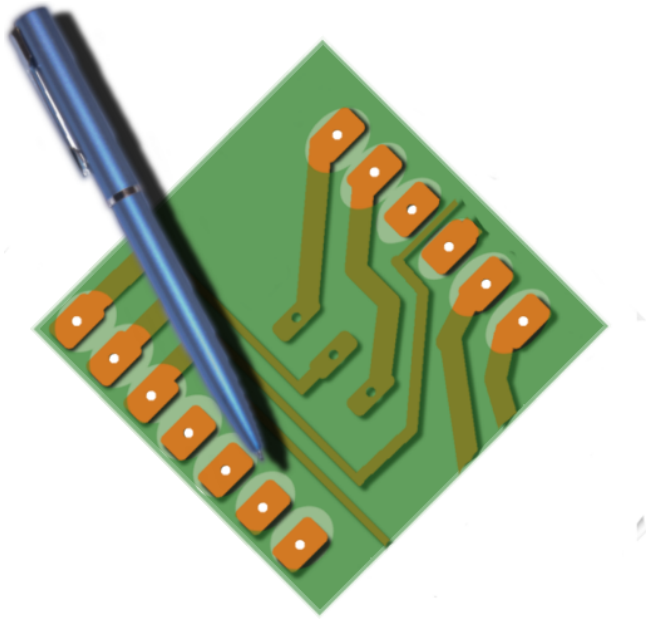
FidoCadJ 0.24.8
user manual

Davide Bucci



July 25, 2023

Abstract

This document is the FidoCadJ user manual. After a short introduction describing the history and the birth of this software, we describe the basic use of FidoCadJ. Our goal is to show how to draw very simple electronic schematics and printed circuit boards. The manual ends with a detailed description of the FidoCadJ format previously used by FidoCAD too. Finally, we give some hints about downloading and installing FidoCadJ using the most widespread operating systems: Linux, macOS and Windows.

This document presents only the program itself and the file format. It does not deal with the integration of FidoCadJ in websites, nor with the organization of the code and the aspects of development. Refer to the GitHub repository if you want to participate to the development!

Acknowledgements

A number of people used this software since its first versions and helped me by providing their advices. I thus want to thank the it.hobby.elettronica newsgroup participants for the very fruitful discussions we had together.

This software has been very carefully tested on Linux thanks to Stefano Martini's patience. He was a very attentive alpha and beta tester! I would like to thank Olaf Marzocchi and Emanuele Baggetta for their tests on macOS.

I would like to thank "F. Bertolazzi" who gave very useful advices about the usability of this software. He also assembled the CadSoft Eagle compatibility library, useful to export FidoCadJ drawings to Eagle. Many thanks to "Celsius," who tested the software functionalities for PCBs, as well as its libraries.

Thanks to Andrea D'Amore, too, for his advices concerning the FidoCadJ user interface on the Apple Macintosh. I would like to thank Roby IZ1CYN for the discussions about libraries and for having written section A.2 of this manual, dealing with its installation on Linux. Thanks to Max2433BO for advices about Linux and for his patient work on GitHub issues.

Macintosh users can use FidoCadJ well integrated in the look of their operating system thanks to the Quaqua look and feel, provided by Werner Randelshofer and more recently thanks to the VAqua7 look and feel. Werner has also given useful advices about FidoCadJ's user interface: thanks for that!

Part of this manual has been translated to English by "Pasu". I would like to thank him for his hard work. I would like to thank Miles "qhg007" as well, for the careful check and very useful remarks. The English manual has been translated into Dutch by Joop Nijenhuis as a thank you for FidoCadJ. The Dutch manual has been translated into German with the help of Google, among others.

In April 2010, FidoCadJ has been integrated in the well known Italian web site www.electroyou.it. It now silently runs on the server and it automagically converts drawings posted on the forum. I would like to express my gratitude to the admin Zeno Martini and webmaster Nicolò Martini and to all users of this site. They gave me a lot of very useful ideas about batch controlling FidoCadJ: a very promising path has been traced that deserves to be fully explored. FidoCadJ is also adopted on http://www.matematicamente.it. It has been used by the popular www.grix.it website thanks to the efforts of Arniek, Sstrix and Stan. Many kudos to them!

Useful hints about FidoCadJ's website came from Federica Garin and Emanuele Baggetta. Sergio Juanez also greatly contributed to it, improving technical aspects and organization.

FidoCadJ License

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we will briefly introduce FidoCadJ. In particular, we will give a description of the philosophy behind this software, as well as a brief history of its development. If you need to install and run FidoCadJ, jump to appendix A, that describes how to install FidoCadJ on the most widespread operating systems.

1.1 FidoCadJ's philosophy

FidoCAD (without the J at the end) was a vector drawing software particularly suited for electrical schematics as well as simple printed circuit boards. It was employed by the italian Usenet community during the late 1990s.

It can still be freely downloaded (a Windows version nationalized in Italian language) from Lorenzo Lutti's page:

http://www.enetsystems.com/~lorenzo/fidocad.asp

The output files generated by FidoCAD was a very compact text code. This feature made it very easy to include drawings in text messages, such as those used in non-binary Usenet groups.

Unfortunately, FidoCAD existed only in a Windows version. Linux users could run it using WInE, but those relying on other platforms like me (I use macOS) had to find a different solution. I thus decided to give a small contribution to the community by writing FidoCadJ (with the final J, this time). This editor is written in pure Java and it is completely multi-platform. FidoCadJ allows showing and modifying a drawing using the FidoCAD file format. Nowadays, I am not alone anymore working on the project and FidoCadJ is a full-fledged and I dare to write successful Open Source project, with contributions coming from all over the world.

Whoever used FidoCAD in the past should become acquainted with FidoCadJ very quickly, since many commands and procedures are quite similar to the original application. FidoCadJ is almost completely compatible with the original FidoCAD, apart from a few details. The goal of a complete compatibility has been pursued until possible, but the needs of the new users have pushed towards some extensions to the original format.

Among the features offered by FidoCadJ and not present in the original FidoCAD are the export possibilities offered. Since I am a LaTeX user, I decided to include an export feature for a number of vector formats, including Encapsu-

lated PostScript (EPS) and scripts for the PGF package. Of course, FidoCadJ can export towards the very well known PDF format, with some limitations.

1.2 History of FidoCadJ

I have long been interested in electronic circuits. When I began following several dedicated Italian Usenet newsgroups, I noticed that many schematics were provided using the FidoCAD for Windows format. This was much better than awkward ASCII drawings. Since I do not use Windows, it was almost impossible for me to look at them and I wanted to try to do something to solve this problem.

By the way, I think that it is better to actively work at a solution rather than whining, because an operating system different from Windows lacks a particular software.

The first thing I did towards the end of 2006 was to study the file format used by FidoCAD and write a Java applet called FidoReadJ. It was able to parse the circuit and show it in a web page. The internet was different back then! To do that, I searched more or less everywhere (old posts, web pages), and did a lot of reverse engineering from existing FidoCAD files. I downloaded and studied the FidoCAD sources, too, written in a pretty clear C++ by Lorenzo Lutti.

I wrote the core of the applet about in March 2007. A few months later, the applet was on line and being tested by part of the community gravitating around it.hobby.elettronica and it.hobby.fai-da-te.[1]

To be honest, I made a first attempt at writing a 2D vector drawing system around 1993.

Since I had an interpreter of the FidoCAD format, it was interesting to obtain a complete editor. Much of the initial work was done in several steps, between January and July 2008: FidoCadJ is not a port of FidoCAD for Windows, but it is a new program, rewritten from scratch.

The choice of using Java is due to the fact that I currently use many different operating systems. Spending time and energy on something that is not completely portable does not appeal to me anymore. The effort of learning the Cocoa framework would have probably given a better result on macOS, but it would have made FidoCadJ completely non-portable. I am not a computer guy. The time I spend to program is time taken away from my electronics interests. In fact, especially at the beginning, the FidoCadJ code source showed that I am not a very Java and object oriented programming purist and several solutions were more practical than elegant. However, I tried to improve continuously the overall code quality FidoCadJ. It has been quite successfully compared to other Open Source projects at least in one occasion[2]. Continuing in this way, static and dynamic analysis of the source code has been utilized to suggest where to refactor and improve the code organization.

What matters are the impressions of those who use the program, much more than the choice of a particular language. For this reason, I am always listening to your suggestions, in order to understand how to further develop this project. Java may not the perfect choice or the solution to every problem. However it has proven to be flexible enough for a tool like FidoCadJ.

[1] FidoReadJ is still available today at the following address, in the unlikely possibility that your web browser still supports Java applets:
http://davbucci.chez-alice.fr/index.php?argument=elettronica/fidoreadj/fidoreadj.inc.

[2] See for example Rahman, M. M., Roy, C. K., and Keivanloo, I. "Subjective Evaluation of Software Quality Using Crowdsource Knowledge: An Exploratory Study". Univ. of Saskatchewan, Department of Computer Science technical report 2013-01

Without aiming for perfection and knowing the limits of my programming skills, my intent is to make sure that FidoCadJ will NOT be another poor quality application. For this reason any bug report or comment on the program's usability will be more than welcome.

In November 2009, I opened a SourceForge project dedicated to FidoCadJ. From this place, you could download all executables, manuals as well as the source code. Unfortunately, SourceForge progressively adopted some shady practices since 2013: advertisements with fake download buttons appeared everywhere and crap-ware added to installers for some projects. In 2015, the site was down for about 15 days: it become clear that it was time to change and we migrated the FidoCadJ project on GitHub. From August 2015, all development on FidoCadJ is done using Git: feel free to fork FidoCadJ and have a look at the sources. The SourceForge page remains, though, as someone still likes to load packages from there (but the source code is not up-to-date).

A deep refactoring of the existing code has been done by Kohta Ozaki and myself between 2013 and 2014, then gradually continued in 2015. This allowed to start porting the program towards Android platforms, which was made possible by the contribution of other people, in particular Dante Loi. Many other persons have given a hand in that moment. Even if the port was successful and a pre-release beta was ready and distributed, the work on Android unfortunately stalls since 2016.

If you want to help the community with FidoCadJ, do not worry: you do not need to be an expert Java programmer. You can for example translate the user interface or the manuals in a new language, check carefully the documentation for inconsistencies, typos and errors. FidoCadJ is currently available in Italian, French, English, Spanish, German, Chinese, Japanese, Greek, Czech and Dutch. Some translations may not be up-to-date anymore and require some work from the community. Search among the Issues on GitHub (with the "translation" tag) to see what is needed. It is quite easy to translate menus and commands, so if you would like to see FidoCadJ in your native language, please create a new Issue on GitHub!

There is also work to be done to check the libraries (at least the standard ones) and of course this manual.You can also work on the standard library... I consider that only half time I dedicate to FidoCadJ is in reality dedicated to the coding itself. The rest is spent answering questions from users, writing and improving the documentation and things like that. On GitHub, you can participate to the discussions, as well as suggest improvements or give a bug report. Don't hesitate to participate:

https://github.com/DarwinNE/FidoCadJ

1.3 FidoCadJ and the future

In April 2010, I stumbled almost accidentally on a thread in a well known italian website www.electroyou.it. Piercarlo Boletti proposed there to integrate a schematic capture tool directly inside the forum, as something similar has already been done with LaTeX equations. Since FidoCadJ was cited in his intervention, I created an account to write to the forum and I offered to collaborate. I interacted with the very nice webmaster for a few days, and the system was ready: a simple copy and paste of the FidoCadJ code and a couple of tags. The forum runs automatically FidoCadJ on its servers and obtains an image, directly from the code. This image is then shown in the forum post, but its source code always remains available. This way, who is reading the discussion does not need to have anything installed, but if one desires to change something, the code can be obtained with a few clicks.

Figure 1.1 shows part of one of my posts done on www.electroyou.it. This system is so powerful and flexible that I have been the first to be astonished of the enthusiasm demonstrated by the users![3]

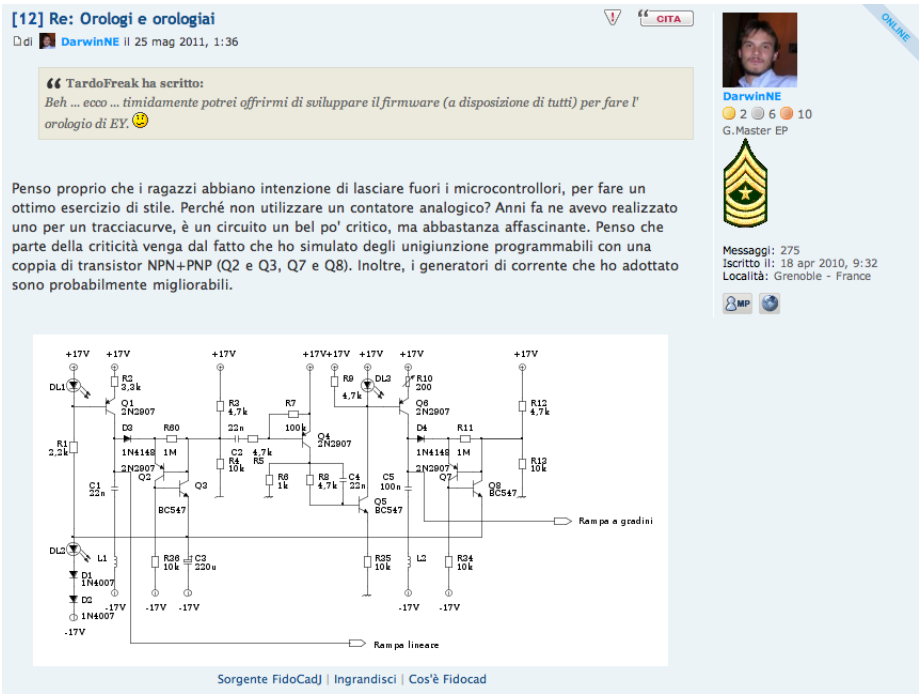


Figure 1.1: Part of one of my posts on www.electroyou.it. With just one click you can zoom on the schematic. With a second one, you can obtain immediately the source code that you can paste on FidoCadJ to modify it.

[3] If you read Italian, here is an article I wrote:
http://www.electroyou.it/darwinne/wiki/fidocadj

FidoCadJ has shown to be useful also for drawings not directly related to electronics (see appendix B and C to see what I mean).

The success that FidoCadJ had on www.electroyou.it stimulated some requests to implement a similar system on other platforms, and in particular on www.grix.it, another well known Italian electronics-dedicated website. FidoCadJ is also used by the Matematicamente community. In the case of www.grix.it, the server could not run a Java program and so the FidoReadPHP class has been written. It can be run on a PHP interpreter to obtain the images from the FidoCadJ source code. The FidoReadPHP project is open source and it is available on SourceForge:

https://sourceforge.net/projects/fidoreadphp/

but it is unfortunately not currently actively maintained. The result is somewhat similar to what it has been obtained on www.electroyou.it, even if the graphical capabilities of PHP are quite limited with respect to Java[4]. Figure 1.2 shows an example of a drawing obtained with FidoReadPHP.

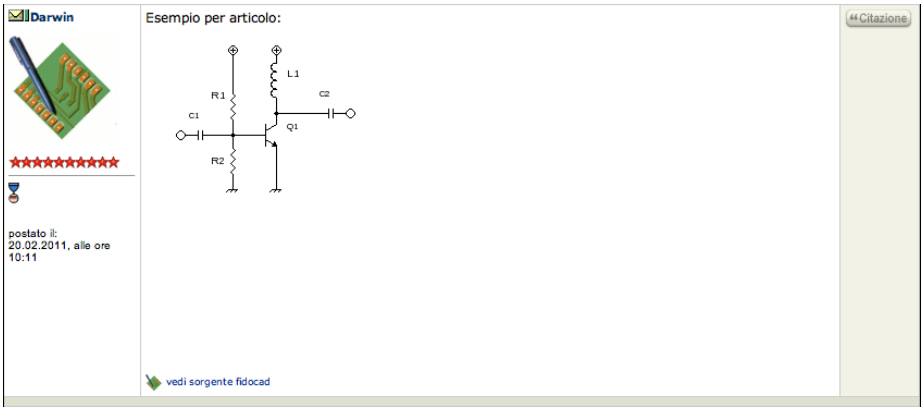


Figure 1.2: An example of a FidoCadJ drawing integrated in a www.grix.it forum post. You can obtain the drawing source code with just a click.

I think that the future of FidoCadJ is not to become more complex, as a complete CAD for electronics. Instead, integration with discussion groups and forums deserves to be developed. Experience has shown that this is possible and the enthusiasm of the users has been tantalizing.

[4]Once again, if you read Italian, here is an article:
http://www.grix.it/viewer.php?page=9335

Chapter 2

Drawing with FidoCadJ

Expert Mac-users will notice that all the menus are at their own place!

