# RocketCad

## Manual and Notes

## Contents

**The RocketCad License Agreement**

Important: Read this before using RocketCad.

This document is a legal agreement between you and Rocket Software Ltd. (Rocket). Use of any portion of the RocketCad package indicates your acceptance of these terms. As used in this License Agreement, the term "Software" means the RocketCad package or any part thereof.

If you do not agree to these terms and conditions, you must erase or destroy any copies of the software package or any part thereof, including but not limited to program files, configuration files, descriptive or instructional document files, and drawing files, printouts, and copies made in any form whatsoever.

1. Proprietary Rights.
The Software and any accompanying documentation are the proprietary products of Rocket and are protected under international laws and international treaty provisions. Ownership of the Software and all copies, modifications, translations, and merged portions thereof shall at all times remain with Rocket or its licensors.

2. Grant of License.
The Software and accompanying documentation are being licensed to you, which means you have the right to use the Software only in accordance with this License Agreement. The Software is considered in use on a computer when it is loaded into temporary memory or installed into permanent memory. This License may not be assigned or otherwise transferred without prior written consent from Rocket, and any unauthorized transfer is null and void. You may not sublicense, lease, sell, or otherwise transfer the Software or any of the accompanying documentation to any other person.

Number of Copies Licensed.
If you have not purchased a license that authorizes use of the Software on multiple computers or by multiple individuals, then you are authorized to use only a single copy of the Software on a single computer. Only one copy of the Software may be created for archival or backup purposes. All copies of the Software must include the Rocket copyright notice and other legal notices.

Term.
This license is effective from your date of purchase and shall remain in force until terminated. You may terminate the license and this License Agreement at any time by destroying the Software and the accompanying documentation, together with all copies in any form.

3. Nonpermitted Uses.
Without the express prior written permission of Rocket, you may not (a) use, copy, modify, alter or transfer, electronically or otherwise, the Software or documentation

except as expressly permitted in this License Agreement, or (b) translate, reverse program, disassemble, decompile, or otherwise reverse engineer the Software, provide RocketCad or any portion thereof to any party for any of these purposes, or purchase or use software which has been copied from RocketCad or based on the functionalilty or look and feel of RocketCad or any part thereof.

4. Limited Warranty.
(a) Rocket warrants to you, the original end user, that the Software, other than third-party software, will perform substantially in accordance with the accompanying documentation. This Limited Warranty extends for ninety (90) days from the date of purchase.

(b) This Limited Warranty does not apply to any Software that has been altered, damaged, abused, mis-applied, or used other than in accordance with this license and any instructions included on the Software and the accompanying documentation.

(c) Rocket's entire liability and your exclusive remedy under this Limited Warranty shall be the repair or replacement of any Software that fails to conform to this Limited warranty or, at Rocket's option, return of the price paid for the Software. Rocket shall have no liability under this Limited Warranty unless the Software is returned to Rocket or its authorized representative, with a copy of your receipt, within the warranty period. Any replacement Software will be warranted for the remainder of the original warranty period or 30 days, whichever is longer.

(d) this warranty is in lieu of and excludes all other warranties not expressly set forth herein, whether express or implied, including but not limited to any warranties of merchantability, fitness for a particular purpose, non-infringement, or warranties arising from usage of trade or course of dealing.

5. Limitation of liability.
In no event shall Rocket's liability related to any of the software exceed the license fees actually paid by you for the software. Except for a return of the purchase price under the circumstances provided under the limited warranty, neither Rocket nor its suppliers shall in any event be liable for any damages whatsoever arising out of or related to the use of or inability to use the software, including but not limited to direct, indirect, special, incidental, or consequential damages, and damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss, even if Rocket Software Ltd. has been advised of the possibility of such damages, whether such liability is based on contract, tort, warranty, or any other legal or equitable grounds.

6. No waiver.
Any failure by either party to this agreement to enforce a specific part of the agreement in a specific situation is not a waiver of rights under the agreement. The party may still enforce the rest of the agreement in that situation and may still enforce some or all of the agreement in other situations.

7. This License Agreement constitutes the entire agreement between you and Rocket pertaining to its subject matter. Even if part of the agreement is held invalid, the rest of the agreement is still valid, binding and enforceable.

**Introduction.**

RocketCad is an AutoCAD package for Electrical design. It contains a complete set of electrical symbols, programs to insert them, and a large number of utilities to work with them and with other drawing entities.

RocketCad is fast, simple and powerful. It allows the construction of drawings quickly and cleanly, requires no complex setup or initialization, and will work with drawings which were drawn with other packages or with an unmodified version of AutoCAD.

RocketCad is easy to begin using: blocks are inserted from the the icon menus, entity creation and modification is performed with commands selected from the pulldown menus, from toolbars selected by the user, or from the command line. As the user becomes familiar with the advanced features he will find the program allows him to solve problems more easily and to save time by automating repetitive tasks. Rocket is both powerful and easy to get started with. It makes drafting easier and more interesting and if carefully used will allow you to turn out more professional drawings which convey information more quickly and with greater clarity.

RocketCad was developed based on, and is heavily influenced by user feedback. As a result features which are irritating, non-intuitive, or which require an unnecessary amount of input from the user have been either ruthlessly deleted or rewritten. This process continues: if you are unhappy with a feature, can suggest a way in which it might be improved, or wish to suggest a new function or block, please phone or email us; if you find a part of Rocket difficult to use, or feel that the it could operate in a more intuitively correct way, let us know and we will try to improve it.

**Installation**

These instructions are for RocketCad with AutoCAD 2000i. If you wish to install Rocket with AutoCAD R14 you will have to download the installation instructions from our website: http://www3.telus.net/rocket

If you wish to set Rocket up manually or wish to use a different location for the Rocket files, there is a more detailed instruction file, Install15.txt, on the web site.

1. Unzip the file Rocket15.zip, maintaining the directory structure, onto your hard drive, typically C:. This should give the directories
C:\Rocket
C:\Rocket\Lisp
C:\Rocket\Blocks
C:\Rocket\Terminals
each containing a number of files. The file manager program Windows Commander, available from http://www.ghisler.com allows you to treat a .Zip file as a directory and makes this procedure extremely simple.

2. Start AutoCAD. Open the Preferences dialog box by entering Config at the Command: prompt.

3. Under the Profiles tab, select Import, and then use the dialog box to show Acad where the file Rocket.arg is located, and it will create a new profile called Rocket. Make this the current profile. (If you wish to keep the original profile select "Add to List", call the new profile Rocket, overwrite it with Rocket.arg, and make it current.)
Click on the OK button at the bottom of the box to save your changes.

4. Now exit and restart AutoCAD.

You should have proper icons in the screen menus and not happy faces, the icon menus should display pictures, and commands from the pulldown menus should work properly.
If you have trouble, see the section *Troubleshooting*.

Since the desired toolbar setup varies from user to user, only the Rocket icon menu toolbar is displayed by default. The Toolbars command under the Settings menu allows the user to activate any of the standard Acad toolbars.

**Getting Started**

RocketCad doesn't require any complex drawing setup. However:

**Scaling**

RocketCad uses the value of the system variable Dimscale to scale entities (typically blocks and text) which are scaled to match the scale of the drawing.

You must set Dimscale to match the scale of the drawing – this will typically be the scale of the title block. Type *Dimscale* and you will be asked for a number, input the scale of the drawing and press <Enter>. Dimscale is not specific to Rocket, so it will already be set correctly in most existing drawings.

Schematic drawings are usually done at a scale of 1:1, layouts at a large enough scale for the title block to cover the desired area. If you don't understand how a scaled drawing is set up, see the section *Scaling in CAD*.

**Linetype Scale**

Ltscale, the linetype scale, is set to 10 x Dimscale. This setting is correct if you are using the standard AutoCAD linetypes in the file Acad.lin. If you have your own linetypes then you can set Ltscale to whatever value is appropriate.

RocketCad uses Acad.lin with a number of added composite linetypes. These are found under the *Linetype* pulldown menu.
in the setup file Acad2.lsp

**Snap**

Blocks in schematic drawings are intended to be used with a snap of 2.5. Those in layout drawings are based on a snap of 1 x Dimscale. We recommend setting snap to these values and leaving it on whenever possible. Snap settings are found under the Settings pulldown menu.

The F12 key turns Snap on and toggles it between Dimscale and 2.5 x Dimscale. (As always, F9 turns Snap off and on.)

Using snap allows you to make neater and more professional drawings in less time, and makes them immensely easier to edit. Snap is a tool which will make your work easier, not an infringement on your freedom.

**Acad2.lsp**

This is the file which Rocketcad uses to initialise drawings, set up system variables, load programs, etc. It is loaded each time a drawing is opened, and if you want to change the way AutoCAD behaves this is where most changes can be made. This isn't usually necessary, but here is an outline.

Acad2.lsp is divided up into sections as follows:

- *File header, timers, Lawn.lsp* – none of this stuff should need to be changed. Ideally the settings most likely to be changed should be at the top of the file, but some programming needs to be loaded before anything else is done.
- *Rationalise a few system settings.* This is where system variables are set for each drawing. They are in alphabetical order with a brief note on each one. Note that the sysvars set here are only a fraction of the total, you can add your own if there

are things you need to set up. The online AutoCAD help contains a complete listing.

- *Load and optionally run lisps.* Everything after this controls lisp loading.
- *This section is for user lisps and settings…* As it says, any programs you want to load or settings you want to change can be put here so that you can easily find them again. This is just a suggestion of course, you can change anything and put them anywhere you like. You can also add comments anywhere you want by preceding them with a semicolon, and stuff you want the program to ignore but don't want to erase can be commented out in the same fashion.
- *Load the drawing history tracker …* down to *End of setup routines* and then to *Block inserters and line breakers.* None of this should need to be changed.
- *Autoload stuff – fully load only when used.* Everything after this loads lisp files, in alphabetical order as described in the last section of this manual.

**Loading your own Lisps**

1. You can preload your own files by adding them to Acad2.lsp in the format *(autoload "abc" '("abc")) ; alphabetise text vertically* but substituting your file name for *abc* and an optional brief description for the text after the semicolon. This preloads the file which is faster than completely loading each file before it is needed. The first abc is the file name, and the second is the name of the command which, when entered, will cause the file to be loaded. You can put more than one command inside the second set of parenthesis, separated by a space: *(autoload "abc" '("abc" "def" "ghi")).* Entering any of the last three commands will load the file *abc.lsp* and then run the command which was entered.
2. You can put the files you want to load in the directory *\Rocket\Lisp\Load*. This isn't quite as fast as the first method, but if you aren't loading hundreds of files it is much simpler.

**Layers**

RocketCad comes with preset layers. If you wish to alter the name it uses for a layer you can insert a line as follows in the file Acad2.lsp, ideally in the *User Lisps and settings* section:
(setq oldname "Newname")
For instance if you want to put all your text which would ordinarily be on the layer Text2 onto a layer called NormalText, then use the following:
(setq text2 "NormalText")
(You can place these lines anywhere in the file, but putting them in the *User…* section ensures that you can find them again, and that you don't accidentally dump them into the middle of a block of code.)

**The Icon Menus**

Most of the Rocket Blocks are inserted from the Icon menus: Single Line, Layout, Miscellaneous, I/O Schematic, and Terminal Blocks. Each one contains the blocks required to work on that particular type of drawing. Any block can of course be inserted

into any drawing if required, but since a specific device may be represented differently on (for instance) a single line diagram and on a layout it is wise to check under the appropriate menu first.

A printout of the menus would be superfluous here since each one contains both an image of each block and a description. The icon menus have as far as possible been restricted to one screen each, but in some cases it may be necessary to scroll up and down. It is recommended that the new user spend some time looking at the menus to acquire a basic familiarity with their arrangement.

**The Pulldown menus**

AutoCAD currently allows 16 pulldown menus, and RocketCad uses 15, the 16th being intended for company specific programming. A complete listing of every command available in every menu would be both pointless and impossible to slog through while retaining any useful information, this section is intended only to give an idea of which types of commands are found under which pulldown so as to provide a starting point for the novice.

Most of the RocketCad menus are based on the standard AutoCAD pulldown menus of the same name. It is a good idea to play with the menus and familiarize yourself with the different commands and see what is available that might be useful. If you are bored it is probably time you explored another menu, you will as a result become more proficient and you might find something to help deal with the boring part of your work more quickly.

If there are commands you don't know or do not understand, try them out, look them up in this file and in the Acad manual, and read the help text which is displayed on the status line at the bottom of the screen when the cursor is held over a menu item.

The 15 pulldowns are as follows:

**File**

This contains all the commands necessary to deal with files: open, save, close, audit, recover, also xref and image controls, import, export, plot, windows and web stuff, previously opened files, Autocad configuration, etc.

**Settings**

A variety of snap settings, the main ones relative to dimscale, this is a good place to look to see just what dimscale is and what snap you are using. Also contains pickbox size adjustment, ucsicon settings, and any drawing settings which can be set through a dialog box. At the very bottom is the list of currently active drawings, you can switch to another by clicking its name, or use the Window Stuff flyout under Files to tile or cascade them.

**Assist**

Undo, redo, drawing diagnostics and cleanup, cut and paste, osnaps, group creation, management, and deletion utilities which are worth taking a look at – Autodesk aren't pushing groups much any more, but they are very useful, like blocks you can turn on and off - calculator, spell check, rename, and the time.

**Draw** and **Construct**

The entity creation commands are located under the Draw and Construct Pulldowns, and the layout is similar to that of the stock menu. Other less standard items are found under Electrical and Geometry.

**Modify**

This includes most of the stock modify commands, and a few Rocket ones. While much of this stuff can be as easily accessed using the command line or toolbars, this serves as a handy reminder of what is available, and the status line help is useful. Again, don't assume that what you know is all there is – AutoCAD is very powerful. I worked with a fellow who after eighteen months was aghast to find that there was an Extend command. Missile is worth trying out if no-one is watching.

**View**

Zooms, Pan, Paper Space, Viewports, Lock and Unlock viewports, Hide, Shade, Render. Some of this stuff is superfluous if you like the new dynamic zooms and pans, but if you find that they only slow you down then this will be useful. The remainder is essential for viewport management, especially if you don't want to clutter up your screen with toolbars for things you must have but don't use very often.

**Text**

Make it, find it, edit it, realign it, rejustify it, change height, style, width scale, centre it in boxes. You will spend a lot of time working with text, and this is a comprehensive set of tools.

**Electrical**

Line numbers, wire and cable tags, junction boxes, building lights coil and tape spares, and a variety of other utilities. Completely useless to other disciplines, this is the nuts and bolts of electrical drawings. You should read the status line help on each of these at least once so that when you need it you will at least know that it exists.

**Hatching**

Area classification hatching and associative bhatching, cable tray hatching, also hatch edit and the simple one click disassociator, area classification legend, tray makers, heat trace installers and legend and the heat trace length measurement utility, and the ground a whole building in one shot routine. Actually this isn't just hatching, but there is a limit to what you can put in a title.

**Colour**

This is quite simple, just a few utilities to change the colour of an existing entity and reset the creation colour for new entities. These are very handy despite the general rule that all entities should be coloured bylayer and then put on a layer having the appropriate colour.

**Linetype**

Change things to a given linetype, make linetypes current, draw different ones. Also includes a selection of composite linetypes, although these should be used sparingly if the drawing isn't to become busy and confusing.

As with colours, most entities should be linetyped bylayer.

**Layers**

All the standard RocketCad layers with a status line explanation of what each one is for, and an array of layer management utilities. Proper layering isn't difficult to do unless you are working for a company with a pointless and impossibly complex layering system, but it is unusual to see a drawing package which is correctly and consistently layered. The problem is exacerbated when jobs which are not brilliantly drafted are copied and used as the basis for other work or become the de facto standards.

If nothing else, it is worth trying out the first item on this menu, Ezlay, which lets you see what is on which layer and helps to do something about it. More complete Ezlay help is included in its own section.

**Geometry**

This is kind of a catch-all area for anything having to do with general geometry – put existing entities on snap, clean up around them, neaten up the relationship between objects, reposition them, make clouds, brackets, spirals, vessels, cylinders, and various other things. Most of this is serious but a few items are purely entertaining.

This section can make a big difference between a drawing which is barely legible and one which people are happy to have.

**Blocks**

Write (wblock) an existing block to the disk by selecting it, find all of one type of block, erase all of one type of block, replace all of one block, minsert edit, insert bill of material lists and lamicoid schedules and update them from a text file.

The final pulldown is labelled *Available* and it is intended for programs which are specific to an individual company.

**Pen Weights**

Here are the suggested lineweights for use with the Rocket block set. These are of course suggestions and can be modified to suit drawings which contain different block sets or which have been rehashed under so many different sets of standards that there is no order at all.

In the latter case an overall lightweight line width (say 0.2mm) with a couple of heavier widths for the title block colours (unless they are also prominent in the drawing) is the best way to produce a legible result without completely redoing the drawing.

| Colour | Pen # | A Size | B Size | C Size | D Size | E Size |
|---|---|---|---|---|---|---|
| Red | 1 | 0.075 | 0.15 | 0.3 | 0.45 | 0.6 |
| Yellow | 2 | 0.1 | 0.2 | 0.4 | 0.6 | 0.75 |
| Green | 3 | 0.05 | 0.1 | 0.187 | 0.25 | 0.35 |
| Cyan | 4 | 0.2 | 0.375 | 0.7 | 1.05 | 1.5 |
| Blue | 5 | 0.15 | 0.325 | 0.6 | 0.9 | 1.2 |
| Magenta | 6 | 0.05 | 0.1 | 0.187 | 0.275 | 0.35 |
| White | 7 | 0.125 | 0.23 | 0.5 | 0.75 | 1 |
| Grey | 8 | 0 | 0 | 0 | 0 | 0 |

**Paper Sizes**

These are the most common ANSI paper sizes with the closest match in ISO standards.

Note that each ANSI size is exactly twice as large as the next smaller one – two A size sheets will exactly cover one B size, and so on. ANSI sizes with the exception of A come out to an even number of inches, the corresponding ANSI metric size is rounded to the nearest mm.

ISO sizes don't work quite so neatly, but each sheet size is more or less twice the size of the preceding one and comes out to an even (if weird) figure in mm. A9 is included for those who have always wondered if Post-It notes were a legitimate paper size.

Imperial sizes aren't included, but if you want a really large sheet the Imperial Eight Crown measures 57½x 41¾inches.

| ANSI | Inches | mm | | ISO | Inches | mm |
|---|---|---|---|---|---|---|
| A | 8.5 x 11 | 216 x 279 | | A4 | 8.27 x 11.69 | 210 x 297 |
| B | 11 x 17 | 279 x 432 | | A3 | 11.69 x 16.54 | 297 x 420 |
| C | 17 x 22 | 432 x 559 | | A2 | 16.54 x 23.39 | 420 x 594 |
| D | 22 x 34 | 559 x 864 | | A1 | 23.39 x 33.11 | 594 x 841 |
| E | 34 x 44 | 864 x 1118 | | A0 | 33.11 x 46.81 | 841 x 1189 |
| | | | | A9 | 1.46 x 2.05 | 37 x 52 |

**Troubleshooting**

If something doesn't function correctly it is probably because AutoCAD can't find a file it needs, or has an old or incorrect copy.
First go Config: Profiles, and make sure that you are using the Rocket profile. Then check your directories and the settings under the Files tab.

The major configuration files which Rocketcad uses are Acad.lsp, Acad2.lsp, AcadDoc.lsp, Acad.pgp, Acad.lin, and Electric.slb.
They should all be located in the Rocket directory, so that if there are other copies of any of them on your system Rocketcad will see the correct ones first and ignore the others.

The included utility Trouble.lsp will tell you which files you are using. If Trouble can't be run then you are probably not loading Acad2.lsp properly, which means that AutoCAD either can't find it or is using the wrong Acaddoc.lsp - this usually means that there is a path problem.

The command (findfile "filename.ext") can be also used to see which copy of a specific file AutoCAD is using. Enter the text, including the brackets and quotation marks, at the command line, substituting the name of the file for which you are looking for filename.ext. (You must include the extension.)
Autocad will return the path to the copy of the file it is using, or nil if it can't find one. If it can't find one then you will have to either add it or add a path to it, if it is using the wrong one then you have to delete it or put the correct copy ahead of it in the path.

Beyond this, reboot, look for corrupt files, make sure that AutoCAD itself is operating correctly, check to see that you have a computer and not an aquarium, call someone who knows what they are doing, email me.

**Block Problems**

If a block does not come in as expected – it is too small or too large or does not look right, it is probable that the block definition in the drawing is not the one from RocketCad but has the same name. Use the DDinsert command, find the correct block in C:\Rocket\Blocks or C:\Rocket\Terminals and insert it, when AutoCAD asks if you should redefine it, say Yes. Regenerate the drawing and the blocks should all change to be the correct rocket one.
This may cause existing blocks to display at the wrong size or to change their appearance. In this case you can either Undo until everything is back to normal, use Scabl to rescale all of the redefined blocks, or Rename the original block to something else before you insert the Rocket block.

**Linetype Problems**

If linetypes don't display properly then you may have Ltscale set to something wildly wrong, you may be loading your linetypes from the wrong file (often Acadiso.lin), or they may have come with the drawing from a company which used different standards.

Linetypes are stored in the drawing and are not reloaded when it is opened, so after checking that Ltscale is correct and that it is not assigned by entity (this is usually not a good idea) you should reload them using the Linetype command.

Linetypes which are assigned by entity can be changed with the properties dialog – an Ltscale of 1 matches the overall Ltscale of the drawing.

**Text Problems**

There are so many things that can go wrong with text that it is very difficult to diagnose. The program Styx resets the Standard style to use the Romans font rather than the Txt font, which clears up a lot of problems. Also using the Style command to reset individual styles so that they use a stated text height of 0 - and are thus not fixed height – is a good idea unless your company actually prefers to use fixed height text.

Many people like Mtext a lot, even for single lines of text, but in practise it is clunky and irritating. Mex.lsp will explode it all back into normal text.

If text refuses to behave properly it may be loose attdefs, which are typically the result of someone working on a drawing who doesn't know how to edit the attributes in a block. Cat.lsp will make all loose attdefs into text. (And attributes in blocks are edited with with D, or with De, or - if you don't have RocketCad - with Ddatte.)

**Corrupt and Crashing Drawings**

If AutoCAD crashes but offers to save your changes it is a very good idea to say "No." This will quite often leave you with a corrupt drawing which can't be opened.

Your Automatic save time (under Config>Open and Save) should be set to no more than ten minutes, and you should check the *create backup copy with each save* pickbox. Thus if a drawing crashes you can look for files with the .ac$ extension in your Temp directory and not lose more than ten minutes of work; also you will probably have a backup file.

If you are not familiar with backups and drawing recovery then it is best, if at all possible, to get expert help before trying to deal with lost information.

AutoCAD 2002 update: it is safe in 2002 to allow Acad to save the changes when it crashes, the possibly damaged information is saved to a different file which can be examined without overwriting the original.

**General Drawing Troubleshooting**

I recently tried to edit a drawing of moderate size (about 400k) which took several minutes to load and two minutes to regen. Here is a list of the things I did to find the problem.

- First of all save the drawing (unless you have just opened it) and then Audit it. If it crashes on auditing, or you can't open it, you can Recover it from within Acad

without opening it first. If Audit finds errors I run it a second time just in case, if a drawing can be successfully recovered I Audit it as soon as it is opened.

- Run Beaker. This will often delete problem entities. There is little point in going to great lengths to fix something you don't need.
- Purge the drawing, using the purge command or Splurge. This usually reduces the size of the drawing (sometimes by as much as 95%) and often solves size and speed problems.
- Check for groups – drawings containing upwards of 40,000 aren't all that unusual. The *Groups* flyout under the *Assist* menu contains several ways to deal with them. I use Grpid to count and identify them, and then either Gremp to kill all of the empty ones or Grak to kill all of them, depending on whether or not they are really needed. (Killing a group doesn't affect the entities it contained.)
- Look for excessive numbers of shape files – typically any at all. Shag can find these and tell you which linetypes and blocks contain shapes. A shape which needs a file which is unavailable doesn't display or affect the area displayed by a Zoom Extents, but it slows down regens. The problem in the drawing I was trying to fix was 908 shapes which required a missing file. Shag located them and I used Erase previous to kill them, after which the drawing behaved normally.
- Entities on Frozen and Off layers can affect drawing performance. An electrical schematic I recently edited contained a huge amount of piping on several dozen frozen layers, and fifty or so unrelated rev clouds on others. Turn on all layers, zoom extents, and use Ezlay to see what is going on and erase superfluous ones.
- If Paper space isn't used, check it for junk.
- Finally you can search for invisible entities by zooming out – Zoom 0.01x – and using Fint to locate entities out in space.

This is a good starting point for a variety of types of trouble – huge files, zoom trouble, slow regens, and other miscellaneous misbehaviour.

