FidoCadJ 0.23.2
the user manual

Davide Bucci

March 15, 2010

ABSTRACT

This document is the FidoCadJ official user manual. After a short introduction of the history and the birth of this software, we will describe the basic use of FidoCadJ. Our goal is to learn how to draw very simple electronic schematics and their printed circuit boards. The manual will end with a detailed description of the FidoCad (and thus FidoCadJ) format, which has not yet been documented.

ACKNOWLEDGEMENTS

FIDOCADJ LICENSE

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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.

1.1 FIDOCADJ'S PHILOSOPHY

FidoCad (without the J at the end) is a vectorial drawing software particularly suit for electrical schematics as well as printed circuit boards. It is diffused in particular in the italian Usenet community from late 1990s.

It can 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 this software is a very compact textual descritpion. This feature makes it very easy to include drawings in text messages, such as those used in non binary Usenet groups.

Unfortunately, FidoCad exists only in a Windows version. Who uses Linux can run it using WInE, but for those using other platforms like me (I use MacOSX) have to find a different solution. I thus decided to give a small contribution to the Usenet 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.

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. At the time of writing, to the best of my knowledge, FidoCadJ is almost completely compatible with the original FidoCad, apart from a few details. In particular the library and macro handling could probably be improved.

My goal has been to offer a minimalist yet complete solution to drawing needs of small schematics as well as simple printed circuit boards (P.C.B.). I tried to respect as much as possible the original FidoCad philosophy, which was to use a simple tool which was completely open. Of course, my middle term goal is to achieve 100% compatibility with the original FidoCad for Windows.

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 vectorial formats, including Encapsulated PostScript (EPS). Another file format useful for electronic schematics is the CadSoft Eagle script, which is available from version 0.21 of FidoCadJ. In this way, a schematics drawn with FidoCadJ can be exported to Eagle. The appendix A briefly describes how to install FidoCadJ under the most diffused operating systems.

1.2 HISTORY OF THIS SOFTWARE

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 avoided awkward ASCII drawings. Since I do not use Windows since a few years ago, 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 work at a solution rather claim that an operating system different from Windows lacks in software

The first thing I did was to study the file format used by FidoCad and write a Java applet called FidoReadJ, able to parse the circuit and to show it in a web page. I started searching more or less everywhere (old posts, web pages), doing a lot of reverse engineering from existing FidoCad files. I downloaded the FidoCad sources, written in a pretty neat C++ by Lorenzo Lutti.

I did this work more or less around March 2007. A few moths later, the applet was on line and it was being tested by part of the community gravitating around it.hobby.elettronica and it.hobby.fai-da-te.[1]

Since I had an interpreter of the FidoCad format, it was interesting to continue the work in order to obtain a complete editor The most part of the work was done in several steps, between January and July 2008. FidoCadJ is not an adaptation or a porting of FidoCad for Windows, but it is a completely rewritten program.

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

The choice of using Java is due to the fact that in the last few years I changed a lot of operating systems. Spending time and energies on something which is not completely portable does not appeal to me anymore. The effort of learning the Cocoa framework would have probably given a better result under MacOSX, 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, a simple analysis of the FidoCadJ code source shows that I am not a very Java and object oriented programming purist and I am sure that several solutions could be found that are more practical than elegant.

What matters is the end user impressions while using the program, 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. To summarize, I am aware that Java is not the perfect choice or the solution to every problem. However I am sure that its bad name derives mostly from badly written applications which are not very well integrated with the user desktop.

Without aiming for perfection and knowing my programming skills limitations, 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.

Since november 2009, I opened a SourceForge project dedicated to FidoCadJ. From this page, you can download all executables, manuals as well as the source code. You can actively participate to FidoCadJ development working on the source code using Subversion (SVN), or the SVN browser provided by SourceForge:

http://fidocadj.svn.sourceforge.net/viewvc/fidocadj/

If you want to help me 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 existing manuals for inconsistencies and errors. You can also work on the standard library. . . On SourceForge, you can participate to the forums, write a program review as well as suggest improvements or give a bug report.

---

[1] FidoReadJ is still available, even if not regularly updated, at the address: http://davbucci.chez-alice.fr/index.php?argument=elettronica/fidoreadj/fidoreadj.inc.

DRAWING WITH FIDOCADJ

The use of FidoCadJ should be quite intuitive for those who have ever used a vectorial drawing application. A screenshot of the program running under MacOSX is shown in Fig. 1; A few details may be different when running under other operating systems (for example Fig. 2 shows the result with Look and Feel Metal on Sun), but the philosophy remains the same. We will see what the features of the program are, and its basic elements (the primitives) which compose a FidoCadJ drawing.

2.1 DRAWING TOOLS

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

The command bar 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 the menu "View/Options".[2] Any change in the settings will be applied at the restart of the application, since it is arguably something that we may want to change every day. Figures 1 and 2 show the command bar (right below the window's title), configured to show text and icons in their smallest version. On the second row, starting from the left we can see the zoom settings and the buttons "Fit", "Show grid" e "Snap to grid". The first allows us to automatically select the most suitable zoom settings in order to show the whole drawing on the screen. The second toggles between visible and invisible grid, while when we press the third button the elements added will stick to the nearest step of the grid.

On the right are shown in a tree list the elements (called macro) of the libraries loaded into the application. To insert an element from the library we only need to 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.

From version 0.22, you can do quick research inside the libraries charged in FidoCadJ. You just need to type something in the text field which appears just over the tree representing the installed libraries (have a look to figure 3) By typing up and down keys, you can navigate through the results found.