Using FidoCadJ should be quite intuitive for who already used a vector drawing application. A screenshot of the program running on macOS is shown in Fig. 2.2; A few details may be different when running on other operating systems (for example Fig. 2.3 shows the result with Look and Feel Metal on a Raspberry Pi 400), but the philosophy remains the same. We will describe the features of the program, as well as its basic elements (the primitives), which compose a FidoCadJ drawing. Below is an image showing the three menubars and how they are displayed in FidoCadJ.

Figure 2.1: The three menubars of FidoCadJ 0.24.8.

2.1 Drawing Tools

This behavior in user interfaces is usually called "radio buttons" and is inspired to the old vacuum tube radios and to the switches that were fashionable until 1970s.

In the toolbar at the top of the window, we can find the most used features for the creation and the editing of a drawing. Table 2.1 shows a brief summary of the functionalities, the commands and describes the possible actions. You will notice that once a button in the toolbar is depressed, it remains in that position until another function from the toolbar is selected. From the toolbar, we can select which drawing primitive will be used[1]. In the workspacebar on the right, a drop-down menu shows the current working layer (see 2.3 for more information).

The toolbar can be partially customized. In particular, we can choose whether we want to see the icon on each button or the icon and its text description. Icons are also available in two selectable formats. To change these settings we can select in the menubar the menu "File/Options"[2]. Such changes

[1] For more information on the drawing elements, see 3.3.

[2] Except on macOS, where this item is found in the FidoCadJ's menu and it is called "Preferences".

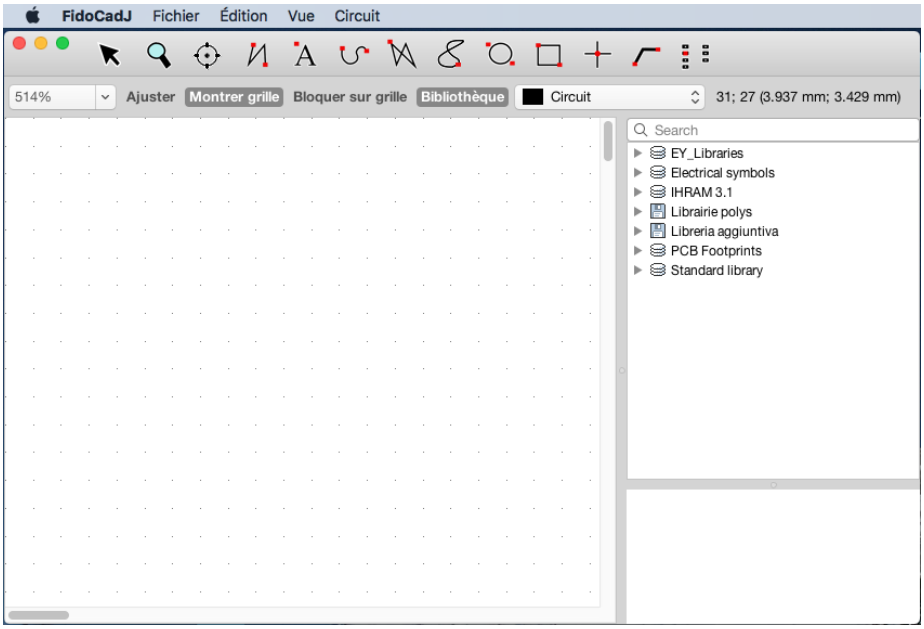


Figure 2.2: A FidoCadJ 0.24.8 session running on macOS 10.13 High Sierra. Appendix A describes the peculiarities of the version for Macintosh computers.

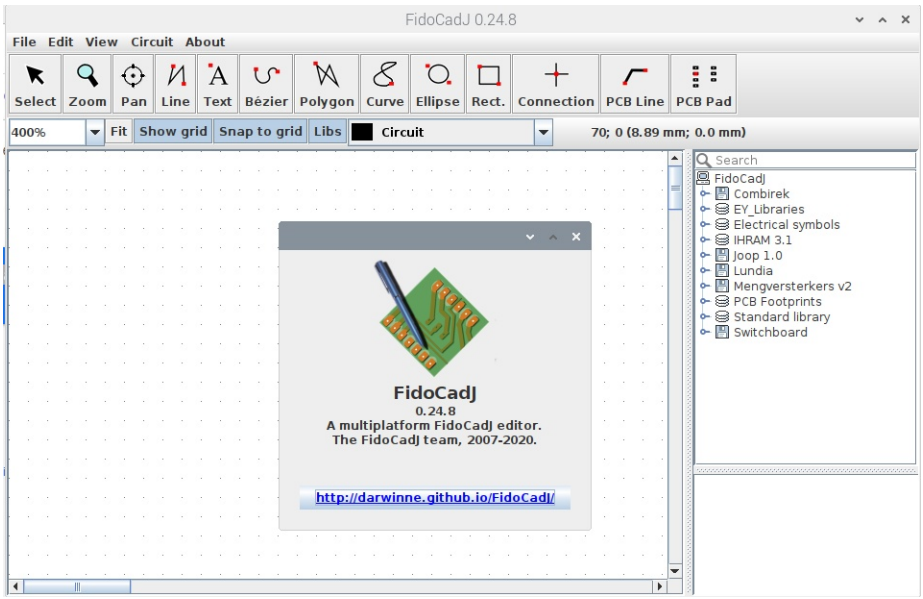


Figure 2.3: A FidoCadJ 0.24.8 session running on a Raspberry Pi with the Look and Feel Metal.

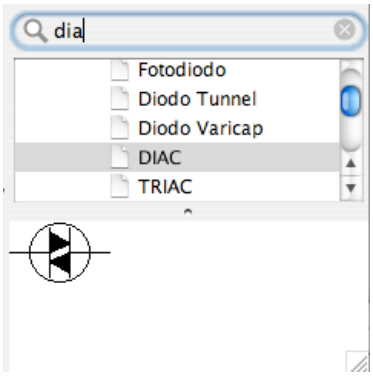


Figure 2.4: The quick search function in the installed libraries.

in the settings become effective after a restart of the application, since it is arguably not something that we may want to change every day. Figures 2.2 and 2.3 show the toolbar, configured to show text and icons in their smallest version. In the Workspacebar we can see the zoom settings and the buttons "Fit", "Show grid", "Snap to grid" and "Libs". The first one allows to automatically select the most suitable zoom settings, in order to show the whole drawing on the screen. The second toggles between a visible and invisible grid. The third button controls the snapping, i.e. the automatic alignment of elements to the grid. The last one allows to toggle the visibility of the library panel on the right of the window. If you need to align carefully the elements, you may find useful to keep pressed Alt, while using the cursor keys: the selected objects will be moved of a single unit in the direction you want.

On the right part of the FidoCadJ main windows, the symbols (also called macros) are shown in the loaded libraries, shown as a tree. To insert in the drawing an element from the library, select it from the list an click on the drawing. FidoCAD libraries include all standard symbols used in electrical schematics and a wide selection of footprints for drawing PCBs.

You can also do quick research inside the libraries. Just type something in the text field that appears just over the library tree (have a look to figure 2.4). You can navigate through the results typing return.

Figure 2.5 shows an example of what can be obtained by double-clicking, in selection mode, on a drawing element (in this case a text string). Within this window it is possible to modify all the parameters (coordinates, rotation...) of any drawing element. The aspect of this window depends on the selected element.

Table 2.1: Summary of the drawing commands available in FidoCadJ. The key shown on the leftmost column allows a rapid selection with the keyboard. Right click in one of the primitive placement modes allows to access its properties.

Key	Command	Use
A, ESC or space	SELECT	Select one or more graphic elements. Press Control (Command on macOS) for multiple selections or to deselect only one element. Click and drag to select several elements in an area. Press R to rotate the selected elements. Press S to mirror the selected elements. Double-click on an element to modify its properties.
	ZOOM	Left-click to increase the level of zoom. Right-click to decrease it.
	MOVE	Click on the drawing and move the mouse to move the drawing.
L	LINE	Insert a line or a series of lines. Press Esc or double-click to terminate the insertion
T	TEXT	Insert a text string.
B	BÉZIER	Draw a Bézier curve
P	POLYLINE	Draw a polyline filled or empty. Double-click or press Esc, to terminate the insertion of new vertices.
K	CURVE	Open or closed natural cubic spline curve. Double click or press Esc, to terminate insertion of new ver tices.
E	ELLIPSE	Draw an ellipse filled or empty (hold Control or Command in macOS for a circle).
G	RECT.	Draw a rectangle filled or empty (hold Control or Command in macOS for a square).
C	JUNCTION	Insert an electrical junction.
I	PCB TRACK	Draw a PCB track. The default width can be modified through the dialog accessed from the menu "File/Options".
Z	PCB PAD	Draws a PCB pad. The default size can be modified through the menu "File/Options".

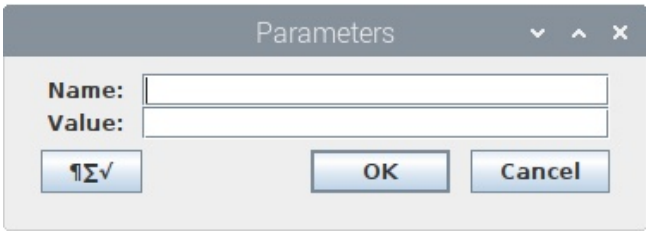


Figure 2.5: Dialog for the text's parameters in a FidoCadJ drawing. The operating system is macOS Yosemite (10.10.4), in French. Depress the button on the bottom left to enter symbols or to have some hints.

2.2 A simple schematic

As an example of use of this application, we will show how to draw the simple electrical schematic of figure 2.6.

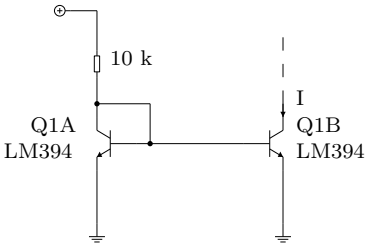


Figure 2.6: The reference schematic: a current mirror made with NPN transistors.

Once FidoCadJ is running, create a new drawing via the menubar and menu "File/New". We then start by placing in the drawing area the two transistors around which our schematic is built. To do so, we need the symbols (aka "macros") contained in the standard library, which is loaded by default and is found on the right-hand side of the window. The symbol that we use is called "NPN transistor" and is included in the "Diodes and transistors" category of the "Standard library". By clicking on the element's name to select the desired symbol, it will then be possible to place it anywhere in the drawing (FidoCadJ shows a preview), by clicking a second time on the desired location. We should now be at a stage similar to the one shown in figure 2.7.

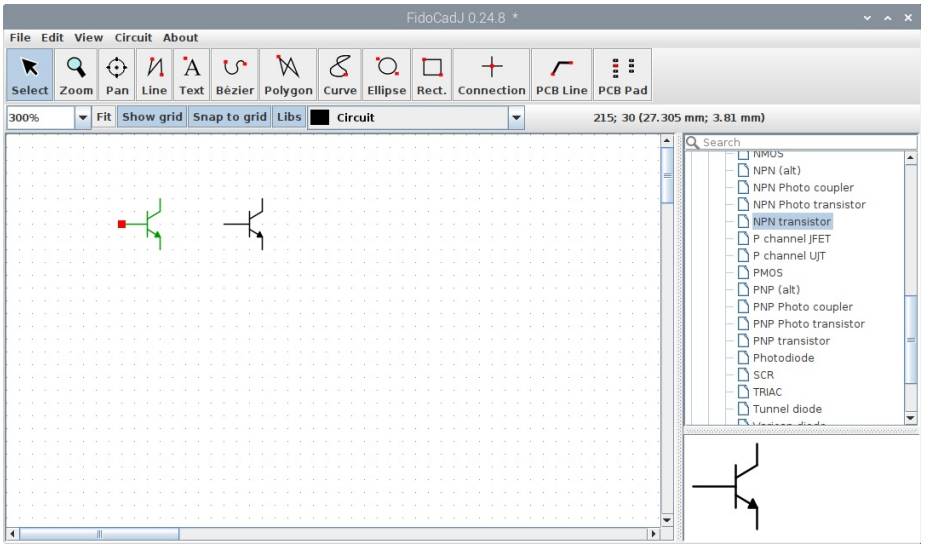


Figure 2.7: We start by drawing a couple of transistors.

We may notice that the bipolar transistor on the left is not correctly oriented. To fix the problem is sufficient to click on "Select", from the toolbar, click on the transistor (that will be highlighted in green, with three control points identified by small red squares) and then press S to obtain its mirrored version. You may also press S when you have selected the symbol in the libraries and you are about to position it in the drawing. We thus obtain a result similar to the one shown in figure 2.8.

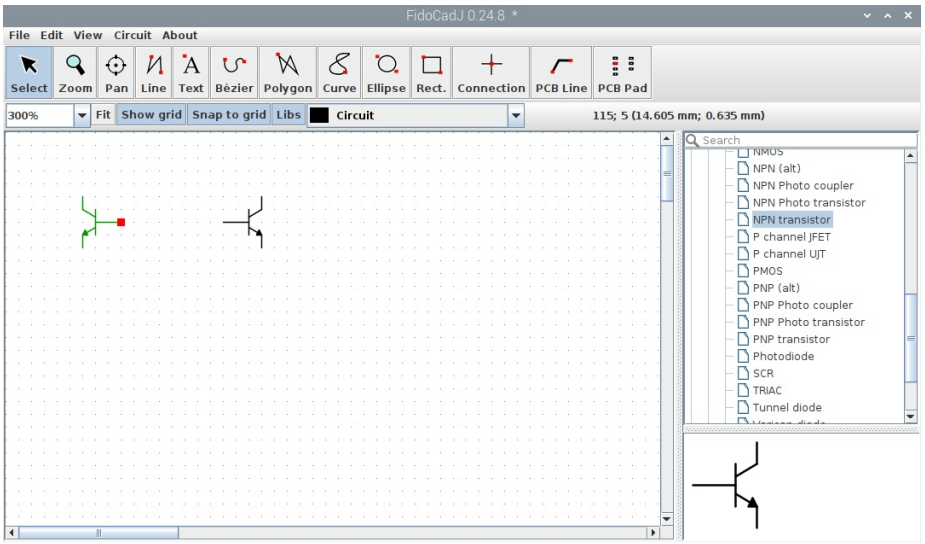


Figure 2.8: Select and mirror the transistor on the left by depressing S.

By using the tool “Line” from the toolbar, we can draw a few electrical connections, until we realize that we started our drawing to close to the edges of the drawing area. The can be easily fixed by selecting the whole drawing: in “Select” mode, we can click on the upper left corner of the drawing and, holding the left button of the mouse depressed, drag the cursor up to the lower right corner. A rectangle with a green contour appears to indicate that we are trying to select all the elements included in it. Since we want to move everything we have drawn so far, we have to select them all first(see figure 2.9). Now, still in select mode, we can click on any selected element to drag the selection to the desired position.

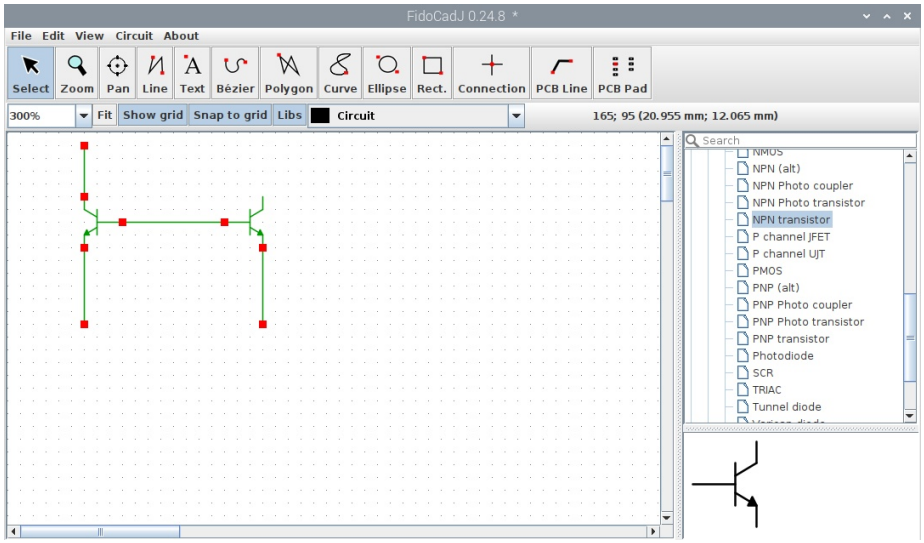


Figure 2.9: We are too close to the top edge of the sheet: select the whole drawing and move it toward the centre.

We can then continue placing the other parts of the circuit, a resistor (Standard Library/Discrete devices/Resistor) and the label for the positive power supply (Standard Library/Basic symbols/Terminal +). We need to rotate the latter in order to place in the desired position. We can select it and press R until we obtain the desired result. We should now have a drawing similar to the one in figure 2.10.

