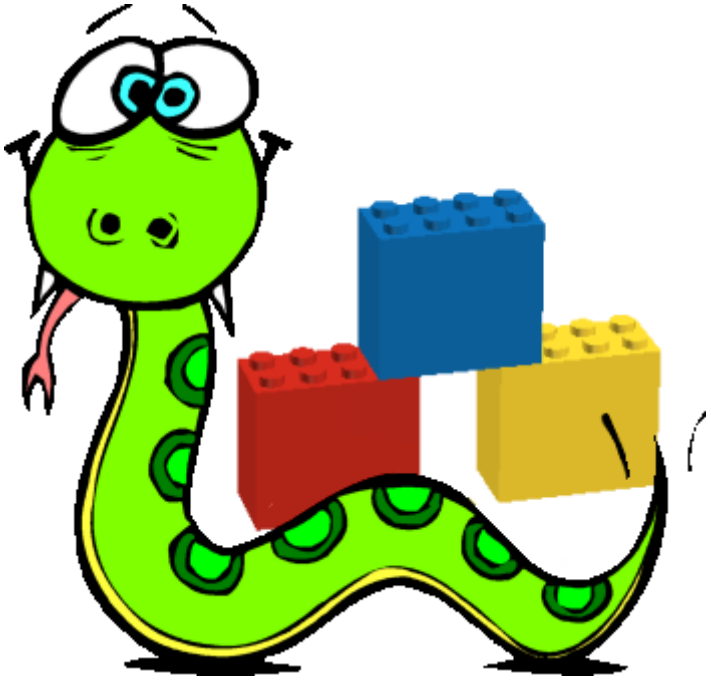
PyLab_Works

documentation alpha-release V0.3
june 2009, at the moment this is just a copy of the website
for installation notes: always see the website : http://pic.flappie.nl



June 2009
Stef Mientki

Contents

210 GUI support

216 Bind - Skip Events

219 ToDo

Introduction  (February 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

What is PyLab_Works

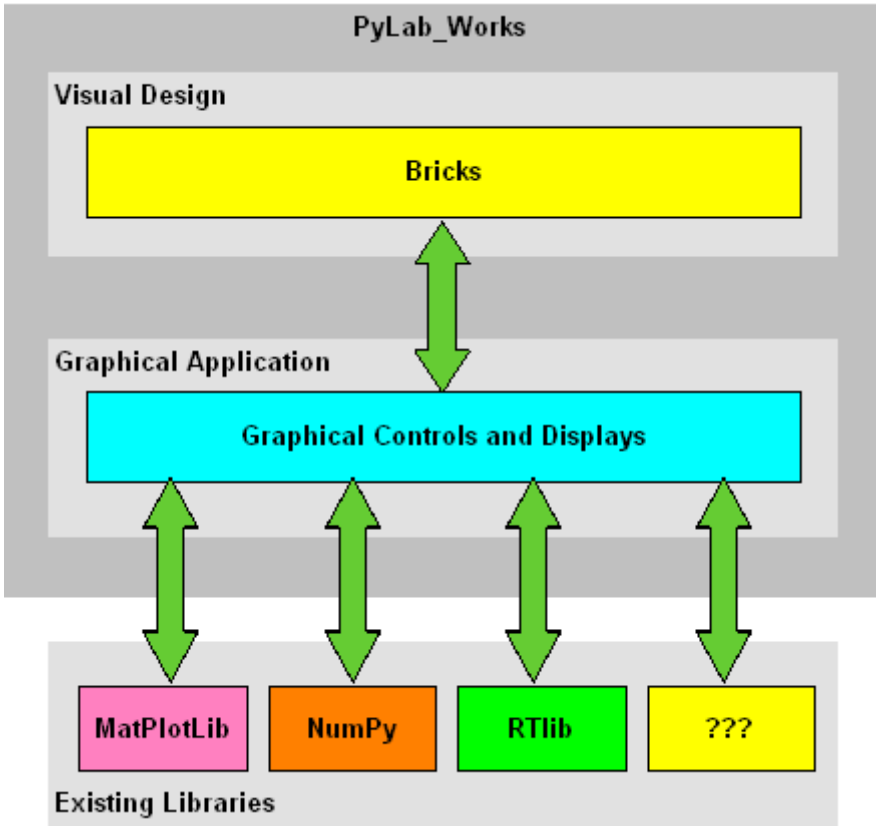
What is PyLab_Works, we can give a few short answers from some different perspectives:
Technical speaking:

PyLab_Works is a highly modular Visual Development Environment, based on data-flow programming technics.

Or in more human language:

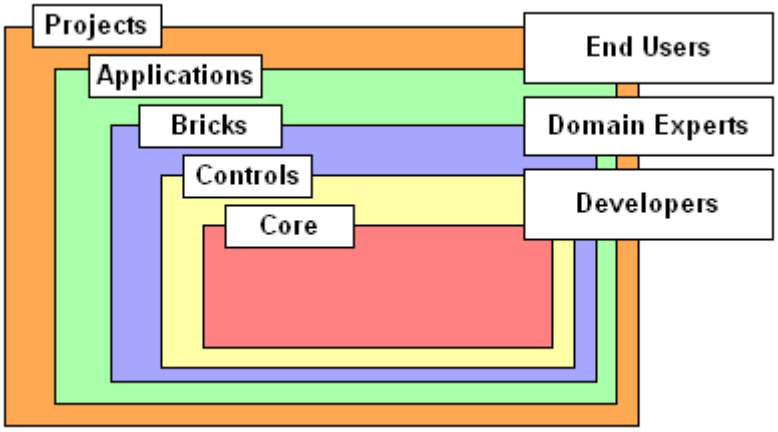
PyLab_Works is an easy development environment, based on the domain knowledge of experts. So the user doesn't need any programming knowledge nor does the user need to be an expert on that particular domain.

And we can give many more long answers, but it might be better to start with a simple example. The figure below shows a simple functional block diagram of PyLab_Works, showing that PyLab_Works is in fact nothing more than a user-friendly wrapper for well-established libraries, like NumPy and MatPlotLib. The Graphical Controls and Displays are an integral part of PyLab_Works, while the domain expert should only create new Bricks, which are just simple connectors between the Graphical Controls and the domain functionality already available in the libraries.



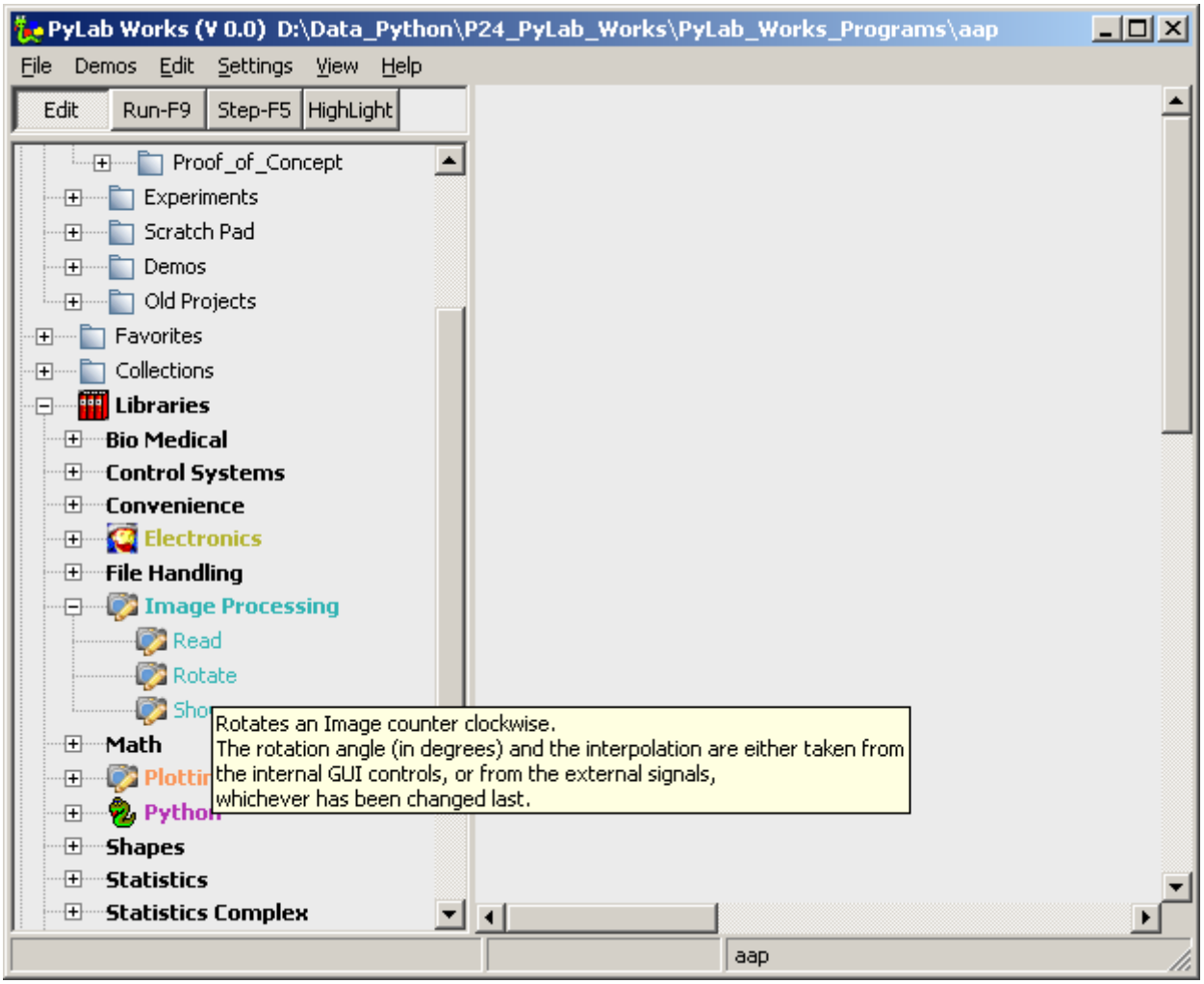
Documentation

The documentation is written during code writing, and therefor might not always be completely correct, or sometimes there are features in the documentation that are intended to implement but are not yet implemented. The code is quit hierarchical, as shown in the schematic below. At the top of each document, there's an indication for which group of users this document is written.

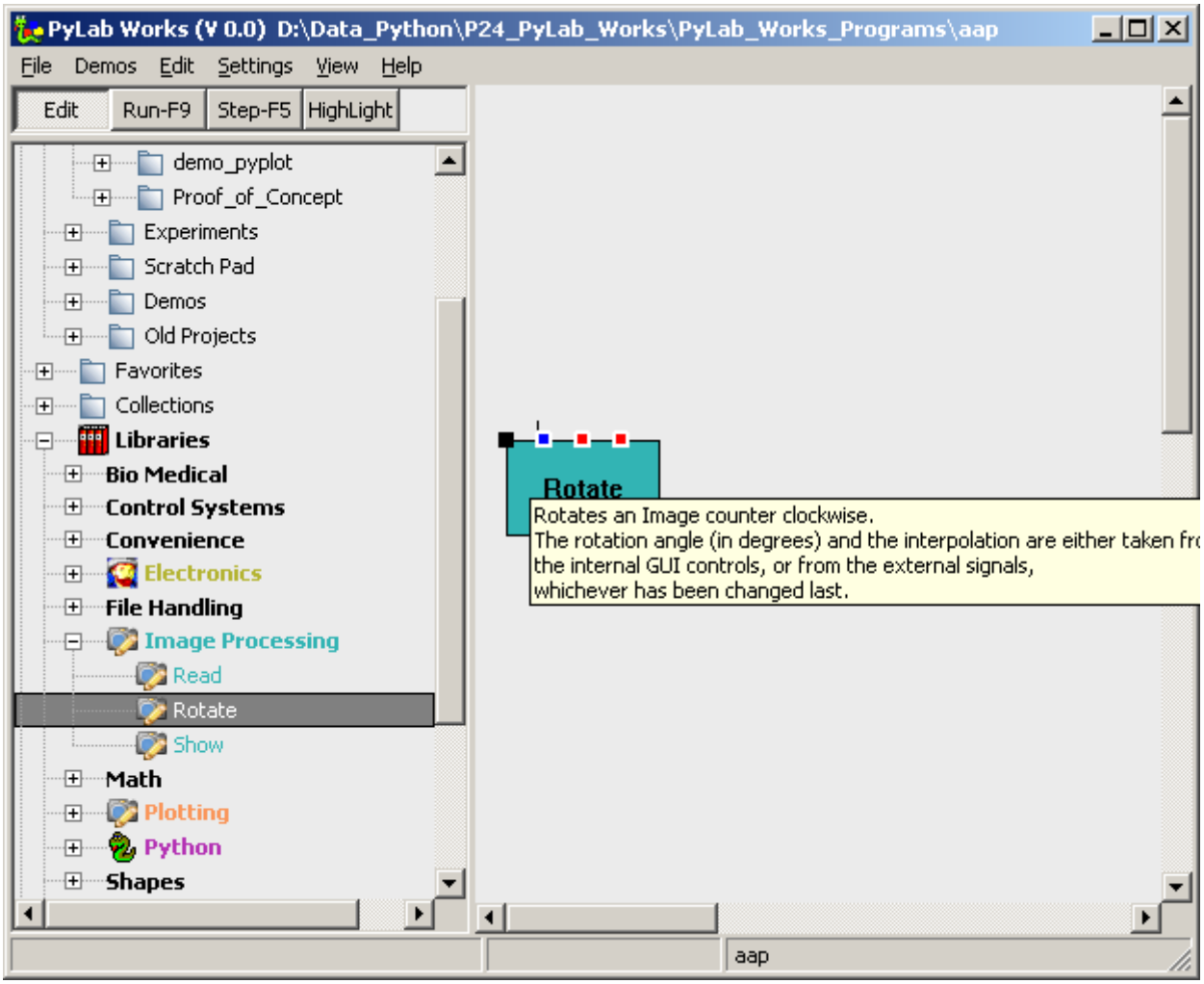


Simple Example

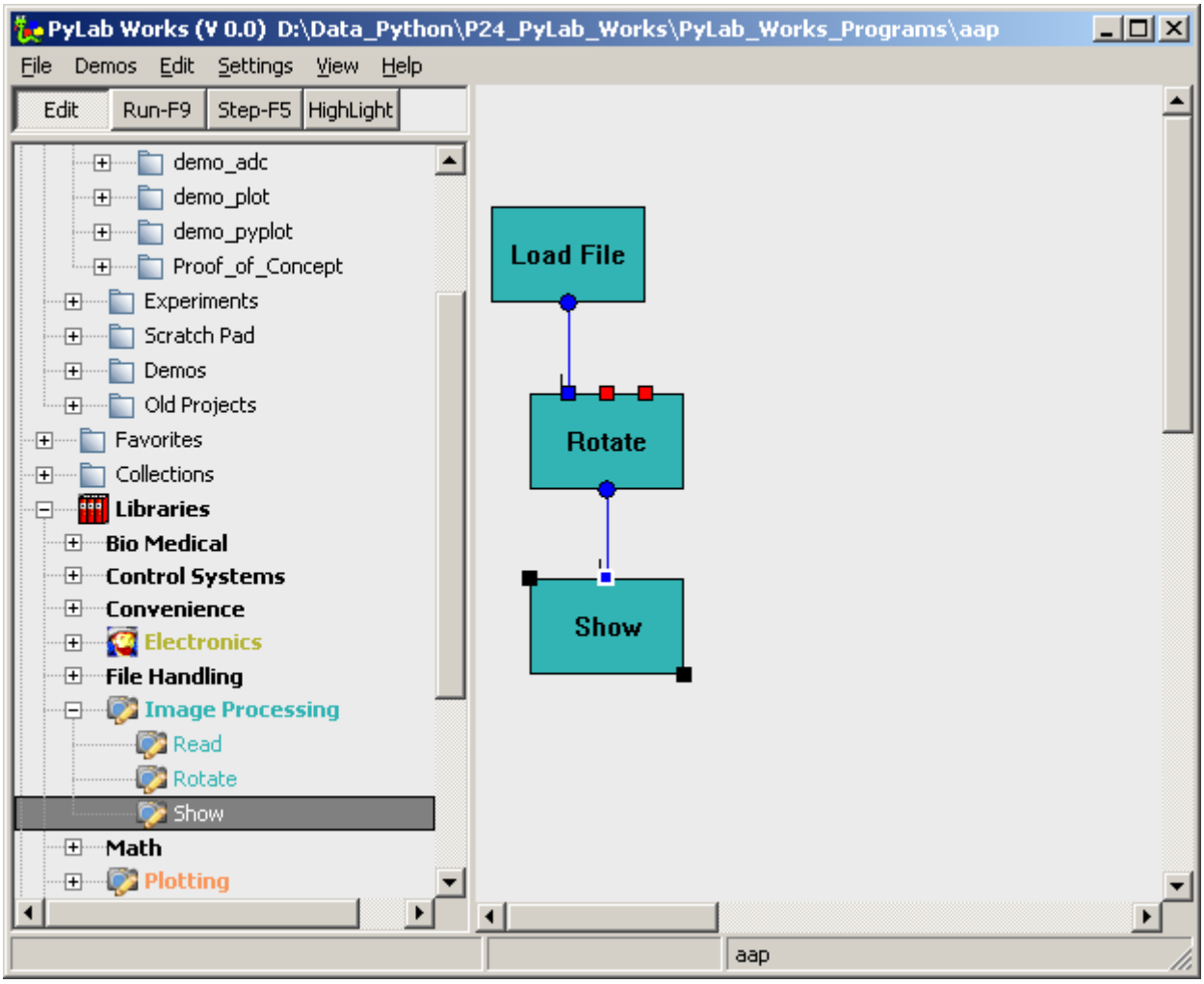
So suppose I want to make a program that can rotate an image, not very meaningful, because there are a lot of programs that can rotate an image, but I've chosen this example because it doesn't expect any specific domain knowledge. This is not a full explanation of PyLab_Works, but here I just want to show what steps are to be performed to make a real working application. So let's start PyLab_Works, and search for the image library in the tree. Open the node of the image library and look for something like "rotate". When we move the mouse cursor over the "rotate", we get an experts explanation of what this function will do for us. And indeed it does exactly what we need.



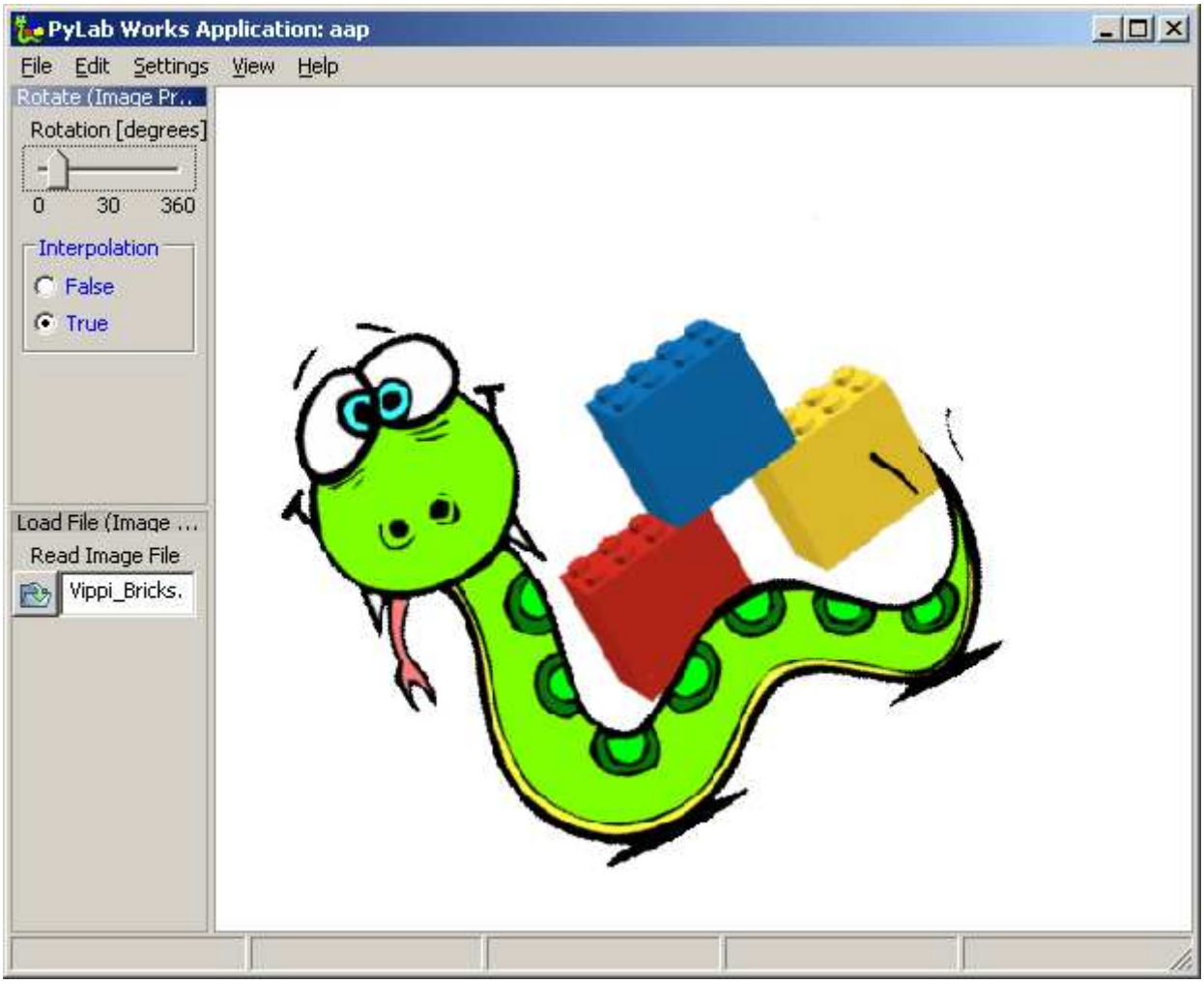
Now simply drag the rotate-node to the design area on the right and drop it there, and the first brick will appear. I can investigate the brick, by placing the mouse cursor on the brick, and again the domain expert will tell me (in my own language) what the brick will do. I can move my mouse cursor on the inputs and outputs (the tiny squares and circles) and again the domain expert will tell me what this input/output is supposed to do and in case of an input if it's required or not (which is also indicated by a small extra line at the required node).



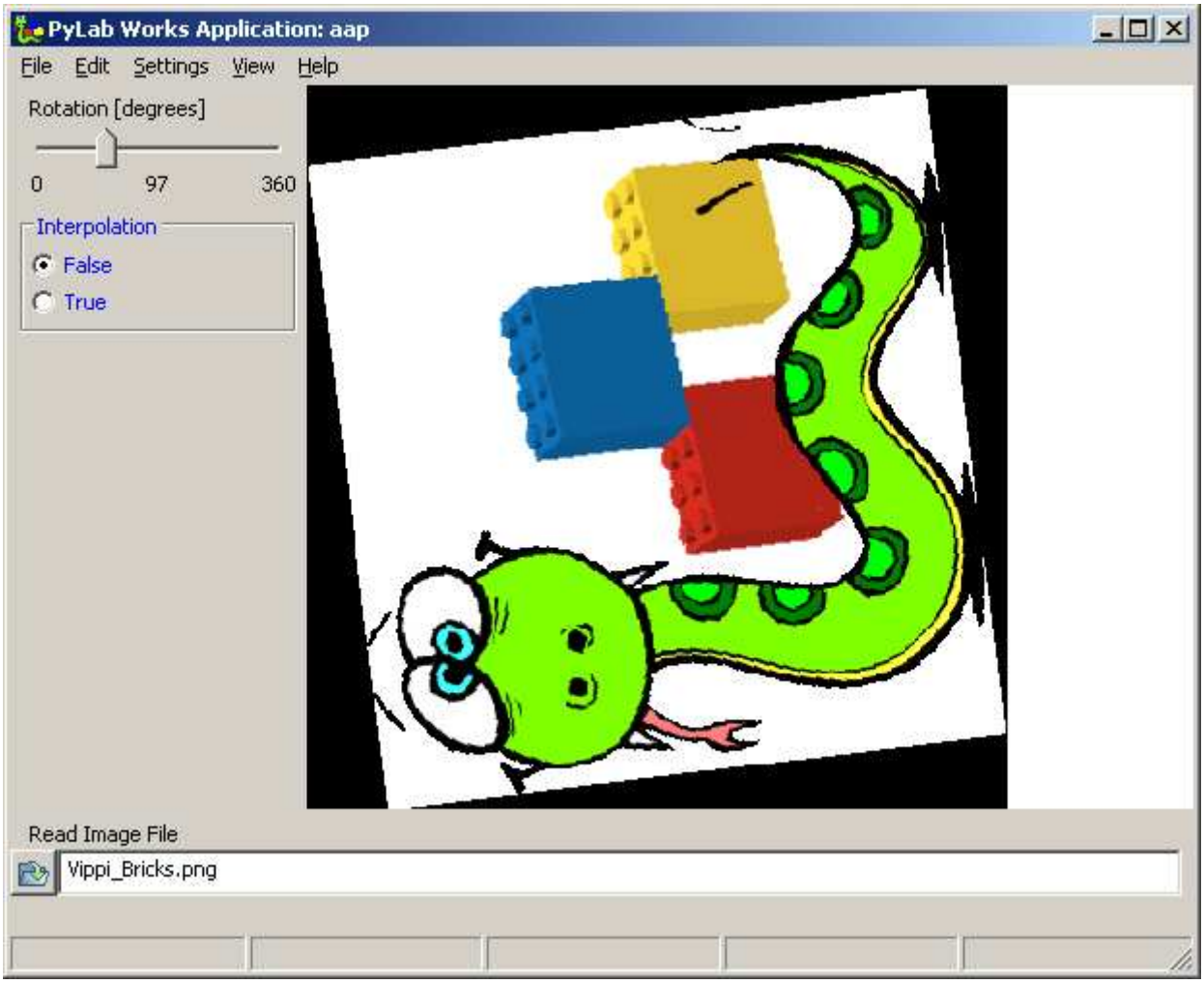
You can see that you need to connect some kind of "signal" to the top-left input, and the signal is of the type image, which seems quit reasonable for a function that rotates an image. The input on the top-right is not required, so I can leave that input open. And as the brick image-rotate only rotates an image, it might very convenient to connect some other brick to the output, something like a store-to-file or a display image. Let's look at the image library again, and we see an load-image-from-file and a display-image brick in the library. So drag these two to the design area.



And after you connect the devices (drag from one IO-port to another, only IO-ports that can accept the type of signal will highlight and are able to bind to the dragged wire), the **functional design** of the program is ready. Yes there might be some details to fill in, and yes we do need to design the desired graphical user interface. This functional design is a very important intermediate step, because it shows exactly what we mean at this abstraction level. Try to tell someone what the above application will do and compare that sentence with the diagram above. You might be missing the word "Image" in each of the above bricks, but that is implicitly indicated by the green color (which can be altered by the user of the library).
So up to the next step, press on the "Run-F9" button and the base of the general application is generated, watch ...



... and now you should have the real program, because changing the slider will change the image rotation directly. So in in fact **our Application** is ready, we just have to fine-tune some details.
As you might notice, each brick is translated in a separate panel with a number of controls. The panels are separated by dragable splitters. These controls allows the final user of our application to change every parameters of that brick. The panels can be dragged and dropped to the desired position and size. Placing a panel outside the main window, will remove the controls (not the brick, and not the functionality) from the final application, so only preventing the end-user to change the according parameters.
In the picture above you can see an another important feature of easy programming, in rotating an image new pixels will be generated, for which the color must be determined by some kind of calculation. The domain expert therefore has added a control "interpolation" that will let us choose from different possibilities (in better rotation algorithms there will even be more chooses). We even don't have to understand all this talking about interpolation, but as fully non-domain-expert we can simply try and see what it does, see the figure below.



Above is our final application, we've shuffled the panels to a bit more logical position and fixed the application (removed the drag possibility of the panels and splitters).

Basic Ideas

The basic idea of this program was born when I had to study the possibilities of using LabView as the new standard tool (instead of MatLab) for medical research on our university. After seeing some life demos, I became very enthusiast, which vanished completely when I started implementing some real stuff. Visual programming, like in LabView did have something very interesting, but I had the feeling that the implementation was somewhere completely wrong. It took several months, before I saw the light. After I knew what was important, I read a lot of articles about visual programming, but couldn't confirm all of my own ideas. There has been a lot of study on the use and benefit of visual programming environments, both by researchers with a background in psychology and researchers with a background in information theory. Many of them compared the speed of development in normal program languages and in visual program languages, and found an increased productivity (at least in the beginning), but I never saw a good comparison, i.e. a double blind setup and/or the right audience for the right tool. The explanations why visual design is better, varies a lot and is often in expressed in terms, not understandable by normal human beings. So after seeing the light, I just started developing PyLab_Works based on some common sense starting-points. The most important issues in my opinion are:

- Offering the necessary **user domain knowledge**, i.e. you don't need to be an expert (some basic knowledge should be enough) to perform actions in a certain domain. In my opinion this is really thé key item, and even far more important than the visual development environment. In the above example you don't need to know if its'"Rotate", "rotation", ..., you don't need to know that you might need an interpolation algorithm etc.
- **Flatness of information**, all information that belonging together should be exposed to the user at once. A wizard is, in general, an example of non-flatness of information. Many programs don't follow this essential rule, in fact this is one of my major objections against LabView.
- **Visual design**,

- **Simplicity / uniformity**,
- **User-flow-programming**, this looks much like data-flow-programming, but sometimes user-flow and data-flow will conflict with each other, in that case user-flow should always win.
- **Modularity**,
-

Key-Features

Advantages of PyLab_Works

- every detail is designed from the user's point of view (and not from the programmers point of view) and therefor very simple for non-programmers
- it's free and open source, so you can check or modify any underlying algorithm
- PyLab_Works doesn't reinvent the wheel, it's just a wrapper around already existing and well established libraries
-

Disadvantages of PyLab_Works

- there is no large installed base
- not all parts are completely finished
- a number of editors only accepts articles based on MatLab or LabView (completely sic in the world of research where every detail should be checkable / traceable)

MainMenu

File	Demos	Edit	Settings	Windows	Help
+ New / Open ^O - Save ^S - Save As ... ---------- - Print ^P - Print Preview - Page Setup ---------- - Export ---------- + Close	-dynamic list of all demo programs	-ToDo	-ToDo	-ToDo	- PyLab Works - Python ---------- - Check New Version + Send BugReport - Ask OnLine Assistance + About

New / Open
Closes the existing file and opens a new or existing file. If another project is opened, the current project will be saved automatically.

Close
Closes the application, all changes are saved automatically.

Send BugReport
Starts the default email client, with the recipient directed to "Punthoofd", the bug directory of PyLab_Works.

About (text aanpassen !!)
Shows the about box, containing a short description, version number and credits.

Similar Packages  (april 2009)

Application Designer / Domain Expert / Control Designer / Core Developer

For most similar packages, local screenshots have been put on the PyLab_Works site, so you can still get an impression if the links are changed.

Similar Packages (open source)

DeVIDE, a Python-based dataflow application builder that enables the rapid prototyping of medical visualization and image processing applications via visual programming.
local screenshot ProjectsDeVIDEGallery - TU Delft Visualisation Website

MathGUIde, mathGUIde ist speziell auf die Vorlesungen "Diskrete Mathematik für Informatiker" und "Lineare Algebra für Informatiker" zugeschnitten
local screenshot mathGUIde

Elefant , is an open source library for machine learning
local screenshot Elefant

Pyphant, Ein Python-Framework zur Modellierung von Datenanalyse-Prozessen.
local screenshot Pyphant — Freiburger Materialforschungszentrum (FMF)

SciLab, SciCos, Scicos is a Scilab toolbox, for dynamic system modeling and simulation.
local screenshot Scilab Home Page Scicos Home Page

Orange , is a component-based data mining software.
local screenshot Orange

Vision , Vision is a visual-programming environment
local screenshot Vision

Sage, Sage is a free open-source mathematics software system. It combines the power of many existing open-source packages into a common Python-based interface.
local screenshot Sage

Octave, GNU Octave is a high-level language, primarily intended for numerical computations.
local screenshot Octave

Python-xy, Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization.
local screenshot python(x,y) - Python for Scientists

OpenAlea, OpenAlea is an open source project primarily aimed at the plant research community, with a particular focus on Plant Architecture Modeling at different scales.
local screenshot OpenAlea

Veusz is a scientific plotting and graphing package written in Python.
local screenshot Veusz

Enthought with Traits UI, they are developing it for a customer, and planned to show us something in december 2007, but I guess it's somewhat delayed.

