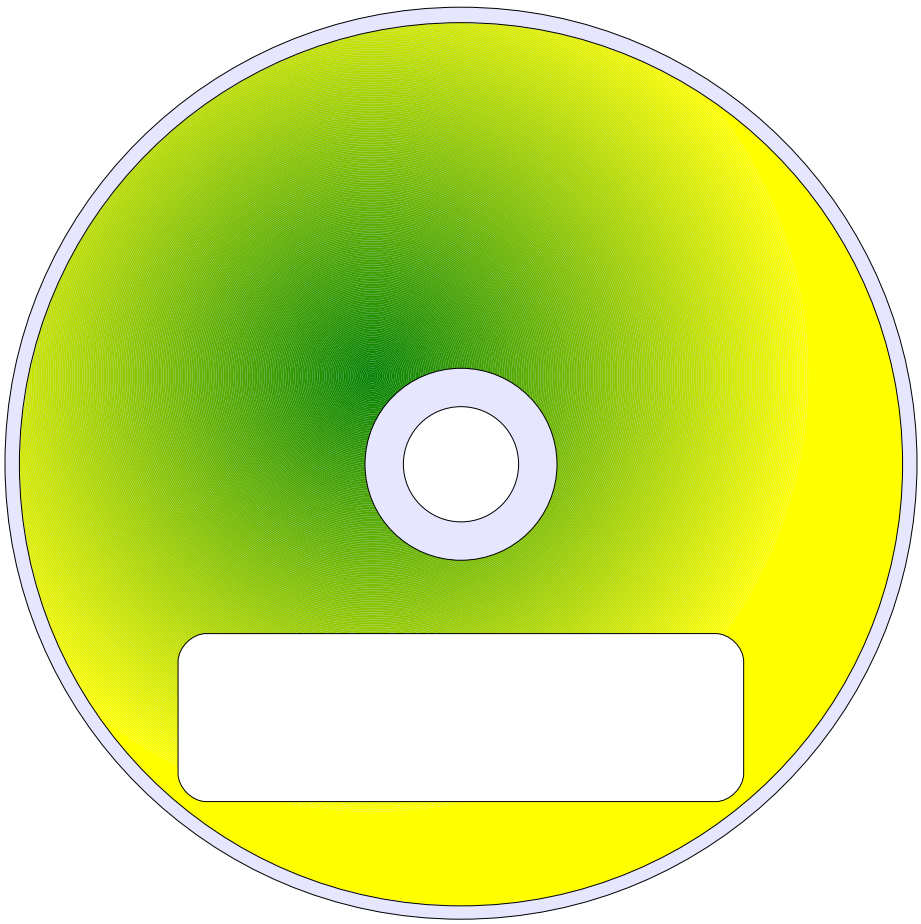
SPM 100800



PythonCAD

Raw Technical Notes

Copyright Page

PythonCAD – Raw Technical Notes

By: Pietro Moras
"Studio P.M."

Published by [*not yet—just appointed*]: Town & Country Reprographics, Inc.
230 N Main St Concord, NH 03301 (U.S.A.)

– –

Version Reference

Nr	*Item*	*Version*	*Note*
1	**Windows XP Pro** **Windows Vista Ultimate**	5.1 (SP 3) 6.0 (6001:SP 1)	Operating System
2	**PythonCAD**	Pre_Alfa_02	
3			

– = –

Foreword

Dear Reader,
...These "Notes" are the result of a precise method of working, a precise vision of what a Test and Documentation process should be all about.

Both as a Designer, ordinary User, and Technical Writer of High Tech products (mainly s/w products) I've realized how crucial is a fair correspondence between Technical Documentation and actual performance. And how crucial is that the process of Test and Documentation be carried on authentically on behalf of nobody else but Mr. User – nobody else, Developers included.