Figure 4 shows an example of what we can obtain 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 (coordi-

---

1 Form more information on the format used to save the drawing elements see 3.3.
2 Except on MacOSX, where this item is found in the FidoCadJ's menu and it is called "Preferences".
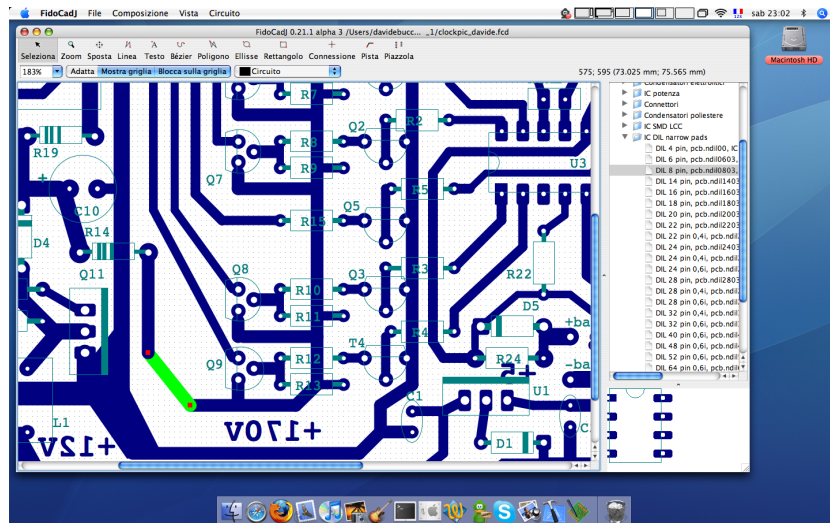
Figure 1: A typical FidoCadJ session running under MacOSX Tiger. Appendix A describes the peculiarities of the version specific for Macintosh.
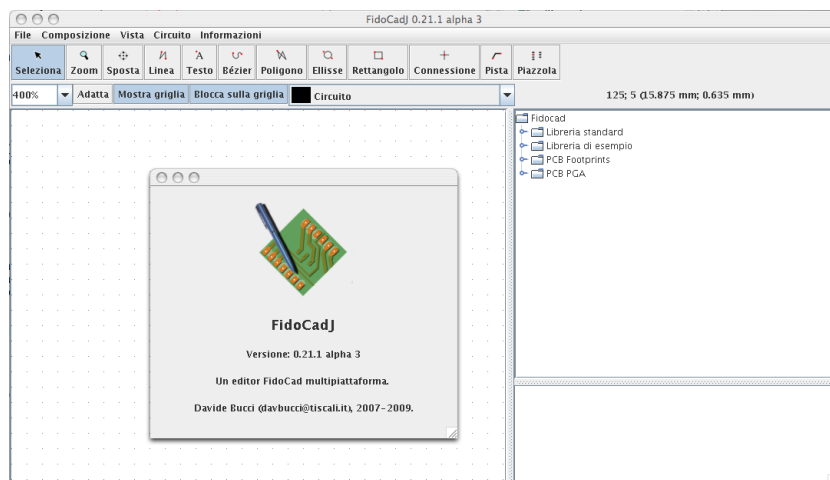
Figure 2: FidoCadJ with the Look and Feel Metal.
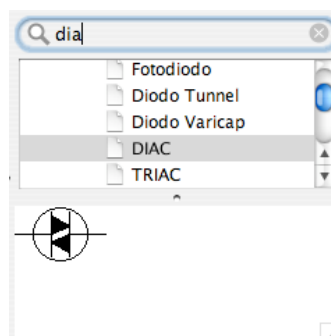
Figure 3: The quick search function in the installed libraries.

4

Key	Comando	Use
A or Spacebar	↖ SELECT	Selects one or more graphic elements. Press Control (Command only under MacOSX) 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	Inserts a line or a series of lines. Press Esc or double-click to terminate the insertion
T	˙A TEXT	Inserts a text string.
B	↶ BÉZIER	Draws a Bézier curve
P	⋈ POLYLINE	Draws a polyline filled or empty. Double-click or press Esc, to terminate the insertion of new vertices.
E	◯ ELLIPSE	Draws an ellipse filled or empty (hold Control to draw a circle).
G	▭ RECTANGOLE	Draws a rectangle filled or empty.
C	＋ JUNCTION	Inserts an electrical junction.
I	⌐ PCB TRACK	Draws a PCB track. The default width can be modified through the dialog accessed from the menu "View/Options".
Z	⦂⦂ PCB PAD	Draws a PCB pad. The default dimensions can be modified through the dialog accessed from the menu "View/Options".

Table 1: Summary of the drawing commands available in FidoCadJ. The key shown on the leftmost column allows their rapid selection using the keyboard. A right click in one of the primitive placement modes allows us to access the properties window.
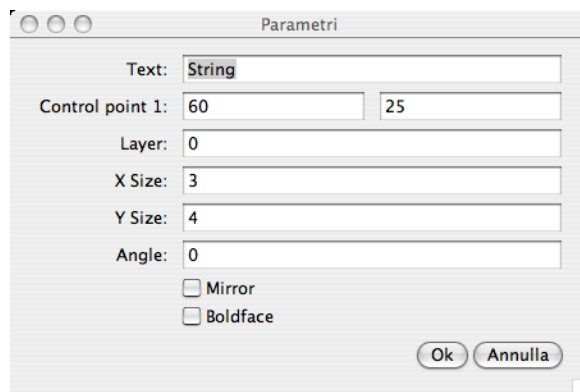
Figure 4: Dialog for the text's parameters in a FidoCadJ drawing.
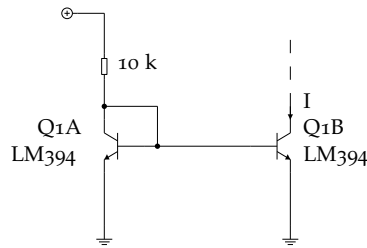
Figure 5: The reference schematic: a current mirror made with NPN transistors.
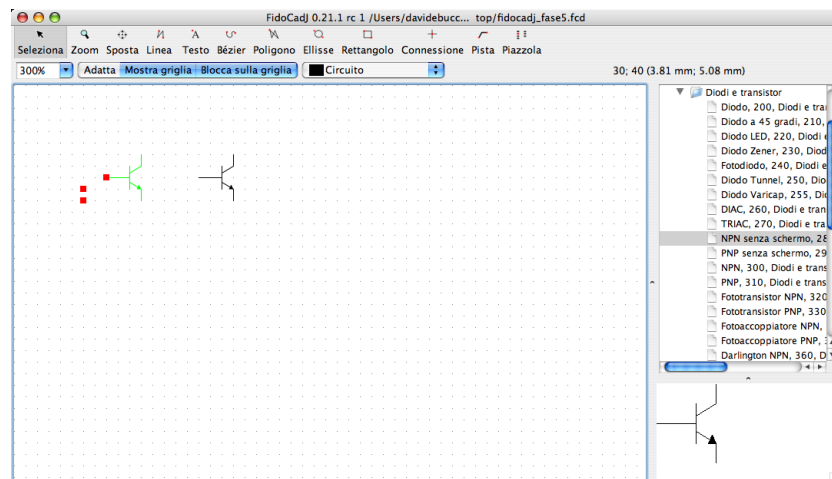
Figure 6: We start by drawing a couple of transistors.

nates, rotation. . .) of any drawing primitive. The aspect of this window will change because the information that can be modified will depend on the selected element.