Similar Packages (commercial)

Mathematica, If you're doing anything technical, think Mathematica--not just for computation, but for modeling, simulation, visualization, development, documentation, and deployment.
local screenshot Wolfram Mathematica Home Page

Maple, Maple is the essential technical computing software for today's engineers, mathematicians, and

scientists.
local screenshot Math Software for Engineers, Educators & Students Maplesoft

OpenModelica, object oriented modeling and simulation.
(I've put this in the commercial section, because the better parts are commercial)
local screenshot OpenModelica Project

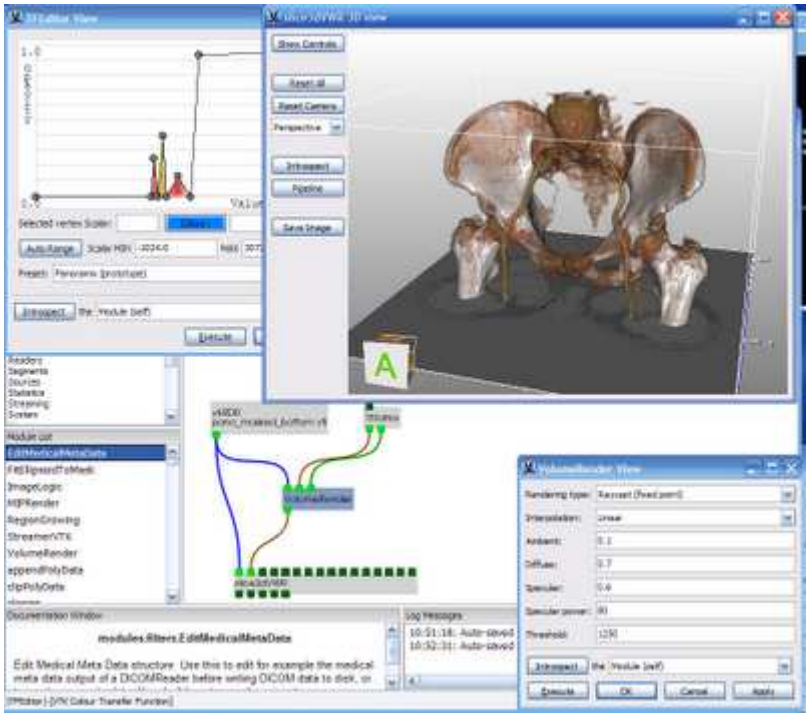
MatLab, MATLAB® is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.
local screenshot MATLAB

LabView, LabVIEW is a graphical programming environment used by millions of engineers and scientists to develop sophisticated measurement, test, and control systems using intuitive graphical icons and wires that resemble a flowchart.
local screenshot NI LabVIEW

DeVIDE

(april 2009)

A Python-based dataflow application builder that enables the rapid prototyping of medical visualization and image processing applications via visual programming.
ProjectsDeVIDEGallery - TU Delft Visualisation Website

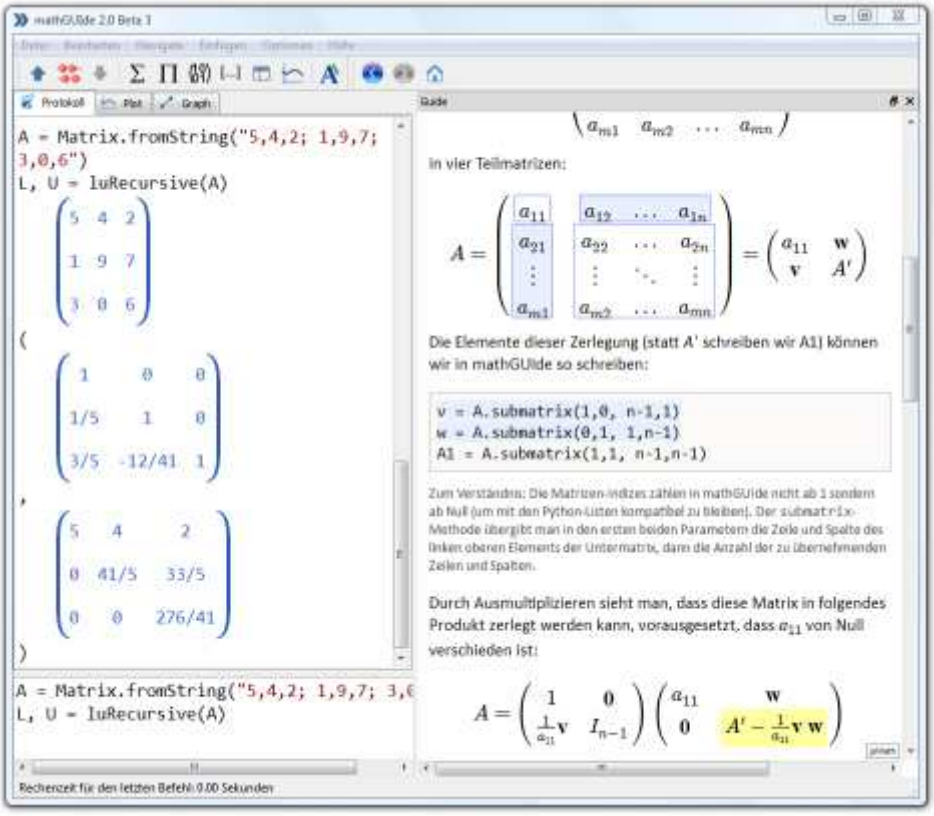


MathGUIde

(april 2009)

mathGUIde ist speziell auf die Vorlesungen "Diskrete Mathematik für Informatiker" und "Lineare Algebra für Informatiker" zugeschnitten.

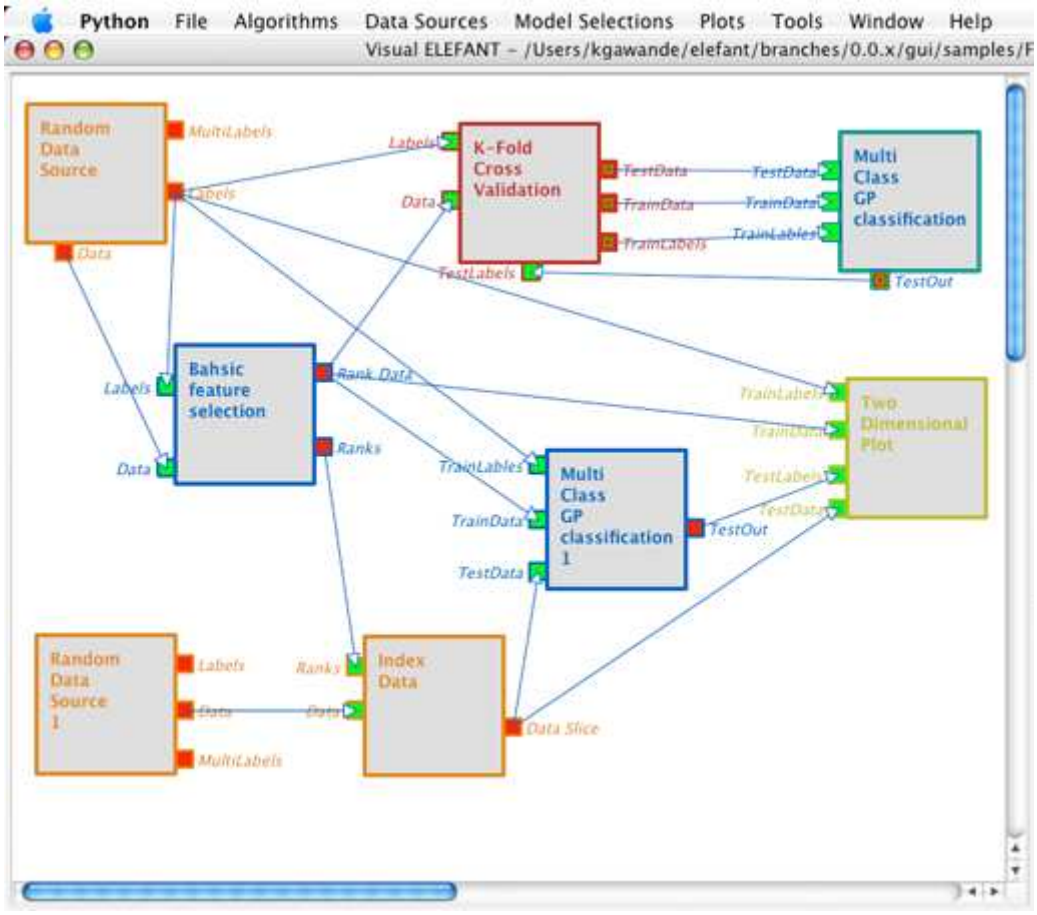
mathGUIde



Elefant

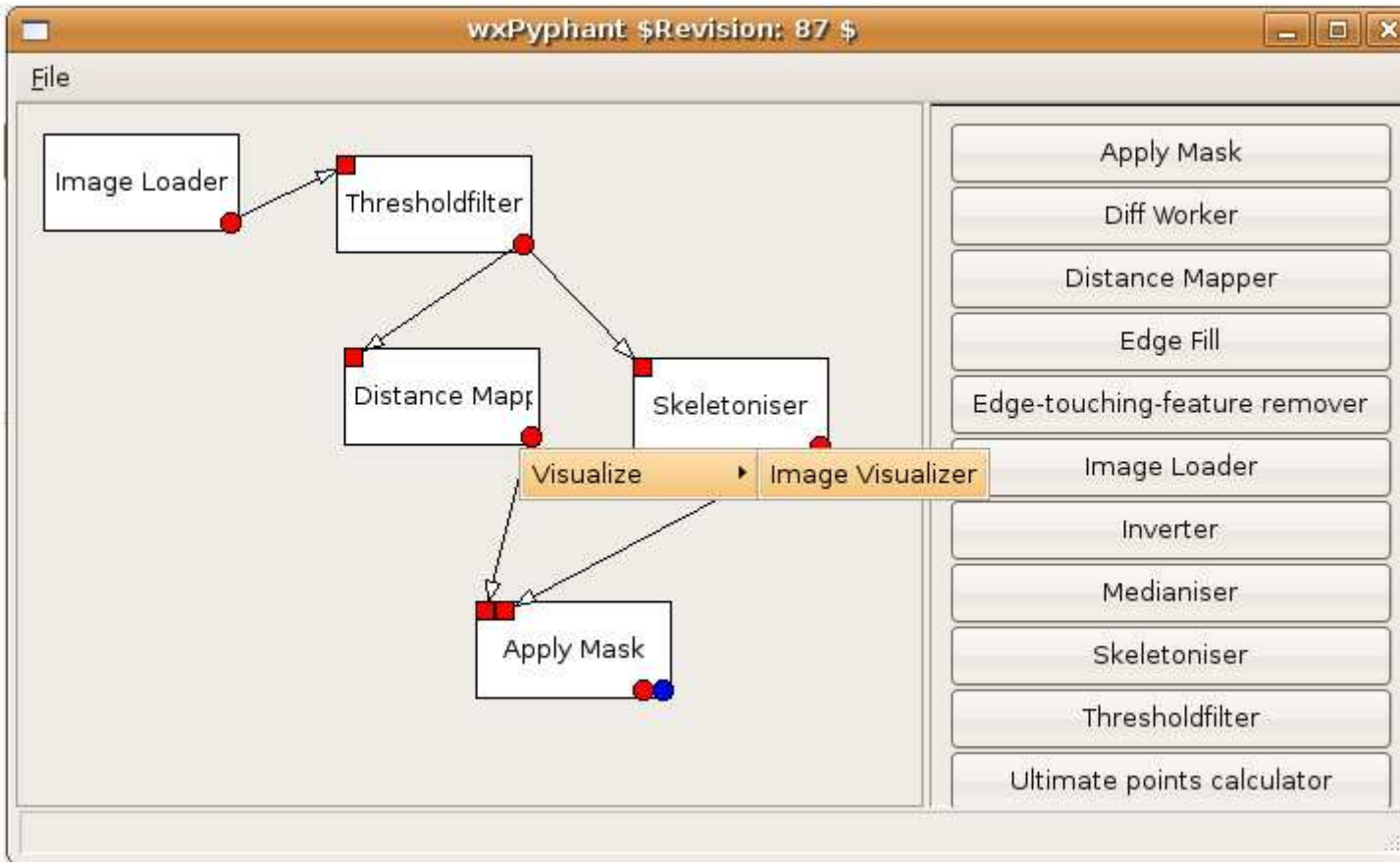
(april 2009)

An open source library for machine learning. Elefant



Pyphant  (april 2009)

Ein Python-Framework zur Modellierung von Datenanalyse-Prozessen.
Pyphant — Freiburger Materialforschungszentrum (FMF)

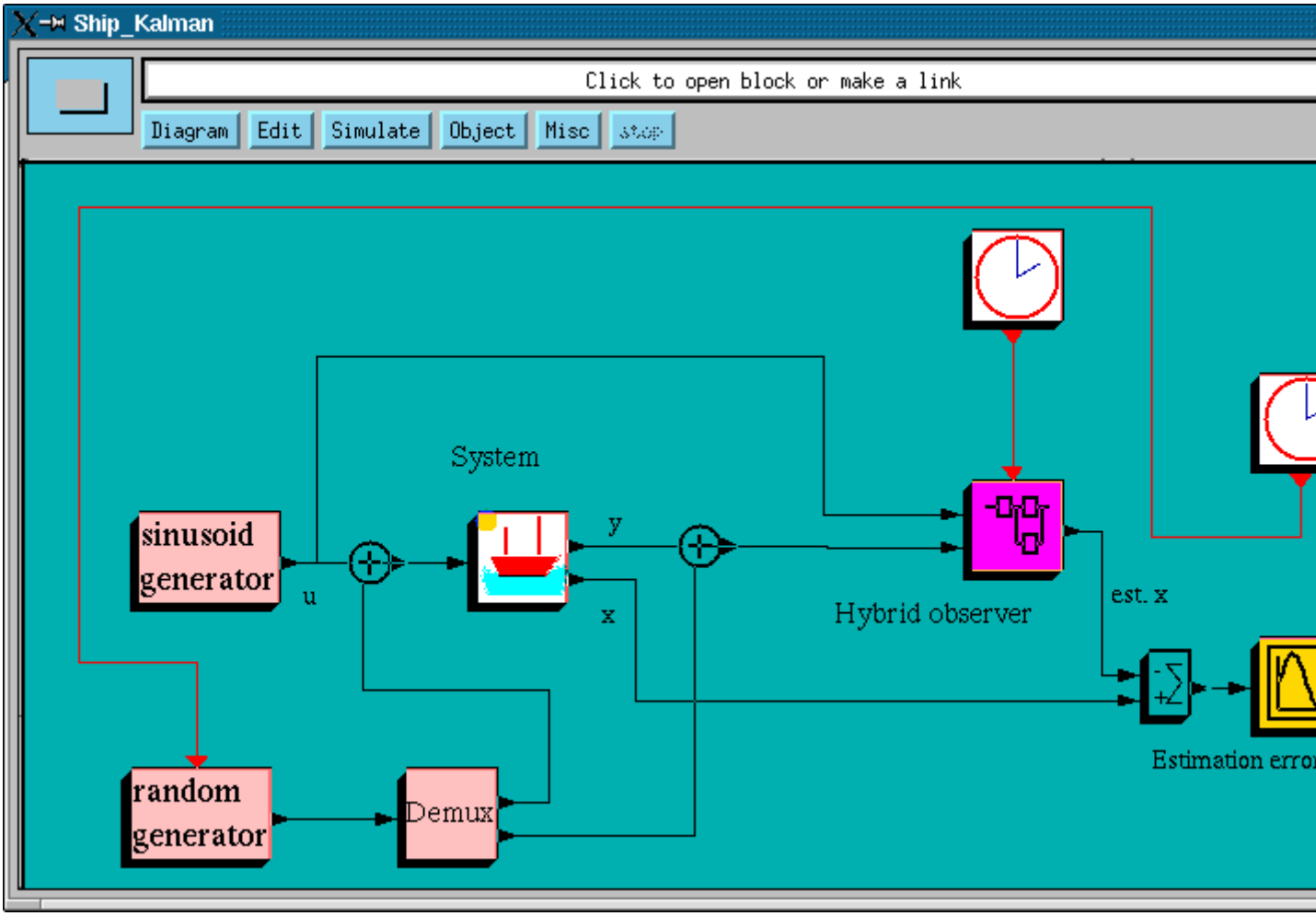


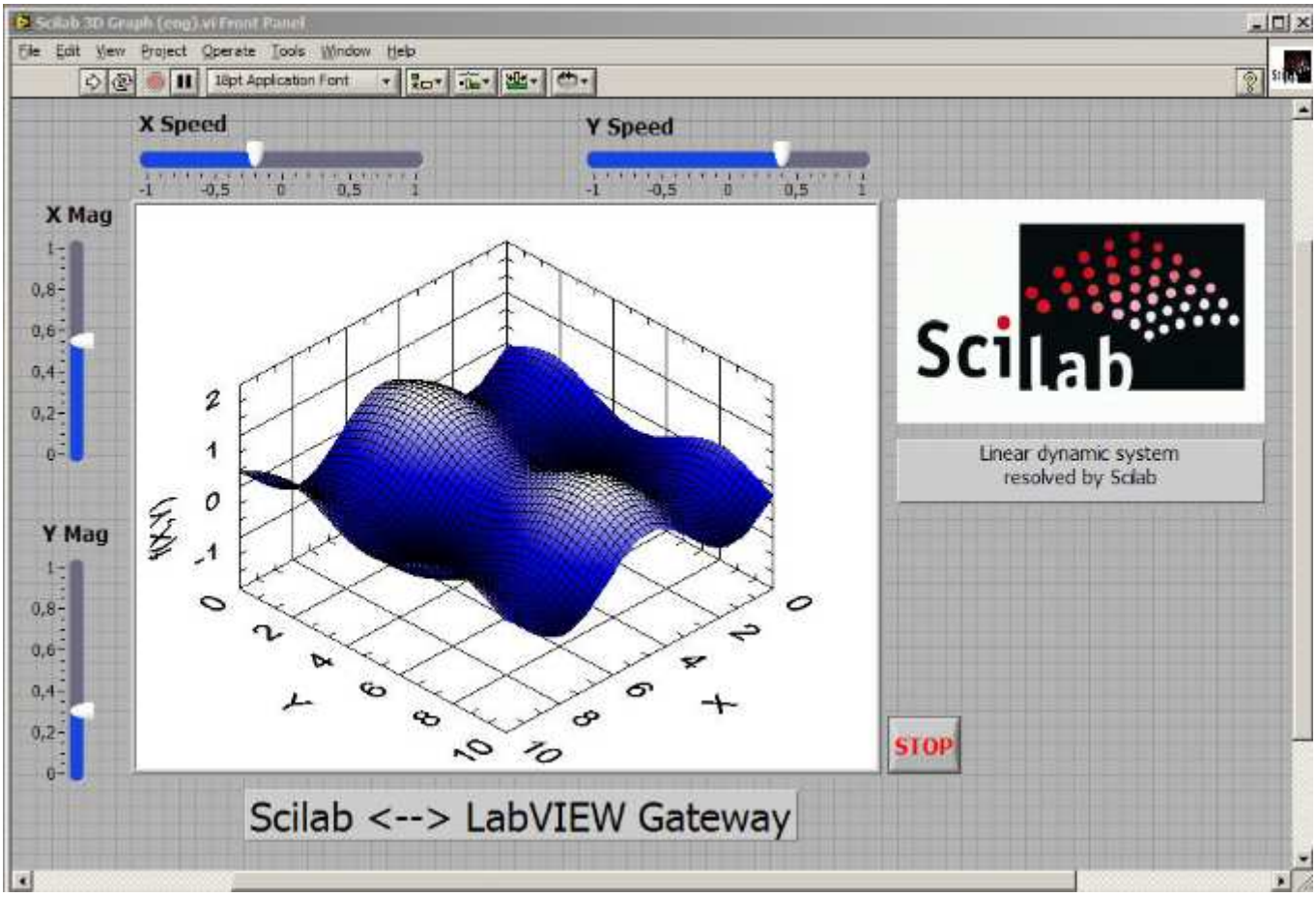
Scilab-Scicos

(april 2009)

Scicos is a Scilab toolbox, for dynamic system modeling and simulation.

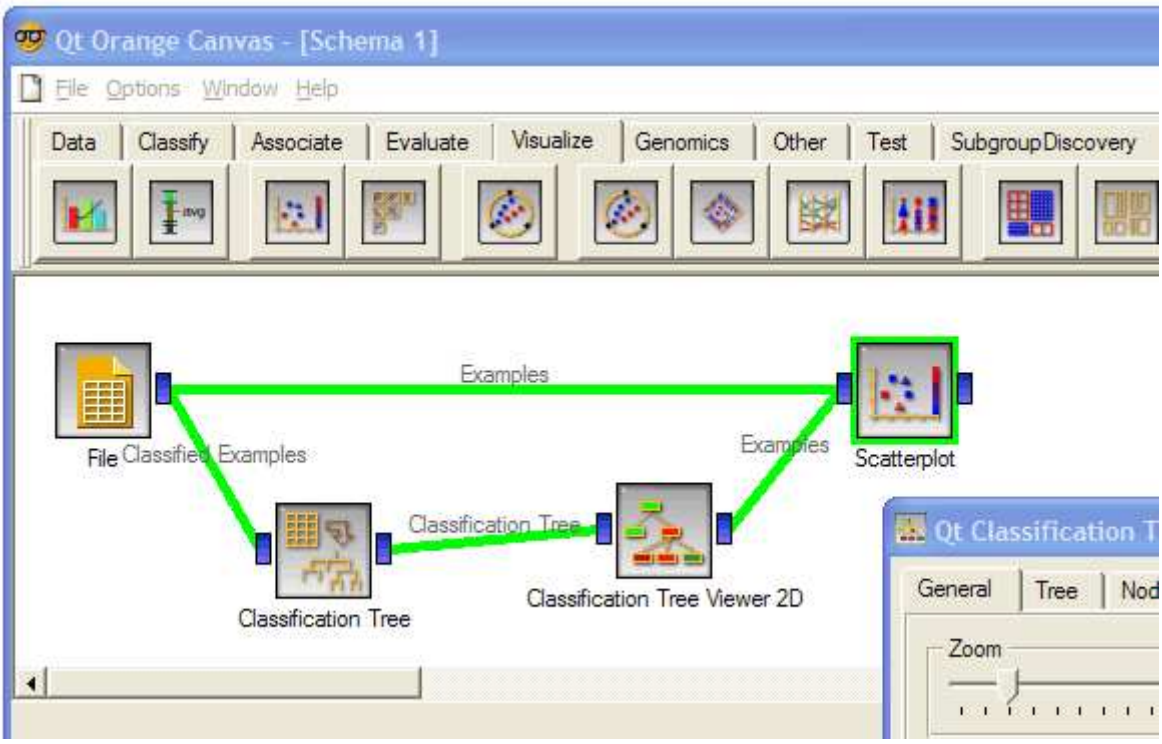
Scilab Home Page Scicos Home Page


Ship_Kalman
Click to open block or make a link
Diagram
Edit
Simulate
Object
Misc
System
sinusoid generator
u
y
x
Hybrid observer
est. x
random generator
Demux



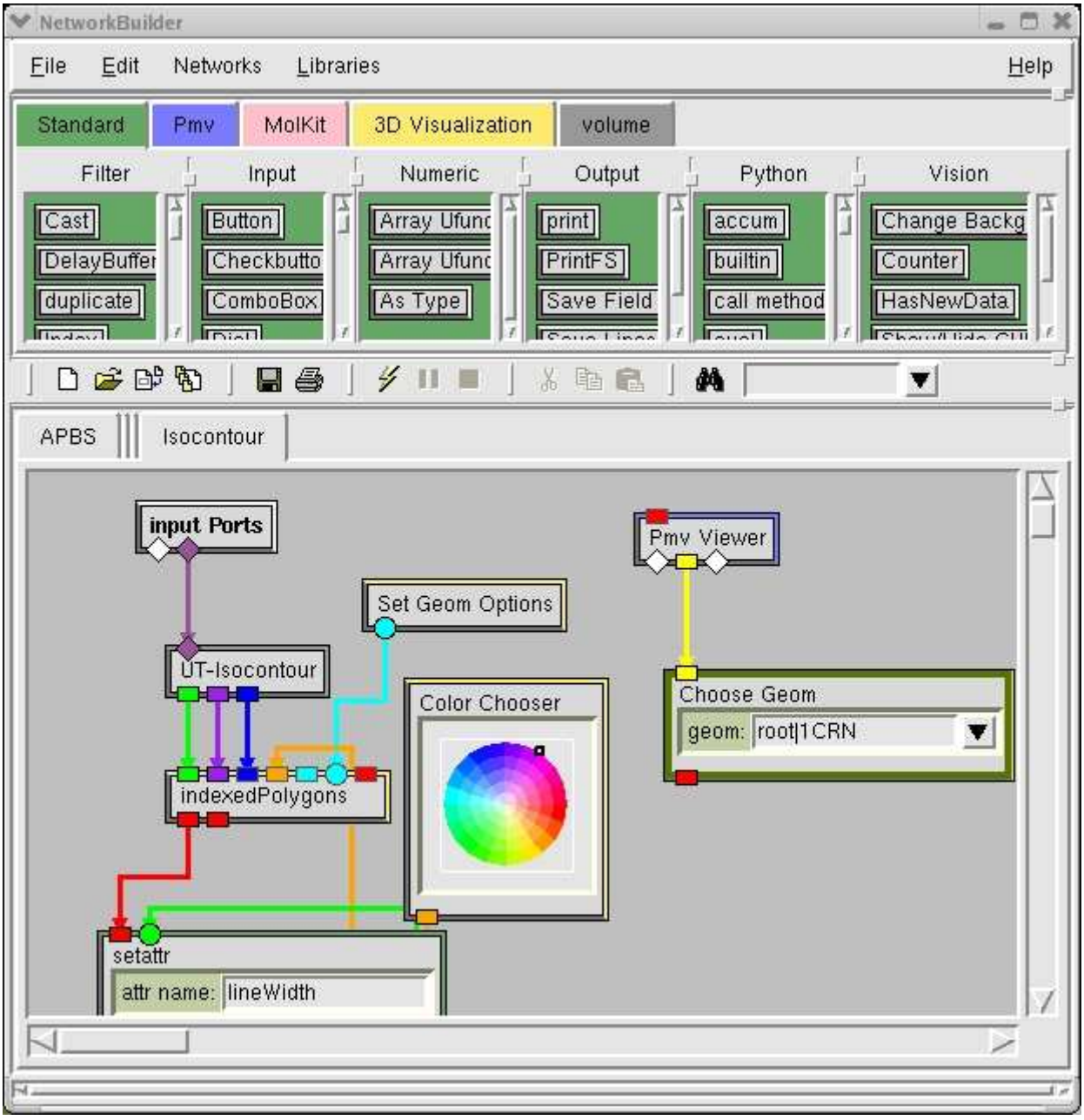
Orange  (april 2009)

is a component-based data mining software. Orange



Vision  (april 2009)

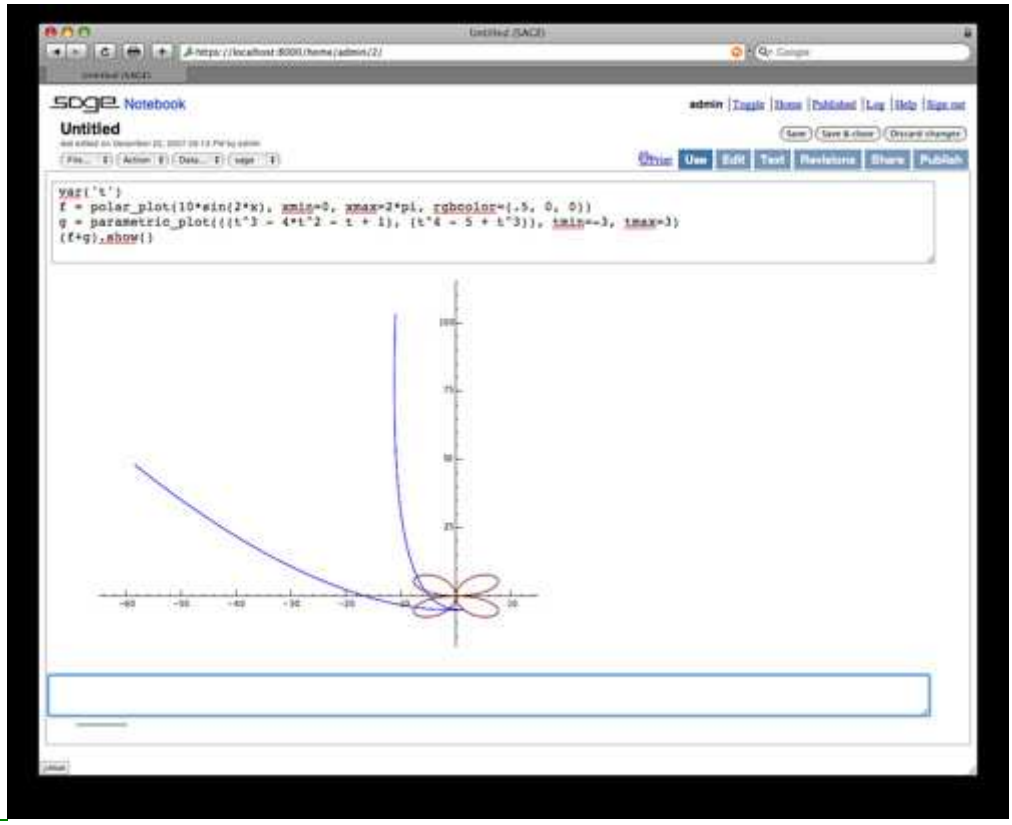
Vision is a visual-programming environment. Vision



Sage  (may 2009)

Sage is a free open-source mathematics software system licensed under the GPL. It combines the power of many existing open-source packages into a common Python-based interface.

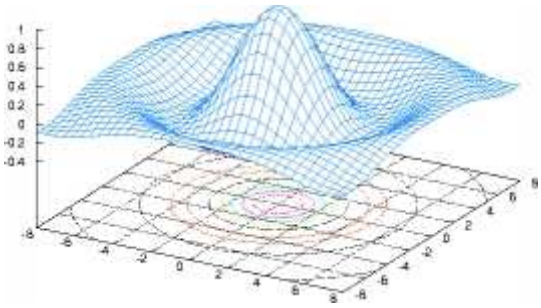
Sage Open Source Mathematics Software



Octave  (april 2009)

GNU Octave is a high-level language, primarily intended for numerical computations.

