Veusz - a scientific plotting package

Jeremy Sanders

Veusz - a scientific plotting package
by Jeremy Sanders

Table of Contents

Chapter 1. Introduction

1.1. Veusz

Veusz is a scientific plotting package. It was written as I was dissatisfied with existing plotting packages as they were either old and unmaintained (like pgplot and qdp, which does not support postscript fonts), not free (like IDL or Matlab), or had user interfaces I do not appreciate (like gnuplot).

Veusz is designed to be easily extensible, and is written in Python, a high level language (in the future it may be necessary to write some parts in another language for speed, but this will be kept to an absolute minimum). It is also designed in an object oriented fashion, where a document is built up by a number of parts in a hierarchy. The advantage of using Python is that is is easy to allow the user to script Veusz using Python as a very powerful scripting language. Indeed, the saved file format is a Python script.

The technologies behind Veusz include PyQt (a very easy to use Python interface to Qt, which is used for rendering and the graphical user interface, GUI) and numpy (a package for Python which makes the handling of large datasets easy).

Veusz has two user interfaces: a graphical one which gives the user a relatively shallow learning curve, and a command line interface. The command line interface is also used by scripting and in the saved file format.

Furthermore, Veusz can be embedded within other Python programs, with plots in their own windows, or it can be controlled from any other application.

1.2. Terminology

Here I define some terminology for future use.

1.2.1. Part

A document and its graphs are built up from parts. These parts can often by placed within each other, depending on the type of the part. A part has children, those parts placed within it, and its parent. The parts have a number of different settings which modify their behaviour. These include the font to be used, the line thickness, and whether an axis is logarithmic. In addition they have actions, which perform some sort of activity on the part or its children, like "fit" for a fit part.

Parts are specified with a "path", like a file in Unix or Windows. These can be relative to the current part (do not start with a slash), or absolute (do not start with a slash). Examples of paths include, "/page1/graph1/x", "x" and ".".

The part types include

1. document - representing a complete document. A document can contain pages. In addition it contains a setting giving the page size for the document.
2. page - representing a page in a document. One or more graphs can be placed on a page, or a grid.
3. graph - defining an actual graph. A graph can be placed on a page or within a grid. Contained within the graph are its axes and plotters. A graph can be given a background fill and a border if required. It also has a margin, which specifies how far away from the edge of its parent part to plot the body of the graph.

   A graph can contain several axes, at any position on the plot. In addition a graph can use axes defined in parent parts, shared with other graphs.

   More than one graph can be placed within in a page. The margins can be adjusted so that they lie within or besides each other.

4. grid - containing one or more graphs. A grid plots graphs in a gridlike fashion. You can specify the number of rows and columns, and the plots are automatically replotted in the chosen arrangement. A grid can contain graphs or axes. If an axis is placed in a grid, it can be shared by the graphs in the grid.
5. axis - giving the scale for plotting data. An axis translates the coordinates of the data to the screen. An axis can be linear or logarithmic, it can have fixed endpoints, or can automatically get them from the plotted data. It also has settings for the axis labels and lines, tick labels, and major and minor tick marks.

   An axis may be "horizontal" or "vertical" and can appear anywhere on its parent graph or grid.

   If an axis appears within a grid, then it can be shared by all the graphs which are contained within the grid.

6. plotters - types of parts which plot data or add other things on a graph. There is no actual plotter part which can be added, but several types of plotters listed below. Plotters typically take an axis as a setting, which is the axis used to plot the data on the graph (default x and y).
   a. function - a plotter which plots a function on the graph. Functions can be functions of x or y (parametric functions are not done yet!), and are defined in Python expression syntax, which is very close to most other languages. For example "3*x**2 + 2*x - 4". A number of functions are available (e.g. sin, cos, tan, exp, log...). Technically, Veusz imports the numpy package when evaluating, so numpy functions are available.

As well as the function setting, also settable is the line type to plot the function, and the number of steps to evaluate the function when plotting. The function part will also support filled regions when implemented.

b. xy - a plotter which plots scatter, line, or stepped plots. This versatile plotter takes an x and y dataset, and plots (optional) points, in a chosen marker and colour, connecting them with (optional) lines, and plotting (optional) error bars. An xy plotter can also plot a stepped line, allowing histograms to be plotted (note that it doesn't yet do the binning of the data).

   The settings for the xy part are the various attibutes for the points, line and error bars, the datasets to plot, and the axes to plot on.