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 5.

Once FidoCadJ is running, let us create a new drawing from the menu "File/New".

We will then start by placing in the drawing area the symbols for the two transistors, around which our schematic is built. To do so, we will need the macros contained in the standard library, which is loaded by default and is found on the right-hand side of the screen. The macro that we will 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 macro, 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 the stage similar to the one shown in figure 6.

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, select the transistor (which will be highlighted in green, with three control points identified by small red squares) and then press S
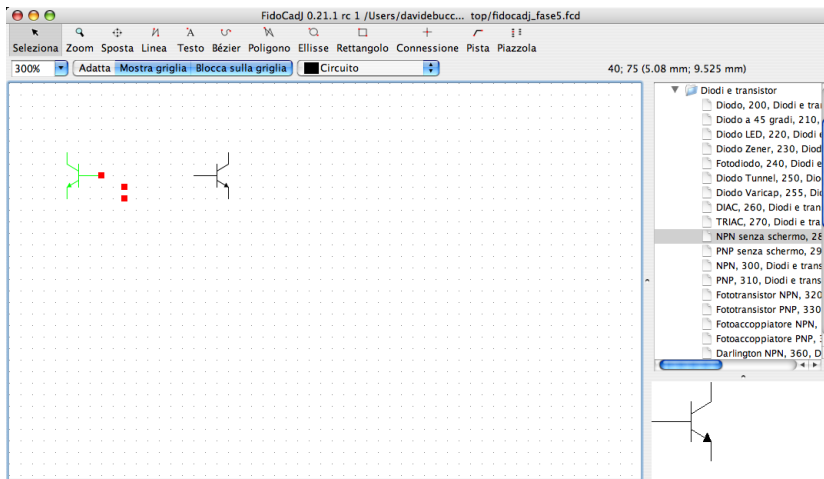
Figure 7: Select and mirror with $\boxed{S}$ the transistor on the left.
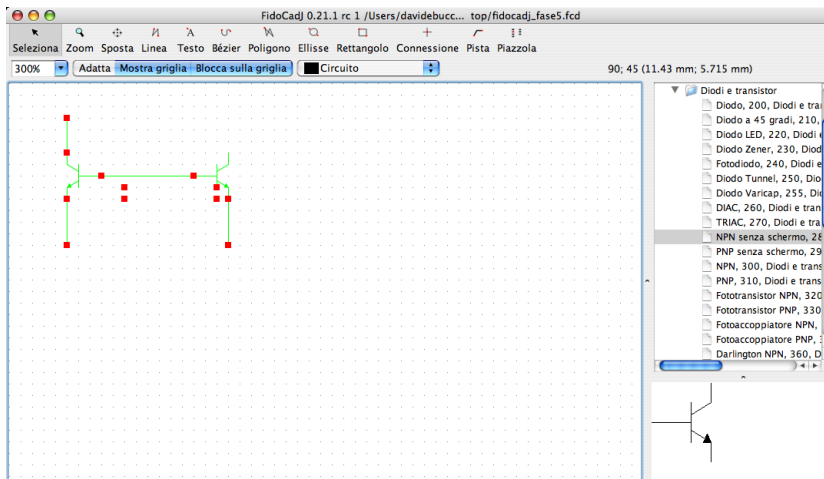
Figure 8: We are too close to the top edge of the sheet: let's select the whole drawing and move it toward the centre.

to obtain its mirrored version. We will thus obtain a result similar to the one shown in figure 7.

By using the tool "Line" from the toolbar, we will be able to make a few electrical connections, until we will realize that we started our drawing to close to the edges of the drawing area. The issue 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 pressed, drag the cursor up to the lower right corner. A rectangle with a green contour will appear 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 will have to select them all first(see figure 8). Now, still in select mode, we can click on any selected element to drag the selection to the desired position.

We can then continue placing the other parts of the circuit, in particular a resistor (Standard Library/Discrete devices/Resistor) and the label for the positive power supply (Standard Library/Basic symbols/Terminal +). We will need to rotate the latter in order to place in the desired position. Again, we can select it and press $\boxed{R}$ until we will
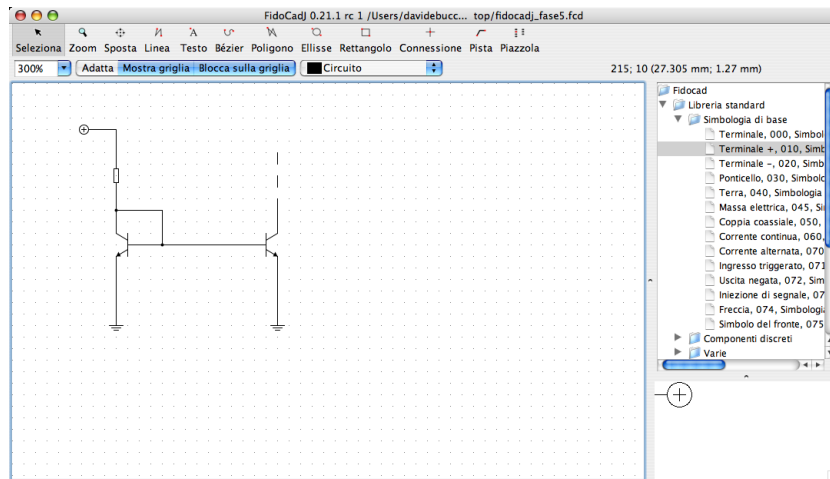
Figure 9: The circuit almost completed.
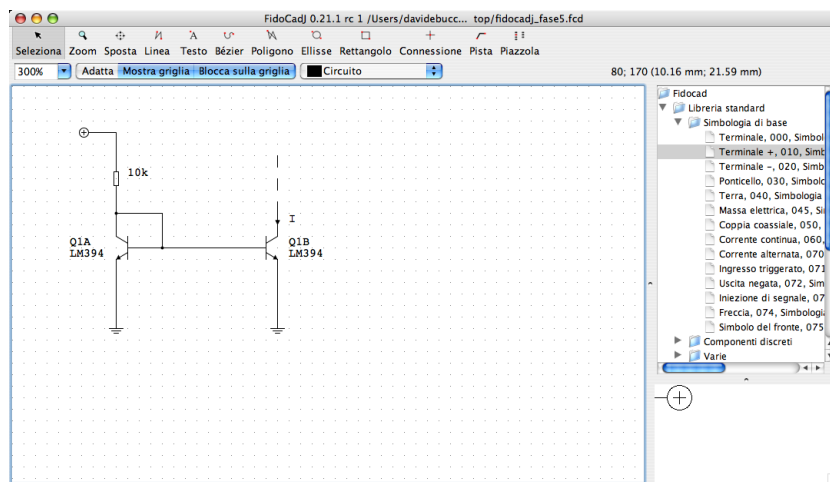
Figure 10: The final circuit.

obtain the desired result. We should now have a screen similar to the one in figure 9.