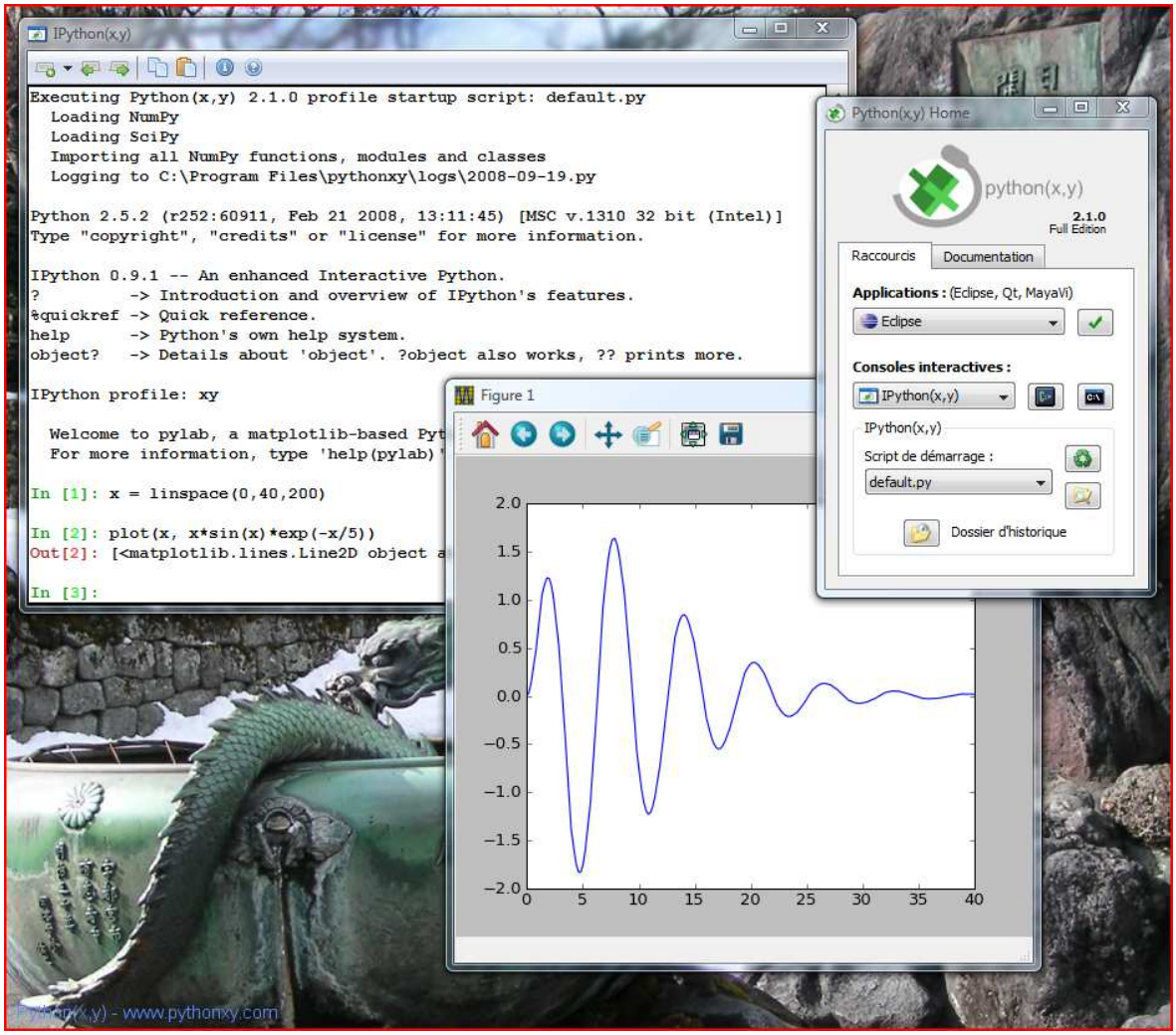
Octave



Python-xy  (april 2009)

Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces (and development framework) and Eclipse integrated development environment.

python(x,y) - Python for Scientists

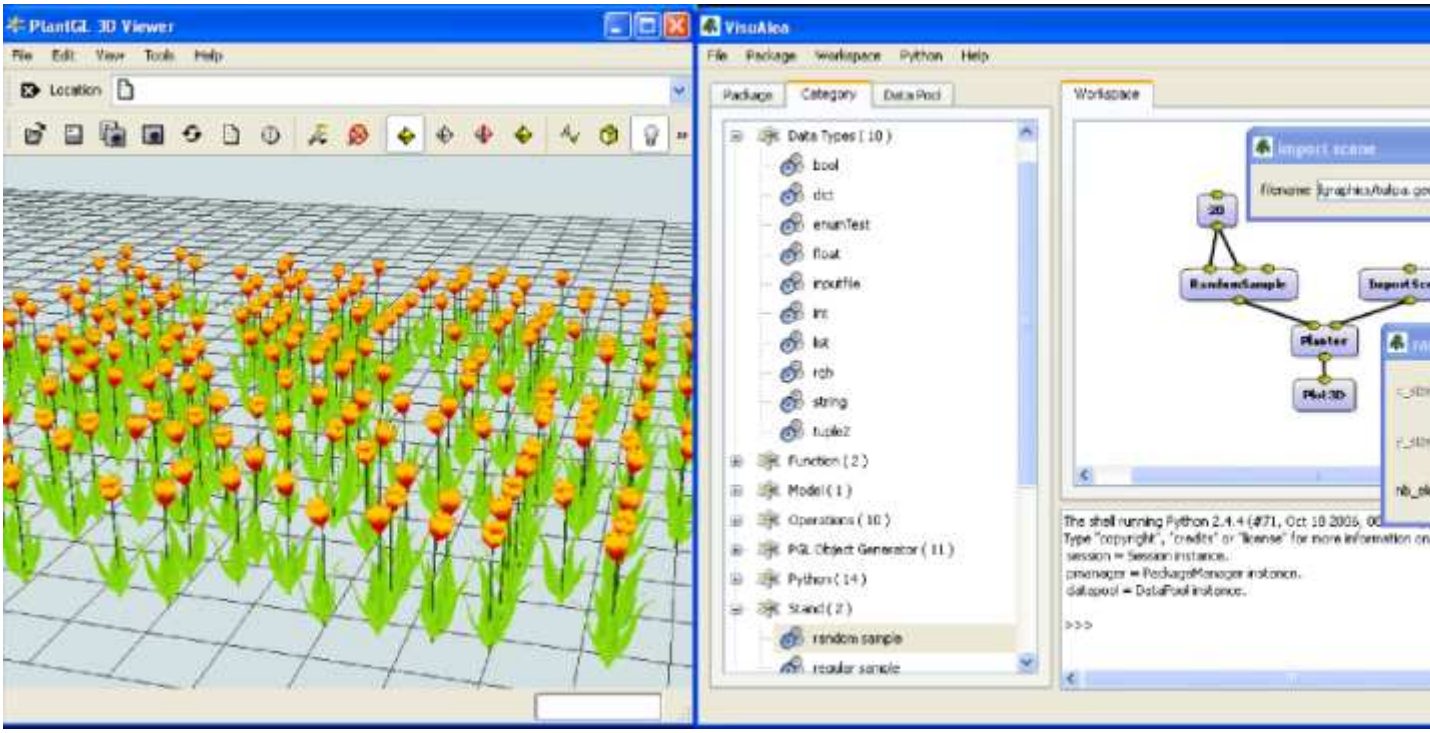


OpenAlea

(april 2009)

OpenAlea is an open source project primarily aimed at the plant research community, with a particular focus on Plant Architecture Modeling at different scales.

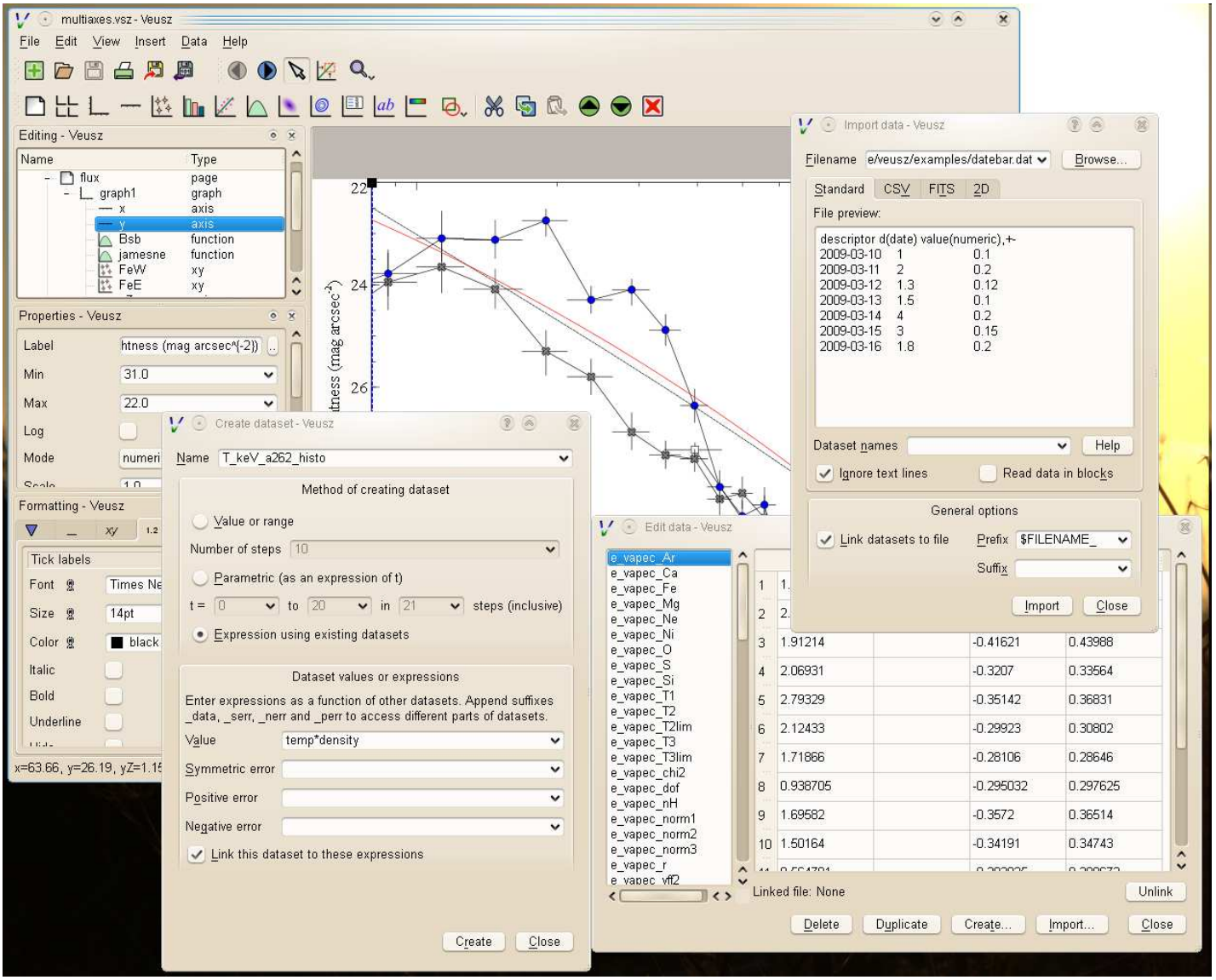
OpenAlea



Veusz  (april 2009)

Veusz is a scientific plotting and graphing package written in Python.

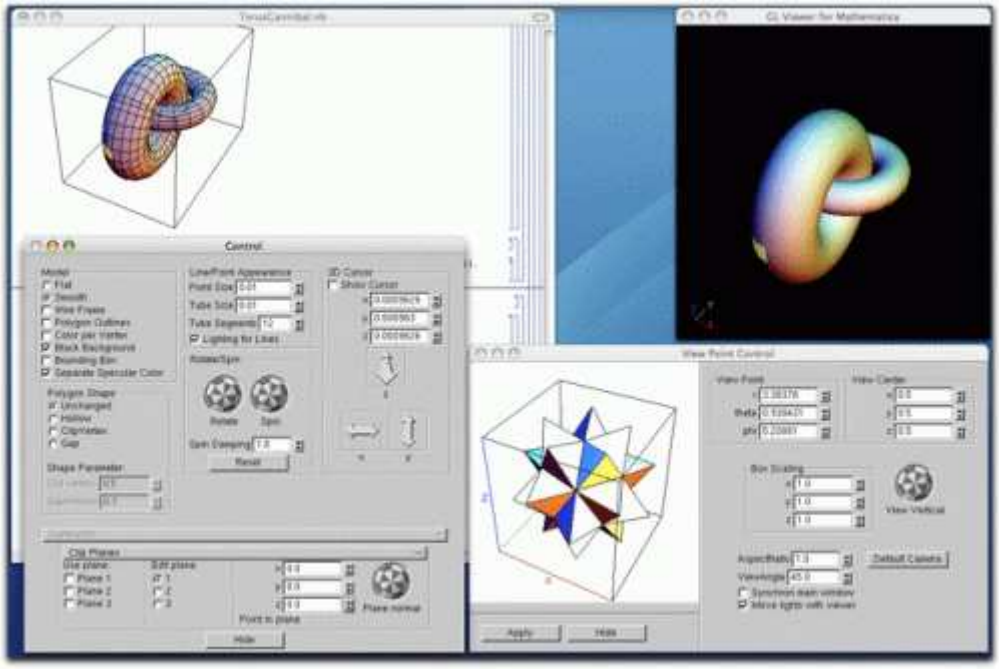
Veusz



Mathematica (april 2009)

If you're doing anything technical, think Mathematica--not just for computation, but for modeling, simulation, visualization, development, documentation, and deployment.

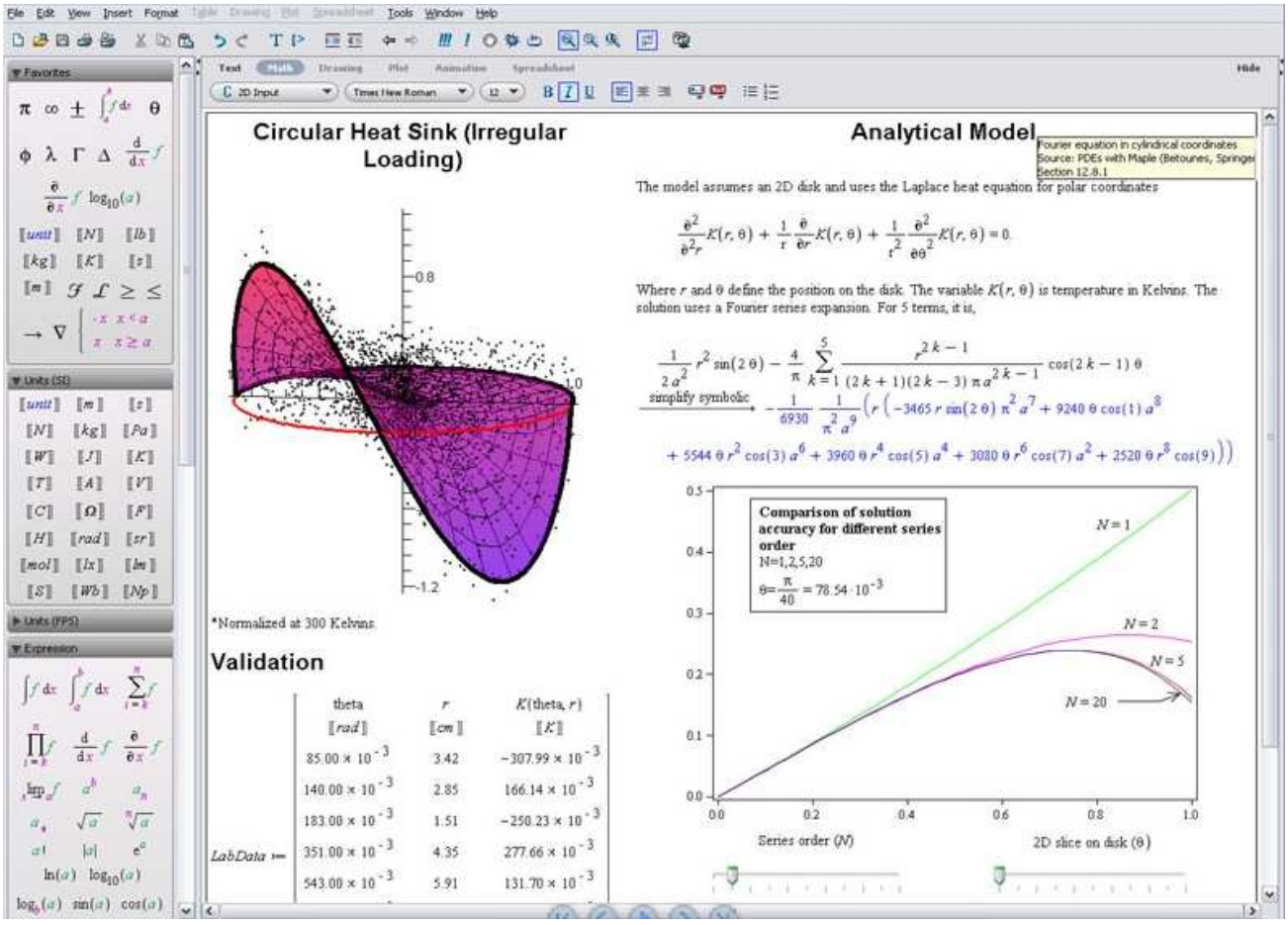
Wolfram Mathematica Home Page



Maple

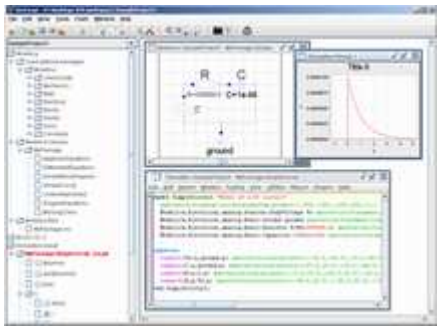
(april 2009)

Maple is the essential technical computing software for today's engineers, mathematicians, and scientists.
Math Software for Engineers, Educators & Students Maplesoft



OpenModelica (april 2009)

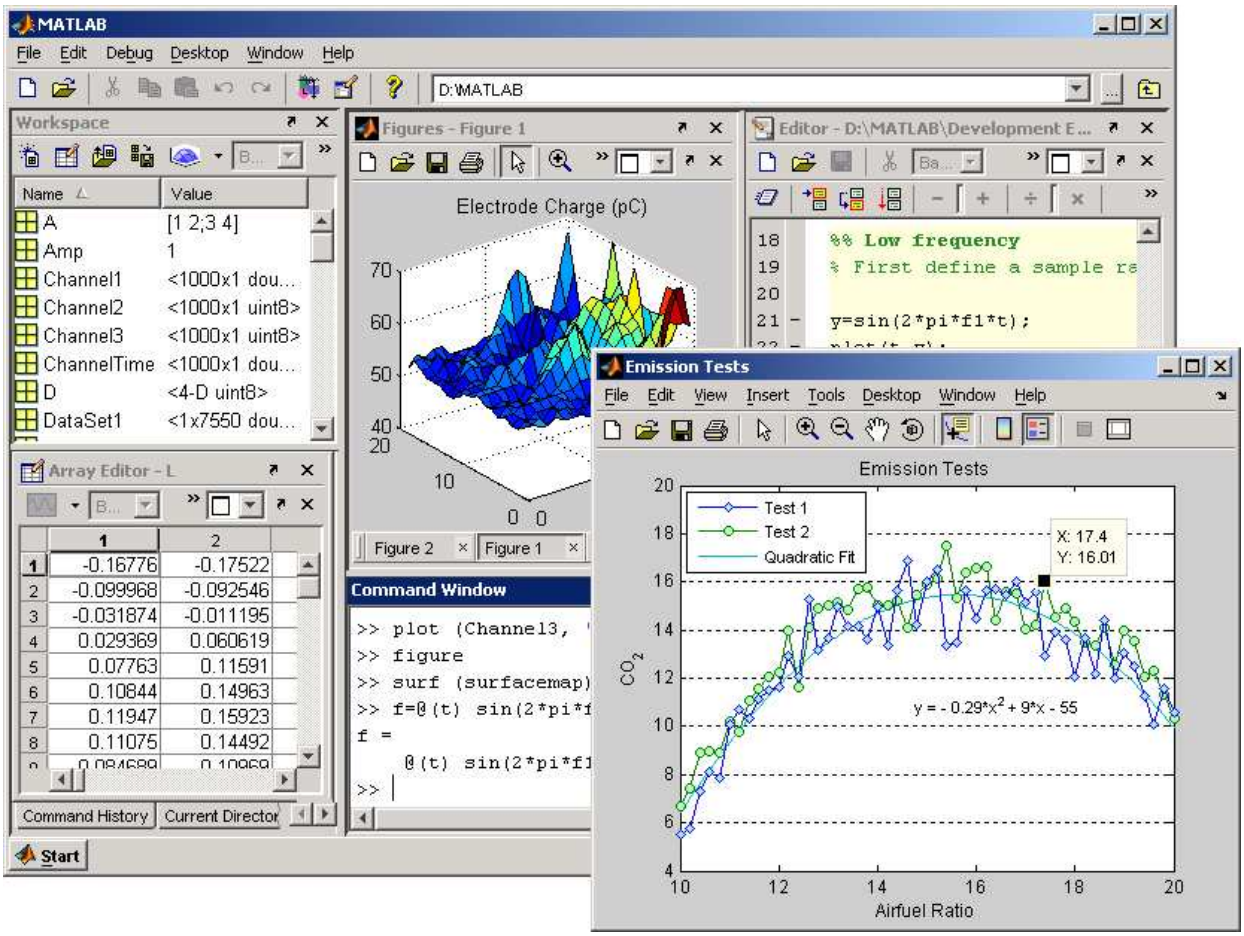
OpenModelica, object oriented modeling and simulation.
(I've put this in the commercial section, because the better parts are commercial)
OpenModelica Project



MatLab (april 2009)

MATLAB® is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.

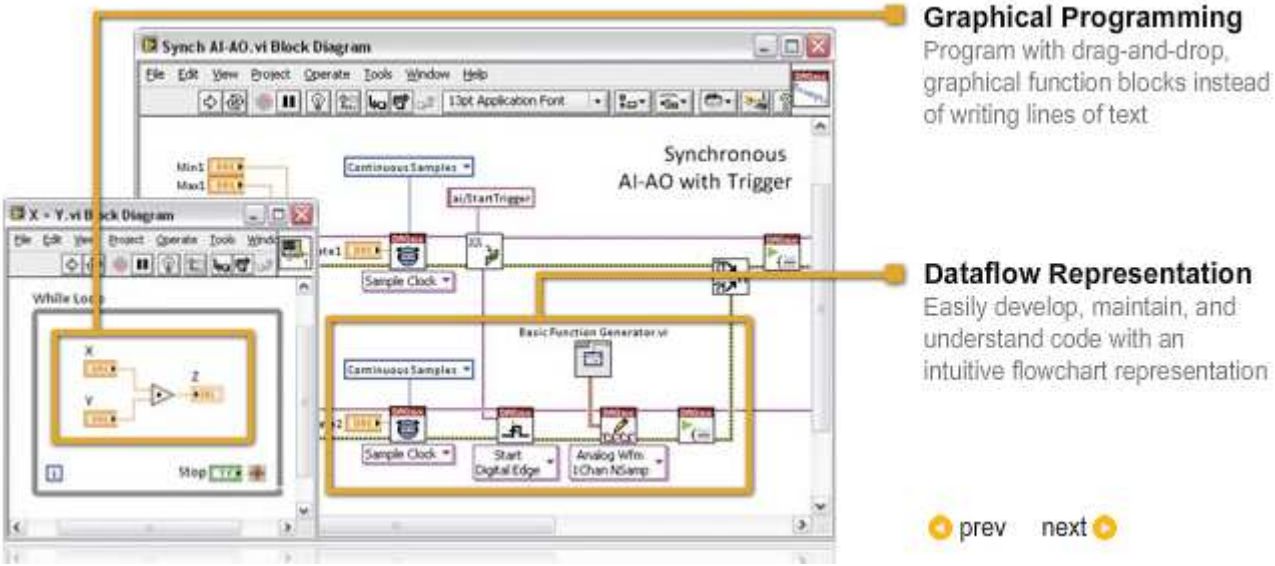
MATLAB - The Language Of Technical Computing



Labview (april 2009)

LabVIEW is a graphical programming environment used by millions of engineers and scientists to develop sophisticated measurement, test, and control systems using intuitive graphical icons and wires that resemble a flowchart.

NI LabVIEW - The Software That Powers Virtual Instrumentation - Products and Services - National Instruments



t enables the rapid

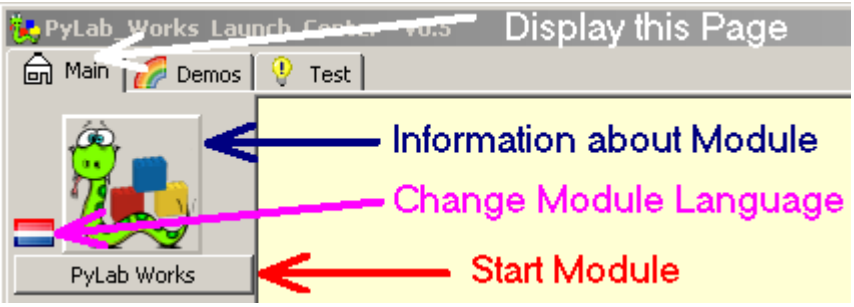
Launch Center (October 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

Getting Started

With the Launch Center all major applications within the PyLab_Works frame can be started. At this moment not all parts are fully functional, but at least they will give you a fair idea of the functionality of PyLab_Works. The Launch Center may contain some core developers part, because it's also used by the core designers to do their development.

Main: this is the major page where all programs can be launched. At this moment the Library manager contains a better IDE than the IDE itself, that's because I use the Launch Center myself as a coding and testing tool.
Demos: This page contains full working demos of PyLab_Works applications and it will gradual grow in time. All demo-projects can be run in application mode and in design mode, in the latter you can modify the design (the original demo can always be recovered)
Test: Don't dare to look at that page, it's mainly meant for core developers, but it might be also useful to track down installation problems on different Operating Systems.
With the Launch Center all major applications within the PyLab_Works frame can be started



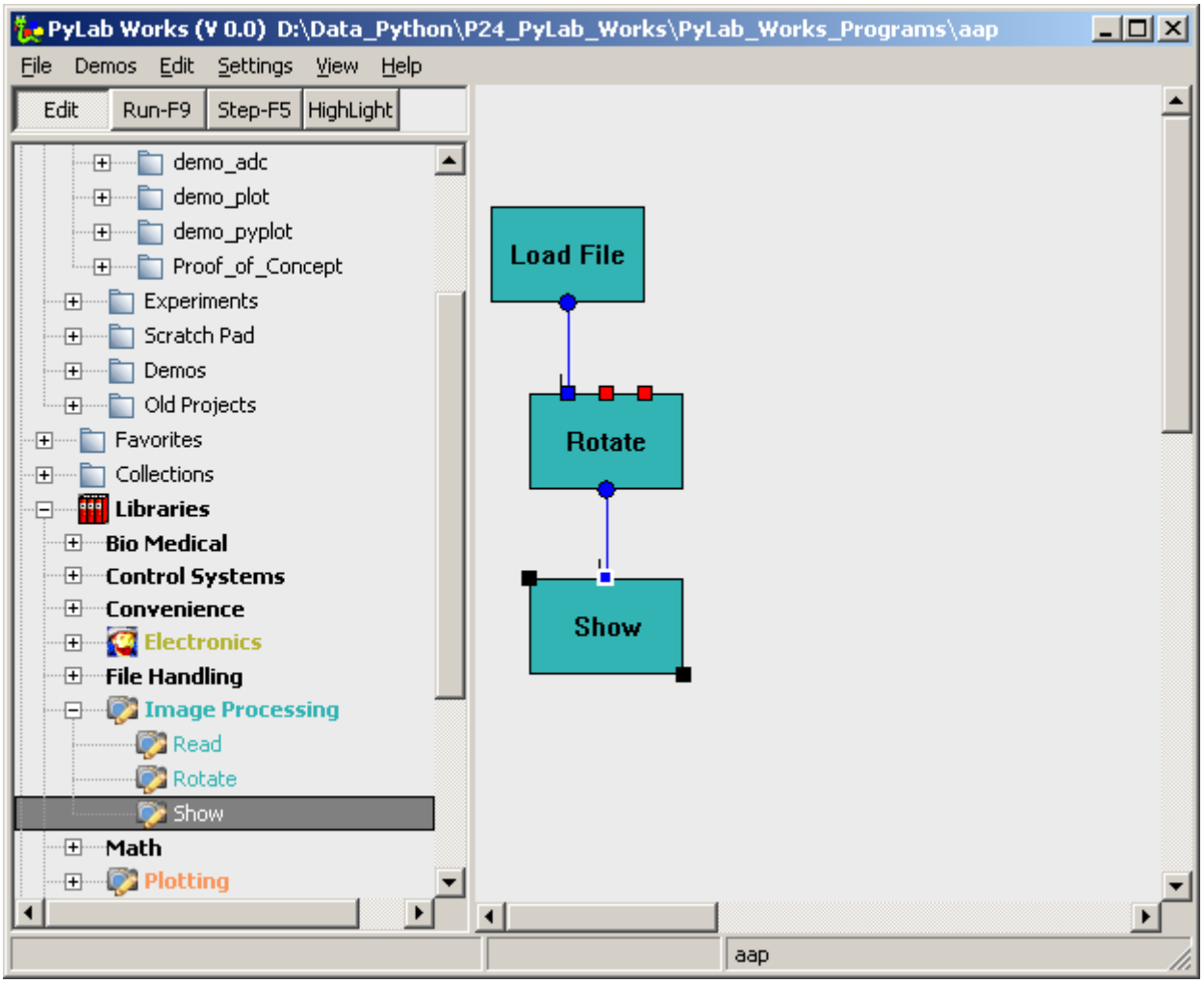
PyLab Works : The main application, this should work pretty well, still it's strongly recommended to start with the demos.
IDE : A simple editor / IDE to edit and test general Python programs. This is a very premature version, just meant for my own to run tests. The Library manager already contains a more mature version of the IDE.
Library Manager : Manage PyLab_Works libraries, download / upgrade from the web. Work in progress, but might give you a good idea of the future version. As already said, this library manager contains a more mature version of the IDE.
Translation Tool : A tool to edit and upload translations of PyLab_Works libraries.

Each of the above modules can be started with a number of **CommandLine Flags** (always lowercase), shown on the lower-left corner:

- debug = print extra debug information
- debugfile = print and append to file (<application>_debug.txt) extra debug information

PyLab Works

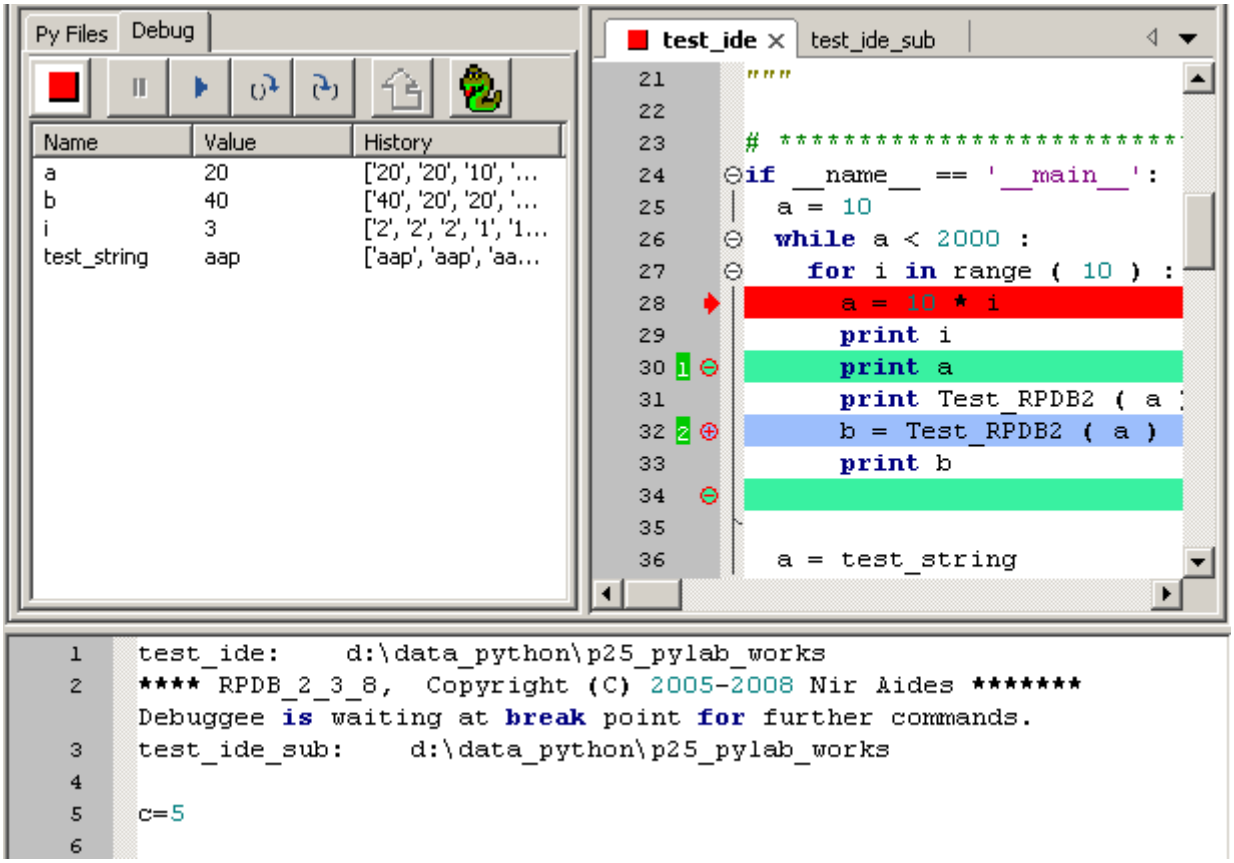
PyLab_Works is an easy development environment, based on the domain knowledge of experts. So the user doesn't need any programming knowledge nor does the user need to be an expert on that particular domain.