   The xy plotter can plot a label next to each dataset, which is either the same for each point or taken from a text dataset.

   If you wish to leave gaps in a plot, the input value "nan" can be specified in the numeric dataset.

c. fit - fit a function to data. This plotter is a like the function plotter, but allows fitting of the function to data. This is achived by clicking on a "fit" button, or using the "fit" action of the part. The fitter takes a function to fit containing the unknowns, e.g. "a*x**2 + b*x + c", and initial values for the variables (here a, b and c). It then fits the data (note that at the moment, the fit plotter fits all the data, not just the data that can be seen on the graph) by minimising the chi-squared.

   In order to fit properly, the y data (or x, if fitting as a function of x) must have a properly defined, preferably symmetric error. If there is none, Veusz assumes the same fractional error everywhere, or symmetrises asymmetric errors.

   Note that more work is required in this part, as if a parameter is not well defined by the data, the matrix inversion in the fit will fail. In addition Veusz does not supply estimates for the errors or the final chi-squared in a machine readable way.

d. key - a box which describes the data plotted. If a key is added to a plot, the key looks for "key" settings of the other data plotted within a graph. If there any it builds up a box containing the symbol and line for the plotter, and the text in the "key" setting of the part. This allows a key to be very easily added to a plot.

   The key may be placed in any of the corners of the plot, in the centre, or manually placed. Depending on the ordering of the parts, the key will be placed behind or on top of the part. The key can be filled and surrounded by a box, or not filled or surrounded.

e. label - a text label places on a graph. The alignment can be adjusted and the font changed. The position of the label can be specified in fractional terms of the current graph, or using axis coordinates.
f. contour - plot contours of a 2D dataset on the graph. Contours are automatically calculated between the minimum and maximum values of the graph or chosen manually. The line style of the contours can be chosen individually and the region between contours can be filled with shading or color.

   2D datasets currently consist of a regular grid of values between minimum and maximum positions in x and y. They can be constructed from three 1D datasets of x, y and z if they form a regular x, y grid.

g. image - plot a 2D dataset as a colored image. Different color schemes can be chosen. The scaling between the values and the image can be specified as linear, logarithmic, square-root or square.

1.2.2. Measurements

Distances, widths and lengths in Veusz can be specified in a number of different ways. These include absolute distances specified in physical units, e.g. 1cm, 0.05m, 10mm, 5in and 10pt, and relative units, which are relative to the largest dimension of the page, including 5%, 1/20, 0.05.

1.2.3. Settings

The various settings of the parts come in a number of types, including integers (e.g. 10), floats (e.g. 3.14), text ("hi there!"), distances (see above), options ("horizontal" or "vertical" for axes).

Veusz performs type checks on these parameters. If they are in the wrong format the control to edit the setting will turn red. In the command line, a TypeError exception is thrown.

In the GUI, the current page is replotted if a setting is changed when enter is pressed or the user moves to another setting.

1.2.4. Text

Veusz understands a limited set of LaTeX-like formatting for text. There are some differences (for example, "10^23" puts the 2 and 3 into superscript), but it is fairly similar. You should also leave out the dollar signs. Veusz supports superscripts ("^"), subscripts ("_"), brackets for grouping attributes are "{" and "}".

Special symbols include greek letters ("\alpha" to "\omega" and "\Alpha" to "\Omega"). Also included are symbols "\times", "\pm", "\deg", "\divide", "\dagger", "\ddagger", "\bullet", "\AA", "\sqrt", "\propto", "\infty", "\int", "\sim", "\odot", "\leftarrow", "\uparrow", "\rightarrow", "\downarrow" and "\circ". Please request additional characters if they are required (and exist in the unicode character set). Special symbols can be included directly from a character map.

Also supported are commands to change font. The command "\font{name}{text}" changes the font text is written in to name. This may be useful if a symbol is missing from the current font, e.g. "\font{symbol}{g}" should produce a gamma. You can increase, decrease, or set the size of the font with "\size{+2}{text}", "\size{-2}{text}", or "\size{20}{text}". Numbers are in points.

Various font attributes can be changed: for example, "\italic{some italic text}" (or use "\textit" or "\emph"), "\bold{some bold text}" (or use "\textbf") and "\underline{some underlined text}".

Example text could include "Area / \pi (10^{-23} cm^{-2})", or "\pi\bold{g}".