To complete the schematic we only need to add the text strings and the arrow to indicate the direction of the current. For the latter there is a symbol called “Arrow”, contained in “Standard library/Basic symbols”. To place the text, we can press the button “Text” from the toolbar e click in the drawing area on the desired position. As the default text “String” is introduced, we need to modify its characteristics by double-clicking in select mode (see figure 2.5). The transistor’s part number and name used are specified in the fields “Name” and “Value” that are accessed in select mode by double-clicking on the symbol. The feature that allows to add a name and a value to an element is an extension introduced with FidoCadJ . See section 3.4 for more information on compatibility. The suggested dimension to work with electrical circuits is 4 units in vertical and 3 units in horizontal. The complete circuit is shown in figure 2.11.

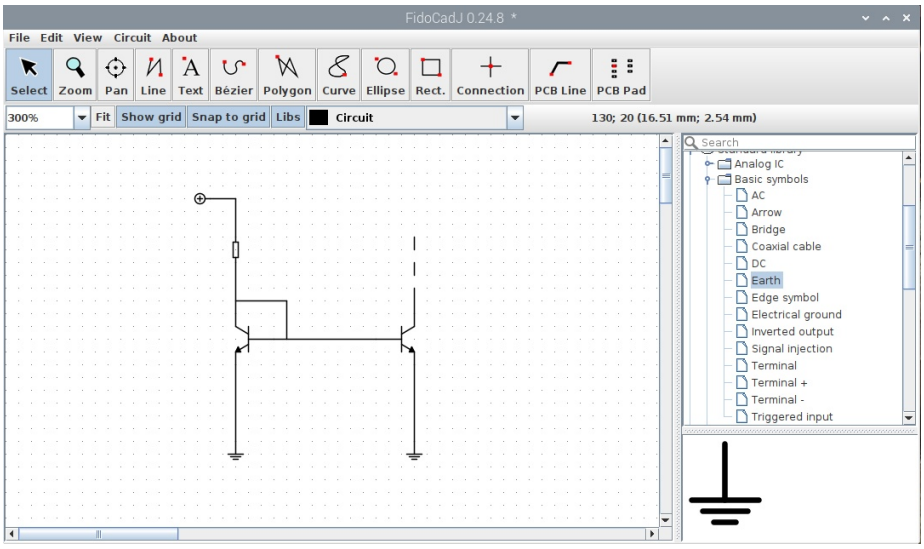


Figure 2.10: The circuit almost completed.

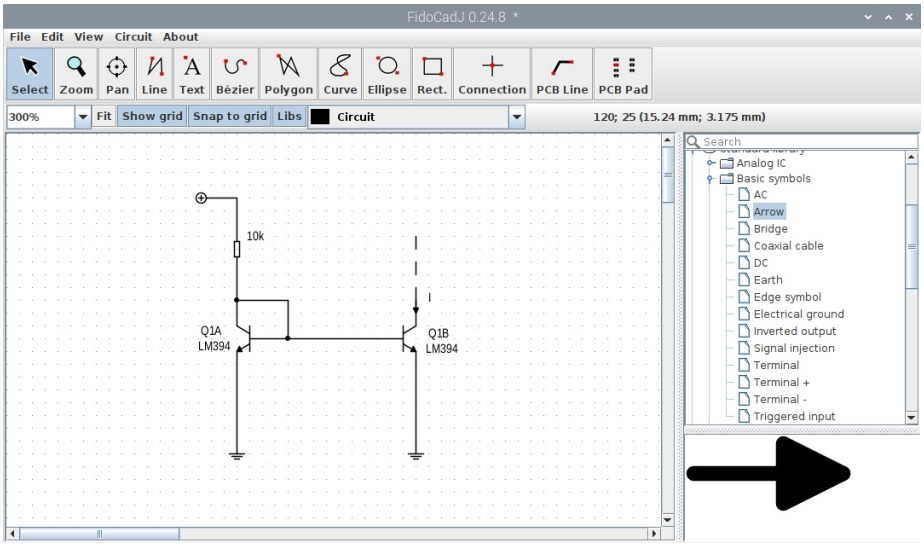


Figure 2.11: The final circuit.

Table 2.2: This code describes the circuit from our example.

```
[FIDOCAD]
MC 95 65 0 0 280
FCJ
TY 115 60 4 3 0 0 0 * Q1B
TY 115 65 4 3 0 0 0 * LM394
MC 55 65 0 1 280
FCJ
TY 20 60 4 3 0 0 0 * Q1A
TY 20 65 4 3 0 0 0 * LM394
LI 55 65 95 65 0
LI 40 75 40 95 0
LI 110 75 110 95 0
LI 40 40 40 55 0
MC 40 30 0 0 115
LI 40 15 40 30 0
LI 30 15 40 15 0
MC 30 15 2 0 010
LI 40 50 60 50 0
LI 60 50 60 65 0
SA 60 65 0
SA 40 50 0
LI 110 45 110 55 0
LI 110 35 110 40 0
LI 110 25 110 30 0
MC 40 95 0 0 040
MC 110 95 0 0 040
TY 45 30 4 3 0 0 0 * 10 k
TY 115 50 4 3 0 0 0 * I
MC 110 50 1 0 074
```

As a curiosity, this is how the code that describes the circuit of our example looks like. To access this code, it is sufficient to select "View code" from the menu "Circuit". We are now ready to copy and paste our circuit on an e-mail message, in a newsgroup, or in a forum post.

If you are interested in the export format used by FidoCadJ, there is a detailed description on chapter 3. However, there is no need to use the "View code" dialog; you can simply select the entire drawing, copy it (by selecting "edit/copy" or by pressing Ctrl+C) and paste it onto the message we are writing and the code is added automatically. FidoCadJ usually applies a diagonal shift to the copied elements; the x and y width and height are equal to the grid pitch. The default behaviour is to proceed in such a way, in order to differentiate the pasted elements from the original ones. Since in some situations this can be problematic, this setting can be modified in program's options.

2.3 The layers

A way to picture layers is thinking of a drawing made on superimposed acetate sheets. Every layer is characterized by a different color and can be visible or hidden. This approach is common in many CAD packages, as it allows an easy representation and management of different parts of the drawing that are superposed, in a PCB design for example. FidoCadJ allows up to 16 layers, numbered from 0 to 15. Conventionally, some of the layers have a specific purpose. Layer zero is customarily used for electrical schematics, layer 1 for the copper soldering side, layer 2 for the copper components side and layer 3 for the silk-screen. The remaining layers do not have any pre-defined purpose and can be used freely. Name and color of every layer can be specified using the menu "View/Layer". From the same menu, we can also select the layers that we want to see on screen or that we want to print. The layers' ordering is important, as layer with a lower number are drawn earlier: elements on successive layers cover the ones already drawn.

2.4 The grid

The logic unit in FidoCadJ is 5 mils (127 micron) and "half units" are not allowed, meaning that the coordinates of any graphic element must be integer numbers. This allows to obtain a resolution sufficiently fine to draw an electrical schematic and the majority of PCBs. However, for ease of drawing, the application allows to set a coarser grid and force the mouse to align with the nearest point of the grid. To enable this functionality there are two buttons in the Workspacebar, "Show Grid" and "Snap", which allow toggling between visible/hidden grid and force the mouse cursor on the grid or let it move freely, respectively. The grid step can be chosen through a dialog window accessible from the menubar and the menu "File/Options" and "Drawing tab".

2.5 A simple PCB

A few scribbles made with paper and pencil (and a lot of erasers) result in time saved by obtaining a clear idea to be developed with the use of the computer.

To practice what we learnt so far, we will see how to design a simple PCB. Unlike other software for electronic CAD that are very powerful but sometimes quite hard to manage, FidoCadJ provides an electronic version of the good old R41 transfers, or the tape on mylar sheets. Obviously, working on a computer allows to benefit from all the flexibility offered by the machine.

It must be noted that the design of a PCB, especially if this is a complex one, is not an easy task. The autoplacer and the autorouter features promise miracles on the publicity flyers from major CAD companies, yet is no doubt that these are still tasks where the user's experience still plays an important role. FidoCadJ constitutes a very immediate and fast way to draw small PCBs on a DIY scale. This approach is very simple, but if you need to design a very complex PCB, FidoCadJ is NOT the tool for you: use for example the excellent KiCad, instead: the additional complexity is justified. We will see here how to draw a very simple, yet complete, one.

I would suggest to start with a clear idea of where to place each component

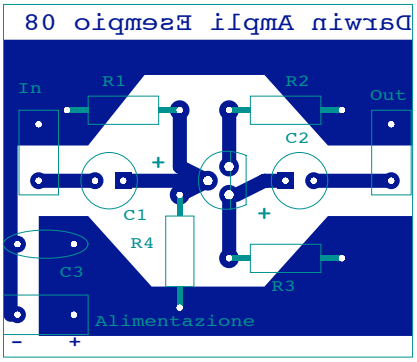


Figure 2.12: A very simple amplifier stage using an NPN transistor connected in a common emitter configuration.

and how to draw all the tracks so that they cross each other as less as possible.

Here we cheat a little and we start from the result we want to obtain, as shown in figure 2.12. It is a simple common-emitter amplifier built around an NPN transistor, type BC547 or similar. It is useful to think of the board as it was transparent, by looking from the components side. For this purpose the silk-screen with the components drawing is very useful, although probably we do not want to print it out onto the actual board for a DIY project. The first thing I would suggest to do is to place all the components as best as possible. In our example these would be the transistor (from the library "PCB footprints/3 terminals semiconductors/TO92"), the resistors ("PCB footprints/Resistors/Resistor 1/4 W 0,4 i"), and the electrolytic capacitors ("PCB footprints/Electrolytic capacitor/Vert. diam. 5 mm 2.5 mm pitch"). In order to delimitate the board, it may be useful to place an empty rectangle on the silk-screen layer (layer No. 3). To do so we can use the *rectangle* primitive making sure that we selected the appropriate layer first. We should get a result similar to the one shown in figure 2.13.

We can then introduce the copper areas they provide the positive and negative power supply. These are specified by drawing a polygon (using the *poly line* primitive) and double clicking near its border in selection mode to tell FidoCadJ (through dialog box) that we want a filled polygon. Before placing the polygon, make sure that the current layer is the one where we want to place the copper area (layer No 1, or copper solder side). The use of a filled area for the power supply lines may be useful to ensure that these connections have a low stray inductance. We should be able to reproduce the result shown in figure 2.14.
To complete the electrical connections we can use the *PCB line* primitive. I chose a track thickness of 10 units (1.27 mm), which helps the soldering process, see figure 2.15. We realize that we need a few connectors: one for the input, one for the output and one for the power supply. We can use the footprint designed for a polyester capacitor. It probably has the right dimensions. Let us not forget that FidoCadJ is thought as a replacement for the transfers...

Be careful with the track widths: something that looks like a motor-way on the screen will probably be a track so thin that will detach from the board during soldering.

We can also place a "+" and a "−" on the copper layer and place a ceramic capacitor in parallel with the power supply. Let us put the writing on the top

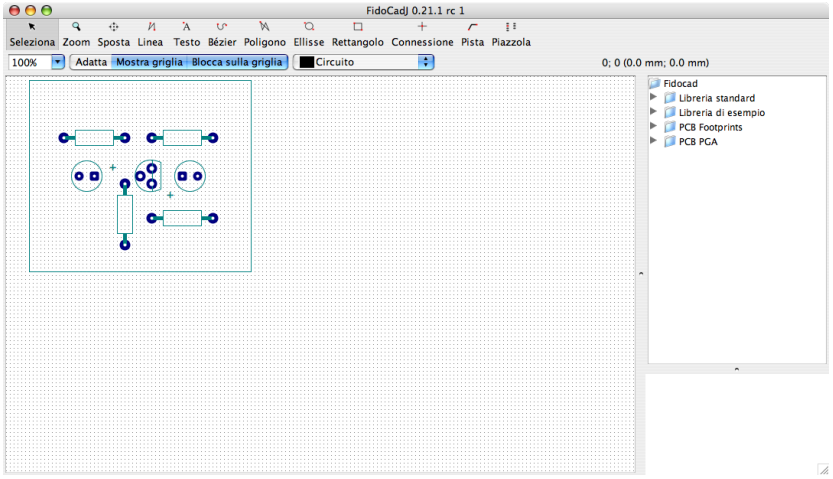


Figure 2.13: The most important devices are placed on the board.

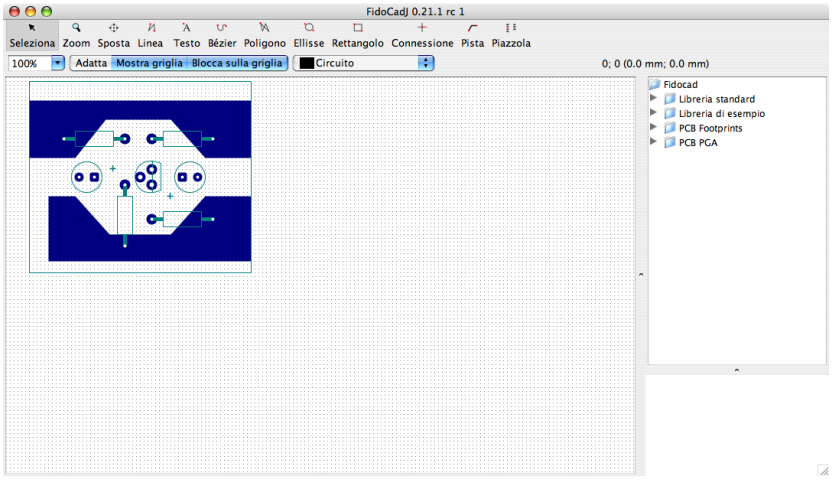


Figure 2.14: Added the power supply connections using polygonal lines.

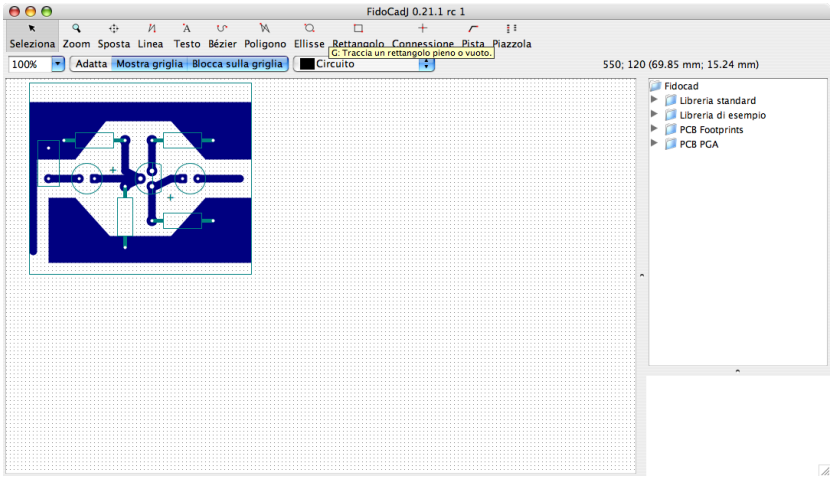


Figure 2.15: Added the remaining connections PCB tracks.

side as well. To write on the copper layer, after selecting the proper layer it is also necessary to mirror all the writings. This is easily done within the usual properties dialog accessed by double-clicking on the string we want to modify when we are in selection mode. You may need to experiment a bit, to obtain the right size for the characters. To give an idea it is useful to have a ratio of 3/4 between the horizontal and the vertical dimensions of the characters. Figure 2.16 shows the result obtained using 11 units for the horizontal and 18 units for the vertical dimension of the text.

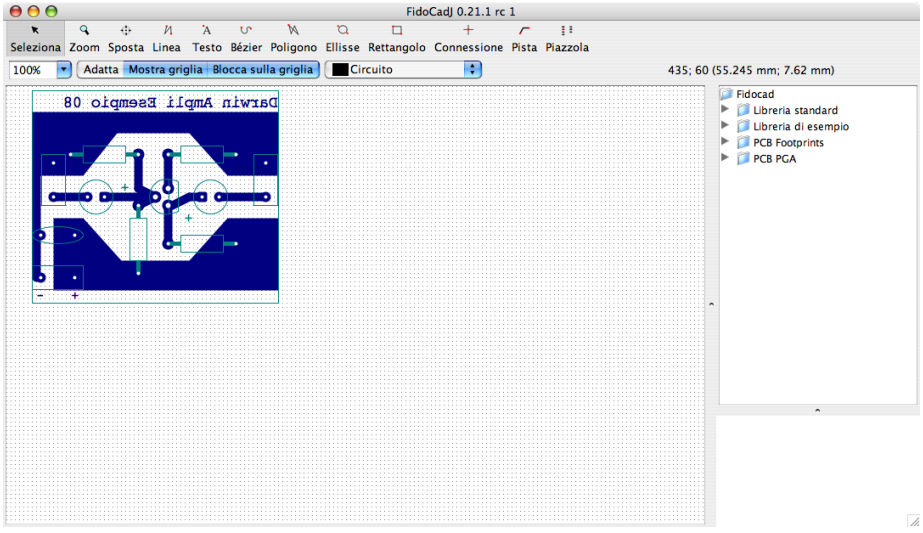


Figure 2.16: The PCB almost completed.

