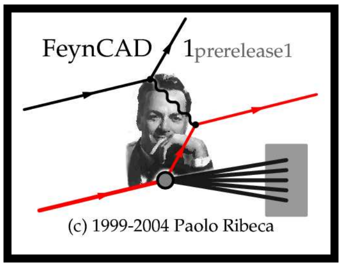


FeynCAD User's Manual

`Version 1 prerelease 1`

August 23, 2004

PAOLO RIBECA
E-MAIL: software@paoloribeca.net

This manual can be found at:
http://www.paoloribeca.net/software/FeynCAD/FeynCAD.ps.gz

All the pictures in this manual which are not screenshots of FeynCAD have been typeset using FeynCAD itself

Table of contents

Chapter 1
Basic informations about FeynCAD

Did it ever happen to you to need a publication-quality drawing of a Feynman diagram for an article, without finding any easy-to-use tool to produce it? In this case, FeynCAD is likely to be what you were looking for.
There are a number of good reasons for using FeynCAD to typeset your Feynman diagrams:

- FeynCAD outputs high-quality diagrams: drawings are first translated to a series of MetaPost macros, which are in turn used to produce a freely scalable Encapsulated PostScript file
- FeynCAD is completely user-friendly, with an intuitive and easy-to-use visual interface
- FeynCAD is WYSIWYG (What-You-See-Is-What-You-Get): the result of what the user draws is immediately displayed in its final form by the visual interface
- FeynCAD is well documented: it comes with an extensive manual, an on-line help system, and a set of on-line examples which the user can modify at will
- FeynCAD is easily customizable: new vertex/line styles can be easily defined by the user, and transparently integrated into the program
- FeynCAD can be easily interfaced: it inputs/outputs files in a clearly-documented XML format and can be invoked in batch mode without visual interface, making easy its use as a standalone rendering engine for external tools which automatically generate/compute Feynman diagrams.

Unfortunately, so far FeynCAD does not help you in deciding which layout is best for your diagram. However, stay tuned: algorithms to make the job are currently being incorporated into FeynCAD architecture, and their introduction will be finished before the final release of Version 1 is shipped.

[Of course, FeynCAD is not a general-purpose drawing system; however, we mention as a last remark that it can also be successfully "misused" to draw other pictures than just Feynman diagrams, at least to some extent: typical applications can be simple geometrical plots and flow charts. Nice results have been reported in various cases.]

1.1 General informations

1.1.1 Contacts

New releases and bugfixes of FeynCAD can be found at

http://www.paoloribeca.net/software/FeynCAD.

Please send bugreports, bugfixes and comments to

software@paoloribeca.net.

1.1.2 License

FeynCAD is free software. It is distributed, and can be used, under the terms of the GNU General Public License Version 2 as stated below.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

PREAMBLE

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software — to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

> **a)** You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
>
> **b)** You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
>
> **c)** If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

> **a)** Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **b)** Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
>
> **c)** Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. Because the Program is licensed free of charge, there is no warranty for the Program, to the extent permitted by applicable law. Except when otherwise stated in writing the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the Program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the Program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the program to operate with any other programs), even if such holder or other party has been advised of the possibilities of such damages.

——————END OF TERMS AND CONDITIONS——————

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
```

Also add information on how to contact you by electronic and paper mail.
If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items – whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

1.1.3 Credits

FeynCAD code is heavily based on some libraries/packages/components in the public domain, as from the following list:

- the graphical user interface (GUI) engine is provided by Qt-libraries, X11 edition (see `http://www.trolltech.com`)
- the actual drawing of Feynman diagrams is based on MetaPost macros derived from the ones which are present in `feynMF` package, by Thorsten Ohl (see `http://theorie.physik.uni-wuerzburg.de/~ohl` and links therein)
- the rendering of PostScript files is borrowed from an early 3.xx version of KGhostscript by the KDE project (see `http://www.kde.org`)
- icons are taken from `Kids` theme, by Everaldo Coelho (see `http://www.everaldo.com`)
- this manual was typeset with GNU TeXmacs (see `http://www.texmacs.org`), which is in my opinion a real masterpiece on its own. It revolutioned my personal way of typesetting documents, and I am pretty sure that the same will happen to everyone who is introduced to it; thus, just give it a try ;)

- Richard Feynman's photograph appearing in FeynCAD logo is the one which accompanies his official portrait for the Nobel Foundation (see http://www.nobel.se), by which he, Sin-Itiro Tomonaga and Julian Schwinger were awarded in 1965 the Nobel prize in Physics, due to "*their fundamental work in quantum electrodynamics, with deep-ploughing consequences for the physics of elementary particles*".

The entire development phase took place on a Linux/GNU system, using GNU `g++` compiler. The Gimp has been used to acquire all the screenshots of the application, and to post-process some of the images.
In addition, the author would like to personally thank

- Urko Reinosa, for some early beta-testing and some good suggestions during the initial stages of FeynCAD development
- Tomasz Korzec, for constant and careful beta-testing during the late stages of FeynCAD development
- his wonderful wife Giulia, who kindly accepted to see a lot of hours of free time turned into hours of software programming. Without her patience and understanding, FeynCAD would not exist at all.

1.1.4 How to contribute to FeynCAD project

There are essentially four ways you can help FeynCAD project:

1. Find out bugs, and report them to the e-mail address software@paoloribeca.net.
2. Produce nice diagrams, and donate the most beautiful of them to the project. Eventually, they will be included as screenshots in the web page very soon.
3. Report to the author your feelings about the program. While good ideas and suggestions can significantly improve FeynCAD, even simple comments will be very appreciated: users' feedback is always important!
4. Following common rules of fairplay, ***cite in your articles this manual, if you used FeynCAD to produce diagrams for publication***. This will hopefully stop some of the people all around, who keep asking me "But why did you spend so much of your time in doing that?".

1.2 Installation

1.2.1 Required external software

To successfully obtain a compiled graphical output from FeynCAD in EPS format, you need the following free software installed on your computer:

1. A modern and complete installation of TEX/LATEX. In particular, the following programs/packages must be present and accessible:
   - MetaPost
   - TEX/LATEX
   - `dvips`
   - All required fonts and macro packages. In particular, Type 1 fonts for TEX (and of course a working support for them) can be profitable for the quality of produced images.

   Please notice that every modern installation of TEX —like for example `teTeX`— comes with all these packages preinstalled (you may wish to have a look at http://www.ctan.org for the most updated news on the subject)

2. A working installation of Ghostscript (see `http://www.cs.wisc.edu/~ghost` as a starting point about it, and for example `http://www.ctan.org/support/ghostscript/aladdin` or `http://sourceforge.net/projects/espgs` for two different free versions of the program). The more PostScript fonts you have, the better for you, since you can use all these fonts from within FeynCAD.

Without any of the required packages, the process of producing graphs will plainly fail. As its default behaviour, FeynCAD assumes that all these executables are accessible from your path; if something goes wrong you can modify this behaviour from the configuration window (`Edit` → `Global Settings`), by specifying explicitly the complete path of all programs which cannot be automatically found by FeynCAD.

1.2.2 Required additional files

In addition to the aforementioned files, FeynCAD needs just one single file to properly display its output. This file, called `FeynCAD.mp`, is provided together with the sources or the binaries of FeynCAD; it contains a modified version of the MetaPost macros for the `feynMF`/`feynmp` package by Thorsten Ohl (see {1.1.3} for credits/copyrights).

1.2.2.1 MetaPost macros in `feynmp` and FeynCAD. Compatibility to `feynmp`

While at the beginning of its history FeynCAD was merely a frontend to `feynmp`, things have changed quite a lot, and in the meantime almost all original `feynmp` macros have been rewritten from scratch — or implemented in a different way. Here is a list of major changes (you may wish to consult reference [2] of Chapter 2 for a cross-check):

Vertices. They have undergone some major changes. In particular:

- Filling algorithm has been reimplemented, to avoid some of its known problems.
- The concept has been introduced of *parametrized styles* — that is, of styles carrying information about themselves in their names like `polygon10`, which implicitly prescribes the number of sides of the decoration. This eliminates the necessity of providing internal aliases for style definitions in the code (like "`pentagon`" for "`polygon 5`"); thus, user can now access all `feynmp` vertex styles of this kind by just asking for `polygon`*nn*, `polygram`*nn* and `polycross`*nn*, where *nn* is a suitable integer number less than, or equal to, `99` — even if no aliases have been explicitly defined in the code, for example, for polygons with *nn* more than 6.

  In addition, four new parametrized styles have been added:

  1. so called `polystars` (which can be regarded as intermediate objects between `polygons` and `polygrams`)
  2. rectangles, rounded rectangles and ellipses. For all these styles, two parameters are required in the name, to specify the proportion between their two sides/axes: in fact, syntaxes are respectively
     `rectangle`*www*`:`*hhh*
     `rrectangle`*www*`:`*hhh*
     `ellipse`*www*`:`*hhh*,
     where *www* and *hhh* are suitable integer numbers less than, or equal to, `999` (remember MetaFONT/MetaPost hard limits about numbers, see {2.2}). For example, `ellipse3:2` style will produce an ellipse whose X axis is 1.5 times larger than the Y axis.

Lines. Here again the list of changes is rather long.

- A few new styles have been introduced; they are the ones called `beam` and `beam_arrow`
- Almost all styles have been heavily modified. In particular:
  1. `curly` and `zigzag`, which used to start and end abruptly, were redesigned to have instead a straight "transition zone" both at their beginning and end; this feature allows lines of these styles to be plugged more nicely into their vertices, and makes `curly` tadpoles look much less awkward

  2. in a similar way, a white transition zone was added at the beginning and end of almost all dotted/dashed styles (`dots`, `dots_arrow`, `dbl_dashes`, `dbl_dashes_arrow`, `dbl_dots`, `dbl_dots_arrow`, `beam`, `beam_arrow`)

- Arrows have been rewritten from scratch. They should look nicer, being now a little carved on the flat side, and able to bend when necessary; in addition, with the new algorithm arrows can in principle automatically adapt their size to that of their carrying line
- Rubout parameter is now fully functional for all styles.

Tags. All `feynMF`/`feynmp` parameters for vertices/lines which were related in a way or the other to labeling have been removed. From FeynCAD's point of view, in fact, labels are standalone objects which can be freely placed anywhere; thus, they are rendered taking advantage directly of MetaPost text-drawing primitives.

However, the basic structure of graphical engine remains the same (in particular with respect to the rendering of vertices and lines, and to the structure/handling of their parameters). This implies that the overall syntax of FeynCAD objects is still very reminiscent of `feynmp` style, and that an experienced user of `feynMF` should immediately be able to recognize/use most of the parameters of FeynCAD objects.

1.2.3 Installing FeynCAD on your system

There are essentially two ways of getting FeynCAD installed on your Unix system:

1. Downloading and installing a precompiled binary or a precompiled package, if any exists which suits your architecture/distribution
2. Compiling FeynCAD on your own.

Even if (at least in principle) compiling FeynCAD should be a fairly simple operation if you have Qt-libraries properly setup on your machine, you should anyway leave it as a last option if precompiled binaries are available for your system. *Please notice that the author cannot provide any support for the operations of compilation/installation of FeynCAD on your system*; thus, if you experience problems please report them directly to your nearest Qt-guru.

1.2.3.1 Precompiled binaries

As at the time of writing, precompiled binary packages are available only for Linux/GNU operating system. Here is the list of supported hardware platforms:

- Intel 32-bit `ix86` architecture, and compatible processors (Pentium I/MMX/Pro/II/III/IV/Mobile/M, Celeron, K6, K6-2/-3, Athlon, Athlon Thunderbird/-4/-XP/-MP, etc.)
- AMD 64-bit `x86_64` architecture (Athlon 64 and Opteron)

but before giving up you should of course check the more up-to-date list at FeynCAD site, `http://www.paoloribeca.net/software/FeynCAD`.

If one of these packages suits your need, just download it and decompress it in your favourite installation directory. Each precompiled archive contains a few files:

i. The FeynCAD executable (unsurprisingly called `FeynCAD`)

ii. The `feynmp`/FeynCAD MetaPost macro file `FeynCAD.mp`

iii. File `COPYING`, which contains the GNU General Public License as reported in {1.1.2}

iv. File `README`, which points out that installation instructions can be found in this manual

v. A PostScript version of present manual (or possibly an updated version of it), file `FeynCAD.ps.gz`.

Once you have carefully read the license, just do the two following things:

1. Move `FeynCAD` program to a location which is in your `PATH` environment variable, or equivalently add to `PATH` the directory where `FeynCAD` is

2. Make file `FeynCAD.mp` visible from FeynCAD. To this end, you have at your disposal two different possibilities:

   a. Install it as a standard package in your MetaPost tree, that is:

      - Copy it to the right location, on typical Linux-`teTeX` system something like

        `/usr/share/texmf/metapost/misc/FeynCAD.mp`

      - Do not forget to rehash the TeX file database to make the installed file visible to other programs; on most system this can be achieved with the command `texhash`.

      Of course, this solution requires from you root privileges.

   b. Put the file wherever you like best, but inform FeynCAD of the new location from within the configuration window (by filling in the field `Edit` → `Global Settings` → `Shipped macros` the first time you execute the program).

   As its default behaviour, FeynCAD assumes that `FeynCAD.mp` is visible from the current directory — i.e. that it has been installed in your MetaPost tree, or that it is present in the directory from which FeynCAD itself has been started (started, *not* installed!); this means that step {b.} cannot in practice be avoided if you decide to keep file `FeynCAD.mp` in a non-standard location (and FeynCAD will just not work properly until it has been done).

Now, at least in principle, you are ready to enjoy your brand-new copy of FeynCAD.

1.2.3.2 Compiling FeynCAD

To successfully compile FeynCAD starting from its sources you need to have a recent version of Qt-libraries, X11 edition installed on your system (currently the bound is that version must be $\geqslant$ `3.3.2`, and <4). These are the required steps:

1. Go to the directory where you actually intend to perform the compilation, and unpack the source archive which you have downloaded from FeynCAD website by means of one of the following commandlines (depending on whether you chose to download the package compressed with `gzip` or the package compressed with `bzip2`):

   ```
   > tar xzvf FeynCAD-1pre1.tgz
   > tar xjvf FeynCAD-1pre1.tbz2
   ```

   The exact form of the needed tar command can be slightly different on your system; if you experience problems you may wish to try with the more standard syntaxes

   ```
   > gunzip FeynCAD-1pre1.tgz | tar xvf -
   > bunzip2 FeynCAD-1pre1.tbz2 | tar xvf -
   ```

   In the end, this operation will produce a directory called `FeynCAD-1pre1`, containing all source files; `cd` into it.

2. Make sure that your environment contains a variable `QTDIR` properly set; following the specifications given by Qt-libraries, this variable must hold the path of the top directory where Qt's are installed on your system. For example, supposing that Qt's can be found in `/usr/local/qt-x11-free-3.3.2`, you will have to type the command

   ```
   > export QTDIR="/usr/local/qt-x11-free-3.3.2"
   ```

   if you are using `bash`, or

   ```
   > setenv QTDIR "/usr/local/qt-x11-free-3.3.2"
   ```

   if you use `tcsh`-compatible shells.

3. Type