And this is exactly what I've tried do to here, that is:

- *To operate as an ordinary Mr. User, exploring the whole process he should undergo to reach a satisfactory productivity, starting from scratch;*
- *And to consequently write these Notes as a Technical Report, or Technical Draft , to the benefit of whoever would bother to know. Sort of quiet and accurate (Beta-)Testing, followed by a Technical Writing/Reporting process, in fair collaboration with the product development team.*

Because of its very nature, such Notes cannot be other than a work-in-progress, therefore requiring an agile updating mechanism, here obtained through an Author (SPM) and Date-code (yymmdd) marker, such as:

SPM 100800

<any Notes segment>

- -

So that each Notes segment, comprised between such Date-code and up to the "- -" End-Of-Segment marks, is assumed to supersede and nullify all possibly preceding one.

As you see, that's rather an ambitious, challenging goal, certainly requiring not short a time, and a great deal of collaboration ensured by a leading moderator, so to be helped in minimizing my possible (or inevitable...) shortcomings. A collaboration actually granted by an extremely supportive moderator, Mr. M.B., to whom I'm deeply grateful.

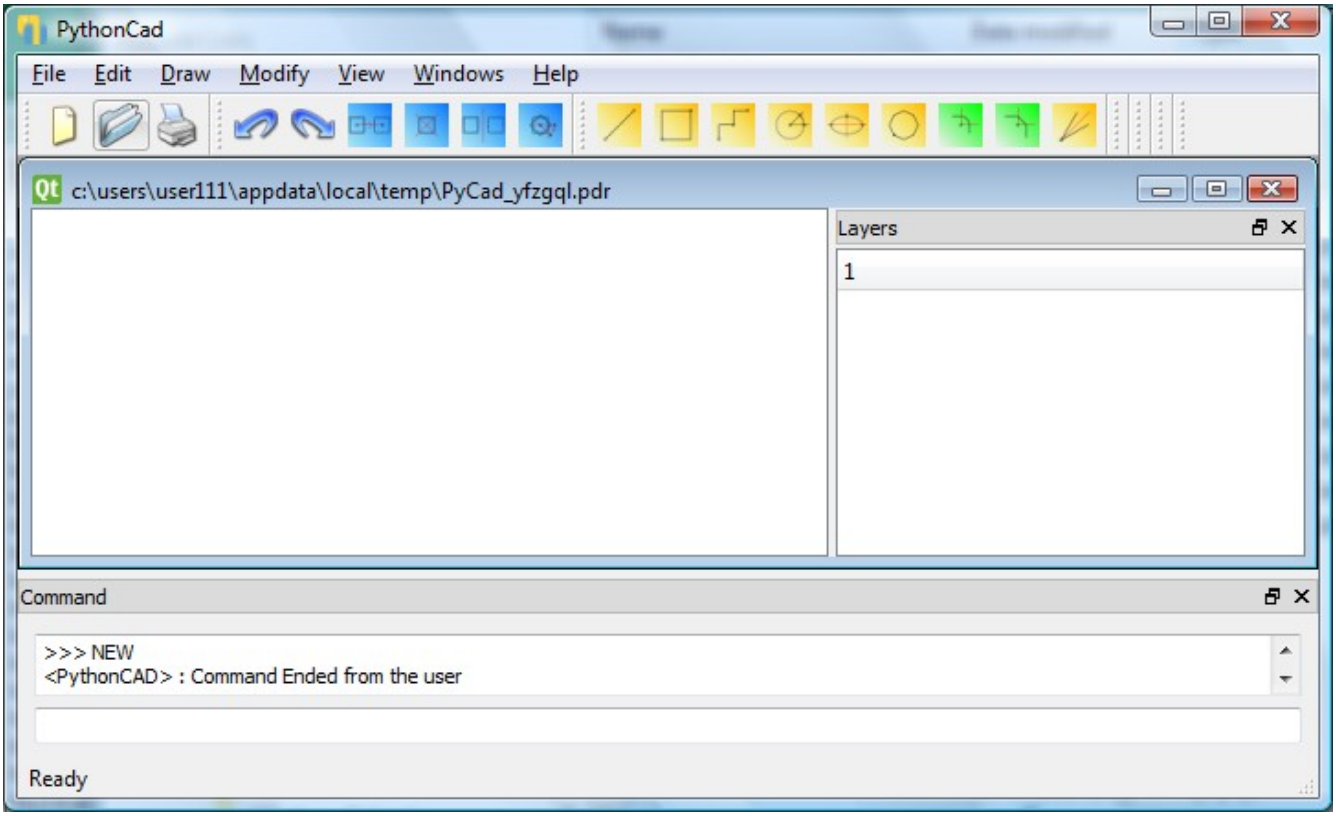
And here you have the present result, hopefully to the common benefit, and about which I'd love to here your criticisms, questions and remarks. See you.