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 macro 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" will be introduced, we will need to modify its characteristics by double-clicking in select mode (see figure 4). The transistor's part number and name used (it is actually a transistor matched pair in our example) are specified in the fields "Name" and "Value" which are accessed in select mode by double-clicking on the macro.[3] 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 10.

As a curiosity, this is how the code which describes the circuit of our example looks like. To access this code, it is sufficient to select "Insert

---

[3] The feature which allows us to add a name and a value to a macro or a component's symbol is actually an extension introduced with FidoCadJ not present in the original FidoCad. See section 3.4 for more information on compatibility.

8

Circuit" 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.

```
[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
```

If you are interested in the export format used by FidoCad, there is a detailed description on chapter 3.

However, there is no need to use the "Insert circuit" dialog; we 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 will be added automatically.

2.3 THE LAYERS

A way to picture a layer is that of a drawing made on acetate sheets. The final drawing will be given by the combination of all the layers, which will be superposed like acetates. Every layer is characterized by a different color and can be visible or hidden. This approach is common to many CAD packages, as it allows an easy representation and management of different parts of the drawing that will be superposed, as for example in a PCB design.

FidoCadJ allows up to 16 layers, numbered from 0 to 15. Conventionally, some of the layers have a specific purpose. In particular, layer zero is 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 will be drawn first. That is, drawings on successive layers may cover the ones on lower layers.

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 us to obtain a resolution sufficiently fine to draw an electrical schematic and the majority of PCBs. However, for ease of drawing, the application allows us 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, "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 that can be accessed from the menu "View/Options".

2.5 A SIMPLE PCB

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

The reader will find their way through: a few scribbles made with paper and pencil (and a lot of erasers) will result in time saved by obtaining a clear idea to be developed with the use of the computer.

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. There is no doubt that these are still a 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. 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 and how to draw all the tracks so that they will cross each other as less as possible .

Here we will cheat a little and we will start from the result we want to obtain, as shown in figure 11. It is a simple common-emitter amplifier built around an NPN transistor, type BC547 or similar. It will be 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 will be 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 12.[4]

---

4 FidoCadJ is currently available in Italian, Italian with Swiss variants, French and English. It is quite easy to translate menus and commands, so if you would like to see FidoCadJ
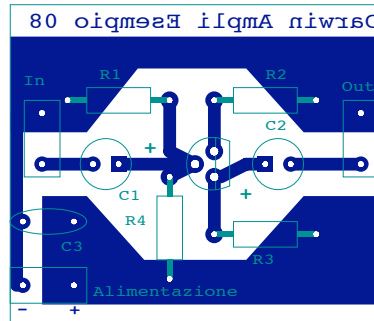
Figure 11: A very simple amplifier stage using an NPN transistor connected in a common emitter configuration.
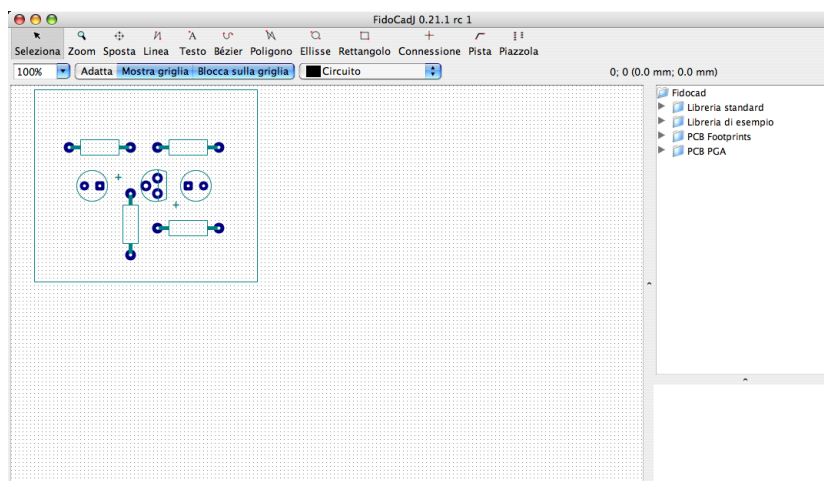
Figure 12: The most important devices are placed on the board.
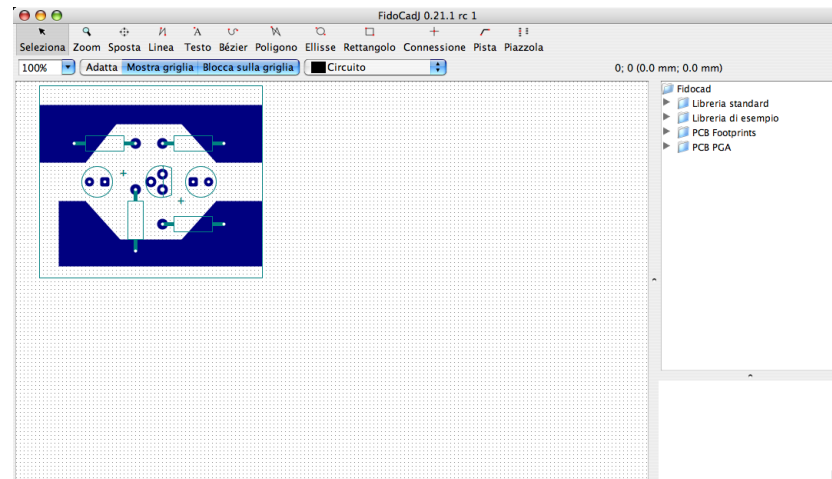
Figure 13: Added the power supply connections using polygonal lines.
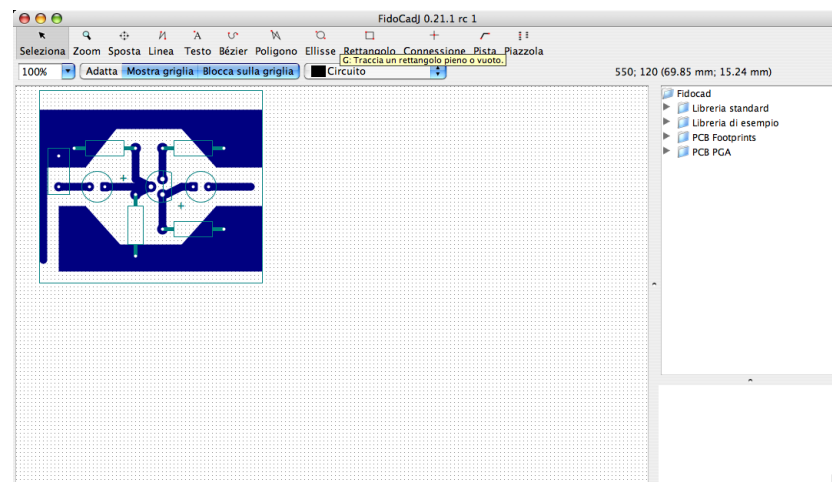
Figure 14: Added the remaining connections PCB tracks.