Veusz plots these symbols with Qt's unicode support. If your current font does not contain these symbols then you may get messy results. If you find this is the case, I highly recommend you to down load Microsoft's Corefonts (see http://corefonts.sourceforge.net/).

1.3. Installation

Please look at the Installation notes (INSTALL) for details on installing Veusz.

1.4. The main window

You should see the main window when you run Veusz.

The Veusz window is split into several sections. At the top is the menu bar and tool bar. These work in the usual way to other applications. Sometimes options are disabled (greyed out) if they do not make sense to be used. If you hold your mouse over a button for a few seconds, you will usually get an explanation for what it does called a "tool tip".

Below the main toolbar is a second toolbar for constructing the graph by adding parts (on the left), and some editing buttons. The add part buttons add the request part to the currently selected part in the selection window. The parts are arranged in a tree-like structure.

Below these toolbars and to the right is the plot window. This is where the current page of the current document is shown. You can adjust the size of the plot on the screen (the zoom factor) using the "View" menu or the zoom tool bar button (the magnifying glass). Initially you will not see a plot in the plot window, but you will see the Veusz logo. At the moment you cannot do much else with the window. In the future you will be able to click on items in the plot to modify them.

To the left of the plot window is the selection window, and the properties and formatting windows. The properties window lets you edit various aspects of the selected part (such as the minimum and maximum values on an axis). Changing these values should update the plot. The formatting lets you modify the appearance of the selected part. There are a series of tabs for choosing what aspect to modify.

The various windows can be "dragged" from the main window to "float" by themselves on the screen.

To the bottom of the window is the console. This is a Veusz and Python command line console. To read about the commands available see Commands. As this is a Python console, you can enter mathematical expressions (e.g. "1+2.0*cos(pi/4)") here and they will be evaluated when you press Enter. The usual special functions and the operators are supported. You can also assign results to variables (e.g. "a=1+2") for use later. The console also supports command history like many Unix shells. Press the up and down cursor keys to browse through the history. Command line completion is not available yet!

1.5. My first plot

On the left of the main window, you will see the document is selected in the selection window. The toolbar above adds a new part to the selected part. Click on the page icon to add a new page to the document. You will see the new page is selected in the editing window. You can now see that other types of part are able to be added.

You can then click on the "graph" button (which looks like a set of axes). You will see a graph along with axes are added.

Select the x axis which has been added to the document (click on "x" in the selection window). In the properties window you will see a variety of different properties you can modify. For instance you can enter a label for the axis by writing "Area (cm^{2})" in the box next to label and pressing enter. Veusz supports text in LaTeX-like form (without the dollar signs). Other important parameters is the "log" switch which switches between linear and logarithmic axes, and "min" and "max" which allow the user to specify the minimum and maximum values on the axes.

The formatting dialog lets you edit various aspects of the graph appearance. For instance the "Line" tab allows you to edit the line of the axis. Click on "Line", then you can then modify its colour. Enter "green" instead of "black" and press enter. Try making the axis label bold.

Now you can try plotting a function on the graph. If the graph, or its children are selected, you will then be able to click the "function" button at the top (a red curve on a graph). You will see a straight line (y=x) added to the plot. If you select "function1", you will be able to edit the functional form plotted and the style of its line. Change the function to "x**2" (x-squared).

We will now try plotting data on the graph. Go to your favourite text editor and save the following data as test.dat:

```
1     0.1   -0.12   1.1    0.1
2.05  0.12  -0.14   4.08   0.12
2.98  0.08  -0.1    2.9    0.11
4.02  0.04  -0.1    15.3   1.0
```

The first three columns are the x data to plot plus its asymmetric errors. The final two columns are the y data plus its symmetric errors. In Veusz, go to the "Data" menu and select "Import". Type the filename into the filename box, or use the "Browse..." button to search for the file. You will see a preview of the data pop up in the box below. Enter "x,+,- y,+-" into the descriptors edit box. This describes the format of the data which describes dataset "x" plus its asymmetric errors, and "y" with its symmetric errors. If you now click "Import", you will see it has imported datasets "x" and "y".

To plot the data you should now click on "graph1" in the tree window. You are now able to click on the "xy" button (which looks like points plotted on a graph). You will see your data plotted on the graph. Veusz plots datasets "x" and "y" by default, but you can change these in the properties of the "xy" plotter.