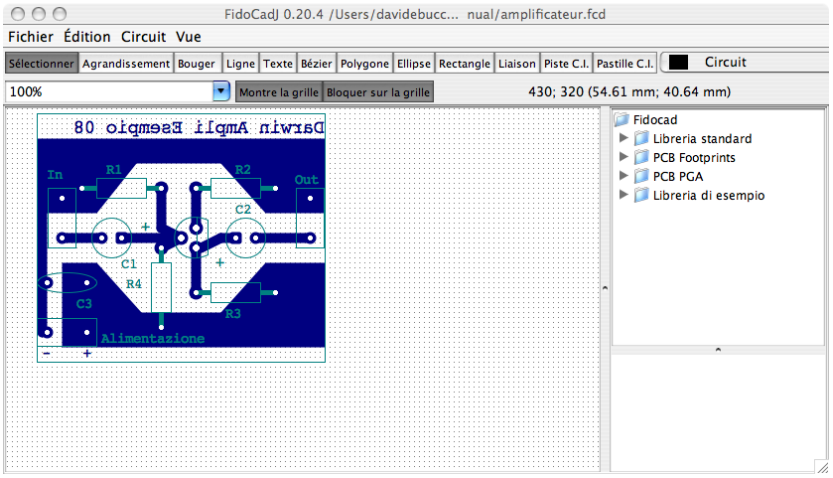


Figure 2.17: The job completed with the silk-screen.

At this stage the only thing missing is the text with the name of each component, that can be placed on layer 3 (silk-screen). The program with the PCB completed is shown in Figure 2.17.

Once the drawing is completed, we probably need to print it either on a transparency to use it with a printing box or to use it with other methods such as the "Press&Peel". To do so we need to make invisible all the layers that we do not want to print. This is done from the dialog window accessed from the menu "Vista/Layer". In our case it is sufficient to hide the layer 3 containing the silk-screen. The program shows the copper layer only.

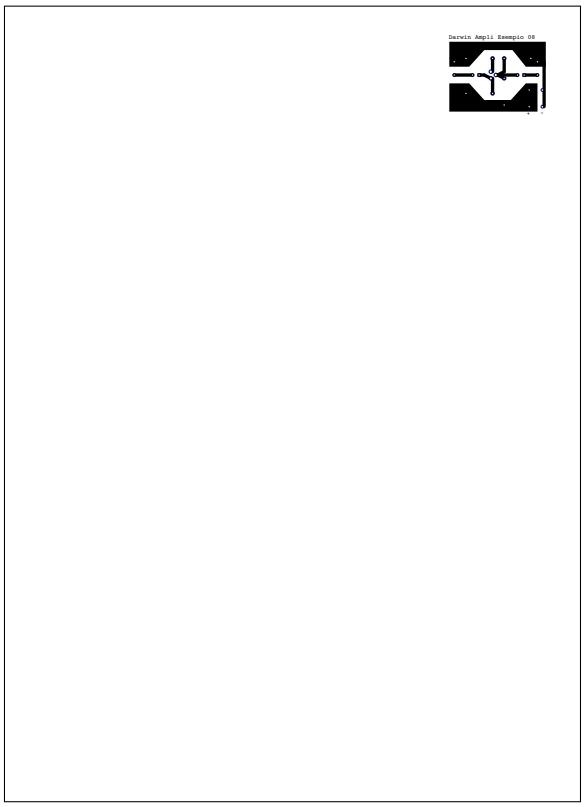


Figure 2.18: The PCB, as it appears when printed (mirrored) on a ISO-UNI A4 sheet.

We then print everything that is shown on the screen (obviously NOT adapted to the size of the page, as we want to have the dimensions set in the drawing), taking care of selecting the black and white printing to ensure the maximum contrast. It may be useful to mirror the whole drawing, depending on the technique chosen for the production of the PCB. Since our PCB is quite small, in its real dimensions it occupies only a small corner of the sheet (assuming a standard ISO-UNI A4), as we can see in Figure 2.18.

Table 2.3: This code describes the PCB from our example.

For information, below is the code generated for the PCB of the example above (be careful with long lines!):

```
[FIDOCAD]
TY 320 10 18 11 0 4 1 * Darwin Ampli Esempio 08
TY 85 240 12 8 0 5 1 * +
TY 44 239 12 8 0 5 1 * -
PL 35 90 35 225 10 1
PL 55 130 95 130 10 1
PL 250 130 305 130 10 1
PL 215 130 230 130 10 1
PL 195 140 215 130 10 1
PL 115 130 175 130 10 1
MC 155 220 3 0 PCB.R01
MC 75 80 0 0 PCB.R01
MC 270 185 2 0 PCB.R01
MC 270 80 2 0 PCB.R01
MC 230 130 3 0 PCB.CE00
MC 115 130 1 0 PCB.CE00
MC 40 175 0 0 PCB.CC50
PL 190 80 190 120 10 1
PL 190 140 190 185 10 1
PL 155 80 155 120 10 1
PL 155 120 175 130 10 1
PL 155 140 175 130 10 1
PP 30 30 30 105 90 105 130 55 215 55 260 105 320 105 320 30
    1
PP 320 240 320 155 260 155 215 205 135 205 90 155 55 155 55
    240 1
MC 190 120 0 0 PCB.TO92
MC 305 90 1 0 PCB.CPBX352
MC 55 90 1 0 PCB.CPBX352
MC 80 225 2 0 PCB.CPBX352
TY 290 65 12 8 0 0 3 * Out
TY 40 60 12 8 0 0 3 * In
TY 95 225 12 8 0 0 3 * Alimentazione
TY 70 190 12 8 0 0 3 * C3
TY 230 95 12 8 0 0 3 * C2
TY 115 150 12 8 0 0 3 * C1
TY 120 170 12 8 0 0 3 * R4
TY 220 200 12 8 0 0 3 * R3
TY 230 55 12 8 0 0 3 * R2
TY 100 55 12 8 0 0 3 * R1
RV 30 5 320 255 3
```

2.6 Using the ruler

When drawing a PCB, it is often useful to measure distances in the working area. For example, you can check a track width, the clearance between two tracks or the total size of a PCB. FidoCadJ offers (since version 0.23.2) a ruler feature that allows you to easily perform those tasks. Just right click and drag. You should obtain a green ruler, like the one shown in figure 2.19. If the right click and drag action is not easy to obtain in your system, you can alternatively left click and drag while pressing the Shift key. The total length measured is

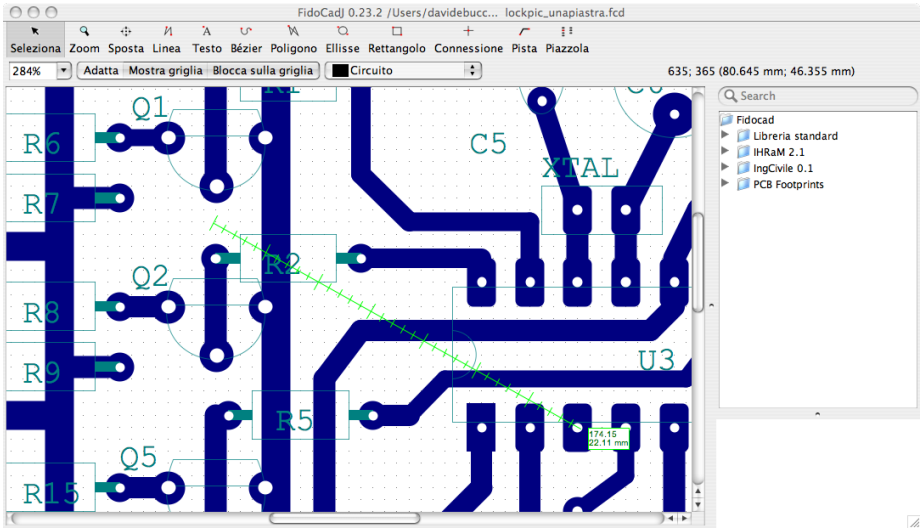


Figure 2.19: Right click and drag to activate the FidoCadJ ruler.

shown in FidoCadJ logical units as well as in millimeter. This is useful when the drawing is printed in the 1:1 mode (such as for PCB's).

2.7 Arrow and stroke styles

FidoCadJ allows to draw arrow heads at the beginning and at the end of lines, Bézier curves and natural cubic splines. It also allows to specify whether the arrow should be at the beginning or at the end of an element (or both), and offers you a few different arrow styles. Arrow length can be negative: in this case, the arrow extends outside the line or curve. Feel free to experiment! A few stroke styles are available too, for technical or mechanical drawings. Figure 2.20 shows an example in which an electrical circuit (a GIC) is enclosed in a dashed rectangle. An arrow at the end of a Bézier curve is also being used. By doing double-click on this element, FidoCadJ shows a dialog like the one shown in figure 2.21. You notice that the option box "Arrow at start" has been checked. FidoCadJ traces an arrowhead by taking care of his orientation. There are several different drawing styles for the arrow as well for dashing . You may try to play a little bit with them to appreciate their differences. If the arrow length is negative, the arrow extends outside the length of the element, pointing towards it.

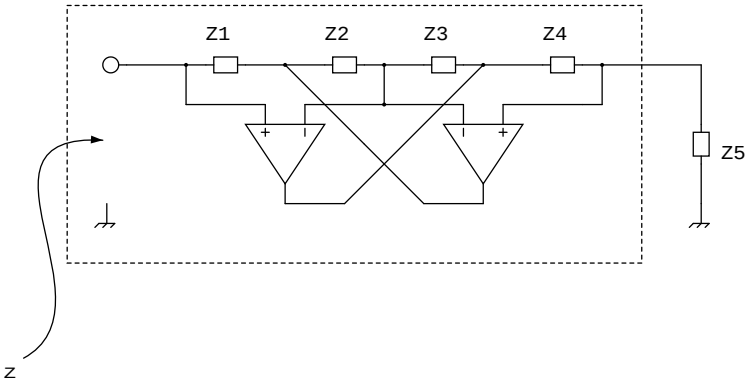


Figure 2.20: An electrical drawing (an Antoniou's GIC) in which some FidoCadJ extensions have been used.

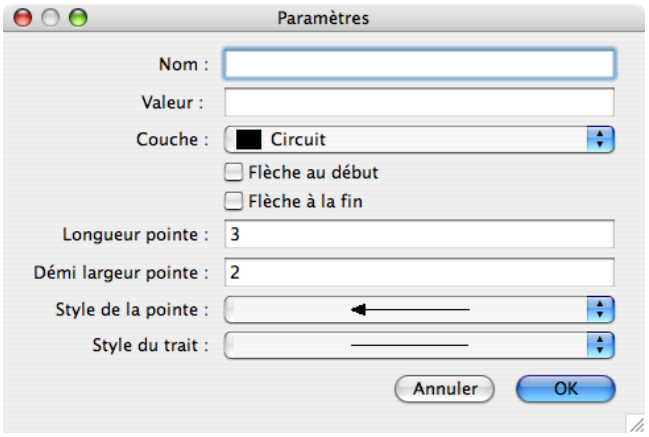


Figure 2.21: The parameter window of the Bézier curve shown used in the schematics of figure 2.20 (French locale).

The possibility of choosing a dashing style and putting arrowheads was not comprised in the original FidoCAD format. This unfortunately means that FidoCadJ drawings using those functionalities are not completely backward compatible with FidoCAD per Windows. If you need to have a complete FidoCAD compatibility, you can activate the option "Strict FidoCAD compatibility" in the "FidoCadJ extensions" tab of the "FidoCadJ preferences" window. In this way, the program does not allow you to introduce graphical elements that would give compatibility problems with FidoCAD. If you need more details about the compatibility of the new graphical elements with FidoCAD, have a look at section 3.4.

2.8 Exporting

A vector format stores the elements that compose the drawing. A bitmap format works on a matrix of pixels.

One of the most important thing to me about FidoCadJ is the possibility to create simple schematics for typographic use. For this reason I introduced a feature that allows the exportation of drawings through a number of different file formats. To export the current drawing, select the command "Export" from the menu "File". The Table 2.4 shows a list of graphic file formats currently available. For every file format, the Table (but also the export dialog in FidoCadJ) specifies whether it is vector or bitmap[3]. For the bitmap file formats it may be useful to enable the option "Anti aliasing", to reduce the annoying effect of the quantization, visible especially on diagonal lines. The resolution and the "Anti aliasing" options are not used when exporting to a vector file format. In this case, you may instead specify a scaling factor. The option "Black&White" allows the printing of any visible layer in solid black. This is important for the preparation of films to be used for typographic purposes or with a printing box.

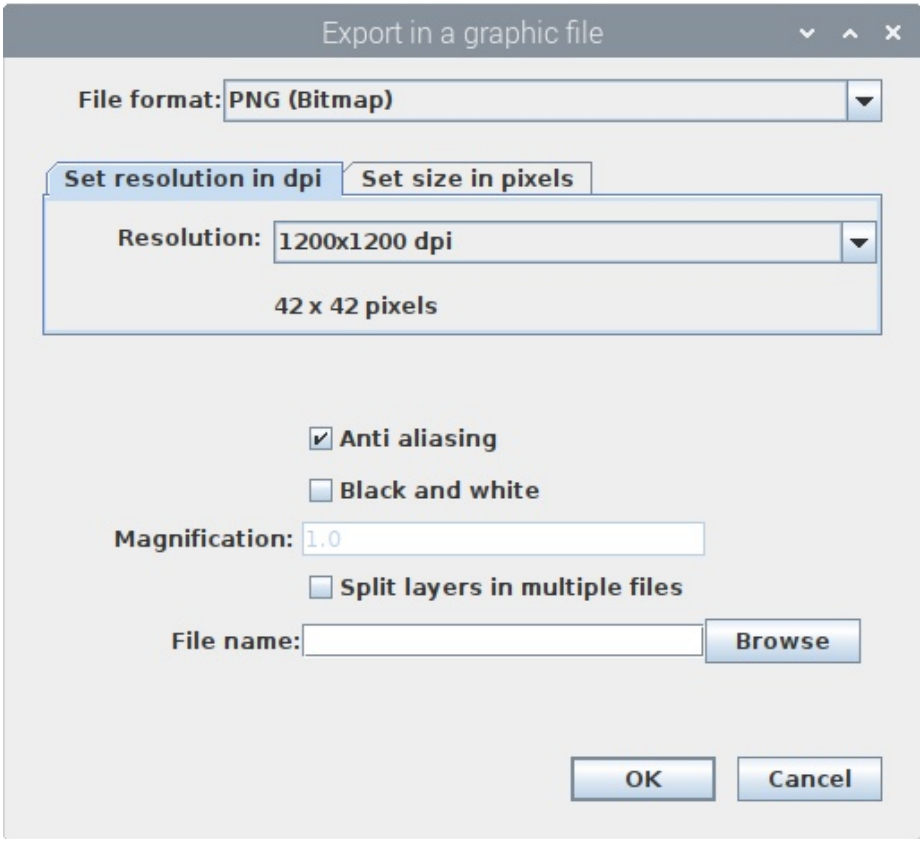


Figure 2.22: The export to image setup menu.

[3]The code structure of FidoCadJ allows the addition of other file format quite easily. Please contact me if you like to participate to the project.

Table 2.4: List of all the export file formats available in FidoCadJ.

Format	Comment
JPG	Ubiquitous bitmap format. The compression used is lossy, therefore it is not perfect to for exporting FidoCadJ schematics as it may yield some visible artifacts.
PNG	Compressed bitmap format, suitable for exporting schematics. This is probably the best way to export a FidoCadJ drawing when a vector format cannot be used.
SVG	W3C standard vector format. Internet browsers can show it within a web page. Very good format for graphics and schematics, it can be used with applications such as Inkscape, to modify the drawings made with FidoCadJ. Some limitations exist with the exportation of rotated and mirrored text.
EPS	Encapsulated Postscript vector format. Used on professional graphics applications, convenient for integrating a FidoCadJ drawing in a LaTeX document. This has been used used to obtain Figure 2.12, page 16 in this manual (passing through a PDF conversion, since I use pdfLaTeX).
PDF	The very well known Portable Document File. Exported files does not include fonts, so rendering of text may be slightly altered on some platforms.
PGF	Vector format to be used directly in a LaTeX document, when using the *pgf* package, available in the CTAN archive. This exportation option was thought to export schematics towards editable script. Text attributes are not translated. This allows the introduction of LaTeX code directly into the drawing and this is the technique used in this manual to obtain Figure 2.6, page 10.
SCR	FidoCadJ allows the exportation of a drawing to a script that can be imported in CadSoft Eagle. To use this feature, it is necessary to install the library `FidoCadJLIB.lbr` into the directory `lbr` of the current installation of Eagle. The library can be downloaded from FidoCadJ's website. At the time of writing, this option works only with schematics containing only the most common symbols. Some drawing elements such as pads and tracks are not available and are not exported.