We can then introduce the copper areas which will provide the positive and negative power supply. These are specified by drawing a polygon (using the *poly line* primitive) and double clicking inside the area 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 will have a low stray inductance. We should be able to reproduce the result shown in figure 13.

To complete the electrical connections we can use the *PCB line* primitive. I chose a track thickness of 10 unità (1.27 mm), which helps the soldering process.

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 will probably have the right dimensions. Let's not forget that FidoCadJ is thought as a replacement for the transfers. . .

---

in your native language, please let me know! There is also work to be done to translate the libraries (at least the standard ones) and of course this manual.
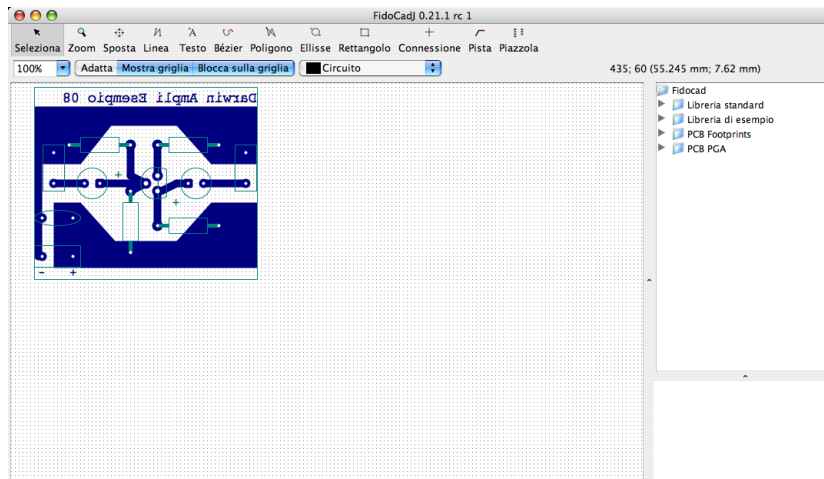
Figure 15: The PCB almost completed.
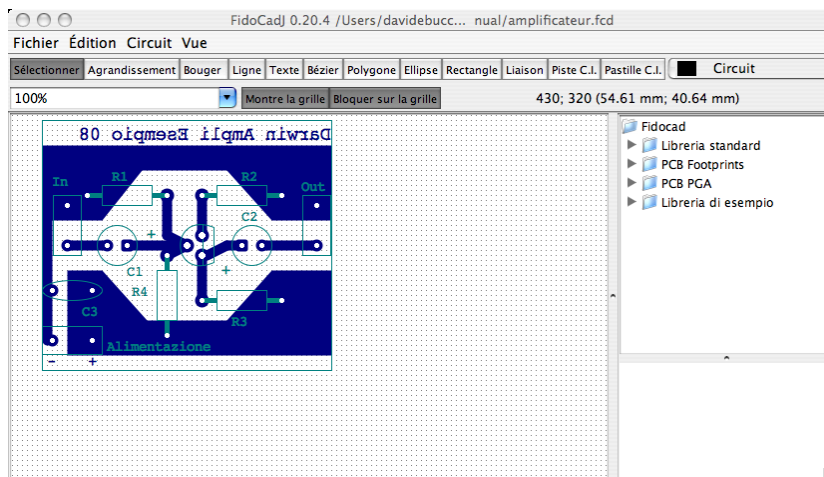
Figure 16: The job completed with the silk-screen.

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 side as well. To write on the copper layer, after selecting the proper layer it will also be 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. It will need a few tries to obtain the right dimensions 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 15 shows the result obtained using 11 units for the horizontal and 18 units for the vertical dimension of the text.

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 the soldering.

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 16.

Once the drawing is completed, we will 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

Figure 17: The PCB, as it appears when printed (mirrored) on a ISO-UNI A4 sheet.

it will be sufficient to hide the layer 3 containing the silk-screen. The program will show the copper layer only.

We will 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 will occupy only a small corner of the sheet (assuming a standard ISO-UNI A4), as we can see in Figure 17.

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
```
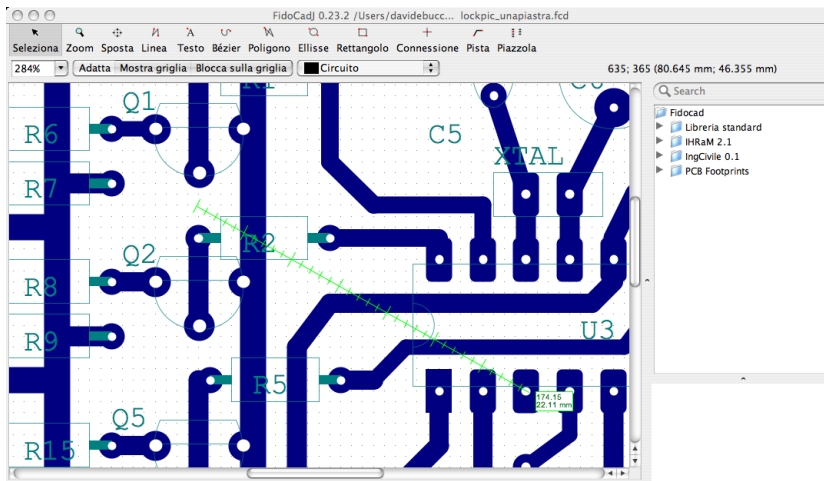
Figure 18: Left click and drag to activate the FidoCadJ ruler.