IDE

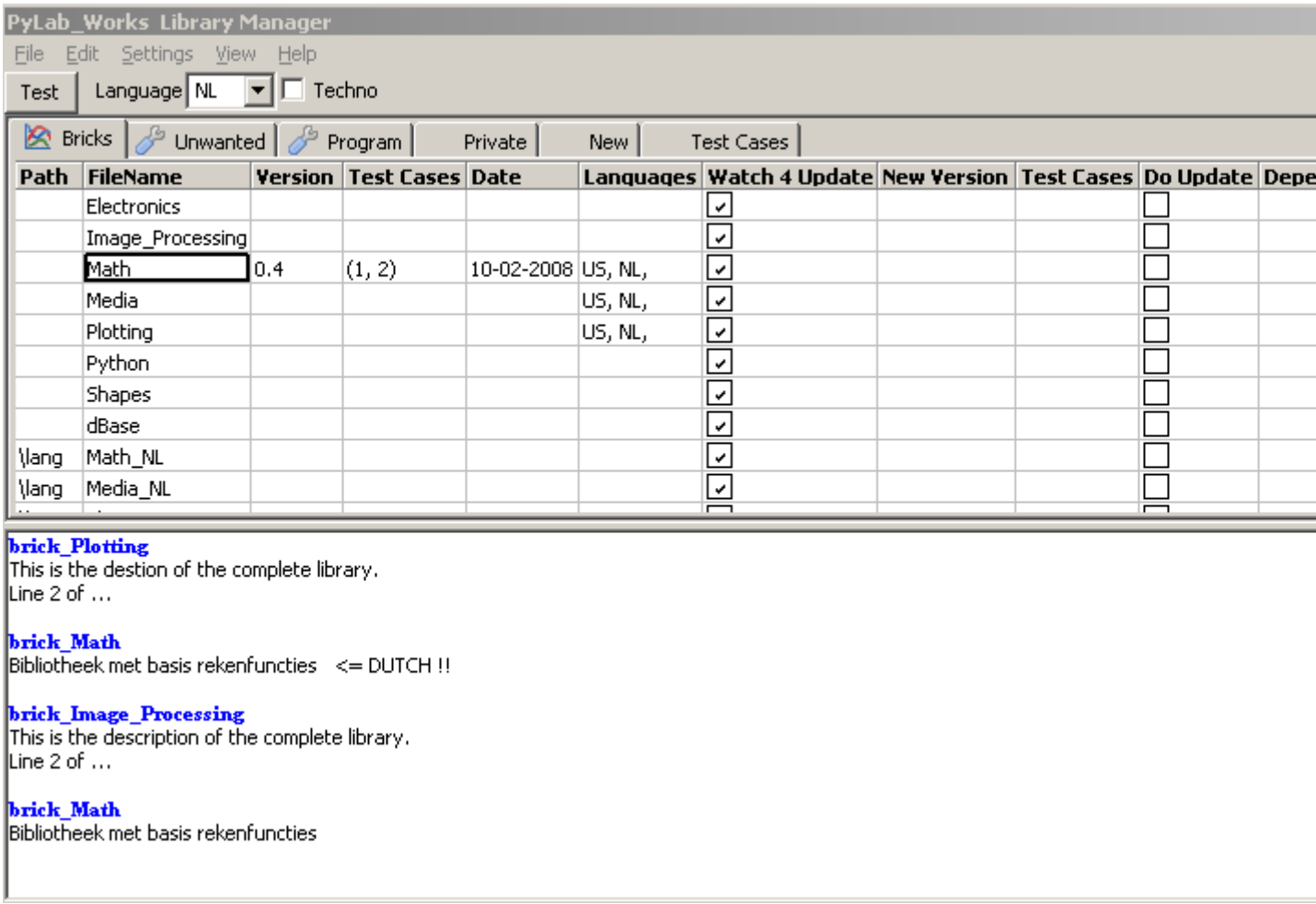
This is a simple but yet quite functional Python-only IDE. Some of the features

- syntax color highlighting
- brute force auto-completion and calltip
- syntax checking while you're typing
- multiform context sensitive help
- integrated debugging, conditional breakpoints, history of variables
- external debugging through winPDB
- auto-save
- interactive command window, using the namespace of the program
- all interactions are logged and file is auto saved / restored



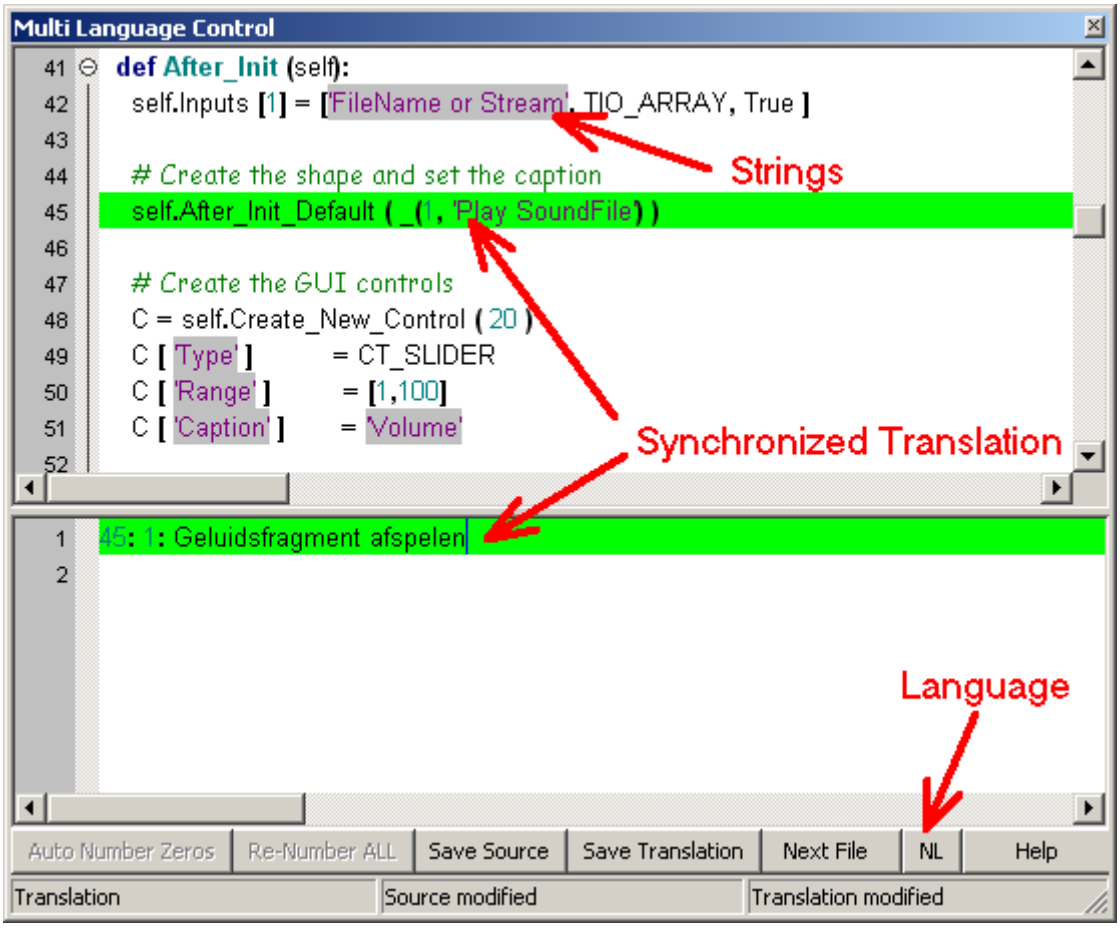
Library Manager

With the Library Manager it's easy to compare / update / rollback your local files with the latest files on the web. The library manager scan your PyLab_works directories and the PyLab_Works Website, compares these two and gives you a nice overview of the differences. The you can decide on an individual file base which files should be updated to the latest version.
Below is image of the Bricks tab, which is a good example of all the other groups.



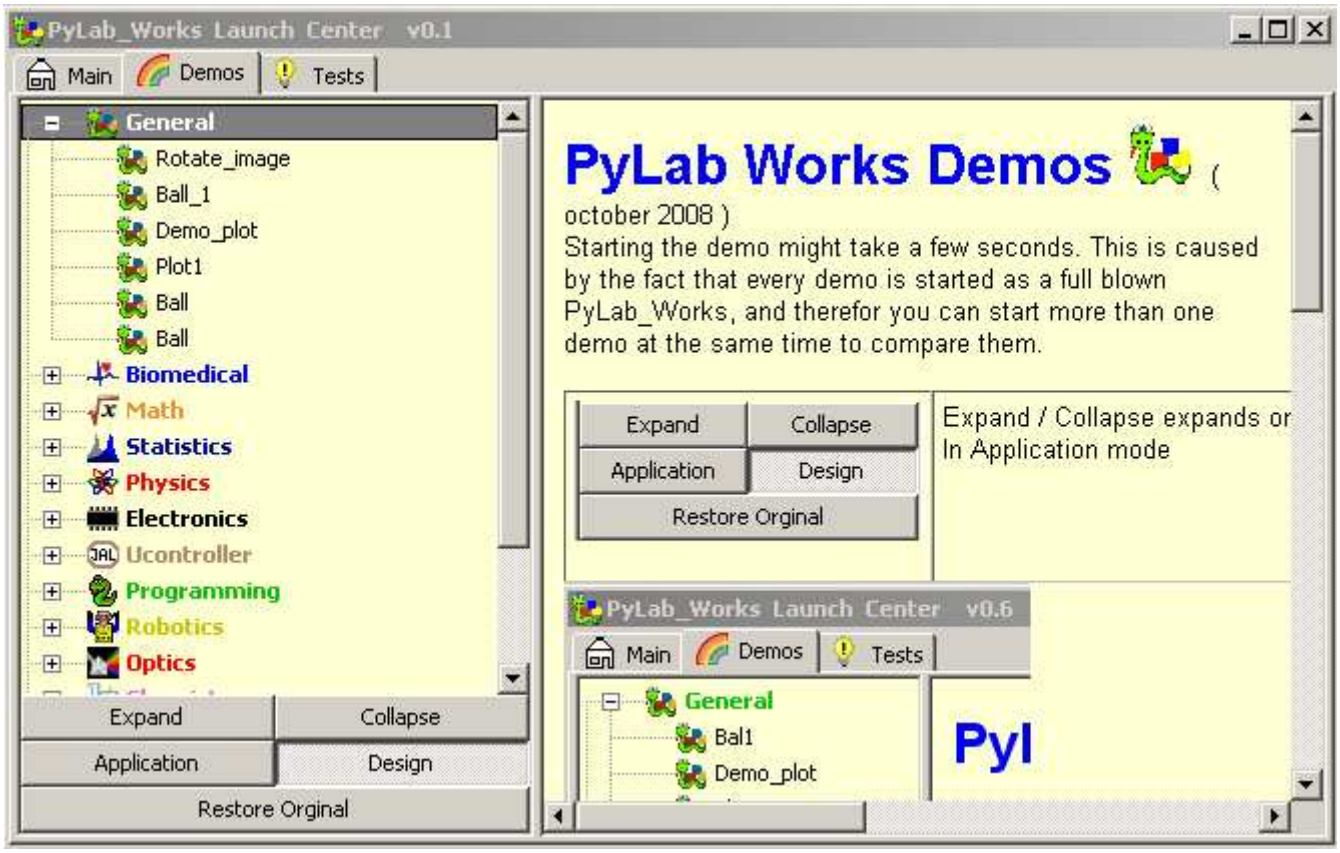
Translation Tool

With the Translation tool it's easy to make a translation of the strings in one or more files, and also to test the original files for strings and see if these strings are made multi-lingual.



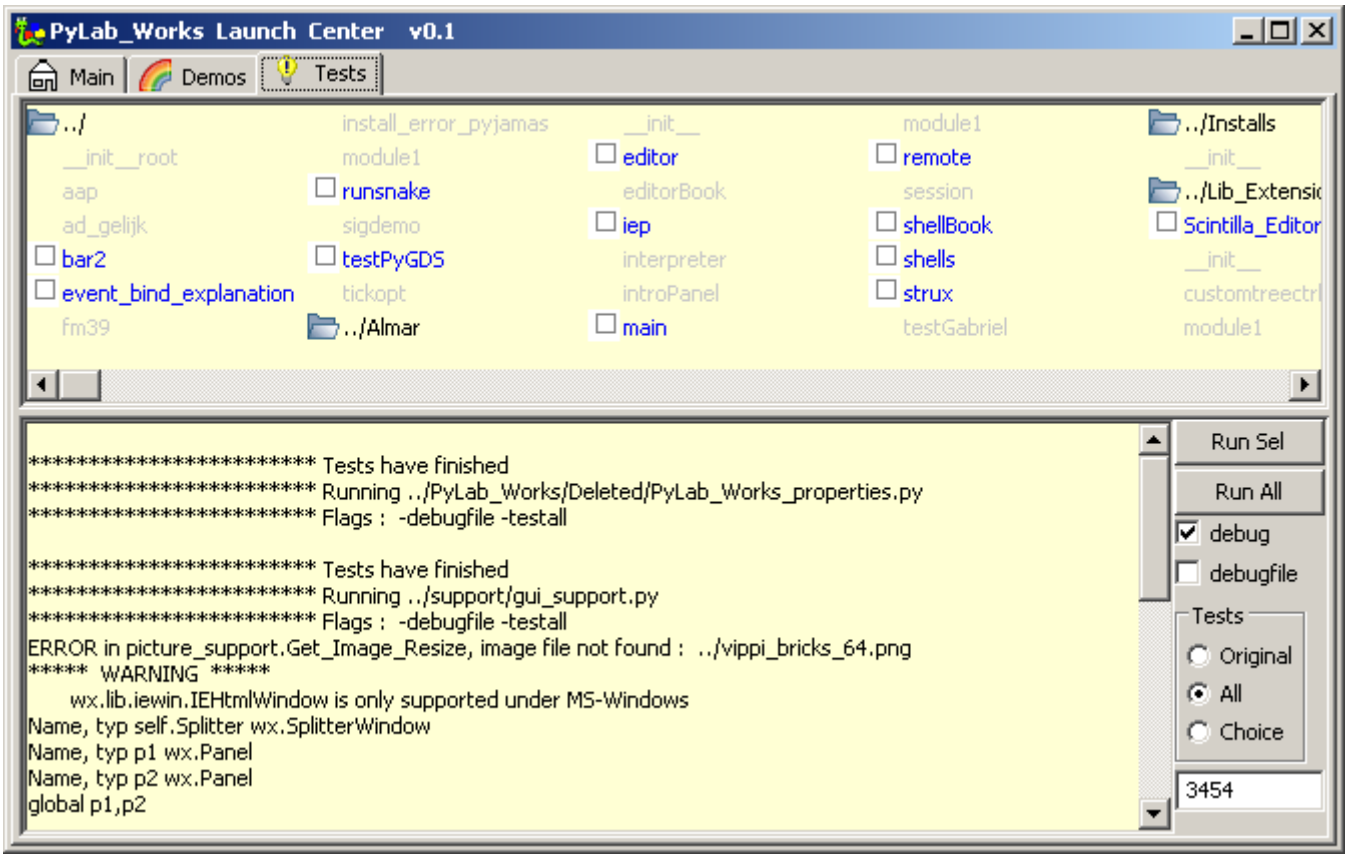
Demos

This page contains a connection to all demos, although quit empty fro the moment, this should grow to a full showcase of many demos in all kinds of applications. You can run each demo in both application mode or design mode, and in the latter you can make changes and see the effects. You can always restore the orginal demo.



Tests

The tests page lists all the files in the project. This is mainly meant for the core developers, but it can be handy to test certain problems in other Operating Systems.



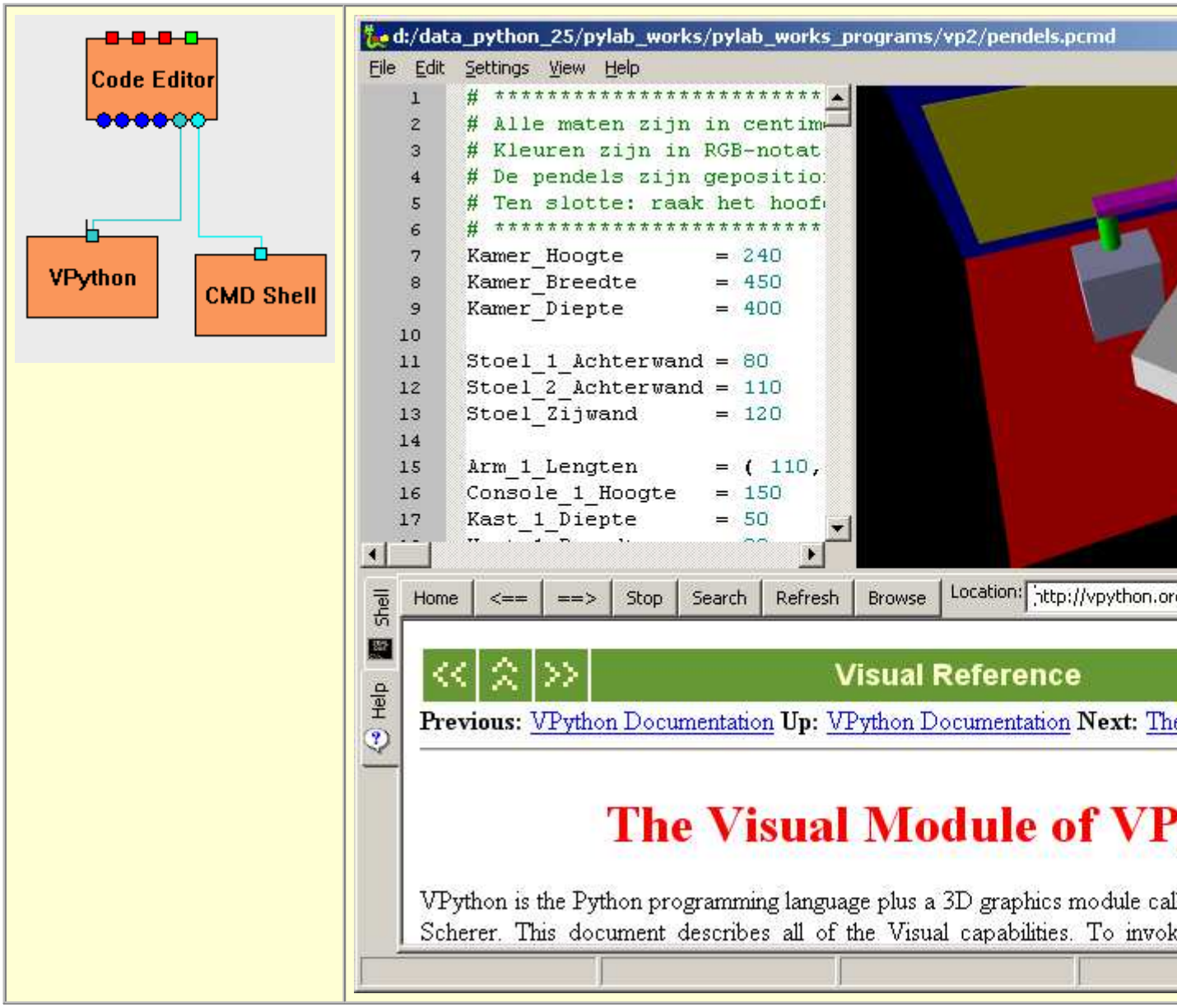
Animations / Screen-shots (november 2008)

Introduction

This page contains a number of screen-shots and links to animated movies of "PyLab Works". The screen-shots / animations below are in reversed chronological order, i.e. the newest screen-shot / animation is on top.
Until ***now*** all screen-shots / animations are made during code cleanup and bug fixing of the very first "proof of concept" version, Therefore you might see somes notes about the specific bug in the description or you might even see the bugs in the demo.
For the time being, the actual documentation can be found here (chapter PyLab Works).

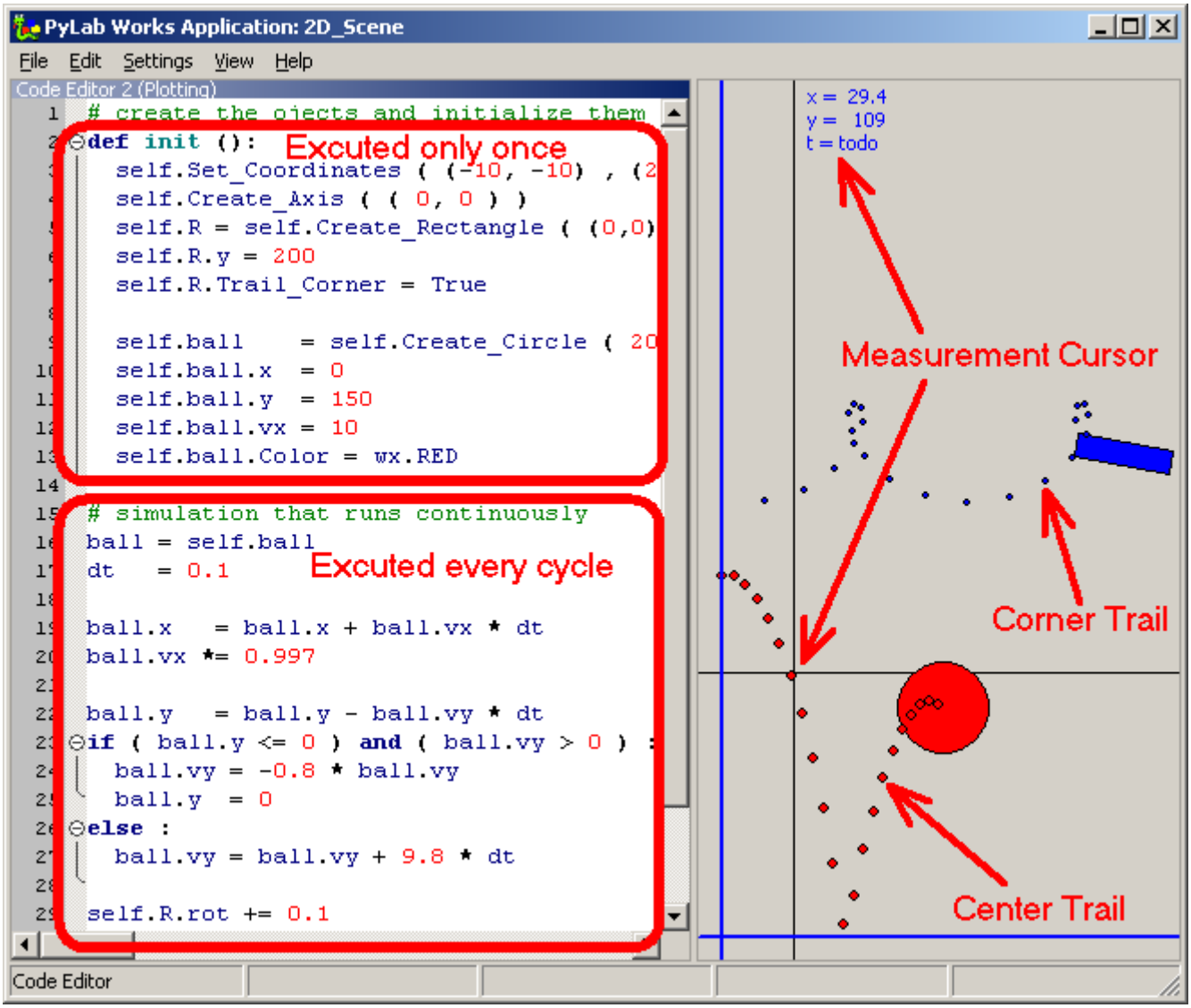
VPython, 3D Scene, CMD-Shell

By combining a code-editor, VPython and a cmd-shell, a quit powerful environment for 3D scenes can be created. These 3D scenes can be used to show physics or to visualize real world environments. Here a flash movie (10 min, 20 MB) can be seen.



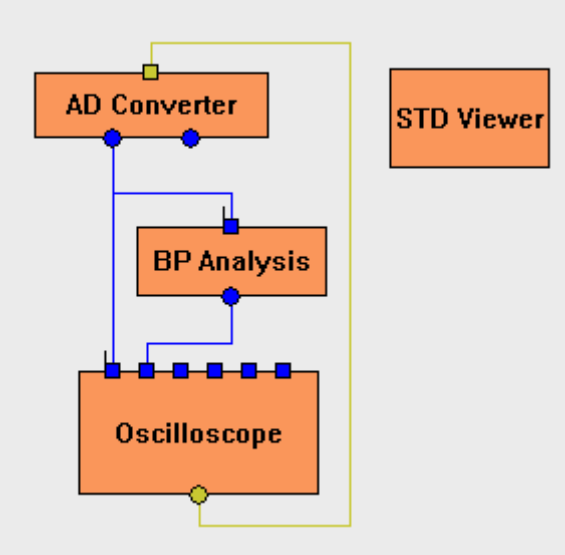
Simple Physics (todo)

This example shows a scenery, where you can place objects, e.g. a canon and a target and by controlling the parameters (i.e. launching angle, ignition energy, gravity, mass of the bullet, wind) you can try to hit the target. The example also includes a code editor, where you can make your pre-calculations. A first attempt can be seen here.



RealTime DataAcquisition

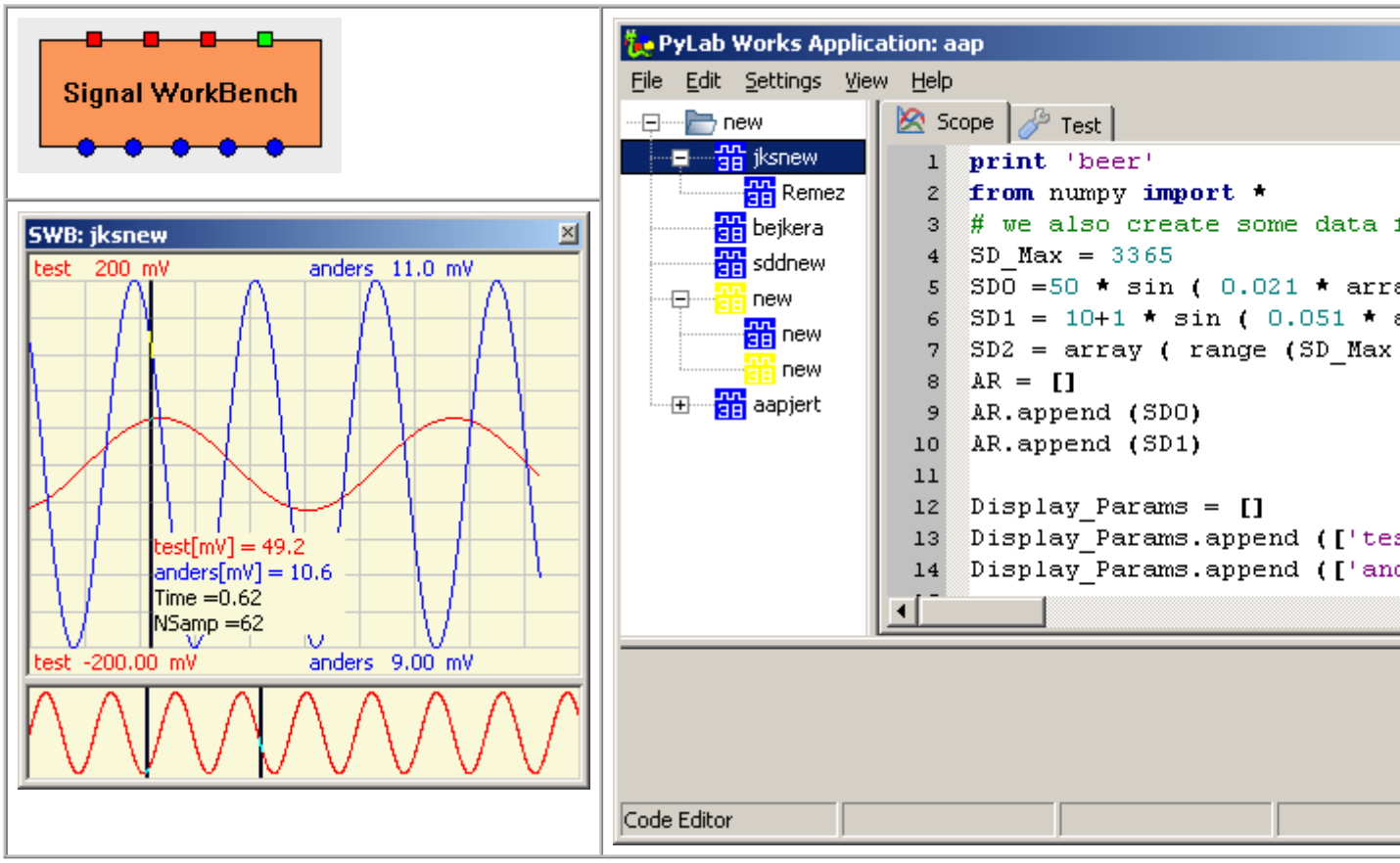
... Shows realtime data-acquisition with hardware AD-cards. Unfortunately this only works on MS-Windows. A number AD-cards is supported: SoundCard, FysioFlex, DataFile, National Instruments DAQmx - cards. This demo also shows the internationalization feature of PyLab_Works.



Simple Math

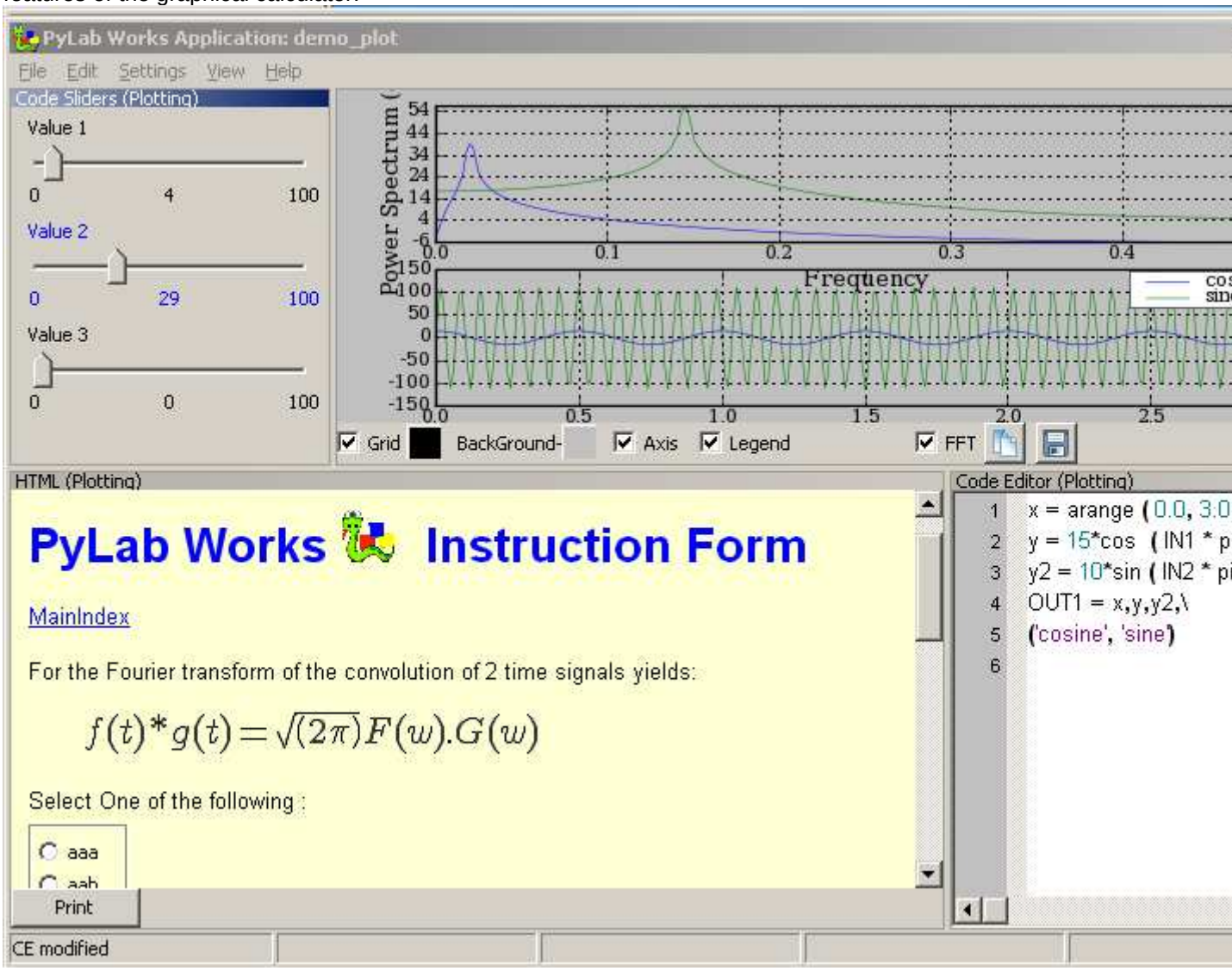
Signal WorkBench (march 2008)

The signal workbench is highly sophisticated graphical calculator, specially equipped for off-line analysis of time series and development of algorithm for real-time analysis. The manual can be seen here. A practical example using the Signal WorkBench for a filter design.