```
> qmake FeynCAD.pro
```
.

If Qt's are properly installed and setup on your system, this operation will automatically generate a `Makefile` containing all the instructions which are needed to successfully compile FeynCAD on your system.

4. Type

```
> make
```
;

this should be enough to generate in your current directory the executable, which will be called `FeynCAD`.

In principle, after this step you can complete the installation by following the indications given in {1.2.3.1} about the placement of files `FeynCAD` and `FeynCAD.mp`; in practice, however, you can very well experiment instead a failure of `make` command, due possibly to small bugs in your installation of Qt-libraries, or in Qt-libraries themselves. Two common such examples are

1. a failure in the compilation stage, due to the fact that the `moc` program shipped with some versions of the Qt-libraries prior to `3.3.2` produces some incorrect `C++` code while trying to include application pixmaps
2. a failure in the linking stage on systems where Qt's have been compiled with multithreading support enabled, due to the fact that Qt-produced `Makefile` wrongly attempts to link object files against unexisting single-threaded library (`-lqt` flag) — while the correct way would rather be to link them against the multithreaded version which is effectively installed, flag `-lqt-mt`.

In conclusion, before being able to complete the compilation you may need to make use of some additional workarounds; to find them, please refer to your nearest Qt-guru previously mentioned in section {1.2.3}.

1.3 Commandline syntax

There are two different modes for FeynCAD to work:

1. The **GUI** (Graphical-User-Interface) **mode**. It is the usual mode, where the user can draw graphs by means of mouse, windows, etc. It can be started with the syntax

   ```
   FeynCAD [inputFile1 [inputFile2 ... ]]
   ```

   All the specified optional files `inputFile1`, `inputFile2`, ... (which should be valid FeynCAD-XML files, see {Appendix 2}) are opened at startup, each one in a separate tab of the editing window.
2. The **batch mode**. In this mode FeynCAD does not open a graphical interface, but just acts as a filter, compiling one single FeynCAD-XML file into its final EPS (Encapsulated PostScript) form. Here is how to invoke FeynCAD in batch mode:

   ```
   FeynCAD -b inputFile outputFile
   ```

   The graph file `inputFile` must be a valid FeynCAD-XML file here. The suffix `.eps` is appended to `outputFile` if not already present.

1.3.1 Other commandline options

`-c configFile` `--configFile configFile`	Allows the user to specify a non-standard configuration file `configFile` at startup (the default is `~/.FeynCADrc`, where "~" means "user's home directory"). The configuration file must be a valid FeynCADrc-XML file (see {2.7.4} and {Appendix 3})
`-h` `--help`	Shows the correct syntax of the commandline options and exits without performing any operation.

Chapter 2

Editing graphs with FeynCAD

In FeynCAD, graphs live on a discrete two-dimensional grid.
There are three different kinds of graphic objects which can be used to build up a graph:

1. The **vertices** of the graph. They can have different shapes, sizes, colors, filling, etc. *Vertices must be unique*, i.e. two different vertices cannot share the same position.

2. The **lines** of the graph. They *carry an orientation*, and can have different styles, sizes, starting/ending angles, colors, etc. *Lines cannot exist in isolation*, i.e. they are always bound to two existing vertices (or the same vertex in case of **tadpoles**). As dictated by the underlying MetaPost engine, lines are rendered as splines — that is, as smooth curves whose shape is determined by a few control parameters; in particular, MetaPost/MetaFONT splines are completely specified by their starting and ending angles — and by an additional parameter, which in FeynCAD will be called the *tension* of the line. Angles are measured as usual starting from the positive x axis of the Cartesian plane, and are expressed in degrees. For more particulars about MetaPost drawing system, please refer to [1].

3. The **tags** (or **labels**) of the graph. They are arbitrary strings of text, with their own fonts, sizes, control points, angles, colors, etc. Tags can be mathematical formulas typeset in TEX (but *not* in LATEX!). *Tags must be unique*, i.e. two different tags cannot share the same position.

Some words more should be spent about the produced MetaPost output.

All the informations about the graph which are entered by the user through the graphical interface are converted to a MetaPost file; this means that *the content of every field where the user can freely enter text* (they have an orange background color in the graphical user interface to emphasize their importance) *must be valid MetaPost code*, otherwise the graph will not be successfully converted to its final form.

In the following, we will specify in much more detail what is valid and what is not (see {2.2}); for example, this observation applies to all fields which specify line styles —which must be the names of valid `feynmp`[2]/FeynCAD or user-defined macros— or colors —which must be entered as valid MetaPost color specifications—, but does not usually apply to the case when integer or floating point numbers are required (in this case, in fact, FeynCAD will automatically take care about the correctness of your input in the majority of cases).

If FeynCAD complains that it could not compile your graph, please be patient and just correct your input; in this situation you will not be able to see the graphical result of your work until what you entered is acceptable again, but you can still modify/save your graph. If the error persists, you can turn on the option `Edit` → `Global Settings` → `Paths & Commands` → `Keep generated MetaPost & LaTeX files` and try to run MetaPost directly on the generated file, to identify where your wrong code is located. In case of doubt, refer to [1] and [2].

[1] John D. Hobby *A User's Manual for MetaPost* (http://www.paoloribeca.net/software/FeynCAD/mpost.ps.gz)

[2] Thorsten Ohl *feynMF: drawing Feynman Diagrams with LATEX and METAFONT* (http://www.paoloribeca.net/software/FeynCAD/feynmf.ps.gz).

2.1 Drawing graphs

As it is common practice in fairly all GUI-based programs, new (empty) drawing widgets can be opened through menu `File`→`New`, or existing drawings (corresponding to existing FeynCAD files) can be retrieved by means of `File`→`Open`.

[By the way, since the purpose of almost all items in menu `File` should turn out to be self-explanatory as well —excluding perhaps `File` → `Modify signature`, which is dealt with in {2.4.2}— we will not talk about this menu any more; we just mention that any graph can be exported at any time to its Encapsulated-PostScript form by selecting `File`→`Export as EPS`.]

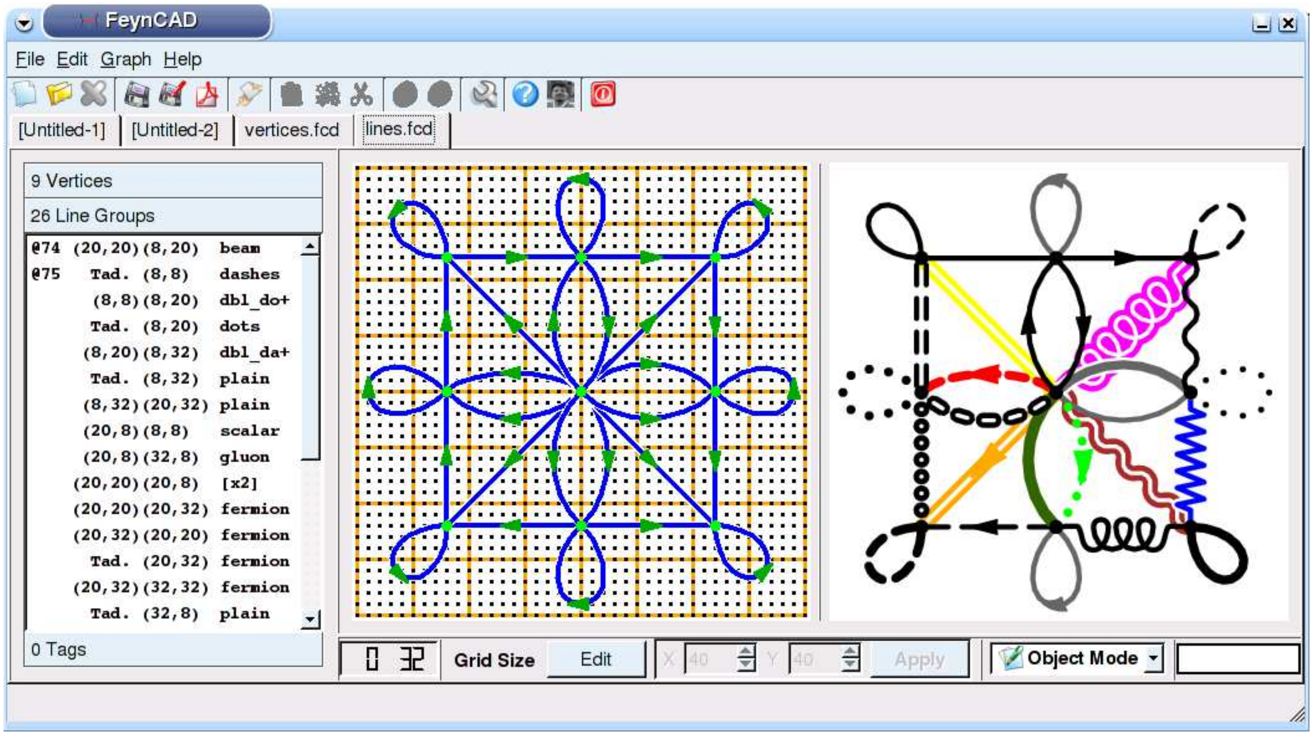


Figure 2.1. An example of FeynCAD session. Four different graph editors are open in the main window, and the one on top is shown.

Once you have created a graph in one way or the other, you can modify it at will. To this end, each tab of the graph editor is split up in two parts:

1. The left part, where the user can place/move/edit/cancel graphical objects by making use of the mouse, and/or of keyboard accelerators (see resp. {2.1.1}, {2.1.2} and {2.6}); from now on, we will refer to it as to **the drawing board**. The drawing board is equipped with a cartesian **grid**

2. The right part, where the result of the operations made in the CAD (Computer Aided Design) part is made visible in a WYSIWYG (What-You-See-Is-What-You-Get) fashion; we will call it **the viewer** in the following.

Some other GUI-facilities are present in FeynCAD graph editors; of particular importance are, describing them in order as we can see them from the left to the right:

i. A LED-styled pair of numbers, which —continuously tracking all its movements on the drawing board— show in real time the updated coordinates of the mouse w.r.t. the grid

ii. The `Grid Size` → `Edit` button, which allows the user to change the values of the upper limits of X and Y coordinates for current graph (see {2.4.1})

iii. The `Object Mode/Select Mode` **selector**, which can be used to toggle the operating mode of graph editor (see below)

iv. A rectangular strip, called the **viewer progressbar**, showing the current status of the viewer (see below). It is usually white when no editing is being performed by the user

v. Finally, placed at the full bottom of graph editor, a **status bar**, where text messages appear from time to time to notify the user about what is going on, or to possibly give him/her some suggestions about how to use the GUI.

Please keep in mind some basic facts while drawing:

- Since producing graphs is a complicated and resource-demanding operation (it involves among all other phases the creation of a MetaPost document, which is in turn converted to PostScript and then incorporated in a LaTeX document; this one is in the end compiled into the final Encapsulated-PostScript form), not every editing operation on the drawing board triggers a corresponding update of the viewer; for example, when you select objects and move them the viewer is refreshed only when you merge again your selection to the other objects on the drawing board —that is, when you unselect—, and this only if the selection has not been placed again in the same position it holded before.

- The production chain for rendering graphs in EPS form can take a (relatively) long time to end, in particular on slow machines; this condition is indicated with a flashing green color displayed by viewer progressbar. Please be patient and wait until the process terminates naturally, when the indicator becomes white again: repeated triggers to the update process can create instabilities in the synchronization of graph editor — depending on whether the underlying system is busy or not, on the load of the processor and on other unpredictable facts. Here is the list of all possible states of the progressbar, as indicated by its color:

  Ok. The viewer is idle (*white color*)

  Busy. The viewer is updating the graph, repainting on the screen its refreshed form (*flashing green color*)

  Timeout. The viewer has tried to update the graph, but the operation did not end before the rendering timeout set by the user —see parameter `Edit` → `Global settings` → `Paths & Commands` → `Rendering timeout` and its description in {2.7.2}— was reached; thus, refreshing process has been terminated (*orange color*). A new update operation will be tried anyway the next time it is required

  Error. A serious error has occurred during the steps performed by the viewer to update the graph (*red color*); in this case, the anomaly is in general pointed out by additional error messages. Such a condition can occasionally show up in connection to weird problems affecting your system (for example, too many processes are running on it, and FeynCAD is unable to span a new Ghostscript process needed by the viewer).

- FeynCAD is a WYSIWYG editor in the truest sense of the name: what you see in the viewer is exactly the same file that you will get while exporting you graph as an EPS file: thus, no surprise at all can show up in your final result.

There are two different modes to enter objects into the drawing board, or to manipulate them once the have already been entered:

1. The **Object mode** (see {2.1.1})

2. The **Select mode** (see {2.1.2}).