```
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 card. FidoCadJ offers (from version 0.23.2) a ruler feature which allows you to easily perform those tasks. Just right click and drag. You should obtain a green ruler, like the one shown in figure 18. If the right click and drag action is not used in your operating system, you can alternatively left click and drag while pressing the Shift key. The total length measured is 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).
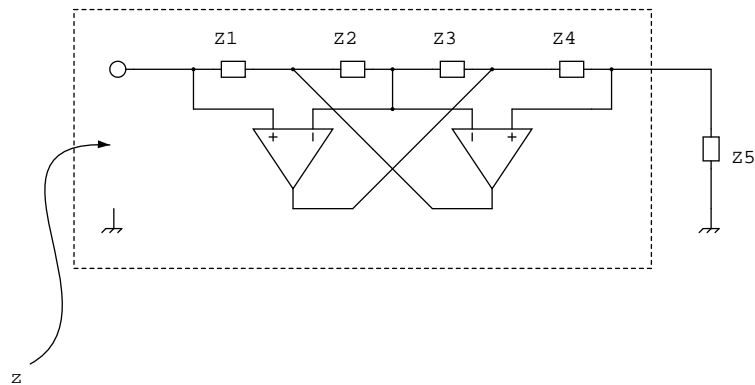
Figure 19: An electrical drawing (an Antoniou's GIC) in which some FidoCadJ extensions have been used.
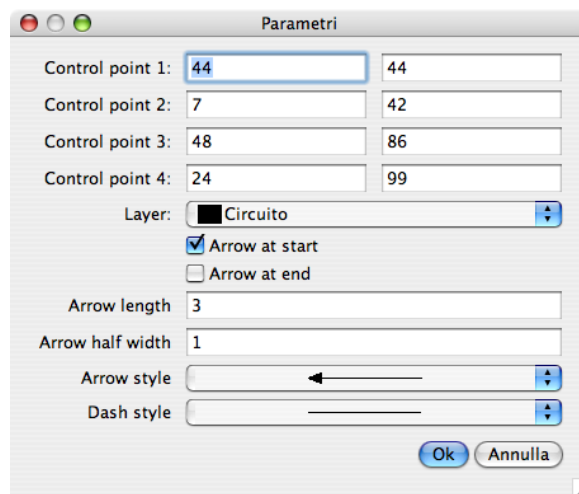
Figure 20: The parameter window of the Bézier curve shown used in the schematics of figure 19

2.7 ARROW AND STROKE STYLES

Starting from FidoCadJ version 0.23, the software allows to draw arrow heads at the beginning and at the end of lines and Bézier curves. FidoCadJ allows you 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.

A few stroke styles are now available for technical or mechanical drawings. Figure 19 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 parameter window which should be similar to the one shown in figure 20. You can notice that the option box "Arrow at start" has been checked. FidoCadJ will thus trace 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.

The possibility of chosing 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 com-

pletely backward compatible with FidoCad per Windows. When you do chose to use a FidoCadJ extension, you need to know what you are doing. 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 will not allow you to introduce graphical elements which 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

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 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 vectorial or bitmap. Whenever possible it is better to choose a vectorial format to obtain the best results.[5]

A vectorial format stores the primitives that compose the drawing. A bitmap format works on a matrix of pixels.

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 vectorial file format.

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.

2.9 COMMAND LINE OPTIONS

The application is distributed as a file `.jar`, which is a Java archive.[6] 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 Sun'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 1.4, 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.jar
```

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

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

FidoCadJ will be run and it will try to open the file `~/FidoCadJ/test.fcd` (provided that this exists).

Another interesting option, although a feature due to Java more than FidoCadJ, is the possibility to modify the look of the application (in the

---

5 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.

6 Except for the Macintosh version, which is a stand alone application.

Format	Comment
JPG	Very diffused bitmap format. Since the compression used is lossy, it is not suitable to for exporting FidoCadJ schematics. It is made available due to its diffusion.
PNG	Compressed bitmap format, suitable for exporting schematics. This is probably the best way to export a FidoCadJ drawing when a vectorial format cannot be used.
SVG	W3C standard vectorial format. Some internet browsers (such as the recent versions of Safari) allow its visualization within a web page. Very good format for graphics and schematics, it enables the use in applications such as Inkscape to modify the drawings made with FidoCadJ. There are currently some limitations with the exportation of rotated and mirrored text.
EPS	Encapsulated Postscript vectorial format. Very useful to those who use professional graphics applications, or want to use a FidoCadJ drawing in a LaTeX document. The drawings exportation should be fully working. This is the option used to obtain Figure 11, page 11 in this manual (passing through a PDF conversion, since I use PDFLaTeX).
PGF	Vectorial format to be used directly in a LaTeX document, when using the *pgf* package, available in the CTAN archive. This exportation option was thought in particular to export schematics and uses an easily editable script. The text attributes will not be translated. This allows the introduction of LaTeX code directly into the drawing and it is the technique used in this manual to obtain Figure 5, page 6.
SCR	Starting from version 0.21, 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 yet. These will not be exported to the Eagle script.

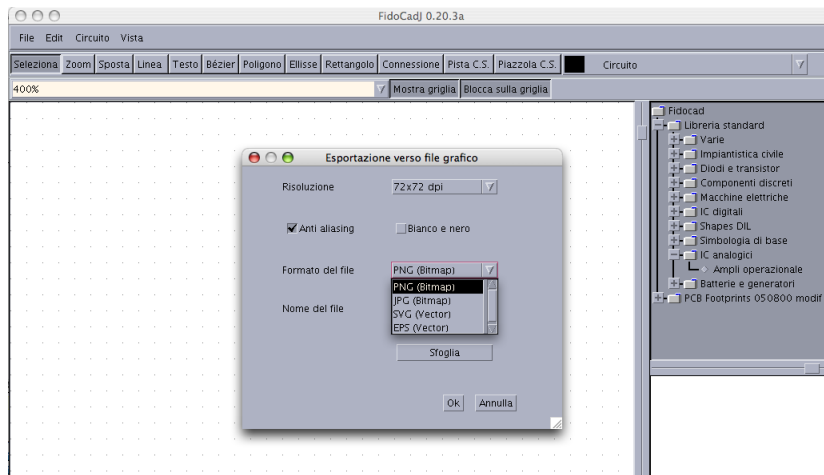Table 2: List of all the export file formats available in FidoCadJ.

Figure 21: The appearance of the program under MacOSX, using the Motif look & feel.

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 21:

```
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.

