next user-input
(initget *[bits] [string]*)
- (inters) Finds the intersection
(inters *point1 point2 point3 point4 [flag]*)
- (itoa) Converts integer to string
(itoa *integer*)

L

- (lambda) Defines an unnamed LISP function
(lambda *arguments expression*)
- (last) Returns the last item in the list
(last *List*)
- (length) Returns the number of elements contained in a list
(length *List*)
- (list) Create a list
(list *expression ...*)
- (listp) Confirms that an item is a list
(listp *item*)
- (load) Loads the LISP file
(load *filename [flag]*)
- (load_dialog) Loads the DCL file
(load_dialog *filename*)
- (log) Calculates the natural logarithm
(log *number*)
- (log10) Returns the base-10 logarithm of x.
(log10 *x*)
- (logand) Determines what is the logical AND
(logand *integer1 integer2 ...*)
- (logior) Determines what is the logical OR
(logior *integer1 integer2 ...*)
- (lsh) Does a bitwise shift
(lsh *integer1 integer2*)

M

- (mapcar) Applies the function to the list
(mapcar *function List1 [List2]*)
- (max) Returns the largest number
(max *number1 number2 ...*)
- (mem) Displays the status of the LISP memory
(mem)
- (member) Identifies the first occurrence of an item in the list
(member *item List*)
- (menucmd) Executes the menu command
(menucmd *string*)
- (menugroup) Determines whether a menu group is loaded
(menugroup *name*)
- (min) Returns the smallest number
(min *number1 number2 ...*)
- (minusp) Determines whether a value is a negative number
(minusp *number*)
- (mode_tile) Sets the mode of the dialog box tile
(mode_tile *key mode*)

N

- (namedobjdict) Returns the current drawing's named object dictionary (Root)
(namedobjdict)
- (nentsel) Prompts the user to select an entity within a complex entity
(nentsel [*prompt*])
- (nentselp) Operates like nentsel but without user input
(nentselp [*prompt*] [*point*])
- (new_dialog) Displays a dialog box
(new_dialog *dialog dcl-id [function point]*)
- (not) Determines whether an item is nil
(not *item*)
- (nth) Determines the nth item in the list
(nth *integer List*)
- (null) Determines whether the item is bound to nil
(null *item*)
- (numberp) Determines whether an item is a number
(numberp *item*)

O

(open) Opens file for access by LISP read-write-append functions
(open *filename mode*)

(or) Calculates the logical OR
(or *statement*)

(osnap) Returns 3D point as result of applying the specified entity snap
(osnap *point mode*)

P

(polar) Returns 3D point defined by angle and distance of specified point
(polar *point angle distance*)

(position) Returns the index of item in list or nil (first index is 0).
(position *item list*)

(prinl) Prints string
(prinl [*expression [file-descriptor]*])

(princ) Prints string taking into account control characters
(princ [*expression [file-descriptor]*])

(print) Prints string using formatted printing
(print [*expression [file-descriptor]*])

(progn) Evaluates each expression sequentially and returns the value of the last expression
(progn *statement1 statement2*)

(prompt) Prints message on the command line
(prompt *string*)

Q

(quit) Quits the current LISP routine
(quit)

(quote) Returns an expression without evaluating it
(quote *statement*)

R

(read) Determines the first item in a string
(read *string*)

(read-char) Reads a single character
(read-char [*file-descriptor*])

(read-line) Reads an entire line
(read-line [*file-descriptor*])

(redraw) Redraws the viewport or just a single entity
(redraw [*ename [mode]*])

(regapp) Registers the application
(regapp *appname*)

(rem) Determines the remainder of this division operation
(rem *number1 number2 [number3]*)

(remove) Returns the input list, with item removed from the list.
(remove *item List*)

(repeat) Evaluates each expression a specified number of times
(repeat *number statement1 [statement2]*)

(reverse) Returns a copy of a list with its elements reversed
(reverse *List*)

(round) Returns the integer nearest to x.
(round *x*)

(rtos) Converts real to string
(rtos *number [mode [precision]]*)

S

(search) Searches for list1 in list2 and returns the index where found or nil (first index is 0).
(search *list1 list2*)

(set) Assigns the statement to the symbol
(set *symbol statement*)

(set_tile) Sets the value of the dialog box tile
(set_tile *key value*)

(setcfg) Sets the parameter to the value
(setcfg *parameter value*)

(setenv) Sets the operating system variable to that value
(setenv *variable value*)

(setfunhelp) Registers the command with that Help file
(setfunhelp "*c:filename*" [*helpfile [topic [command-name]]*])

(setq) Sets the symbol to the statement
(setq *symbol1 statement1 [symbol2 statement2]*)

(setvar) Sets the system variable to that value
(setvar *sysvar value*)

(setview) Creates a 3D viewpoint
(setview *view-descriptor [vport]*)

(sin) Calculates the sine
(sin *angle*)

(sinh) Returns the hyperbolic sine of x.
(sinh *x*)

(slide_image) Displays a slide in the dialog box
(slide_image *x y width height slide*)

(sleep) Delays execution for (approx.) given milliseconds

(sleep *milliseconds*)