In each editing window they can be activated from the GUI by making use of the `Object Mode/Select Mode` **selector**, or with the appropriate keyboard accelerator (see {2.6}).

In each of the two modes mouse buttons and clicks have different functions. Here follows a detailed description of what exactly can be done with the mouse in both modes.

2.1.1 Object mode

MOUSE ACTION	LEFT BUTTON	MIDDLE BUTTON	RIGHT BUTTON
Clicking	*If there is no vertex at the clicked point of the grid:* **Place a new vertex** *Otherwise:* choose one of **Open vertex editor** and **Delete vertex**.		*If there is no tag at the clicked point of the grid:* **Place a new tag** *Otherwise:* choose one of **Open tag editor** and **Delete tag**.
Dragging		*If there is no line between the starting and the ending points of the dragging movement*: **Place a new line** *Otherwise*: choose one of **Add line**, **Open line & tadpole editor** and **Delete lines**.	

As a convention, each new object added to the grid is a copy of the last object of the same kind (vertex, line, tadpole or tag) *created or modified while editing the graph.*

In addition, in this mode the user can edit the properties of vertices/groups of lines/tags by just double-clicking on any item of the object lists which can be found at the left-hand side of each graph editor; this action will result in FeynCAD automatically opening the appropriate editor for the object which has been selected.

2.1.2 Select mode

Following the usual conventions, FeynCAD has a **clipboard** (global to the entire program), to which the last selection is copied, no matter which one is the graph currently being edited; this structure allows the user to freely copy groups of objects from one drawing board to any other.

<table>
<tr><th>MOUSE ACTION</th><th>LEFT BUTTON</th><th>MIDDLE BUTTON</th><th>RIGHT BUTTON</th></tr>
<tr><td>**Clicking**</td><td>*If a selection is already present*: **Unselect & merge selection to current graph**.</td><td rowspan="2">*If the clipboard is not empty*: **Paste clipboard to current graph**. The copy of the content of the clipboard can then be freely moved around the graph, and merged to it by clicking with any of the buttons of the mouse.</td><td></td></tr>
<tr><td>**Dragging**</td><td>*If a selection is already present and the mouse is in the middle of it*: **Move selection** *Otherwise*: **Make a new selection**. The selection is then transmitted to the clipboard.</td><td></td></tr>
</table>

Please notice that other operations can be performed on the selection; however, some of them cannot be accessed just by using the mouse, and require instead the use of keyboard or of application menus. The complete list of operations on selections can be found in {2.3}.

2.2 Editing FeynCAD objects

Before examining in detail all properties of vertices, lines and tags, the following observations about some general characteristics of FeynCAD objects can be useful:

- Every dimension in FeynCAD is described in terms of two main sizes:
  1. **The grid step**, whose size is by convention fixed to one millimeter. It is used to describe distances on the grid, such as coordinates and the extension of tadpoles
  2. **The basic width of the line**, expressed in terms of the `thin` unit (see {2.7.3} for more information on this parameter, and open `Edit` → `Global Settings` → `MetaPost` → `thin` if you want to modify it). All concepts which can be compared to line width —such as sizes of vertex decorations and vertex borders— are described as multiples of this unit.

  Please notice that the choice of millimeter as the grid unit is merely a convention; in fact, since EPS files can be rescaled at will what really matters to the nicety of produced graphs is actually the *ratio* between the length units for grid and line, and not any of the absolute magnitudes of the two. There are many other parameters in `feynmp`/FeynCAD MetaPost macros, and their values too have been patiently tuned to quantities which usually produce results pleasant to the eye; however, should you not be satisfied by the defaults you can easily modify all the scales in the game through menu `Edit` → `Global Settings` → `MetaPost` (see {2.7.3}). *In this case, be aware that the new settings will of course also affect the rendering of your old graphs, which you may have stored as files.*

- Since PostScript objects are not transparent, PostScript interpreters/printers need some additional information when they draw the graph as a stack of layered objects; thus, every vertex/line/tag in FeynCAD has a peculiar parameter called **depth**, specifying in which temporal order each object is rendered. The greater in magnitude the depth, the sooner the object will be drawn; in other words, the deeper the object, the less it will be visible. Depth is expressed as a number belonging to a conventional range, which goes from `1` to `99` (inclusive). If the same depth is shared by a set of more than one object, and the objects in the set overlap, then they will be stacked by FeynCAD in an unpredictable order.
  The default values for the depths of objects are as follows:
  i. Tags are placed at depth `25`
  ii. Vertices are placed at depth `50`
  iii. Lines are placed at depth `75`.

  The rationale behind these assignments is that, as a general rule, in usual plots we would like to have vertices stacked over lines (in fact, we do not want to see line terminals), and tags stacked over everything else (we want labels to be entirely visible even when they are lying over other objects). *However, since there can be a lot of exceptions to this rule, the user is completely free to change the value of this property to whatever value he or she wants.* Please notice how this choice of default values divides the range of allowed values for depth parameter in four intervals of exactly the same span.

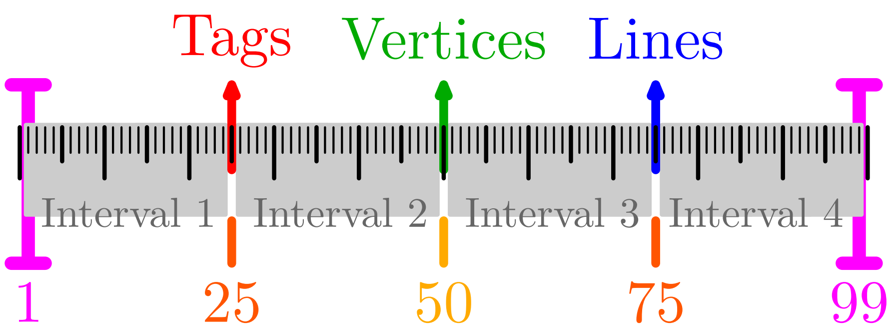


Figure 2.2. About the setup of depth parameter.

- In general, the introduction of wrong parameters (that is, numerical values and strings) in at least some of the fields is not harmless, and can result whether in an incomplete rendering of the graph —if the error is silently trapped by `feynmp`/FeynCAD macros—, or in a failure of the generated MetaPost code; in the latter case the viewer will be unable to refresh itself, and will prompt the user with an error message.

  As a general typographical convention, in the following all fields/parameters/concepts which are related to possibly invalid MetaPost code will be highlighted in orange. However, *please keep in mind that every numerical field can be dangerous to MetaPost, even if this fact is not esplicitly indicated*. In fact, MetaFONT, and MetaPost too for legacy reasons, are based on a fixed-point arithmetics whose maximum legal value is `4095.99998`; thus, every computation producing a quantity or a measure which is bigger than that will make MetaPost stop.

  When the user types in numbers he/she has to be conscious that:

  1. All the fields in the GUI requiring the insertion of numerical values are *validated* — that is, *numbers introduced by the user in these fields are rejected/modified if they do not comply to a pre-specified format*; for example, only up to 4 precision digits are usually allowed

  2. After having being inserted, numbers are eventually checked against underflow/overflow, and ceiled to the maximum/floored to the minimum of the allowed range if necessary

  3. Reasonable defaults/ranges have been picked up for all quantities in the game;

  it must be said, however, that even if all these precautions have been taken a malicious user can still generate invalid MetaPost code in a lot of fairly easy ways.

2.2.1 Editing vertices

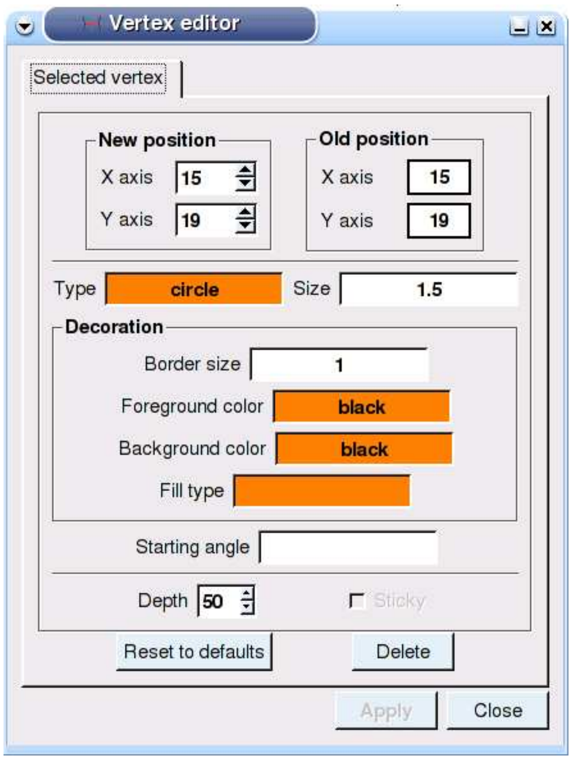


Figure 2.3. How a FeynCAD vertex editor looks like.

Here is the list of all parameters allowed for objects of type vertex:

X position — specifies the horizontal position of the vertex on the grid.
It is expressed in grid units — that is, in `mm`

Y position — specifies the vertical position of the vertex on the grid.
It is expressed in grid units — that is, in `mm`

Type — specifies the shape of the vertex.
It must be the name of an existing `feynmp`/FeynCAD vertex style (see table below for a list of possible values) or the name of an existing user-defined MetaPost macro.
Please notice that rendering `polygrams` and `polycrosses` of too high a degree can be fatal to some versions of Ghostscript PostScript interpreter. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Size — specifies the size of the vertex.
It is expressed in multiples of the basic line-width unit — `thin` MetaPost parameter, see `Edit` → `Global Settings` → `MetaPost` → `thin` and {2.7.3}

Border — specifies the size of vertex border.
It is expressed in multiples of the basic line-width unit — `thin` MetaPost parameter, see `Edit` → `Global Settings` → `MetaPost` → `thin` and {2.7.3}

Foreground color — specifies the foreground color used while drawing the vertex.
It must be a MetaPost color specified by valid MetaPost code. Examples: an RGB triple as `(.25,1/3,2/3)` or a linear combination of predefined colors `red`, `green` and `blue` like `0.7red+1/5blue`. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Background color — specifies the background color used while drawing the vertex.
It must be a MetaPost color specified by valid MetaPost code. Examples: an RGB triple as `(.25,1/3,2/3)` or a linear combination of predefined colors `red`, `green` and `blue` like `0.7red+1/5blue`. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Fill type — specifies how the interior of the vertex (that is, the part of the vertex which does not belong to its border) is filled.
It must be a valid `feynmp`/FeynCAD string (see table below for a list of possible values). Please notice that all styles defined by means of tiles (such as `gray10`, `gray25`, `gray50`, `gray75`, `gray90`, `shaded` and `hatched`) are deprecated already at `feynmp` level in favour of halftones — which are specified instead by a continuous number between `0` and `100`. [Actually, tile styles were born as `feynMF` simulations of native PostScript/MetaPost halftoning capabilities, which are not present in MetaFONT; they have not been suppressed in FeynCAD just for historical interest and compatibility reasons.] *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Starting angle — specifies the starting angle (in degrees) used while drawing vertex decorations. It is useful in the case when decorations are not symmetric with respect to their centers

Depth — specifies at which depth vertex is drawn: in PostScript objects are stacked, and upper layers hide lower layers

Sticky — specifies whether the position of the vertex is kept fixed or not while layout is automatically computed for the graph (*not yet implemented in this version!*).

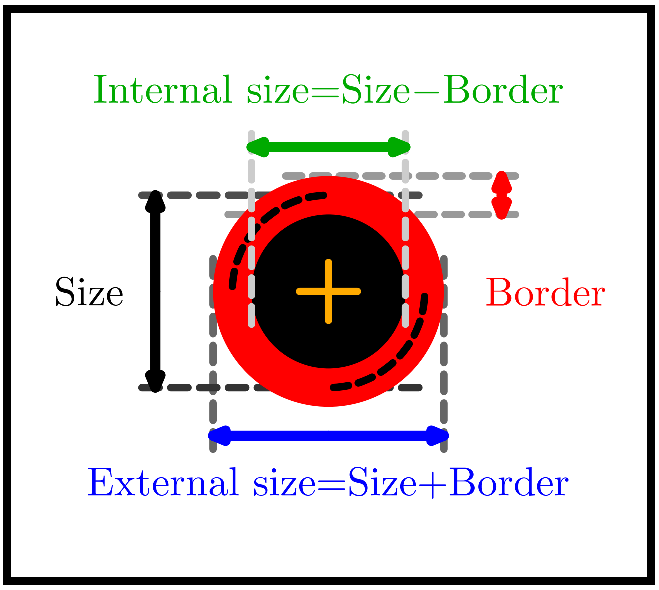


Figure 2.4. Meaning of vertex parameters **Size** and **Border**.

Here follows a table with the description of allowed values for each of the fields:

FIELD	TYPE	ALLOWED VALUES	DEFAULT
X position	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Y position	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Type	*string*	various { `circle`, `square`, `rectangle`*nn*`:`*nn*, `ellipse`*nn*`:`*nn* }, polygons { `triangle`, `diamond`, `tetragon`, `pentagon`, `hexagon`, `polygon`*nn* }, crosses { `cross`, `triacross`, `pentacross`, `hexacross`, `polycross`*nn* }, polygrams { `triagram`, `tetragram`, `pentagram`, `hexagram`, `polygram`*nn* }, stars: `polystar`*nn*, transparent (empty string) (For crosses, parameter **Border** is of no relevance.)	`circle`