2.10 LIBRARIES MANAGEMENT

The program allows us to specify a directory in which all the library files are placed (extension `.fcl`). This can be done through the menu "View/Options". If a file named `FCDstdlib.fcl` is present, its content will supersede the standard library directly available in the application. Analogously, if a file named `PCB.fcl` is present, its content will replace the PCB library[7]. Other files with `.fcl` extension are considered as add-on libraries and loaded together with the standard library. This is done at start time, which means that any modification will take effect when the application is restarted. From version 0.23, thanks to Roby IZ1CYN, I could include the IHRaM 2.1 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 will be loaded at the place of the version embedded in the program.

Other files with `.fcl` extension in the library search path will be considered as libraries and FidoCadJ will try to load them when it is

---

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

starting. This is done only once, so if you need to add or remove libraries in the directory you use for them, remember to restart FidoCadJ.

From version 0.23, FidoCadJ allows to split non standard macros (as the original FidoCad does). This can be very useful when posting a drawing in a newsgroup, since in this way each macro which does not belong to the standard FidoCad libraries is expanded into its graphic primitives. The reader thus does not need to have the very same libraries you have installed in your system to read your post. To activate this behaviour, you have to activate "Split non standard macros" in the "FidoCadJ extensions" tab of the "Preferences" window. You can choose if you want to do that when saving files or when doing a copy/paste of parts of the drawing.

3

DRAWING FORMAT, MACROS AND FIDOCAD 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 which has the advantage of being very compact and efficient. Since the format has never been described in detail in a document, I will try to summarize all that I learnt about it. Since I added a few extensions which are only available on FidoCadJ, I will try to describe them here as well.

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 us to obtain a good resolution for even the smallest SMD package without being too fine for everyday use.

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 PRIMITIVES

FidoCadJ can manage 11 drawing primitives as follows

- Line

- Filled or empty rectangle

- Simple text (obsolete)

- Advanced text

- Filled or empty polyline

- Filled or empty ellipse

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

Table 3: Meaning of the a parameter for the presence of an arrow head at the sides of a line or Bézier primitive.

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

Table 4: Meaning of the b parameter for the arrow head style.

- Bézier curve

- Electrical junction

- PCB pad

- PCB track

- Macro

We will proceed by analyzing each of them. In general, every primitive 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

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

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

```
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
```

where a is an integer which represent the presence or not of the arrow heads at the extremes of the segment (have a look at table 3), b represents the arrow head style (as table 4). Paramenters c and d give respectively the total length and the half width of the arrow head, while e is an integer which gives the dash style.

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
```

where e is an integer giving the dashing style.

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 interpreted, it is not available on the toolbar. FidoCadJ will store this element exactly as it was advanced text and it will use 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" will be positioned. Notice that the layer information is missing. FidoCadJ will treat 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" will be positioned. The value of s_y and s_x indicates the horizontal and vertical dimensions of the text in logical units. I chose to let FidoCadJ respect the vertical dimension starting from the horizontal one, and that aspect ratio will be chanced only if this is strictly necessary. The text rotation is managed by the term a, expressed in sexagesimal degrees, while the value of s determines the text style, following table 5. 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 the symbol +.

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.

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

Table 5: Function of the bits in the text style term.

Empty or filled 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). . . 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 at 20, to avoid very long command lines.

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

```
FCJ  e
```

where *e* is an integer giving the dashing style.

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
```

where *e* is an integer giving the dashing style.

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 $P_1 \equiv (x_1, y_1)$, $P_2 \equiv (x_2, y_2)$, $P_3 \equiv (x_3, y_3)$ e $P_4 \equiv (x_4, y_4)$ 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

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

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
```

where a is an integer which represent the presence or not of the arrow heads at the extremes of the curve (have a look at table 3), b represents the arrow head style (as table 4). Paramenters c and d give respectively the total length and the half width of the arrow head, while e is an integer which gives the dash style.

Electrical junction

The primitive electrical junction 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.

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.

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.

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 calla macro is `MC`, and the call is done as follows:

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

The macro is drawn using position (x_1, y_1) as the reference, and the orientation is defined by the value of o (multiplied by 90°) 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*.

FidoCadJ extension: Starting from version 0.21, FidoCadJ introduced some extensions to the original FidoCad format, which are identified by the command `FCJ`. For the moment, the only one that has been implemented is a feature that allows the association of a name and a value to a macro. It is used as follows:

```
[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 command `FCJ` indicates that the definition of the macro is not finished, but the fields that indicate name and value are specified by the commands `TY` on the lines that immediately follow `FCJ`. So far, only the coordinates of the commands `TY` are in use.

This has the advantage that if a file containing such an extension is read by FidoCad, this will ignore the lines containing `FCJ` (returning an error message), but the drawing obtained will be identical to the one produced by FidoCadJ (see section 3.4).

3.4 fidocadj extensions

Since version 0.21, FidoCadJ has started to introduce a few refinements over the original FidoCad forma. 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.

FidoCadJ allows to activate a "strict FidoCad compatibility mode", in which all extensions are disabled. This way, FidoCadJ will be perfectly compatible with the original 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 will prompt a few errors and will give a result which is different from the original FidoCadJ drawing. The results will depend on what extension has been used: a dashed line will be rendered as continuos, the arrow heads will be not drawn. The text associated to the name and the value of a macro will instead be perfectly rendered.

Figure 22 shows what can be obtained using FidoCad to read the FidoCadJ file which describes 19. After ignoring the FidoCad errors, there are some details missing (the dashing and the arrow), but the overall drawing is still understandable.
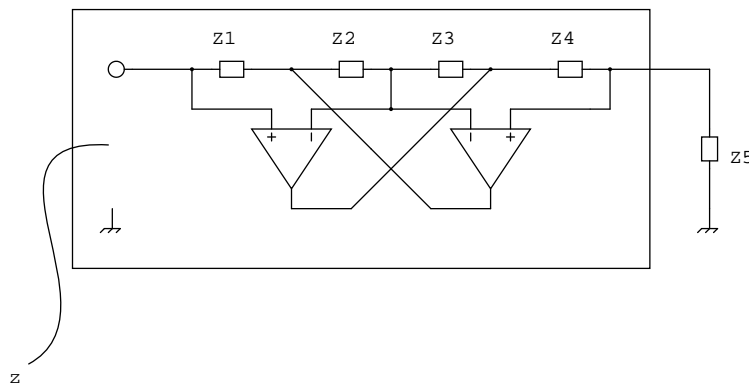
Figure 22: Figure 19 as it would appear under FidoCad for Windows.