You are able to choose from a variety of markers to plot. You can remove the plot line by choosing the "Plot Line" subsetting, and clicking on the "hide" option. You can change the colour of the marker by going to the "Marker Fill" subsetting, and entering a new colour (e.g. red), into the colour property.

Chapter 2. Reading data

Currently Veusz supports reading data from a text file (a qdp-alike interface is also planned), FITS format files or from CSV files. Reading data is supported using the "Data, Import" dialog, or using the ImportFile and ImportString commands which read data from files or an existing Python string (allowing data to be embedded in a Python script). Alternatively, a user may read the data themselves using a Python script, and import it into the program with the SetData command.

CSV files are intuitive to use and are described below.

In addition data may also be read in from FITS files if the PyFITS Python module is installed. FITS is a widespread astronomical data format. FITS files are read using the FITS tab on the import dialog or using the ImportFITSFile command.

Two dimensional data are also supported using the 2D tab on the Import dialog box, ImportFile2D and ImportString2D commands.

2.1. Descriptors

The "Data, Import" dialog box, ImportFile and ImportString commands use a "Descriptor" to describe how the data are formatted in the import file. The descriptor at its simplest is a list of the names of the datasets to import (which are columns in the file). Additionally modifiers are added if errors are also read in. Examples of descriptors are below:

1. **x y** two columns are present in the file, they will be read in as datasets "x" and "y".
2. **x,+- y,+,-** two datasets are in the file. Dataset "x" consists of the first two columns. The first column are the values and the second are the symmetric errors. "y" consists of three columns (note the comma between + and -). The first column are the values, the second positive asymmetric errors, and the third negative asymmetric errors.

   Suppose the input file contains:

   ```
   1.0  0.3  2   0.1  -0.2
   1.5  0.2  2.3 2e-2 -0.3E0
   2.19 0.02 5    0.1 -0.1
   ```

   Then x will contain "1+-0.3", "1.5+-0.2", "2.19+-0.02". y will contain "2 +0.1 -0.2", "2.3 +0.02 -0.3", "5 +0.1 -0.1".

3. **x[1:2] y[:]** the first column is the data "x_1", the second "x_2". Subsequent columns are read as "y[1]" to "y[n]".

4. **y[:]+-** read each pair of columns as a dataset and its symmetric error, calling them "y[1]" to "y[n]".
5. **x,,+-** read the first column as the x dataset, skip a column, and read the third column as its symmetric error.

When reading in data, Veusz just uses whitespace for separating columns. The columns do not actually need to be in columns! Furthermore a "\" symbol can be placed at the end of a line to mark a continuation. Veusz will read the next line as if it were placed at the end of the current line. In addition comments and blank lines are ignored. Comments start with a "#", ";", "!" or "%", and continue until the end of the line. The special value "nan" can be used to specify a break in a dataset.

Veusz now has support for reading in other types of data other than numbers. The type of data can be added in round brackets after the name. Veusz will try to guess the type of data based on the first value, so you should specify it if there is any form of ambiguity (e.g. is 3 text or a number). Supported types are numbers (use numeric in brackets) and text (use text in brackets). An example descriptor would be "x(numeric),+- y(numeric),+,- label(text)" for an x dataset followed by its symmetric errors, a y dataset followed by two columns of asymmetric errors, and a final column of text for the label dataset.

A text column does not need quotation unless it contains space characters or escape characters. Quotation marks are recommended if you wish to avoid ambiguity. Text is quoted according to the Python rules for text. Double or single quotation marks can be used, e.g. "A 'test'", 'A second "test"'. Quotes can be escaped by prefixing them with a backslash, e.g. "A new \"test\"". If the data are generated from a Python script, the repr function provides the text in a suitable form.

Data may be optionally split into "blocks" of data separated by blank lines (or the word "no" on a line, for obscure reasons). The name of each read in dataset has an underscore and the number of the block (starting from 1) added. This is specified by clicking the blocks checkbox in the import dialog, or by using the useblocks=True option on the ImportFile or ImportString commands.

Instead of specifying the descriptor in the import dialog, the descriptor can be placed in the data file using a descriptor statement on a separate line, consisting of "descriptor" followed by the descriptor. Multiple descriptors can be placed in a single file, for example:

```
# here is one section
descriptor x,+- y,+,-
1 0.5  2 0.1 -0.1
2 0.3  4 0.2 -0.1

# my next block
descriptor alpha beta gamma
1 2 3
4 5 6
7 8 9

# etc...
```

If data are imported from a file, Veusz will normally save the data in its saved document format. If the data are changing, quite often one wants to reread the data from the input file. This can be achieved using the "linked=True" option on the ImportFile command, or by clicking the "Link" checkbox in the import dialog.