The Author

- = -

1. Introduction & Generality

PythonCAD is a 2D general purpose CAD (Computer Aided Design) program, medium sized, open source, cross-platform. When running in a standard Windows environment it displays such a MDI (Multiple Document Interface) parent window, for the user's interaction:



Where you can see:

- Usual resizable borders and corners
- A "PythonCAD" Title Bar, with the usual icon and standard Min-Max-Close buttons
- A Menu Bar
- A Tool Bar
- One of the possibly more MDI child windows, that is a double-panned Qt[1] drawing window with the related drawing file path displayed on the title bar. With "Qt " that refers to the well known cross-platform application development framework, widely used for the development of

1 By the way, coincident with the suffix on the PyCAD executable file name (see).

GUI programs.

- A unique, fixed sized, Command child window, docked to the bottom of the parent window client area, where you have a Message Display Field and a Command Input field.

Remarks

- Here on, we assume Windows as the system environment of reference. Notes will be offered in case of relevant differences for different cross-platform environments, such as Linux[2].
- Launching this same program from Start button or directly from its "`.exe`" file leads to different conditions (see section Run, hereafter).
- We have learned that messages currently displayed on the Command window are not yet meant to be significant for the User, being still at alpha-level, and possibly used by Developers for debug purposes.
- So far the entire purpose and use of this Command window is unknown to us.

- -

SPM 100900

Down-Load

PythonCAD, as an usable product, ready for setup, is currently free down-loadable from the PyCAD web site, at URL: `http://sourceforge.net/projects/pythoncad/files/` where you'll find such a package as: `PythonCad_Pre_Alfa_02.exe Pre Alfa 02` aimed at the Microsoft target platform.

Remarks

- It's our choice here to consider always the most recent edition, as currently made available by the PyCAD Development Team. This, of course, may be equivalent to a continuous Beta-Test activity to the benefit of the next-to-came stable edition, in fair collaboration with developers[3].
- Unfortunately the `PythonCad_Pre_Alfa_03` version, currently available for down-loading, turned out to be NOT usable by a normal Mr. Users, but only by privileged Mr. Administrator. Therefore, being a normal User, I'm forced to work still on the more democratic former `PythonCad_Pre_Alfa_02` version.
- It's also our choice not to consider any PyCAD source material as used by its developers team, but only the executable material aimed at Mr. User.
- By the way: why "Alfa"? Shouldn't it be: "Alpha"?
- In other words: we intend to play precisely the role of an ordinary, experienced Mr. User.

2 Well, currently this is only a commitment, given that Windows is the only system we've used so far...

3 Collaboration carried on in public via two communication channels: the *Developer Forum* (see) and, less frequently, the *Developer Mailing List* (see). And then, from time to time, via selected up-loading of this very "Tech Notes" to an appointed contact person, a leading member of the PyCAD team.

- Download documentation currently available in the Wiki section of the PyCAD web site is severely outdated, to be ignored.

– –

Set-Up

Very little to say here, you just need to run the unique, self-contained, automatic "`.exe`" set-up procedure as down-loaded. Smooth execution, as we've tested on a Microsoft Windows Vista Ultimate operating system (Ver. 6.0, Build 6001, S.P. 1).

- Requires Administrator privileges
- Offers multi-language choice, besides English
- Typical default directory: `C:\Program Files\PythonCad`

Remarks

- Presumably the unins000.exe program file, spotted in the `...\PythonCad` setup directory (see), is the proper uninstall tool. We assume that the presence of such a tool implies that, in this case, the usage of the standard system uninstall command is discouraged, in favor of this specific procedure. Procedure that, anyhow, we haven't yet tested.
- It is our intention to test this same setup procedure in all the other declared platforms, chiefly Linux.
- The installation documentation currently available within the Wiki section of the PyCAD web site is severely outdated, to be ignored.