**Zoom Problems**

These are usually caused by invisible entities, and Beaker will delete the majority of them. Failing this try Shag and then the list in the previous section.

**Attdefs in The Wrong Place**

Sometimes when you open a block to edit it the attdefs are not where they should be. This is a bug in AutoCAD. If you ignore it the block will insert correctly, but if it irritates you the routine Irg.lsp (short for Independent Regen) will fix it – just select the problem attdefs or enter <Return> to select everything onscreen.

Bear in mind that this may recur if you open the same block again, since the problem is in AutoCAD and has nothing to do with the attribute definitions themselves.

**Xref Trouble**

If a block can't be xrefed, AutoCAD will usually display a cryptic message saying what went wrong.

The more common problems are:

1. The file doesn't exist, or at least isn't where it was. At this point you can go and complain to mechanical or piping. They will point out that you should have put in the "Warning this file is in use as an Xref" which is found under the File pulldown, Xref flyout. In theory this prevents anyone from moving or renaming the file, although in practice other departments are more like a herd of wildebeest than a book on organization.
2. Some layer names in the xref file are too long, so when the filename is prefixed to them to make the layer name the result is over 32 characters. The solution is to get into the xref and shorten any layer names over about 25 characters, this shouldn't be a problem since legitimate layer names are usually short. You can also check to see if they have nothing on them and if so purge them out of the drawing. This shouldn't be a problem as of 2000i, since it allows much longer names and also is aware of this problem and warns you.
3. There is a block xrefed into the drawing that you are trying to xref which is the same as an ordinary block in the current drawing. Go into the reference drawing and then go Xref and Bind and Insert the offending block. If anyone complains, laugh maniacally.

**Changes to the Wrong Drawing**

Sooner or later you will make changes to the wrong drawing, probably when a designer gives you markups on a drawing and forgets to tell you that it is to be used as a prototype. So you want to save the changes, but you also want the original drawing. How to do this?

1. *Saveas* the drawing to a new name. This saves the current state.
2. *Saveas* again, back to the original name, overwriting the old file.
3. Undo back to the original state.
4. Qsave.

That's all there is to it. You should try this out *before* you are in a panic, though, just to make sure you understand.

**Hatch Problems**

You try to hatch a large area with a small hatch scale, or with a pattern like Dots which generates a lot of entities. Nothing happens.

This is generally because you have exceeded the maximum number of hatch subentities which AutoCAD is set to allow. This can be changed by typing:
(setenv "maxhatch" "100000")

You can put whatever value you like instead of 100,000: The minimum is 100, the default is 10,000 and the maximum is 10,000,000. (Presumably there is some tradeoff between performance and size.)

Note that this is an environment setting and not a system variable – you can't set it with Setvar.

**Selecting just a Hatch**

The system variable Pickstyle controls what entities are selected along with the thing you pick. There are four possible settings:
0 = Off: only the entities you actually select are highlighted.
1 = Any group the entity is part of is also selected.
2 = If the an associative hatch is selected then the boundary entities are also selected.
3 = Both 1 and 2.

The System Settings section of Acad2.lsp sets Pickstyle to 1: groups are selected, associative hatch boundaries aren't.

You can turn this off by putting a semicolon in front of this line in Acad2.lsp:
*(setvar "pickstyle" 1) ; allow group selection*
Then you can set Pickstyle directly from the command line or in the *Options* dialog box, on the *Selection* tab, using the last two check boxes in the *Selection Modes* area.

**Erasing an OLE picture**

Some companies insert logos and other images into drawings as OLE (Object Linking and Embedding) objects. These are generally not a good idea, but as always the ability is there so people have to use it.

You can resize and move these irritating things by clicking on them and moving them with their grips. They are quite difficult to erase, especially if you have the right click shortcut menus disabled, which is quite common if you want to use Acad for something, i.e. producing drawings.

So: type *Config*, which opens the preferences dialog box. Go to the *User Preferences* tab and select the *Shortcut Menus in Drawing Area* check box. Select *OK* to exit the dialog.

Now right click on the OLE object. There is no Delete command, but if you use *Cut* it will be removed from the drawing (and placed in the clipboard, and overwritten next time you cut and paste something.)

It is a good idea to turn off the shortcut menus again.

**View Twist Problems**

The geometry is straight when viewed in paper space, but when you get into model space it is at an angle and is thus difficult to work on. You can rotate it to a reasonable angle but then everything is crooked in paper space.

The view twist is set with the dview command. First get into paper space and into the viewport which is twisted.

*Command:* **DVIEW**
*Select objects or <use DVIEWBLOCK>:* **<Return>.**
*Enter option*
*[CAmera/TArget/Distance/POints/PAn/Zoom/TWist/CLip/Hide/Off/Undo]:* **TW**
*Specify view twist angle <314.52>:* **0 (or whatever angle you require)**
*Enter option*
*[CAmera/TArget/Distance/POints/PAn/Zoom/TWist/CLip/Hide/Off/Undo]:* **<Return>.**
*Regenerating model.*

**Scaling in CAD**

In manual drafting (and printing, and thus everything we ever see on paper) everything is made much smaller so that it will fit on the page. We intuitively expect that electronic drafting will be the same.

It isn't.

The only rule you need to know is that everything in a CAD drawing is drawn life size. A coke bottle is drawn 240mm high (assuming that you are working in metric units), and aircraft carrier is drawn say 160,000mm long.

When you want to make a paper copy, you typically want to put a title block around it. If you insert a 24" x 36" title block around the coke bottle, everything will be fine. If you insert it onto the aircraft carrier it will be too small to notice, so you have to scale it up. If you make it 200 times as large, it will be 182,880mm long (36" x 200 in millimeters), which will perfectly surround the carrier.

When you print the two drawings you tell the printer that the paper size you want is 24" x 36", and to make the drawings fit on the page. The coke bottle prints out as is. The aircraft carrier drawing is 200 times too big, so the printer shrinks it to fit on the paper, which is $1/200^{th}$ of the electronic size (since you had to make the title block 200 times as big) and prints it.

Since the title block was made 200 times bigger in the file and then printed at $1/200^{th}$, the printed title block will be exactly the right size – 24" x 36". Everything else in the drawing will be $1/200^{th}$ the original size: drawing is 1:200 scale.

If you took the aircraft carrier drawing and inserted it into the coke bottle drawing, the carrier would be exactly the right size compared to the coke bottle since it was drawn life size. The title block would be much too big.

Also, any text in the aircraft carrier drawing must be made 200 times bigger, so that when everything is shrunk down 200 times to fit on a sheet of 24" x 36" paper it won't be too small to see. Our standard text height is 2.5mm, so on a 1:200 drawing it has to be drawn 500mm high. So any text in the aircraft carrier drawing is going to be bigger than the coke bottle.

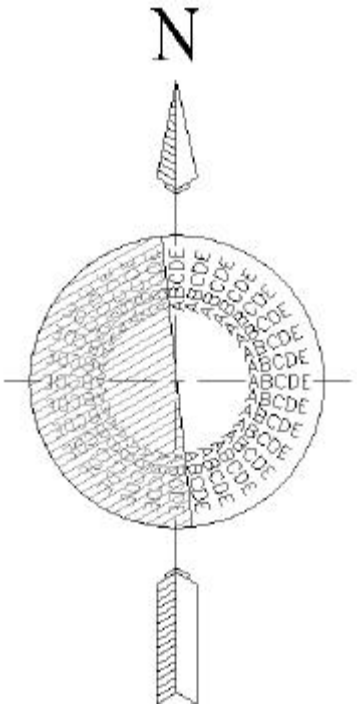
That is all there is to scaling CAD drawings – real objects are drawn at life size, text and title blocks are drawn bigger depending on the drawing scale. Usually the most convenient way to decide what scale a drawing needs to be is to draw everything in it, then insert a title block and scale it up to fit around the drawing. If you have to scale it up to ten times the original size then the drawing will be 10:1, and text must be inserted at ten times the normal height, which will be 25mm.

Of course drawing scale is based on a title block size. If you use a 36" x 48" title block (rather than 24" x 36") and put it around the aircraft carrier and make it 200 times larger, then the printer will scale it down to $1/200^{th}$ of the original size and it will be a 1:200 drawing. But if you print it on 24" x 36" paper, then the printer will have to scale it down even more to get it to fit, and so the drawing scale in the title block will be completely wrong *for that print*.

It is very easy to make a new print of a drawing using a different size of paper depending on whether you want to put it in a binder, hang it on a wall, or make a handy wallet-sized card. Unless the drawing is printed on paper which matches the size of the title block, the scale will be wrong. Anyone who doesn't understand this and attempts to take dimensions from the drawing based on the stated scale will get figures that make no sense at all.

The combination of real things and text in a drawing causes problems if you need to change the scale of a drawing. If you want to change a 1:30 drawing to 1:50 you can scale up the title block by 5/3 and the drawing will print at 1:50. Unfortunately all the text, which was 75mm high, will have to be changed to 125mm. The real objects in the drawing will remain exactly the same size so the text may have to be rearranged, which can be very time consuming.

**North Arrows**




North arrows should always point to the left or straight up. This is arbitrary, but there has to be a common orientation so that one doesn't have to tear the set apart because some of the drawings are upside down with respect to the others. The convention that North is at the top of the page goes back about half a millennium.

Sometimes the contents of a drawing fit better if rotated 90°, in this case they should be rotated counter-clockwise so that North is to the left and you can look at the sheet from the right hand side. If North is to the right then pointing it up would – in a bound set - put the back of the previous sheet between you and the drawing you wanted to look at.

Similarly if text must be rotated it should be between 90° and minus 90° - relative to the direction of North on the page - as shown in the diagram. If it doesn't fall between these two angles then rotating it exactly 180° will fix it.

The text on the spine of a book is backwards from this, so that when the book is lying face up on a table the writing is upright.

**Dates**

The standard date format (by the end of the last century) had come to be 99.12.02, meaning December 2, 1999. In the abbreviated form it was pretty clear that 99 meant 1999, and one could hazard a guess that 12.02 meant the twelfth month and the second day.

Year.month.day is not a universal standard, and the last two digits of the year are now small enough to be mistaken for either the month or the day: 02.04.08 can be taken as February 4, 2008 or as April 8th, 2002.

A better method would be to include all digits belonging to the year, writing 2002.04.08, so that the order in which they are to be read is clear and the year is unmistakable. Some title blocks have date areas which are too small to allow this format (although the use of the period rather than a dash reduces the width by about 20%) but where it can be used this approach seems superior.

**Real and Imaginary Entities.**

An engineer once asked me to rescale a drawing by simply making everything twice as big. When I explained that this would cause text height problems he asked how we normally scale things. I replied that we draw everything life-sized, and he replied "So why not just draw the text life-sized too?"

So why is there no life size for text?

Entities fall into three classes:

1. Solid objects which are drawn life-sized – buildings, panels, large equipment, compressors, vessels, piping, roads, etc. These are the reason for scaled drawings – if the entity is too small to show up or too large to fit, the drawing scale must be changed.
2. Those which don't exist – text, tags, leaders, and other symbolic things – which are drawn at whatever scale makes them legible without being so large that they take up too much space. Typically this is a fraction of the drawing size, so that when the drawing package is printed on the same size of paper all the text is the same size, regardless of the scale of any individual drawing.
3. Things which are too small to show up at larger scales, or which don't have a fixed appearance – switches, instruments, receptacles, etc. These are represented by blocks which don't look much like the thing they represent and which scale with the drawing.

Some imaginary things have a real size: grid lines, lease boundaries, the equator. They are drawn life size.

Lights are a special case, and depend on what type they are: incandescent lights vary in size but are fairly small and usually located on the ceiling where they are unlikely to interfere with anything, so their exact size is typically unimportant. Fluorescents are also ceiling mounted but come in standard sizes and are large enough that they may overlap if they are not carefully placed, so they are drawn life-sized.

Real entities (categories 1 and 3) are usually put more-or-less where they belong in the drawing. Purely imaginary ones can be moved around to make space for the fixed ones, hence the use of leaders.

It is notable that the most realistic things in an electrical drawing are those created by other disciplines. Electrical design is concerned with the function of equipment rather than its appearance, and unlike piping and civil design, where the form of an artifact is very closely related to its function, the exact appearance of electrical equipment doesn't really matter. A pipe needs an exact length, location, and diameter; a buried cable has be in the right general area and connect to appropriate devices. Similarly vessels have a very definite size and shape, but instruments from a variety of manufacturers will be physically different and perform the same function, and there usually wouldn't be much point in drawing an exact image of any specific one.

**Text Formatting Codes**

Here are the standard formatting codes: placing these in a line of text displays the matching symbol. Underline (%%U) causes the text to be underlined until another %%U is encountered, %%O functions in the same way.

| | |
|---|---|
| %%U | Underline |
| %%O | Line above text |
| %%C | Diameter symbol ∅ |
| %%D | Degree symbol ° |
| %%P | Plus/Minus ± |

**Rocket.shx**

The standard formatting codes are sometimes not adequate, as they do not include either a delta or an omega symbol.

Rocket.shx is an AutoCAD font which includes both of these and also allows proper stacked fractions, super- and subscripts, Mu, and Phi, and has a proper crossed Zero and a capital I with crossbars to distinguish it from a lowercase L.

Instructions:

First place a copy of Rocket.shx in the AutoCAD fonts directory. You may have to restart Acad for it to find the new font.

1. Fractions: 3 1/2" must be reformatted to 3[1\2]". The "[" shrinks the text height and moves the insertion point up by about half a line. The "\" moves the insertion point down below the original line and inserts a "/". The "]" moves the insertion point back to the original level and sets the height back to the original value.
Rocket.lsp takes a text string "Text etc 3 1/2 more text." and returns it in the form that Rocket.shx likes - "Text etc. 3[1\2] more text."

2. Superscripts: Since the "[" can be used to move characters above the body of the text and shrink them one can write things like $e=mc^2$. In order to restore the original height and position without finding a "/" in the finished text you have to use the character code %%6, as in "e=mc[2%%6".

3. Subscripts: this is the same process in reverse: use %%007 to lower and shrink the text and "]" to raise and reinflate it. $H_2O$ would be formatted as "H%%0072]O"

4. Mu: %%008 displays as the greek micron symbol μ.

5. Delta symbol: %%9 will place a small triangle Δ in the text string.

6. Phi Φ: this matches the one in the Greeks font and is inserted with %%11.

7. Omega Ω: similarly, an omega is inserted with %%12.

8. A proper multiplication sign. Rather that using a "X", which sits on the text baseline, %%5 inserts a multiplication sign.

9. The zero character has been modified to include a diagonal bar to avoid confusion with the capital "O".

10. The capital I has crossbars added to distinguish it from the lowercase L.

Summary:

- [ raises and shrinks the text
- \ inserts a / and lowers the text
- ] raises and enlarges it
- %%006 lowers and enlarges
- %%007 lowers and shrinks
- %%005 multiplication sign
- %%008 Mu μ
- %%009 Delta Δ
- %%011 Phi Φ
- %%012 Omega Ω

Notes:

Generally the short form of the character number can be used, for example %%5 can usually be written instead of %%005. However, if the next character is a numeral then AutoCAD may interpret it as part of the number: %%54 is interpreted as a 6, if the intent was to display "x4" then %%0054 should be used. A space after the number is unambiguous: "%%5 4" displays "x 4" as intended.

Some of the longer forms are a bit convoluted, although possibly they are no more trouble than cutting and pasting true-type symbols. Rocket.shx is a proper AutoCAD font, and thus does not slow down text and drawing operations the way that windows fonts can, and displays more legibly at smaller sizes.

Rocket.shx does not work with Mtext – apparently mtext does not properly recognize AutoCAD fonts. Although Mtext seems like a good idea any serious attempt to use it soon becomes an exercise in frustration, and we do not recommend it except under specialized circumstances which require large blocks of text to be manipulated.

People are hesitant to use specialised fonts because they can't distribute them freely. You can give a copy of Rocket.shx to anyone you want – it is our hope that it will become the standard for electrical drawings.

**Keyboard Shortcuts and The .pgp File**

The next page lists the more useful keyboard shorcuts from the Pgp file, placed on one sheet so that you can print it out. This is only a selection, chosen to avoid commands like dynamic 3D rotation which are fascinating but not likely to be used on a daily basis, and to fit neatly in a single sheet of paper. If you prefer a shortcut that isn't listed here you can either look in the file C:\Rocket\Acad.pgp or just try it out – many commands have several duplicate aliases. You can change the ones in the file if any of these really bother you or you really miss those you are used to.

A few other keyboard shortcuts which may come in handy:

Control – R Make the next Viewport active – useful if they overlap and you can't select one directly.
Control – I Toggle Isomode On/Off.
Control – E Switch between the isoplanes.
Control – A Toggle Groups on/Off.
F12 Toggle between 1.0 snap and 2.5 snap, turn snap on if it was off.

| | |
|---|---|
| A, | *ARC |
| AB, | *ABOVE |
| AL, | *ALIGN |
| AR, | *ARRAY |
| -B, | *-BLOCK |
| B, | *BHATCH |
| BO, | *BOUNDARY |
| BB, | *BREAK |
| BR, | *BREAK |
| C, | *COPY |
| CC, | *CIRCLE |
| CCC, | *CHGTEXT |
| CH, | *CHANGE |
| CP, | *PROPERTIES |
| CO, | *COPY |
| D, | *DDX |
| DA, | *DDATTE |
| DD, | *INSERT |
| DDL, | *LAYER |
| DDR, | *DDRMODES |
| DE, | *DEEP |
| DED, | *DIMEDIT |
| DI, | *DIST |
| DIV, | *DIVIDE |
| DO, | *DONUT |
| DR, | *DRAWORDER |
| DS, | *DSETTINGS |
| DST, | *DIMSTYLE |
| DT, | *DTEXT |
| DV, | *DVIEW |
| E, | *ERASE |
| ED, | *DDEDIT |
| EE, | *ELLIPSE |
| EL, | *ELLIPSE |
| EX, | *EXTEND |
| F, | *FILLET |
| FI, | *FILTER |
| G, | *GROUP |
| GR, | *DDGRIPS |
| -H, | *HATCH |
| HE, | *HATCHEDIT |
| HI, | *HIDE |
| I, | *INSERT |
| IM, | *IMAGE |
| IMP, | *IMPORT |
| J, | *JOIN |
| JU, | *JUMPER |
| K, | *CHAMFER |
| L, | *LINE |
| LA, | *LAYER |
| LEN, | *LENGTHEN |
| LL, | *LIST |
| LLL, | *LAYER |
| LT, | *LINETYPE |
| LTS, | *LTSCALE |
| LW, | *LWEIGHT |
| M, | *MOVE |
| MA, | *MATCHPROP |
| ME, | *MEASURE |
| MI, | *MIRROR |
| ML, | *MLINE |
| MS, | *MSPACE |
| MT, | *MTEXT |
| MV, | *MVIEW |
| O, | *OPEN |
| OF, | *OFFSET |
| OP, | *OPTIONS |
| OS, | *OSNAP |
| P, | *-PAN |
| -P, | *PAN |
| PE, | *PEDIT |
| PL, | *PLINE |
| PO, | *POINT |
| POL, | *POLYGON |
| PP, | *PLOT |
| PR, | *OPTIONS |
| PS, | *PSPACE |
| PU, | *PURGE |
| Q, | *QSAVE |
| R, | *REDRAW |
| REN, | *RENAME |
| RG, | *REGEN |
| RGA, | *REGENALL |
| RR, | *ROTATE |
| S, | *STRETCH |
| SC, | *SCALE |
| SCR, | *SCRIPT |
| SET, | *SETVAR |
| SK, | *SKETCH |
| SN, | *SNAP |
| SO, | *SOLID |
| SP, | *SPELL |
| SPL, | *SPLINE |
| SPE, | *SPLINEDIT |
| SS, | *SCRUB |
| SSS, | *SELECT |
| ST, | *STYLE |
| TI, | *TIGER |
| TR, | *TRIM |
| TT, | *TATER |
| TTT, | *TEXT |
| TX, | *TEXAS |
| UC, | *UCS |
| UN, | *UNITS |
| VP, | *DDVPOINT |
| -VP, | *VPOINT |
| W, | *WBLOCK |
| X, | *EXPLODE |
| XC, | *XCLIP |
| XL, | *XLINE |
| XR, | *XREF |
| Z, | *ZOOM |

**EzLay**

Ezlay is a layer manager: it allows you to display and manipulate the contents of each layer.

Ezlay should be loaded on your system and is run by typing (surprisingly) *Ezlay*. The contents of a layer are shown; each <Return> causes the next layer to be displayed.

Layers are displayed in the order in which they exist in the drawing, which is typically not alphabetical. You can go straight to a layer by typing its name, so 0 returns to the start. (0 is always the first layer.) You can also enter E (for Entity) and select something on the layer you want to see.

Layers containing no entities are not displayed, since looking at a blank screen isn't very informative. Blocks are displayed on the layer they are inserted on - if a block is on layer *Text* and all of its subentities are on layer *Phone_jacks* the whole thing will be displayed on *Text* and it won't show up on *Phone_jacks* at all.

Ezlay isn't useful with Xrefs, since you can't insert anything onto an xref layer. If you need to examine them in detail you should get into the original drawing and use Ezlay there.

Ezlay remembers which layer it was on, so you can quit, look at the whole drawing again to see where the layer in question fits into the overall scheme of things, and pick up again where you left off. You can also transparently pan and zoom while it is active.

Ezlay contains a number of subcommands so that in addition to examining the layer structure you can do something about it. They are all single letters so as to save wear and tear on your cartilage.

**B** Freeze all layers **B**ut the current layer.

**C** **C**olour all entities on the current layer bylayer. This is useful if you move a whole lot of things onto a different coloured layer and they don't change.

**D** **D**elete the current layer. It asks if you are sure, largely because it's traditional. (Undo will restore it.)

**E** Show the layer of a picked **E**ntity.

**F** **F**reeze the current layer.

**G** Make the contents of the current layer **G**o to the marked one.

**I** Set the l**i**netype of all entities on the current layer to bylayer.

**J** Show the text to William Blake's **J**erusalem.

**L** Explicitly **L**inetype entities to the linetype of the current layer.

**M** **M**ark the current layer. The name of the marked layer appears after the *Mark* option on the command line. The contents of the current layer can be sent to the marked one with G and its contents can be brought onto the current layer with S.

**O** Explicitly c**o**lour entities to the layer colour, so that you can move entities to another layer without changing their colour. (Sorry, C was already used.)

**P** Go to the **P**revious layer. Handy if you overshot.

**Q** **Q**uit (Esc also works.) When you quit Ezlay the contents of the last displayed layer become the Previous selection set, so you can find the layer you want, quit, and manipulate its contents.

**R** **R**edisplay the current layer. This doesn't seem to be as necessary under R14 and up, but if you want to transparently zoom and pan or get rid of marker "X"s you may need it.

**S** **S**uck the contents of the marked layer onto the current one.

**T** **T**haw the current layer.

**X** Mark (with an **x**) the insertion of everything on the current layer. This also shows the insertion of invisible entities, which is useful if Ezlay stops on an apparently empty layer or claims that a layer containing only a single dimension has 400 denizens. R will remove the markers, as will redrawing.

Related commands:
Play.lsp can find out why an apparently empty layer can't be purged.
Lump.lsp can move the contents of one layer to another.
RL.lsp can move the contents of one layer to another, and includes subentities.

**Chart**

It is often easier to type data into a text file or extract it from another program than to type it directly into a drawing.

Chart takes a character separated text file and imports it into a drawing as a chart. The resulting entities are plain text, lines, and polylines and can be further manipulated as you like.

**Setup:**

The first line of each file to be processed by Chart must contain configuration data. The first character is the character used to separate fields in the remainder of the file, after this are three mandatory fields: text height, cell height, and the gap between the sides of the text and the surrounding box.

So for a comma separated file where the text height was to be 2.5, cell height 6, and end gap 5 the first line would be: ,2.5,6,5

**Options:**

Columns default to left justification, with a width of the length of the longest text entity in that column plus twice the end clearance.

You can add a justification string for each column: C for centred and L for left justified. Any justification code can be followed (immediately - no intervening separator character) by a number indicating a fixed width to use for that column. Text entities that are too long for this space will be compressed to fit.

So if you have three columns, all to be centre justified and forced to a width of 12 units, you would add this to the end of the configuration line: ,C12,C12,C12

The configuration method is a bit odd: why not a dialog box?

Most problems suggest one of several standard solutions - explosives, firearms, drugs, legislation, and dialog boxes. Typically the first idea that comes to mind isn't the best or the only approach. The method used - the initial configuration line - saves the settings with each file, so one can import a number of charts with different layouts without losing the previous settings each time a new chart is created, and settings can be copied to other files: they don't evaporate when the program ends.