2.2. Reading other sorts of data

As mentioned above, a user may write some Python code to read a data file or set of data files. This is convenient if the data data are not ordered in trivial columns.

Suppose an input file "in.dat" contains the following data:

```
1   2
2   4
3   9
4   16
```

Of course this data could be read using the ImportFile command. However, you could also read it with the following Veusz script (which could be saved to a file and loaded with **execfile** or Load. The script also places symmetric errors of 0.1 on the x dataset.

```
x = []
y = []
for line in open("in.dat"):
    parts = [float(i) for i in line.split()]
    x.append(parts[0])
    y.append(parts[1])

SetData('x', x, symerr=0.1)
SetData('y', y)
```

2.3. Reading CSV files

CVS (comma separated variable) files are often written from other programs, such as spreadsheets, including Excel and Gnumeric. Veusz supports reading from these files.

In the import dialog choose "CSV", then choose a filename to import from. In the CSV file the user should place the data in either rows or columns. Veusz will use a name above a column or to the left of a row to specify what the dataset name should be. The user can use new names further down in columns or right in rows to specify a different dataset name. Names do not have to be used, and Veusz will assign default "col" and "row" names if not given. You can also specify a prefix which is prepended to each dataset name read from the file.

To specify symmetric errors for a column, put "+-" as the dataset name in the next column or row. Asymmetric errors can be stated with "+" and "-" in the columns.

The data can be linked with the CSV file so that it can be updated when the file changes. See the example CSV import for details.

Text datasets are not yet autodetected for CSV files. You can specify a text dataset by using "(text)" after the name of the dataset at the top of the column. A future release show allow a more general method for autodetecting data or specifying, as the standard Veusz file format currently does.

2.4. Manipulating datasets

Imported datasets can easily be modified in the Edit data dialog box, by clicking on a value and entering a new one. What is probably more interesting is using the Create dialog box to make new datasets from scratch or based on other datasets.

New datasets can be made by entering a name, and choosing whether to make a dataset consisting of a single value or over a range, from expressions based on a parametric expression, or from expressions based on existing datasets.

For instance, if the user has already imported dataset d, then they can create d2 which consists of d**2. Expressions are in Python syntax and can include the usual mathematical functions. Error bars can also be given as expressions of other datasets. By appending _data, _serr, _perr or _nerr to the name of the dataset in the expression, the user can base their expression on particular parts of the given dataset (the main data, symmetric errors, positive errors or negative errors). Otherwise the program uses the same parts as is currently being specified.

A particularly useful feature is to be able to link a dataset to an expression, so if the expression changes the dataset changes with it, like in a spreadsheet.

Data can also be chopped in this method, for example using the expression x[10:20], which makes a dataset based on the 11th to 20th item in the x dataset (the ranges are Python syntax, and are zero-based). Negative indices count backwards from the end of the dataset.

Chapter 3. Command line interface

3.1. Introduction

An alternative way to control Veusz is via its command line interface. As Veusz is a a Python application it uses Python as its scripting language. Therefore you can freely mix Veusz and Python commands on the command line. Veusz can also read in Python scripts from files (see the Load command).

When commands are entered in the command prompt in the Veusz window, Veusz supports a simplified command syntax, where brackets following commands names, and commas, can replaced by spaces in Veusz commands (not Python commands). For example, **Add('graph', name='foo')**, may be entered as **Add 'graph' name='foo'**.

The **numpy** package is already imported into the command line interface (as "*"), so you do not need to import it first.

The command prompt supports history (use the up and down cursor keys to recall previous commands).

3.2. Commands

We list the allowed set of commands below

3.2.1. Action

Action('actionname', componentpath='.')

Initiates the specified action on the part (component) given the action name. Actions perform certain automated routines. These include "fit" on a fit part, and "zeroMargins" on grids.

3.2.2. Add

Add('parttype', name='nameforpart', autoadd=True, optionalargs)

The Add command adds a graph into the current part (See the To command to change the current position).

The first argument is the type of part to add. These include "graph", "page", "axis", "xy" and "grid". **name** is the name of the new part (if not given, it will be generated from the type of the part plus a number). The **autoadd** parameter if set, constructs the default sub-parts this part has (for example, axes in a graph).