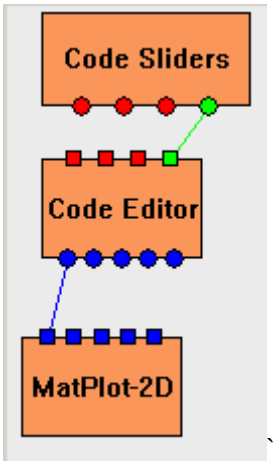
Interactive HTML (january 2008)

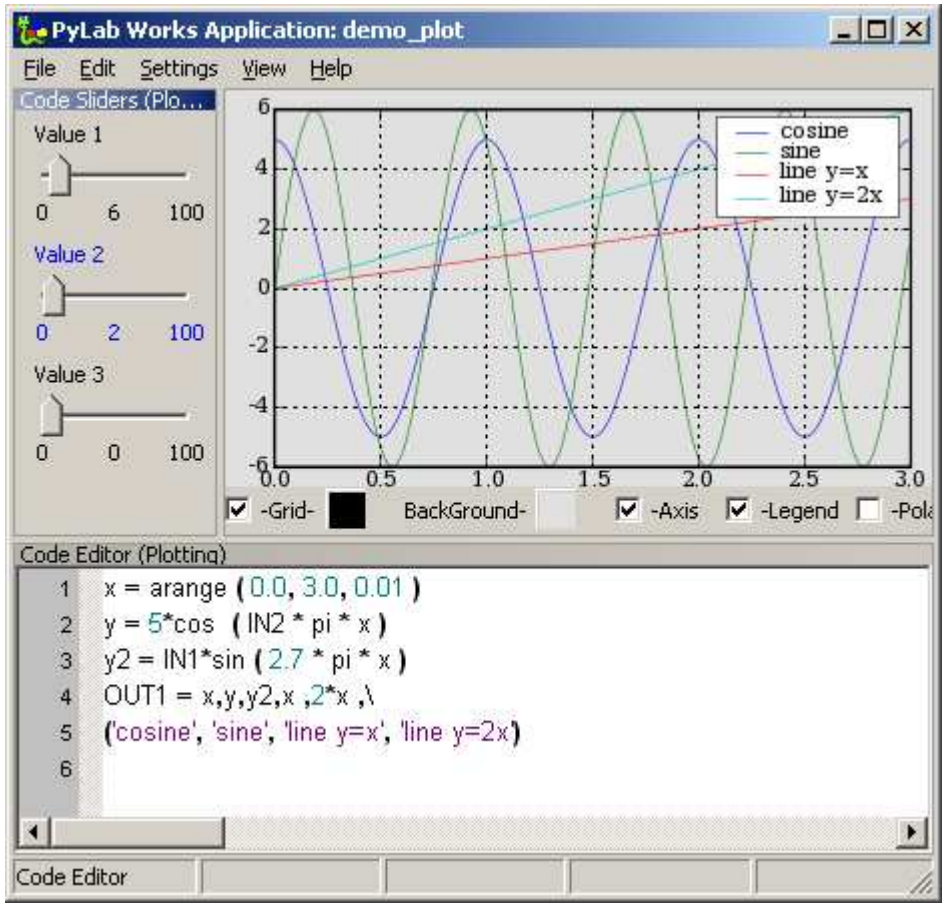
Interactive HTML (2 MB, 3:00) , Showing an interactive HTML page with wxWidgets, that can be used for courses, to give instructions and getting the answers of students back. This demo also shows a few new features of the graphical calculator.

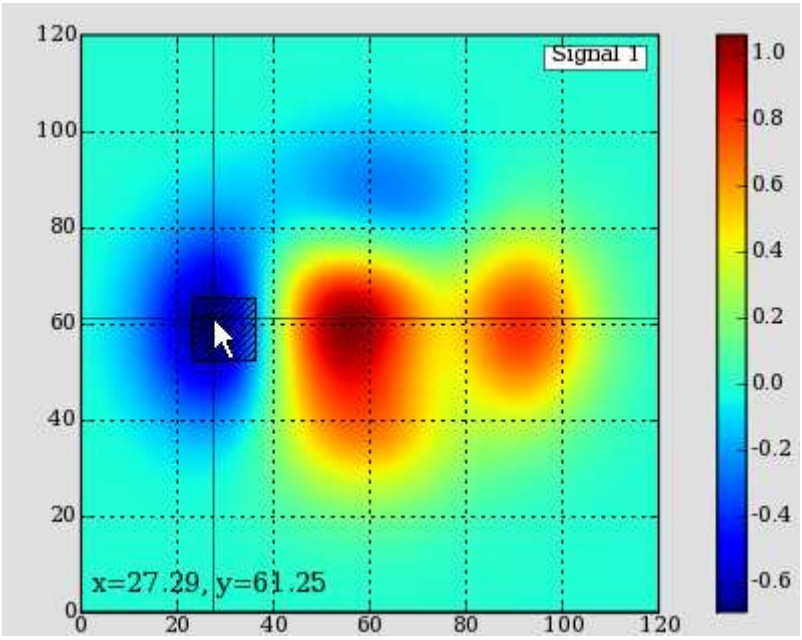


Graphical Calculator (january 2008)

Graph-Calc-Demo (2 MB, 5:30) , Create a quit sophisticated graphical calculator with just a few mouse clicks. The demo shows most of the possibilities of the graphical calculator. With a set of sliders (as shown in the figure below) you can easily change one or more parameters in your code and get directly feedback of the changes of these parameters to the total functions(s) results.
The demo also shows some of the major features of the code editor, like code-completion and code snippets manager.

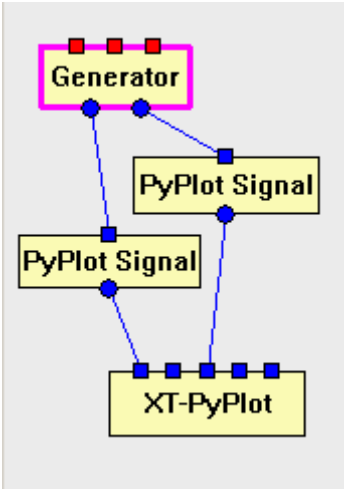


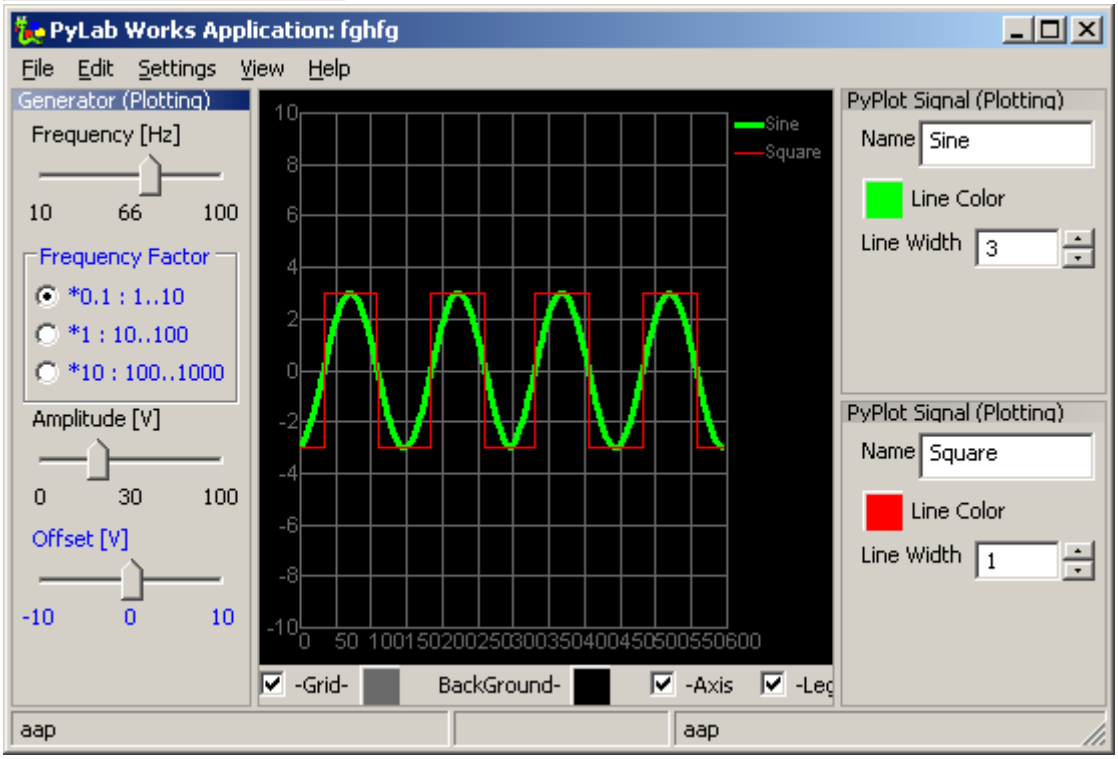




Generator + PyPlot (december 2007)

PyPlot-Demo (3.4 MB, 4:30) , Showing sine / square generator with a realtime XT-plot based on a slightly modified PyPlot.
Demonstrates how signals can be extended with extra features, in this case: signal color, line width and signal name.
This demo reveals a fundamental bug in unequal path lengths for some specific controls.
Probably PyPlot is too slow for fast realtime signals, so we'll also implement ScopePlot and for even more beautiful pictures we'll introduce MatPlot.

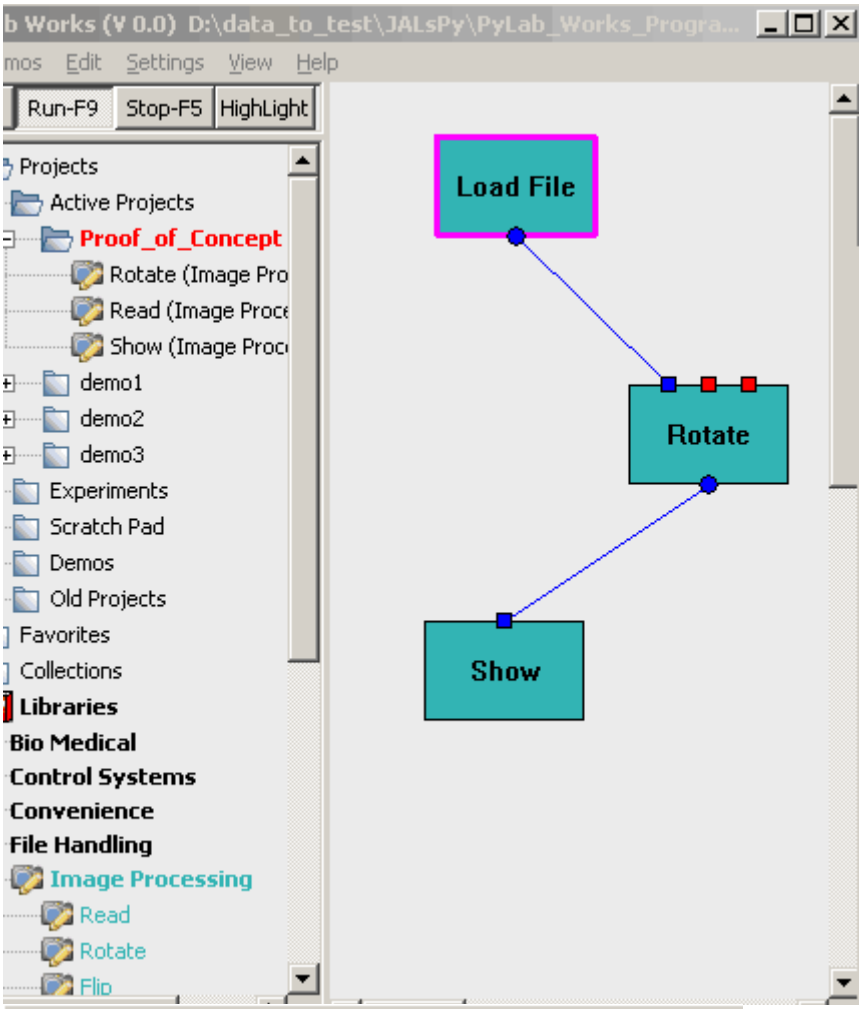


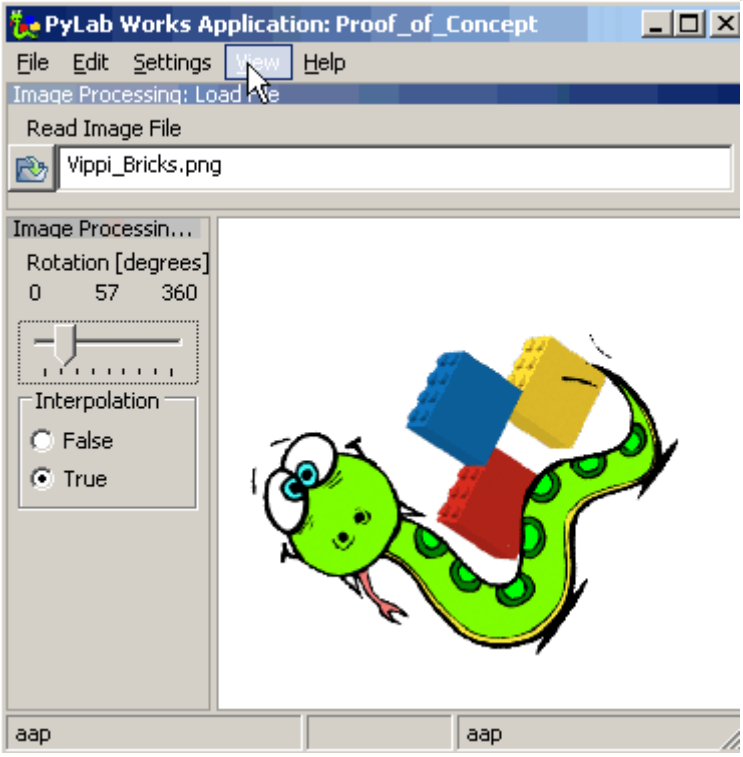


Proof of Concept (december 2007)

Proof of Concept (10MB, 8 minutes)

This demo shows the most important parts of the concept I've in mind. What you see is already working, and what I don't show you, is probably not yet implemented. Judge for yourself !





march 2008

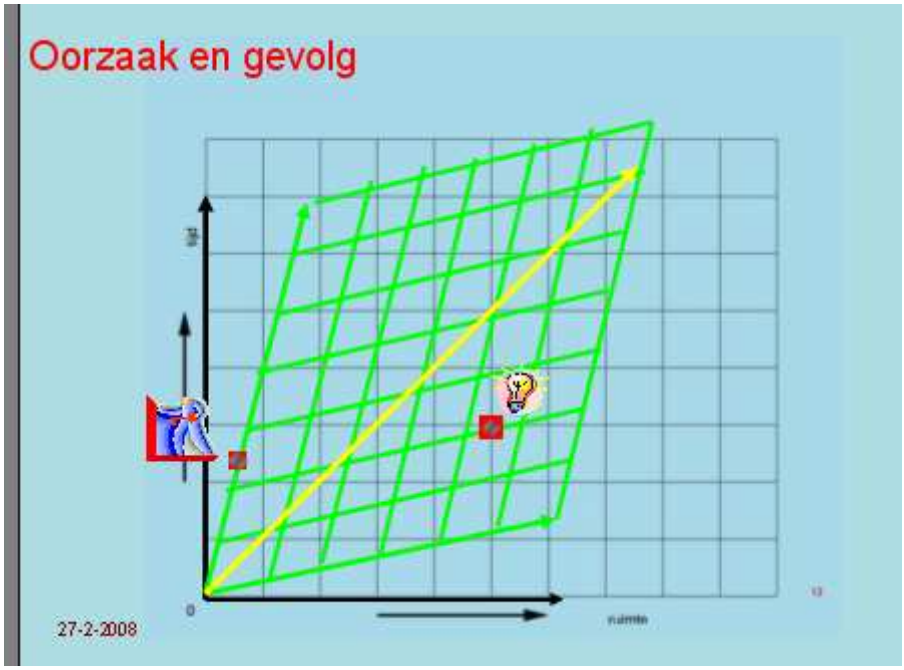
Application Ideas

This page contains a number of ideas, I want to implement in PyLab_Works.
Most of the pictures are borrowed from other applications / websites.

Time-Space-Relativity-Theory

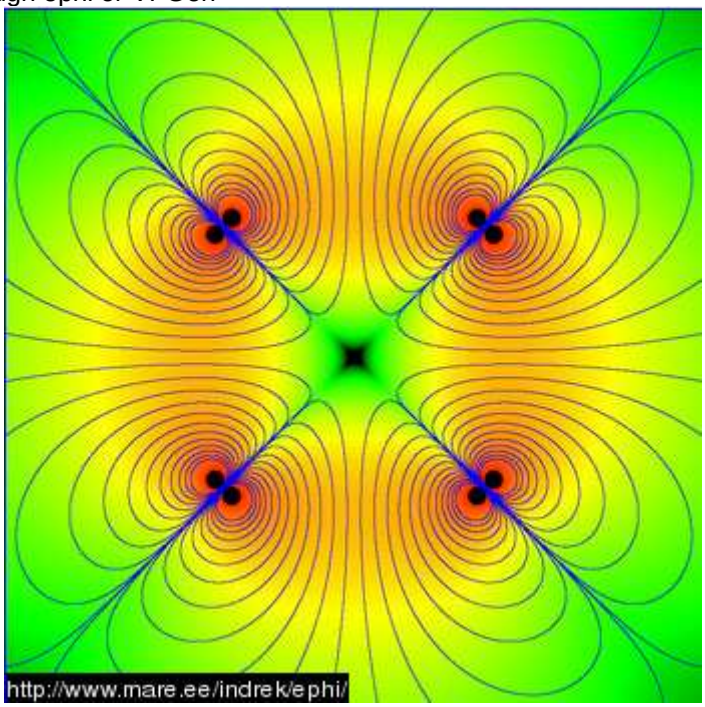
see also Visual demos

Visualizing Proper Time in Special Relativity [with LightClocks] (Rob Salgado)



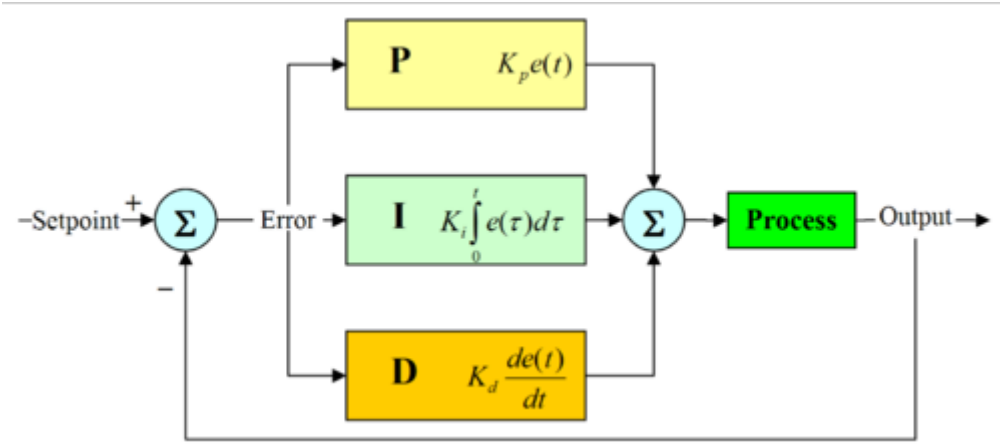
Magnetic Field Visualization

through ephi or VFGen



Course Time-Frequency domain
Fourier / Laplace / Wavelet

Control Systems & Modelling
Implementing Control Systems is a very interesting task. If you look at a control system, you can see it's very easy to implement it in the Brick designer. But this has two disadvantages, you don't want the normal domain user to use the designer and secondly the simulation is probably to slow and closed loop solutions are not easy to obtain. Further more you want interaction from the graphs (e.g. shifting poles and zeroes) back into the system. So probably it will end up with a new control, which is an almost copy of the brick designer. (There are applications were something similar is needed, like the robot in a room with obstacles).

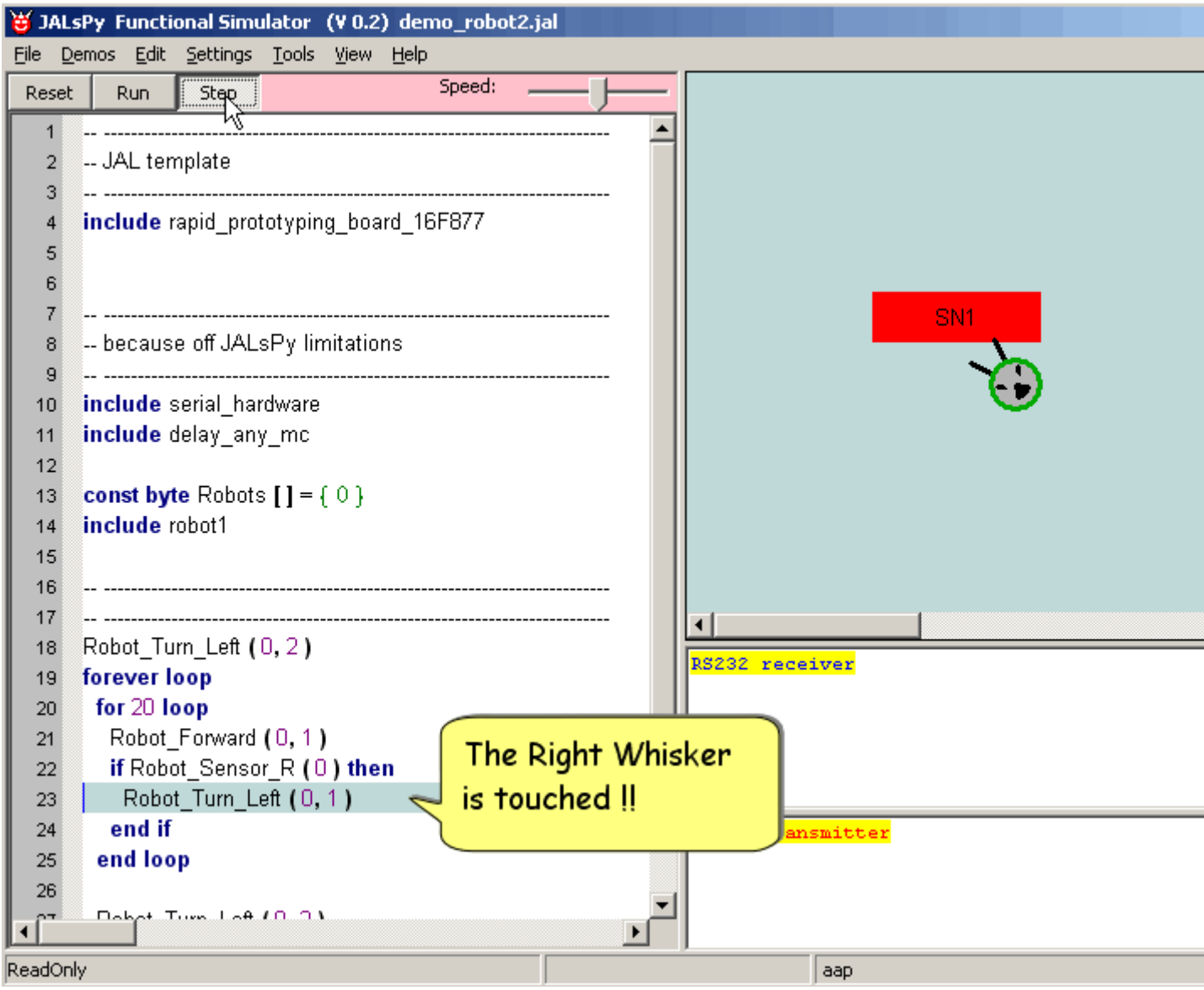


Neural Nets

Geometric Optics

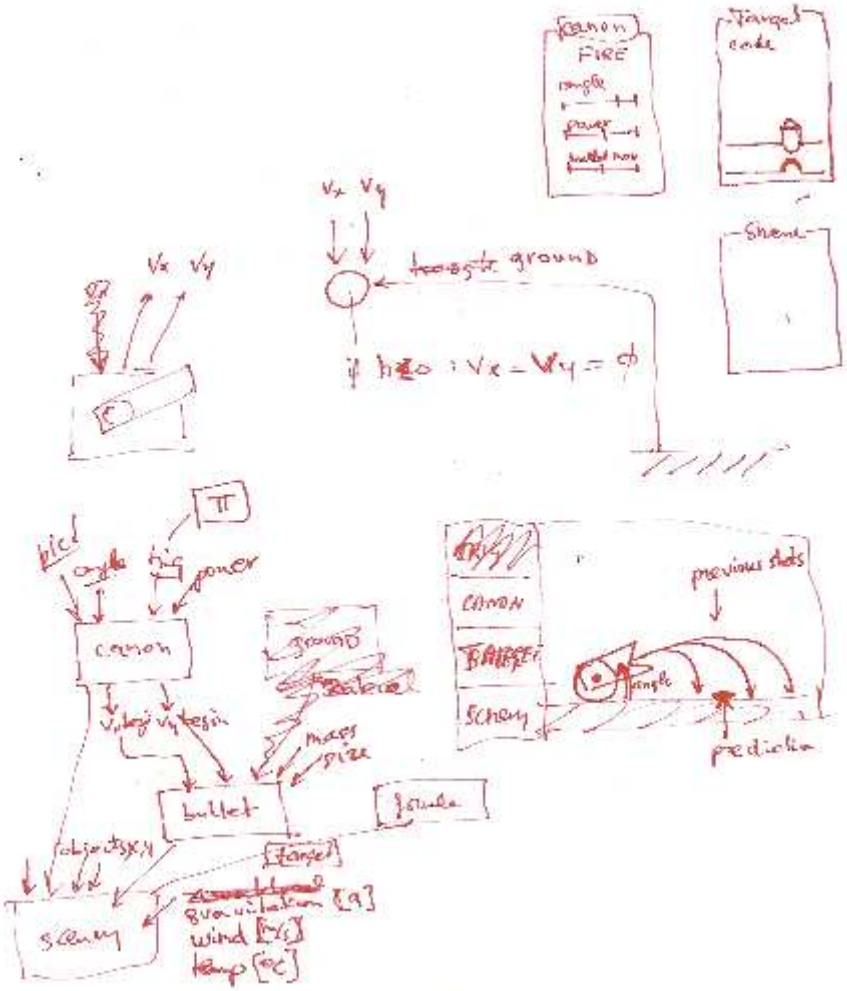
Laser Optics

Robots in a 2D space
PyLab_Works started as a fork of a micro-controller simulator, called JALsPy, also written in Python. Implementing parts of this project would be fun. Here an image of JALsPy.

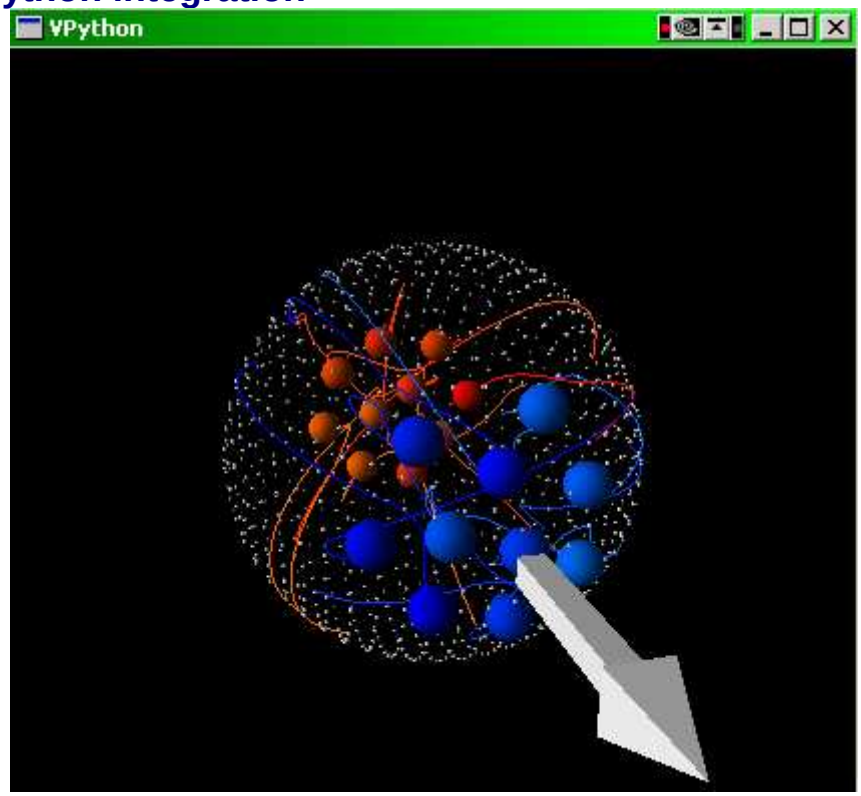


Canon ball trajectory

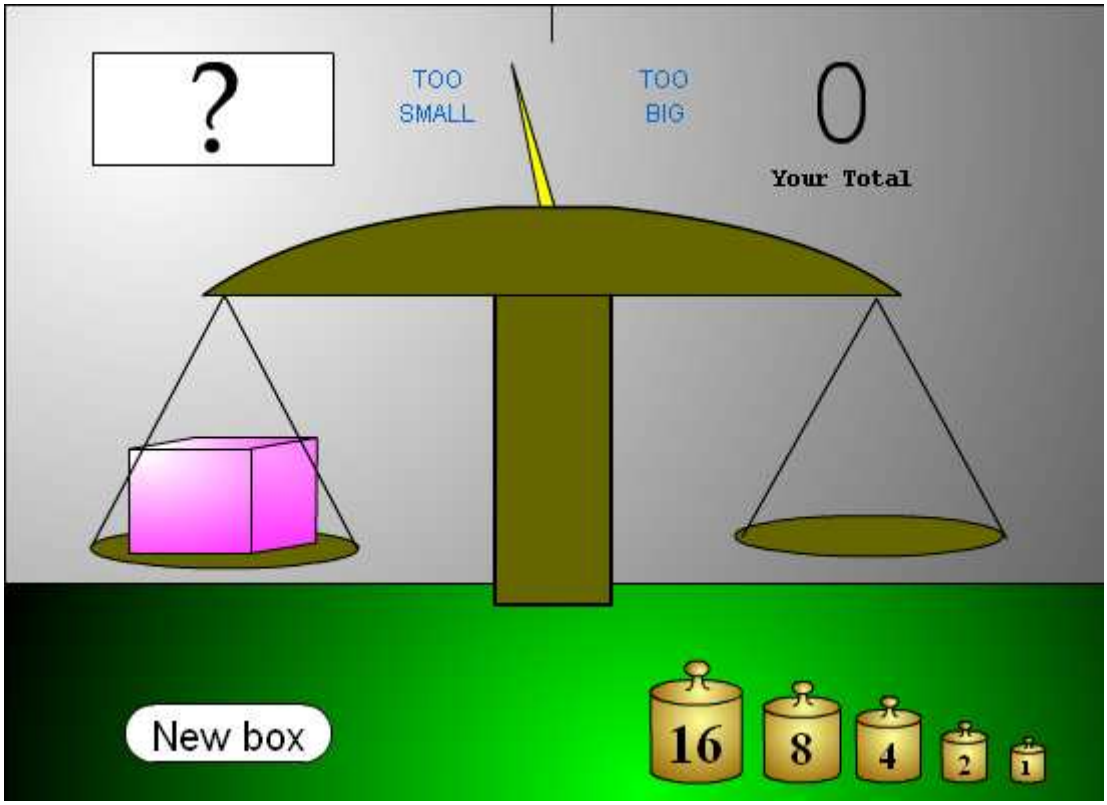
This example shows a scenery, where you can place objects, e.g. a canon and a target and by controlling the parameters (i.e. launching angle, ignition energy, gravity, mass of the bullet, wind) you can try to hit the target. The example also includes a code editor, where you can make your pre-calculations.