The config line method is scary because it doesn't have little pictures and requires one to read the instructions. It is about as complicated as feeding a goldfish.

**Notes**:

1. All settings in the configuration line are multiplied by the current Dimscale value.
2. The chart will be drawn on the current layer using the current entity colour, linetype, text style, etc. Text width scale factors other than 1 are ok, but fixed height text is bad.
3. A string can contain spaces, but leading and trailing spaces are ditched.
4. Columns which contain nothing are given a width of twice the end gap so that they won't vanish, as are columns with a fixed width of 0.
5. Empty rows are drawn as empty rows if they contain at least one separator character; completely empty lines in the data file are ignored.
6. For fixed width cells, if subtracting the end clearance from the cell width leaves zero or a negative number then the gap will be adjusted.
7. Weird results? You have an extra separator character (i.e. a comma) in the wrong place.

**Zlin**

Zlin creates polylines in complex patterns by drawing a repeating pattern of polyline segments at specified distances and angles between two points. The resulting entity is a single polyline.

**Use**

Zlin has four related functions which are offered when it is called from the command line:
"Save pattern/Read pattern/New pattern/<Draw line>:"
(If no pattern is currently defined one can either define or load one and the prompt is changed to:
"Read pattern/<New pattern>:")

**New Pattern** - Create a new pattern

The user is prompted to pick a start point. Successive points are saved as a set of angles and distances until a <Return> is entered: the pattern is followed with a set of temporary marker lines as it is drawn. The points used to enter the pattern do not affect the default start point.

Caution: the number of patterns drawn in a given distance space depends on the pattern length - the distance between the start and endpoint of the pattern. As the pattern length becomes smaller the number of patterns increases, as the pattern length approaches zero the number of patterns drawn becomes close to infinite, something which computers don't like - Zlin will therefore crash.

Once the new pattern is complete Zlin calls the line drawing procedure.

**Read pattern** - Read a pattern from a file

Zlin asks for a filename and if the file is found loads it into memory, replacing the existing pattern (if any). If the new pattern is successfully loaded Zlin calls the line drawing procedure.

**Save pattern** - Save a pattern to a file

Zlin asks for a filename. If the file exists the user is asked whether to overwrite it or quit (the default). Pattern files may have any name and extension, although this should be something descriptive.

A pattern may be saved to a file at any time once it is defined – you can try it out first, or save it right away. Pattern files can be manually edited: each file contains a header explaining the format.

**Draw line** - the default - Draw a Zline

Zlin asks for start- and endpoints, offering the endpoint of the last Zline (if one exists) as the default start.
It then asks for a scale:
“Number of patterns/Scale/Previous scale (1.0)/<Maximum unscaled>:”
If this is the first Zline drawn in the current editing session or the previous ones were not scaled the "Previous scale" prompt is omitted.

If **Number of patterns** is chosen Zlin prompts "Number of patterns:" (If a number is entered at this prompt it is taken as a number of patterns and prompt this prompt is bypassed.) Zlin scales the pattern so that the given number of patterns fit between the endpoints, draws the Zline, and saves the scale factor used as the next Previous scale.

If **Scale** is chosen Zlin asks for a scale factor, again offering the previous one as the default. (If there is no previous scale factor then 1 is used. The default is written as a real truncated to two decimal places, its accuracy is not affected.)

Zlin draws the greatest number of patterns that will fit between the endpoints at the resulting scale and makes up the remaining space to the endpoints with two straight line segments of equal length.

**Previous scale**: this is the same as choosing Scale and accepting the default. Zlin uses the previous scale factor again so that the size of the patterns in the Zline matches those in the previous one.

**Maximum unscaled**: the greatest number of patterns is drawn that will fit between the endpoints at full scale and the extra distance is filled with two equal straight segments, as though a scale factor of 1 was input.

**Revision history**

| | |
|---|---|
| 1.0 | The basic Zlin. |
| 1.1 | Added ability to scale patterns. |
| 1.15 | Added ability to read and write pattern files. |
| 2.0 | Immersed computer and all backups in coffee. Complete rewrite. |
| 2.1 | Removed bug which caused monitor to burst into flames. |
| 2.2 | Added ability to duplicate previous scale. |
| 2.2a | Noticed and ignored crash on zero pattern length feature. |
| 2.25 | Redesigned file format for greater clarity, added header, rewrote file reader to ignore comment lines. |
| 2.9 | Changed file format - angles are now saved in degrees to avoid multiples of pi. |
| 3.0 | Took two weeks off to hunt weasels in Mongolia. |
| 3.6 | Cleaned up subroutine organization, rewrote initial prompt. |
| 4.0 | Failed in attempt to interest coworkers in beta testing. |
| 4.1 | Wrote documentation. |

## Xrefs, Paper and Model Space

**Xrefs** are External Reference Files - they are blocks inserted into the drawing with the Xref command, and behave like ordinary block insertions except that each time the drawing is opened they are updated: if the original drawing is changed, the block displayed in the drawing is changed to match.

The advantage of this is that drawings made by other disciplines (such as piping) can be used as backgrounds and our drawings will always contain the latest information even if piping forgets to tell us that they have moved buildings, rotated the plant, installed a flare stack, etc. (It won't be less irritating, but at least we will know.)

The down side is that if the xref drawing is renamed or moved then the xref will just disappear from the drawing and be replaced with a line of text detailing where it should be but isn't. If on the other hand the drawing is abandoned and a new one made and updated then we will carry on using the wrong drawing in blissful ignorance.

Xrefs are a very powerful tool for allowing collaboration between disciplines. It is very easy to allow them to substitute for proper communication, at which point they become an attempt to solve an organizational problem with a technical fix. This will inevitably cause friction between people and departments, and it will be tempting to blame the problems on the computer system and try to fix them with more technology. If you are xrefing drawings it is essential tomake the owner aware of this and keep in touch with him so that he will keep you informed of changes which are not apparent from looking at the xref rather than ignoring you as a potential irritation located at the far end of the network.

## Title Blocks

Putting the title block in a drawing as an external reference has become quite a popular approach recently. We are of the opinion that it is a mistake.

There are two apparent motivations for this: the saving in space and greater consistency. The first makes very little sense: a well made title block should be substantially less than 100 kilobytes, and a 100 megabyte hard drive currently costs less than $300. So the drive will absorb (assuming that the tb is exactly 100k and the xref takes up no space at all) the difference between the xref and a straight insertion for one million drawings. The additional cost per drawing is 3/100 of one cent. Even were this significant, and assuming that one actually wanted to keep a million drawings on line at one time, the space savings which could be realised by purging drawings and erasing backup, error, and log files would completely swamp this trivial expense.

Secondly it is claimed that one can change the title block in the reference drawing and have the changes appear in each drawing as it is opened. This is true so far as the non-text entities are concerned, but one presumes that any change which wasn't very trivial would involve rearranging attributes, which of course can't be contained in an xref – there must be a block in the drawing which contains all of the variable data.

This means that the xref is actually quite hazardous: if it is ever changed then chances are that it won't match the arrangement of attributes on old drawings, and there will be no way to decide to leave them as-is – they will either have to be modified or a separate xref will have to be made for new drawings. The ultimate outcome of this is a very complex and confusing system, balanced against a lack of any appreciable gain.

We recommend making up a compact and well designed title block, and if it ever needs to be replaced swapping it out with program. We have used this approach very satisfactorily many times, and would be happy to provide both programming and guidelines for good title block design.

**Paper and Model Space**

Ordinarily AutoCAD operates in model space. This is the standard 3D environment which extends to infinity (or $10^{13}$ units) in each direction. Paper space is another environment which exists alongside Model Space. It is strictly 2 dimensional, and holes (or Viewports) can be cut into it which look down onto model space, but each viewport can look onto the drawing from any desired angle and show it at any desired scale. This means that a drawing can be done in 3D and then different viewports can show it from different angles and enlarge sections to show small details.

When paper space was introduced it caused a considerable uproar in the drafting community. It was a major change, and it was necessary to find a use for it. This was fairly easy: under AutoCAD R12 it was not possible to display only part of an xref, so in order to remove unwanted parts like the title block, sections of pipe floating outside the border, and buildings we don't care about we had to clip off the excess. This could have been done by turning off the layers containing the title block and other unwanted entities and by encouraging the disciplines making the reference drawings to put everything on the appropriate layers, and in many cases this workd and the xref could be treated as just another block.

However, it was again tempting to use technology to solve organizational problems, and we did so: our title block was inserted in paper space and a viewport was used to cut off and unneeded parts of the xref. This didn't really solve anything, since we had to clandestinely edit the reference drawing to clean up the layering on things that were visible in the viewport, but it gave everyone a use for paper space, so they were happy, except that it made life slightly more difficult.

With Release 14 it became possible to clip the displayed area of an xref and show only part of it, and so paper space became what it was intended to be: a tool for mechanical designers to use to show different views of one 3D model. Unfortunately many people felt that they were somehow missing something if they didn't use it, and we came to see aberrations like title blocks in paper space and everything else in model space, or text and title blocks in paper space and everything else in model space, or even everything in model space and all plotting done in one big viewport in paper space.

The average electrical drawing does not need paper space, as a general rule of thumb if you only have one viewport, if all of your viewports touch, or if they are all at the same scale, you don't need to use paper space. It is interesting to play with and useful to know about and understand, but for simple 2D drafting it will make your life more complicated without any noticeable benefits.

**Inserting an Xref**

An xref is inserted into the drawing with the command Xref. This asks for a drawing to xref into the current one and an insertion point, scale, etc. We usually insert xrefs in model space at 0,0, scale 1, although they can be put in paper space, and like any block they can be inserted at any point and scale you want.

Now use the layer dialog box to freeze any layers you don't want, like text, centrelines, titleblock, misc, topo, picture_of_my_dog, stuff_we_never_built, old_useless_revs, etc. Xref layers names are in this format: Xrefname|Layername, they can be frozen and their colour can be reset but they can't be made current. It is a good idea to change most xref layers to colour 8, which we print as a very fine line so that xref stuff - which is primarily used for a background - doesn't visually overwhelm entities we are installing.

RocketCad sets the system variable Visretain to 1, so that the xref layer settings are saved in the drawing and don't snap back to those in the reference drawing when the one it is xrefed into is opened. This is worth knowing because 1. It is wrong in the manual; and 2. We might suffer from dementia and change it.

It is a good idea to get into the xref and check that everything is on the right layers, which it won't be. You can check with Ezlay, which allows you to display the layers one at a time and suck stuff from one to another easily. (Bear in mind that people can be very territorial about their drawings and may become abusive if they find out.) Be very careful of stuff which will be on layers which are frozen in your drawing, since if it is something important you will never see it again. Similarly if someone installs a huge motor which you might like to know about but puts it by mistake on a layer that is frozen in your drawing, you will never know that it is there.

**Viewports in Paper Space**

If you are really determined to use paper space: (assuming that you are back in the original drawing and the layers are set correctly) get into paper space by clicking on the Layout tab at the bottom of the drawing area. (This is for Acad 2000 and up only, if you need to do this in R14 you will have to read the manual, set Tilemode to 0, and type PS.) A dialog box will appear and ask for all sorts of settings and give you a single viewport which can be resized by dragging the corners.

Additional viewports are inserted with the Mview (Make Viewport) command, and you can suck a title block in from wherever you like. Make the viewport about as big as the free space available in the titleblock, you can resize it later. (If you started with an existing drawing or paper space prototype you will presumably already have a title block and a viewport in paper space.)

Now make the viewport active by typing MS, or by double clicking on it. The cursor will show as crosshairs when it is over the active viewport, you can change to another one by clicking in it. While you are not over a viewport the cursor will be the standard windows arrow, you can make paper space active again by typing PS or by double clicking in an area which isn't in a viewport – the cursor will be an arrow instead of the crosshairs. Snap, scale, layer visibilities, and Ltscale can all be set independently in paper space and in each viewport.

Now (in a viewport) zoom to extents to get the whole xref on screen, and then Zoom 1XP which sets the viewport to the same scale as paper space, or if you insist on having paper space at a different scale from model space, Zoom 0.1XP will give you a scale of ten to one in the viewport. Use the Pan command to move the drawing around in the viewport until it is

positioned as you like, get into PS and resize the viewport to cut off anything you don't want, or if there is nothing you don't want, to look neat. You should also use Properties to put the viewport on the Defpoints layer so that it won't plot (the contents will).

Once everything is nicely set up you can put text and electrical stuff on top of the xref, but in paper space. Large amounts of text can be wblocked out of the original xref drawing, edited, and sucked into the current drawing in paper space. It is handy to use the corner of a building as the insertion point so that you know where to put it. All zooming and panning should be done in paper space (typically you won't need to get into model space much) so that entities in the two spaces stay aligned and at the same scale.

It is a good idea to lock your viewports once they are set up: under the View menu select Lock Viewports. Now if you try to change the view in the viewport – i.e. you zoom extents in model space without thinking – AutoCAD will drop into paper space, zoom extents, and get back into model space and the alignment between paper and model space won't be totally destroyed.

**Utilities.**

**Moss.lsp** can be used to suck entities from model into paper space.

**Spam.lsp** moves stuff from paper space into model space. Both attempt to leave them in the same position relative to existing architecture.

**Igloo.lsp** allows a line to be drawn in paper space which is then sucked into the same location in model space. This is handy for lining one entity up with another since you can osnap onto something in model space from paper space, but not the reverse.

**Markups**

The next section is provided not so much for the use of the draftsman as for the edification of those designing the drawings he will be creating or modifying – you can print it out and use it to demonstrate that it isn't just you complaining, there really are accepted and sensible ways to mark up a sheet of paper, and that if they are followed you can produce a better drawing package in less time.

**Markups: The Unlikely Ideal**

A markup isn't supposed to show exactly what the finished drawing should look like - what we really need is a document which tells us what information to put on it.

This is an attempt to clarify the process so that designers won't waste time being too neat or precise and so that they will tell us what we need to know to work as quickly and accurately as possible.

This may be a good place to point out that it is better to actually put some marks on the paper than to hand someone a blank sheet and then dictate the contents of the drawing.

**Standard pen colours**

Except for red, these aren't used much any more, but it is nice to know how things used to be. (Also this is probably a good time to introduce the painful concept of standards.)

**Red**: our favourite, because it stands out from the original black print. Anything in red is put on the drawing. (If you are working on a red drawing you may mark it up in black.)
**Green**: comments and notes to the draftsman. These are read but not drawn in.
**Blue**: this traditionally meant: "Erase this," but wildly scribbling over something in red is just as good. Detail freaks can write: "Delete this stuff" beside it so that we don't just draw the scribbles.
**Black Pencil**: this doesn't exist, even if it says it does.
**Eyeliner**, Purple crayon, Diet Coke, Transmission fluid: these aren't used much…

We are very happy if everything is done in red, although green is also useful as it saves us having to ask whether to cad the comments. If you don't have a red or green pen we will find you one.

**Legibility**

Neatness does *not* count. A lumpy trapezoid with the words: "Lighting Panel" squashed into one corner is just as easy to draw as a perfect square with illuminated calligraphy. The drafting department will make your drawings look nice, you just have to give us the information you want on the paper. Anything else is a waste of your time.

(Note, however, that time spent writing letters that don't look like a row of discarded banana peels is well repaid when we don't have to ask you to interpret twelve out of fifteen words.)

**Whiteout and tape**

We admire the handiwork of those who use these, and some of the drawings they produce are works of art. We also don't notice things we should erase because they are so perfectly obliterated, and spend a lot of time looking at the back of the drawing trying to see under the whiteout.

It is also not unknown for one to erase something only to find that it is needed three lines down. It is better to circle it and put an arrow pointing to the new location than to cover it with whiteout and redraw it.

In cases where the designer can paste in a detail cut from an existing drawing, it is very helpful if he notes where it came from so that we don't have to search for it.

**New Drawings and New Jobs**

Many times we are given a markup on an existing drawing, make the changes, and are told that it was intended to be used as a prototype for a new one. Writing: "New Drawing" in the title block saves having to undo everything we have done or redraw the original. A project name and drawing number don't hurt either.

**Markups on Signed Drawings or on Outdated Drawings**

This is a very bad thing. So is doing a markup on an old copy of a drawing so that the changes you are making conflict with the ones that have been made since the copy you are using was printed. If there is any doubt, if the drawing is more than a few days old, or if you are not the only person working on the drawings it is a good idea to either get a fresh print or to ask the draftsman to see if there is a later electronic copy.

**Scale**

Since items which have a real size are usually drawn to that size it is much easier for us to see if an object will fit into a space than it is for the designer to draw it to scale on paper.

We can also move things and try different arrangements - this is one of the few times when we don't mind having the designer stand behind us and say "Now move that to the left..."

Again, an uneven rectangle with measurements scrawled beside it is just as useful to us as a Xerox of the front of the actual artifact. If you don't know how big something is, put a note by it, preferably in green ink so that you don't have an endless series of notes: "Not sure of size - fake it." "Don't put that on the drawing." "Or that." "Or that..."

**Size**

A physical object has three dimensions, measured along the X, Y, and Z axes. In English they are Length, Height, Depth, Width, and Thickness. "240w x 300h x 1175d" doesn't tell us clearly which direction is which, and we don't need the third one. "350 x 200, long side against wall" is much clearer, or draw a rectangle and scrawl the dimensions on two sides. (If it's for a bill of materials we don't care – the guys in the shop know not to put the side with the buttons on it against the wall.)

**Abbreviations**

These can save time, provided that we know what they mean. We will typically use the long form where there is space unless you indicate that you prefer the abbreviation. We also try to use standard abbreviations: Cct. for Circuit, Gnd. for Ground, Dbmn. for Doberman. An abbreviation is always followed by a period.

Don't even think of trying to use abbreviations containing letters not in the original word.

**Project Data**

It is a good thing to have the information common to all drawings in a project as early on as possible. This helps us determine which project a drawing belongs to, and ensures that when a drawing is finished at the last minute it doesn't have to be opened again and replotted to add something to the title block.

**Timing**

We like to have some idea of when you want a job finished, unless it is in fifteen minutes in which case we don't want to know.

**Trust**

Even if you expect the job to be built by idiots there is a limit to the amount of information it is desirable to cram onto the paper. There is a point beyond which clarity gives way to confusion and the desire to convey an idea to the builder is replaced by the neurotic fear that he may miss the one thing you didn't detail.

Typically the standard details are included with a project to protect us from both builders and our own paranoia. We are also always happy to volunteer an opinion on how much detail is needed. Or in fact on just about anything.

We also have a lead sheet to explain our symbols to those who are new to the industry, and including this with a drawing package saves the time and space required to put a legend on each drawing.

**Standards, why we have them, and why you can't change them**

We have spent some time working out the best way in which to present information, and while we realise that we were placed here solely to serve as a conduit through which your thoughts may drip soggily onto otherwise pristine sheets of paper, we actually do know what we are doing.

Surprisingly, attempts to rehash standards or to ignore them just this once usually result in everyone feeling free to ignore the standards while still expecting the time savings which would have come from adhering to them.

For example our shutdown key import software allows us to produce an Sdk in ten minutes which once would have taken the better part of a day. This has been so successful that everyone has felt comfortable asking for slightly different drawings and presenting us with weird "this-time-only" spreadsheets which then become either another standard or a base for further variation. Currently we expect to spend about an hour per key searching for the drawing which goes with a given spreadsheet and the software to import it. This is time wasted not only on the oddball jobs but one every job.

**Changing your mind**

If you scribble something out and then decide that you didn't mean to erase the fire detection system, circle it and write, "ok" beside it. There is no need for long explanations or to redraw it. If you change your mind again (say you realise that the Pacific Ocean isn't that likely to catch fire) you can scribble out the "ok" or write "Kill" beside it. We are pretty forgiving of this sort of thing, although we will ridicule you.

**Units**
Here are the common units abbreviations as specified by Apegga, just as a reference for what is in upper vs. lower case.

**Units**

| | |
|---|---|
| Amperes: | A |
| Kilograms: | kg |
| Kilowatt: | kW |
| Metres: | m |
| Millimetres: | mm |
| Pascals: | Pa |
| Volts: | V |
| Watts: | W |

**Prefixes**

| | |
|---|---|
| Kilo: | k |
| Mega | M |

If these seem arbitrary and inconsistent you may take the metric system as an example of what you get when you ditch the existing standards and hack something together on the spur of the moment.

**Text Size**

Although the Law of Frontality hasn't been widely used since the end of the Egyptian Empire, I am increasingly seeing text made larger in proportion to its perceived importance. This is unfortunate when the text is part of a block that must then be redrawn and saved under a different name and any associated programming rewritten and distributed. When it is just text it is merely irritating. We typically use only 2.5 and 3mm text, but the National Enquirer usually has job openings for people who like the larger sizes.

**Later, and in a more reasonable mood: an explanation**

Originally (before there were engineers) drawings were hand drawn on 'D' size (22" x 34") sheets, mostly because at this size they held a reasonable amount of information. In cases where a smaller drawing was required – say for inclusion in a book - the original was drawn very precisely and then photographically reduced.

When computer drafting became common we continued to use the old standard page sizes, and all of our drawings are based on them.

But: smaller printers are faster than full sized plotters. We started printing drawings on B sized sheets, and since CAD is more precise than any but the most careful and time-consuming manual drafting they were still legible. The builders liked not having to wrestle with tablecloth-sized sheets of paper, and they became the standard issue size.

So we now print all of our drawings at about half the size for which they were designed, and the old manual 2.5mm text height, which was too large for machine-drawn type, ends up being 1.25mm, which is about the same size as standard printing, but too small if one is farsighted.

*The solution: actually there is no solution.*

We could make a new set standard drawings based on a 'B' size sheet, but we would have to develop and maintain two sets of standards, resulting in more time spent on every job, and much more time spent training new draftsmen.

In addition, everyone with a half-baked idea for a new standard would crawl out of the woodwork and we would end up with badly designed chaos. (As opposed to the more common well designed chaos.) Our current standards are the result of over a century of refinement and more-or-less continuous argument, and even if we had 100 years of non-billable time we aren't anxious to go through that again.

Also, as the size of text increases we can fit less of it on a drawing so we would have to split many of our drawings in two, or settle for a lot less detail.

Finally, a drawing with more than two sizes of text usually looks awful.

**What can be done?**

We can print drawings in a larger size – 'C' or 'D'. A 'D' size drawing printed as a 'C' leaves the text printed at about the size found in most books. In cases where a drawing will be consulted regularly a larger copy is usually a good idea in any case.

We can strive for better layout – I was recently asked to fix an almost illegible column of text. The problem wasn't the text height, but the spacing between lines. A properly laid out drawing is more attractive, conveys information more clearly, and is easier to read.

Finally, there comes a point when one has to admit that glasses may be a requirement.