Optionally, default values for the graph settings may be given, for example **Add('axis', name='y', direction='vertical')**.

Subsettings may be set by using double underscores, for example **Add('xy', MarkerFill__color='red', ErrorBarLine__hide=True)**.

Returns: Name of part added.

3.2.3. Close

Close()

Closes the plotwindow. This is only supported in embedded mode.

3.2.4. EnableToolbar

EnableToolbar(enable=True)

Enable/disable the zooming toolbar in the plotwindow. This command is only supported in embedded mode or from veusz_listen.

3.2.5. Export

Export(filename, color=True, page=0)

Export the page given to the filename given. The **filename** must end with the correct extension to get the right sort of output file. Currrenly supported extensions are '.eps', '.svg' and '.png'. If **color** is True, then the output is in colour, else greyscale. **page** is the page number of the document to export (starting from 0 for the first page!).

3.2.6. Get

Get('settingpath')

Returns: The value of the setting given by the path.

```
>>> Get('/page1/graph1/x/min')
'Auto'
```

3.2.7. GetChildren

GetChildren(where='.')

Returns: The names of the parts which are children of the path given

3.2.8. GetClick

GetClick()

Waits for the user to click on a graph and returns the position of the click on appropriate axes. Command only works in embedded mode.

Returns: A list containing tuples of the form (axispath, val) for each axis for which the click was in range. The value is the value on the axis for the click.

3.2.9. GetData

GetData(name)

Returns: A tuple containing the datasets with the name given. The tuple is (data, symerr, negerr, poserr), with each a numpy array of the same size or None. data are the values of the dataset, symerr are the symmetric errors (if set), negerr and poserr and negative and positive asymmetric errors (if set).

```
data = GetData('x')
SetData('x', data[0]*0.1, *data[1:])
```

3.2.10. GetDatasets

GetDatasets()

Returns: The names of the datasets in the current document.

3.2.11. GPL

GPL()

Print out the GNU Public Licence, which Veusz is licenced under.

3.2.12. ImportFile

ImportFile('filename', 'descriptor' [, linked=False])

Imports data from a file. The arguments are the filename to load data from and the descriptor.

The format of the descriptor is a list of variable names representing the columns of the data. For more information see Descriptors.

If the linked parameter is set to True, if the document is saved, the data imported will not be saved with the document, but will be reread from the filename given the next time the document is opened. The linked parameter is optional.

Returns: A tuple containing a list of the imported datasets and the number of conversions which failed for a dataset.

Changed in version 0.5: A tuple is returned rather than just the number of imported variables.

3.2.13. ImportFile2D

ImportFile2D('filename', datasets, xrange=(a,b), yrange=(c,d), invertrows=True/False, invertcols=True/False, transpose=True/False, linked=True/False)

Imports two-dimensional data from a file. The required arguments are the filename to load data from and the dataset name, or a list of names to use.

filename is a string which contains the filename to use. datasets is either a string (for a single dataset), or a list of strings (for multiple datasets).

The xrange parameter is a tuple which contains the range of the X-axis along the two-dimensional dataset, for example (-1., 1.) represents an inclusive range of -1 to 1. The yrange parameter specifies the range of the Y-axis similarly. If they are not specified, (0, N) is the default, where N is the number of datapoints along a particular axis.

invertrows and invertcols if set to True, invert the rows and columns respectively after they are read by Veusz. transpose swaps the rows and columns.

If the linked parameter is True, then the datasets are linked to the imported file, and are not saved within a saved document.

The file format this command accepts is a two-dimensional matrix of numbers, with the columns separated by spaces or tabs, and the rows separated by new lines. The X-coordinate is taken to be in the direction of the columns. Comments are supported (use "#", "!" or "%"), as are continuation characters ("\"). Separate datasets are deliminated by using blank lines.

In addition to the matrix of numbers, the various optional parameters this command takes can also be specified in the data file. These commands should be given on separate lines before the matrix of numbers. They are:

1. xrange A B
2. yrange C D
3. invertrows
4. invertcols
5. transpose

3.2.14. ImportFileCSV

ImportFileCSV('filename', readrows=False, prefix=None, linked=False)

This command imports data from a CSV format file. Data are read from the file using the dataset names given at the top of the files in columns. Please see the reading data section of this manual for more information. prefix is prepended to each dataset name, and linked specifies whether the data will be linked.