3.5 SYNTAX ERRORS TOLERANCE

FidoCadJ is designed to tolerate errors or 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 will not be able to correct errors but it will simply skip (and delete) all the lines involved.

An exception to this behavior is that a number of primitives can be specified without the layer, which will be 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, will 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* (which must be unique within the library) and its description. The part name will be used within a FidoCad script, while the description helps the user while browsing all the macros contained in the file. The commands are nothing else than FidoCad drawings, where the coordinate point (100,100) is used as the origin. This is the point that will be used as the

reference when the macro will be called. In a FidoCad script a macro is identified by "library.macro" used with the MC primitive.

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

3.7 standard libraries

FidoCadJ already contains two libraries traditionally supplied with FidoCad. In particular these are the standard library and the library that contains the PCB symbols.

However, it is possible to override the content of these two internal library by specifying ("View/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 will be used in place of that of the standard library. If a file named `PCB.fcl` is present, its content will be used in place of that of the PCB library. Other libraries having different file names (but still with extension `fcl`) will be loaded together with the standard libraries.

CONCLUSION

4

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, you can write an email (davbucci@tiscali.it, but do not send me any attachment!).

A

PLATFORM-SPECIFIC INFORMATION

A.1 MACOSX

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 under MacOSX. Starting from 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 under MacOSX*

FidoCadJ can work with a MacOSX version more recent than 10.3.9 (Panther). The reason is that under the hood FidoCadJ needs at least the 1.4 version of Java, which Apple provides with this operating system.

Even if you can use directly the Java archive `fidocad.jar` as you would do under other operating systems, under MacOSX you can use the specifically tailored application. Everything works just like a native application: you can download the disk image at the following link:

http://sourceforge.net/projects/fidocadj/files/FidoCadJ.dmg/download

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

At the moment, there are no extensions dedicated to this platform.

A.3 HOW TO DOWNLOAD AND EXECUTE FIDOCADJ IN A LINUX SYSTEM

by Roby IZ1CYN

---

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

Prerequisite: JRE 6 from Sun and/or OpenJDK 6 JRE (o previous versions compatible with the program's specifications) must be installed. In paragraph A.3.1 the installation of the program using only commands from a terminal will be described. In paragraph A.3.2, the interaction with a graphical environment will be used instead.

A.3.1 *Using any platform, from terminal*

Download the program using the `wget` command:

```
$ wget http :// davbucci . chez - alice . fr / elettronica /
   fidocadj / fidocadj . jar
--23:51:22 - -   http :// davbucci . chez - alice . fr / elettronica
   / fidocadj / fidocadj . jar
           => 'fidocadj . jar '
Resolution of davbucci . chez - alice . fr is being done ...
   212.27.63.127
Connection to davbucci . chez - alice . fr
   |212.27.63.127:80... connected .
HTTP request sent , waiting for answer ... 200 OK
Length: 276 ,694 (270K) [application/x-java-archive]

100%[=================================== >] 276 ,694
        71.75K/s    ETA 00:00

23:51:26 (71.59 KB/s) - "fidocadj . jar" saved        [
   276694/276694]
```

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

```
# mkdir /usr/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 >/fidocadj . jar /usr/bin/fidocadj
```

Let's make the file an executable

```
# chmod +x /usr/bin/fidocadj/fidocadj . jar
```

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

```
# exit
```

And now, we can execute the program:

```
$ /usr/bin/fidocadj/fidocadj . jar
```

A.3.2 *On a graphical system*

In the example, it will be an Ubuntu 8.04, but things does not change so much for older or newer versions:

- Let's download the file from the browser, or with Gwget or similar tools.

- We can then launch our File Manager (Nautilus, Konqueror...) as a root (if we do not have a specific command in the menu, we
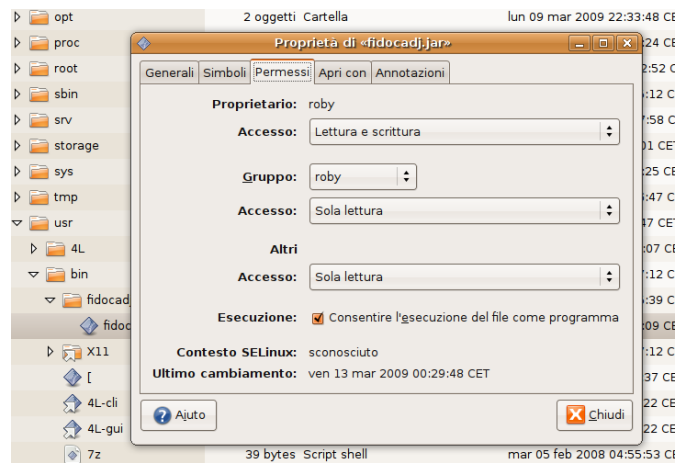
Figure 23: The setting of file rights, under Ubuntu 8.04.
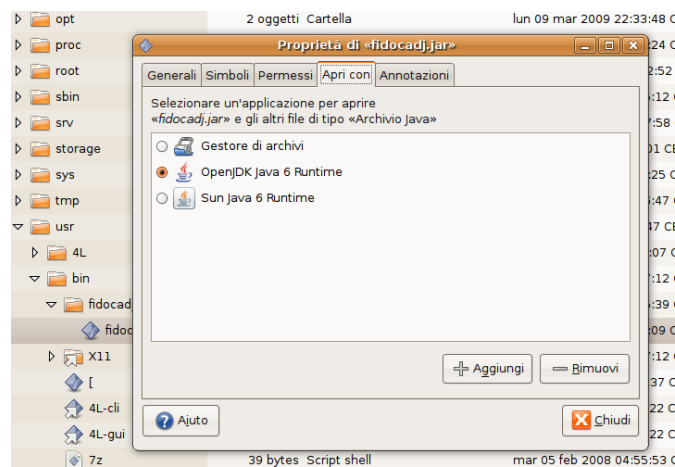
Figure 24: Set the execution with the Java virtual machine, under Ubuntu 8.04.

just need to launch it from the console by using `sudo nautilus`). We can create the directory `/usr/bin/fidocadj` and move there the file we just downloaded (it will be somewhere in `/home/<user>/`)

- Let's do 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 23

- We can select the tab "Open as" and select "OpenJDK Java 6 Runtime" or "Sun Java 6 Runtime", as it can be seen in figure 24.

- Let's 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/bin/fidocadj/fidocadj.jar`.

A.4 WINDOWS

Since version 0.23, when FidoCadJ recognizes a Windows platform, it will try to use the native Look and Feel.