– –

Run

The executable program is typically located in directory: `C:\Program Files\PythonCad` and is currently named: `pythoncad_qt.exe`, with the "`qt`"[4] suffix that refers to the well known Qt cross-platform application development framework, widely used for the development of GUI programs.

As usual, this program can be run via standard desktop Start button, where it appears with the name: PythonCad, or via double-click on anyone of its drawing files, or acting directly on the cited executable file in the setup directory.

Remarks

- An oddity (probably a tiny bug): only in this last case standard PyCAD icons will appear on the tool bar, on the title bar, and on the drop-down menus.

– –

4 By the way, coincident with the icon label on the title bar of the PyCAD drawing window (see).

Related Files

`.pdr` PythonCad standard drawing file extension. In essence, all PyCAD drawing files are "SQLite" database files.

`home.pdr` usual default drawing file name

`home.pdr-journal` SQLite database service file, automatically created, and normally deleted upon exit

`Pythoncad_baseDb.pdr` SQLite database file automatically created as a template model for all subsequent PyCAD drawing file

Remarks

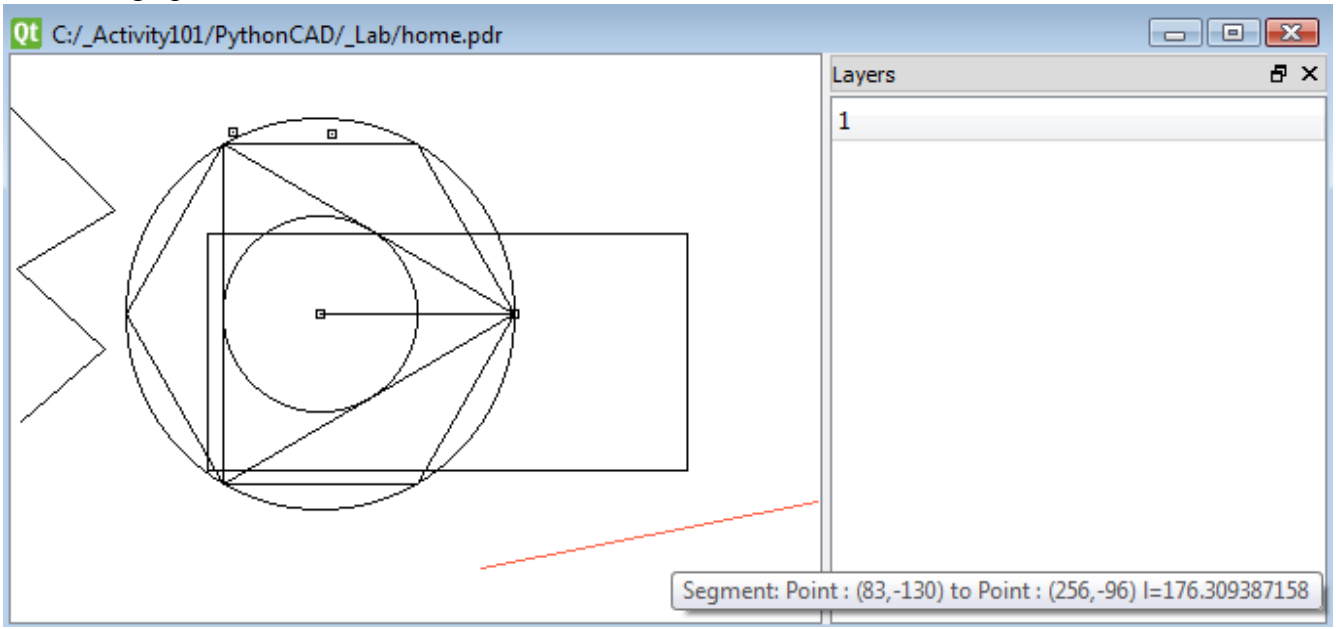
- Of course here the practical question is how much of these files is of real, operative interest for Mr User, so to conveniently go deep down into the technical details of their internal structure.