## An Alphabetical List of Lisp Routines with a short description of each.

| | |
|---|---|
| ~ | Line break installer |
| 19 | Pipe break installer |
| Abc | Alphabetise text vertically |
| Abg | Renumber text in sets with suffixes |
| Above | Add text above an existing entity |
| Ac | Copy attribute values from block to block |
| Acla | Area class generator |
| Addtext | Add text below an existing entity |
| Angel | Change layers to PCP standards |
| Atcase | Change attr. prompt strings to lower case |
| Attinc | Incr one or all att vals in selected blocks |
| Axe | Text last/first word add/remove |
| Bar | Scale bar inserter |
| Beaker | Erase invisible and useless crap |
| Bean | Sequentially renumber terminal blocks. |
| Bic | Insert lights by rows and columns |
| Bingo | Rotate text 180 around centre point |
| Bk | Draw a break line |
| Black | Colour everything in the drawing white |
| Blam | Explode a block replace atts with text |
| Bldg | Make a cut-and-fold building |
| Bliz | Mark everything with snowflakes |
| Blockout | Wblock all blocks in a drawing |
| Blora | Vertically reshuffle block insertions |
| Blur | Replace all inserts of a block w. another |
| Blurss | Block replace only selected blocks |
| Boo | Zoom previous mark original view |
| Bock | Cloud text and/or attributes in blocks |
| Bomb | Rotate text/atts 180 round common centre |
| Bomin | Suck a csv into BOM blocks |
| Bounce | Purge a drawing down to the bare metal |
| Box | Draw a box, allows drag or x&y sizes |
| Bp | Bind permalta if it is an xref |
| Brac | Draw a bracket |
| Bracket | Draw a bracket on the end of two lines |
| Brat | Derotate attributes in all of one block |
| Brot | Rotate all of one block type 180 |
| Btray | Draw and hatch a straight tray section |
| Bullet | Suck data from all of one block to a .cdf |
| Bungee | Save layer settings to a list and a data file |
| Splash | Restores them from the internal list |
| Flash | Restores them from the external data file |
| C | Enhanced copy |
| Cable | Import a .csv into a cable schedule |
| Cat | Convert loose attdefs to text |
| Centre | Centre justify a text entity |
| Cf | Vertically rearrange text in a box |
| Chall | Text search and replace for drawings |
| Chart | Make a chart from a text file |
| Chat | Explode and reblock an insert |
| Chgtext | Everybody's favourite |
| Clk | Simple clock maker |
| Cls | Like Dos |
| Cm | New and improved cloud maker |
| Cma | Flatter cloud maker |
| Curl | Move entities to current layer |
| Coil | Coil and tape block/text inserter |
| Cols | Change entities to colour of selected one |
| Cr | Circle repair - make an arc into a circle |
| Crypt | Encryptor |
| Cspi | Temp. graphics - star multi counterspiral |
| Curl | Move entities to the current layer |
| Cyl | 2D cylinder maker |
| Dc | Dcl (dialogue box) file editor |
| Ddd | Swap empty att. values with ... |
| Ddx | Select text/attdef use matching editor |
| Deep | Direct text/attrib edit with dialog box |
| Derot | Derotate - doesn't work on circles |
| Detitle | Insert a detail tag |
| Dg | Destroy all groups containing an entity |
| Diaper | Search & replace layer names |
| Lance | Convert all to upper/lowercase |
| Dimf | Restore the default text in a dimension |
| Dimp | Change dimension leg endpoints |
| Dn | Show nesting for a block |
| Dna | Show nesting for all blocks |
| Dogbite | Chop around a circular block |
| Doglite | Chop around a light block |
| Doon | Install a conduit down symbol |
| Drawlist | Suck a .csv file into a drawing list dwg |
| Du | Write a block updater lisp |
| Earth | Draw an earth sized circle |
| Ecen | Horizontally centre an ss of entities |
| Eno | Text to incr. numbers with pre & suffix |
| End | Replaces the discontinued command |
| Enplot | Automatic plot command |
| Elsp | Spiral: ellipt./circ. straight/expon. |
| Epi | Epicycle generator |
| Exget | Get extended entity data |
| Ezlay | Same as Reptile see below |
| Fand | Text search - see also chall |
| Fang | Script file write and run |
| Fcase | Add initial caps to text as appropriate |
| Fibl | Find block by name or all blocks |
| Filo | Write ent. data to a file - see also Lox |
| Filk | Search & replace the last char in ss |
| Filthy | Mark insertions of any or all entities |
| Fint | Find entities by insertion point |
| Fifty | Increment line numbers etc. - whole dwg. |
| Firk | Search & replace the first char in ss |
| Fixwit | Find and optionally fix overly-wide text |
| Flame | Select layers freeze others or thaw all |
| Flat | Remove elevation from entities |
| Flay | Freeze all layers except the selected one |

| | |
|---|---|
| Flip | Flip single/double sided wire tag |
| Flood | Floodlight maker |
| Fluke | Chop around text/attributes |
| Front | Bring an entity to the front |
| Get | Get entity data listing |
| Getq | Extract variables from a routine |
| Gnd | Put a ground grid around a building |
| Gouge | Extract block subentity data |
| Gr | Find groups to which an entity belongs |
| Grak | Explode all groups |
| Gremp | Destroy all empty groups |
| Ground | Insert a ground symbol ideally on a line |
| Grpid | Count highlight and list all groups |
| Gyc | Centre selected entities in a box |
| Slatch | Xhatch thatch hatch installers |
| Hose | Replace solids with hatches |
| Hoss | Make an index drawing of a dir of dwgs |
| Ht | Change selected text height |
| Hvb | Move/rejustify text along a horiz line |
| Idle | Remove all rev clouds and blocks |
| Igloo | Draw a line in mspace from points in ps |
| Incend | Insert a block on the end of a line/pline |
| Inf | Print useful information about the dwg |
| Intbreak | Break a line at two intersections |
| Irg | Regen one or more entities or screen |
| Join | Join lines/plines into one pline |
| Jumper | Install a jumper on terminal blocks |
| Junc | Make a junction box |
| Kibl | Erase all insertions of one block type |
| Klat | Turn off a selected attribute |
| Ladd | Make two lines of text into one |
| Lash | Swap pline linetype gen: vertices/overall |
| Lcb | Change layer ltype and colour bylayer |
| Lch | Change layer by example/name input |
| Ldr | Load and run a lisp |
| Leda | Non-associativeleader |
| Left | Left justify a text entity |
| Lineup | Make sure lines with text are upright |
| Lix | Set limits to extents |
| Lld | Lisp loader with dialog box |
| Lms | Move stuff to a layer make it current |
| Look | Trim everything out of a rectangular area |
| Lox | Write assoc groups from ss to a file |
| Lset | Set layer to that of a selected entity |
| Lt | Change linetype to match another entity |
| Ltr | Write letters from the command line |
| Lump | Move contents of one layer onto another |
| Lx | Pull an attribute out of a block |
| Map | File editor load - like xx narrower cols |
| Marmot | Multi-attribute squash |
| Align | Text with matchlines pw lt layer |
| Mex | Explode all mtext |
| Middle | Middle justify a text entity |
| Min | Minsert editor |
| Miss | A more interesting explode command |
| Mold | Mark every entity in the drawing ... |
| Moss | Suck stuff from model into paper space |
| Mp | Match props - layer colour and linetype |
| Mull | Reorder cable tags by type |
| Mva | Move lower left of dwg to 0 0 Stockholm |
| Mx | Edit the current menu file |
| N | List entity information no flipscreen |
| Nepo | A strange point connector |
| Next | Go to the next drawing in the directory |
| Nn | List subentity information no flipscreen |
| Nt | Incremented numbers with optional text |
| Numf | Write series of numbers to a file |
| Nxlay | Go to the next layer freeze all others |
| Ofc | 2D sphere maker |
| Pang | Screen shaker |
| Pc | Draw a circle in ps over one in ms |
| Penguin | Multiple line/polyline intersec breaker |
| Pfp | Full screen text editor |
| Phdel | Delete all frozen layers |
| Pit | Rotate text into the next isoplane |
| Pivot | Rotate text & atts into the next isoplane |
| Pj | Join two noncontinuous polylines |
| Pkill | Erase all points |
| Play | See why you can't purge an empty layer |
| Poxx | Temp. graphics - spiral box |
| Punk | Insert multiple blocks increment break |
| Pur | Repostion leaders/blocks/lines/circles |
| Pw | Rewidth a polyline to half of dimscale |
| Qeb | Wblock an existing block |
| Rat | Make text into an attdef |
| Repall | Text search & replace for batch files |
| Reptile | Cold blooded layer rehasher |
| Right | Right justify a text entity |
| Rl | Relayer with subentities |
| Scabl | Rescale blocks |
| Sch | Restyle text |
| Scream | Replace one att in a number of blocks |
| Scrub | Chgtext for multiple blocks |
| Sd | I/o schematic text extractor for s/d keys |
| Sdel | Delete one type of entity from selected |
| Sec | Section arrow maker |
| Shag | See why you can't get rid of shapes |
| Shark | Kill entities under a certain size |
| Shutdown | Import an excel file into a shutdown key |
| Sklork | Vert. alphabetize att vals between blocks |
| Snake | Put entities on snap |
| Sname | Put the drawing name in the screen menu |
| Sort | Export loose text into a csv by position |
| Snat | Toggle snap dimscale - 2.5 x dimscale |
| Snort | Export loose text into a csv by location |
| Spam | Suck stuff from paper into model space |
| Splurge | Automatic purge to completion |
| Spi | Temp. graphics - star spiral |
| Spyr | Pattern maker |
| Squid | Squash shutdown atts to fit in cells |
| Squab | Rewidth attributes in a variety of blocks |

| | |
|---|---|
| Ssq | Edit multiple text ents in order by pos |
| Sta | Temp. graphics - star circlet |
| Stand | Standard detail tag installer |
| Stlay | Rehash layers |
| Strafe | Break text entities at spaces |
| Sun | Print subentity data list direct select |
| Sundog | Turn a cloud inside out |
| Swat | Swap devices open/closed etc. etc. |
| Swx | Swap positions of two sets of entities |
| Sz | Reduce the width of text or attribute |
| T | Trim with snap control |
| Tater | Change text even if it's in a block |
| Tbone | Fillet lines without trimming |
| Tch | Replace multiple text strings |
| Tea | Centre text in a box |
| Tech | Chgtext for text atts attdefs etc. |
| Term | Put an arrowhead on a line |
| Tess | Tesseract (4D cube in 3D) maker |
| Texas | Multiple text/attr/etc. change |
| Tg | Draw polyline and insert cable tag block |
| Tga | Draw polyline and insert cable tag block |
| Tic | Reposition any text in selected circles |
| Tiger | Rejoin lines if they are colinear |
| Tlen | Measure heat trace lines |
| Tray | Draw a tray bend |
| Trayhach | Hatch cable tray |
| Trouble | Diagnostic information |
| Trx | Draw a vertical tray section |
| Tv | Columnize text |
| Txkill | Kill empty text strings |
| Txs | Selective global text height change |
| Txtt | Change text to drawing name |
| Typ | Swap the Typ. tag from side to side |
| Uc | Convert text to uppercase |
| Upright | Rotate text which is at a bad angle |
| Us | Cycle through superimposed entities |
| Uv | Uv=show user sysvars Kuv=empty them |
| Vb | Multiple text fit rejustify |
| Vbcx | Multiple text centre rejustify |
| Vblx | Multiple text left rejustify |
| Vbrx | Multiple text right rejustify |
| Vbml | Multiple text middle left rejustify |
| Vbmr | Multiple text middle right rejustify |
| Vbmx | Multiple text middle rejustify |
| Vess | Vessel maker |
| Vis | Turn invisible attributes on |
| Vl | Vu lock/unlock viewports |
| Vvb | Text line respacer |
| Wir | Jb wiring installer |
| Wireline | Change all line numbers in a block |
| Wiretag | Insert a horizontal or vertical wire tag |
| Wiretagr | Insert horiz. or vert. wire tag on right |
| Wm | Move words from one text line to another |
| Wobble | Make an existing entity into heat trace |
| Wog | Draw wiggly heat trace lines |
| Wolf | Join lines if they are ~ colinear |
| Wormdog | Move attributes |
| Wt | Multi text width change input/ example |
| Xa | Search & replace attdef tags and prompts |
| Xing | Multiple line intersection breaker |
| Xl | Extendo-trim |
| Xnames | Place/update the xref list block |
| Xswitch | XSwitch: go to xref XBack: return |
| Xx | File editor load - shares name with ld |
| Yalf | Freeze a layer by entity selection |
| Yang | Pipe end drawer |
| Zc | Zoom to the extents of what is onscreen |
| Ze | Zoom extents without regen |
| Zing | Multiple directional line intersec break |
| Zlin | Polyline pattern drawer |
| Zp | Zoom previous |
| Zz | File editor load - like xx, diff. file |

**Autolisp by Categories**

1.) Lisp Utilities
2.) Drawing Repair/Reset/Diagnostic
3.) Batch File Editing
4.) Data Extraction/Replacement
5.) General Editing
6.) Line/Polyline Editing
7.) Text/Attribute Editing
8.) Block Editing
9.) Layer/Colour/Linetype Editing
10.) Dimension Editing
11.) Hatch Editing
12.) Miscellaneous Entity Creation
13.) Paper/Model Space/Xrefs
14.) Zoom/View Utilities
15.) Drawing Derotation
16.) Electrical Department Specific
17.) Mechanical Department Specific

**1.) Lisp Utilities**

| | |
|---|---|
| Ld | Lisp loader. |
| Ldr | Load and run a lisp. |
| Lld | Lisp loader with dialog box. |
| Lldr | All of the above. |

**2.) Drawing Repair/Reset/Diagnostic**

| | |
|---|---|
| Beaker | Eradicate dangerously superfluous entities. |
| Bounce | Anytime complete drawing purge, reload same drawing. |
| Cat | Replace all loose attdefs in a drawing with identical text. |
| Cobra | Put entities (including dimensions) back on snap. |
| Descale | Reverse the effects of Scaler.lsp |
| Filthy | Mark the insertion of every entity of a given type. |
| Fint | Find entities by their insertion point. |
| Fist | Find and mark any text in a given style. |
| Flat | Move floating entities back to ground level. |
| Front | Bring an entity to the front. |
| Inf | Prints (to the screen) various useful information about the drawing. |
| Lix | Sets the limits to the extents. |
| Mva | Move the lower left point of the drawing to 0,0. |
| N | Lists all relevant entity data on the command line, no flip screen. |
| Pkill | Locates and optionally removes all point entities from a drawing. |

| | |
|---|---|
| Play | Find out why you can't delete a layer. |
| Shag | Shape locator. |
| Scaler | Scale a drawing up to full size, adjust sysvars. |
| Snake | Put entities (not including dimensions) back on snap. |
| Sname | Replace the top line in the screen menu with the drawing name. |
| Splurge | Anytime complete drawing purge. |
| Sun | Get entity data listing for subentities/entities in xrefs. |
| Vis | Turn invisible attributes on (typically in CadPipe blocks.) |

**3.) Batch File Editing**

| | |
|---|---|
| Blurb | Blur for use in script files (see Blur). |
| Fang | Run Lisp routines on a directory of drawings. |
| Idle | Erase all clouds and Rev triangles. |
| Nobeav | Engineer stamp remover. |
| Onerev | Remove all revs from a drawing, insert a new first one. |
| Ploot | Plot to extents. This must me modified for a particular setup. |
| Repall | Chgtext for every text entity and attribute in a drawing. |
| Replogo | Replace a company logo. |
| Scrim | Write a script file to copy dimvar settings to other drawings. |
| Tball | Rename block definitions in a drawing. Can be edited. |

**4.) Data Extraction/Replacement**

| | |
|---|---|
| Sort | Write text out to a file in the same arrangement as in the drawing. |
| Trout | Extract data from title blocks. |

**5.) General Editing**

| | |
|---|---|
| Bk | Put a break line on the end of two lines. |
| Bock | Clouds selected text and attributes in selected blocks. |
| Dogbite | Trim a space around a circular block. (i.e. an instrument tag) |
| Doglite | Trim a space around a light. |
| Erie | Restratify text. Don’t ever use this. |
| Fluke | Trim all trimmable entities away from text and attributes. |
| Look | Remove everything from within a rectangular area. |
| Sdel | Get selection set, erase only entities of a specified type. |
| Swx | Switch positions of two groups of entities. |

**6.) Line/Polyline Editing**

| | |
|---|---|
| Igloo | Draw a line in Model space from points entered in paper space. |

| | |
|---|---|
| Jall | Make all lines and arcs in the drawing into polylines. |
| Join | Make several entities into one polyline (if possible). |
| Lash | Toggle a polyline between vertex-vertex and overall linetype display. |
| Lic | Change a line, move the end closest to the pick point. |
| Mattress | Put matchlines on the correct layer, rewidth them, move text to the correct position, height, and layer. |
| Melon | Draw lines in inches, scale them to metric. |
| Penguin | Multiple line and polyline crossing locator and breaker. |
| Pj | Join two polylines, lines, and/or arcs even if they don't touch. |
| Pur | Reorient arrowheads and lines, instrument tags and lines, etc. |
| Pw | Set the width of a polyline, line, or arc to 1/2 of Dimscale. |
| Tbone | Fillet lines without trimming them to the fillet ends. |
| Tiger | Line joiner - combines colinear lines into one, ignores others. |
| Wolf | Line joiner – allows more slop than Tiger. |
| Xing | Multiple line crossing locator and breaker. |
| Zing | Multiple line crossing locate and breaker - cut either direction. |
| Zlin | Repeated pattern polyline drawer. |
| Zlp | Zero length polyline remover. |

**7.) Text/Attribute Editing**

| | |
|---|---|
| Abc | Puts text entities in vertical alphabetical order. |
| Above | Create new text line above an existing text line. |
| Ac | Copy attribute values from one block into others. |
| Atro | Middle rejustify an attribute and rotate it 90°. See Wormdog. |
| Attinc | Increment numerical attribute values in selected blocks. |
| Axe | Removes the first or last word from text or adds a new one. |
| Cat | Replace all loose attdefs in a drawing with identical text. |
| Centre | Converts text to centre justified text. |
| Chall | Search and replace for every text entity and attribute in a drawing. |
| Ct | Multiple text lines, prefix, incremented letter, suffix. |
| Ddx | Edit any textlike entity. Aliased as D in the pgp file. |
| Dph | Change text entities from big font styles to standard. |
| Dupe | Draw polylines between duplicate text strings. |
| Eno | Change existing text to incremented numbers, prefix, suffix. |
| Fand | Find and mark text strings in text and attributes. |
| Filk | Change the last letter in each text entity in a selection set. |
| Firk | Search and replace for the first character in text and attributes. |
| Fit | Converts text to fitted text. |
| Fixwit | Find and fix text wider than 0.85 width scale. |
| Ht | Text height changer. Allows example selection. |
| Hvb | Moves text insertion points vertically to lie on same horizontal line. |
| Ladd | Combines two lines of text. |
| Lx | Extract an attribute from a block (as text) without exploding it. |
| Marmot | Compresses width of multiple attributes. |

| | |
|---|---|
| Nt | Incremented numbers with text. |
| Otx | Text at a chosen spacing and angle. Defaults for everything. |
| Pfp | Edit multiple text lines with a real editor. |
| Pid | Adds a prefix or suffix (or both) to a text string. |
| Rtx | Reverse a text string. Handy if you accidentally typed backwards. |
| Sand | Search and destroy text containing a given substring. |
| Scrub | Chgtext for any or all attribute values in selected inserts. |
| Skull | Replace selected att. values with numbers in sequence by position. |
| Spray | Wraps text around entities. |
| Ssq | Sequential text changer. |
| Squab | Globally adjust attribute widths. |
| Strafe | Break text strings at each occurrence of a given number of spaces. |
| Sz | Incrementally reduce the width scale in a single attribute. |
| Tater | Text, attribute, dimension and attdef editor. |
| Tch | Changes one or more text lines to a string, allows example selection. |
| Tea | Centre one or more lines of text in a box. |
| Tech | Chgtext for text, attributes, dimensions and attdefs. |
| Texas | Multiple text, attribute, dimension and attdef editor. |
| Txs | Global text size changer - you specify initial and desired sizes. |
| Uc | Converts text and attributes in selected blocks to upper case. |
| Vbc | Centre rejustify a column of text. |
| Vbl | Left rejustify a column of text. |
| Vbm | Middle rejustify a column of text. |
| Vbml | Middle left rejustify a column of text. |
| Vbmr | Middle right rejustify a column of text. |
| Vbr | Right rejustify a column of text. |
| Vvb | Respace a column of text vertically, default = correct spacing. |
| Wormdog | Attribute rotater. See also Atro. |
| Wm | Move a word to the next line, or the previous one, or to a new line. |

**8.) Block Editing**

| | |
|---|---|
| Blam | Explodes a block and replaces all of its attributes with text. |
| Blek | Extend to a selection set optionally including one or more blocks. |
| Blockout | Wblock all blocks in a drawing (write them to the disk as drawings.) |
| Blora | Vertically respace blocks. |
| Blot | Trim to entities within a block. |
| Blur | Replace all inserts of one block with another, save attribute values. |
| Blurss | Blur only for selected blocks. |
| Chat | Explodes block, permits revisions, reblocks results. |
| Dd | Replace empty attributes with "...", and vice versa. |
| Dent | Search and replace attribute prompts in the block tables. |
| Derot | Derotates blocks and text. |
| Expl | Explodes blocks with different X, Y and Z scale factors. |
| Ezo | Move all block subentities to layer 0, colour byblock. |

| | |
|---|---|
| Fibl | Locates all inserts of a block in the drawing, or all blocks. |
| Fluke | Trim all trimmable entities away from text and attributes. |
| Hoss | Make a drawing of all blocks in the current directory, with names. |
| Kibl | Erase all insertions of a picked block, or all blocks. |
| Lx | Extract an attribute from a block (as text) without exploding it. |
| Pilc | Copy all atts after the first one from one block into another. |
| Prop | Sets the X, Y and Z scales in a number of blocks to 1. |
| Pur | Reorient arrowheads and lines, instrument tags and lines, etc.<br>**Note**: Pur is a good routine and you cretins should use it more. |
| Qeb | Wblocks a picked block. |
| Rat | Make text entities into attdefs. |
| Repo | Reinsert every block in a drawing. |
| Rinse | Reinsert a picked block, update the selected insertion. |
| Scabl | Rescale all of one type of block in a drawing. |
| Scream | Insert a new value into the same attribute in numerous blocks. |
| Scrub | Search and replace one or all attributes in multiple blocks. |
| Sklork | Alphabetize the attribute values in several blocks, vertically by the first attribute value in each. |
| Trz | Trim using block subentities as cutting edges. |
| Urb | Explode blocks, allow for X=-Y, X&Y/=Z, put entities back on the block's layer. |
| Wlay | Wblock each layer in a drawing. |

**9.) Layer/Colour/Linetype Editing**

| | |
|---|---|
| Boff | Turn off all layers except those selected. |
| Bonk | Turn the layer off which contains a selected entity. |
| Bungee | Save the status of all layers for later restoration. |
| Cols | Change colour of entities by selecting one in the colour you want. |
| Colset | Change the current entity creation colour by selecting an entity. |
| Curl | Moves entities to the current layer. |
| Diaper | Layer name search and replace. |
| EzLay | Show each layer sequentially, modify/erase/etc. etc. |
| Ezo | Move all block subentities to layer 0, colour byblock. |
| Flame | Freeze all layers except selected ones. |
| Flay | Freeze all layers except the one containing a selected entity. |
| Layout | Write the names of all layers to a file. |
| Lcb | Move objects to new layer, colour and linetype bylayer. |
| Lch | Moves objects to a new layer by selecting an entity thereon. |
| Lms | Move entities to another layer, make that layer the current one. |
| Lset | Pick an object and thus set the current layer. |
| Lt | Changes linetype by example selection. |
| Lump | Move all entities on selected layers to a destination layer. |
| Mp | Match properties with a pattern entity -Layer, Colour, Linetype. |
| Play | Find out why you can't delete a layer. |
| Slam | Find the layer a block/xref subentity lies on. |

| | |
|---|---|
| Trunc | Fix layer names which are too long to Xref, log results. |
| Wlay | Wblock each layer in a drawing. |
| Yalf | Freeze the layer containing a selected object. |

**10.) Dimension Editing**

| | |
|---|---|
| Dimf | Restore the default value to a dimension. |
| Dimp | Change the length of extension lines on dimensions. |
| Scrim | Copy dimension settings from one drawing to another. |

**11.) Hatch Editing**

| | |
|---|---|
| Chat/Chab | Chat explodes hatch, permits revisions, Chab reblocks them. |
| Cp | Draws a circle in Paper space over one in Model space (so that you can hatch it.) |
| Hatchet | Erase all hatch patterns in a drawing. |

**12.) Miscellaneous Entity Creation**

| | |
|---|---|
| Arrow | Put an arrowhead on the end of a line closest to the pick point. |
| Bk | Put a break line on the end of two lines. |
| Box | A better than usual rectangle drawer. |
| Brac | Bracket maker. |
| Cm | Cloud maker - works either cw or ccw, allows undo while working. |
| Cma | Makes flatter clouds than Cm, for use in crowded areas. |
| Cr | Make an arc into a circle. |
| Cyl | Draw 2D cylinders. Useful for steam engines. See Ofc. |
| Gal | Leaders (angles forced to multiples of 30°) with multiple text lines. |
| Leda | Leaders with multiple text lines. |
| Life | John Conway's game of Life |
| Ofc | 2D sphere maker. See Cyl. |
| Pcb | Draws a curly bracket indicating the ends of two selected lines. |
| Pcc | Puts a curly bracket on the end of lines/polylines/in blocks. |
| Pur | Reorient arrowheads and lines, instrument tags and lines, etc. |
| Rover | Install a wall break symbol. |
| Sundog | Fix clouds which were drawn inside-out. |
| Yang | Pipe end maker. |

**13.) Paper/Model Space/Xrefs**

| | |
|---|---|
| Cp | Draws a circle in Paper space over one in Model space (so that you can hatch it.) |
| Fungus | Repath/Rename Xrefs. Requires editing. |
| Fungoid | Similar to Fungus, but less specialized. Also requires editing. |

| | |
|---|---|
| Igloo | Draw a line in Model space from points entered in paper space. |
| Moss | Move entities from model to paper space. |
| Nautilus | Rename/change Xref layers back to normal. |
| Rex | Repath Xrefs in a number of drawings. |
| Slam | Find the layer a block/xref subentity lies on. |
| Spam | Move entities from paper space into model space. |
| Spat | Fixes entities which are partly in paper and partly in model space. |

**14.) Zoom/View Utilities**

| | |
|---|---|
| Boo | Zoom extents, draw a temporary box around the previous view. |
| Boom | Zoom previous, draw a temporary box around the previous view. |
| Ze | Zoom Extents without a regen. |
| Zvs | Zoom to the virtual screen limits with no regen. |

**15.) Drawing Derotation**

| | |
|---|---|
| Bomb | Rotate a group of text/attdefs around their common centre point. |
| Brat | Derotate attributes in all insertions of a block. |
| Brot2 | Rotate all blocks of one type 180°. |
| Derot | Derotates blocks and text. |

**16.) Electrical Department Specific**

| | |
|---|---|
| Bom1 | Insert a cdf file into a BOM drawing. |
| Btray | Draw a length of cable tray. |
| Bump | Bump up line and wire numbers by a specified amount. |
| Coil | Put a “Coil and tape n spares” block on a line. |
| Hatch | Cable tray/area class hatch installers. |
| Instag | Instag & Pneutag - install an instrument tag with optional leader. |
| Ju | Draw jumpers on terminal blocks, break at intersections with lines. |
| Lna | Copy (and increment) line number text into wire tags. |
| Melon | Draw lines in inches, have them appear in millimetres. |
| Mull | Put cable tags in the right order vertically, using the Mullen-Feenstra system. |
| Sd | I/o schematic text extractor for shutdown keys. |
| Squid | Adjust attribute widths in the monster shutdown key block. |
| TagFlip | Switch single sided wire tags for double and vice versa. |
| Tb | Terminal block drawer. |
| Tga | Draw a cable tag leader line. |
| Trayhach | Hatch a cable tray. |
| Tx | Draw a cable tray end view. |
| Wobble | Entity wobbler. |
| Wog | Draw a heat trace line. |

| | |
|---|---|
| Xhatch | Cable tray crosshatch installer. |

**17.) Mechanical Department Specific**

| | |
|---|---|
| Convoy | Piping data block metric/Imperial and Imperial/metric translator. |
| Godot | Count the non-overlapping weld dots in a drawing |
| Ins | Draw insulation - simple |
| Insu | Draw insulation – slightly more complex |
| 19 | Inserts a pipeline break. |
| Pilc | Copy all atts after the first one from one block into another. |
| Piles | Insert piles with incremented numbers. |
| Pilout | Piles - write the X and Y coordinates of all inserts to a file. |
| Pit | Rotate text into the next isoplane. |
| Pivot | Rotate text, attributes, attdefs into the next isoplane. |
| Slope | Draw an iso plane indicator grating. |

## The Lisp Index

This section is an alphabetical listing of the Lisp files contained in this package, with explanations and notes on various topics and a certain amount of editorial comment.