Size	*float*	from `0` to `2000` (in `thin` units)	`1.5`
Border	*float*	from `0` to `2000` (in `thin` units)	`1`
Foreground color	*string*	a MetaPost color specified by valid MetaPost code: RGB triples { `(1,0,0)`, `(.25,1/3,2/3)`, ... }, names { `black, white`, `red, green, blue`, `0.4white`, `0.7red+1/5blue`, ... }	`black`
Background color	*string*	a MetaPost color specified by valid MetaPost code: RGB triples { `(1,0,0)`, `(.25,1/3,2/3)`, ... }, names { `black, white`, `red, green, blue`, `0.4white`, `0.7red+1/5blue`, ... }	`black`
Fill type	*string*	halftones { `0`, `1`, `..`, `99`, `100` }, tiles { `gray10`, `gray25`, `gray50`, `gray75`, `gray90` }, various { `empty`, `full`, `shaded`, `hatched` }	empty

Starting angle	*float*	from `-360` to `360` (in degrees)	`0`
Depth	*unsigned*	from `1` to `99`	`50`
Sticky	*bool*	`true` or `false`	`false`

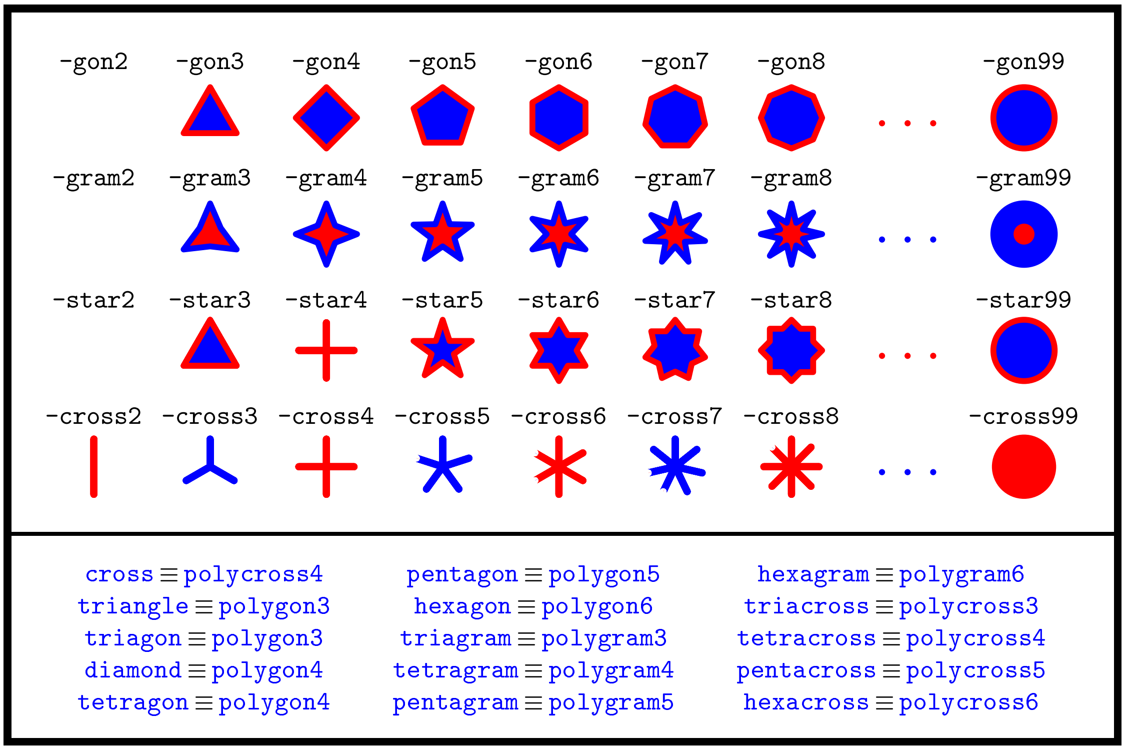


Figure 2.5. Synopsis of `poly-` **Types**. To compact the table the names of each style have been shortened: the complete name can be obtained by replacing the initial dash "-" with the prefix `poly` (for example: `-gon4` ⟶ `polygon4`). A list of existing aliases/shortcuts is shown in the lower part of the table. Please notice that rendering `polygram`s and `polycross`es of too high a degree can be fatal to some versions of Ghostscript PostScript interpreter.

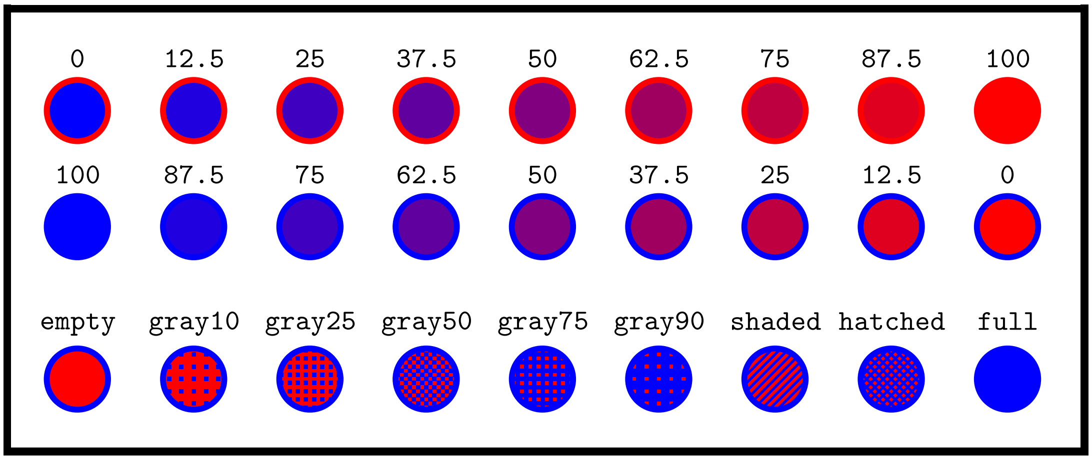


Figure 2.6. Some examples of what different **Fill** parameters result in. In all vertices of the first row **Foreground color**=`red` and **Background color**=`blue`, while in the last two rows **Foreground color**=`blue` and **Background color**=`red`. Please notice that the usage of all fills in the last row except `empty` and `full` is deprecated (see above).

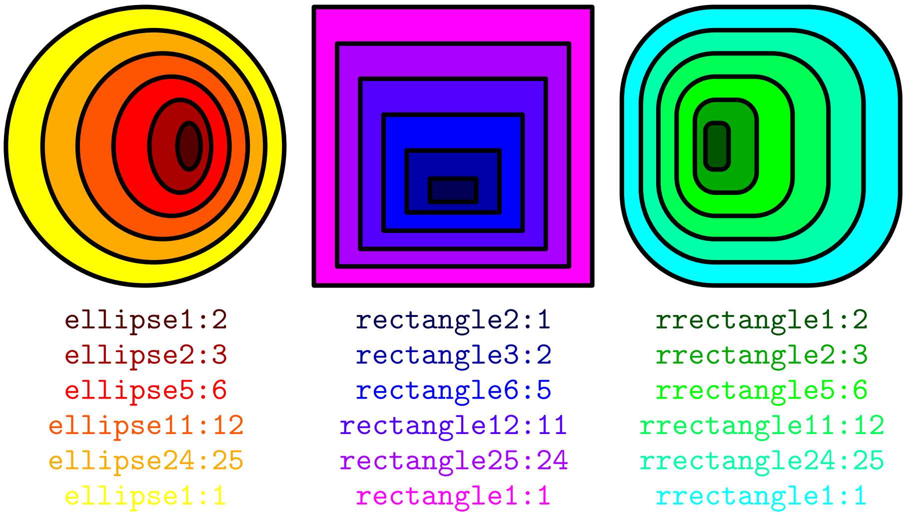


Figure 2.7. Three “Hanoi towers”, resp. of `ellipse`-s, `rectangle`-s and `rrectangle`-s.

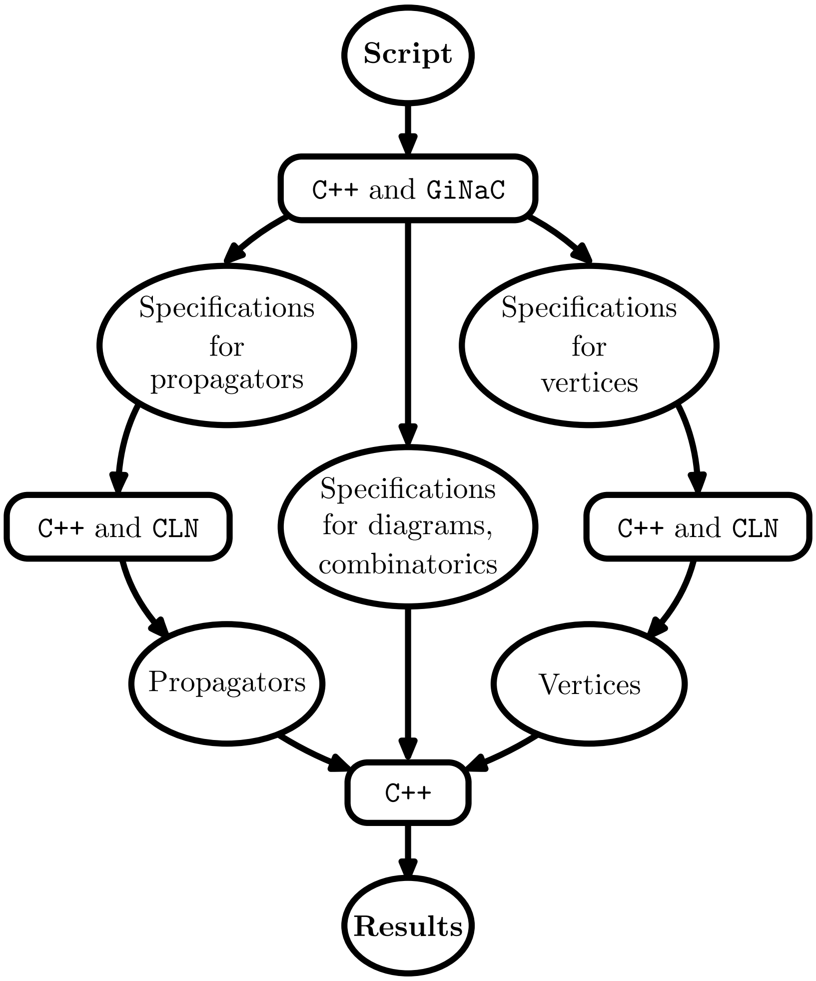


Figure 2.8. An “abuse” of `ellipses` and `rrectangles`.

2.2.2 Editing lines and tadpoles

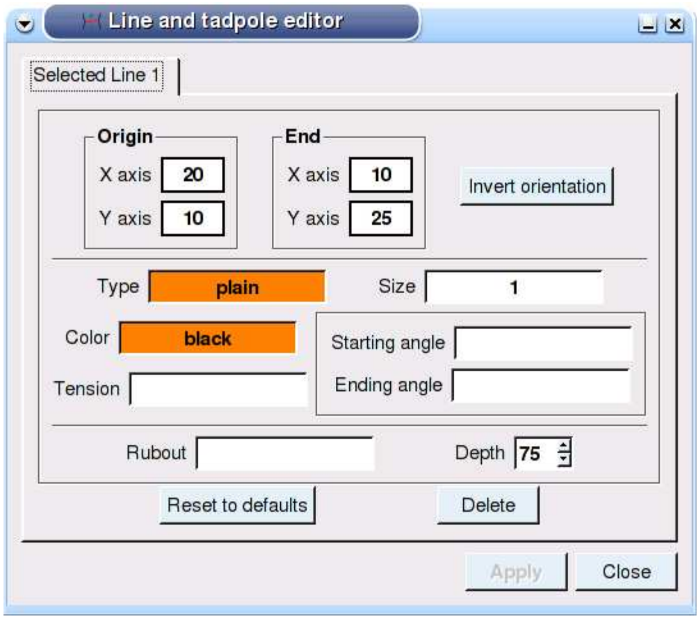


Figure 2.9. How a FeynCAD line editor looks like.

The properties which are used to parametrize lines and tadpoles can be divided in two different groups: some items which are common to lines and tadpoles, and some other items which have a different interpretation whether the object under examination is a tadpole or a line.

2.2.2.1 Parameters common to lines and tadpoles

Here follows a list of all parameters which have the same meaning both for lines and tadpoles:

X position of the origin — specifies on the grid the horizontal position of the vertex from which line originates.
It is expressed in grid units — that is, in `mm`

Y position of the origin — specifies on the grid the vertical position of the vertex from which line originates.
It is expressed in grid units — that is, in `mm`

X position of the end — specifies on the grid the horizontal position of the vertex at which line ends.
In the case of a tadpole, it holds the same value as that of property **X position of the origin**. It is expressed in grid units — that is, in `mm`

Y position of the end — specifies on the grid the vertical position of the vertex at which line ends.
In the case of a tadpole, it holds the same value as that of property **Y position of the origin**. It is expressed in grid units — that is, in `mm`

Type — specifies the graphical style of the line.
It must be the name of an existing `feynmp`/FeynCAD line style (see table below for a list of possible values) or the name of an existing user-defined MetaPost macro. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Size — specifies the size of the line.
It is expressed in multiples of the basic line-width unit — `thin` MetaPost parameter, see `Edit` → `Global Settings` → `MetaPost` → `thin` and {2.7.3}

Color — specifies the color used while drawing the line.

It must be a MetaPost color specified by valid MetaPost code. Examples: an RGB triple as `(.25,1/3,2/3)` or a linear combination of predefined colors `red`, `green` and `blue` like `0.7red+1/5blue`. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Rubout — if a parameter greater than `1` is specified here, a second line of this size is drawn under the actual one, with

1. width equal to **Size** × **Rubout**
2. identical **Type**, **Depth** and remaining geometric parameters
3. the same color as the background of the graph;

as net effect, an erased "border" is produced around the original line. Values of **Rubout** in the range `1.2`–`2.0` are usually enough to make the job.

This feature is typically used to mark surfacing lines against deeper ones, when a lot of them —maybe with complicated graphical styles— intersect.

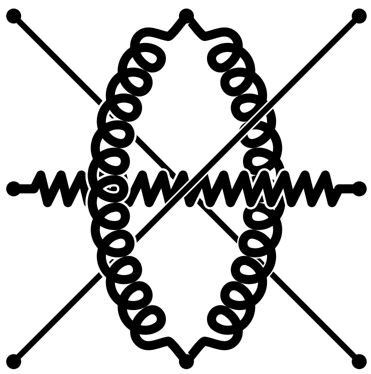


Figure 2.10. Example of stacked lines with non-default values of **Rubout** parameter.

Depth — specifies at which depth line is drawn: in PostScript objects are stacked, and upper layers hide lower layers.

Here follows a table with the description of allowed values for each of the fields which are common to lines and tadpoles:

Field	Type	Allowed values	Default
X position of the origin	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Y position of the origin	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
X position of the end	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Y position of the end	*integer*	from `-1000` to `1000` (in `mm` grid units)	none

Type	*string*	single { `plain`
		`dashes`
		`dots`
		`wiggly`
		`curly`
		`zigzag`
		`vanilla`
		`photon`
		`boson`
		`gluon` } , single arrowed { `plain_arrow`
		`dashes_arrow`
		`dots_arrow`
		`fermion`
		`quark`
		`electron`
		`ghost`
		`scalar` } ,
		double { `dbl_plain`
		`dbl_dashes`
		`dbl_dots`
		`dbl_wiggly`
		`dbl_curly`
		`dbl_zigzag`
		`beam` } , double arrowed { `dbl_plain_arrow`
		`dbl_dashes_arrow`
		`dbl_dots_arrow`
		`beam_arrow`
		`heavy` } ,
		transparent { `phantom`
		`phantom_arrow` }
			`plain`

Size	*float*	from `0` to `2000` (in `thin` units)	`1`

Color	*string*	a MetaPost color specified by valid MetaPost code:
		RGB triples { `(1,0,0)`
		`(.25,1/3,2/3)`
		`...` } , names { `black, white`
		`red, green, blue`
		`0.4white`
		`0.7red+1/5blue`
		`...` }
			`black`

Rubout	*float*	from `1` to `3000` (pure number)	`1`
Depth	*unsigned*	from `1` to `99`	`50`

2.2.2.2 Parameters specific to lines

Three additional parameters are needed to fully specify a line (or a tadpole). Here is the list of such parameters in the case of the line:

Starting angle — specifies in degrees the starting direction for the line (which is an oriented cubic Bézier spline, see [1]); if left empty, a suitable value for it is automatically computed following MetaFONT/MetaPost rules (see [1])

Ending angle — specifies in degrees the ending direction for the line (which is an oriented cubic Bézier spline, see [1]); if left empty, a suitable value for it is automatically computed following MetaFONT/MetaPost rules (see [1])

Tension — describes line stretch in MetaFONT/MetaPost terminology (see [1]). Allowed values range from `3/4` to "infinity", `1` being the default; the larger is the tension, the better the spline approximates the straight line between specified starting and ending point (but always respecting prescripted starting and ending angles too!).

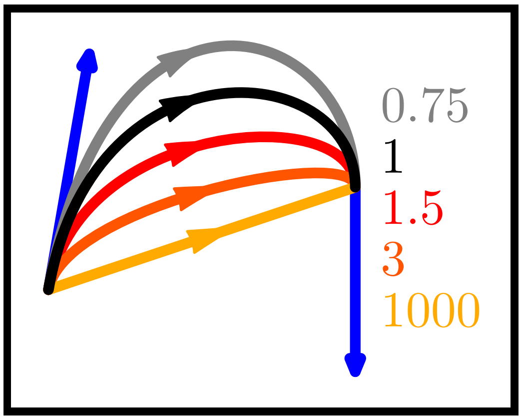


Figure 2.12. How line parameter called **Tension** works.

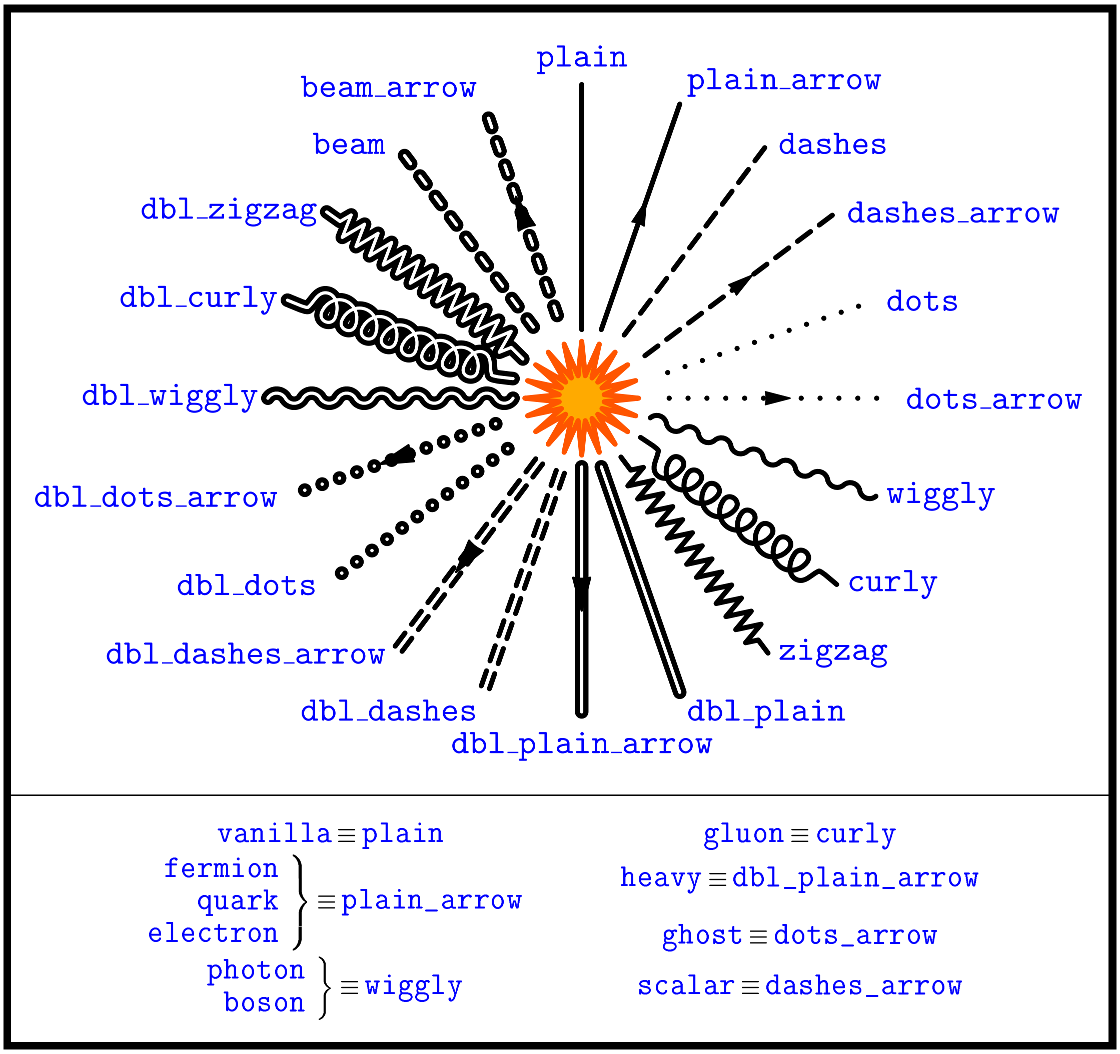


Figure 2.11. A synopsis of main line styles existing in FeynCAD; transparent styles `phantom` and `phantom_arrow` have been omitted. A list of existing physical aliases/shortcuts is shown in the lower part of the table.

Here follows a table with the description of allowed values for each of the fields which are specific to lines:

FIELD	TYPE	ALLOWED VALUES	DEFAULT
Starting angle	*float*	from `-360` to `360` (in degrees)	empty
Ending angle	*float*	from `-360` to `360` (in degrees)	empty
Tension	*float*	from `3/4` to `3000` (pure number)	`1`

2.2.2.3 Parameters specific to tadpoles

Coming now to tadpoles, a preliminary description can be useful to help geometrical intuition about them. They are modeled as lines which start from the base point (**X**,**Y**) —called the *starting point*—, reach another point —called *vertex* of the tadpole— which has maximal distance from the origin, and finally come back to the starting point; tadpoles are guaranteed to be symmetric w.r.t. the *axis* which can ideally be traced to join their starting point and their vertex.

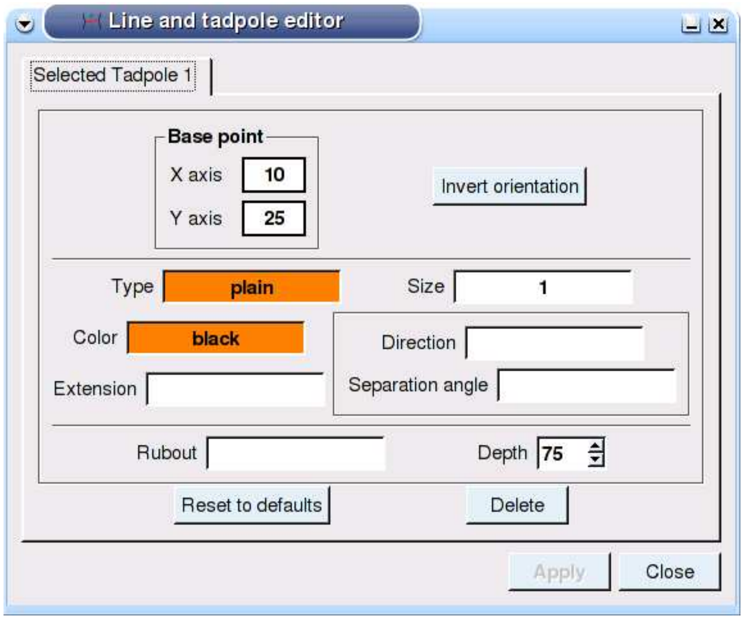


Figure 2.13. How a FeynCAD tadpole editor looks like.

As for lines, even in the case of tadpoles three additional fields other than parameters common to lines and tadpoles are necessary to obtain a complete definition. These are:

Direction — describes in degrees the orientation of tadpole axis — that is, of the ideal line which can be traced to pass through the starting point and the vertex of the tadpole

Separation angle — describes the amplitude in degrees of the fan angle which encapsulates the tadpole; rigorously, it is delimited by starting direction (taken with its natural orientation) and ending direction (taken reversed) of tadpole line.

Extension — describes the distance of tadpole vertex from the starting point — that is, the maximal distance from the origin which is reached by tadpole line.
It is expressed in grid units — that is, in mm.

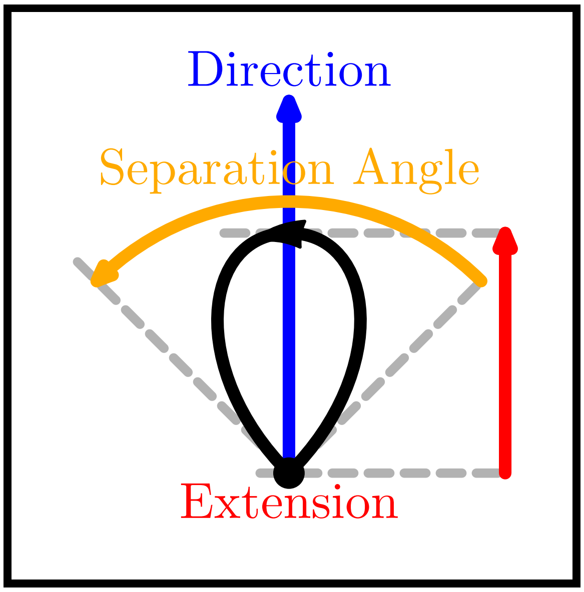


Figure 2.14. The three basic properties which specify a tadpole.

Here follows a table with the description of allowed values for each of the fields which are specific to tadpoles:

FIELD	TYPE	ALLOWED VALUES	DEFAULT
Direction	*float*	from `-360` to `360` (in degrees)	`90`
Separation angle	*float*	from `-360` to `360` (in degrees)	`90`
Extension	*float*	from `0` to `1000` (in `mm` grid units)	`4`

2.2.3 Editing tags

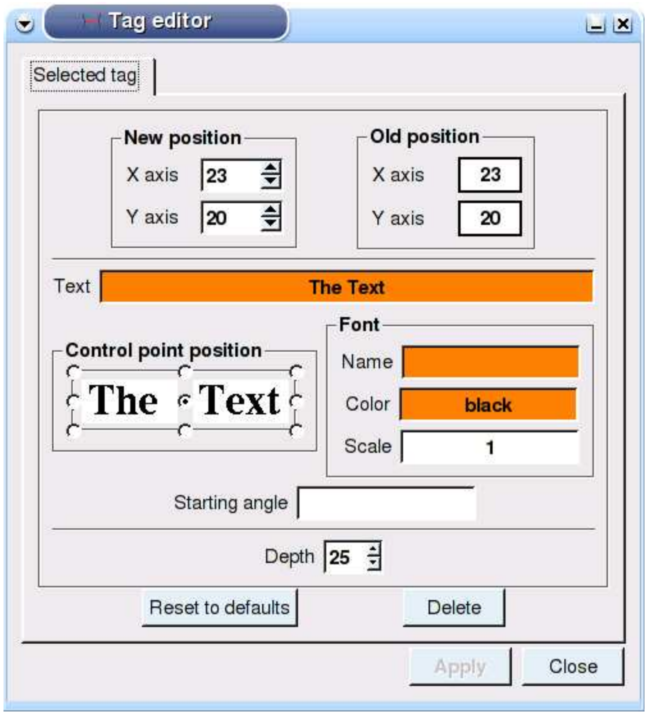


Figure 2.15. How a FeynCAD tag editor looks like.

Here is the list of all parameters allowed for objects of type tag:

X position — specifies the horizontal position of the tag on the grid.
It is expressed in grid units — that is, in `mm`

Y position — specifies the vertical position of the tag on the grid.
It is expressed in grid units — that is, in `mm`

Text — specifies the text of the tag.

I. **When the field "Font name" below is left empty**, *it must be valid TEX code* — but *not* LATEX code!

II. **Otherwise**, it will be rendered as ordinary text.

Please notice that, for example, character `_` is not valid TEX code — and you should rather employ its escaped version `\_` if you intend to typeset it; similar considerations related to TEX syntax imply that as a general rule *in TEX mode you have to be careful even when typing simple text*. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Control point position — specifies in MetaPost style the placement of the text of the tag with respect to its (**X**,**Y**) point (see reference [1] for more informations).

Allowed values are top/center/bottom for Y-coordinate and right/center/left for X-coordinate, but FeynCAD tag editor allows you to directly select the control point position in graphical form