– –

Work Windows & User Interaction

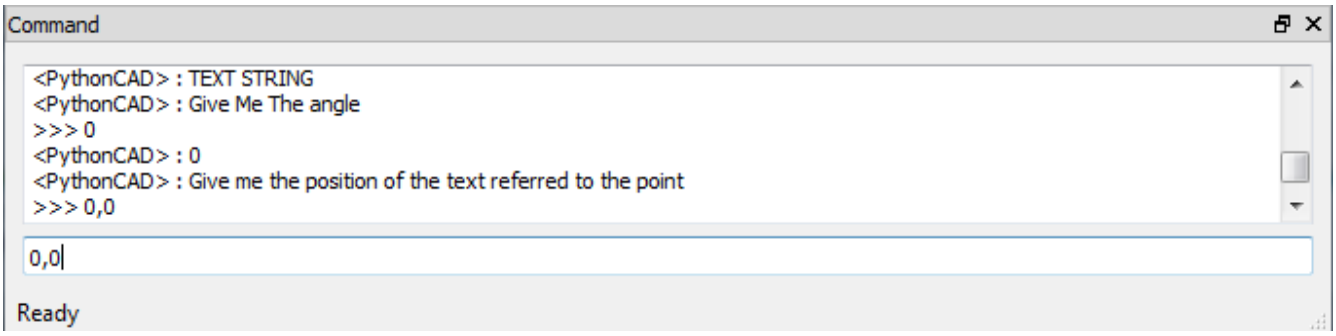
User can interact with program PyCAD on two distinct type of windows:

- A graphic Qt window



Where the user can interact analogically, typically with the mouse, and see the graphic result of his operations along with various information granted by the program, such as a pop-up property hint upon entity selection.

- A text Command window



Where the user can type text data and watch printable information granted by the program.

– = –

2. Menu Bar

File Edit Draw Modify View Windows Help

User interaction can be carried on in two distinct ways, via mouse or via keyboard entry on the Command window input field (see). Text entered this last way can assume the following format and meanings:

`generic text` as requested by such menu command as Draw > Text, unformatted
`1.23,4.56` 2D Cartesian coordinates
`...`

\- -

Menu Commands

File

New Drawing
Open Drawing...
Import Drawing...
Save In A Different location...
Close
Print
Quit PyCAD

Drawing file top management commands, and program exit.

\- -

SPM 100817

File > New Drawing

To create and open immediately a new drawing file, along with the corresponding "Qt" drawing window. The new drawing file is automatically located into a user-temporary directory, such as:

`C:\user\user111\appdata\local\temp\`

and is automatically named this way: `PyCad_lovmha.pdr`, where you'll note the random name suffix "*_6x*" and the "`.pdr`" PythonCAD Drawing file extension. More than one New Drawing can

be created during the same PyCAD session, with the last created becoming the active one.

Remarks

- The cited \AppData\... is a standard Windows directory, normally hidden. To make it visible set: Folder Options > View > Hidden files and folders > Show hidden files and folders
- Each time you act on this command you'll create such a new file in the given directory. That is to say that, from time to time, it could be wise to go there and see if you really need all of them.
- It is up to you to keep track of each such newly created file, PythonCAD will not do it for you.
- Along with such a file another file will also be created nearby, a: `Pythoncad_baseDb.pdr` whose role and structure, at the moment, is totally unknown to us.
- A double-click on such PythonCAD Drawing files will act as a usual Windows Open command. Anyhow only the PythonCAD program will be opened, not the related drawing file.

– –

File > **Open Drawing**

To open an existing drawing file, and possibly set a corresponding "Qt" drawing window on the program's main window. It's up to the user to tell name and location of this drawing file. More than one drawing file can be opened during the same PyCAD session, with the last opened becoming the active one.

Remarks

- Initial default drawing file name is: `home`. No former history name list is offered.
- No directory history list is offered either.
- The initial directory here pointed by the Open Drawing control form is different from that one where the New Drawing automatically operates. These two commands operate independently, and with no synchronization nor default mechanism offered to the benefit of the [poor] user.
- Sooner or later, here too, as with the New Drawing command (see), a so far mysterious "`Pythoncad_baseDb.pdr`" file will also be found. And, curiously enough, it could even be here opened as it were an ordinary drawing file, which we presume it isn't...
- Sooner or later another mysterious file will also appear here, a such: `home1.pdr-journal`, of course to be clarified too.