**~**

Puts a line break symbol (which in case you have missed the point looks like a tilde: ~) on the end of a line or polyline. Uses the same block as the one in the Misc icon menu but can be called from the keyboard.

**19**

Puts a pipe break symbol on two lines and trims them back to the selected point, or chops a section out of the pipe the lines represent and puts a break on each cut end. The break is installed the right way round with respect to the pipe, so time is saved both drawing and thinking.

Note that 19 will crash if used on two lines which have different elevations. This might not seem to be a common problem, but there is an oddly popular piping package which puts entities at strange elevations. If this happens you can fix the entities (or the whole drawing) with Flat.lsp and then use 19.

**Abc**

Puts text entities in alphabetical order vertically.

See also Sklork and Abcat.

**Abcat**

Select a number of blocks, Abcat puts them in alphabetical order vertically by the value of their first attribute. Actually the blocks remain where they were and only the attribute values are shuffled between blocks, so any odd positioning is preserved.

Sklork.lsp is similar but will only allow one type of block to be alphabetized at a time.

**Abg**

Renumber text in sets with suffixes. Someone numbered a set of terminals 11A, 11B, 11G, 12A, 12B, 12G, etc.; there were seventy five triads on each of six pages. Abg was worth writing for just that one use, it is included here in case it crops up again. Select the text, give it the first number, the rest is automatic. If you are dealing with blocks you will have to use Eno and do three passes, one for each letter.

**Above**

Put a new line of text above an existing one, at the correct distance, match the original in style, rotation, etc.

Aliased to Ab.

**Ac**

Sucks attribute values out of one block and spits them in the same order into other blocks. Ac also displays the attribute values on the screen menu while the routine is active so that you can see what you are getting, but since you don't have the screen menu on that won't help you.

Notes:

1. Despite the use of the word "suck", Ac leaves the original block untouched.
2. Ac doesn't care whether the blocks are similar enough for there to be any point to the process, but you might. Computers do make mistakes.

**Acla**

Area class generator. Select a building and Acla will generate an area class for it. Due to the complexity of the problem this is inclined to be a bit flaky, but it can save a significant amount of time.

**Align**

Convert text to aligned text: select the text string, pick two endpoints.

Aligned text is one of the stranger things which Autodesk did: the endpoints are fixed, so the height is adjusted based on the length of the string. In other words it is like fitted text, except that as more letters are added the height is reduced so that the width scale factor can remain the same. The result is that each line is a different height but has the same endpoints.

To the best of my knowledge nobody has ever found a use for it, although if you want a stack of text that looks like a seventeenth century recruiting poster then this may be exactly what you need.

**Asctext**

AutoCAD used to include a program which would import text from a file, here it is. Choose a height, justification, spacing, and various other things: small, quick, easy to use. You can also cut and paste, but this is a bit more elegant.

**Atcase**

A routine for the purist. Select some attdefs, change the prompts to lower case with initial capitals on each word. Possibly not the thing when you are in a hurry, but if you are revamping a block so that inserting it isn't like being yelled at by an illiterate then this may save both time and sanity.

**Attinc**

Increment the numeric value of one or all attribute values in selected blocks. Originally made for incrementing rev numbers in drawing lists, this has proven to be of value in a variety of situations.

**Axe**

Cartesian text editor - add text to the beginning or end of an existing text entity, or remove the first or last word. Choosing among these options is done by picking points respectively to the right or left and above or below the point by which the entity was selected. Those who hear this described are act to dismiss it as the work of an individual not strongly anchored in reality; this is quite possibly true but Axe is also a very useful command.

Unfortunately now that Acad makes use of double clicks the speed at which you can enter points without having it either ignore the second one or activate some useless editing box has sharply declined. In keeping with the Windows paradigm Autodesk has gone from having an amazing array of useful features to having an array of features that can't be turned off. Picture yourself an explorer lost in the Amazon rainforest, each tree and vine blocking your path labeled with "Exciting New Feature" or "Windows Interface Compliant."

See also Pid.lsp.

**Ba**

Pumps, motors and other equipment often come in pairs, labeled A and B. Ba toggles them between A and B - pick a text entity, or an attribute, Ba finds the descriptive letter and changes it A to B or B to A as required.

I stopped doing this sort of thing when I learned to type well enough that finding a given letter on the keyboard took less time than organizing an expedition to the headwaters of the Amazon. Now that we are using little tiny windows dialog boxes it typically takes two or three tries to find that you can't highlight just the one letter you need: typing is slow and tedious again but for different reasons, and anything that saves a little frustration is worthwhile.

**Beaker**

Beaker may be the most used of any program in this package. It erases each invisible and useless entity in a drawing, places a temporary X at its insertion so you can see where they were. This makes files smaller, speeds up loading, and fixes drawings which when zoomed to extents shrink down to postage stamp size in one corner of the screen and continue to so do even after several repetitions.

It also prints totals for destroyed entities by type and overall. It is unusual to run Beaker on a drawing and not have it find something. The record stands at 10,834 invisible entities. Removing them reduced the drawing from 1.6 megabytes to about 70k.

If you use points for anything (i.e. you are heavily into pointillism) you should not run Beaker. Similarly it will destroy the data blocks used in some piping drawings to store information about fittings, so be careful.

**Bean**

Sequentially number terminal blocks. A terminal strip has to be enlarged in the middle of the job so you copy the last twenty blocks to the bottom and then renumber them: 21, 22, 23, 34, 35, etc. Even if your command of the integers is good it is a tedious procedure.

Bean asks you to select some blocks and then renumbers them from top to bottom. It will do any blocks you show it, not just Rocket terminal blocks, so don't include any blocks in your selection that you don't want changed.

Bean also includes the function **Beatn**, which can renumber text in the same way that Bean does blocks.

**Bfg**

Big fonts can be irritating if you don't have the font file. Bfg redefines each text style in a drawing to use the same font without the big font. Typically this doesn't change anything, since big fonts are often attached to a drawing but no spcial characters are used, but it is a good idea to keep an eye on things anyway.

See also Phd.lsp, which removes the big font only from text entities, and Shag.lsp which deals with shape files, which can be just as irritating .

**Bic**

Insert lights: indicate how many rows and columns you want in an area, pick two corners, Bic draws a grid of standard pendant lights in the proper arrangement. If you don't want pendant lights you can use Blurss to swap them for some other type, or for some other type of device. If you're bored you can obliterate the whole building with thousands of lights.

**Bingo**

A drawing rotation utility. Rotate all selected text 180 around their individual centre points. Note that this is not the same as rotating around their insertions which might cause them to land on top of nearby objects.

See also Bomb which does a group around a common centre point.

**Bk**

Draw a break line. Pick two points, Bk puts a standard break line between them, overlapping the ends past the two points by the usual small amount. The resulting polyline has small end segments which wrap back to the insertion points, these allow the break line to be joined to the lines it is breaking in case you want to hatch the whole thing.

**Black**

Colour everything in the drawing white, hence the name. Written for a colleague who was on site and trying to deal with a printer which stubbornly refused to print in black and white. It is also handy for making everything monochrome before saving to another file format for use in word processors. It is not a good idea to save your changes after using this program.

**Blam**

Explodes a block and replaces all of its attributes with text, deleting any which consist solely of " " or "". You are then free to edit them as your heart desires. Great for freeing non-attribute entities trapped in blocks like flies in amber, though I could probably be shot for such a treasonous thought.

Revised: now handles the new justification types (mid-left etc.) and moves any released entities on layer 0 onto the layer on which the block was inserted, that being the layer they would have been displayed as being on. Also changes any entities coloured byblock to bylayer.

**Bldg**

Some time ago I was looking at a mechanical drawing which contained a plan view and four elevations of a building. It would have folded up almost perfectly into a model. Inspired by this I wrote Bldg, which takes all the dimensions and produces a model with tabs ready for gluing. One could then position the elevations on the walls and even put the floor plan in if one had a really accurate double-sided printer. Sadly it was greeted with a lack of enthusiasm that could have stopped a runaway locomotive, and further development has come to a grinding halt.

**Bliz**

Suppose that you were stuck in the Argentine desert over Christmas with nothing to entertain you but a laptop computer. If you were Bing Crosby (alright, Irving Berlin, but it was Bing in the movie) you could write White Christmas, but that having been done you might instead write a block finder (like Fibl) which marked each insertion of a specified block (or all blocks) with a snowflake instead of an X. See also Fibl, the more rational version, or Mold, the much less rational one.

**Blockout**

Wblocks (writes to the disk as independent drawing files) all block definitions within a drawing. Ignores anonymous blocks (hatch patterns, etc.) and at your option either ignores or overwrites existing drawings of the same name. It also spits out a few details of what was done to which blocks.

Notes:

1. Now you can see which blocks are parts of your wiring diagrams and which are little stick men.
2. Wblocking a block definition doesn't (unlike wblocking a group of entities) remove the originals from the drawing.
3. Blockout will wblock all blocks including those which are defined in the drawing but not inserted anywhere.

**Blora**
This is a quick bit of code based on Vvb. Select a group of blocks, it respaces them vertically so that they are a given distance apart.

**Blur**
When I first encountered the lisp program Attredef (which is included with AutoCAD) it seemed like black magic. It still does, because in a curious parallel to my complete lack of success at changing my coworkers into frogs and making gold from lead, I am totally unable to make Attredef work. Although I couldn't find anyone who shared this problem, I decided to write a block replacer so that I could design it to match my own preconceptions. Blur is the result.

- Blur asks for a block name to replace, and for a new block name. In each case you can either type a name or select a block.
- If the new block isn't defined in the drawing it searches the Acad path and - if one is found - offers to use it.
- If there are any default attribute values in the new block definition Blur asks whether to use them in the event that there is no string in the attribute being replaced.
- Then it asks whether to replace the attribute values: By tag name/In the order in which they occur/In order but ignoring empty strings in the original block/Use only the default values *or* Leave all the attributes empty. (Only one of the last two options will be offered, depending on whether the user wished to use the default values.)
- Finally it asks at what scale to insert the blocks. If only <Return> is pressed each block is inserted at the scale of the one it is replacing. The rest is automatic.

Blur is very quick and simple to use, despite this description.

**Blurb**
A variation of Blur for use in script files (See Fang.) You have to edit the file to give it the names to use: the old and new block names are on lines 37 and 38 in the file, 37 is the name of the block you want to replace, 38 is the new block. When you run Blurb it replaces every insertion of the first block with the second one.

Attribute values are extracted from the original block and placed into the attributes of the same name in the new one, if there is no similarity between the attribute names then change the word “Attribute” on line 48 to “Order” and Blurb will reinsert the attribute values in the original order.

Many people won’t use a file if it has to be edited, I am in the process of modifying Blurb so that it uses a very simple external setup file.

**Blurss**
Exactly like Blur, but replaces only selected blocks.

**Bock**
Cloud text and/or attributes in blocks: instead of laboriously drawing a rev cloud with Cm, select the blocks and text you want to cloud, Bock makes a rectangular cloud which encompasses all text and attributes, ignoring everything else. This is especially useful for clouding lines in cable schedules and other charts.

**Bomb**
Rotate text 180° around their common centre point. Pick several text entities and they will rotate as a unit.
Other drawing rotation utilities: Brat, Brot, Derot, Bingo, Lineup.

**Bomin**
Suck a .csv file into BOM blocks. This is located in the Bom Import section under the Blocks pulldown menu. The steps are:

1. Insert the BOM block, it is designed to go in the upper left hand corner of the drawing but you can put it anywhere you like.
2. Explode it.
3. Use Bomin (from the same pulldown) to import a .csv file into the BOM area, and **Lamicoid** to import one into the Lamicoid area.

**Boo**

Lost in your drawing? Too many little detailed areas and you need to zoom out but then won't know where you were before? Boo Zooms to extents and then draws a temporary box around where you were. Sounds silly and saves me from total disorientation and panic about twice a year.

**Bounce**

The purge command (see Splurge) cleans out unreferenced blocks, text styles, and other debris from a drawing, making it smaller, quicker to load, and easier to email. There is another way to purge a drawing: wblock it out to a file. Acad has the ability to wblock a drawing to the same name, leaving a very completely purged file: call –wblock on the command line, and when asked to select entities enter *. Then quit the drawing without saving (which would immediately overwrite the block you just created) and open it again.

Bounce automates this procedure: it wblocks the drawing back to itself and then reopens the file, leaving you exactly where you started but with a cleaner file. Bounce also checks for Xrefs and asks, if any are found, if you want to continue. Bouncing a drawing with xrefs seems to work fine, but at one point it was suggested that this might not be a good idea. It appears that the problem was caused by some other factor, but it might be a good idea to be careful anyway.

See also Splurge and Beaker.

**Box**

A "draw a box by picking the two opposite corners" routine. Displays all four sides of the box as the second corner is dragged into position, or allows input of height and width numerically.

Acad sometimes includes a function something like this, but it seems to change at random with each release, I ditched Box at one point but had to unearth a copy when the latest Acad one became too brilliant to be of any use.

**Brac**

There are a number of bracket drawing programs floating around, most of them awful. I like to think that this one is a superior alternative rather than an addition to the debris. Square brackets, by the way, are ugly, despite assertions that a real man doesn't use anything else. (Sorry, Jack.)

**Bracket**

Draw a bracket on the end of two lines – like Brac, but you select the lines and it does the rest.

**Brat**

Derotate the attributes in all inserts of one block, but leave the block untouched.

**Brot**

Rotate all inserts of one block 180° around their insertion point.

**Btray**

Draw and hatch a straight tray section. Like Box, you can draw the section either by indicating two corners or by indicating one corner and a length and width.

If the segment is too large then the hatch command will fail – it will only draw a certain number of entities in a hatch, and the dot pattern produces more entities than other patterns. In this case you should do smaller areas at a time (use the tray hatch command under the Hatching pulldown), erase the borders and make them all into one.

**Bullet**

Suck data from all of one block to a .csv file: pick one of the blocks, give it a file name, the contents of each block of that type will be extracted to the file, one block per line, with commas separating the values. The data from the highest insertion of the block is read in first, and the rest follow in order of decreasing Y coordinate.

Bear in mind that if there are any values containing commas they will have bad effects on the file – Bullet will work fine, but if you suck the results into a spreadsheet then the values won't be in the right cells.

See also Candy which can reverse the procedure, and Cable which is a specific solution for cable schedules.

**C**

Enhanced copy: turns snap off while you select things to copy and then restores it to the previous state when asking for copy points. Also makes multiple copies the default, it being easier to Return or cancel when you are finished than to turn on multiple when it is too late.

**Cable**

Import a .csv file into a cable schedule. This requires a cable schedule drawing and block, and a matching spreadsheet, both of which can be had from Rocket.

**Candy**

Show Candy a .csv file, select a block, indicate a vertical distance between insertions, Candy will suck the data into blocks one at a time until the file is finished. It doesn't care if there are too many fields for the number of attributes, or if there aren't enough, if it goes off the bottom of the page, or if it sucks a bitmap into a cable schedule – computers are supposed to handle the drudgery for you, not the thinking.

There are a number of programs that can do this with a dedicated .csv file and matching blocks on a properly designed border, fill multiple columns, and make new pages when one is full. Candy can't do that, but it can handle any file and block without modifications to the code. This is in many ways a superior solution, since dedicated programs require people to use the standard spreadsheet and the drawing with the correct border and matching blocks - in other words to follow the standards and pay attention.

See also the matching program Bullet, which sucks data out of blocks into a .csv file. Those from the North of England will understand why the two file names match.

**Cat**

A routine for the purist - Cat finds any free-floating attdefs in a drawing and replaces them with text. A surprising number of drawings have loose attdefs being used instead of text, and they are irritating to edit and hard to locate.

I had considered making Cat, Beaker, Splurge, and Audit run in each drawing as it is opened, but the mayhem that would result if this was done to wrong drawing convinced me otherwise.

If you are one of those who like to explode blocks without creating a variety of naked attdefs, see Blam.

**Centre**

Convert text to centre justified text: select the text entity, hit Return to use the original insertion, pick two points and the text will be placed at a point halfway between them, or pick a point and Return and the text will be centred there.

If you can't be bothered to read all this then just try it out.

See also Vbcx which can centre rejustify a column of text.

**Cf**

Vertically rearrange text in a box – select the text, pick the top and bottom points (or just one point) and input a line spacing, the text is respaced and centred either between the points or on the single point. Very useful if you are working on awful drawings.

**Chall**

Like AutoCAD's original Chgtext, Chall finds and replaces a given text string with another, but it does every attribute and text entity in the drawing. Sometimes one prefers fast to precise. See also Fand.

**Chart**

Make a chart from a text file. See the more elaborate instructions elsewhere in this manual.

**Chat**

Explode and reblock a block. Pick the block, which is exploded. It is best to do this to an unscaled block, but chat will scale it back to 1:1 otherwise. Make any changes you want to the block. If you want to ad new entities then copy them into place, merely moving stuff on top of existing things doesn't work.

Then run Chab, which will reblock the entities, either to the original name or to a new one, which makes a new block.

AutoCAD now includes the ability to modify a block in place, but having tried to use it I find it so convoluted as to be nearly impossible, so I still use Chat/Chab, and include it here.

Chat can by the way explode a block, then explode a block contained in the first one, and allow modification and reblocking of them in order.

**Chgtext**

The original search-and-replace, from the days when there was no other way to modify text short of retyping the whole thing. Later versions had so many options that the basic function tended to get lost in features, so here is the original.

Select text, enter old and new strings, that's it. I have modified it so that it remembers the search string and it tells how many changes were made in how many entities, if there are more changes than entities you should usually have a closer look at the results.

Aliased to ccc in the acad.pgp file.

**Clk**

Draws a clock, set to the correct time. I have used a circle as the face, but a line has been included in the file to insert a block called "clk" if you want to draw one. (You will have to move the semicolon from the beginning of the "insert" line to the beginning of the "circle" line)

People always ask on seeing this if the clock continues to keep time once it is inserted. It doesn't.

**Cm**

Cloud maker: draws revision clouds. There are several of these available, but most need an existing polyline and all require that it be drawn in a specific direction.

Cm draws a polyline while points are entered. A "U" causes the last segment to be removed, and a <Return> closes it and makes it into a cloud. The bulge factor (which controls what portion of a circle is drawn between each two points) is set to 0.9. This is higher than the usual 0.5 or 0.6, if you don't like it you can change the line (setq bulge 0.9) which just coincidentally happens to be the first line of code in the main routine.

Revised - the direction finding algorithm is now much improved, and uses the method used by humans to determine where a point lies in relation to a closed line. If the polyline crosses itself all bets are off, since 1. it is impossible to say in this case which areas are supposed to be which, and 2. only an idiot would do such a thing.

Revised again: now adds extra vertices if the main ones are too far apart, so that one can draw large clouds without having to pick hundreds of points. Thanks for insisting on this feature, Ana.

**Revised** yet again: **Cm** now contains four additional routines. **Cn** draws narrow clouds and functionally duplicates Cma but with one less keystroke. **Cb** draws a box shaped cloud by picking two opposite corners. **Cs** constructs a rectangular cloud enclosing selected entities, and **Cx** makes an existing polyline into a cloud.

See also Bock.lsp.

**Cma**

Flatter cloud maker – essentially the same as Cm, but uses a smaller bulge factor. Cma is good for clouding in congested areas without having the clouds overlap things one might want to see.

**Cobra**

A modified form of Snake, which puts things on snap. Snake will not put a dimension entity on snap, Cobra will, you should generally use snake unless you are very sure what you are doing.

**Coil**

Coil and tape block/text inserter: pick a line, Coil inserts the coiled up wire block and two lines of text: "Coil and tape xxx spares." You have to change the xxx to a number. Actually you don't have to, but it is a bit neater.

**Cols**

Pick some entities, pick one in a colour you like, the first ones will be made to match the latter in that respect. If of course your drawings are coloured by layer rather than by entity you will have to use LCH instead.

Many people like the new change properties dialog box, but I don't as a rule like to give up a third of my screen space to something that is slow, irritating, and almost a textbook example of poor interface design. Using a

computer program should be like driving a sports car – something that is smooth and graceful and in which you can immerse yourself. The Windows interface seems to urge people to obliterate the graphics screen with dozens of superfluous toolbars and dialog boxes until it is more like building cheap plastic models through a hole cut into the lid of a shoebox. Possibly this will pass with time…

**Cr**

Circle Repair - makes an arc back into a circle.

**Crab**

Crab is for aligning leader lines between instrument blocks and tags. Select two circles or blocks, and a line. The line will be moved to be perpendicular to both circles, or if a block was selected to the largest circle in the block. If the block doesn't contain a circle then the insertion point of the block will be used.

Crab is similar to Pur, but requires less input: rather than requiring each of the entities to be selected individually Crab will accept a windowed or crossing selection. You can also select only the line and Crab will search for the nearest block or circle to each end of the line.

**Crypt**

This is a text encryptor. Explaining how it works spoils it to some extent, but it isn't that obvious what is going on.

Crypt asks for a text file and a point. Each line in the file is split into two text strings containing alternate letters interleaved with spaces, then two text entities are made, one for each string, and placed on top of each other starting at the indicated point. The text is drawn in a style based on a monospaced font so that the letters line up perfectly. Then one of the two lines of text is mirrored horizontally about a vertical line passing through the point. When this is printed out it looks like gibberish, but folding the print in half so that the text is on the inside and holding it up to the light allows it to be read again.

**Cspi**

Draws a multiple counterspiral of stars in temporary lines. A redraw will erase it. Utterly useless.

**Curl**

Moves selected entities to the current layer. I have been informed that this is of no use whatsoever, but since I use it, here it is anyway.

**Cyl**

The opportunity to draw steam engines doesn't present itself as often as it once did. One does see side views of cylindrical objects, though, and quite often the lines indicating curvature are unevenly or carelessly placed.

Cyl allows you to draw an edge-on view of a cylinder exactly as it would appear if it had lines parallel to the axis inscribed at regular intervals around the outside. If this doesn't seem worthwhile then you are a philistine. If it doesn't seem intelligible then try it out.

**Datt**

Every drawing should have some indication of what the file is called, where it is located, and when it was last worked on. Datt places a block which displays this information in drawings with known title blocks and updates it whenever the drawing is opened. It has to be set up for a given title block, which we can do for you, and it will save a lot of time and frustration.

**Ddd**

This is a companion to Tater: it places three dots "..." in every empty attribute of every selected block, so that you can use tater or deep to edit them even if there is nothing there. Thus you can edit title and other blocks directly without having to search through the ddatte dialog box and figure out which attribute corresponds to which prompt, something which may be difficult if the prompts "Client," "Client Name," and "Enter Name of Client" are all found in the same block, or worse yet if they are all "X" or sequential numbers.