2.9 Command line options

The application is distributed as a file `.jar`, which is a Java archive.[4] In many operating systems, to run the application it should be enough to double-click on the file, provided that a recent version of Java is installed on the machine. Using Oracle's terminology, the so called JRE, or the Java Runtime Environment, is all that is needed to run a program written in Java (but not to write it: in that case the SDK would be necessary...). The minimum Java version needed to run FidoCadJ is the 9, which has been around for a few years now.

In some cases, it may be useful to run FidoCadJ from a command line (the terminal in the Unix systems, or the MS-DOS Prompt in Windows). To do so, it is sufficient to run the command `java`, with the option `-jar`:

```
java -jar fidocadj-0.24.8.jar
```

If a file is specified in the command line, FidoCadJ tries to open it. For example (on a Unix machine):

```
java -jar fidocadj-0.24.8.jar ~/FidoCadJ/test.fcd
```

FidoCadJ runs and tries to open the file `~/FidoCadJ/test.fcd` (provided that this exists). There are some other interesting things FidoCadJ can do. Option `-h` shows a listing of the FidoCadJ options:

```
[davidebucci@davide-bucci-portable]$ java -jar fidocadj-0.24.8.jar -h

This is FidoCadJ, version 0.24.8.
By the FidoCadJ team, 2007-2023.

Use: java -jar fidocadj-0.24.8.jar [-options] [file]
where options include:

 -n     Do not start the graphical user interface (headless mode)

 -d     Set the extern library directory
        Usage: -d dir
        where 'dir' is the path of the directory you want to use.

 -c     Convert the given file to a graphical format.
        Usage: -c sx sy eps|pdf|svg|png|jpg|fcd|sch outfile
        If you use this command line option, you *must* specify a FidoCadJ
        file to convert.
        An alternative is to specify the resolution in pixels per logical
            unit
        by preceding it by the letter 'r' (without spaces), instead of giving
        sx and sy.
        NOTE: the correctness of the file extension is checked, unless the -f
        option is specified.

 -m     if a file export is done towards a vector graphic file format, split
        the layers and write one file for each layer. The file name will be
        obtained by appending _ followed by the layer number to the specified
        file name. For example, the following command will create files
        test_0.svg, test_1.svg ... from the drawing contained in test.fcd:

           java -jar fidocadj.jar -n -m -c r2 svg test.svg test.fcd

 -s     Print the size of the specified file in logical units.
```

[4] Except for the Macintosh version, which is a stand alone application.

```
continued list of the FidoCadJ options


 -h     Print this help and exit.

 -t     Print the time used by FidoCadJ for the specified operation.

 -p     Do not activate some platform-dependent optimizations. You might try
        this option if FidoCadJ hangs or is painfully slow.

 -l     Force FidoCadJ to use a certain locale (the code might follow
        immediately or be separated by an optional space).

 -k     Show the current locale.

 -f     Force FidoCadJ to skip some sanity tests on the input data.

 [file] The optional (except if you use the -d or -s options) FidoCadJ file
      to
        load at startup time.

Example: load and convert a FidoCadJ drawing to a 800x600 pixel png file
        without using the GUI.
  java -jar fidocadj.jar -n -c 800 600 png out1.png test1.fcd

Example: load and convert a FidoCadJ drawing to a png file without using the
        graphic user interface (the so called headless mode).
        Each FidoCadJ logical unit will be converted in 2 pixels on the image
             .
  java -jar fidocadj.jar -n -c r2 png out2.png test2.fcd

Example: load FidoCadJ forcing the locale to simplified Chinese (zh).
  java -jar fidocadj.jar -l zh


[davidebucci@davide-bucci-portable]$
```

The simplest option is `-n`, with which the software... does nothing, i.e. it does not activate the GUI and just exits. In this case, the Java environment variable `java.awt.headless` is set to true. Obviously, this option is not so much useful alone, but is precious when using in combination with other functionalities we are about to describe. Option `-d` allows to specify the directory where FidoCadJ seeks for libraries to be loaded at startup. Option `-c` allows to make FidoCadJ convert a FidoCAD file (that must be specified) into an image in a vector or raster format. This can be very useful, as FidoCadJ can be used as a converter in a non interactive way (along with the `-n` option).

We can thus have a look at the first example given in the help:

```
java -jar fidocadj.jar -n -c 800 600 png out1.png test1.fcd
```

FidoCadJ runs without activating the GUI and exports in the png format the file `test1.fcd`. The output file will be called `out1.png` and will have a 800x600 pixel size.

There is an alternative version of the `-c` option, which allows to specify how many pixels should be used to convert one logical unit (we call this factor r_p). FidoCadJ does not deal with half logical units (they are always integers), by choosing, let's say, r_p equal to two pixels per logical unit ensures that schematics

is always understandable (even if probably a little bit small). The r_p factor can be non integer, as in the following example:

```
java -jar fidocadj.jar -n -c r1.25 png out2.png test2.fcd
```

To know the total size (in logical units) of a schematic, you can use the `-s` option. Remember anyway that, during the exports, FidoCadJ adds always a $t_\mathrm{margin} = 3$ logical units margin for each side of the drawing. So, if t_w is the width of the drawing in logical units given by `-s`, you might expect that p_w the width of your drawing in pixel is calculated as follows:

$$p_\mathrm{w} = r_\mathrm{p}(t_\mathrm{w} + 2t_\mathrm{margin}) \tag{2.1}$$

Another interesting option, although a feature due to Java more than FidoCadJ, is the possibility to modify the look of the application (in the jargon of Java called look & feel). You can choose the look&feel that you like without modifying a single line of code. Here is something that Linux users appreciate, the GTK+ look:

```
java -Dswing.defaultlaf=com.sun.java.swing.plaf.gtk.
    GTKLookAndFeel -jar fidocadj.jar
```

There is also the classic Motif look & feel, shown in Figure 2.23[5]:

```
java -Dswing.defaultlaf=com.sun.java.swing.plaf.motif.
    MotifLookAndFeel -jar fidocadj.jar
```

Obviously, the commands listed above are meant to be sent from a terminal, making sure that the current directory contains the file `fidocad.jar` and writing everything on the same line.

[5] This style may somehow shock those well acquainted with very refined graphical interfaces such as Aqua, with macOS. However, I saw many years ago a synchrotron control system that had a graphical interface based on Motif and it commanded respect.

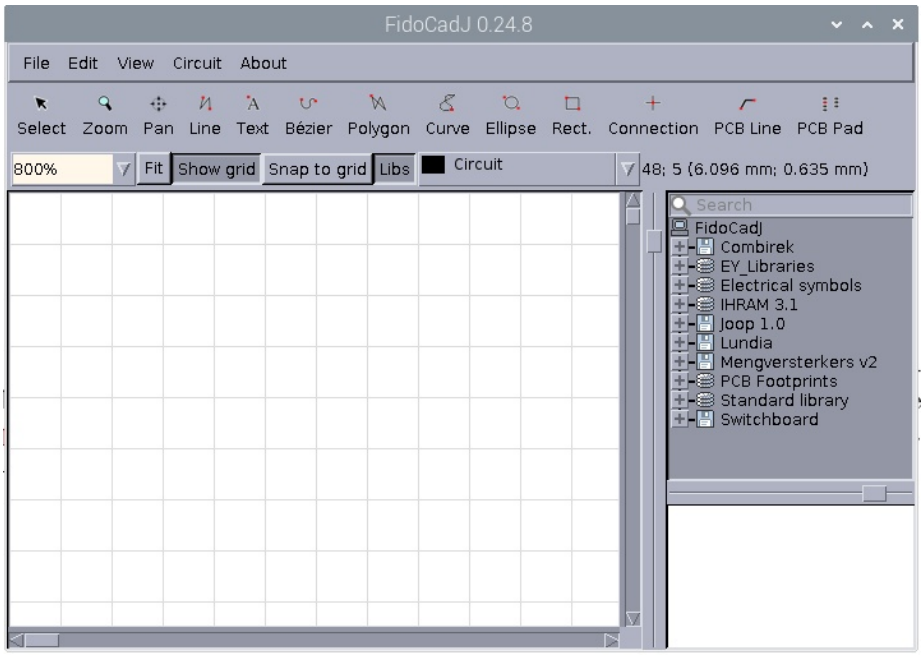


Figure 2.23: The appearance of a very old version of FidoCadJ, using the Motif look & feel.

2.10 Library management

2.10.1 Using library files

A library is a collection of symbols that the user can include in his drawings. There is a collection of libraries included in the FidoCadJ packet, as well as the possibility of defining new symbols and libraries. In fact, FidoCadJ allows to specify a directory in which all the user library files are placed (with the file extension `.fcl`). This can be done through the menu "File/Options".
In some very special cases (or for testing), the libraries included in the FidoCadJ packet can be replaced by external ones. If a file named `FCDstdlib.fcl` is present, its content supersedes the standard library directly available in the application. Analogously, if a file named `PCB.fcl` is present, its content replaces the PCB library[6].

In version 0.23, thanks to Roby IZ1CYN, I could include the IHRaM 3.1 library directly inside the FidoCadJ distribution. I have done this because among all the libraries I have seen, this one has appeared to be one of the most complete and rationally constructed. Exactly as it happens for the other libraries embedded in FidoCadJ, if there is a file called `IHRAM.FCL` in the current library search path, this one is loaded at the place of the version embedded in the program. You also have an electrical symbols library whose file is called `elettrotecnica.fcl`.

[6]Pay attention to the use of capital letters, if your operating system distinguish uppercase and lowercase letters in the files management.

Other files with `.fcl` extension in the library search path are considered as libraries and FidoCadJ tries to load them when it is starting. This is done when the application starts, when the user change the library search path, or when the "Update libraries" option in the "Circuit menu" is chosen.

FidoCadJ allows to split non standard symbols (as the original FidoCAD does). This can be very useful when posting a drawing in a newsgroup, since in this way each symbol not belong to the standard FidoCAD libraries is expanded into its graphic primitives. Who reads your post, thus, does not need to have the very same libraries you have installed in your system. For copy/pasting, a command and a shortcut are available in the menubar in the "Edit" menu, called "Copy, split non standard macros" (or Control+M[7]), in order to make sort that the copied drawing will have all the non standard macros split. You might also save a file with the non standard macros split, by choosing "File" in the menubar and "Save as..., split the non-standard macros". A dialog appears to let you choose a new file name, since usually you do not want to overwrite the one you are working with.

2.10.2 Defining new symbols

FidoCadJ allows to create new libraries and new symbols in a way that should be reasonably intuitive. Here are the steps to follow:

- FidoCadJ needs a place where to store the produced files. Make sure that a directory containing the user libraries has been defined. If not, specify it in the FidoCadJ user settings.
- Draw your new symbol.
- Select what you just drawn and right click on it. A popup menu appears, such as in figure 2.24.
- Choose "Symbol-o-matic." A dialog for the definition of the details of the newly created symbol appears (see figure 2.25).
- In the first field "Library filename" you find the file name of the library in which the new symbol must be included. If you type a new name there, FidoCadJ creates a new file.
- The second field "Library complete name" is the complete name of the library, the one that is shown in the library tree. You might write what you want there, but it is better to choose a short and meaningful one.
- The third field "Group" allows you to choose in which group you want to put the new symbol, inside the library. Once again, if you type a new name, a new group is created.
- The fourth field "Name" is the name of the symbol, what it is shown in the library tree. Once again, choose a short but meaningful symbol.

[7] On macOS systems, the shortcut is Command+M

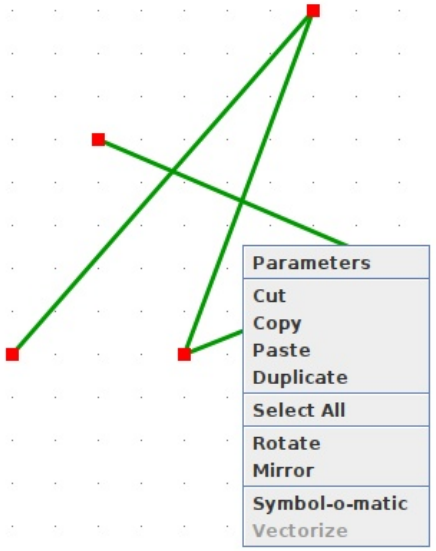


Figure 2.24: The pop-up menu appearing with a right click allows to transform drawing elements into a symbol

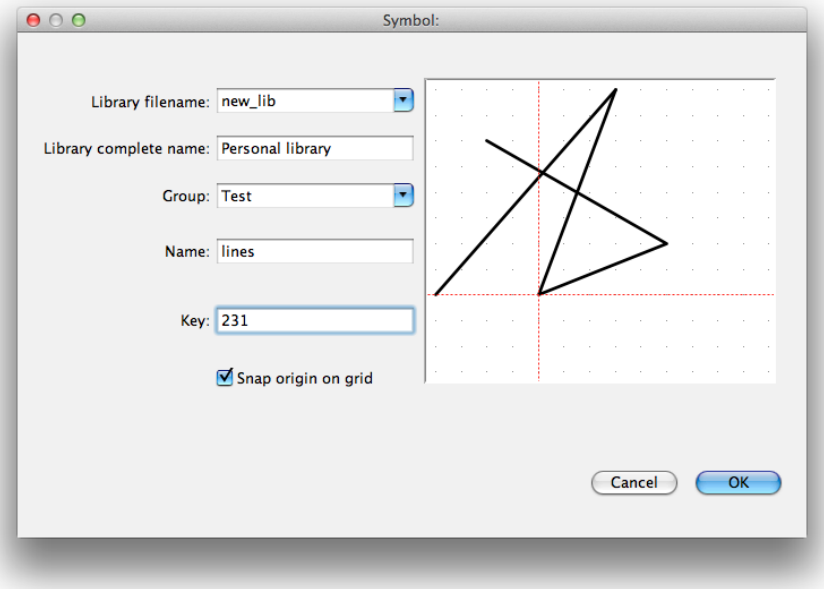


Figure 2.25: The new symbol definition dialog. Here you can set up all the important characteristics of the symbol. Note the origin defined by the two red axis.

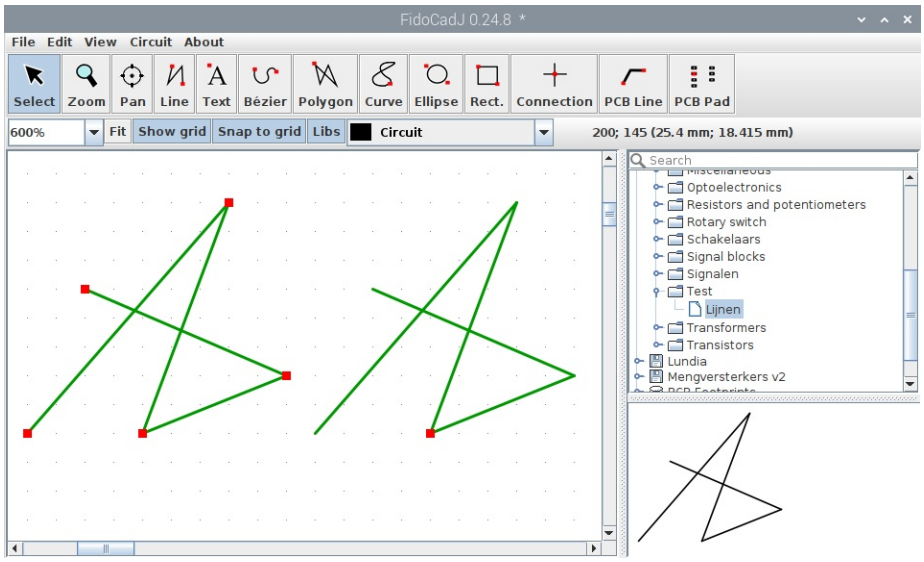


Figure 2.26: The freshly created symbol, shown in the symbol list and in the drawing. On the left, there is still the drawing that has been used for the symbol definition. Notice that just one control point is present for the new library symbol in the drawing.

- The fifth field "Key" is a very short tag that identifies the symbol in the code. It should be unique inside each library and should not contain spaces as well as characters such as parenthesis, dots, and so on. FidoCadJ proposes a short numerical code for you, but you might use mnemonic tags as well.
- By clicking in the field on the right, you must choose the origin of the symbol, the point that is to be used as a reference for placing the symbol inside the drawing. You might choose to snap on the grid the position you choose by clicking on "Snap origin on grid".
- Once you click on "OK", you should find your newly created symbol in the user library in the library tree in the main FidoCadJ window.