– –

File > **Save In A Different location...**

To save a copy of the currently active drawing file–that is: that one related to the active "Qt" drawing window–into a directory and with a name of your choice.

Remarks

- Usually such a command is simply called: "Save As..."
- Here too the default file name offered is: "`home.pdr`"
- Here too no historical default location is offered.
- Actually this command reveals a totally unreliable situation, where both the original and the copy files can result only partially saved, corrupted, or inoperable. Sensation is that the mess is not related only with this command, but has a different, deeper, origin. An origin that could be investigated starting from the following step-by-step reproducible procedure:

1. Crate a New Drawing, such as: `PyCad_sbnann.pdr` (take note of the name)
2. Draw a Segment, then add a Rectangle
3. Save this drawing in a different directory and with a different name, be it: `home.pdr`
4. As a first flaw you'll see that no other Qt drawing window is created.
5. Quit PyCAD
6. Re-enter PyCAD and Open both these drawings. You'll realize that both are incomplete: the Segment is there, but the Rectangle isn't.
    - Note that this one is not the only flaw encountered in such circumstances, many other flaws have been randomly encountered, but it's reasonable to assume that all have the same origin. So that, once detected and fixed this one, an entire set of only apparently different bugs will probably disappear.

– –

File > **Close**

To close current "Qt" drawing window, along with the corresponding drawing file, with the updated contents saved.

Remarks

- It's up to you to take note and remember the location and the name of this file, so to possibly re-open it later on. No related support is offered by the program. That's particularly annoying in case of brand new drawing file, where both location and name are automatically set by the program, out of user control (see: New Drawing).

– –

File > **Print**

Presumably to print the current "Qt" drawing file but, as a matter of fact, what we've got so far are only white sheets.

Remarks

- No idea, no info in sight about physical size and scale of the expected printed output.
- Anyhow, apparently it doesn't work at all, in the sense that the only thing so far obtained are white sheets...
- Anyhow it has the interesting side-effect of updating the drawing file up to last editing actions performed by the user, offering a work-around and, even more important, a diagnostic tool, for the related serious bug encountered with different commands, particularly with: Quit PyCAD (see).

– –

File > **Quit PyCAD**

To exit this program, without saving what you've done during last editing session.

Remarks

- Usually, instead of a "Quit", here you'd expect also, or only, a command named "Exit".
- We are not sure if it's by deliberate odd design, or by bug that, as a matter of fact, if you exit via this command you'll lose all work you've last done during the current drawing session (see also the *Remarks* at Save In A Different location...).

– –

Edit

Undo

Redo

Move

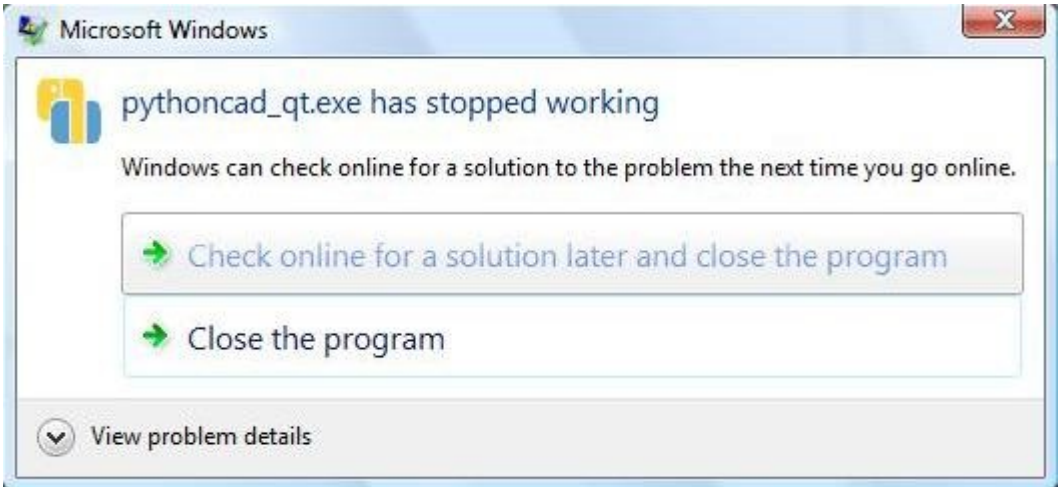
Delete

Mirror

Rotate

Top general-purposes graphic editing commands.