3.2.15. ImportFITSFile

ImportFITSFile(datasetname, filename, hdu, datacol='A', symerrcol='B', poserrcol='C', negerrcol='D', linked=True/False)

This command does a simple import from a FITS file. The FITS format is used within the astronomical community to transport binary data. For a more powerful FITS interface, you can use PyFITS within your scripts.

The datasetname is the name of the dataset to import, the filename is the name of the FITS file to import from. The hdu parameter specifies the HDU to import data from (numerical or a name).

If the HDU specified is a primary HDU or image extension, then a two-dimensional dataset is loaded from the file. The optional parameters (other than linked) are ignored. Any WCS information within the HDU are used to provide a suitable xrange and yrange.

If the HDU is a table, then the datacol parameter must be specified (and optionally symerrcol, poserrcol and negerrcol). The dataset is read in from the named column in the table. Any errors are read in from the other specified columns.

If linked is True, then the dataset is not saved with a saved document, but is reread from the data file each time the document is loaded.

3.2.16. ImportString

ImportString('descriptor', 'data')

Like, ImportFile, but loads the data from the specfied string rather than a file. This allows data to be easily embedded within a document. The data string is usually a multi-line Python string.

Returns: A tuple containing a list of the imported datasets and the number of conversions which failed for a dataset.

Changed in version 0.5: A tuple is returned rather than just the number of imported variables.

```
ImportString('x y', '''
1   2
2   5
3   10
''')
```

3.2.17. ImportString2D

ImportString2D(datasets, string)

Imports a two-dimensional dataset from the string given. This is similar to the ImportFile2D command, with the same dataset format within the string. This command, however, does not currently take any optional parameters. The various controlling parameters can be set within the string. See the ImportFile2D section for details.

3.2.18. List

List(where='.')

List the parts which are contained within the part with the path given, the type of parts, and a brief description.

3.2.19. Load

Load('filename.vsz')

Loads the veusz script file given. The script file can be any Python code. The code is executed using the Veusz interpreter.

Note: this command is only supported at the command line and not in a script. Scripts may use the python **execfile** function instead.

3.2.20. ReloadData

ReloadData()

Reload any datasets which have been linked to files.

Returns: A tuple containing a list of the imported datasets and the number of conversions which failed for a dataset.

3.2.21. Rename

Remove('partpath', 'newname')

Rename the part at the path given to a new name. This command does not move parts. See To for a description of the path syntax. '.' can be used to select the current part.

3.2.22. Remove

Remove('partpath')

Remove the part selected using the path. See To for a description of the path syntax.

3.2.23. Save

Save('filename.vsz')

Save the current document under the filename given.

3.2.24. Set

Set('settingpath', val)

Set the setting given by the path to the value given. If the type of **val** is incorrect, an **InvalidType** exception is thrown. The path to the setting is the optional path to the part the setting is contained within, an optional subsetting specifier, and the setting itself.

```
Set('page1/graph1/x/min', -10.)
```

3.2.25. SetData

SetData(name, val, symerr=None, negerr=None, poserr=None)

Set the dataset name with the values given. If None is given for an item, it will be left blank. val is the actual data, symerr are the symmetric errors, negerr and poserr and the getative and positive asymmetric errors. The data can be given as lists or numpys.

3.2.26. SetData2D

SetData2D('name', val, xrange=(A,B), yrange=(C,D))

Creates a two-dimensional dataset with the name given. val is either a two-dimensional numpy array, or is a list of lists, with each list in the list representing a row.

xrange and yrange are optional tuples giving the inclusive range of the X and Y coordinates of the data.

3.2.27. SetData2DExpressionXYZ

SetData2DExpressionXYZ('name', 'xexpr', 'yexpr', 'zexpr', linked=False)

Create a 2D dataset based on three 1D expressions. The x, y expressions need to evaluate to a grid of x, y points, with the z expression as the 2D value at that point. Currently only linear fixed grids are supported. This function is intended to convert calculations or measurements at fixed points into a 2D dataset easily. Missing values are filled with NaN.

3.2.28. SetData2DXYFunc(self, 'name', xstep, ystep, 'expr', linked=False)

Construct a 2D dataset using a mathematical expression of "x" and "y". The x values are specified as (min, max, step) in xstep as a tuple, the y values similarly. If linked remains as False, then a real 2D dataset is created, where values can be modified and the data are stored in the saved file.