Running Ddd again on the same blocks removes the dots.

**Ddx**

Ddx prompts one to select text to edit. If the entity picked is text or an attdef it calls Ddedit, if it is a block Ddx calls Ddatte. This sounds like a waste of time, or at least it did to me when MI suggested it, but I now use it for

almost all text type editing. I have taken the additional step of aliasing Ddx to D in Acad.pgp, you can thus edit the most common entities with a single keystroke.

**Deep**
Select text or an attribute, edit it with a dialog box. Deep also allows you to copy the text out of another entity, modify it, and put it into the first one.
See also Ddd.

**Derot**
Derotates blocks. If you really can't figure out what that means, try it out.
Revision note: Derot will now also work with Text. Circles and points you will have to handle on your own.

**Detitle**
Insert a detail tag.

**Dg**
Destroy all groups containing a selected entity. This doesn't affect any of the entities in the groups, it just kills the groups themselves.

**Diaper**
Search & replace layer names. This isn't often useful, but there are times when you want to make a global change. Also contains the utilities **Lance**, which changes all layer names to upper or lower case, and **Bloot**, which prints out the layer tables.

**Dimf**
Dimf restores the default value to a dimension containing an arbitrary string. This is useful in cases where you either want to know what the correct value is and so that you can stretch it and have it automatically update.
Side note: why doesn't the List command print the true value a dimension should display rather than just noting that the displayed one isn't correct?

**Dimp**
Select a dimension entity, pick a point, the nearest extension line to the point will be changed to the same perpendicular distance from the dimension line as the point. In other words, Dimp allows you to change extension lines without using the stretch command and having to work around other entities, which are presumably present, free-floating dimensions being uncommon.
Note: it is usually easier to manipulate dimensions using grips, but every so often one finds a situation in which dimp is the neatest way to do the job. Whether or not this makes it worth remembering I am not sure.

**Dn**
Can't purge a block? Dn asks for a name and prints out the nesting of all occurrences of that block within others. Useful for making prototype drawings where everything has to be exactly right, and for finding out exactly what is in your drawings rather than having just a general idea.

**Dna**
The same idea as Dn, but writes a text file detailing the nesting of every block in the current drawing.

**Dogbite**
Chop around a circular block. Select the block, dogbite finds the biggest circle therein and chops away everything that comes within a reasonable distance of it. Used for cleaning background stuff away from equipment tags and similar blocks.

**Doglite**
Chop around a light block – the same as dogbite, but cuts further away to allow for the little lines coming off the light, which I have always assumed were supposed to be individual rays of light.

**Doon**

Install a conduit down symbol, either on a line or at the intersection of two perpendicular lines. Either pick the two lines, the top one first, or pick one and then indicate the direction of the open side of the block.

**Drawlist**

Suck a .csv file into a drawing list drawing. You can use a script to run a program which extracts the requisite title block data from each file in the set, but such a thing is, as manuals typically say, "beyond the scope of this document."

**Dread**

The File menu has the two entries *Import Redline File* and *Kill Redline Layer*. The first imports an .rml (redline markup) file produced by Voloview into the current drawing as standard entities placed on the "_Markup_" layer. This is very useful for collaborating with someone who doesn't have AutoCAD, as Voloview can be downloaded from the Autodesk website, and the markups can be pulled directly into the drawing they refer to.

Markups can be erased either as the changes are made (you will have to unlock the markup layer) or all at once when you are finished – Dread does this and purges the markup layer. It also contains the utility **RML** which changes the layer colour to Magenta, which is less used than the Red which it is created in) unlocks the layer, and resizes all text to 2.5 x Dimscale, since in some of the examples I have seen the text was too small to read.

**Du**

Write a block updater lisp. Select the block you want to update, the attributes you want to replace, and enter the new values. Du will write a lisp routine (you also have to specify a name) which will make the same changes to other copies of the same block, so that you can repeat the process in other drawings without having to enter all the data again and again.

See also Fang.

**Earth**

Draws a circle centred on the current view and having the same diameter as the Earth. Immensely useful. If you don't try it at least once then you're not human. (Assumes that your drawing units are millimeters, but values for other units are provided in case you want to change this.)

**Ecen**

Horizontally centre a number of entities – select the entities and indicate both sides of the space, they will move so that there is an equal space on either side.

**Elsp**

Draw a spiral: elliptical or circular, increase arithmetically (by a certain distance per revolution) or exponentially (by an increasing distance.)

**Eno**

Change text and attributes to incremented numbers with an optional prefix & suffix as they are selected. Very popular.

**Epi**

Epi is short for epicycle - the curve traced by a point on a circle as it rotates around another circle. Epicycles were invented by Hipparchus around 140 BC to explain the movement of the Solar system around the Earth, since with the exception of the sun and moon nothing seemed to follow a circular path.

A Spirograph is actually a mechanical epicycle generator; Epi does the same thing, but is much more flexible and less prone to skipping a tooth or having the pen come out of the little hole, and you can play with it at work without your co-workers realizing how tenuous your grip on reality really is.

Epi will prompt you for the sizes of the two circles, whether you want the moving circle inside or outside the fixed one, the eccentricity (the distance between the pen and the centre of the moving circle) and various other numbers.

It will also draw the base and rolling circles if you want so that you can see how the finished pattern relates to them.

**Exget**

Ever wonder where in a hatch definition the name and scale are stored so Bhatch can copy them? I didn't really think so, but now that the question has been raised, here is the answer: extended entity data can be attached to any entity and contain a variety of data types. It's like being able to attach invisible attributes to a line or circle. The possibilities are amazing - any entity could tell you when it was created, what block it was part of, who made it, what layer it was supposed to be on... So can Exget do all these things? No. But it can tell you if someone else has done them to an entity. The rest you'll have to wait for.

**Ezlay**

Another small particle in the sediment of attempts to deal with layering. This manual has a whole page on Ezlay which I will not duplicate here.

**Fand**

Text and attribute search. Originally written to see what Chall would do before running it. If a string other than Y or N is entered directly at the "Case sensitive?" prompt, it is taken as a non-case-sensitive search string. AutoCAD now has a built in search and replace function, but I prefer Fand, so it is still included.

See also Chall and Firk.

**Fang**

Writes and runs a script file to run one or more lisp routines on a number of drawings. This allows one to purge drawings, update title blocks, extract data and create index drawings and database files, search and replace, insert blocks, and a variety of other things, all in batches. This is immensely faster than letting a human operator sit and hit the same command sixty times in a row, and much less error prone.

Depending on what you want to do with Fang a certain amount of customization will be required, but none of the procedures are difficult, once set up they are very easy to use. You will have to contact Rocket to get started with this, the results are very worthwhile.

See also Du.

**Fcase**

Convert text to all lower case with initial capitals.

**Fibl**

Find blocks - prompts for a block name, then inserts a temporary X over the insertion point of each occurrence. The marked blocks become the "Previous" selection set in case you want to do something with them once you know where they are. This routine does not do a Zoom All before placing the X's, since many people object to programs which force a regen which may be either unnecessary or very time consuming, but it is a good idea to have the entire drawing on screen before searching it (unless you want to count the blocks but don't care where they are.)

Revision note: If no block name is specified Fibl will now mark all blocks in the drawing.

Second revision note: Fibl now allows the user to set the size of the marker X's, so that they don't overflow the screen if only a small area is displayed.

Third revision note: Fibl now allows either entry of a block name or selection of an example block. As a result it now takes one extra <Enter> to get to the "Mark all blocks" stage, but this doesn't seem too awful.

**Fifty**

Schematic drawings with line numbers present a special problem if they are repositioned in the set – if the line numbers change then every wire tag, terminal number, and relay or other device which depends on them must also be renamed.

Fifty replaces the original first line number with the new one in the selected entities (or everything), then increments the two numbers, and repeats the whole process fifty times, hence the name.

It also puts a coloured temporary marker at the insertion of each thing it changes and changes the colour with each number, so that you can see if anything goes wrong – if the colours on a line are all different or if they show up wildly all over the place then you may have a problem.

Does this mean that Fifty can take a tedious process that takes hours and run through it in a few minutes? Yes.

**Filk**

Search & replace the last character in all selected text and blocks. More useful than you might think.

See also Firk, which does first characters.

**Filo**

A simple routine which takes a set of entities and writes their data out to a text file in the format returned by entget.

"But what use is this?" you ask. Well...hmmm...you could make a printout of the resulting file and leave it on your desk to baffle the unwary. Or make a program which takes the data and makes a script file to make the drawing as you watch - this might make an interesting demo.

The only practical thought which suggests itself is to write selected entity data out to a file and then use it later to restore the entities to their condition at the time the routine was run - sort of an entity specific Undo. The entities would have to be accessed using handles since there is no way to make a string read from a file back into an ename, but this would provide a way to undo changes to entities even several drawings later. (If they had been deleted they could be reinstalled with the entmake function.) One could also edit the entity data with a text editor and update the entities…

**Filthy**

Fibl's twin (fraternal): finds and marks the insertion point of any entities of a given type in the entire drawing, or optionally everything in a drawing. Written to locate invisible entities, but probably has many other uses. Maybe. You can quite often find interesting patterns by zooming 0.1x and then running filthy.

**Fint**

Find entities by insertion point. When you have run Filthy and found several hundred invisible entities way out in space you can pick their insertion points with Fint, indicate how far off you may be, and then see what they are from the resulting entity data listing.

Fint also offers to erase the things it finds, but this is generally not a good idea.

See also Beaker, which erases a large variety of useless and invisible entities.

**Firk**

When supplied with a character to find and one to replace it with, Firk changes the former to the latter every time it occurs in the selected entities (text or attributes) if it is the first character in the string or the first one after the %%u which underlines a string. You will probably either find this very useful or completely pointless.

See also Filk and Chall.

**Fist**

Prints a list of all text styles in the drawing, which fonts they use, which blocks contain text or attributes or sub-blocks which contain text etc., which linetypes and dimstyles contain which fonts. Useful for the purist and for odd drawings with too many text styles.

See also Shag, which does the same thing but with shapes, which are much more irritating.

**Flame**

Freezes all layers in a drawing except those containing selected entities. If nothing is selected Flame will thaw all layers and tell you how many there are, which is sometimes surprising.

See also Flay.lsp and Yalf.lsp.

**Flat**

Entities in drawings sometimes do not behave properly - typically lines have no intersection even though they appear to cross. This is often because someone has drawn part of the drawing at an elevation other than zero, which is understandable if the drawing is a 3D version of a building, less so if it is a wiring schematic.

Flat will move any chosen entities and subentities back to ground level. It is also useful for squashing drawings done in 3D if the urge overcomes you.

**Flay**

Pick an entity on a layer, Flay freezes all the layers but that one. It thaws all layers first, since it is difficult to select something on a frozen layer.

See also Ezlay.

**Flip**

Swap an open device for the matching closed on in an I/O schematic, or vice versa. Also swaps wire tags between single and double sided in case the sight of a tag hanging off into the air around a terminal strikes you as being wrong on some primal level.

See also Typ, which does the (Typ.) string in detail tags.

**Flood**

Floodlight maker. Pick a centre point and a direction, indicate how many lights you want (the maximum is 4) on the same standard. The resulting entity is a single rather convoluted but neat looking polyline.

**Fluke**

Fluke allows one to trim everything away from around a line of text, or from around attributes within a selected block, or around the text in an associative dimension.

Easy enough. Now the small print:

1. If a block insertion is selected, Fluke will only cut around visible attributes, because otherwise there would be holes chopped in nearby entities for no apparent reason.
2. Fluke will not cut entities which can't be trimmed - blocks, other text, hatching, etc.
3. Fluke uses the same clearance that is left around text by the Hatch command - 0.25 x text height. Most of my colleagues didn't feel that this large enough, so I made several variants which differed only in the cut distance. This soon became ridiculous (a condition we already encounter often enough) so I rewrote Fluke. Each time you pick the same entity the cut distance becomes greater by 0.25 x height: after the second pick it is 0.5x, after the third it is 0.75x, etc. Pick a different text entity and it returns to 0.25x. (Picking something that wasn't either text or a block doesn't change anything.)
4. If you want to change the initial cut distance you can alter the initial setting of the variable Cutdis.
5. I can't remember where the name came from.

**Front**

Bring an entity to the front. AutoCAD now has a feature which allows you to change the display order, and it is very powerful, but if you just want quick and easy Front is better: pick an entity, it is brought to the front of the drawing database. Handy for reordering attributes, just pick them in the order you want them to show up in the ddatte box.

Also contains **Fronts**, which moves a whole selection set to the front – this is not as precise but allows wholesale reordering.

**Get**

Prints the data for a selected entity to the screen. The default is the entity after the last one selected, so a polyline or block can be stepped through one part at a time. (See Chg.lsp to actually edit entities in this manner.)

**Getq**

Good programming practice dictates that variables be declared local wherever possible, but going through the routine to find the setqs is tedious and error prone.

Getq reads a file, extracts all the variable names, and writes them to a file named after the original file but with the extension ".var". It also puts the routine name in front, so you just have to append them to the lisp file and move them to the Defun statement.

Aside from the input file name, Getq only asks one question: "Repeat variables after Defun?" If you answer Yes, any variables which are used in more than one routine within the file will be written to the .var file the first time they occur in each routine. If you answer No then a variable will only be extracted the first time it is found in the whole file. Caveats:

1. Getq doesn't search for arguments to place before the "/", so you will have to do that yourself, and if you have any variables which are supposed to be global you will have to delete them from the list.
2. I don't generally use the multiple (setq a 1 b 2 3 4 pi 5) format - I find it less clear, less attractive, and no faster than (setq a 1) (setq b 2) - and Getq won't pick up variables set in this way.

**Gnd**

Put a ground grid around a building – pick both corners of the building, the rest is automatic. If you like this you can thank Jonathon Wu, who wrote the first really weird prototype version.

**Gouge**

Writes entity data to a file like Filo.lsp, but does block (and hatch) definitions, which includes all attributes, constant and otherwise, and all non-attribute subentities - text, lines, that sort of thing. The stuff List doesn't see.

**Gr**

List the groups to which an entity belongs.

Groups seem to be downplayed in the latest versions or AutoCAD, but they are quite useful, like blocks that can be turned on and off. The group commands are located in the Groups flyout under the Assist pulldown menu.

**Grak**.

Explode all groups. I have come across drawings with as many as 45,000 groups (there can be empty groups and one entity can belong to several groups) which makes them very large and very slow to work on. Erasing or modifying an entity does the same thing to every entity in any group to which it belongs, so if they haven't been carefully made they can make life very difficult.

Exploding a group does not affect the entities it contained in any way.

**Gremp**

Destroy all empty groups. Typically programs which make groups tend to create several times as many empty ones as ones which contain something.

**Ground**

Insert a ground symbol on the end of a line. You can also put one off by itself, but wiring being the sort of integrated whole that it is this may not be all that useful.

**Grpid**

Count and list all groups, and highlight anything which belongs to a group.

**Gyc**

Centre selected entities in a box. Like Tea, but works with the majority of entity types. Good for putting client logos in boxes in the title block.

**Halo**

Every drawing has a title block, and these have attributes. (If your drawings don't have proper title blocks then go and make one up before you read any further.)

There are certain areas of the title block which are the same on every drawing in a job, and someone has to go through each drawing and update them. This often happens at the end of the job when there is a scramble to change wrong information, pick up ones which were missed, and update everything based on a change in title, LSD, or plant name.

Halo reads a text file which tells is which text belongs in which fields, sees if it has a title block which it knows about, and if so updates the drawing. It runs each time you open a drawing, so once the information is in the file every drawing you open will have the correct common title text. You save on time and labour and get a drawing set with no idiotic mistakes in the titles..

What if you don't want to update the title blocks on a job? Then don't make a text file and Halo won't change anything.

Halo requires some setup, which we can do for you, or you can wait for the next update which will run entirely from an external configuration file. Very highly recommended.

**Hd**

Disassociate a hatch. Originally written so that hatches could be quickly disassociated and then relayered without changing everything they were attached to. You can also uncheck the *associative hatch* box in the *selection modes* area of the *selection* tab of the Options dialog box, which allows you to select a hatch without selecting its boundary entities but still updates it if they are repositioned or otherwise changed.

Note: the Pickstyle variable is now automatically reset so that hatch boundaries aren't selected along with the hatch, but Hd is still included.

**Hoss**

Imports all of the drawings in a directory into the current one and places them in a grid pattern specified by the user. Also places the name beneath each one and optionally inserts the tag names into the attributes and marks the insertion points.

Useful if you are expected to know which blocks you have and what they look like. Also popular with management, who can look at pages of blocks and make noises which seem to indicate that they approve and then put them in a binder labeled standards where they will remain untouched until our civilization is excavated by archaeologists several thousand years from now.

**Ht**

Text height changer. Select one or more entities, change them to a chosen height. Also works on fixed height text, and will ignore non-text entities so you can window an area and change only the text. (Revised: now allows picking an entity of the desired height.) See also Wt.lsp which does the same thing with text width scale factors.

**Hvb**

Horizontal Venetian Blind. Rearranges text so that it's all at the same level vertically, without moving it horizontally. (Yes, it should have been called Vvb.lsp, but the name was already taken. You can rename it if you really want.) Pick the text, specify a point, and choose whether you want to align it by the text insertion point or on the left end of the text line (or what would be the left end if it were horizontal.)

**Idle**

Remove all rev clouds and blocks from a drawing. Can be used with Fang to de-cloud a whole set of drawings before the next issue.

**Igloo**

You can object snap to an entity in model space from in paper space, but not vice versa. Igloo lets you draw a line in paper space and then moves it into model space where you can snap onto it. Very useful for lining stuff up between the two spaces, although it might be better if this deficiency was corrected in the next release of Acad.

**Inf**

Prints (to the screen) various useful information about the drawing, punctuation, wildlife, etc. Tries to tell you not what you want to know, but what you need to know, in contrast to Windows which tells you what you don't need to know and Macs which won't tell you anything.

**Ins** & **Insu**

These are both insulation makers, for the standard representation of fibreglass.

Ins takes a width and two endpoints and draws the insulation. The resulting entity is a single polyline which is the correct width but which overlaps the ends of the space indicated so as to avoid having to start in the middle of an arc. In other words you will probably have to do some trimming.

Insu is essentially the same but it places a box around the required area and trims off anything which falls outside it. This usually results in a small polyline arc which is separated from the main entity, so this is joined back on with a straight segment.

**Intbreak**

Break a line at two intersections. Used for clearing lines out of blocks which cross them. The ability to trim to entities within a block is wonderful, but small blocks often contain text which prevents one from hitting the line it is sitting on top of, Intbreak isn't bothered by this at all.

**Irg**

You are working on a 4.5 megabyte drawing on your XT. The entities displayed on the screen - specifically the arcs and circles - are a bit lumpy and you would like to reassure yourself that they are correctly drawn. Unfortunately you don't have two weeks to spare while the drawing regenerates. If you had a copy of Irg you could select just the entities you wanted to display correctly and cut the required time by half. Maybe more if you have a faster computer. Irg is short for Independent Regen, by the way.

There appears to be a bug in Acad2000 which causes attdefs to display in the wrong place when you open a block directly, Irg will also fix these.

**Join**

Making a group of lines into a polyline has always been slightly more involved than I like - pick a line, Do you wish to make it into a pline? Join, select entities to join, etc.

Join asks for a selection set, sees if anything in it can be made into a polyline, and adds the rest if possible. This isn't a very complex routine, and unlike Pj it won't make the ends touch if they don't already, but it saves a lot of time and tedium.

**Jumper**

Install a jumper on terminal blocks – pick two points, pick a horizontal location, Jumper draws the jumpers and breaks them around existing lines.

**Junc**

Make a junction box. You probably have prototype drawings, but sometimes it is easier to just make one. Asks if you want analog or discrete and how many terminals, makes the terminal strip, puts in wiring and shields and default text, throws a box around the whole thing to represent the jb. Is quicker and more accurate than constructing jbs by hand, and at least gives the impression that you have some type of standards.

**Kibl**

Pick a block, Kibl will erase all insertions of that block in the drawing. Hit a return instead and Kibl will erase every block in the drawing. (It asks for confirmation first in case you didn't want to do anything that drastic.)

**Klab**

This must have been written for a purpose, but I can't remember why. It was never used. This may seem odd, but I have several times written software to be used as either a threat or a bargaining tool and which was never used. Anyone who has ever witnessed a professional demonstration of any program will have seen magical features which left everyone in awe and turned out to be of absolutely no use when the program was purchased..

Klab erases all blocks contained in an internal list, which is easy to modify, so it could be run on startup or from a batch with Fang.

**Klat**

Turn off a selected attribute – select it and it is invisible. I was going to make it so that you could toggle them back on, but it proved difficult to select them once they were invisible. Vis will do this, though.

**Kx**

Displays a dialog box with the name of each application having extended entity data in the drawing. Selecting an application name and then pressing the *Kill* button removes all xdata for that application from the drawing.

See also Xmark.lsp.

**Ladd**

Combines two lines of text. Pick the base line, pick the second, this will add the second one onto the end of the first; also it removes any excess spaces from between the two. Designed for those situations where Wm.lsp might be a bit tedious.

**Lash**

Release 12 added the capacity to display a polyline with a linetype other than continuous either with the pattern starting and ending at each vertex (which generally meant that the segments were too short to show the pattern) or displayed continuously over the whole thing. The display mode is set when the polyline is drawn and can be changed later on, but the method usually escapes me. Lash toggles a polyline between the two settings.

**Lcb**

Like Lch.lsp, moves entities to a layer indicated by selecting something on it, but this one changes the colour and linetype to bylayer. Note that if your entity is on the right layer but the wrong colour you can pick it and then use it again to indicate the destination layer.

**Lch**

Select objects, pick an object on the desired layer, move the former to coincide in that respect with the latter.

**Ld**

Load a lisp. All of the lisp files included with Rocketcad are loaded automatically, but you may want to write your own or get some from other sources. The standard load procedure is (load "lispname"), but it is easier just to type Ld and then the lisp name.

Most people prefer Ldr, but Ld is included because I use it. I would include a note on buying only software made by people who use what they write, but of course this is self-limiting – once software becomes profitable, the process that prompted its creation is usually either relegated to a secondary role or dropped. Which may be why software so often succumbs to feature-itis and random changes: it's not that they're trying to force an upgrade, they've just lost sight of what they were trying to do in the first place.

**Ldr**

The next evolutionary step past Ld.lsp - loads a lisp and then runs it. One would think that the ability to have one lisp run another would be very useful, but beyond this not a lot comes to mind. I spent some time writing a routine which allowed one to embed lisp in a block and then to run the routine by selecting it, but while this was fascinating the world did not beat a path to my door bearing boxes full of unwanted mice.

**Leda**

Associative leaders were a great idea, but impossible to set up exactly the way everyone wanted them, which led to some really awful drafting. (Or maybe there was awful drafting going on already…) Leda uses the associative leader entity but dead ends it and puts text on the end. You can still drag the leader and have the arrowhead realign itself, and you have a decent looking leader. Quick and easy to use, handles multiple text lines, doesn't require any special setup.

**Left**

Convert text to left justified text; specify a new insertion point or by default use the original one.

**Lineup**

Make sure composite lines containing text are upright so that the text doesn't display upside down. If the ten end of the line is to the right of the eleven end then the line is rotated 180° and the text is miraculously upright. Probably Autodesk will make this the default display mode in some future release.

**Lix**

Sets the limits to the extents. Useful if you find the grid occupying only a postage stamp sized area of your screen, if your Zoom All produces mostly empty space, or if plotting to limits produces a handy wallet-sized picture.

Piping people will foam at the mouth if they see this, because they like to leave lumps of excess piping out in space and then print to limits after setting them to match the extents of the title block. They are also very prone to the idea that the universe will cease to exist if Electrical doesn't plot in the same way that piping does, and in fact if everything isn't done their way. It is usually not a good idea to go along with them.

**Lld**

Lisp loader, similar to Ld, but uses the file selection dialog. Most of the routines listed here are already loaded, but sometimes one needs to go looking.

**Lms**

Move stuff to a layer, make that layer current.

**Look**

Pick two opposite corners of a rectangle, Look will trim and erase everything trimmable or erasable out of it.

**Lox**

Similar to Filo.lsp - writes entity data listings to a file, but this one only writes the sublists you request. This makes the resulting file a lot easier to decipher, since the screen isn't clogged with justification types and extrusion directions.

If you don't understand that then you probably don't need it.

**Lset**

Pick an object and thus set the current layer. Also tells you what layer you *were* on and what layer you *are* on. You could just pay attention to the status line, but your mind might be having coffee or you just might have my fondness for commands that tell you what you have just done as opposed to what you think you have just done.