An example of an user symbol employed in a drawing is shown in figure 2.26. Note the difference between the selected part of the drawing on the left (actually what it has been used for defining a new symbol) and the new symbol placed on the right. In fact, just one control point is available for the placement of the symbol and this control point corresponds to the choice of the origin while doing the symbol definition. It is thus useful to choose for that a point that has some relevance in your symbol. You can split a symbol again in its primitives by selecting it and clicking on "Vectorize".

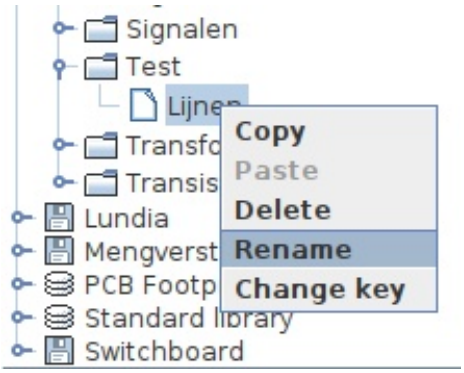


Figure 2.27: The popup menu used for modifying symbol properties in an user library.

2.10.3 Modifying existing symbols

A few menu actions are available on symbols in the library tree. Once you select a symbol in an user library and do a right-click on it, a popup menu appears, as shown in figure 2.27. Here you have several options:

- "Copy" copy an entire library or a library symbol.
- "Paste" paste the entire library or a library symbol at the desired location.
- "Delete" throws away the symbol. Be careful with this: if a drawing contains the symbol, it will be invalidated and disappear.
- "Rename" changes the displayed name of the symbol.
- "Change key" is useful to modify the key associated to the symbol. Once again, this is something that must be done when the symbol has not been used in the drawing.

By using drag and drop, you might change the library or the category on which a symbol is categorized. Take into account that a symbol complete identification tag is composed by the library name, followed by the key of the symbol. Thus, moving a symbol from one library to another invalidates drawings containing it.

All library operations are undoable, exactly as drawing operations. Bear in mind that empty libraries or groups are not shown.

Chapter 3

Drawing format, macros and libraries

This chapter contains a detailed description of the format used by FidoCAD and, as a consequence, by FidoCadJ to store a drawing. It is a simple text format that has the advantage of being very compact and efficient. Since the format has never been described in detail in a document, I try to summarize all that I learnt about it. Since I added a few extensions that are only available on FidoCadJ, I describe them here as well. Remember that since version 0.23.4, FidoCadJ uses only the UTF-8 encoding on all platforms.

3.1 Header description

All the files containing a drawing in the FidoCAD format must start with the tag `[FIDOCAD]`. A program can therefore recognize the presence of FidoCAD commands by reading this tag. In this regard, FidoCadJ is more tolerant than the original FidoCAD and it recognizes and correctly interprets a file that does not contain the standard header. Even commands containing text can therefore be interpreted correctly as long as the number of incorrect lines does not exceed a value set internally in the program (approx. 100). This ensures that FidoCadJ does not waste time working for a few minutes for example trying to open a very large binary file.

3.2 Coordinates system

FidoCadJ works on a very simple coordinates system. In practice, it has at its disposal a very large area identified only by whole and positive coordinates. The length of every unit in x and in y is fixed at 127 μm, a value that allows to obtain a good resolution for even the smallest SMD package without being too fine for everyday use. In typographical terms, the FidoCadJ resolution is 200 dots per inch.

The original FidoCAD had two different modes of operation: PCB and electrical schematic. In FidoCadJ this difference has been blurred and appears only at the moment of printing the drawing. It would therefore be advisable to set the program to resize an electrical schematic to make better use of the size of the page, otherwise the printed result will have the size of a postage stamp.

3.3 Drawing elements

FidoCadJ can manage 12 drawing elements as follows

- Line
- Filled or empty rectangle
- Simple text (obsolete)
- Advanced text
- Filled or empty polyline
- Filled or empty ellipse
- Bézier curve
- Natural cubic spline curve
- Electrical junction
- PCB pad
- PCB track
- Macro

We will describe each of them. In general, every element is identified by a command and a number of parameters (usually integer numbers or text strings) placed on the same line and separated by a space character.

Line

The line primitive is identified by the command `LI` and its definition requires only the begin and end coordinates and the layer:

Mathematicians would probably find the term "segment" more appropriate.

```
LI x1 y1 x2 y2 l
```

the points (x_1, y_1) and (x_2, y_2) represent respectively the initial and final coordinates, while l is the layer, characterized by an integer number between 0 and 15.

FidoCadJ extension: starting from FidoCadJ version 0.23, `LI` can be followed in the next line by an extension:

```
FCJ a b c d e nv
```

Table 3.1: Meaning of the a parameter for the presence of an arrow head at the sides of a line or Bézier primitive.

a	Arrow head
0	none
1	at the start side
2	at the end side
3	both sides

Table 3.2: Meaning of the b parameter for the arrow head style.

b	Arrow head style
0	filled standard arrow head
1	filled standard arrow head with quota line
2	empty arrow head
3	empty arrow head with quota line

where a is an integer that represents the presence or not of the arrow heads at the extremes of the segment (have a look at table 3.1), b represents the arrow head style (as table 3.2). Parameters c and d give respectively the total length and the half width of the arrow head, while e is an integer that gives the dash style. If nv is equal to 1, the `FCJ` command should be followed by two `TY` commands giving the name and the value associated to this element.

Filled or empty rectangle

A rectangle filled or empty is identified by the commands `RP` and `RV` respectively, followed by the coordinates of the two vertices on one of the two diagonals, and the layer.

```
RP x1 y1 x2 y2 l
RV x1 y1 x2 y2 l
```

the points (x_1, y_1) and (x_2, y_2) represent the two vertices and l is the layer, characterized by a whole number between 0 and 15.

FidoCadJ extension: starting from FidoCadJ version 0.23, `RP` and `RV` can be followed in the next line by an extension:

```
FCJ e nv
```

where e is an integer giving the dashing style. If nv is equal to 1, the `FCJ` command should be followed by two `TY` commands giving the name and the value associated to this element.

Simple text (obsolete)

The simple text was the first primitive provided with the early versions of FidoCAD. FidoCadJ recognizes it and simply writes text with size 12, regardless the current zoom.

Since this primitive was considered obsolete by Lorenzo Lutti, the creator of FidoCAD, FidoCadJ does exactly the same and, although this is correctly

Table 3.3: Function of the bits in the text style term.

Bit	Weight	Behavior
0	1	text in bold
2	4	mirrored text

interpreted, it is not available on the toolbar. FidoCadJ stores this element exactly as it was advanced text and it uses the command TY when saving the file.

The command is TE and the format is as follows:

```
TE x1 y1 *text to be written
```

the point (x_1, y_1) is where the string "text to be written" is to be positioned. Notice that the layer information is missing. FidoCadJ treats this object as it was placed on layer zero (circuit).

Advanced Text

The primitive advanced text offers much more flexibility with respect to simple text introduced above.

It is identified by the command TY, followed by a number of parameters to determine the text orientation (rotated or mirrored), as well as dimensions along x and y and the font used. Due to the quantity of information to provide, the resulting command string is quite complex:

```
TY x1 y1 sy sx a s l f *text to be written
```

The point (x_1, y_1) is where the string "text to be written" is to be positioned. The value of s_y and s_x indicates the horizontal and vertical dimensions of the text in logical units. FidoCadJ respects the vertical dimension starting from the horizontal one, and that aspect ratio is changed only if strictly necessary. The text rotation is described by the term a, expressed in sexagesimal degrees, while the value of s determines the text style, following table 3.3. The layer is given by the usual term l, and f indicates the font to be used, or it can be an asterisk, to indicate the use of the standard Courier New font If the font name contains spaces these must be replaced with "++".

Text commands support subscripts: to obtain them, insert text between the underscore _ and the caret ^; for example _12^ writes 12 as a subscript. They support superscripts as well: insert text between the caret ^ and the underscore _; for example ^56_ writes 56 as a superscript. In the two cases, the last ^ or _ are optional. To print the _ or ^ character, use a backspace, i.e. _ to obtain _ and \^ to obtain ^. Use two backspaces to obtain a single \. Example of text with superscripts: 3^2_+4^2_=5^2 or C_12

The maximum length of the text is about 80 words. The counting is done in words and not in characters because in the internal structure of the program the words (and command terms) are separated when a line is being interpreted.

Filled or Empty polyline

A polyline filled or empty is indicated with the commands PP and PV respectively, followed by the coordinates of the vertices that define the polyline, and the layer. The commands are

```
PP x1 y1 x2 y2 ... l
PV x1 y1 x2 y2 ... l
```

where the points (x_1, y_1), $(x_2, y_2)\ldots$ are the vertices that define the polyline and l is the layer, characterized by a whole number between 0 and 15. The length of the command line can thus vary depending on the number of vertices used. The maximum number of vertices available has been arbitrarily fixed to a little less of 5000.

FidoCadJ extension: starting from FidoCadJ version 0.23, PP and PV can be followed in the next line by an extension:

```
FCJ e nv
```

where e is an integer giving the dashing style. If nv is equal to 1, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

Filled or Empty ellipse

An ellipse filled or empty is identified by the commands EP e EV respectively, followed by the coordinates of the two vertices on the diagonal and the layer number.

```
EP x1 y1 x2 y2 l
EV x1 y1 x2 y2 l
```

the point (x_1, y_1) represents the first vertex on the diagonal, (x_2, y_2) is the second vertex, and l is the layer, identified by a whole number between 0 and 15.

FidoCadJ extension: starting from FidoCadJ version 0.23, EP and EV can be followed in the next line by an extension:

```
FCJ e nv
```

where e is an integer giving the dashing style. If nv is equal to 1, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

Bézier curve

A Bézier curve segment, in its cubic variant, is identified by four vertices, which are thus required by its associated command BE:

```
BE x1 y1 x2 y2 x3 y3 x4 y4 l
```

where $P_1 \equiv (x_1, y_1)$, $P_2 \equiv (x_2, y_2)$, $P_3 \equiv (x_3, y_3)$ and $P_4 \equiv (x_4, y_4)$ are the four control points of the Bézier curve segment, while l is the layer, identified by a

whole number between 0 and 15. Given the four points defined above, the curve segment is computed through the expression

$$B(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4, \tag{3.1}$$

where $t \in [0, 1]$ is a parameter.

FidoCadJ extension: starting from FidoCadJ version 0.23, `BE` can be followed in the next line by an extension:

```
FCJ a b c d e nv
```

where a is an integer that represents the presence or not of the arrow heads at the extremes of the curve (have a look at table 3.1), b represents the arrow head style (as table 3.2). Parameters c and d give respectively the total length and the half width of the arrow head, while e is an integer that gives the dash style. If nv is equal to 1, the `FCJ` command should be followed by two `TY` commands giving the name and the value associated to this element.

Natural cubic spline (complex curve)

A natural cubic spline is defined by a certain number of vertices. The curve crosses each vertex and it is calculated in such a way that it is very smooth. In the FidoCadJ format, a spline is identified by commands `CV` and `CP`:

```
CV aa x1 y1 x2 y2 ... l
CP aa x1 y1 x2 y2 ... l
```

the aa parameter is equal to 1 if the curve is closed, or it is 0 otherwise. Vertices (x_1, y_1), $(x_2, y_2)\ldots$ define the spline and l is the layer, an integer ranging from 0 to 15. The lenght of the command line can thus vary depending on how much vertices are considered. As for polygons, the maximum number of vertices available is internally fixed to a little less of 5000. This primitive has been introduced in version 0.24 and is not present in the original FidoCAD.

FidoCadJ extension: `CV` and `CP` can be followed in the following line by an extension:

```
FCJ a b c d e nv
```

where a is an integer that represents the presence or not of the arrow heads at the extremes of the curve (have a look at table 3.1), b represents the arrow head style (as table 3.2). Parameters c and d give respectively the total length and the half width of the arrow head, while e is an integer that gives the dash style. If nv is equal to 1, the `FCJ` command should be followed by two `TY` commands giving the name and the value associated to this element.

Electrical junction

The electrical junction primitive is simply a filled circle of constant dimensions and it is used to represent a connection in an electrical schematic. It is identified by the command SA and it only requires its coordinates and layer:

```
SA x1 y1 l
```

With FidoCadJ, the diameter of the circle is fixed internally into the program to two logical units.

If SA is followed by FCJ, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

PCB pad

A PCB pad is identified by the command PA and it is characterized by its style (round, rectangular, rectangular with smoothed corners) and the diameter of its internal hole:

```
PA x1 y1 dx dy si st l
```

where the point (x_1, y_1) represents the position of the pad, d_x is the pad's width (along the x axis), d_y is the height (along the y axis). The value s_i is the diameter of the pad's hole, while s_t is the pad's style:

0 oval pad

1 rectangular pad

2 rectangular pad with smoothed corners

The value of l must be a whole number to indicate the layer where the pad is to be placed. If PA is followed by FCJ, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

PCB track

The PCB track is essentially a segment, the width of which can be specified. The corners at the extremes of the segment are always rounded to facilitate the connection with other PCB tracks or pads. The command to be used is PL, with the following format:

```
PL x1 y1 x2 y2 di l
```

The track is drawn between the points (x_1, y_1) and (x_2, y_2), with total width d_i, and the layer used is l. The width d_i can be non integer. If PL is followed by FCJ, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

Macro call

A macro is a drawing or a symbol contained in a library. Generally, this is the way frequently used electrical symbols are represented. The command used to call a macro is MC, and the call is done as follows:

```
MC x1 y1 o m n
```

The symbol is drawn using position (x_1, y_1) as the reference, and the orientation is defined by the value of o (multiplied by 90° clockwise) and if m is equal to 1 the macro is mirrored. the last parameter, n, is the macro's name within the library, specified as *library.code*.

If MC is followed by FCJ, the FCJ command should be followed by two TY commands giving the name and the value associated to this element.

3.4 FidoCadJ extensions

Since version 0.21, FidoCadJ has started to introduce a few refinements over the original FidoCAD format. The FidoCadJ extensions are represented in the code by the command FCJ. This command is not used alone, but means that FidoCadJ needs to specify additional information on what it is specified in the previous line. Here is an example:

```
[FIDOCAD]
MC 40 30 0 0 080
FCJ
TY 50 35 4 3 0 0 0 * R1
TY 50 40 4 3 0 0 0 * 47k
```

The presence of the FCJ command indicates that the name and the value of the macro specified in the first line are given by the two TY commands that follow. Only the coordinates and the fonts are taken into account for the text rendering.

FidoCadJ allows to activate a "strict FidoCAD compatibility mode", in which all extensions are disabled. This way, FidoCadJ is perfectly compatible with the original FidoCAD, except that FidoCadJ continues using the UTF-8 encoding instead of the old CP-1252 used by FidoCAD. FidoCAD for Windows is unable to understand the additional informations carried by the FCJ command. When reading with the original FidoCAD a FidoCadJ drawing containing some extension, the program prompts a few errors and gives a result that is different from the original FidoCadJ drawing. The results depend on what extension has been used: a dashed line is rendered as continuous and the arrow heads is not drawn. On the other hand, the text associated to the name and the value of a macro is rendered perfectly.

Figure 3.1 shows what can be obtained using FidoCAD to read the FidoCadJ file that describes 2.20. After ignoring the FidoCAD errors, there are some details missing (the dashing and the arrow), but the overall drawing is still understandable.

An important difference from the original FidoCAD is that FidoCadJ allows to save a certain number of configuration hints in the output files. The command that is used for that is FJC and it should normally be placed at the very beginning of the file. The next paragraphs describes the cases that are treated.

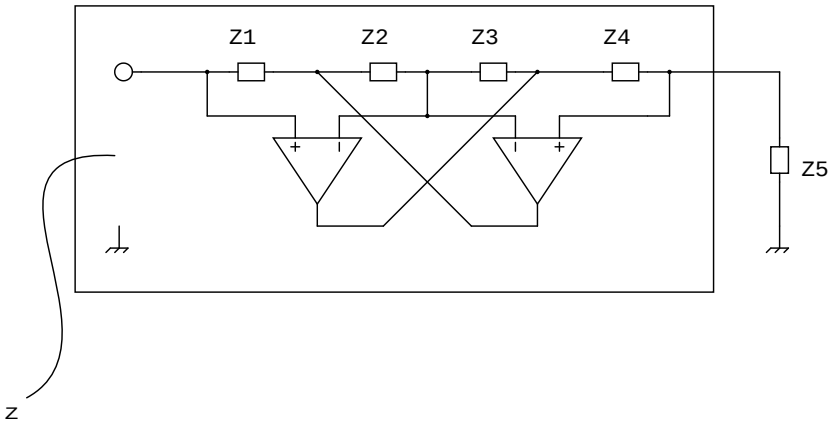


Figure 3.1: Figure 2.20 as it would appear on FidoCAD for Windows.