(svalid) Determines whether the symbol is made-up of valid characters

(svalid *symbol [flag]*)

(sqrt) Calculates the square root

(sqrt *number*)

(ssadd) Adds the entity to the selection set

(ssadd [*entity-name [selection-set]*])

(ssdel) Deletes the entity from the selection set

(ssdel *entity-name selection-set*)

(ssget) Creates a selection set

(ssget [*mode [point1 [point2]] [point-list] [filter-list]*])

(ssgetfirst) Determines which entities are highlighted and/or gripped

(ssgetfirst)

(sslenth) Determines how many entities are in the selection set

(sslenth *selection-set*)

(ssmemb) Determines whether an entity is in the selection set

(ssmemb *entity-name selection-set*)

(ssname) Identifies the nth entity in the selection set

(ssname *selection-set index*)

(ssnamex) Retrieves information about how a selection set was created

(ssnamex *selection-set index*)

(sssetfirst) Determines which objects are selected and gripped

(sssetfirst *grip-set [pick-set]*)

(startapp) Launches Windows application

(startapp *appname [filename]*)

(start_dialog) Starts the dialog box

(start_dialog)

(start_image) Starts creating a dialog box image

(start_image *key*)

(start_list) Starts processing a list box

(start_list *key [operation [index]]*)

(strcase) Converts string to all upper- or all lower-case

(strcase *string [flag]*)

(strcat) Concatenates strings

(strcat *string1 [string2] ...*)

(string-split) Returns a list of strings divided by any character in string-of-delimiters. Example: (string-split bbb;ccc,ddd) => ("aaa" "bbb" "ccc" "ddd")

(string-split *string-of-delimiters string*)

(strlen) Returns the number of characters in a string

(strlen [*string1 [string2] ...*])

(subst) Returns a copy of a list with its elements substituted
(subst *new old List*)

(substr) Returns a substring of a string
(substr *string start [length]*)

T

(tablet) Retrieves and sets digitizer (tablet) calibrations
(tablet *code [row1 row2 row3 direction]*)

(tan) Returns the tangent of x - x must be in radians.
(tan x)

(tanh) Returns the hyperbolic tangent of x .
(tanh x)

(tblnext) Finds the next item in a symbol table
(tblnext *table-name [flag]*)

(tblobjname) Returns the entity name of a specified symbol table entry
(tblobjname *table-name symbol*)

(tblsearch) Searches the table for a symbol
(tblsearch *table-name symbol [flag]*)

(term_dialog) Terminates the dialog box
(term_dialog)

(terpri) Prints a carriage return
(terpri)

(textbox) Returns the bounding box of a text entity
(textbox *entity-list*)

(textpage) Switches focus from the drawing area to the text screen
(textpage)

(textscr) Switches focus from the drawing area to the text screen
(textscr)

(trace) Turns on debug mode
(trace *function*)

(trans) Translates that point from one coordinate system to another
(trans *point from to [flag]*)

(trim) Removes leading and trailing blanks
(trim *string [flag]*)

(type) Returns the type of a specified item
(type *item*)

U

(unload_dialog) Unloads that dialog box
(unload_dialog *dcl_id*)

(until) Repeats the expression(s) until test-expression evaluates as T (true).
(until *test-expression [expression ...]*)

(untrace) Turns off debug mode
(untrace *function*)

V

(vector_image) Draws a vector in the dialog box
(vector_image *x1 y1 x2 y2 color*)

(ver) Determines the version number of this LISP
(ver)

(vmon) Turns on virtual memory
(vmon)

(vports) Gets information about this viewport
(vports)

W

(wcmatch) Performs a wild-card pattern match on a string
(wcmatch *string pattern*)

(while) Evaluates other expressions while test expression is true
(while *test statement ...*)

(write-char) Writes the character to a file
(write-char *character [file-descriptor]*)

(write-line) Writes the string to a file
(write-line *string [file-descriptor]*)

X

(xdroom) Determines the amount of space for xdata still available for an entity
(xdroom *entity-name*)

(xdsiz) Determines how much space a list takes up as xdata
(xdsiz *list*)

Z

(zerop) Determines whether this number is zero
(zerop *number*)

#

- (+) Returns the sum of all numbers
`(+ number1 number2 ...)`
- (-) Subtracts second (and following) from first number
`(- number1 number2 ...)`
- (*) Returns the product of all numbers
`(* number1 number2 ...)`
- (/) Divides the first number by the following numbers
`(/ number1 number2 ...)`
- (~) Applies the 1s compliment (bitwise NOT)
`(~ number)`
- (=) Compares arguments for equality
`(= item1 item2)`
- (/=) Compares arguments for inequality
`(/= item1 item2)`
- (<) Returns T if first argument is less than others
`(< item1 item2)`
- (<=) Returns T if first argument is less than or equal to all other arguments
`(<= item1 item2)`
- (>) Returns T if first argument is greater than all other arguments
`(> item1 item2)`
- (>=) Returns T if first argument is greater than or equal all other arguments
`(>= item1 item2)`
- (1+) Increments number by 1
`(1+ number)`
- (1-) Decrements number by 1
`(1- number)`