Remarks



- Several times (at least two) we got a program crash during random, not well identified and, therefore, not reproducible, Edit operations.

– –

Edit > **Undo**

To disable the effect of the last editing action possibly executed during the current drawing session (see also Redo). Can be repeatedly entered in a Last-In-First-Out chain.

Edit > **Redo**

To re-enable the effect of the last editing action possibly disabled during the current drawing session (see also Undo). Can be repeatedly entered in a Last-Out-First-In chain.

Edit > **Move**

To ...

Remarks

- Nothing happens. Either it doesn't work at all, or it does in a way we don't know.

– –

Edit > **Delete**

To delete an entity, to be selected after calling this command.

Remarks

- Unreliable – When, before calling this command, an entity happens to be, and remain, selected somewhere in the drawing, then the command Delete-then-select process doesn't work any more, for any entity. Sometimes there is no way to regain operativity: the program turns corrupted, with this command nor working any more, also for other drawings...
- Unreliable – We've verified that if a Point happens to be selected among the entities to delete, the Edit > Delete command simply doesn't work.

- -

Draw

Point

Segment

Rectangle

Polyline

Arc

Ellipse

polygon

Fillet

Chamfer

Bisect

Text

To create and add graphic entities and graphic constructions on the current drawing.

- -

Draw > **Point**

To add a *Point* graphic entity on the current drawing, defined by: point coordinates.

Remarks

- By trial-and-error we have here "discovered" that, besides a mouse action, also numeric coordinates could be entered, on the bottom field of the Command window, with such a format: `10,20`
  Of course complete information about all coordinate entry possibilities, and related units of measurement, should be thoroughly described.

- Another aspect of basic relevance is still to be clarified: size and units of measurement of each geometric entity of the drawing.

Draw > **Segment**

To add a line *Segment* graphic entity on the current drawing, defined by: coordinates of the two extremes.

Draw > **Rectangle**

To add four line Segments forming a Rectangle, defined by: coordinates of two diagonal points. It's a construction, not a geometric entity, where each single Segment retains its identity.

Remarks

- We've realized that this Rectangle is a construction, NOT a geometric entity in itself, indeed it's made up by four distinct Segments that, as such, could be singularly deleted.

Draw > **Polyline**

To add an entity Polyline, defined by: a succession of points, ended by a return entered in the Command field.

Remarks

- At first, here we got a program crash. Not reproducible.
- As for Points (see), menu Edit > Delete has no effect on it.

– –

Draw > **Arc**

To add an Arc of a circle, defined by: center point, radius, first-last angle.

Remarks

- Unreliable, sometime it works, sometime it doesn't.
- It's a good case for investigating once for all ALL accepted ways to enter geometric values:

-- coordinate Cartesian, polar, ...
-- length measured in what unit
-- angles Degrees, Radian, ...
-- not numeric, analogical entry via mouse
-- Here the analogical entry of angles is particularly awkward...

- As in other similar circumstances, particularly in this case bits and pieces of rubber-bad graphics, meant as a graphic aid to the User, can remain randomly scattered on the scene...

- ... and with no "redraw" command available to regain a neat drawing.

- With Arch entity too, as with Points and Polylines (see), the menu Edit > Delete is ineffective.

– –

Draw > **Ellipse**

To add an Ellipse, defined by: center point, major-minor axis (NOT radius); generates a circle of the given diameter when equal.

Remarks

- At first, here we got a program crash. Not reproducible.

- Message displayed on Command field is misleading, indeed it asks for radius but means axis.

- With this Ellipse entity too, as with Points, Polylines, Arch (see), the menu Edit > Delete is ineffective.

– –

Draw > **polygon**

To add a set of line Segments in shape of a regular Polygon, defined by: center point, vertex point, number of sides, "`E/I`" flag for External/Internal (=circumscribed/inscribed). It's a construction, not a geometric entity, where each single Segment retains its identity.