VPython integration



Education for Kids



VPython Collection

(january 2009)

Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

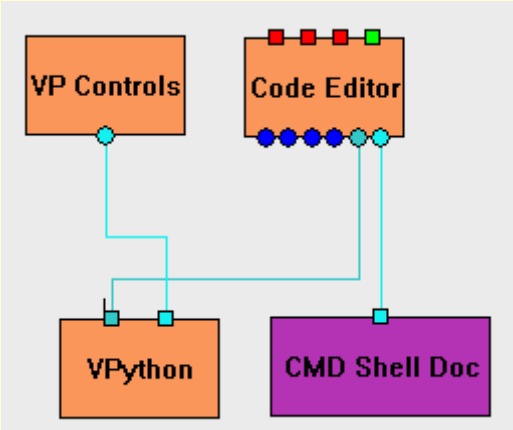
This is a PyLab_Works example application, that manages a large set of VPython scripts. All python scripts in a fixed directory are assumed to be VPython scripts. A selected Vpython script can be run and edited. The whole "code" to build this application with PyLab_Works is shown on the right, and the resulting program can be seen below this table.

Some explanation:

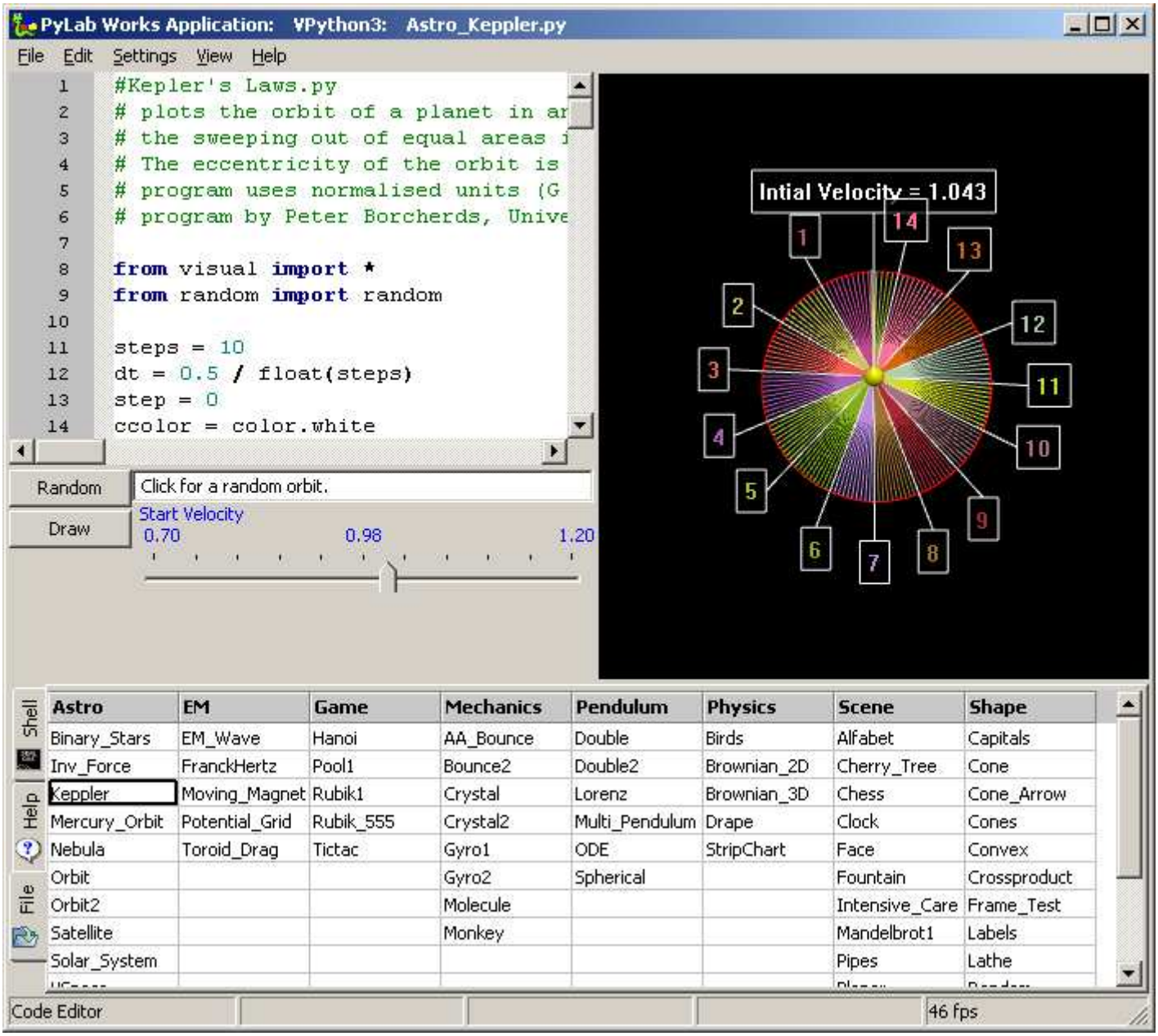
The **VPython window** is the dock window where the scene of Vpython will be projected. The **Code Editor** will contain the selected VPython script, and is build around a full blown Scintilla editor. The **CMD Shell Doc** is a multi tab control, that contains

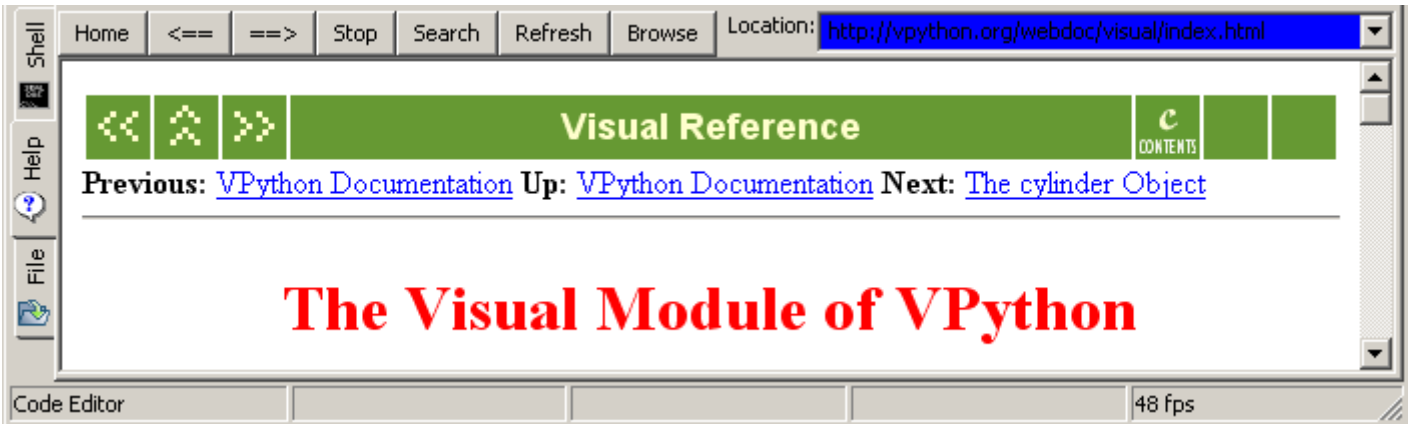
- a command shell, that works in the namespace of the VPython Script.
- a simple web-browser, with powerful shortcuts, that display all kind of files (pdf, ...)
- a sorted file-list that lists all available VPython scripts

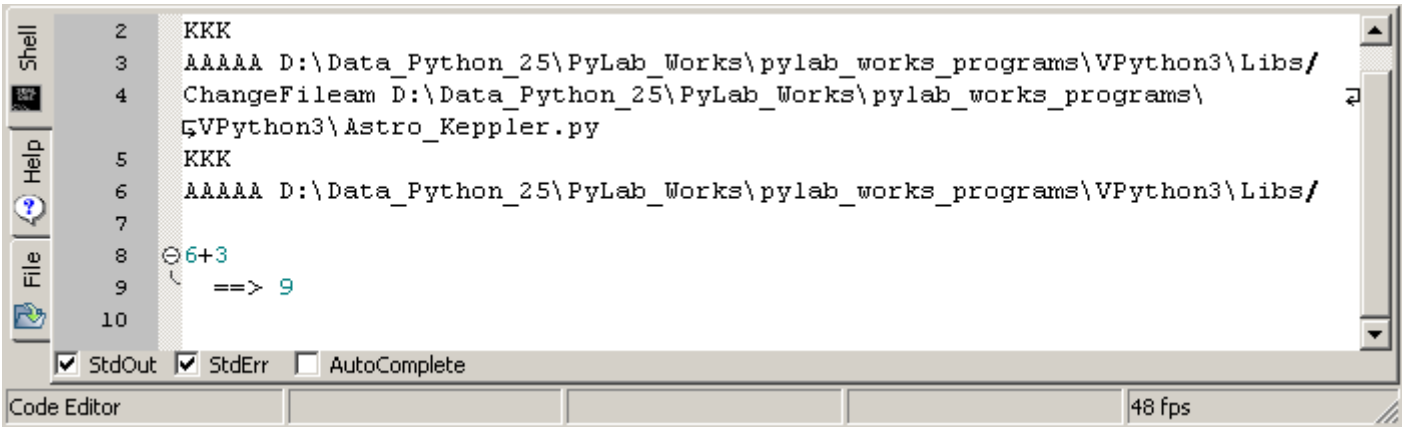
The **VP Controls**, is a dynamically generated set of controls, that is created by and can interact with the VPython script.



Normally the program will run VPython-5, but you can force VPython-3 with the commandline switch -VP3

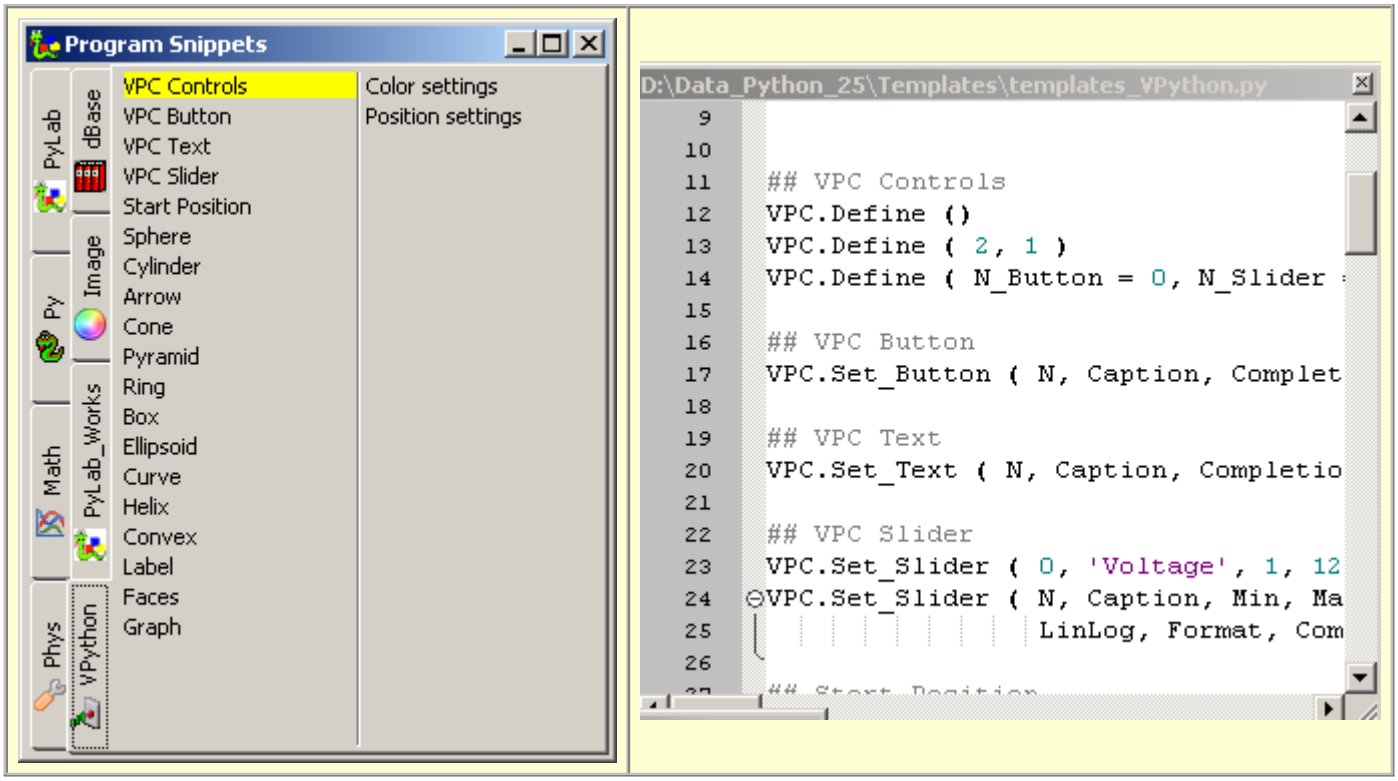

PyLab Works Application: VPython3: Astro_Keppler.py
File Edit Settings View Help
#Kepler's Laws.py
# plots the orbit of a planet in ar
# the sweeping out of equal areas i
# The eccentricity of the orbit is
# program uses normalised units (G
# program by Peter Borcherds, Unive
from visual import *
from random import random
steps = 10
dt = 0.5 / float(steps)
step = 0
ccolor = color.white
Random
Click for a random orbit.
Draw
Start Velocity
0.70
0.98
1.20
Intial Velocity = 1.043
Shell
Help
File
Astro EM Game Mechanics Pendulum Physics Scene Shape
Binary_Stars EM_Wave Hanoi AA_Bounce Double Birds Alfabet Capitals
Inv_Force FranckHertz Pool1 Bounce2 Double2 Brownian_2D Cherry_Tree Cone
Keppler Moving_Magnet Rubik1 Crystal Lorenz Brownian_3D Chess Cone_Arrow
Mercury_Orbit Potential_Grid Rubik_555 Crystal2 Multi_Pendulum Drape Clock Cones
Nebula Toroid_Drag Tictac Gyro1 ODE StripChart Face Convex
Orbit Gyro2 Spherical Fountain Crossproduct
Orbit2 Molecule Intensive_Care Frame_Test
Satellite Monkey Mandelbrot1 Labels
Solar_System Pipes Lathe
Code Editor
46 fps


Shell
Help
File
Home <== ==> Stop Search Refresh Browse Location: http://vpython.org/webdoc/visual/index.html
Visual Reference
CONTENTS
Previous: VPython Documentation Up: VPython Documentation Next: The cylinder Object
The Visual Module of VPython
Code Editor
48 fps



Code Snippets

The Code Editor is equiped with a easy to use and easy to modify code snippet manager which can be invoked by F7. Of course the code snippet manager uses the same Scintilla editor.



Dynamic Controls

In the code you can define what controls you need. At this moment only Button,Text and Slider are available, and Text is not fulley working as planned.
All calls should be made to a virtual instance "VPC", as shown above in the code snippets. "Completion" is the method that should be called by the control after it's value has changed.

```
299  def Define ( self, N_Button = 0, N_Slider = 0, N_Text = 0 ) :
```

```
260  def Set_Button ( self, N, Caption, Completion ) :
```

```
267  def Set_Text ( self, N, Caption, Completion = None ) :
```

For a logarithmic slider you should set LinLog = 'Log'

```
276   def Set_Slider ( self, N, Caption,
277                    Min, Max, Value,
278                    LinLog, Format, Completion ):
```

Examples

This program contains a dynamically controlled set of VPython-3 examples. You can add more files to the collection, just be copying them to the appropriate VPython3 directory. If you want to occur the file in a specific group, add a prefix to the filename, indicating the group. e.g. to add a VPython3 program named "moonlanding.py" to a group "Astro", just rename the file to "Astro_moonlanding.py", because the first part of the filename before the first underscore is used to identify the group.
Almost all VPython demos included here are found on the web. Where possible I've noted the author, the creation date and the website where I found it. If anyone discovers creations of her his own, let me know, and I'll add your name.
Most of the scripts were modified, either to fulfill the PyLab_works conditions or to show the effect of the dynamical PyLab_Works controls.

Changes General

In this paragraph, the changes made to each program are described. If you want to make the changes in such a way that outside PyLab_Wors the program runs unmodified, you can use the constant PW_Embed, which is available when running the module inside PyLab_Works:

In general the following rules yields for VPython files to behave well in PyLab Works:
Not allowed (means forbidden)

- Name the main display other than the default "scene", otherwise it can't be captured
- Setting the height / width / x / y of the scene (not allowed after creating the scene and embedding it)
- Setting the title of
- Setting scene.visible = False (because then the VPython window get's undocked)
- scene.mouse.getclick(), because this blocks the GUI thread. Use ... instead :

```
87        if scene.mouse.clicked > 0 :
88          while scene.mouse.clicked > 0 :
89            scene.mouse.getclick()
```

- Tripple quoted strings in the main body
- from __future__ import division (doesn't seem to work, only if it's the first import of PyLab_Works)
- object derived classes gives sometimes? problems: class carrow(): #object):
- changing fov, mangles the rescaling of all other projects (userzoom can not be determined)

Preferred

- don't use rate, because the update speed is fixed by PyLab_Works (for the moment at 50 fps). Setting a slower rate, might degrade the GUI performance. Setting a higher rate doesn't has any effect.
- Setting camera direction (forward) and scaling (range)

```
128   # PyLab Works: added
129   scene.forward = ( -1, -2, -1 )
130   a = 3
131   scene.range   = ( a, a, a )
```

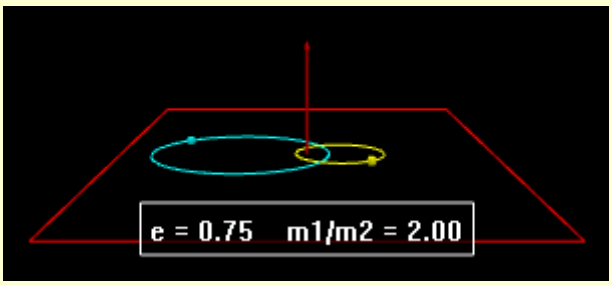
-

Not necessary

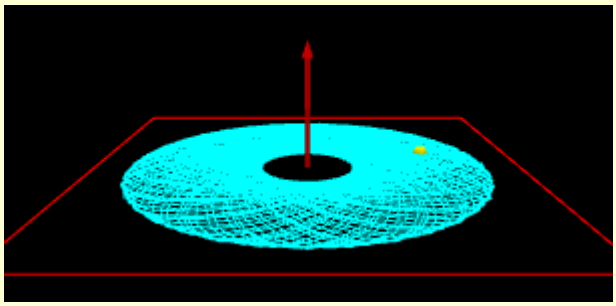
- "from visual import *", because it's done automatically, but it doesn't harm
-

Astro group

Binary_Stars (by E. Velasco. November 2004)
This program visualizes the orbits of two stars (Binary stars). We use scaled variables. If alpha = G*m1*m2, a = major semiaxis of the ellipse of m2 around m1, and e = excentricity of that ellipse, we rescale the distance r between the stars and the time by: r->r/a, t->t*sqrt(alpha/mu*a^3), where mu is the reduced mass. The equations of motion for the relative position of m2 with respect to m1 are, in the rescaled variables are:
r_dot_dot = -1/r^2+(1-e^2)/r^3 and phi_dot = sqrt(1-e^2)/r^2
with initial conditions: r = 1-e, r_dot = 0, phi = 0. With the rescaled variables the period of the orbit is 2*pi. The program uses a very rudimentary midpoint solver for the
equations of motion, but good enough for this demo. It only computes one cycle and then repeats it, to avoid drift. Change the value of e and the masses to get different orbits, with e = 0 to get the circle.



Inv_Force (by E. Velasco, October 2004)
This program visualizes the orbit of a particle of mass m in an atractive central force field = -alpha/r. The initial conditions are given by r=r0, r_dot=0 (i.e r0 is a turning point in r) and phi=0 phi_dot=k/r0*sqrt(alpha/m). The dimensionless parameter k adjusts the value of the angular momentum (k=1 corresponds to a circular orbit). We now rescale variables to a new r =r/r0 (so the starting point is now r=1) and rescale the time to a new time t=t*r0*sqrt(m/alpha), so phi_dot = k. The equations of motion in these new variables are:
r_dot_dot = -1/r+k^2/r^3 and phi_dot = k/r^2
with initial conditions: r=1, r_dot=0, phi_dot=0. The program uses a very rudimentary midpoint solver for the equations of motion, but good enough for this demo. Change the value of k to get different orbits (k=1-> circle).
** To clear the orbit left-click the mouse **

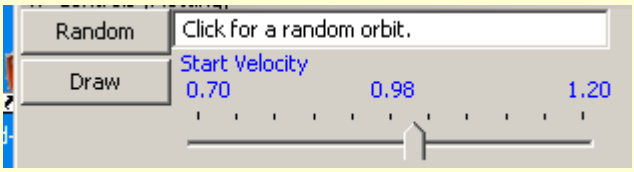


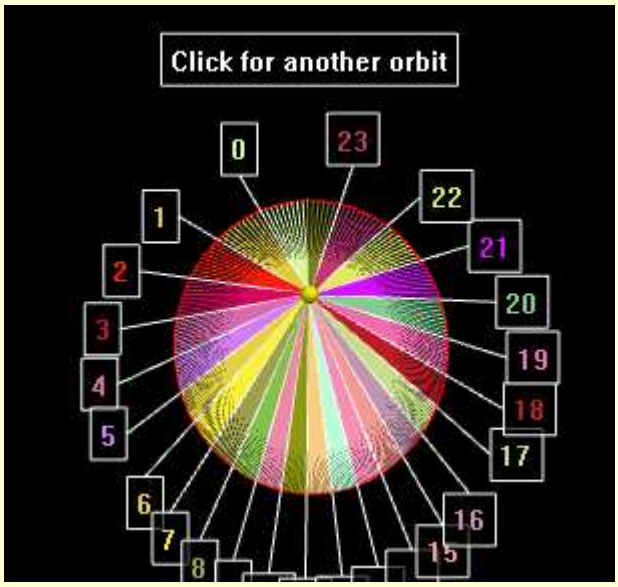
Keppler (by Peter Borcherds, University of Birmingham, England)

(Kepler's Laws.py)

Plots the orbit of a planet in an eccentric orbit to illustrate the sweeping out of equal areas in equal times, with sun at focus. The eccentricity of the orbit is random and determined by the initial velocity program uses normalised units (G =1)

This demo also shows how you can very easy and effectively use wxPython controls.



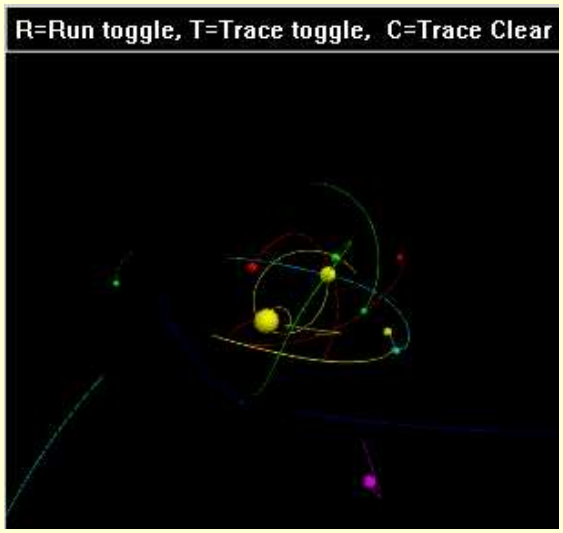


Mercury_Orbit (by Andrew Williams, 2002, website)

Demonstration of tidal locking in Mercury's orbit. There are three complete rotations of the planet for every two orbits of the sun. Hopelessly out of proper scale, of course, it's meant to illustrate the motion

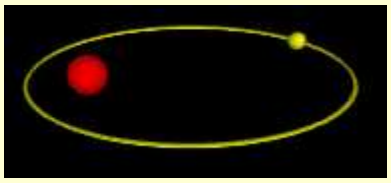
Nebula
(originally by Chabay and Sherwood, adapted by Lensyl Urbano January 2006, website)

from the above website:
"This is a model of the gravitational collapse of a number of particles in space. Particles start with semi-randomly assigned initial positions and masses and their motion is governed by their initial velocity and Newtonian attraction. When particles collide they coalesce and eventually, a central star forms while a few planets remain in orbit. Because the final orbits do not lie on a single plane this model can be used to explain both the advantages and limitations of numerical modeling."



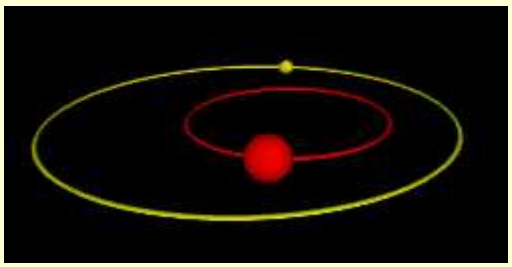
Orbit (by ??)

Nice to add a vector with the magnitude of the speed / force.



Orbit2 (by ??)

Nice to add a vector with the magnitude of the speed / force.



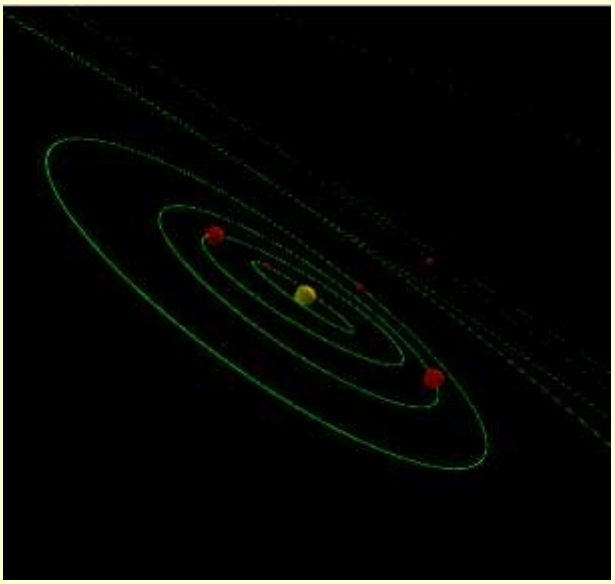
Satellite (by ??)

This satellite crashes after a while on the earth. It would be nice to draw the trail. In VPython-5 the earth crash, also causes a VPython-5 crash.

Solar_System (by Lensyl Urbano, Jan. 2006, website)

Simple solar system model showing the orbits of the planets. In this version the orbits are not tilted so the longitude of the ascending node is not correct.

It would be nice to give the planets more "natural" colors.
Also buttons, to switch between an overview of inner and outer planets would be nice.
At this moment the orbits are not drawn, because it takes to much time, must be improved.



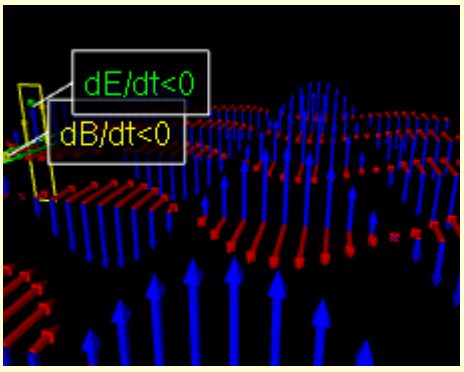
VSpace (by Ron Adam, March 2005, website)

A space voyaging demo.
By moving the mouse you can determine the direction in which you'll travel.
Runs very slow in VPython-5

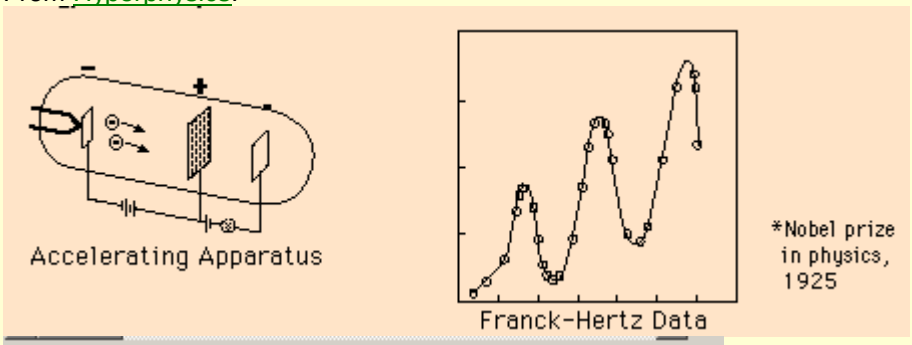
EM group

EM Wave (by Rob Salgado, march 2004, website)

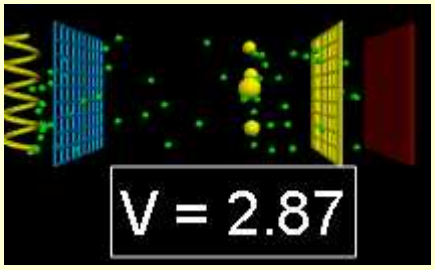
(EMWave.py) illustrates the relationships among electric and magnetic fields in electromagnetic radiation (plane wave).



FranckHertz (by Aaron J. Miller)
One of the first experiments, that indicated that the exited states of atoms are quantized.
From Hyperphysics:



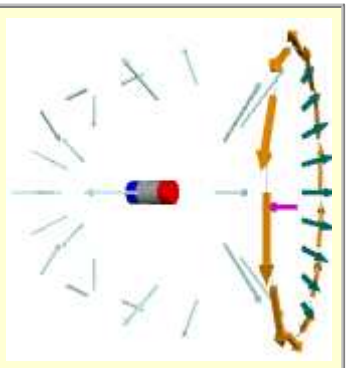
It would be nice to extend the example with a current meter, which also makes a plot of the current versus the exitation voltage.



Moving Magnet (by Ruth Chabay, 2007-08-07, Revised Bruce Sherwood 2007-11-10)

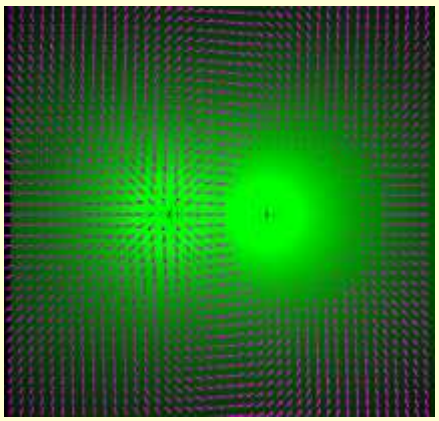
Faraday's Law: bar magnet moves at constant velocity, then briefly comes to rest and reverses

Choose whether to show a ring or a disk through which passes magnetic flux
Choose whether to show B at many places, not just in flux region
Magenta arrow represents the vector dB_vec/dt



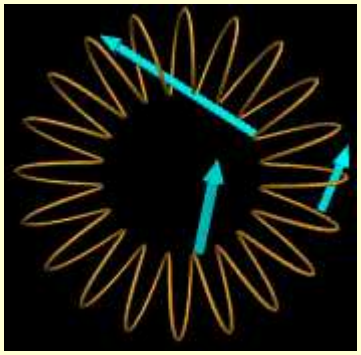
Potential Grid (probably by Bruce Sherwood)

Shows the electrical field of dipole.



Toroid Drag (probably by Bruce Sherwood)

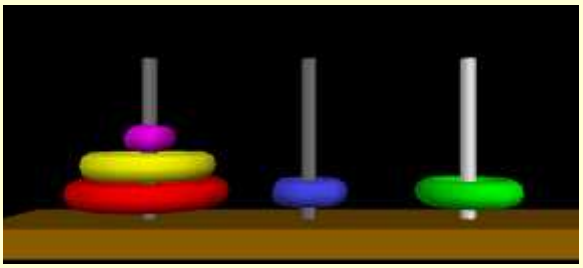
Click to show the magnetic field interactively, by holding the left-mousekey down, you can drag the start location. After you release the mouse the arrow is fixed.



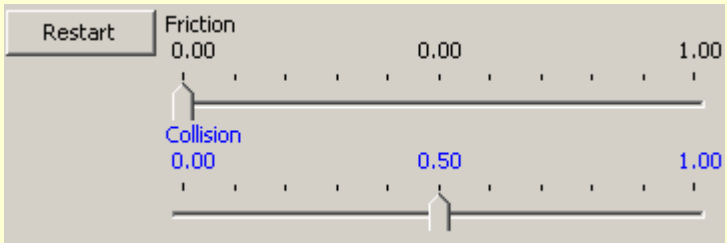
Game group

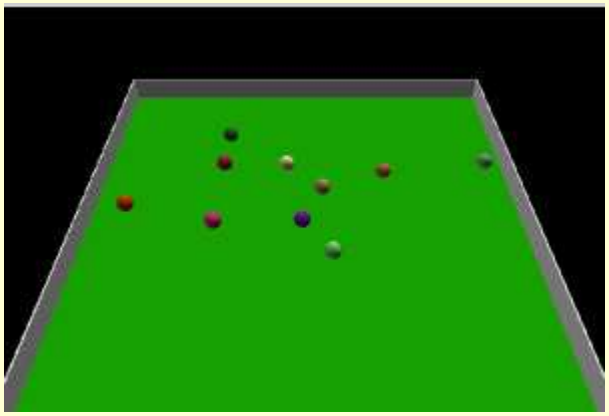
Hanoi (by Ruth Chabay, carnegie mellon university, 2000-06)

Towers of Hanoi: rings may be dragged through stuff - a little surreal



Pool1 (??)