3.4.1 Layer setup

The layer setup is specified with the command `FJC L`. FidoCadJ saves data only if the corresponding layer has been modified from its default state. The syntax is as follows:

```
FJC L n xxxx yy
```

where n represents the layer number (ranging from 0 to 15), $xxxx$ is a 32 bit integer containing settings about the RGB color to be used. The red component is contained in bits 16-23, the green one in bits 8-15 and the blue one in bits 0-7. The value yy is a single precision floating point constant that gives the layer transparency, comprised between 0.0 (completely transparent) and 1.0 (completely opaque).

A second important information is the name of the layer, specified (if the user has changed it) as follows:

```
FJC N n aaaaa
```

where n is the number of the layer (integer from 0 to 15), whereas *aaaa* is the name of the layer to be considered. If this command is not present, the name of the layer is the one used by default by FidoCadJ, depending on the language and on the local configuration of the operating system.

3.4.2 Electrical connection setup

The size of the black circle used to indicate an electrical connection can be modified by the user. When the FidoCadJ extensions are active, the selected value is saved in the file with the `FJC C` command as follows:

```
FJC C aaaa
```

where *aaaa* is a double precision floating point value (positive) that gives the diameter of the electrical connection, in FidoCadJ logical units.

3.4.3 Stroke width

The stroke width for the "pen" used during the drawing of electrical schematics can be modified. FidoCadJ adopts the command `FJC A` to specify the stroke width of the lines, ovals, Bézier, splines, rectangles and polygons. The width can be a double precision constant.

```
FJC A aaaa
```

Where *aaaa* represents the stroke width (in logical units). The default width is internally defined to 0.5 logical units. In older versions of FidoCadJ, a command `FJC B bbbb` was used to specify the stroke width of curved lines (ovals, Bézier). Since version 0.23.6, this differentiation has been eliminated and this command is no longer adopted.

3.5 Syntax errors tolerance

FidoCadJ is designed to tolerate errors and or certain syntax errors in the commands passed to the program. Obviously, unless you have a crystal ball connected as a USB device, the program is not able to correct errors but simply skips (and deletes) all the lines involved.

An exception to this behavior is that a number of elements can be specified without the layer, which is considered part of the layer 0 (schematics). This is for backward compatibility with FidoCAD.

3.6 Libraries format

The file structure of a library file is quite simple:

```
[FIDOLIB Librairie de base]
{Syboles de base}
[000 Terminal]
LI 100 100 102 100
EV 102 98 106 102
[010 Terminal +]
LI 100 100 102 100
EV 102 98 106 102
LI 103 100 105 100
LI 104 99 104 101
[020 Terminal -]
LI 100 100 102 100
EV 102 98 106 102
LI 103 100 105 100
...
```

The first line contains, between square brackets, the library's name (preceded by FIDOLIB). The second line must contain, between curly brackets, the library category under which the macros, specified later on in the file, are to be stored.

Each macro is composed by a header (between square brackets) and a sequence of commands. The header is constituted by a *part name* (that must be unique within the library) and its description. The part name will be used within a FidoCadJ script, while the description helps the user while browsing all the macros contained in the file. The commands are nothing else than FidoCadJ drawings, where the coordinate point (100,100) is used as the origin. This is the point that is to be used as the reference when the macro is called. In a FidoCadJ script a macro is identified by "library.macro" used with the MC command.

Nothing prevents the calling of a macro within another macro. However, recursion (i.e. a macro that calls itself) must be avoided.

A library *must not contain* among the macro definitions any information about the FidoCadJ configuration. In other words, commands `FJC` should *never* appear inside a library file.

3.7 Standard Libraries

FidoCadJ contains two libraries traditionally supplied with FidoCAD. These are the standard library and the library that contains the PCB symbols. Another library that has been developed for FidoCAD and is so useful that has been included as part of FidoCadJ is the IHRAM one. IHRAM stands for `it.hobby.radioamatori.moderato` and it is an Italian Usenet discussion group.

However, it is possible to override the content of these internal libraries by specifying ("File/Options/Libraries Directory" menu) a directory containing the libraries to be loaded. If a file named `FCDstdlib.fcl` is present in this directory, its content is used instead of the standard library. If a file named `PCB.fcl` is present, its content is used instead the internal PCB library. Other

libraries having different file names (but still with extension `fcl`) are loaded together with the standard libraries. If there is a file called `IHRAM.FCL` in the current library search path, this one is loaded at the place of the version embedded in the program. You also have an electrical symbols library whose file is called `elettrotecnica.fcl`. The standard libraries in FidoCadJ can be recognized outside of the name by the symbol with three slices on top of each other (for "hard disk"). Own and other libraries located in the defined folder "File/Options/Libraries Folder" can be recognized by the floppy disk symbol.

Figure 3.2: Example of how libraries appear in the Libraries folder.

FidoCadJ considers the libraries above as part of the standard set. In other words, it does not split the symbols contained in drawings, if they belong to those libraries, except when the option "Strict compatibility with FidoCAD" is active. In this case, only the original FidoCAD library is considered as standard, exactly how it used to do the original software.

Chapter 4

Conclusion

We have seen in this manual how to use FidoCadJ to draw an electrical schematic or a simple PCB. At this stage the reader should possess all the elements necessary to use FidoCadJ effectively for his needs.

FidoCadJ should not be considered uniquely for the electronics design. It can be used for any type of 2D drawing and in many situations, provided that specific libraries are available.

The advantages of a free program is that it is completely open to its user community. For this reason, your feedback is very important (at least to understand if the project is worth to be developed further, and in which direction). To contact me, open an issue on the GitHub FidoCadJ project[1].

[1] https://github.com/DarwinNE/FidoCadJ/issues

Appendix A

Platform-specific information

A.1 macOS

One of the most frequent criticisms raised against the early versions of FidoCadJ from Macintosh users (like me, by the way) was the poor integration of the program on macOS. Starting since version 0.21.1, FidoCadJ is making specific efforts to comply better with the look and the philosophy of Mac's native applications. For this reason, some details in the program are slightly different when FidoCadJ is run on an Apple platform:

- by default FidoCadJ uses the Quaqua[1] look and feel when the complete application (FidoCadJ.app instead of fidocadj.jar) is run. Since Quaqua may slow down the performance on not so recent machines, it is possible to disable it through the settings of the Preferences menu
- The menu bar is shown at its place, that is the top of the screen
- The menu items "Preferences" and "About FidoCadJ" are at their place, which is under the FidoCadJ menu.
- The program tells the operating system that it can open `.fcd` files; These are associated to a specific icon, which should be sufficiently evocative.

A.1.1 How to download and execute FidoCadJ on macOS

FidoCadJ can work with a macOS version more recent than 10.6 (Snow Leopard). Under the hood, FidoCadJ needs at least version 9 of Java. For marketing reasons, Apple does not install Java with the last versions of macOS. By the way, Apple does not allow to distribute his App Store software based on Java or distributed under GPL. This is one of the most important reasons for which it is quite improbable that FidoCadJ will be ported to iPad and iPhone.

Even if you can use directly the Java archive `fidocadj.jar` as you would do on other operating systems, on macOS you can use the specifically tailored

[1] http://www.randelshofer.ch/quaqua/

application. Everything works just like a native application: you can download the disk image at the following link:
https://github.com/DarwinNE/FidoCadJ/releases/download/v0.24.8/FidoCadJ_MacOSX.dmg
You can then open the disk image and move `FidoCadJ.app` in the `Applications` folder, where you can use it exactly like any other Macintosh application. To uninstall FidoCadJ, just drag `FidoCadJ.app` in the trash bin.

A.2 Linux

by Roby IZ1CYN

Prerequisite: JRE 9 from Oracle and/or OpenJDK 9 JRE (or successive versions compatible with the program's specifications) must be installed. In paragraph A.2.1 the installation of the program using only commands from a terminal is described. In paragraph A.2.2, the interaction with a graphical environment will be used instead. A poor configuration of your Java runtime environment determines poor performances of FidoCadJ[2].

A.2.1 Using any platform, from terminal

Download the program using the `wget` command:

```
$ wget https://github.com/DarwinNE/FidoCadJ/releases/
    download/v0.24.8/fidocadj-0.24.8.jar
--00:48:18--  https://github.com/DarwinNE/FidoCadJ/releases/
    download/v0.24.8/fidocadj-0.24.8.jar
           => 'fidocadj.jar'
Resolution of github.com is being done... xxx.xxx.xxx.xxx
Connection to github.com is being done... xxx.xxx.xxx.xxx
    connected.
HTTP request sent, waiting for answer... 200 OK
Length: 343,207 (335K) [application/java-archive]

100%[====================================>] 343,207
    422.48K/s

00:48:30 (420.55 KB/s) - "fidocadj-0.24.8.jar" saved [xxx/
    yyy]
$
```

Alternatively, or if you are experiencing problems, you can download the file from any browser from the following URL:
https://github.com/DarwinNE/FidoCadJ/releases/download/v0.24.8/fidocadj-0.24.8.jar
and save the file for example in your `/home/[user_name]/Download` directory (many modern browsers do that by default).

[2] N.d.c. I swear, it is true! *Please*, do not insult me if your favorite Linux distribution comes with an highly unreliable version of Java.

Create a new directory (but at first we will become a superuser by using `su` or `sudo -s`):

```
$ su
-> enter password
# mkdir /usr/local/bin/fidocadj
```

... and we can move there the downloaded file (substitute `<user>` with the user name of the account where the file has been downloaded):

```
# mv /home/[user_name]/Download/fidocadj-0.24.8.jar /usr/
    local/bin/fidocadj
```

Make the file an executable

```
# chmod +x /usr/local/bin/fidocadj/fidocadj-0.24.8.jar
```

And we must not forget to come back to be normal users:

```
# exit
```

And now, we can execute the program:

```
$ /usr/local/bin/fidocadj/fidocadj-0.24.8.jar
```

A.2.2 On a graphical system

In the example, we will be using Ubuntu 8.04, but things do not change much for older or newer versions:

- Download the file from the browser, or with Gwget or similar tools.
- Launch our File Manager (Nautilus, Konqueror...) as a root (if we do not have a specific command in the menu, we just need to launch it from the console by using `sudo nautilus`). We can create the following directory:

  ```
  /usr/local/bin/fidocadj
  ```

  and then move there the file we just downloaded, which is in many cases put in a directory such as `/home/[user_name]/Download/` after the download.
- Right click on the file, from the window we select the tab "rights", we add the check mark in front of the "Allow the execution of the file as a program", as shown in the figure A.1
- Select the tab "Open as" and select "OpenJDK Java 9 Runtime" or "Oracle Java 9 Runtime" (or a successive version).
- Click on "Close" and we are now ready to execute FidoCadJ: a double click on the executable, or we can add it to the main menu. The command to add is just `/usr/local/bin/fidocadj/fidocadj-0.24.8.jar`.

Figure A.1: The setting of file rights, on Ubuntu 8.04.

A.2.3 "Portable" install

by Max2433BO

If you can not install anything on the system, FidoCadJ can still be used as a "portable" application that can be executed everywhere without requiring super user rights.

To do so, create a directory `/home/[user_name]/fidocadj/` where the following files are put:

- `fidocadj-0.24.8.jar`
- `jdk-9.0.4` (javaSE 9, or greater)
- a script `fidocadj-0.24.8.sh` to launch FidoCadJ, with the appropriate Java version.

```
#!/bin/bash
export PATH=.:../fidocadj/jdk-9.0.4/bin
export CLASSPATH=.:../fidocadj/jdk-9.0.4/lib:../fidocadj
../fidocadj/jdk-9.0.4/bin/java -jar ../fidocadj/fidocadj_0
    .24.8.jar
```

The script must be made executable using `chmod +x fidocadj-0.24.8.sh` in the directory where the script is present. Of course, you will need to adapt the Java version and the corresponding directory to your needs.

A.2.4 Raspberry Pi 400

by Joop Nijenhuis

The Raspberry Pi 400 is a very cheap computer. Being cheap does not mean small system. This is what might be the future. Inside a 64-bit quad-core ARM processor with 4Gb ram, Gigabit Ethernet, Wifi, Bluetooth. It has 2 HDMI video ports, so it can be connected to almost everything and it uses a microSD card as main drive. There are 1 USB 2.0 port and 2 USB 3.0 ports. Power through an USB type-C port. There is an extension port for experiments. It uses only 15 Watts of energy, probably much less because you don't use everything. This type of computer is promoted among youngsters in order to get them programming and give them more understanding what a computer is and can do. Raspberry Pi comes with its own Debian based OS distribution called NOOBS. On the card is also OpenJDK Java version 11.x.x. All is already installed! You could insert the microSD card in the slot and run the system, if that's wise is another matter. If you have a different background you might find Linux not easy. Be aware that you can do it differently! You can go the way described above or you can do it my way. For some reason the Raspberry Pi community doesn't like Java, don't know why, so no answers there. I found "native" solutions not better and sometimes worse then what I already had running in Java. FidoCadJ is one of them and runs fine with/on the Pi. How to proceed? Make in /home/pi a new directory and give it the name "Apps" or something similar. Open this directory and make again a new directory with the name "fidocadj". Copy the download "fidocadj-0.24.8.jar" into this directory.

Start the Text editor and copy next lines in it:

```
#!/bin/sh

#go to script's directory
progdir=${0%/*}
cd $progdir

#launch fidocadj
java -Duser.home=$progdir -jar fidocadj-0.24.8.jar 2>
    fidocadj-bugs.txt
```

Save it as "fidocadj.sh" in the same directory as FidoCadJ. You can add this file to the "Main Menu Editor" program in the NOOBS distribution. With some modification this setup can be used for every other Java program. The "-Duser.home=$progdir" sees to it that always "$progdir" is used. The "2¿fidocadj-bugs.txt" sends every problem to the file "fidocadj-bugs.txt". Don't worry, file stays empty on my system, but in case something did go wrong I have probably some hints where to look for the problem.

The beauty of this system is that it always works (up to now) and that you can find ALL your stuff about the program in ONE directory, in case of trouble or if you just want to erase all. Don't worry about OpenJDK Java, its version 11 and newer versions are in the make. As it is part of the NOOBS distribution it will get updates every time you update the Pi system and if there is an update for Java. Another thing: I work with a Wacom Tablet and a pen as mouse. Works very well with FidoCadJ. If you want to read more; https://forums.raspberrypi.com/viewtopic.php?t=320892

Table A.1: List of make targets available to compile FidoCadJ from sources.

Rule	Description
`make`	Compile FidoCadJ (implicit)
`make clean`	Erase all the compiled classes
`make cleanall`	Do a clean, erase fidocadj.jar, Javadocs
`make compile`	Compile FidoCadJ (explicit)
`make createdoc`	Run Javadoc on all source files
`make createjar`	Prepare jar/fidocadj.jar
`make rebuild`	Do a clean and then run FidoCadJ
`make run`	Run FidoCadJ

A.3 Windows

When FidoCadJ recognizes a Windows platform, it will try to use the native Look and Feel.

A.3.1 How to download and execute FidoCadJ

Very often, if you have Java installed, you just have to download the `fidocadj-0.24.8.jar` file and run it with a double click. If after the download your operating system sees it as a `zip` archive, probably Java is not available on your computer. Oracle allows you to download and install the current version of the Java runtime for free (substitute "it" for "en" or the designation for your country if necessary): http://www.java.com/it/download/ OpenJDK can be downloaded from: https://bell-sw.com/pages/downloads/

A.4 Compile from sources

If you want to compile from sources FidoCadJ, fortunately it is a simple task, as it only requires a functioning Java SDK and GNU make. More details are given in the README.md file. The FidoCadJ project has a build automation system, based on make. You first have to download the full source repository from GitHub. The make rules show in table A.1 have been implemented and can be used from the command line. If you want to study the FidoCadJ source code, run `make createdoc` to obtain the Javadoc description that will be placed in the `doc` subdirectory.

If you use Windows, you may find useful to use the `winbuild.bat` script instead of `make`. The script is placed in the `dev_tools` directory and should be used with an action chosen among `{run|clean|compile|force|rebuild}`.

A.5 Android

A specific version of FidoCadJ for Android has been developed in 2015. It shares much of the source code with the PC version, but all the graphical user interface code had to be rewritten. It should work on a variety of devices with at least Android 4.0: smartphones as well as tablets. For example, figure A.2 shows the appearance of FidoCadJ for Android with a Samsung S5 smartphone.

There are some minor differences between the Android application and the PC application described in the manual. They of course are perfectly compatible with the file format and can exchange drawings.