Remarks

- At first, here we got a program crash. Not reproducible.

- We do not see any difference entering "`I`" or "`E`" parameter.
- A number of two sides is accepted, and executed.
- A number of side less than two is accepted, but apparently nothing happens.
- Is there any reason why this command begins with a lower-case character, instead of the usual Upper-case?

– –

Draw > **Fillet**

To ...

Remarks

- Nothing happens. Either it doesn't work at all, or it does in a way we don't know.

– –

Draw > **Chamfer**

To ...

Remarks

- Nothing happens. Either it doesn't work at all, or it does in a way we don't know.

– –

Draw > **Bisect**

To ...

Remarks

- Nothing happens. Either it doesn't work at all, or it does in a way we don't know.

– –

Draw > **Text**

To ...

Remarks

- Nothing happens. Either it doesn't work at all, or it does in a way we don't know.

– –

Modify

<void>

To ...

Modify > ***<void>***

To ...

Remarks

- Of course there is something missing here, isn't there?

– –

View

Fit

zoomWindow

To manage the portion and scale of the drawing box in display on the current Qt window.

Remarks

- Well, it's rather primordial, isn't it?

View > **Fit**

To fit all graphic elements within the display area of the current Qt window.

Remarks

- Unreliable. In some cases it doesn't work, with significant parts of the drawing left out of the window and both horizontal and vertical bars show on. We think we've found such a condition: it's for perfectly horizontal or perfectly vertical long segments, whereas with diagonal segments of the same size, it works ok...

View > **zoomWindow**

Presumably it is to zoom the display to a given drawing box area, on the current Qt window.

Remarks

- No precise information about how to define the cited box area, particularly for a zoom-out action, practically impossible.
- Is there any reason why this command is written this way, and begins with a lower-case character, instead of the usual Upper-case?

– –

Windows

Tile

Cascade

Next

Previous

To manage and arrange the set of Qt drawing child windows currently present on the main window.

– –

Windows > **Tile**

To arrange and display all Qt child windows side by side, equally sized and all within the available room on the main window.

Windows > **Cascade**

To arrange and display all Qt child windows in cascade, within the available room on the main window. One on top of the other, all equally sized and each one shifted of the width of the title bar.

Remarks

- Oddly enough the initial size of each window is set larger then the available room on the main window, with scroll bars consequently switched on.

– –

Windows > **Next**

To switch to the next Qt window as the active one, that is the child window you are currently working on. Only one window can be active at a time.

Windows > **Previous**

To switch to the previous Qt window as the active one, that is the child window you are currently working on. Only one window can be active at a time.

– –

Help

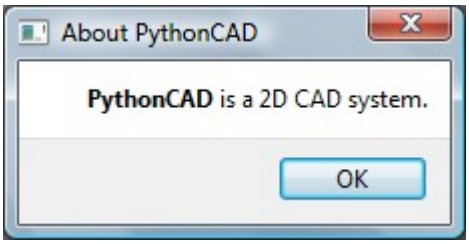
About PyCAD

To display product related information, to the User benefit.

– –

Help > **About PyCAD**

To display the following informative window about this very product.



Remarks

- A more complete info would be here welcome, at least about the current product version.

– = –

Appendix

Typographical Conventions

Some of the typographical conventions here adopted to increase readibility.

Times New Roman	Ordinary text font, throughout all these "Notes"
Arial	Titles and Keywords, usually with first letter Uppercase. That's particularly relevant to make the difference with:
	Rectangle — intended as the specific PyCAD geometric entity
	rectangle — intended in generic geometric sense
`Courier New`	Standard technical elements, such as:
	URL — `http://sourceforge.net/projects/pythoncad/`
	file path — `C:\Program Files\PythonCad`
	file name — `pythoncad_qt.exe`
Georgia	meta-text, as for header, footer and editor's notes
SPM *yymmdd*	Begin-Of-Segment delimiter, an Author and Date-code mark, useful for text updating purposes (see *Foreword*)
- -	End-Of-Segment delimiter

- = -

Contents

– = –