Font name — specifies the name of the PostScript font employed to typeset the text of the tag. This value depends in general on the setups both of your operating system and of your version of Ghostscript (although common valid values should usually be `Palatino-Roman`, `Times-Roman`, etc.); please consult your system administrator if in doubt. *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Font color — specifies the color used while drawing the tag.
It must be a MetaPost color specified by valid MetaPost code (examples: an RGB triple as `(.25,1/3,2/3)` or a linear combination of predefined colors `red`, `green` and `blue` like `0.7red+1/5blue`). *A wrong value for this field will result in a MetaPost error, or in an incomplete rendering of the graph*

Font scale — specifies the magnification at which the text of the tag is drawn

Starting angle — specifies the angle of rotation (in degrees) used while drawing the tag

Depth — specifies at which depth tag is drawn: in PostScript objects are stacked, and upper layers hide lower layers.

Here follows a table with the description of allowed values for each of the fields:

Field	Type	Allowed values	Default
X position	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Y position	*integer*	from `-1000` to `1000` (in `mm` grid units)	none
Text	*string*	if **Font name** is empty, a valid TEX expression; arbitrary text otherwise	empty
Control point position	*string*	top left `tl` top center `tc` top right `tr` center left `cl` center center `cc` center right `cr` bottom left `bl` bottom center `bc` bottom right `br`	`cc`
Font name	*string*	system-dependent: could be something like `Palatino-Roman` `Times-Roman` . ... An empty string specifies TEX typesetting	empty
Font color	*string*	a MetaPost color specified by valid MetaPost code: RGB triples `(1,0,0)`, `(.25,1/3,2/3)`, ... , names `black, white`, `red, green, blue`, `0.4white`, `0.7red+1/5blue`, ...	`black`
Font scale	*float*	from `0` to `3000` (pure number)	`1`
Starting angle	*float*	from `-360` to `360` (in degrees)	`0`
Depth	*unsigned*	from `1` to `99`	`25`

2.3 Editing selections

Once a selection has been created by using the mouse (see {2.1.2}), various operations can be performed on it. Here follows the list of allowed actions:

Move. This action can be performed with the mouse only. See {2.1.2} for details

Paste. This action can be initiated with the mouse (see {2.1.2} for details), with the keyboard (see {2.6} for details) or from menu `Edit` → `Paste selection`. In all these cases, it needs to be completed by placing objects which are being pasted onto their final position; this result can be achieved with an arbitrary sequence of operations from the following list:

i. objects being pasted can be moved with the mouse

ii. paste operation can be abandoned at any time by pressing `Escape` key, see {2.6}

iii. when objects being pasted are finally placed in the desired position, paste operation can be successfully ended by pressing `Return` or `Enter` keys (see {2.6}), or by clicking on the drawing board with any button of the mouse.

Delete. This action can be performed with the keyboard only. See {2.6} for details

Clip. The graph is clipped to the selection, and the grid gets automatically resized; objects outside the boundaries are *not* erased — they are still accessible from the object lists on the left of graph editor. This action is accessible from menu `Edit` → `Transform selection`, which becomes enabled when a non-empty selection is present

Rotate. The selection is rotated a specified angle —whose amount is given as usual in degrees— around a specified point of the grid; a preview of the results of the operation is shown in real-time by the drawing board. This action is accessible from menu `Edit` → `Rotate selection`, which becomes enabled when a non-empty selection is present. While this option can be very useful (for example, while creating drawings of symmetric structures), two caveats should be mentioned:

I. Rotation angle is taken *clockwise* in this case, to allow a more natural formulation of the GUI

II. Due to the discreteness of the grid and to the errors it introduces in the positions of rotated points, a rotation cannot in general be undone by just applying its inverse.

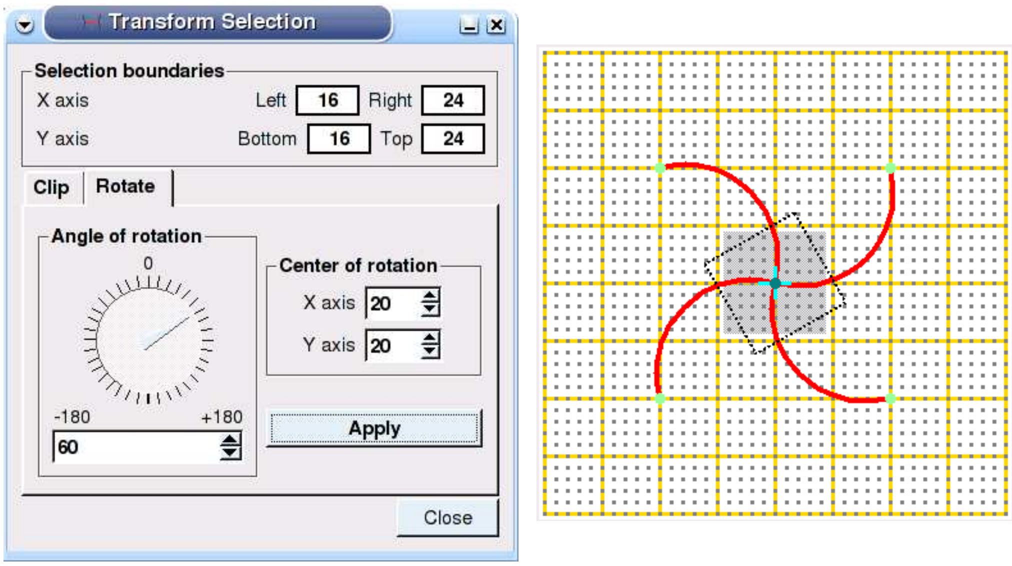


Figure 2.16. Rotating a selection: on the left the selection editor which allows the user to tune rotation parameters, on the right the corresponding real-time preview of the results of the operation.

2.4 Editing global graph properties

Some additional properties are associated to each graph which is being edited —that is, to each open graph editor in the GUI—. These properties are:

1. the **grid size** —that is, the X- and Y-dimensions— of the graph

2. some informations about the graph (creation date, name of the author, etc.) which are collectively denoted as **graph signature**.

2.4.1 Changing grid size

In both modes the user is allowed to change at any time the size of the grid. This can be easily done in four steps:

1. Clicking on button `Grid Size` → `Edit` placed at the bottom of each graph editor. This operation will enable a group of buttons in the `Grid Size` area, and temporarily disable the rest of the application window; button `Grid Size` → `Edit` will toggle to `Grid Size` → `Back`
2. Inserting the new desired size for the grid (if you do it by directly typing in the numbers, do not forget to validate with the `Enter` key when you have finished, otherwise the change will not be transmitted to the GUI, and the `Apply` button will stay disabled)
3. Pressing the `Apply` button placed on the right side of `X` and `Y` fields; this will immediately resize the grid. One can eventually repeat steps 2. and 3. until the size of the grid is satisfactory
4. Clicking on button `Grid Size` → `Back`, which will toggle to `Grid Size` → `Edit` again; this operation will disable the fields in the `Grid Size` area, re-enable the rest of the application window and send the user back to the operation mode he/she was in before starting to edit the grid.

2.4.2 Modifying graph signature

Graph signature is composed of four fields, among which only two can be directly changed by user's intervention. These are:

Author's name — It can be freely edited.

Author's comments — They can be freely edited (and there is almost no limitation about their length)

Date and time of creation — It cannot be modified; its format can depend on local settings of the operating system.

Date and time of last modification — It cannot be modified; its format can depend on local settings of the operating system.

The values of the last two fields are taken care of automatically by FeynCAD; it generates and updates these dates when necessary, and stores/retrieves them as strings formatted in the way internally required by the Qt-libraries.

2.5 History

Each of the graph editors which you open from within FeynCAD is equipped with an independent history, where a number of intermediate stages which lead to the present form of your graph is recorded.

The maximal number of recorded events can be (re)set acting on `Edit` → `Global settings` → `Editor` → `History depth` (the default value is `100`).

When available, preceding/following actions can be reached in three ways:

1. From menus `Edit` → `Undo last action` and `Edit` → `Redo next action`
2. Clicking directly on the two icons associated to the two menus
3. Using keyboard accelerators (see {2.6} below).

Interestingly, every element of the GUI related to history is set up in such a way to show some information about the actions accessible at a given moment: for example, each time the mouse slides over `Undo`/`Redo` buttons the application will show in the status bar a short message, saying something like `Undo last action "Add Line between (21,27) and (25,13)"`.

An aspect of FeynCAD which may sometimes result confusing is the fact that *not all modifications are automatically recorded in the history*: for examples, changes to objects which result in an incorrect MetaPost code are not, and the same is true when a selection is moved onto itself, leaving the situation the same as before.

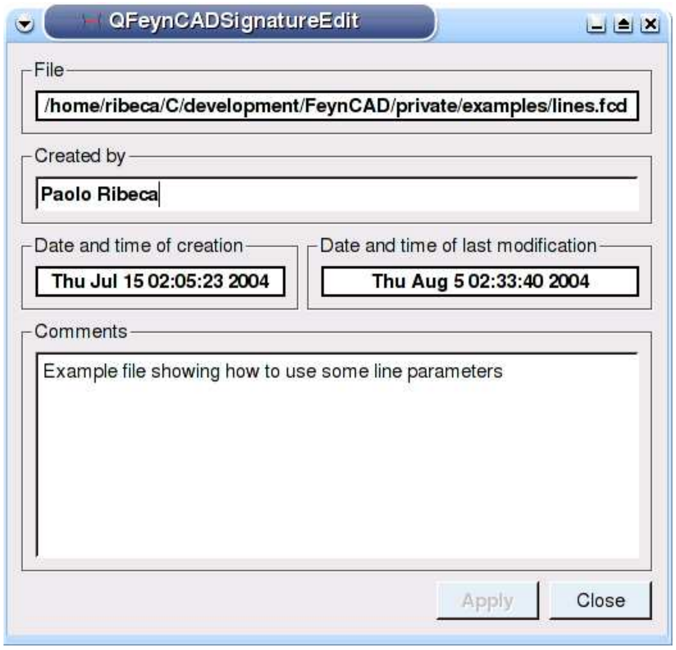


Figure 2.17. Editing the signature of a graph.

2.6 Keyboard accelerators

Keyboard accelerators (sometimes called also *key-bindings*) are fondamental to speedup all operations which involve placing and editing objects on the grid; they can prove very effective, especially when semi-repetitive operations need to be performed. Here follows the complete list:

`Space`	**In Object Mode.** Reselects for editing the last object (no matter if vertex, line or tag) which has been created/modified; thus, if the object has not been deleted in the meanwhile the suitable editing dialog is opened, according to the nature of the object. Productive user should not underestimate the practical relevance of this shortcut. **In Select Mode.** Initiates pasting of last selection, if the clipboard is not empty.
`Insert`	**In Select Mode.** Initiates pasting of last selection, if the clipboard is not empty.
`Delete` `Backspace`	**In Select Mode.** Deletes current selection, if any.
`Return` `Enter`	**While pasting.** Closes paste operation, merging into the graph at its current position the floating group of objects being pasted.
`Escape`	**While pasting.** Abandons paste operation, deleting the floating group of objects being pasted. **Otherwise.** Closes the editing dialogs if any of them is open; quits the application if not.
`F5`	Applies the changes in the editing dialogs, if any of them is open and shows an enabled `Apply` button.
`F1`	Toggles Object and Select mode for current graph.
`PageUp` `ArrowUp`	Undoes the last action, if any.

PageDown ArrowDown	Redoes the next action, if any.

Depending on the part of the GUI he/she is currently using, the user will also find out that other accelerators are active from time to time; they are implicitly defined by Qt-libraries to speed up some frequently-performed operations.

In addition, other accelerators are defined for, and bound to, all most relevant menu actions.

Control N	Opens from scratch a new (empty) graph editor.
Control O	Opens an existing graph which is stored in FeynCAD native format — that is, as a FeynCAD-XML file (currently version 1, see {Appendix 2}).
Control C	Closes the current graph editor.
Control S	Saves the current graph in its native format — that is, as a FeynCAD-XML file (currently version 1, see {Appendix 2}).
Control E	Exports the current graph as an EPS file.
Control H	Opens the on-line-help window.
Control Q	Quits the application.

2.7 Configuring FeynCAD

A lot of aspects in FeynCAD appearance and behaviour can be configured. In particular, there are three major groups of configuration options:

1. Options related to the customization of graph editor(s)
2. Options related to paths and options for needed external auxiliary executables, such as $\TeX$ and Ghostscript
3. Options related to the fine-tuning of global parameters exported by **feynmp**/FeynCAD MetaPost macros.

2.7.1 Configuring graph editor(s)

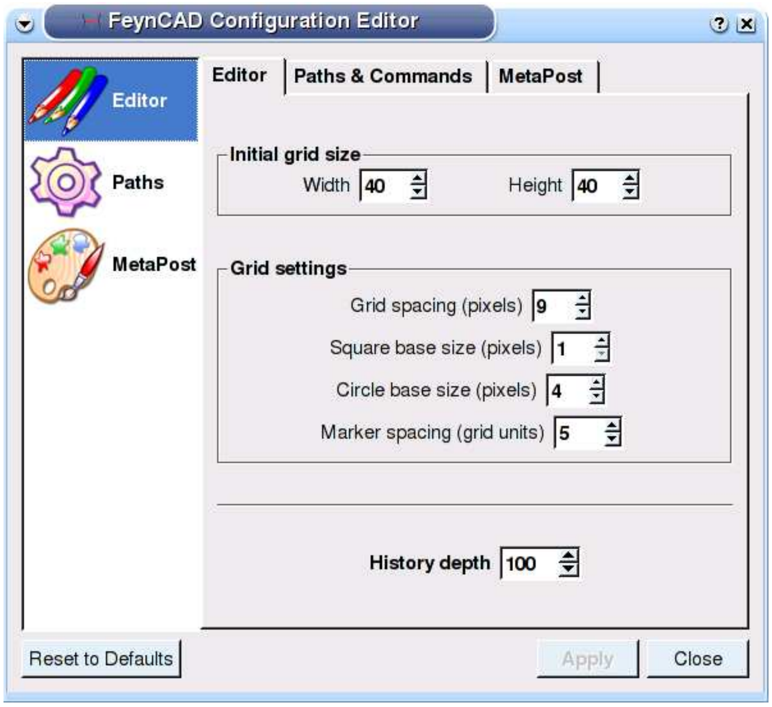


Figure 2.18. Configuring editor options.

Here is the list of configuration options related to graph editor(s):

Initial grid size: Width — sets the upper limit of X grid coordinate for new graphs opened with `File` → `New`; *it does not affect all graph editors which have already been opened*

Initial grid size: Height — sets the upper limit of Y grid coordinate for new graphs opened with `File` → `New`; *it does not affect all graph editors which have already been opened*