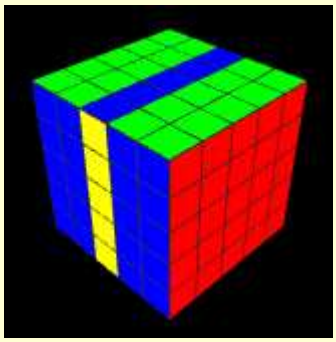


Rubik1 (??)

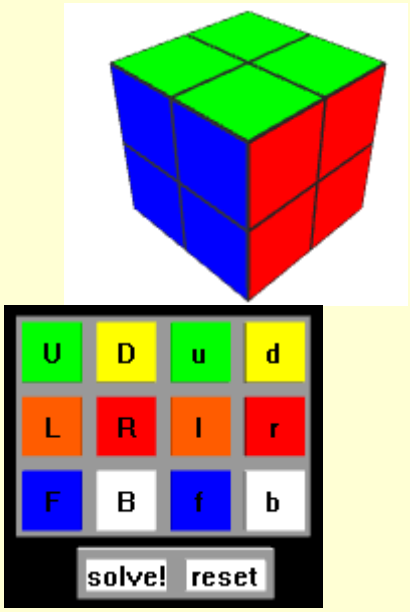
Still don't understand when a click really results in a move.

Rubik_555 (??)

Still don't understand when a click really results in a move.



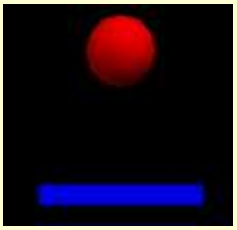
Rubik_Small, Rubik2,3,4 are not implemented, although the controls works well, they are not closed when the script closes. Either the controls must be taken over or the controls windows must be made invisible on closure.



Mechanics group

AA_Bounce

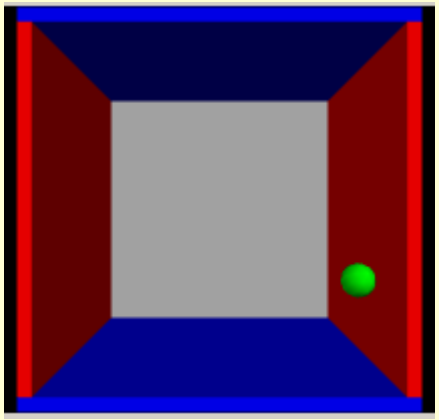
Thé bouncing ball demo.



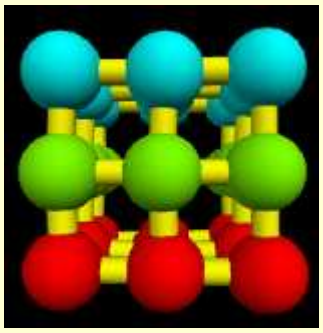
Bounce2

Bouncing ball in a box.

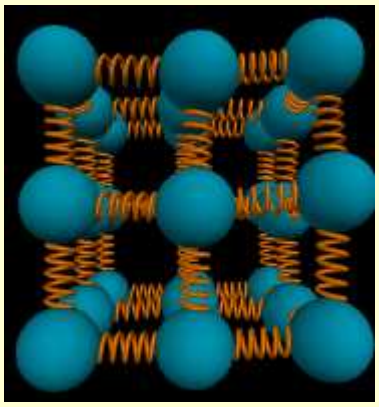
In VPython-5 it would be nice to make the walls opaque, so you can rotate the cube.



Crystal (??)



Crystal2 (by Bruce Sherwood; revised by Jonathan Brandmeyer)



Gyro1 (by E. Velasco. December 2004)

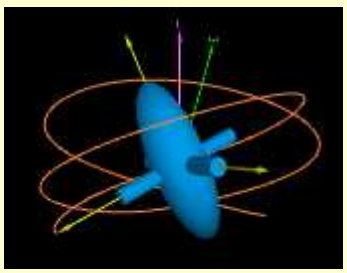
Demo of a symmetric (I1=I2) gyroscope. It uses numerical integration with a primitive mid point solver of Euler's equations using the Euler angles [phi,theta,psi] as variables.

Left mouse toggles the display of the trail of the orbit of the tip of the gyroscope. Right mouse toggles the display of the vectors L and omega (angular momentum and velocity).

Gyro2 (by E. Velasco, December 2004)

Demo of the free motion of a symmetric top in the reference frame of the center of mass (fixed at the origin).

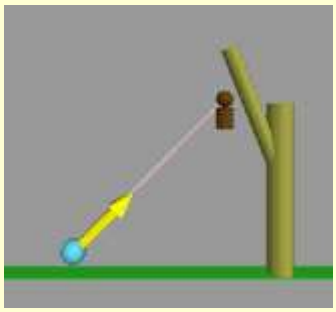
Left-click the mouse to toggle the curve of the tip of the x1 moving axis. Right-click to toggle display of the angular momentum and angular velocity.

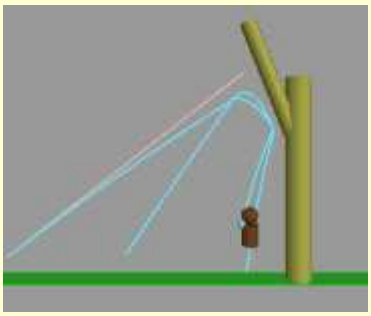


Molecule (??)

Monkey (John W. Keck, February 2003)

The classic Shoot-the-Monkey Demo.





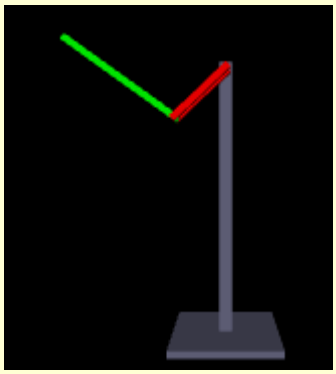
Pendulum group

Double (by Eduardo Sánchez Velasco. September 2005, website)

Shows a double pendulum, in which the tracing of the end joint can be toggled.

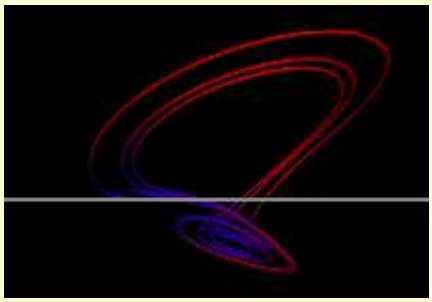
Double2 (??)

The analysis is in terms of Lagrangian mechanics. The Lagrangian variables are angle of upper bar, angle of lower bar, measured from the vertical.

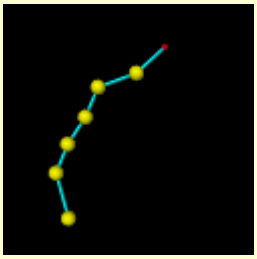


Lorenz (??)

Lorenz differential equation

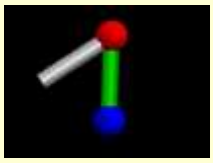


Multi_Pendulum (??)



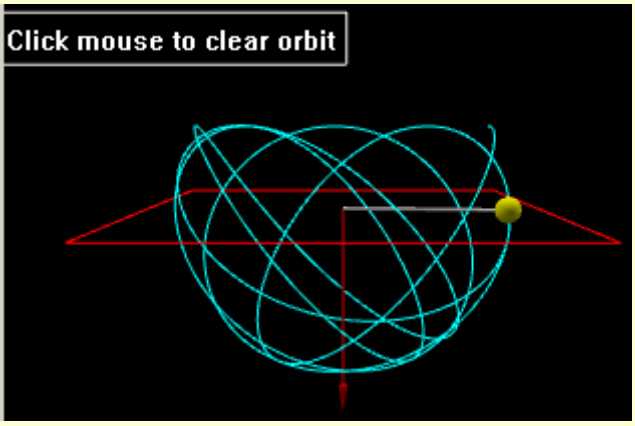
ODE (by Miles Jacobs, , website)

(visualjoints.py) uses pyODE (http://pyode.sourceforge.net), an open-source physics engine



Spherical (by Eduardo Sánchez Velasco. September 2004, website)

Shows a spherical pendulum, a trace is following the end-joint, which can be cleared by clicking on the VPython window.



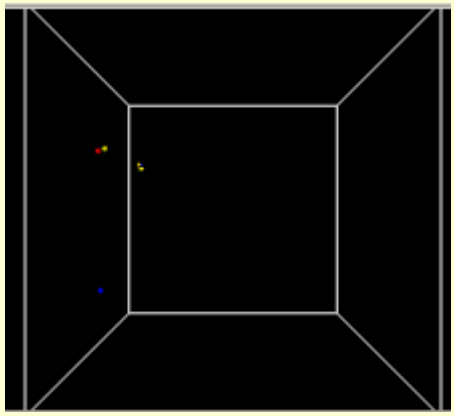
Physics group

Free_Path (14_spark_mean_free_path.py, Bruce Sherwood Fall 2000, website)
watch one atom move through a gas

Birds (by Eric Nilsen, September 2003, website)

boids.py generates bird flocking behavior

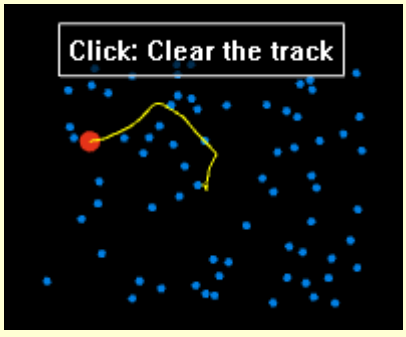
Ideas for version 2.0:
- predators
- obstructions
- perching on the ground for a bit
- prevailing wind
- random flock scattering
- cone boid shape --> change boid axis to indicate direction



Brownian 2D (by E. Velasco. July 2006 version)

Simple demo to illustrate the motion of a Big brownian particle in a swarm of small particles in 2D motion. The spheres collide elastically with themselves and with the walls of the box. The masses of the spheres are proportional to their radius**3 (as in 3D).

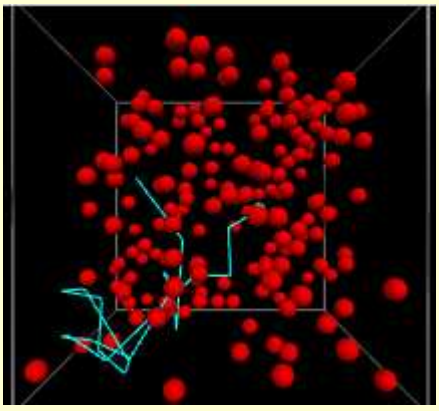
Clicking the left mouse button toggles the display of the orbit of the Big brownian sphere.



Brownian 3D (by Bruce Sherwood Fall 2000)

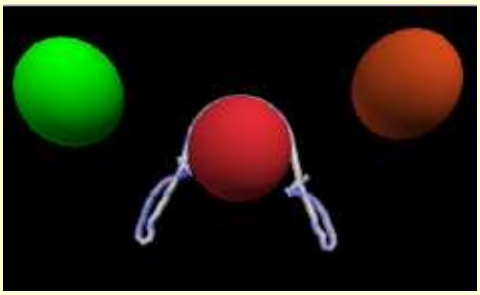
A model of an ideal gas with hard-sphere collisions. One atom is tracked; its mean-free path is determined more by the size of the container than by the gas density.

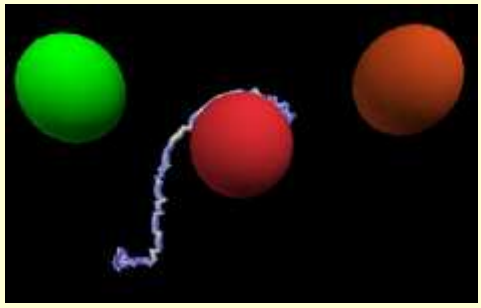
Ideas:
- change number of atoms
- change temperature
- clear trail



Drape (??)

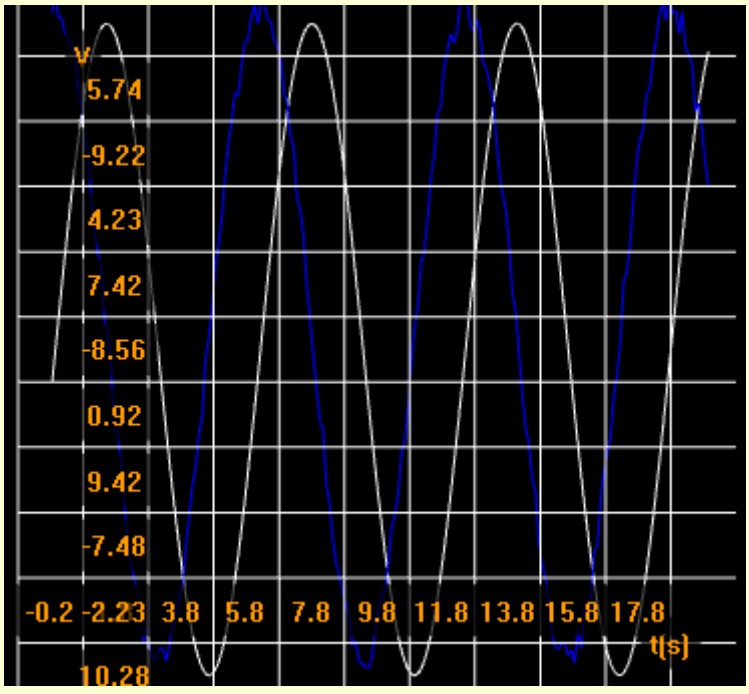
Very Nice !!





StripChart (by Flavio Codeco Coelho, License: GPL)

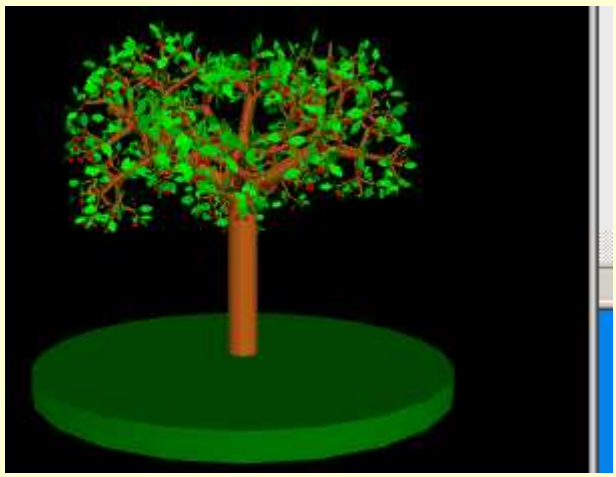
I think there are are much better widgets for stripchart recording, but to complete the overview of the possibilities of VPython, it's here.



Scene group

Alfabet (by Bruce Sherwood, December 2007, website)
(look_around.py) shows how to rotate a camera at the origin and look around at a surrounding scene.
This demo show how you can rotate the ring with characters, by left moues position. At the center, no movement, the further you move the mouse to the right the faster the ring will rotate.
Changed: allow user.spin

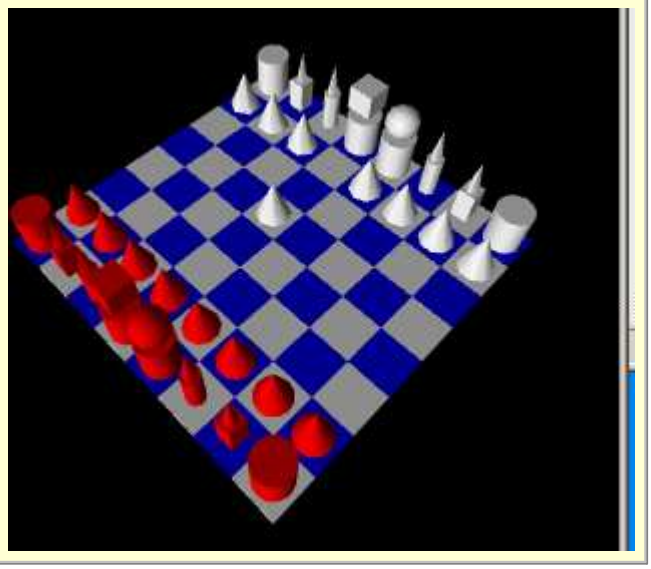
Cherry_Tree (by Viktor Ferenczi, januari 2003, website)
Very Beautiful !
It would be interesting to change the program in such a way, that after a while the cherries would really drop on the ground.



Chess (by Shaun Press, website)

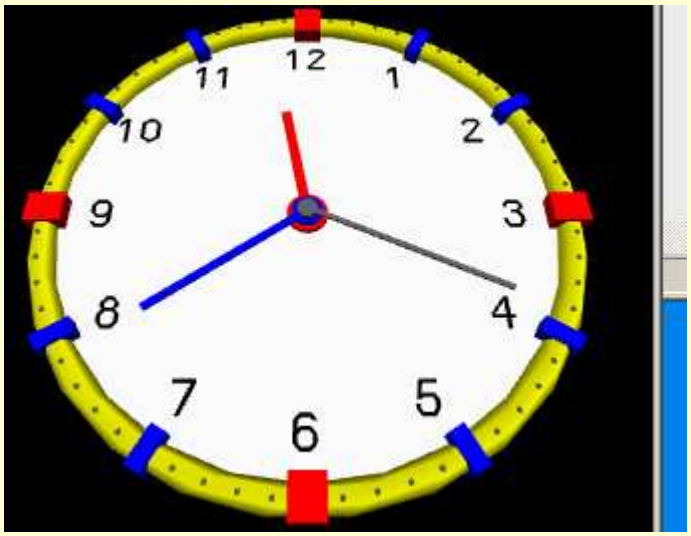
Chess board -- doesn't actually play but will execute moves:

move ('d2d4')



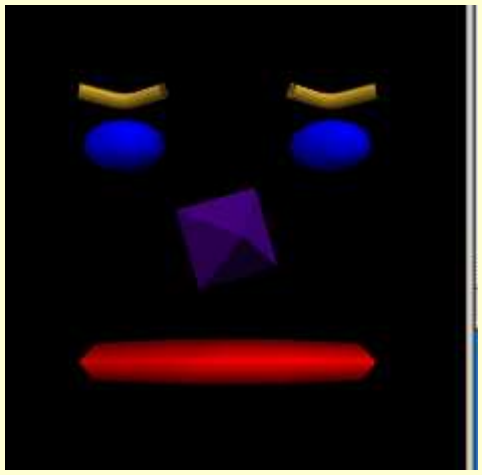
Clock (by Viktor Ferenczi, januari 2003, website)

problems when coming from another "clk" not known ??



Face (by Joel Kahn, 2004, website)

Makes an amusing moving face.

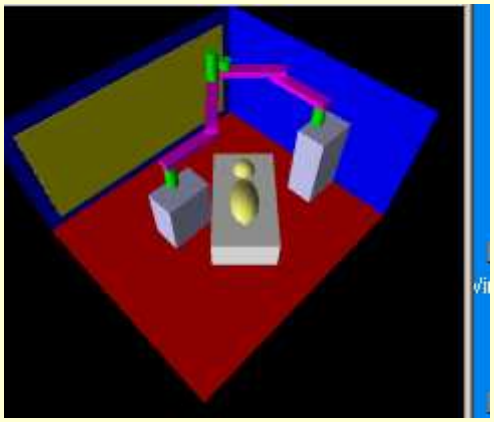


Fountain (??)

Very fast for so many balls !

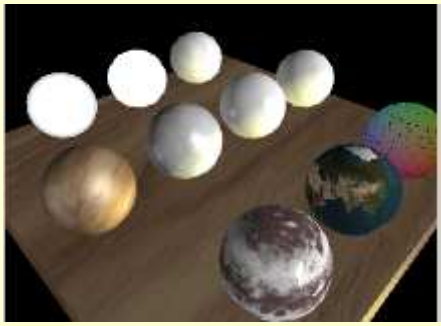
Intensive Care (by Stef Mientki, november 2008)

This demo was used to evaluate the positioning and dimensions of lifting arm systems for medical equipment in an Intensive Care unit.



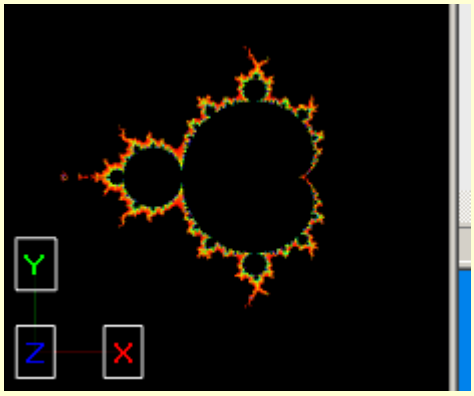
Material 5 (by ??) **VP-5**

Click to center one of the balls, and move your mouse around, because attached to your mouse there's a little light.



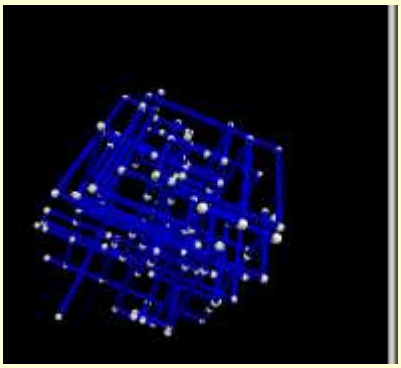
Mandelbrot1 (by Bruce Sherwood, jan 2008, website)

(pixelplot.py) shows a way to do pixel-oriented plotting in VPython



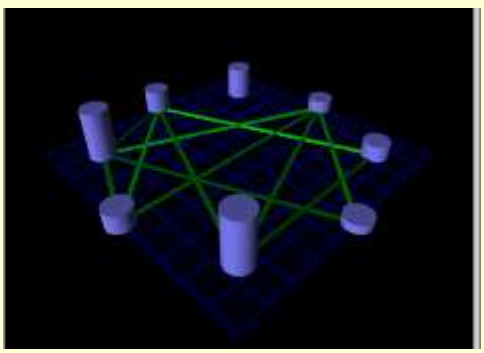
Pipes (by William Wright, ?, website)

A screen-saver-like animation of connecting pipes



Planar (??)

Move the cylinders around.

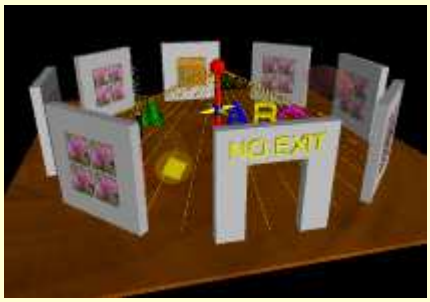


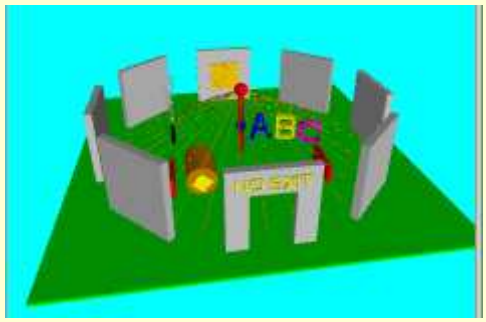
Saraiva (by Eduardo Saraiva, ?, website)

is a fun and attractive demonstration of various VPython capabilities.

Stonehenge (by Bruce Sherwood)
On the left the new VPython-5 version. Press to enter roaming mode, release to exit roaming mode. In roaming mode, with the mouse button down, move the mouse above or below the guide crosshairs to move forward or backward; right or left rotates your direction.

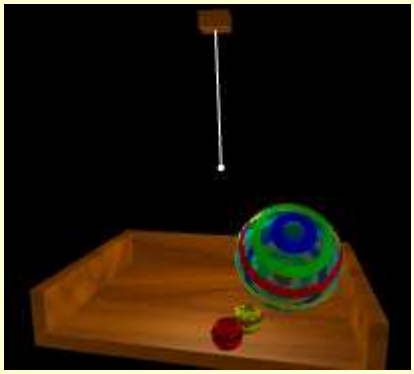
I don't understand how to toggle roaming mode and what it does exactly..





The VPython-5 demo (by Bruce Sherwood, August 2006)

Beautiful light \ shade \ transparency



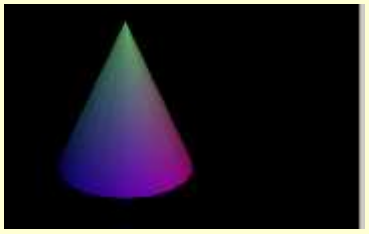
Shape group

Capitals (by Joel Kahn, 2004, website)

(capital_letters.py) displays capital letters using curves

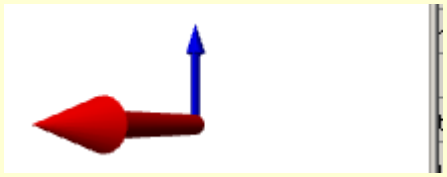
Cone (by David Scherer July 2001)

Example of use of faces object for building arbitrary shapes (here, a cone)



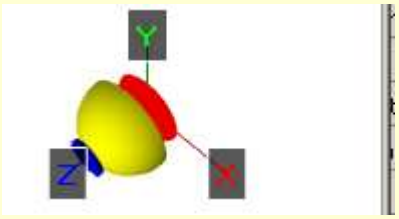
Cone_Arrow (by Aaron Miller, v0.1: 18Jul2008)

VPython carrow, Make a cylindrical-bodied arrow with a cone point of identical dimensions as the standard built-in square arrow.



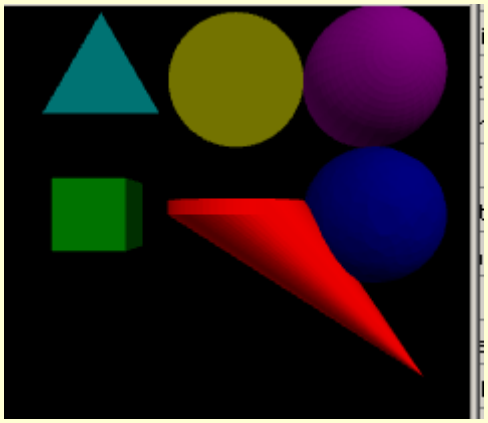
Cones (by Thom Ives, July 2002)

modified by Bruce Sherwood and David Sherer



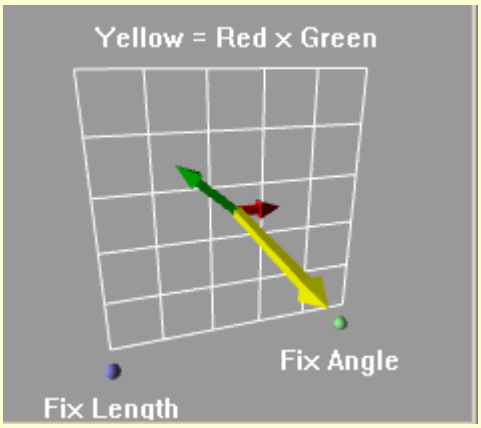
Convex (by David Sherer)

The red Cone follows the mouse position.



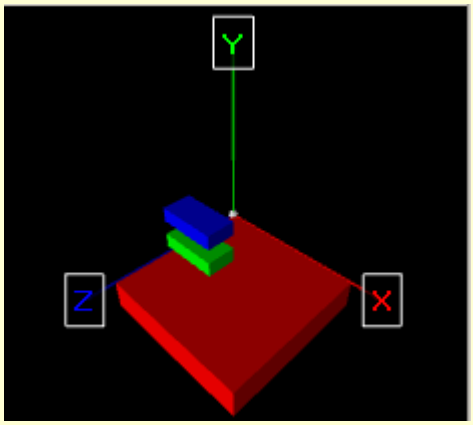
CrossProduct (??)

Vector cross product:
Red x Green = Yellow
Drag to change green vector
Click to toggle fixed angle or fixed length



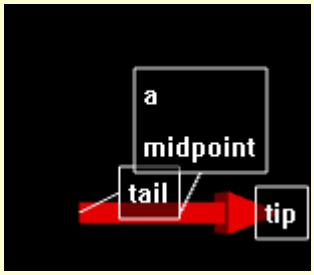
Frame_Test (by Stef Mientki, november 2008)

Just a testscript, to see how translations and rotations of frames are performed.



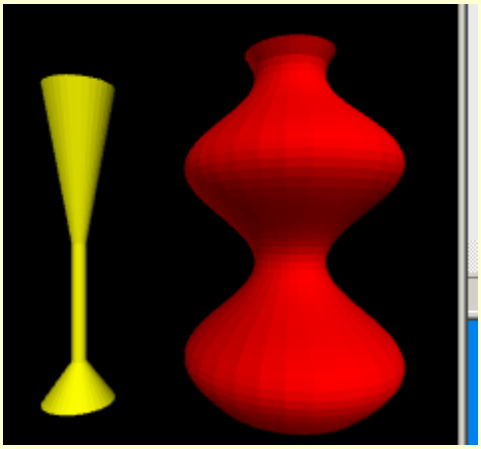
Labels (??)

You should see a single arrow with three labels attached to it. Each label is translucent. The one at the tip should be centered at the tip. The head and tail labels have a single line of text. The midpoint label has two.



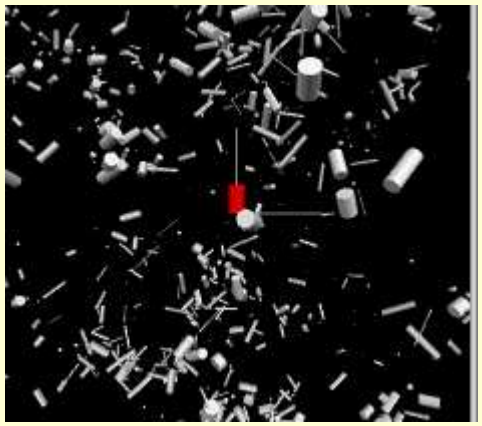
Lathe (by David Scherer)

Create a surface of revolution out of slices that are convex objects



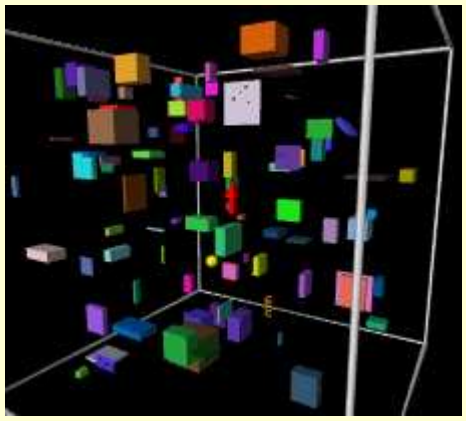
Random (??)

looks like space garbage to me.



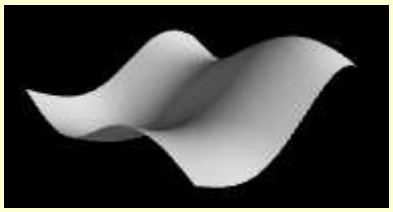
RandomBox (??)

this is mostly for experimenting with zooming and rotating



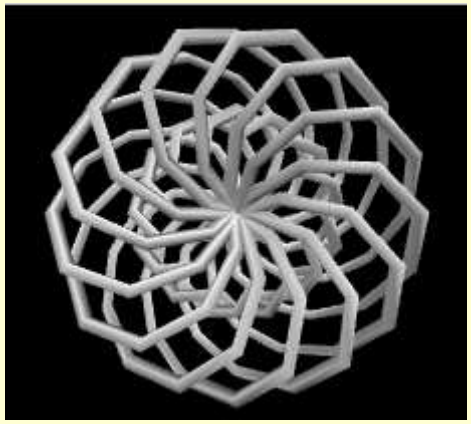
Shape-V5 (by David Scherer, July 2001) **VP-5?**

Demonstrates some techniques for working with "faces", and shows how to build a height field (a common feature request) with it.