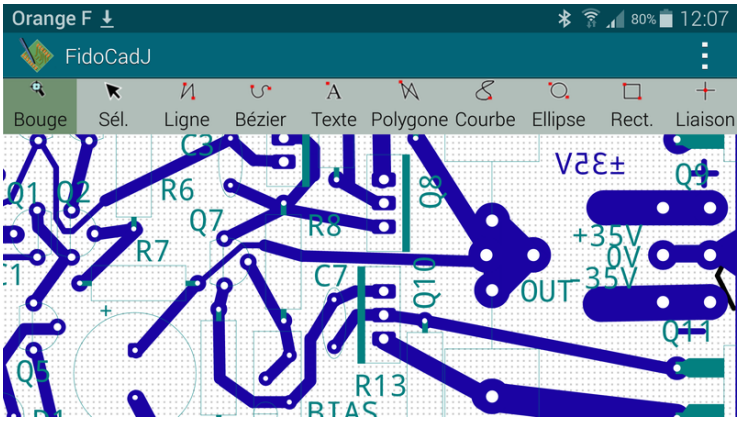


Figure A.2: FidoCadJ for Android, running on a Samsung S5 smartphone.

- You can not create and edit user libraries, but you can use those built on a PC.
- You store the user libraries in the external filesystem (the "SD card"), in the `FidoCadJ/Libs` directory.
- The symbols are not shown all the time to save space: you have to sweep your finger on the right of the screen to show them.
- You have an additional button in the toolbar, which allows you to move and zoom the shown portion of the drawing. While in this "Move" state, touch the screen and move your finger to pan the drawing. Use two fingers and "pinch" them, to increase or decrease the zoom.

However, in 2020 the future of the Android application is rather grim. First of all, it seems that the only effective way to distribute the application is Google play. A developer account to publish an application has a cost. Alternatives exist, but are not widespread. Moreover, the application was developed in 2014 using techniques that are now apparently being phased-out by Google. More importantly, maintaining a modern app is just not something that I want to do alone, given the time I have chosen to dedicate to FidoCadJ. If someone wants to take care of it, please step forward.

Appendix B

FidoCadJ examples

and how it can be otherwise...

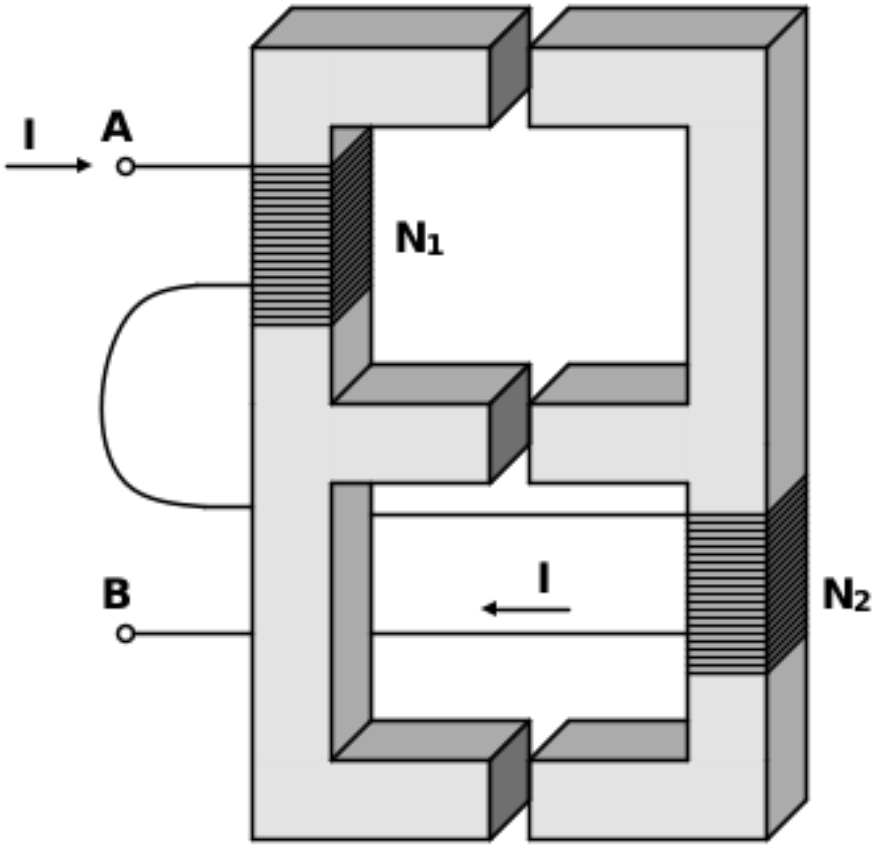


Figure B.1: User example DanteCpp (Austria)

This example is from https://github.com/DarwinNE/FidoCadJ/issues/16 and was originally created in an Ubuntu Linux system. I copied the code from the forum post and pasted it into an ascii text editor and saved it with a file name and the extension ".fcd". Then read into FidoCadJ on a Raspberry Pi 400 (Netherlands). The only thing I had to fix was the "N1" and "N2" captions where the numbers were put on the letter "N". The rest of the drawing is unchanged. Then exported to a png image in FidoCadJ. And this is the result.

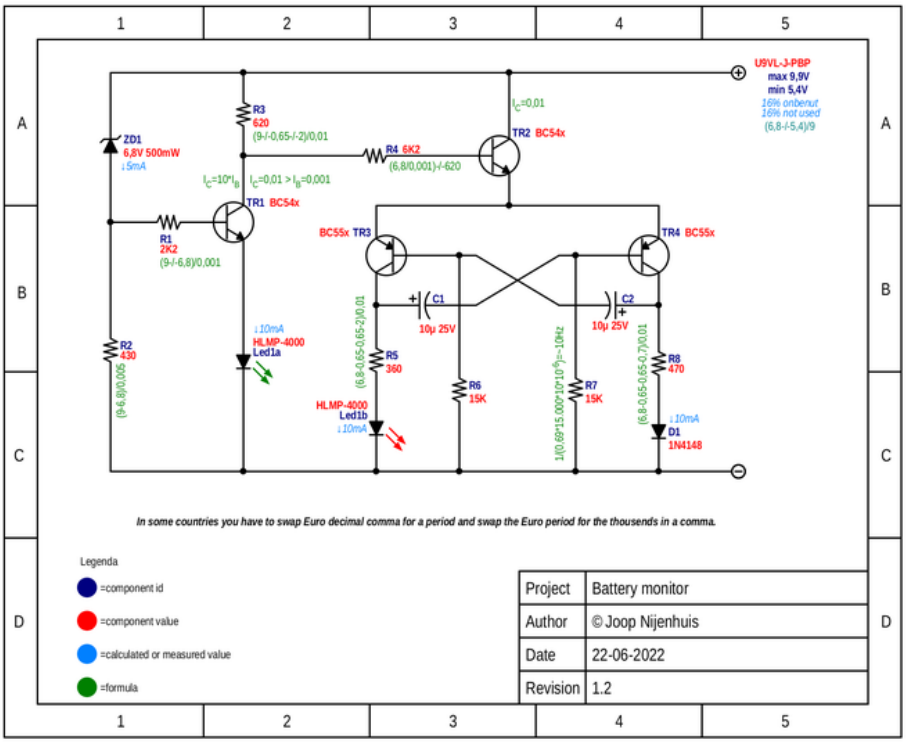


Figure B.2: An example of a FidoCadJ drawing with different use of the available layers (read colors).

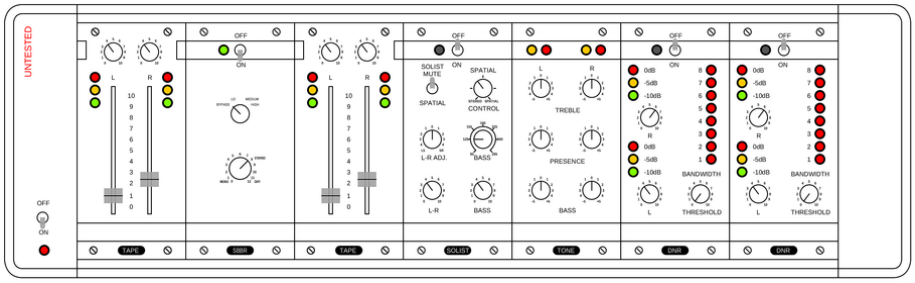


Figure B.3: Modular compound effects rack. Zooming in on the page (>200%) makes buttons and texts more readable.

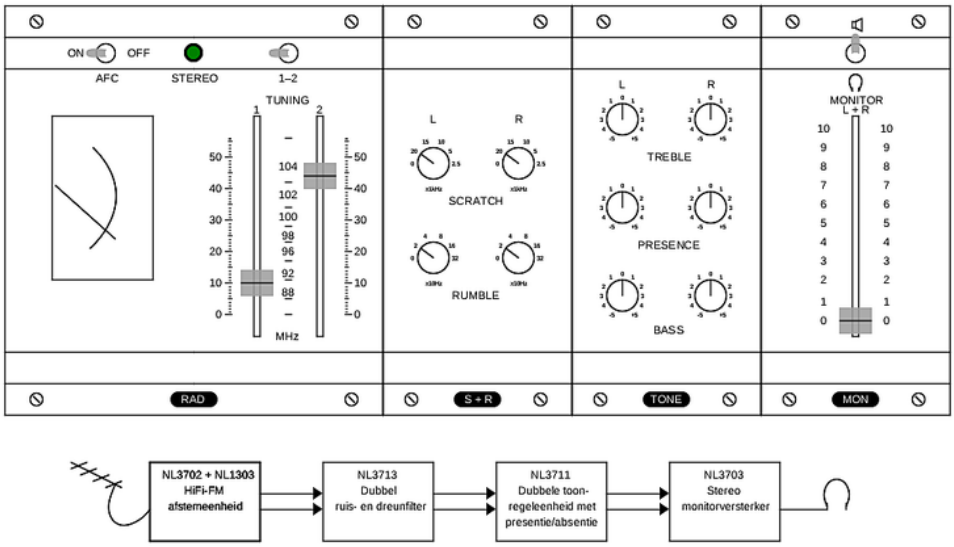


Figure B.4: Tuner assembled from Philips mixing amplifier units. Zooming in on the page (>200%) makes buttons and texts more readable.

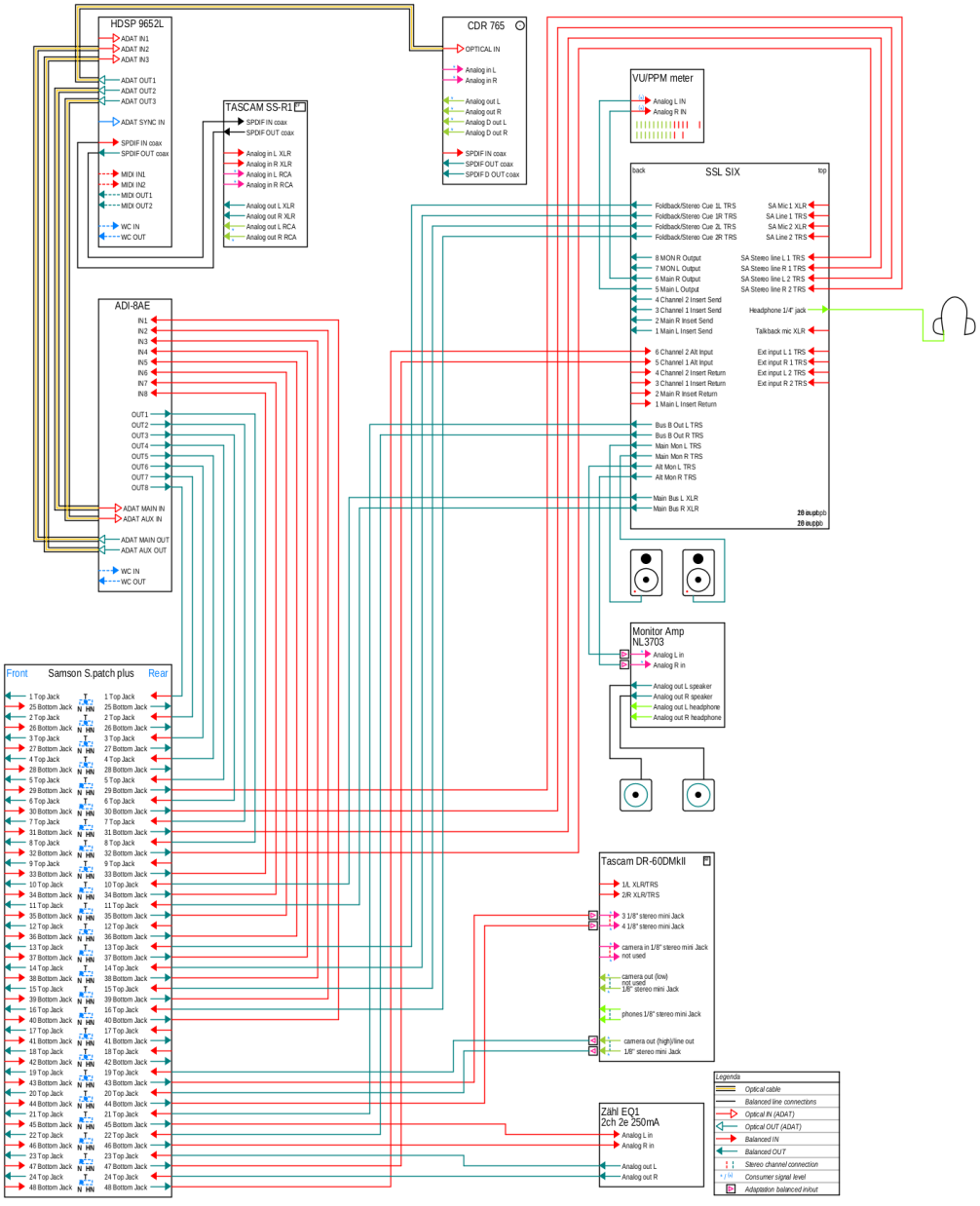


Figure B.5: Example of a sound studio layout (the actual set-up is much more complicated and does not fit on an A4, the idea will be clear). Zooming in on the page (>200%) makes texts more readable.

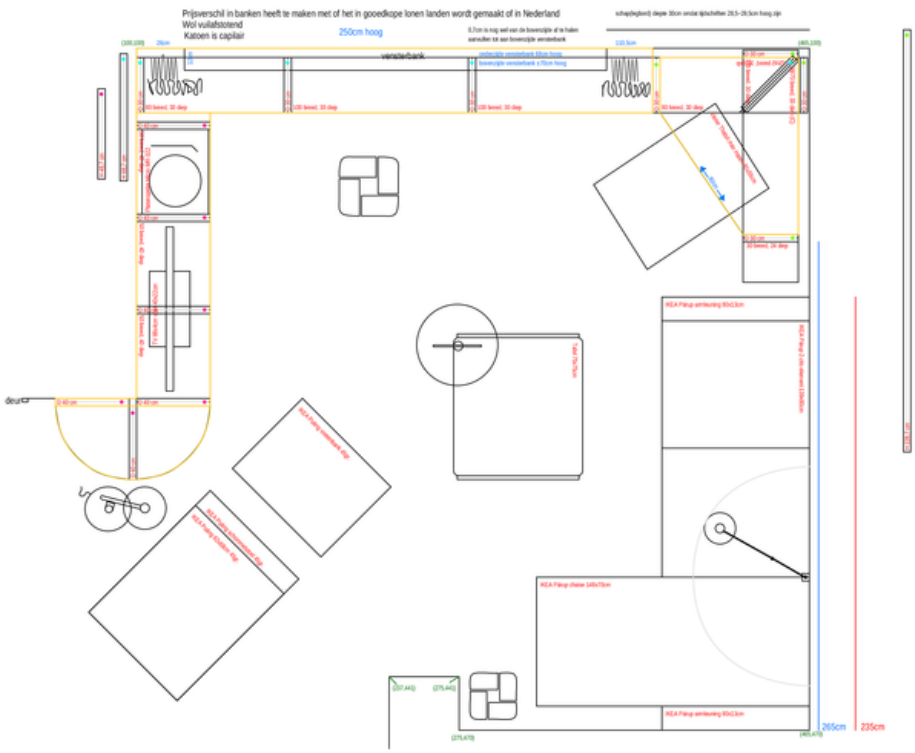


Figure B.6: My attempt at a new layout of the living room.

Appendix C

FidoCadJ art

One of the fathers of circuit analysis, in a portrait by Zeno Martini.

Tiger, tiger, burning bright... By Igor Arbanas “elettrodomus”.

Index

This manual has been written using pdfLaTeX on macOS and updated on a Raspberry Pi 400. Listings have been composed by using the listings packet. The pgf packet has been used for the figure 2.6 as well for the picture at page 60. See the preamble of the tex-file for the needed packets (with some comments). All those packets are available on the CTAN archive.