Grid spacing — it is the distance in pixels between two adjacent points of the drawing grid. Please notice that the size of the viewer too is (indirectly) influenced by this parameter, since the dimension of the viewer is constrained to be the same as that of the corresponding graph editor

Square base size — controls the size of grid points, which are drawn as squares of approximately twice this size in pixels

Circle base size — controls the size of objects on the grid which are rendered as circles — vertices, tag points, etc. (a vertex is approximately twice this size in pixels)

Marker spacing — determines the frequency of tic marks for the grid (rendered as special orange lines drawn below the grid itself); for example, if the value of this parameter is `6` then there will be special lines on the grid at horizontal and vertical coordinates `0`, `6`, `12`, `18`, etc.. A value of `0` disables the markers

History depth — it is the maximal number of modifications which are kept in memory by each FeynCAD graph editor while the user is drawing. If more operations are performed than the specified number, then in correspondence to each new action it will be the oldest record to be discarded to make room.

2.7.2 Configuring options for paths and external executables

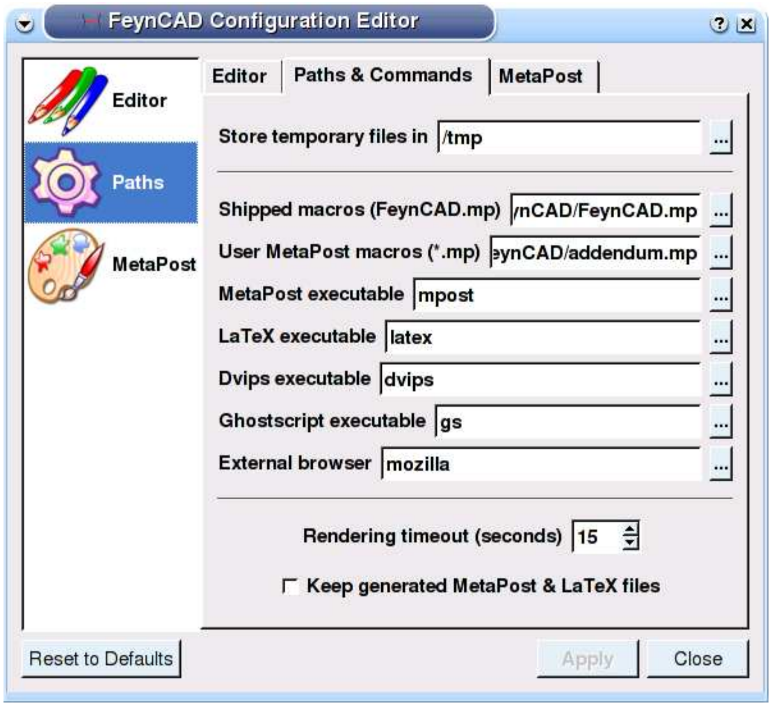


Figure 2.19. Configuring paths and options for required external auxiliary applications.

Here is the list of configuration options related to paths and external executables (please notice that *wrong entries in almost each one of these fields will probably stop FeynCAD from working correctly*):

Store temporary files in — indicates the location of the directory where temporary FeynCAD files will be produced (see below for more informations about them).

The user must have the permissions to access this directory, and to write files into it

Shipped macros — it is the path where file `FeynCAD.mp`, which comes along with standard FeynCAD distribution, has been installed (see {1.2.3.1}).
To avoid problems when this file is included by FeynCAD into generated MetaPost code, *an absolute path should better be specified here*

User MetaPost macros — it is the path where an additional, user-provided file containing MetaPost macros can be found (see {Appendix 1}).
To avoid problems when this file is included by FeynCAD into generated MetaPost code, *an absolute path should better be specified here*

MetaPost executable — it is the path of MetaPost program.
If MetaPost program is not accessible from your usual path, its absolute path must be specified here

LaTeX executable — it is the path of LaTeX program.
If LaTeX program is not accessible from your usual path, its absolute path must be specified here

Dvips executable — it is the path of `dvips` program.
If `dvips` *program is not accessible from your usual path, its absolute path must be specified here*

Ghostscript executable — it is the path of Ghostscript program.
If Ghostscript program is not accessible from your usual path, its absolute path must be specified here

External browser — it is the path of an external browser program, which will be used to open external links while browsing FeynCAD documentation (menu `Help` → `On-line documentation`)

Rendering timeout — it is the maximum span —in seconds— for the refresh of the graph viewer to complete (see {2.1}); if all the rendering chain does not end within this specified duration, it will be terminated.
Please notice that to this timeout contribute the execution times of both the "compilation chain" (MetaPost+LaTeX+`dvips`) and Ghostscript

Keep generated MetaPost & LaTeX files — this boolean flag controls whether, when FeynCAD exits, some of the temporary files produced during the "compilation chain" (MetaPost+LaTeX+`dvips`) should *not* be erased from the temporary directory; if the flag is set to `true`, then all generated MetaPost (`.mp`) and LaTeX (`.tex`) files are kept.
This option can turn out to be very useful while debugging new user-defined macros (see {Appendix 1}) or while trying to understand why the value of some critical parameter makes MetaPost stop (see {2.1}); in this case, one can accurately examine the generated files, and run step-by-step MetaPost and LaTeX on them to try to locate errors.

Some more words can also be useful about the naming convention adopted by FeynCAD while producing temporary files. Each temporary file has a unique name with the structure

`FeynCAD-`*ppppp*`-`*nn*`.`*ext*

where *ppppp* is the process number for that instance of FeynCAD as from an old Unix tradition, *nn* is the number of the graph editor which owns the temporary file under examination, and *ext* is the extension suitable for that particular file — could be `mp`, `tex`, `dvi`, etc.. For example, LaTeX file `FeynCAD-8574-3.tex` will be produced by the third graph editor of a FeynCAD running as process 8574.

Finally, a word of caution: FeynCAD performs an automatical cleanup of all temporary files produced in the meantime at the end of each working session; if you kill it, or exit from it in some other abrupt way, a side effect will be for sure a number of temporary files remaining in your hard-disk (and, possibly, some auxiliary Ghostscript processes which will keep on running, instead of being properly terminated).

2.7.3 Configuring MetaPost macros

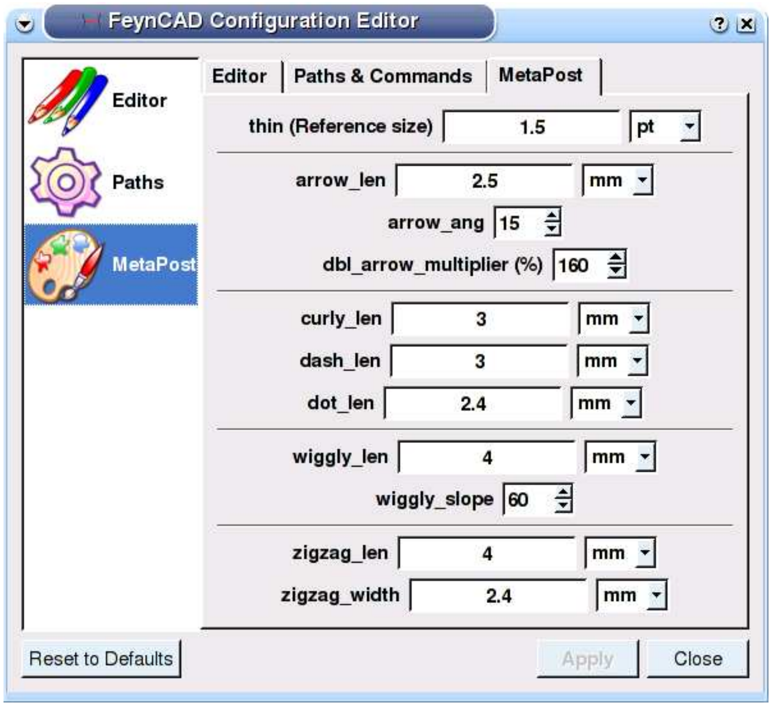


Figure 2.20. Configuring global MetaPost style options.

Here is the list of configuration options related to the behaviour of MetaPost macros:

`thin` — it is the reference thickness for lines; all parameters related to size in lines and in vertex decorations are expressed as multiples of this unit (see {2.2}).
Its "factory value" is `1.5pt`

`arrow_len` — it is the *minimal* size of arrows for non-doubled line styles (FeynCAD arrows can freely grow, automatically adapting their size to that of their carrying lines)

`arrow_ang` — it is the half of the opening angle of the arrow, that is the angle between the direction of the arrow and one of its sides

`dbl_arrow_multiplier` — it is the ratio in percentiles between the size of an arrow for a non-doubled line style and the size of the arrow for the corresponding doubled line style; for example, if the value of this parameter is `160` then the arrows appearing in `dbl_plain_arrow` lines will be 1.6 times larger than the ones appearing in `plain_arrow` lines.
This parameter affects all styles which contain `dbl_` in their names, and these styles alone (plus `beam` style, which is in effect a doubled one, but does not follow usual style naming conventions)

`curly_len` — defines the length of one curl —that is, the length of a single repetition of the basic decoration unit/cell— for `curly` style and its derivative styles (like `dbl_curly`)

`dash_len` — defines the length of one dash —that is, the length of a single repetition of the basic decoration unit/cell— for `dashes` style and its derivative styles (like `dashes_arrow`, `dbl_dashes`, etc.)

`dot_len` — defines the length of one dot —that is, the length of a single repetition of the basic decoration unit/cell— for `dots` style and its derivative styles (like `dots_arrow`, `dbl_dots`, etc.)

`wiggly_len` — defines the length of one wiggle —that is, the length of a single repetition of the basic decoration unit/cell— for `wiggly` style and its derivative styles (like `dbl_wiggly`)

`wiggly_slope` — defines the starting slope of one wiggle —that is, the starting slope of a single repetition of the basic decoration unit/cell— for `wiggly` style and its derivative styles (like `dbl_wiggly`). The larger the slope, the thicker the resulting decoration will be

`zigzag_len` — defines the length of one zigzag —that is, the length of a single repetition of the basic decoration unit/cell— for `zigzag` style and its derivative styles (like `dbl_zigzag`)

`zigzag_width` — defines the width of one zigzag —that is, the width of a single repetition of the basic decoration unit/cell— for `zigzag` style and its derivative styles (like `dbl_zigzag`).

Please notice that the user can choose quite freely in terms of which unit of measure he/she will specify each one of these parameters; allowed units are

1. `mm` (*millimeter*, $1\,\mathtt{mm} \approx 2.83464\,\mathtt{bp}$)
2. `in` (*inch*, $1\,\mathtt{in} = 72\,\mathtt{bp}$)
3. `pt` ("*point*", $1\,\mathtt{pt} \approx 0.99626\,\mathtt{bp}$)
4. `dd` ("*didot*", $1\,\mathtt{dd} \approx 1.06601\,\mathtt{bp}$)
5. `bp` ("*bold point*", $1\,\mathtt{bp} \equiv 1/72\,\mathtt{in}$).

The last three units are quantities traditionally used in typography.

2.7.4 Configuration files for FeynCAD — File `~/.FeynCADrc`

As a conclusive remark, here follows a description of the mechanism employed by FeynCAD to read its configuration. At the beginning of its execution,

i. If an alternative location for the configuration file has been specified on the commandline with the `-c` option (see {1.3}), then the indicated file is opened
ii. If the non-standard configuration file is not a valid one, or no `-c` option has been specified on the commandline, FeynCAD tries to open the standard configuration file `~/.FeynCADrc`, where symbol "~" stands for "user's home directory".
iii. If this second step fails too, FeynCAD prompts the user about the lack of a valid configuration file, and starts using the default "factory" values, which can be found in {Appendix 3}.
iv. At the end of the session, FeynCAD will save the configuration to the non-standard location specified by the user if the `-c` option has been used on the commandline when the program was invoked, or to the default location `~/.FeynCADrc` otherwise.

This implies that during the first session immediately after its installation FeynCAD will notify the user that it could not find the configuration file `~/.FeynCADrc`; however, this message is harmless, since this file will be automatically created when the program exits.

Please notice that configuration is read even if FeynCAD is started in batch mode without graphical interface (option `-b` given on the commandline, see {1.3}). Thus, all observations made before apply as well to this case, too; the same is true also for the convention about names of temporary files as explained at the end of {2.7.2}, with the sole exception that in this case the number *nn* —used to indicate the graph editor which owns the file— will be set to `0` for all files produced during the batch session (meaning that all those temporaries will have names of the form `FeynCAD-`*ppppp*`-0.`*ext*).

Appendices

1 How to create custom MetaPost macros for FeynCAD

Use the Source, Luke...
—Anonymous—

Following the original spirit and structure of `feynMF`/`feynmp` package, the set of macros shipped with FeynCAD can be easily customized/extended.

To be able to do that, you basically need some knowledge of MetaPost (refer again to [1] at the beginning of chapter 2), and some knowledge of the mechanisms used by `feynmp` to define new vertex and line types; the best solution is to carefully read [2], and then just jump directly into the code of `FeynCAD.mp`. If something is still unclear, you can often take advantage of the comments which can be found in file `feynmf.dtx` of package `feynMF`; even if most of the macros used by FeynCAD have been modified with respect to their original `feynmp` version (see {1.2.2.1}), very often those hints and insights about their structure still hold true.

Just to clarify how simple is defining new objects in `feynmp`/FeynCAD, let us present an example. Fire up your favourite editor and type into a new file the code