**Lt**

Change linetype by object selection. Select some entities, pick one of the desired linetype, the previous ones will be changed to match. The default linetype (continuous) is not always explicitly contained in the entity database, so Lt adds that entry to the entity data list if it can't find it, and then modifies it as required. All of the books claim that this is impossible, but it does work. Then again, maybe I imagined the whole thing.

**Ltr**

Accepts a file name. If the file exists, writes the last line to the status line. It then writes all text typed in into the file, starting a new line with each Return. A Return with no input terminates the procedure, and the filename is saved as the default so that you can pick up where you left off. This might be useful for writing batch files or something, but mostly it was designed to allow me to write letters to a friend explaining what a tyrant my boss was for not letting me write letters to my friends, while appearing to be working in AutoCAD.

**Lump**

Dumps everything on one or more layers onto another layer. Pick entities on layers to move, then something on the destination layer, and answer "Yes" to the request for confirmation. Very useful for cutting down the number of layers in a drawing. Also capable of wreaking a fair amount of havoc, so don't use this if you're accustomed to being protected from your own insanity.

See also Ezlay and RL.lsp, which does the same thing but includes subentities.

**Lx**

Pull an attribute out of a block as text – select the attribute and it will appear on your cursor as text, ready to place. Much quicker than making text up from scratch, and also allows you to save a line of text out of a block without using Blam and laboriously erasing everything you don't need. Also leaves you with the vague impression that you've just done something impossible.

**Mach**

Search and replace attributes, select them by picking each one or windowing several – allows you to treat attributes as though they were blocks. The attedit command (called from the command line with a leading dash: -attedit) can do a lot of amazing things, but the syntax is odd enough that it is worth building into little macros like this one.

**Marmot**

Multi-attribute squash – originally made for use with monster shutdown key blocks. Pick several attributes, input a width scale factor, they will all be changed to that width scale. Saves a little time and a lot of irritation. MI, who complains nonstop about idiotic lisp names, contributed both the idea and the name.

**Mattress**

Matchlines: some clients require them and some don't. If they are used then they should have text indicating where they are in space and which drawing they connect to. The text should be of a standard height and offset a standard distance from the line, which should be a polyline of a given thickness, linetype, and colour.

This is all quite tedious to set up, so Mattress makes everything right if you pick the text and the line.

**Melon**

It is desirable for anyone who lives in the modern world to be familiar with both the Imperial and Metric systems of measurement – those who say that one must replace the other for religious reasons are completely missing the point.

Suppose that you need a line on a drawing done in millimetres but the measurements are in inches? Melon takes a length in inches and a direction and draws the line in mm. This isn't a perfect solution, and for complex objects it is better to draw the whole thing assuming that one unit is an inch and then scale everything up by 25.4. Generally one mixes both methods.

**Mex**

Mtext is one of those things which came really close to being a good idea, but whenever I have tried to use it I have become frustrated and exploded it. People who use programs based not on whether they work but on whether they really should often do everything in mtext, even single words. Unfortunately it is appallingly awful for paragraphs, which is really the only place where it should even be considered.

Mex brutally explodes all of it – nothing looks different, but you can suddenly get things done.

**Middle**

Convert text to middle justified text. Fairly self explanatory but if it really baffles you get an adult to take a look. See also Align, Left, Fit, and Right.

Revised: If a <Return> is entered instead of a new centre point the text will be rejustified to middle but will not move at all.

**Min**

The original versions of Acad included the multiple insert block: like a regular block, but instead of just one you could have it display several rows and columns of the same thing. The attributes in a minserted block were disabled, and it couldn't be exploded.

Min allows you to change a normal block into a minserted block. This is interesting but useless, but if you give the block a number of columns or rows and no distance it will be minserted and thus can't be exploded. Under R14 the Zero function would make it back into a normal block, but as of 2000 this no longer works: you have a block which can't be exploded, albeit one with no attributes.

**Miss**

A more interesting explode command. Try this once when nobody is watching.

**Mold**

Mark every entity in the drawing with a tuft of mold. Keep a straight face and say that you don't know what is happening, but that maybe you need a new computer. A redraw will make everything clean again.

**Moss**

Suck stuff from model into paper space.

See also Igloo and Spam.

**Mp**

Match properties (layer, colour, and linetype) with those of a selected entity. The built in command gives you more control, this is quicker.

**Mull**

It used to be common to do a cross section of a trench or cable tray showing the order in which the cables were to be laid. Surprisingly these were not always treated with the respect that the designer would have liked. A more modern approach is to label the cables by type: main feeders in numerical order, followed by power, then control, and so forth. This makes life easier for the electricians, doesn't waste drafting time drawing stuff that isn't ever going to be built that way, and makes it much easier to keep track of the cables since they are always in the same order.

Mull does this automatically: pick a stack of tags and they will be reordered vertically.

**Mullet**

The same as Mull, but reorders text instead of block values, some older drawings having text instead of blocks for cable tags. Mullet has a better warning for tags with duplicate strings.

**Mva**

Caution: this program does not work correctly with R14 and up: the value of the Extmin system variable is typically off by 0.01 units in the last few releases.

Moves the lower left point of the drawing to 0,0. If you have macros to insert a block or edit text at a specific point, this might come in handy. Like many of these, it will only be of use to those who share their drawings with the careless or unbalanced.

Later note: schematics and drawings of things which don't have a definite geographic location should usually be at 0,0. Ones which are located relative to some part of physical reality should be located at the appropriate point in space so that things can be inserted using their real coordinates and come out in the right place. This makes drawing faster and more accurate and makes checking lengths and sizes more reliable. Also dumping several drawings into one file to check for alignment becomes trivial, and copying things from one drawing to another is much less stressful.

**Mx**

The file Sample.pgp suggests building a menu editing routine into your menu so that you can add sudden inspirations from within the current drawing and test the results immediately. Unfortunately not all drawings use the same menu, so MX finds the name of the current menu file and loads it into a text editor for you, then reloads the menu afterwards.

Note: If you are working without a network but are editing drawings which look for the menu on a network drive, you can simulate a drive with the line Subst k: c:\gdrive in Autoexec.bat and Lastdrive=k: in Config.sys. Then you can add whatever path you need to c:\gdrive and your drawings will be happy.

**N**

List entity information in a condensed form on the command line, saves time and doesn't require flipping to the text screen.

**Nail**

Make a set of notes. Pick a point, Nail puts in the Notes: header and then starts a new line below it, numbered 1. It adds a new line after each Return, if Return is pressed again then it starts a new note with the next number in sequence, if nothing is entered (i.e. a third Return) then it erases the new note number and exits.

You can add a new line to an existing note by either clicking on the last number (if the last note is only one line) or on the last line of text, in which case you will be asked what the note number is.

**Nepo**

A strange point connector. I have no idea what this was for or what it is doing in here.

**Next & Nxt**

Each time you enter *Next* you will go to the next drawing in the directory, so that you can make changes (i.e. changes which require judgment, so that Fang can't be set up to do them) to each drawing without the bother of typing *open*, finding the name, etc. etc.

Next turns on Sdi mode (setvar SDI to 1) so that after editing a hundred drawings they won't all still be open. When you are finished you can set it back to 0 if you want to be able to open several drawings at once, just type SDI, then 0.

*Nxt* is exactly the same as Next except that Next saves the drawing before going to the next one, and Nxt does not. Both programs read and write the same drawing name file, so you can use both in the same editing session.

**Nn**

List a subset of subentity information in a dialog box. Mainly intended for attributes and text.

See Sun, which returns a complete subentity data list but is more difficult to interpret.

**Nt**

Incremented numbers with text. I have never used this, or even seen a vague theoretical need for it, but my old buddy MI felt that it would be better for me to spend several hours writing and debugging it than for him to spend several minutes working without it. It's interesting to watch, though.

**Numf**

Sometimes a program suggests itself in the absence of a use - for instance One.lsp, which can convert a number (i.e. 111) to the English equivalent (one hundred and eleven).

Sometimes a program suggests an equally useless (if more productive) program using itself as a base. Specifically, it occurred to me that I could modify One.lsp to convert a number of numbers to the English format and then to write the results to a file.

Numf takes a starting and a final number and writes them and each one in between to a text file. It may appear that this is a pointless achievement - certainly it did to my coworkers. However given a reasonably fast computer, four unattended hours and a script file (num.scr, included) I was able to write the numbers from one to one million to four text files totaling nearly 65 megabytes. (Why four? Because, amazingly, they each Zip down to less than 1.2 megabytes and will fit on a floppy.) So: amaze your friends and relatives, prove that counting really works the way they told you in school, print it out and pretend it's a science project.

Update note: a more powerful computer can write 1 – 10,000,000 in about fifteen minutes, or up to two hundred million in a couple of days. It is advisable to do this in text files as ten million numbers occupies between seven hundred megabytes and one gigabyte, depending on where they are in the series; no operating system with which I am familiar can deal with a file larger than 2.5 gigabytes.

A CD will hold up to two hundred million, zipped. Please stay tuned for further updates.

**Nxlay**

Similar to Ezlay, but instead of displaying each layer it merely freezes all of them except for the one being displayed, freezing the current one and thawing the next each time it is called. Brutal and cantankerous but sometimes very useful, try Ezlay before you resort to this.

**Ofc**

2D sphere maker, for those times when you want your drawings to look like something from the last century. Make that the one before the last one. I did once use this for something productive, but it is included strictly because it goes well with Cyl.

**Pang**

Screen shaker. Originally part of another program, but it seemed to need to be on its own.

**Panic**

Electrical distribution panel (a.k.a. Lighting Panel) maker.

This is found under the Electrical pulldown, so you don't have to remember the name. It makes single and three phase panels in the standard sizes, if you prefer you can specify your own size, although you may not be able to find anyone willing to sell you a two circuit panel. (Pause here while about six people say "I once worked on a project where we had one of those…") Odd numbers of circuits are not allowed, but since the resulting entity is almost entirely composed of loose lines and text you can edit it to your heart's content.

**Pc**

Draw a circle in paper space over one in model space. Good for drawing hatching in PS when the drawing is in MS, and other things, none of which come readily to mind.

**Penguin**

Penguin breaks lines where they intersect, but will not put a break at one end of a line, since if it did it would cut a polyline at each vertex.

See also Xing.lsp and Zing.lsp, which do the same thing but only to lines…actually I am not sure why they are still included, but they are, so there must be a reason.

**Pfp**

My boss (actually the King) showed me a page he had copied out of the Lisp manual with the special commands for the ADE/Lisp text editor (the name escapes me). Feeling that they were a trifle awkward in the absence of a computer with foot pedals I asked why he didn't use Ed.lsp which allows the use of a real text editor. He replied that Ed can only handle one line of text at a time.

An unusual attack of professional pride forced me to put aside work on the solar powered tanning bed and write PFP (Paragraph-File-Paragraph) which handles multiple lines of text, creates and erases lines as needed, and uses a real text editor.

This is handy if you want to create a lot of text using features only found in an editor; Pfp creates extra lines of text if they are needed.

**Phd**

Change any text which uses a style based on a big font style to an ordinary style.

Why are drawings made with fonts that are not available outside the company which created them? The assumption seems to be that although drawings are done for a client and then given to the client, the client will never open them or allow anyone else to work on them. Big font styles are pretty much the same idea, but the one which prompted this program didn't come from the company who created the font, and none of the text used the big font features.

If you want a font which you can freely distribute, with features you can actually use, see Rocket.shx.

**Phdel**

Delete all frozen layers. Very useful for cleaning up mechanical drawings, deleting extraneous stuff after binding an ex-Xref, etc.

**Pid**

Select text and add a prefix and/or suffix. Don't select anything and you will be asked for a new prefix and suffix, an empty space kills an existing one. Works on text only. Originally written to fix errant PIDs, thus the name.

See also Eno.

**Pivot**

Pick text, an attdef, an attribute, or text within a block, and Pivot rotates it into the next isoplane. If it isn't isometric it will be made so.

**Piranha**

This is on the File pulldown under *Vastly More Drawings.* Piranha keeps track of the last five hundred drawings you have opened and displays them in a dialog box, clicking on a name opens the drawing.

Files are added to the top of the dialog box, so that the most recently opened ones come first. If a file is opened which is already in the dialog it is moved to the top.

Piranha is thus useful because it allows you to return to a recently opened drawing without diving through incredible numbers of sedimentary layers of directories, hoping to find the drawing before it fossilizes under the weight of your unnecessarily byzantine directory structure. It is also useful for getting a general idea of what you worked on and in what order.

The file names are stored in the main Acad directory in the file Lastfile. This can be edited with a text editor, and comments can be added, prefaced with a semicolon. One could theoretically add a list of drawing names belonging to a given project to the file so that those working on the project could easily access them.

**Pj**

Join two noncontinuous polylines by drawing a connecting segment. Usually it is better to move the end of one to the end of the other, for which grips can be very handy, but sometimes fixing minute misalignment errors is too time-consuming.

**Play**

Files sometimes have layers that can't be purged. Usually there is either an entity you have missed on the layer (possibly an empty text string) or a block on a different layer with a subentity on the stuck one.

Play checks the layer for empty strings and other entities, marks them, and offers to erase them. Then it searches the block table for blocks with subentities on the layer in question, and prints a list of their names. At this point you can insert a copy of the block, explode it, move the problem entity to another layer and reblock the entities, redefining the block. HAT.LSP will make this process relatively painless, or you can incorporate the code into Play and make the whole thing automatic. See also Dn.lsp. Revised: there are a number of lisp routines which can change the layer on which an attribute is placed without affecting the block itself, so that the attribute is on a layer other than that specified in the block tables. Play now checks for this condition and marks the edited blocks.

**Poxx**

One of the major ways of killing time - at least at the schools I attended - was doodling. The form that leaps to mind was a square which was repeatedly divided by drawing a line from one corner to a point about a quarter of the way along the opposite side, and continuing toward the centre until you wore a hole in the paper or the class was over.

This routine does the same thing. It probably won't amuse you more than two or three times, but it's worth keeping around so that when you start reminiscing about your school days you can remind yourself how stupefyingly boring they really were.

Revised: Poxx now allows you to specify how many divisions you want per side, and draws the pattern in either temporary lines which are erased by a Redraw, or normal ones which can be plotted.

**Punk**

Insert multiple blocks a user specified distance apart, put in incremented tag numbers, break the lines they hit, etc. Called from the second Terminal icon menu.

**Pur**

Pick two entities, Pur will decide what they are and modify them accordingly. Note that the selection order is significant - as a general rule the first thing picked will be the one changed or moved.

- Line and Circle: move the end of the line closest to the pick point so that the line is perpendicular to the circle.
- Circle and Line - move the circle to the end of the line.
- Block and Circle - Make the line perpendicular to the largest circle in the block.
- Circle and Block - move the block to the end of a line.
- Line and Arrowhead - move the closest end of the line to touch the point of the arrowhead, rotate the arrowhead to match the line angle. The arrowhead must be a solid - arrowhead blocks are ignored since it's impossible to guess that they are supposed to be arrowheads, where the base point is, etc, etc.
- Arrowhead and Line - Move the arrowhead to the closest end of the line and rotate it to match the line angle.
- Circle and Circle - move the first circle along a line between its centre and that of the second circle until they touch.
- Block and Block - move the first block insertion along a line between the centre of its largest circle and the centre of the largest circle in the second block until they touch.
- Text and Circle – move all selected text as a set to the centre of the circle. Most entities are selected with a single pick and then Pur asks for the second of the required pair, if the first one is text then Pur keeps asking for entities until you hit Return, then it asks for the second entity.
- Text and polyline – if the pline is a rectangle then the text is centred in it when it is selected, if not then Pur asks for a second corner and centres the text in the same way that Tea does.

Further refinements are in the works, most notably dealing with the arrowhead solid vs. block problem.
See also Crab, which can realign two circles or blocks and a line.

**Pw**

Widthed polylines are often used for borders, matchlines, etc. which need a little more emphasis. This isn't always consistently done even where standards so dictate, probably because it uses up time, thought, and whatever is the opposite of minor irritation.

Pw gives a polyline a width (by default equal to half of Dimscale - the most popular width), if it is used on a line or an arc it first makes it into a polyline. It can do more than one at a time.

**Qeb**

Pick a block, Qeb wblocks it out to disk unless it is already present in the current directory in which case it asks if you want to overwrite the existing one. Not much simpler than answering the wblock prompts, but it saves typing the name twice and is much easier than explaining to ones coworkers for the thousandth time how they have managed to create a self referencing block.

**Rat**

The attdef command always seemed a bit convoluted to me - why couldn't one just make an existing text entity into an attribute? Now you can.

Rat asks you (just to make the routine complete) if you want to change any of the flag settings (Invisible, Constant, Verify, Preset) and lets you do so if you wish. In case you have already changed them it tells you the current settings: ivcp is all off, IVCP is all on, etc. Note that they are all reset to Off at the start of each drawing session. Then it prompts for the Tag name, Prompt, and Default value, allowing you to use the original text string for any or

all of them. (A space is interpreted as a lack of desire to input any value, a <Return> uses the default.) You can go from text to attribute in a pick and four returns if you're in a hurry.

If you want to explode a block and make the attributes into text see Blam.lsp.

**Repo**

Search for any blocks found in the drawing, if any are found reinsert them bringing their definitions up to date. This should be used with caution since if a block has been redefined in a drawing there is probably a good reason and changing it back to the original may cause problems.

**Repall**

Text search & replace for batch files. Can do more than one search at a time, but requires the user to edit the code, which is a pretty minor thing to do. Although this might be simplified in a future release if enough people bitch and someone suggests a good alternative.

See Fang, which makes and runs batches.

**Right**

Convert text to right justified text. The default insertion point is the original or you can pick a new one - it's one of those decisions which the program can't make for you.

**RL**

Relayer with subentities: enter an origin layer name and a destination layer name, Rl moves everything from the former to the latter, including block subentities.

See also Lump.

**Rocket**

Reformat text so that fractions will display properly when used with the font Rocket.shx.

**Scabl**

Rescale all of one block type, or all blocks in the drawing. The latter option might be a little bit dangerous.

**Sch**

Restyle text: pick the text, select the style you want from the dialog box.

A drawing begins to look badly designed if it has more than two text heights or styles, unless one is used for the company name, Sch helps retroactively restrain other people’s artistic impulses.

**Schlock**

Warns if any layers are off, frozen, or locked. Runs when a drawing is opened so that you know in advance what you are dealing with.

**Scream**

A variation on Skull. (That information may be of limited interest since Skull is no longer included, having been rendered weird by the passage of time.)

Pick a number of blocks, Scream replaces all of the attributes in one with the tags so that you can select them even if they are empty in all of the selected blocks. Select one of these (or an unmodified attribute in any block) to indicate which attribute you want to edit, Scream then restores the old values and asks for a text string which is then used to replace the picked attribute in each block.

In other words this allows you to replace a common attribute in several blocks with a single value.

**Scrub**

Search and replace for blocks: pick the blocks, indicate which attribute to search using the same method as in Scream, or search all of them, input the old and new strings, the rest will happen without you.

**Sdel**

A departure from tradition: a program written for my own use. This routine lets you select items using the standard procedure and then prompts for a type of entity - Line, Circle, Insert (Block), Text, etc., and erases from the

selection set all objects of that type. If you want it can also delete every entity of a given type from the entire drawing. The erased objects can be returned with Oops, at which point they become the Previous selection set.
Not the type of thing one uses every day, but (like dynamite) likely to come in handy if you know it's there.

**Sec**
Section arrow and tag maker.

**Shag**
Shapes were a good idea that didn't catch on, probably because they were more difficult to make than blocks and required that another file be distributed in addition to the drawing. They still exist in some drawings, and the shape files still aren't distributed, so one gets annoying error messages about missing files.
They are also almost impossible to get rid of, because they can be incorporated into a drawing in a number ways:

- As loose entities.
- As part of a block.
- As part of a linetype definition.

Also if the file isn't available they are invisible.
Shag can locate loose ones and you can then use Erase Previous to eradicate them. It also tells you about shapes in blocks and linetypes so that you can purge or redefine them.
A recent drawing of moderate size took two minutes to regen. It zoomed normally and had no frozen layers, auditing didn't fix any problems. Eventually I ran Shag, which found 900 shapes way out in space, and I erased them, fixing the problem.

**Shark**
Kill any circles, arcs, lines, and polylines under a certain size.

**Shutdown**
Import an excel file into a shutdown key drawing. Requires some setup for which you will have to contact Rocket Software. Saves an immense amount of time, minimizes the possibility of transcription error, and produces a better finished drawing.

**Sklork**
Vertically alphabetize blocks - select a number of insertions of one block, Sklork puts their attributes in alphabetical order vertically. Alphabetization is done based on the value of the first attribute, if there are others then they are carried along with the first one.
See also Abcat.lsp.

**Snake**
Put things on snap: move the insertion point of entities to the nearest snap point, and where appropriate alter their dimensions to make them fit. Both ends of a line are put on snap, as are both endpoints of fitted or aligned text and all vertices of a polyline; the centre of a circle is snapped and the radius is altered to the nearest multiple of the current snap, all four corners of a solid or trace are put on snap. Arcs, point entities, block insertions and centre, middle, right and left justified text entities are moved unaltered to the nearest snap point. Dimensions are ignored, being a special case, the routine Cobra is a modified Snake which will also put them on snap.
Snake is very powerful, and is best used with a degree of caution, especially on things which are very badly drawn.

**Snat**
Toggles snap between Dimscale and 2.5 x Dimscale, and turns snap on if it is off. F12 runs this.

**Snort**
Export loose text into a csv by location. Or, in more detail: Snort is the latest text to file extractor. It writes an array of text entities to a comma delimited file in the same arrangement as in the drawing, and tries to deduce where columns are so that if there is no text in a column in a given row then a blank can be placed in the text file. The resulting file can be imported by any decent spreadsheet program.
Caveats:

1. Snort only works with text, so if you are trying to export attribute values you must either try Bullet.lsp or use Blam to explode them. Mtext must be exploded first.

2. If the text is so badly aligned that you can't figure out where the columns are then Snort may also become confused. You can realign them with Vbcx, which will also help if half of the text in a given column is left justified and the rest is centred.
3. Sometimes what looks like columns of text is lines with lots of spaces. Strafe.lsp can fix these.
4. Sometimes what looks like text is loose attdefs. Cat.lsp can fix these.
5. The time Snort takes to sort out the rows and columns rises with the square of the number of entities being exported, so if you are doing several thousand entities it may be best to either do them in strips and manually cut the files together or let it run over lunch.

**Sort**

The predecessor to Snort – writes out arrays of text to comma delimited files, but doesn't care if there is no entry in a column. Included because although Snort seems to render it obsolete it still occasionally gets used.

**Spam**

Suck selected entities from paper space into model space. This can be used when some misguided person has put a schematic in model space and the title block in paper space, usually because paper space is more scientific and needs to be used as much as possible.

Which brings up the question of whether abilities should be used because they exist or because they make sense in the current situation. As a rule of thumb there are a few things that it is a good idea to do just because you can: launching rockets, setting off explosives, and flying, if you have made friends with Superman and he has taught you how. Most other things should be held in reserve until you need them.

Especially, in case you have missed the point here, paper space.

**Spi**

Temporary graphics (you can't plot or save them) – a spiral of stars.

**Splurge**

The purge command now comes with a dialog box so that you can set everything up neatly, choose from numerous options, and then purge everything exactly as you wanted. All splurge does is purge everything it can, which is what you want 99.999% of the time.

**Spray**

There have been a number of routines which could draw text in an arc or a circle. Spray allows one to apply it to a line, arc, circle, or any polyline, and to type it directly or read it in from a text file. You can do some amazing things with Spray, and unless you are heavily into architectural detailing almost all of them are completely useless. I use it quite a lot when working on projects of a more decorative and less strictly work-related nature.

**Spyr**

A fascinating routine which came into being by accident while I was writing a plant maker.

Spyr prompts for a start point, segment length, start angle, angle increment, length multiplier, and number of segments. Then it draws lines, incrementing the angle and length each time. The result is one of a variety of patterns which are much more interesting to see than to read about. The increments (which are the heart of the matter) are used as defaults the next time around in case you want to do the same thing again or try some close variation.

I recently occupied a few idle minutes by making a number of spirals with Spyr, making slides of them (with the Mslide command, in case you can't be bothered to look it up), and writing a script file to display the slides in a repeating loop to the amazement of my colleagues, none of whom had ever read the manual.

The general format of the script file, in case you are interested, is:

Vslide Slidename
Delay 1000
Vslide AnotherSlidename
Delay 1000
Rscript
(The delays are optional.)

**Squab**

Rewidth attributes in a variety of blocks so that they fit properly. Squab finds the blocks on its own and adjusts the widths, you just tell it to run.