3.2.29. SetVerbose

SetVerbose(v=True)

If **v** is **True**, then extra information is printed out by commands.

3.2.30. To

To('partpath')

The To command takes a path to a part and moves to that part. For example, this may be "/", the root part, "graph1", "/page1/graph1/x", "../x". The syntax is designed to mimic Unix paths for files. "/" represents the base part (where the pages reside), and ".." represents the part next up the tree.

3.2.31. Quit

Quit()

Quits Veusz. This is only supported in veusz_listen.

3.2.32. Zoom

Zoom(factor)

Sets the plot zoomfactor. This is only supported in embedded mode or veusz_listen.

3.3. Security

With the 1.0 release of Veusz, input scripts and expressions are checked for possible security risks. Only a limited subset of Python functionality is allowed, or a dialog box is opened allowing the user to cancel the operation. Specifically you cannot import modules, get attributes of Python objects, access globals() or locals() or do any sort of file reading or manipulation. Basically anything which might break in Veusz or modify a system is not supported. In addition internal Veusz functions which can modify a system are also warned against, specifically Print(), Save() and Export().

If you are running your own scripts and do not want to be bothered by these dialogs, you can run veusz with the **--unsafe-mode** option.

Chapter 4. Using Veusz from other programs

4.1. Non-Qt Python programs

Veusz supports being embedded within other Python programs. The calling program can open up any number of plot windows, and manipulate the graphs using the Veusz scripting commands, which are exposed as methods of graph objects.

Thanks to some pretty nifty programming (if I say so myself), the Python program can ignore the fact that Veusz is a Qt application. The embedding program continues to run when a graph is shown, can can take further user input. In addition Veusz can be used from a standard Python command line. This is achieved by Veusz and PyQt running in a separate thread. Commands are transparently passed from the Python program to the Veusz thread, and any results or exceptions are passed back (and rethrown in the case of exceptions).

Veusz must be installed in the PYTHONPATH for embedding to work. This can be done with the **setup.py** distutils script. An example embedding program is in **examples/embedexample.py**.

An example Python program embedding Veusz is below:

```
import time
import numpy
import veusz.embed as veusz

g = veusz.Embedded('new window title')
g.To( g.Add('page') )
g.To( g.Add('graph') )
g.SetData('x', numpy.arange(20))
g.SetData('y', numpy.arange(20)**2)
g.Add('xy')
g.Zoom(0.5)

# wait 20 seconds
time.sleep(20)

win2 = veusz.Embedded('second window example')
win2.To( win2.Add('page') )
win2.To( win2.Add('graph') )
win2.Add('function', function='x**2')
win2.Set('x/label', 'An example axis \\emph{label}')

time.sleep(20)

g.Close()
```

The supported commands are the same as in Commands, with the addition of a **Zoom(zoomfactor)** command to control the size of the plot in the window, the **EnableToolbar()** to enable to graph toolbar, and the **Close()** which closes the window given.

Embedding works on the Python versions I have access to, but may be affected by threading bugs. In addition Veusz must be installed in an appropriate place, so that it is in the PYTHONPATH.

4.2. PyQt programs

There is no support for PyQt programs embedding Veusz. However this will be easy to do, and so I will support this setup if requested or for a near future release.

4.3. Non Python programs

Support for non Python programs is available in a limited form. External programs may execute the **veusz_listen** executable or **veusz_listen.py** Python module. Veusz will read its input from the standard input, and write output to standard output. This is a full Python execution environment, and supports all the scripting commands mentioned in Commands, a **Quit()** command, the **EnableToolbar()** and the **Zoom(factor)** command listed above. Only one window is supported at once, but many **veusz_listen** programs may be started.

veusz_listen may be used from the shell command line by doing something like:

```
veusz_listen < in.vsz
```

where **in.vsz** contains:

```
To(Add('page') )
To(Add('graph') )
SetData('x', arange(20))
SetData('y', arange(20)**2)
Add('xy')
Zoom(0.5)
Export("foo.eps")
Quit()
```

A program may interface with Veusz in this way by using the **popen** C Unix function, which allows a program to be started having control of its standard input and output. Veusz can then be controlled by writing commands to an input pipe.

4.4. C, C++ and Fortran

A callable library interface to Veusz is on my todo list for C, C++ and Fortran. Please tell me if you would be interested in this option.