```
style_def arrow expr p =
  drawarrow p withcolor foreground;
enddef;

endinput;
\endinput
```

Once you are done, save the file with a name you like (for purposes of clarity we will suppose in the rest of the section that this name is `addendum.mp`), start FeynCAD and open its configuration editor; then add the full path of file `addendum.mp` under the location `Edit` → `Global Settings` → `Paths & Commands` → `User MetaPost macros`. From the moment you apply the change on, FeynCAD will include your brand-new file `addendum.mp` at the beginning of each generated diagram ("*at the beginning*" means here "*immediately after file* `FeynCAD.mp`*, and just before the generated MetaPost code for the graph*").

What now? Now, you will automatically have at your disposal for use in all your graphs a new line style, called `arrow`, which draws nice arrows in place of simple lines. Defining code is so short in this case because it takes advantage of MetaPost' predefined macro `drawarrow`; after having read MetaPost manual you should be able, for example, to immediately declare other line styles which trace in a fairly similar way double-arrowed lines (one arrow at both ends of the line, towards the exterior or as well towards the interior of the line); of course, much more complicated definitions can be given as well.

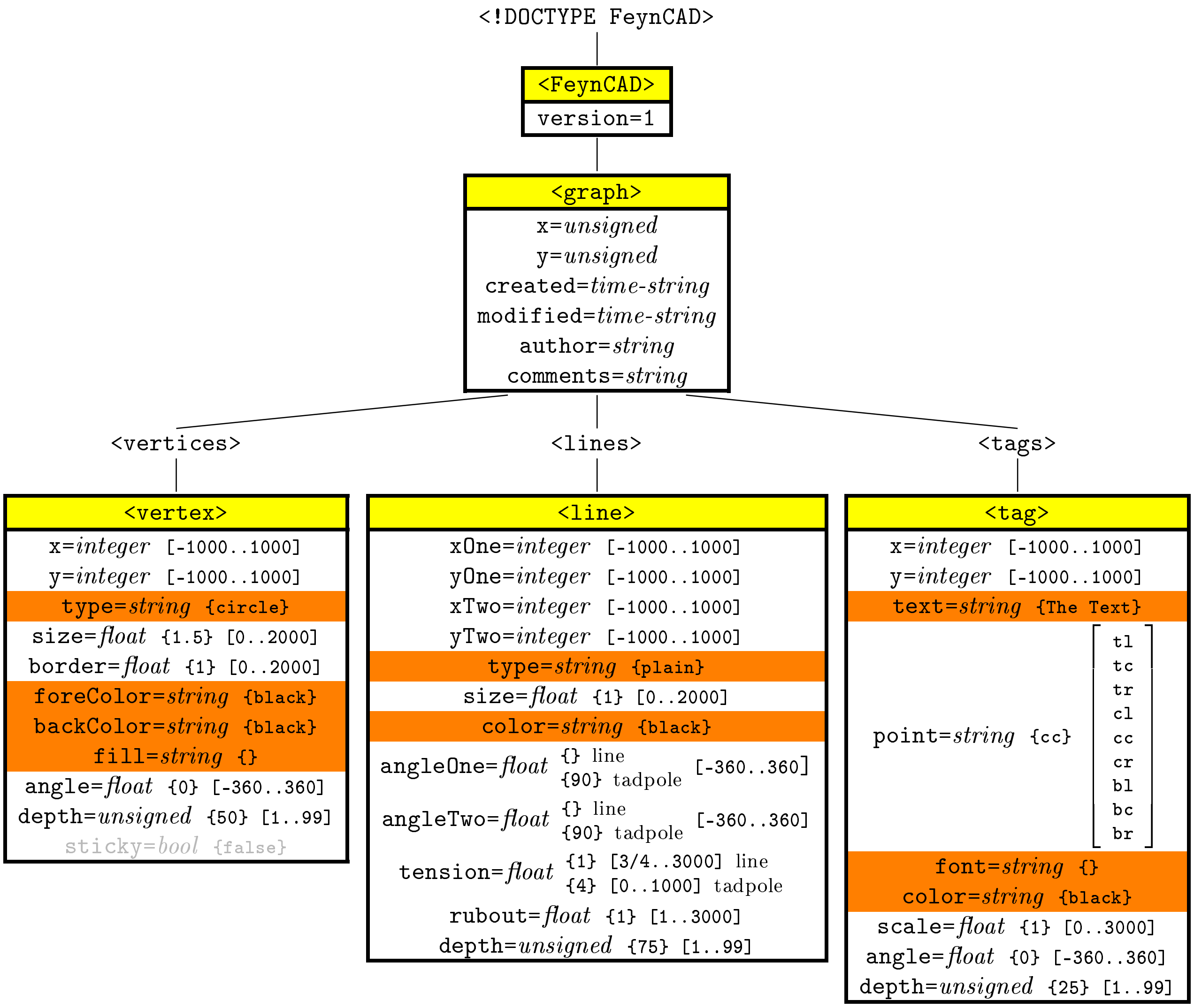
It should be quite clear by now that, despite its simplicity, this mechanism which FeynCAD provides of incorporating user-specified extensions by including a custom MetaPost file into the generated graphs is very general, since arbitrarily complicated macro files can easily be written; thus, FeynCAD user is free to enlarge at will the set of available vertex and line styles.

[And remember: for very difficult cases, when you are not entirely satisfied with your results, you are still free to modify/patch the style file `FeynCAD.mp` itself.]

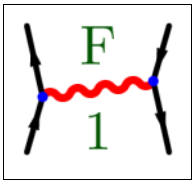
2 Syntax of FeynCAD-XML version 1

Here follows a schematic specification of version 1 of FeynCAD-XML; this document structure is used internally by FeynCAD to store/retrieve graphs. Some typographical conventions are adopted:

- As usual, fields with an orange background are those pieces of information whose correctness (in the sense of `feynmp`/FeynCAD and user-defined macro specifications, see {2.2}) is essential to ensure a proper visualization of the specified graph
- The notation [*a*..*b*], where *a* and *b* are two numerical values, indicates a continuous range of allowed values
- A value in curly brackets (like {3/4} for example) is the one which is picked up by default.



As a practical example, the following valid FeynCAD-XML-v1 file describes the FeynCAD application icon shown on the right.



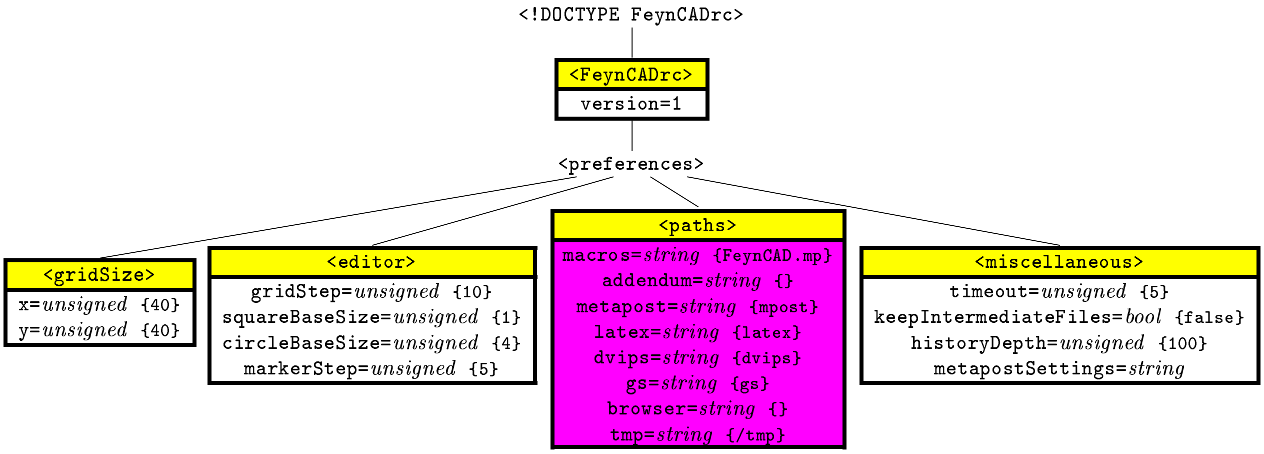