**Squid**
Adjust the width scale factor of attributes in shutdown key drawings to fit in their cells. Run after a shutdown key has been imported with Shutdown.lsp.

**Squish**
Squashes text so that it is not over a certain physical length. Similar to Vb, but doesn't generate fitted text so the entities can be further edited without problems and if the number of letters becomes less (due to either bit rot or editing) the text doesn't become ridiculously wide.

**Ssq**
Edit multiple text entities in order by position: select a column or row of text and Ssq highlights them one at a time and asks for a replacement value. Doesn't speed up the typing, but saves time because you don't have to keep track of where you were.

**Sst**
Another pattern maker, like Spyr.lsp but capable of more elaborate patterns. Sst stores multiple line lengths and angles in lists so that rather than drawing a pattern of single lines it uses as many as you care to draw. The angle of each line segment is the input angle taken as an angle from the direction of the previous segment and not an absolute angle, so the results are pleasantly hard to predict.
Why, you ask, is such a thing necessary? Well, it led to Zlin.lsp, which might be of some use...or lead to something which might be...

**Sta**
Temporary graphics – a circle of stars. Located under the Geometry>Other pulldown with a few similarly relaxing functions.

**Stand**
Standard detail tag installer. Typically called from the Misc. Icon menu.

**Strafe**
One sometimes finds one line of text filling several columns, with spaces added between words to reposition them. This clever trick must give someone a great deal of satisfaction, but it is sloppy and a pain to edit.
Strafe breaks a text entity at each occurrence of a specified number of spaces, so you can convert these abominations into proper columns of text and neaten them up with Vbmx or Tar or something.
Strafe makes new entities which match the original perfectly – the new text should look exactly like the old stuff, unless the current text style doesn't match the one in which the text was drawn, in which case it will appear to suddenly change. If this happens then you can use Sch.lsp to restyle the text. You could also use the Change Properties dialog box, but the thing always reminds me of one of those combination hammer/wrench/pliers/screwdriver tools – it sort of works, but everything turns out to be slower and more difficult than most people can stand.

**Styx**
A lot of drawings use the Txt font for the Standard text style, and do not look good as a result. Styx changes Standard to use simplex and a width of 0.85, which seems to be pretty universal; Simplex is essentially the same font as Romans but the insertion points are exactly where they should be, whereas with Romans they are a fraction of a unit off.
See also the section on the Rocket font.

**Sun**
Select an entity or a block or polyline subentity and get its entity data list. Handy if you know entity data lists, otherwise this is one of the few things in life that you can safely ignore just because you don't understand it.

**Sundog**

Most cloud drawing routines require that points be picked in one direction so that the resulting cloud bulges out and not in. Cm.lsp can decide which way the bulges need to go, but screws up very occasionally, in which case Sundog can turn the cloud the right way out.

**Swx**

Switch positions of two groups of entities by selecting one group, a base point, the second group, and another base point. Easier than moving one group out of the way, putting the second group in its place, and moving the first group to where the second one used to be.

Generally this one doesn't get rave reviews - largely I think because it seems to require a brain to use. Actually a modicum of common sense will suffice, if you have any this is worth at least a try. I use it all the time.

**Sz**

Blocks often surround attributes with other entities so that there is only room for a limited number of letters - typically seven for a filename, five for the date, etc. Sz reduces the width scale factor so as to squash the text into a smaller space. Each time you pick the attribute it will be compressed more until it falls below the threshold of legibility (0.5) at which point it will be restored to full width and you can start again.

Notes:

1. This will have no effect on fitted attributes, so if it won't work check the attribute for fitted justification and the designer for sanity.
2. Sz will also work on text, although I prefer Wt.lsp which allows selection by crossing and windowing.
3. See also Sqz.lsp which does the same to every attribute in a block.

**T**

This isn't really a function, just a modification which makes trim a little easier to use by turning off snap while the trim command is active and then turning it back on afterward. Like C.lsp, the benefits are out of all proportion to the complexity of the code.

**Tar**

Pick a ratty looking chart done with loose text in boxes. Tar will find the location of the box surrounding each text entity and neatly centre it therein. Makes a major difference in a very short time, although if the frame itself is a mess this isn't really going to help.

You should be zoomed in close enough to read the text for this to work well, and none of the boxes surrounding text you have selected can be off the screen.

**Tater**

Tater will replace the string in any text, dimension text, attribute or attdef entity with one either typed in or taken from an existing entity of one of the above types.

It can also replace the string in constant attributes - I was recently working on a set of drawings in which the revision number and drawing name were contained in constant attributes which had to be updated in the block table in order to change the displayed value, a stroke of genius which made it difficult for saboteurs to upset the filing system at the small cost of making it nearly impossible for anyone else to work with it at all.

Tater handled these oddities with no trouble at all, although since it does so (as presumably did the original method) by updating the block definition in the tables, the value in this attribute will be the same for all insertions in the drawing - alter one and you alter them all. The same is true of text contained in a block (as distinct from an attribute.)

Tater, by the way, can get at text no matter how deeply nested it is.

Other notes:

1. Tater updates block subentities after changing them so that they are displayed correctly. The Undo command doesn't know about this, so if you undo changes made by Tater to a block or dimension you will have to either Regen or update the de-tatered entities with Irg.lsp.
2. Tater changes the 1 association list in an entity. If it is used on a loose attdef it will therefore change the prompt string and not the displayed value, which is the tag name. I was tempted to make it change the tag, but decided to bow to the superior wisdom of Autodesk - there was a remarkable intuitive correctness to the layout and use of the early releases of AutoCAD, and while this is sometimes obscured by the recent avalanche of features I tend to respect the intentions of its designers as far as possible.

3. Tater is a nice command and is dedicated to all the English teachers who have attempted in their dilute way to stamp out the word "nice". Speaking of words, the drawing recovery command in Acad 12 used the term "elision". For those not possessed of a dictionary, this is a real word meaning, as you might suspect, removal.

**Taz**

Sometimes one wants to move an entity to a different layer. This isn't too difficult for most entities, but when dealing with blocks, dimensions, etc., subentities have an annoying habit of remaining on their original layer and in their original colour.

Taz gets around this by modifying the block tables - it places all subentities on layer 0 and colours them bylayer so that they appear on the layer the block is inserted on.

Taz asks for a layer name, which can be entered from the keyboard or indicated by picking another entity. If the layer doesn't exist you will be asked if you want to create it. Taz then prompts for entities to move, modifies the subentities, and moves everything to the chosen layer.

Caveats:

1. Since the block tables are changed all insertions of any picked block will be changed - the insertion layer will remain the same for ones not in the selection set, but all subentities will be placed on that layer.
2. Undo seems to be able to handle direct modifications to the block tables, but I strongly recommend saving before attempting to Undo through Taz.
3. Taz has nothing to do with the dog of the same name – it was written before I got him, and he came with the name. Also a suitcase full of dog toys and an old electric guitar he couldn't play, but that's another story.

**Tbone**

Brad asked me the other day if I could write a routine which would let him fillet two lines without necessarily truncating them to the ends of the arc. This presented some difficulties since I didn't want to recreate the Fillet command from scratch; also, while a line which doesn't reach the arc end should be extended to meet it, it is not intuitively apparent when a line which is too long should be truncated and when it should be ignored.

I started by writing a routine to call the Fillet command with two line (entity name and pick point) lists. This worked ok, so I modified it to restore the lines to their original length after filleting. My first thought was to extract the line start and end for each line, (subst) them back into the current data list, and (entmod) them back to the original length.

As I wrote the code it occurred to me that since I had already assigned the entity data to a variable I could just (entmod) it and restore the entity to its pre-fillet condition, a much simpler solution. This left deciding when to restore the lines. This is somewhat arbitrary but seems to work well: if a line crosses the intersection point of the lines it is restored, otherwise it is truncated. This part is less elegant, so if you want the mechanism you will have to examine the code.

**Tch**

Change one or more lines of text to a single text string which is either typed in or copied from an existing text string by picking it. One of the few truly indispensable Lisps.

**Tea**

Centre text in a box. Select a stack of text, pick two corners of a box, the text will be neatly middle rejustified and moved so that the centre point of the text is at the centre point of the box.

Makes neatly drawn tables so easy that you will be tempted to sneer at those who make sloppy drawings.

See also Tv.lsp, Tar.lsp, and Tic.lsp.

**Teal**

The same as Tea, but the text is left justified along the left side of the box. Dedicated to Kim Buye, who suggested the program and unnervingly also came up with the name I think I would have picked.

**Tech**

Search and replace, like the immortal Chgtext, but will do its thing on any entities that look like text.

Tech retains the search string as the default for the next use. This is harder to do for the replace string since it is hard to differentiate between a <Return> meaning "Accept the default" and a <Return> indicating an empty string.

It also indicates how many replacements it made, and in how many lines - if there were more replacements than lines but you only wanted to change one string in each entity then you can assume something went wrong.
See also Tater, Texas, and Mach.

**Ten**
It is difficult to use the insertion point of a block as a base point for movement, rotation, etc. if it consists only of attributes – using the insertion object snap will give you the insertion of the attribute you pick rather than that of the block. Ten returns the block insertion point, it is located on the popup menu with the standard osnaps.

**Term**
Put an arrowhead on the end of a line closest to the pick point. Motivated by the standard "I think I seen some other program do that once" rather than any perceived need. Matches the arrowhead size to that used by Dim.

**Tess**
Draws a 3D polyline representation of a tesseract - a structure that is the four dimensional equivalent of a cube, often referred to as a hypercube.
Since a four dimensional object can't be drawn completely in three dimensions, this bears the same relation to the (hypothetical) real thing that a drawing of a cube on paper does to a real cube - each is an attempt to represent something in one less dimension than it has.
In any event, Tess draws a tesseract with its base coplanar with the current UCS, so you will have to view it from some other angle for it to look like something other than a picture frame. Then you can put on your resume that you have some experience in 3D drafting and some in 4D.

**Texas**
I can't be bothered to rewrite the notes for Tater here, so: Texas is like tater but will replace the text in multiple entities (attributes, text, dimension text, text within blocks, constant attributes, etc.) at once.
The Nentsel function on which it is based doesn't support windowing so entities will have to be individually picked, but it still saves a lot of time over individually using ddatte. Also Texas, like Tater, allows copying a text string from an existing entity in case you are not an enthusiastic typist.
See also DDD.lsp.

**Tga**
Draw a polyline loop around one or more lines and insert cable tag blocks. Allows you to specify which direction the tags are arrayed in and keeps adding tags while text is entered to occupy them.
See also Mull.lsp, which reorders the tags.

**Tic**
Reposition text in circles: select the circles, if there is any text in them it will be centred in them. Great for cleaning up archaic drawings where the relays aren't blocks and the text is nowhere near the centre. Also draws a nice (temporary) pattern around each circle.

**Tiger**
A line repair utility, like Wolf, but one which only repairs lines if they are collinear to several decimal places. Very useful. The long-awaited update which allows it to deal with polylines is still being awaited.

**Tlen**
Measure heat trace lines – select the lines in question and Tlen will tell you how long they are. For some reason designers are often reluctant to use CAD for anything other than drawing nice pictures, but they all seem to have a weakness for using it to determine the amount of heat trace they need.

**Tray**
Draw a hatched cable tray elbow. Input the bend radius and tray width, pick the centre point and the quadrant of the circle you want the elbow drawn in, Tray draws the elbow and hatches it. Nobody seems to rave about this but I find it very useful.

**Trayhach**

Hatch a cable tray – sets up the hatch parameters and prompts you to select the tray in question. Much quicker than setting everything up manually.

**Trouble**

Diagnostic information, for use if things aren't working right. If you can't run this from the command line then RocketCad isn't loading properly, if you can then it will help diagnose the problem.

**Trx**

Draw a vertical tray section – a box with an X through it. The resulting figure is a polyline so that it can be easily moved, hatched, or erased.

**Ttb**

This is new: take a chart which is done in loose text but for which you have a block for each row. Show Ttb the text, give it the block name and the first insertion point and vertical spacing, it will replace the text with blocks containing the values.

Caveats:

- Ttb calls some subroutines from Snort.lsp, which must be available.
- Each column must have at least one text entity in it or it will be ignored.
- If there aren't enough attributes in the block to accept the text in each row then the excess will be discarded.

**Tv**

Columnize text and centre it in a box. This is like Tea.lsp, but it adjusts the line spacing on the text which makes it ideal if you want a neat column, but if your box contains text in different heights or that you want to leave nonstandard spaces between then Tea is a better bet.

**Txs**

Global text size changer: you specify initial and desired sizes, and anything in the drawing of the initial size will be changed to the desired one. I would like to be able to say that this was mystically inspired by a source of ideas that only I have access to but it was actually suggested by MI, who has an amazing talent for ferreting out half-baked lisps and who was dissatisfied with a similar routine which changed every text line in a drawing to one height - a useful ability if your output device is a daisy wheel printer.

**Txtt**

Change any selected text string, attribute, etc. to the filename without the .dwg extension or the path.

**Typ**

Swap the Typ. tag from side to side in a material tag block. Not Earth-shaking, but it is quite satisfying to have a routine to deal with one of the minor irritations which crop up when rearranging a drawing. Typ also adds a Typ. tag if the block doesn't already have one, you can tack a number on if you like.

**Uc**

Convert text to upper case. Similar to Lcs.lsp, which isn't included any more because it never got used. Uc works on text and will also convert all attribute values in any picked block.

Lower case with initial capitals is the most legible method of writing, hence the relative rarity of books printed in all caps, but there is something about engineering drawings that makes people want to emphasize things – possibly the knowledge that most designers are about ninety years old and half blind.

**Ud**

A routine containing a list of every current dimension setting with a very brief explanation of what it does. You can copy this to a different name, edit the settings to match your standards, and then run it in every file conforming to the standard for which it was made.

You could of course use a prototype drawing and do things right from the start, or use the design centre to drag dimension styles between drawings, but these approaches don't help to deal with the millions of drawings that were not drawn to any standard or which originated at companies using different ones.

Ud can be run as a batch with Fang.

**Upright**

There are some acceptable angles for text, and some which aren't. (See the note under North Arrows) Upright finds the ones which are upside down and rotates them 180 °. It can also rejustify them so that the origin remains in something like the original location even when the text has been rotated, and it can do the same thing to blocks. This is very powerful, so be careful and remember the Undo command.

**Us**

A number of people (and it got pretty tedious after the first two) have told me that the wonderful thing about their favourite Cad program (not AutoCAD, in case you aren't following this) is that it allows one to pick a point and then cycle through the entities which cross it so as to select one which is otherwise inaccessible. Once in a blue moon I feel the need for such an ability, but I mostly wrote this so that I could ignore them.

Run US, select the objects you want to look at - a small crossing box is generally best, but I've left that open - and then keep hitting <Return> until the one you want is highlighted. US will cycle through the selection set until you stop it, so you don't have to choose your target on the first pass. Then hit any letter and <Return>, or just *Cancel*.

The entity in question is now the Previous selection set.

**Uv**

This contains two routines: Uv, which lists all of the user system variables, and Kuv, which empties them. This is not all that useful but there are some programs which screw up and fill the user sysvars with trash and can only be forced to reset by emptying them.

**Vb**

Venetian blind - multiple text rejustify to fit. This takes a number of lines of text and fits them between two specified points while keeping the original vertical placement in the drawing.

Vb and the other similar programs are very nice for cleaning up sloppy columns of text. I once had occasion to work on a set of drawings done by a party who did not use snap, considered all text styles to be identical, and who liked to mix variable and fixed height text. Vbc enabled me to fix the average drawing in about five minutes, which impressed both the boss and my fellow employees who had refused to have anything to do with the situation. (An unretouched testimonial by an innocent party.)

Another trick worth keeping in mind if you're bored: Type Vb, window the whole drawing and then watch all of the text suck into a single column. Then tell your supervisor it just happened by itself.

The variants:

- Vbcx - Centred. This one allows you to pick two points and centres the text on a point halfway between them, or pick one point and <Return> to centre on it.
- Vblx - Left justified.
- Vbrx - Right justified.
- Vbmx - Middle justified.
- Vbml – Middle Left justified
- Vbmr – Middle Right

**Vess**

Vessel maker – drag a box, Vess fills it with a vessel, elliptical endcaps and all. Useful if you have to show something and you can't get a mechanical drawing to chop up, although there is the danger when faking piping that someone may try to take it seriously.

**Vis**

One of the popular add-on packages for AutoCAD inserts all of its standard blocks with invisible attributes so that they can initially be edited only with its own built in routine. Vis remedies this situation – it turns any invisible attributes in a selected block back on.

If you find Vis useful then you might also want to look at Flat.lsp – there isn't much similarity between the two, but drawings which require Vis often seem to benefit from Flat.

**VL /VU**

VL locks all viewports and Vu unlocks them.

One of the more brilliant and well thought out features in the recent releases of AutoCAD is the ability to lock viewports. Previously if you panned or zoomed while in a model space viewport in paper space you would completely destroy the alignment between the two spaces for that viewport.

If the vport is locked and you try to zoom or pan then Acad switches to paper space, runs the command, and then gets back into the viewport.

It might be a good idea to lock all viewports each time a drawing is opened, they could thus be created and aligned with the desired model space geometry, then once the drawing was reopened they would be safe from accidental zooming unless they were specifically unlocked.

**Vvb**

Vertical venetian blind. Takes a column of text and respaces it vertically. You specify the start point and the vertical spacing. Note that this does not move text horizontally, only vertically. Vvb offers either the correct vertical spacing or the last spacing entered - if any - as the default. If you don't like the last one you tried you can snap it back to the default by entering a D.

**Wir**

Jb wiring installer – draws wiring down one side of a terminal strip, complete with fillets. If you use this consistently it will save you enough time to have an extra long lunch once every six weeks.

**Wireline**

Change all line numbers in a block. Select a block and input a starting number, Wireline replaces the attribute values with sequential numbers. It is intended for quickly renumbering line number blocks, but could also be used for other things.

**Wiretag**

Insert a horizontal or vertical left side wire tag block.

**Wiretagr**

Insert a horizontal or vertical right side wire tag block.

**Wlay**

I was recently asked to print out some survey drawings which were drawn under severe time and sense constraints. As they displayed on the screen the background vanished under layer after layer of text, each area being repeatedly overwritten until I expected the whole mess to fall over and slide off the bottom of the screen. By the time it was all displayed there was no unoccupied space left.

The fellow who was supposed to be dealing with the drawings asked if anything could be salvaged. He rejected my suggestion that we find a delivery service in the city that would tar and feather someone, so I wrote Wlay.

This program wblocks each layer in a drawing. Each resulting drawing is named after the layer with a number prefixed to it to prevent long layer names from truncating to the same filename and overwriting each other. Wlay prints the layer name and drawing name in the lower left corner, draws a border around the extents so that the drawings can be plotted on transparencies and lined up with each other, and writes a log file matching each layer name to the resulting drawing name and listing which layers were empty and thus ignored.

Is this generally useful? Not really, but if your clients want to know about layers they can't fail to be impressed with drawing dissected into a coil-bound stack of transparencies.

**Wm**

Move words from one text line to another. Pick two lines, if the first is above the other then the last word is moved from the upper line to the start of the lower one, if the first is below the second then the last word is moved from it to the start of the upper line. In other words it does just what you might expect. If there is no destination line then a new one is created in the right position relative to the first one.

The critic might point that Mtext makes this unnecessary, but:

- There are a vast number of existing drawings which already have text in them.
- Mtext sucks.
- You don't need Mtext if you have Wm and Vbcx.

**Wobble**

Make an existing entity into heat trace. Wog can't draw circles, but wobble can make an existing circle into heat trace line, which is handy if you have to heat trace the base of a flare stack. Its default wavelength and amplitude are both 1.5 times Dimscale, which is half of what Wog uses, but which seems to look better on small entities.

Wobble also has the ability to wobble the entity it just created, and to repeat this as many times as you would like, which is interesting if you have a fast machine. For real work the number of cycles should be set to 1.

**Wog**

Draw wiggly heat trace lines. These are put on the Htrace layer and have a wavelength and amplitude of 3 times Dimscale. Wog draws lines until a <Return> is entered and then puts a power kit symbol on the start end and an end seal on the other end.

**Wolf**

A line repairer with discretion - Wolf will only combine two lines into one if they are collinear, that is if one could be extended to lie over top of the other without rotating it or shifting it sideways. Select some lines, Wolf will join the ones which need to be fixed and ignore the rest.

Wolf allows for lines being out of alignment by a number of drawing units equal to the value of the sysvar Dimscale, so if it's being too liberal you can either reset dimscale or modify Wolf. So far this has worked ok, though.

See also Tiger which does the same thing but is much less forgiving.

**Wormdog**

Rotate an attribute by directly selecting it. Another example of the amazing things you can do with the archaic and powerful –attedit command.

See also Mach.lsp.

**Wt**

Multiple text and attdef width scale factor change. Occasionally invaluable.

**Xa**

Search & replace attdef tags and prompts. Originally created for making stacks of attdefs for use in the revision areas of title blocks. This will save enough time that you will be able to remember not to put the prompts in all caps and not to preface every single prompt with "Enter."

**Xing**

Crossing locator and breaker. Select a number of lines, Xing will find the intersections between them (if any) and break the vertical line of each crossing unless it is an endpoint in which case it will be untouched. Not only is this much faster than doing it by hand (it prints the elapsed time in case you are in doubt), it is more uniform than your co-workers who think that snap is an infringement on their freedom of expression and is much more interesting to watch (unless they are strikingly good looking or belong to a cult which requires the wearing of a gorilla suit at all times).

Notes:

1. If your lines aren't on snap then they may either overlap slightly in which case endpoints will be treated as intersections and cut, or they may not meet in which case there will be no intersection and nothing will happen. You just can't place lines to sixteen decimal places by eye.
2. This is an updated version of Xing. Unlike the original it doesn't care how far out you are zoomed, and it adjusts the breakpoints so that angled lines break at the same vertical distance as perfectly vertical ones. It also prompts you for a break distance (halfwidth) and saves that as the default.
3. Note saved from the end of the old description: Xing is a very useful and reliable routine and only modesty Prevents my running on at some length about it.

See also Zing.lsp and Penguin.lsp.

**XL**

Extendo-trim: extend lines to a target line or trim them back, depending on whether they are too log or too short.

**Xmark**

Put a marker at the insertion of each entity in the drawing which contains extended data.

The marker is ring shaped, all markers for a specific application are the same colour, if there is data for more than one application attached to an entity then successive markers are scaled so that they don't overlap.
See also Kx.lsp.

**Xnames**
Place or update the block Xrefname which lists all the xrefs in the drawing. Has to be modified to work with a specific title block. Intended to be run as each drawing is opened so as to keep track of which xrefs are in use.
Xrefs are a good idea and very useful for some purposes, but using them for title blocks or logos or to keep file sizes down are all seemingly good ideas which end up being immensely more trouble than they are worth.

**Xpath**
Repath xrefs, can do them by name or repath every one in the drawing. This will require editing the file, but the last three lines are all that will need to be changed. Under R12 this required about a hundred lines of code, under R14 half that again, now we are down to seven.
Can be run from a batch with Fang.

**Yalf**
After writing Flay.lsp, which freezes all layers but the one selected, I found myself requiring exactly the opposite - a routine which would freeze only a selected layer. Yalf is the result.
See also Flay.lsp and Flame.lsp.

**Yang**
Pipe end drawer – makes the traditional Yin/Yang symbol with hatching which pipers use to indicate that you are looking down the open end of a pipe. Show it a circle or tell it to make one, it does the rest.

**Ze**
Zoom Extents by doing a Zoom Window using the Extmin and Extmax system variables as the corners. Saves doing Zoom previous six times, or doing a Zoom All if your computer has forgotten the first three zooms you did. If you have not forced a regen zooming inward, this shouldn't force one going out (and if it does you would have had to do one anyway). Also a Zoom Previous will undo the effect of this, whereas it won't undo the effect of another Zoom P.
See Boo.lsp which is similar but doesn't let you get lost.

**Zing**
A slightly more elaborate version of Xing.lsp - this one allows you to break lines either horizontally or vertically.

**Zlin**
Polyline pattern drawer. Enter a pattern of points and two endpoints, Zlin draws a polyline repeating the pattern a given number of times between the endpoints. The pattern can be saved to a named file which can be edited, commented, etc. See the more elaborate notes which you can find in the table of contents.

**Zp**
Zoom Previous with timer.
The original version of this had a line: (write-line "Command: Zoom") and another: (write-line "Zoom: All/Nothing/Hi mom/Infinite/Oblivion/(x): P") or whatever the prompt is so as to give the impression that I had actually entered the proper command, my superiors taking a dim view of any type of customization for some reason not clear to me (or, I imagine, to them). You can insert some version of them before the line (command "zoom" "p") if you're bored or paranoid.

**Zx**
Zoom to the extents of selected entities or by default what is onscreen.

Notes

Notes

Notes

Notes