```
<!DOCTYPE FeynCAD>
<FeynCAD version="1" >
 <graph x="20" y="20" created="Sun Jul 25 22:50:35 2004" author="Paolo Ribeca"
        modified="Sun Jul 25 23:34:17 2004" comments="" >
 <vertices>
  <vertex x="1" y="2" fill="0" size="1.5" foreColor="black" type="plain" backColor="black" depth="50"
          angle="0" />
  <vertex x="1" y="18" fill="0" size="1.5" foreColor="black" type="plain" backColor="black" depth="50"
```

```
          angle="0" />
  <vertex x="3" y="9" fill="0" size="1.5" foreColor="blue" type="circle" backColor="blue" depth="50"
          angle="0" />
  <vertex x="17" y="11" fill="0" size="1.5" foreColor="blue" type="circle" backColor="blue" depth="50"
          angle="0" />
  <vertex x="19" y="2" fill="0" size="1.5" foreColor="black" type="plain" backColor="black" depth="50"
          angle="0" />
  <vertex x="19" y="18" fill="0" size="1.5" foreColor="black" type="plain" backColor="black" depth="50"
          angle="0" />
 </vertices>
 <lines>
  <line yOne="2" xOne="1" size="1.5" tension="0" angleTwo="-361" yTwo="9" xTwo="3" type="fermion"
        angleOne="-361" depth="75" color="black" rubout="1" />
  <line yOne="9" xOne="3" size="1.5" tension="0" angleTwo="-361" yTwo="18" xTwo="1" type="fermion"
        angleOne="-361" depth="75" color="black" rubout="1" />
  <line yOne="9" xOne="3" size="2" tension="0" angleTwo="-361" yTwo="11" xTwo="17" type="photon"
        angleOne="-361" depth="76" color="red" rubout="1" />
  <line yOne="11" xOne="17" size="1.5" tension="0" angleTwo="-361" yTwo="2" xTwo="19" type="fermion"
        angleOne="-361" depth="75" color="black" rubout="1" />
  <line yOne="18" xOne="19" size="1.5" tension="0" angleTwo="-361" yTwo="11" xTwo="17" type="fermion"
        angleOne="-361" depth="75" color="black" rubout="1" />
 </lines>
 <tags>
  <tag x="10" y="9" scale="2" point="1" depth="25" color="1/3green" angle="0" font="" text="1" />
  <tag x="10" y="11" scale="2" point="6" depth="25" color="1/3green" angle="0" font="" text="F" />
 </tags>
</graph>
</FeynCAD>
```

3 Syntax of FeynCADrc-XML version 1

Here follows a schematic specification of version 1 of FeynCADrc-XML; this document structure is used internally by FeynCAD to read/write configuration files.

Fields with a magenta background are those (system-dependent) pieces of information whose knowledge is essential to ensure a proper FeynCAD execution.



Some words more should be spent about the parameter `metapostSettings`: it contains in an encoded way all the information necessary to initialize global parameters for `feynmp`/FeynCAD MetaPost macros, which appear in menu `Edit` → `Global settings` → `MetaPost` (see {2.7.3}). If necesssary, its syntax (which is related to the structure of the GUI and thus can in principle change from one version of FeynCAD to another) can be deduced directly from FeynCAD source code.

As a practical example, the following valid FeynCADrc-XML-v1 file describes how FeynCAD is currently configured on the author's machine (it is, indeed, a verbatim copy of his `~/.FeynCADrc` file, see {2.7.4}):

```
<!DOCTYPE FeynCADrc>
<FeynCADrc version="1" >
 <preferences>
  <gridSize x="40" y="40" />
  <editor gridStep="9" markerStep="5" circleBaseSize="4" squareBaseSize="1" />
  <paths tmp="/tmp" dvips="dvips" addendum="/home/ribeca/C/development/FeynCAD/addendum.mp"
         macros="/home/ribeca/C/development/FeynCAD/FeynCAD.mp" browser="mozilla" latex="latex"
         gs="gs" metapost="mpost" />
  <miscellaneous timeout="15"
                 metapostSettings="2 1.5 0 2.5 0 3 0 3 0 2.4 0 4 0 4 0 2.4 15 160 60"
                 keepIntermediateFiles="1" historyDepth="100" />
 </preferences>
</FeynCADrc>
```

Index