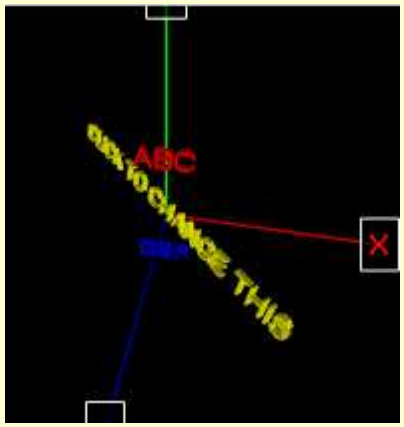
Spider (??)

readdata.py
This program is meant to be a simple example of how to read data from a file. It reads a plain ASCII text file containing three columns of numbers separated by spaces and treats the numbers in each line as a three dimensional (x,y,z) coordinate for a point on a weird knotlike 3D figure.



Text (by Bruce Sherwood, Carnegie Mellon University, begun March 2000)

Display extruded text (uppercase only at present). By default, display text along x axis, with letters pointing up parallel to y axis.



Unused Demos

- **PyGeo**, (by Arthur Siegel, who unfortunately died in 2007; write to Joel Kahn <jj2kk4@yahoo.com>, website) needs to change pickloop in vpyframe.py
- **Computation** using VPython of planetary ring formation (Michael Cobb). See FAQ for details on how the VPython displays were captured and then made into a video. No code available.
- **VPNBody** multibody solar system dynamics (Rodney Dunning), complete package must be unpacked
- **Physics projects** and links (Rob Salgado)
- **KineticsKit** a module for quickly building ball-and-spring models (Markus Gritsch)
- **Earth Science** a collection of Earth Science programs (Lensyl Urbano)

Changes individual programs

Pendulum group

Double

- lowered rate from 1000 to 50
- increased dt from 0.002 to 0.02
-

Chess (Author: Shaun Press)

- Setting forward, range
- Added convenience procedure "move"

Keppler (Peter Borcherds, University of Birmingham, England)

Scene group

Clock (Viktor Ferenczi, januari 2003, website)

- conflicts with spaces and tabs, converted all to spaces
- tripple quoted string at the start

Mandelbrot1 (by Bruce Sherwood, jan 2008, website)
(pixelplot.py) shows a way to do pixel-oriented plotting in VPython
problems with from __future__ import division

VP5 - VP3

- Although most of the scripts seem to run faster, Fountain runs a slower after a while VP3=12fps / VP5=8fps. Also Rubik_555 runs slower in VP5

- hide() must be replaced by visible = False, and show() by visible = True
-
-

march 2008

Signal WorkBench 

History

About a year ago (april 2007) I build this functionality in Delpi with embedded MatLab. After discovering SciPy, I replaced the embedded MatLab with embedded Python and used it with even greater pleasure than with the embedded MatLab. I then wrote:

"Although it was my first intention to create this program completely in Python, I had the feeling that my knowledge of Delphi would result in a much quicker result, especially because of the calculation tree and the signal displays. (Maybe the overwhelming collection of different graphical libraries in Python also forced this choose. At the moment I've the feeling that wxWidgets is the future way to go, but I've no idea yet if and what wrapper to use for wxWidgets.)"

Half a year ago I needed a GUI for another project, and decided to go for wxPython (Qt seemed better, but has a weird license). Unfortunately I didn't succeeded in getting one of the visual IDE's working, but by now I'm completely used to write GUI in plane ASCII, although I use several homemade convenience libraries. By now the Signal Workbench written as a Control / Brick within the PyLab_Works frame is almost identical to the Delphi version and at some points even better.

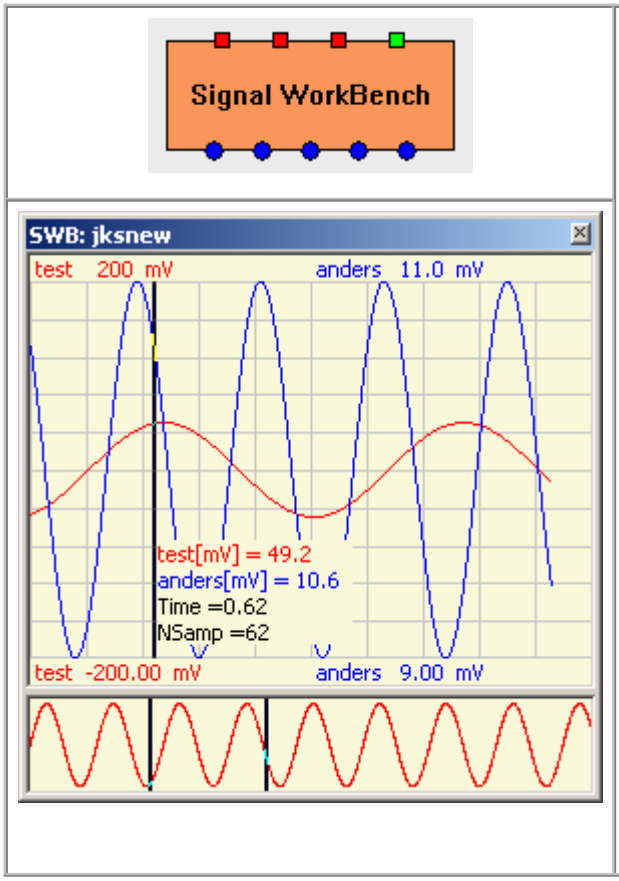
ToDo

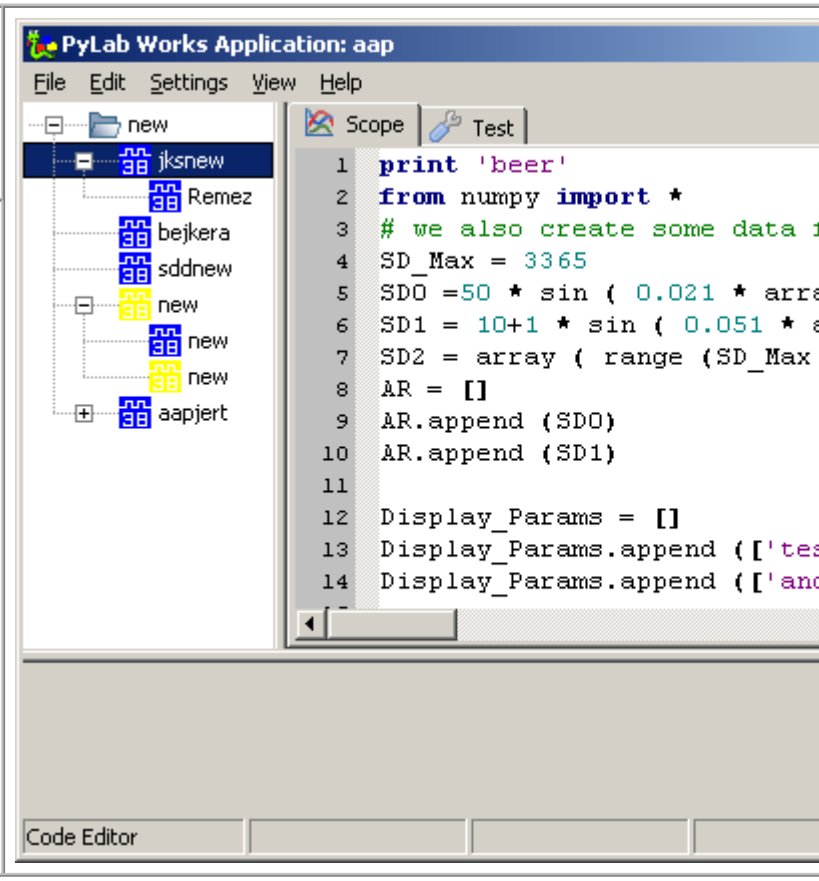
- function of the top node
- selection of visible signals through MM/RM on top/bottom labels (leave the bottom / top labels visible)
- time measurement, add history_start*history_compression, use own expansion/compression
- autoscaling of history signal not always to well ??

Introduction

The signal workbench is highly sophisticated graphical calculator, specially equipped for off-line analysis of time series and development of algorithm for real-time analysis.
The main items of the signal workbench are:

- the **calculation tree**, giving a graphical presentation of the calculation flow
- the **code window**, where the pre-processed python code can be edited
- the **signal displays**, to visualize, compare and measure the final and intermediate calculated results



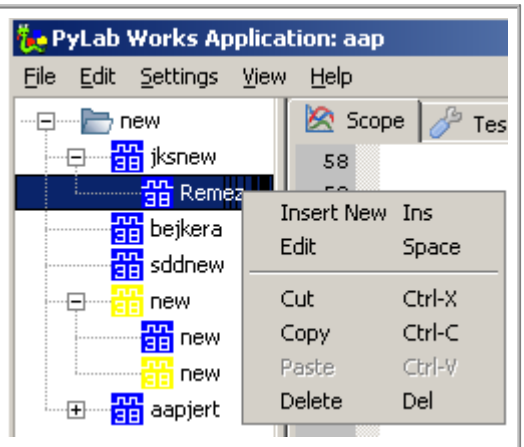


Calculation Tree

The global view of a total calculation is represented by a tree, where each node in the tree performs a (small) part of the calculation. The tree is fully synchronized with the editor on it's right side, and leaving a node, automatically saves any changes to the text in the editor of the previous node. You can either run a single node (by pressing **F9**), or run the current node with all branches below it (by pressing **shift-F9**). The nodes in the tree can dragged and dropped, nodes can be copied / deleted including all their children. You can also put a general library (under construction) in a branch that's not executed, so the Signal WorkBench will serve as a nice editor.Don't use the top node at the moment, because I've special intentions with it.

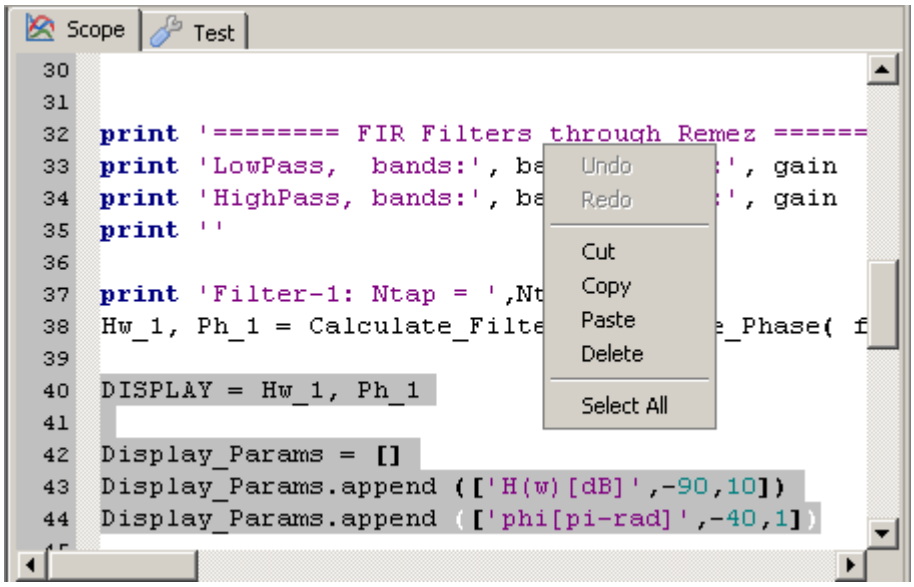
Each node has an icon, clicking on the icon, toggles between 2 images, indicating if the output goes to the graphical display or not.
For the following functions, there's a difference if you click the node text or the node icon: copy / insert / drop. When the action is performed on the icon, the new or moved item will be placed as the last child of that node, while the action on the text of a node, does place it just above the clicked node.

Editing the name of the node (and thereby the filename where the code is stored) can be done by selecting and clicking again (not double clicking) or by select and space or by the right mouse menu. Leaving the edit, can be done by Enter-key or by selecting another node or something else.



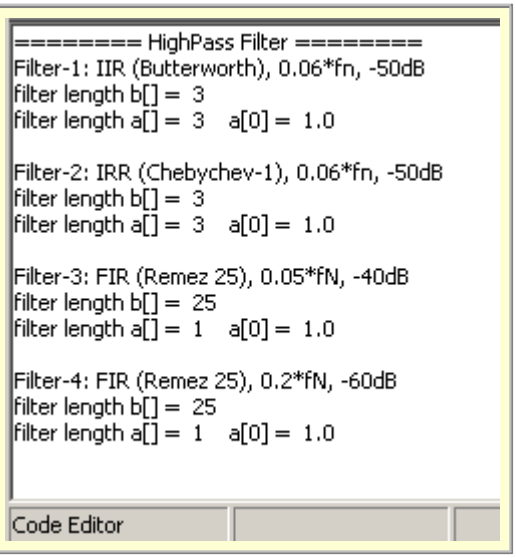
Code Editor

Each node in the calculation tree, is connected with a piece of Python code. Let's take a look at a simple example below:



The variable **"DISPLAY" in line 40**, tells which signals should be ported to the graphical display (in case the tree icon allows it). The following signals can be ported to the graphical display: **tupple, list, 1-dim array, 2-dim array or any combination of these types**. It isn't even necessary to make all elements equally length, Signal WorkBench will take care of that.
The variable list **"Display_Params" in line 42..44** tells the graphical display the parameters of each of the signals. The parameters consists of the name of the variable (where you can specify the units of that signal between square brackets), the value at the lower side of the display and the value at the upper border of the display. This variable doesn't need to be present, and you're also allowed to specify parameters for the first set of parametersor just parts of the parameters, e.g. only the signals names.
Through **F7**, you can popup the **code snippet manager**, which is fully integrated,

Compiler Errors
When a compiler error occurs, the Python error message is captured in the lower interactive editor (see picture below) and the cursor automatically jumps to the line with the error in the upper editor window. Note that the line numbers in the error message doesn't need to be the same as the line-number in the editor, this is because the script is pre-processed and so there are some extra lines added. You can always jump to the error, by clicking on the line in the lower interactive editor with the left mouse button, or by pressing Enter, when cursor is on the line "#<< File "<string>", ..."



Signal Displays

The signal displays are highly automated, while leaving great flexibility both by programming and user manipulation. Signals can be viewed, zoomed and measured. The ability to make one or more signal displays transparent, gives a unique feature to overlay several signal displays for easy comparison. The signal displays are quit light-weighted, so you can have a lot of them open simultaneous.
Each signal display consists of 2 graphical windows, the bottom window (also called time-base window),

holding the complete time-history of one selected signal, meant for navigating and selecting along the time-dimension. The top window, holding all (or a selected set) signals over a (possibly) limited time segment, is meant for zooming into details and for doing exact measurements.
The following items are used to control the appearance of the signal display:

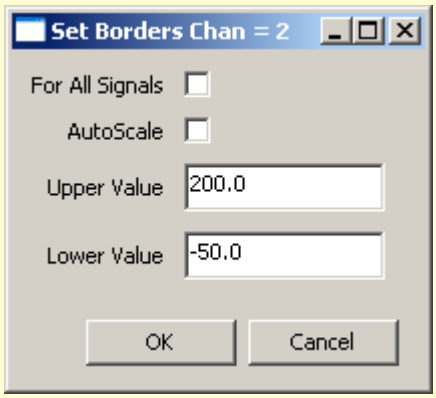
- LM, clicking in an image will draw (and drag) a cursor, clicking on a label will show and let you modify the properties
- MM, clicking in an image will draw (and drag) the other cursor. (We can't use the RM, because it;s used of normal popups.
- RM, will popup a menu
- splitter can be moved to set the vertical size of the time-base window
- form drag&drop, form-resize
- (shift) arrow left/right, page up/down for moving the cursors

Signal Selecting

through the RM-menu in the upper window you can select which signals are visible or not. Labels are automatically shown, with as much information as possible. Which part of the visible signals along the time-axis will be shown, is determined by the 2 cursors in the time-base window (showing the complete time-axis), which can be placed by LM and removed by MM (or RM-menu).
Through the RM-menu of the time-base window or by clicking on the signal label in the time-base window, you can select the signal that's used in the time-base window. The color of the signal in the time-base window is the same as the signal color in the main-window. The amplitude scaling of the signal in the time-base window is always automatic.

Signal Appearance

By clicking on one of the lower or upper signal labels, the border values of the selected signal or all signals can be changed.

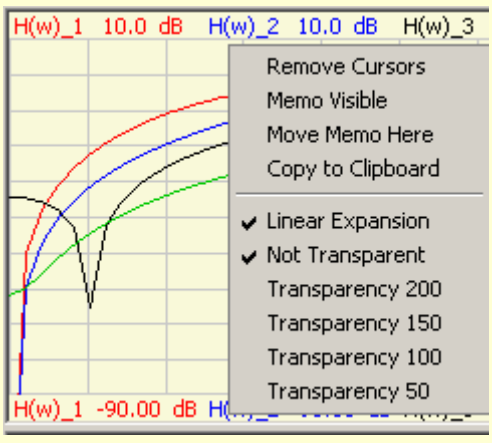


Transparency

Each signal display can be made transparent for easy comparing different algorithms. Transparency can be set through the RM-menu of any of the signal windows. Transparency only works for windows-2000 and up.

The memo, displays a self expandable memo, to make some notes if you want to save the graph window.

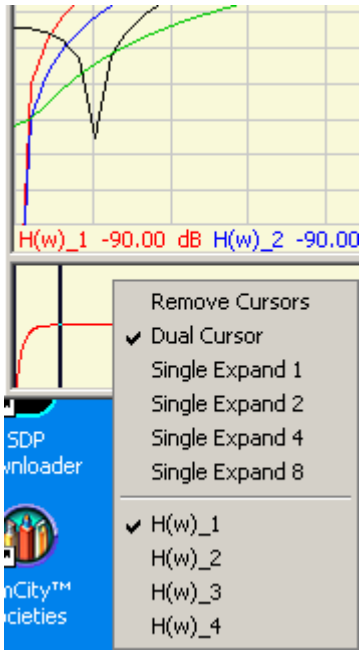
Linear expansion, does draw the expanded signal with linear interpolation, otherwise the signal is drawn stepwise (zero-order hold).



History, RM-menu

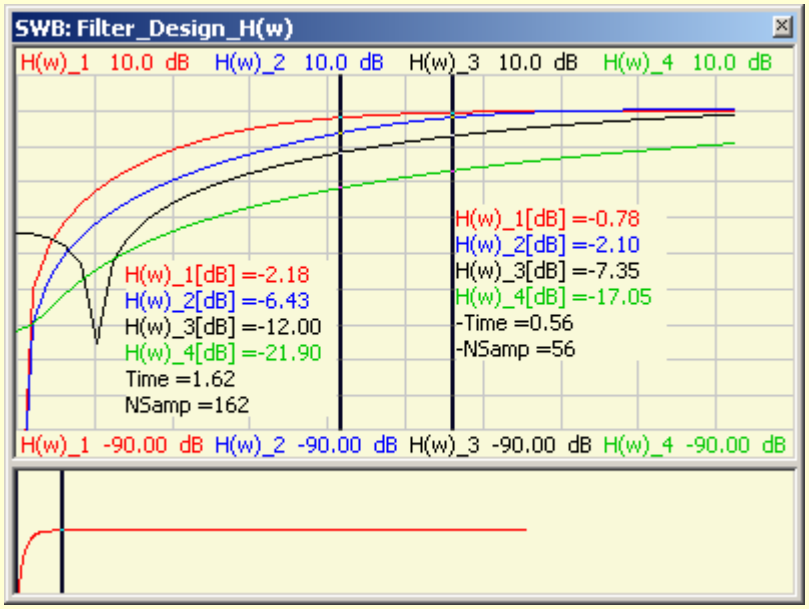
Remove Cursors, can also be done by clicking on the right of the visible signal.
Dual Cursor, gives two cursors (both through the RM), which select the part of the signal which will be displayed in the normal window (top window).
Single Expand 1, gives just 1 cursor, that displays exactly all the samples of the signal, starting at the cursor position.
Single Expand 2, ditto, but signal is expanded 2 times.

The lower items in the menu are the available signals, of which one can be selected to be shown in the history window.



Cursor Measurement

With the left-MB in the main signal window, you can position 2 cursors in the upper signal window for measurement. Removing of the cursors is done through the middle-MB or the RM-menu. Each cursor has a panel attached, in which the actual real-world values of the visible signals are shown. The vertical position of the panels is determined by the mouse cursor. The left panel will also show the startpoint of the signals, both in samples and in (mili-)seconds. The right panel will also display the distance between the 2 cursors both in samples and in (mili-)seconds. When the signal window has focus, the position of the cursors can be fine-tuned with the left-arrow and right-arrow key for the left cursor and shift-left-arrow and shift-right-arrow key for the right cursor. Larger steps can be made by the page-up and page-don key (+shift for the right cursor)



Time Selection

The two top-items of the RM-menu are designed to select certain time regions for further calculations. With "Selection Set", a new marker is placed on the time-axis, toggling the selection on or off. In the figure above the dark green lines at the bottom, are the selected regions. Removing a toggle point is also done through the RM-menu.
If the selection is used in the next calculation is furthermore determined by the calculation function itself.

march 2008

Filter Design

Introduction

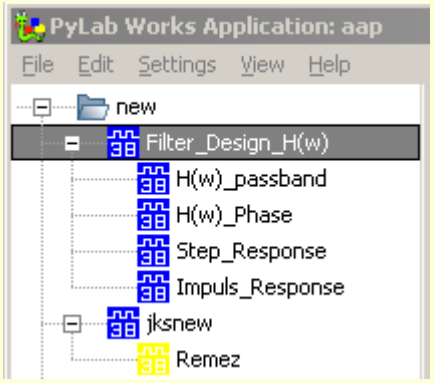
This demo shows how you can easily compare a number of different filter designs with Signal WorkBench. This document also shows, how terrible beautiful, but clumsy, we did filter design in the past with MatLab. The good thing about MatLab is, they have a very extensive and reasonable (yes it contains a lot of errors) good online book (free), covering the Signal Analysis package, which looks reasonable identical to the possibilities of SciPy.
In fact generating this document was the first real-life test of Signal WorkBench, so some of the examples and pictures shown on this page, might be premature or contain small in-perfections.

Goal

Suppose we want to design a filter and compare several filter-designs with each other. We not only want to compare the results like frequency response and impulse response but also we want to see and compare the effect of the different filters on the real signal. Let's say we want to compare a maximum of 4 filters at a time, and we want to compare frequency response (separated in amplitude and phase, and both in overall view and passband details), impulse response (overall and detail) and of course the real signal response (and maybe we also want step response). Now there are several ways to accomplish this test setup with Signal WorkBench, but the most obvious solution is to create one graphical window for each parameter (where total overview and detailed view will be placed in separate windows). So this will lead us to a calculation tree, which might look something like this:

The **top node** contains a reference to the **real signals**, in this case the signals are read from a file.
The **second node** is the **start of the filter calculations**, and will generate the overall view of the amplitude-frequency response. All other graphical views are created as a child of the filter start-node. The reason for this is simple, we now can save the filter start-node as a **template** and reuse the complete start-node (with all its children), whenever we need it.

The nodes with the blue icons will show their output signals, the nodes with the yellow icons, will not popup a graphical window. Toggling graphical view on/off can simply be done by clicking on the icon.



Filter parameters

We'll use Scipy.Signal naming as much as possible, but here and there we'll try to be a little more consistent.

- **wp** = passband frequency, expressed as a fraction of the Nyquist-frequency (= half the sample frequency)
- **ws** = stopband frequency [fraction of Nyquist-frequency]
- **gp** = maximum passband deviation [dB]
- **gs** = minimum stopband attenuation [dB]

```
from signal_workbench import *
import signal_workbench
reload ( signal_workbench )

#Test_Signal = Better_ECG
# Example of Remez HighPass Filter design
fsamp = 100     # sample frequency [Hz]
fs    = 1.0     # stopband corner [Hz]
fp    = 5.0     # passband corner [Hz]
att_s = 40.0    # stopband attenuation [dB]
att_p = 0.0     # passband attenuation [dB]
Ntap  = 70      # number of filter taps
                #   must be ODD for bandpass filters
                #   must be EVEN for differentiators

wp = fp / fsamp
ws = fs / fsamp

gp = power ( 10.0, -( att_p / 20 ) )   # 10.0 is essential for accuracy
gs = power ( 10.0, -( att_s / 20 ) )

bands_lp = ( 0, 0.5-wp, 0.5-ws, 0.5 )
bands_hp = ( 0,     ws,     wp, 0.5 )

gain_lp = ( gp, gs )
gain_hp = ( gs, gp )
```

Filter Start-node

As we should have only 1 node, where all essential code should be located (for easy manipulation), this will become the filter start-node. All child nodes just serves as a place holders for the graphical views (but can contain also non-essential code). Below the code for the filter start-node is given, for simplicity only one of the four filters is defined here.
So the code below will calculate 4 filters, calculate the amplitude and phase of the transfer function and print out an overview of the filter paramaters:

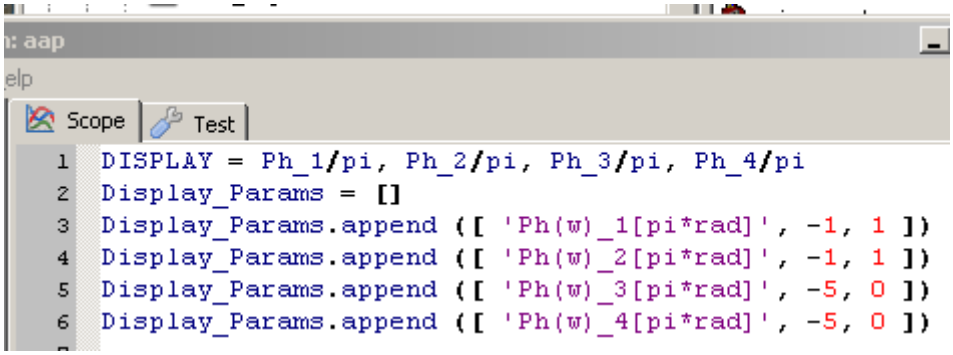
```
print '======== HighPass Filter ========'
print 'Filter-1: IIR (Butterworth), 0.06*fn, -50dB'
#filt_3 = signal.cheby1(N, 1, Wn, btype='high')
filt_1 = signal.iirdesign( 0.06, 0.002, 1, 50, 0, 'butter')
Hw_1, Ph_1 = Calculate_Filter_Amplitude_Phase( filt_1 , 1)

print 'Filter-2: IRR (Chebychev-1), 0.06*fn, -50dB'
N,Wn = signal.cheb1ord(0.06, 0.002, 1, 50) #wp, ws, gp, gs)
filt_2 = signal.cheby1(N, 1, Wn, btype='high')
Hw_2, Ph_2 = Calculate_Filter_Amplitude_Phase( filt_2 , 1)

print 'Filter-3: FIR (Remez 25), 0.05*fN, -40dB'
#remez(numtaps, bands, desired, weight=None, Hz=1, type='bandpass', maxi
filt_3 = signal.remez(25,(0,0.01,0.05,0.5),(0.01,1)) #,Hz=2)
filt_3 = ([ filt_3, ones(1)])
Hw_3, Ph_3 = Calculate_Filter_Amplitude_Phase( filt_3 , 1)

print 'Filter-4: FIR (Remez 25), 0.2*fN, -60dB'
filt_4 = signal.remez(25,(0,0.001,0.2,0.4),(0.001,1),(1,1)) #,Hz=2)
filt_4 = ([ filt_4, ones(1)])
Hw_4, Ph_4 = Calculate_Filter_Amplitude_Phase( filt_4 , 1)
```

Now we can use this node also to display the amplitude of the transfer function, like this:

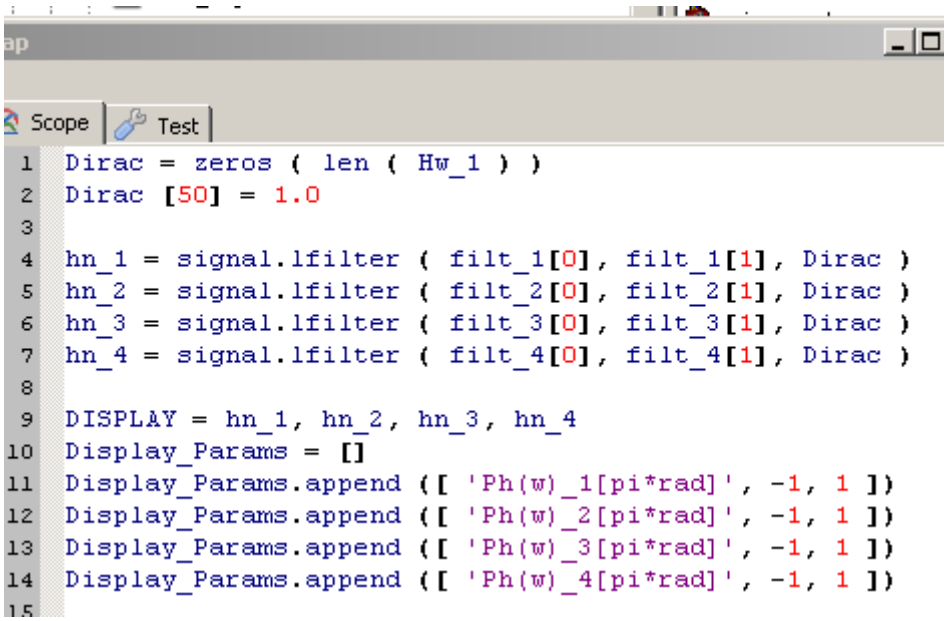
```
81  DISPLAY = Hw_1, Hw_2, Hw_3, Hw_4
82  Display_Params = []
83  Display_Params.append ([ 'H(w)_1[dB]', -90, 10 ])
84  Display_Params.append ([ 'H(w)_2[dB]', -90, 10 ])
85  Display_Params.append ([ 'H(w)_3[dB]', -90, 10 ])
86  Display_Params.append ([ 'H(w)_4[dB]', -90, 10 ])
```

Other filter nodes

Most of the other nodes don't contain any code, just the definition of the signals to be displayed.
So for the "H(w) phase" node we have:



For the impulse response node, we can do some parameterless calculations (so the filter start-node will have as least code as possible). Of course we have to define the output signals, to be shown in the graphical view. (Note, we didn't use the function signal.impuls here, because the version of signal we used, during writing this page, crashed. Probably this is solved in the newer version.)

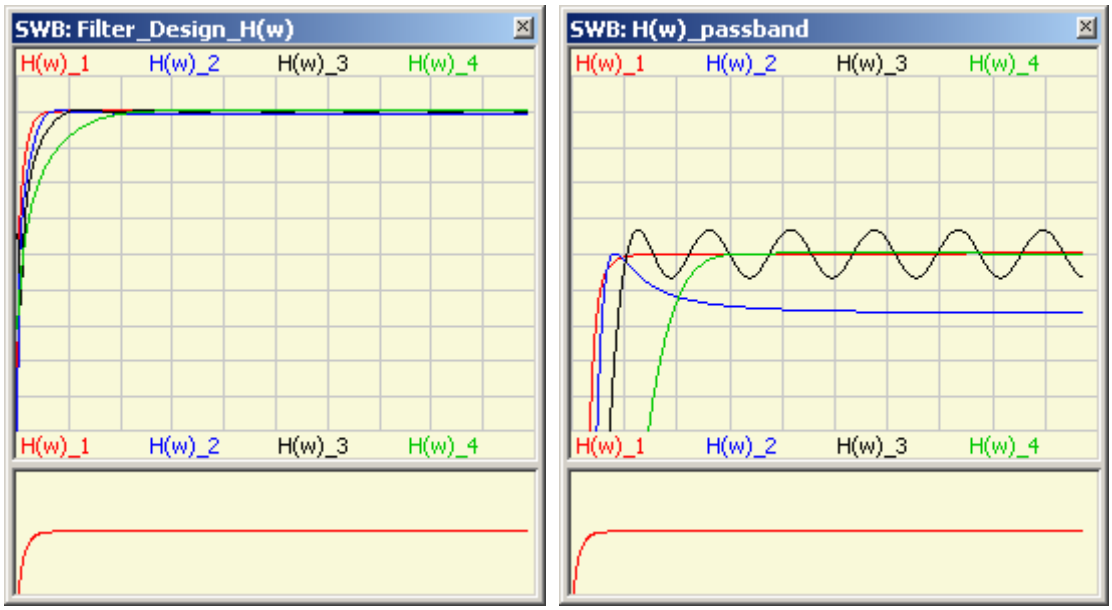


Action

After defining all the display nodes (and save the start-node as a template for future use), select the filter start-node in the calculation tree, press Shift-F9, and watch ...
... we get a nice overview of the design parameters of the different filters,
... and all graphical views we've defined will popup with their specific signals
all with just 1 click !!
After you've initialized all your graphical windows, just edit 1 or more parameters in the filter start-node, press Shift-F9 and you'll see the effect of your parameter change in all graphical windows.

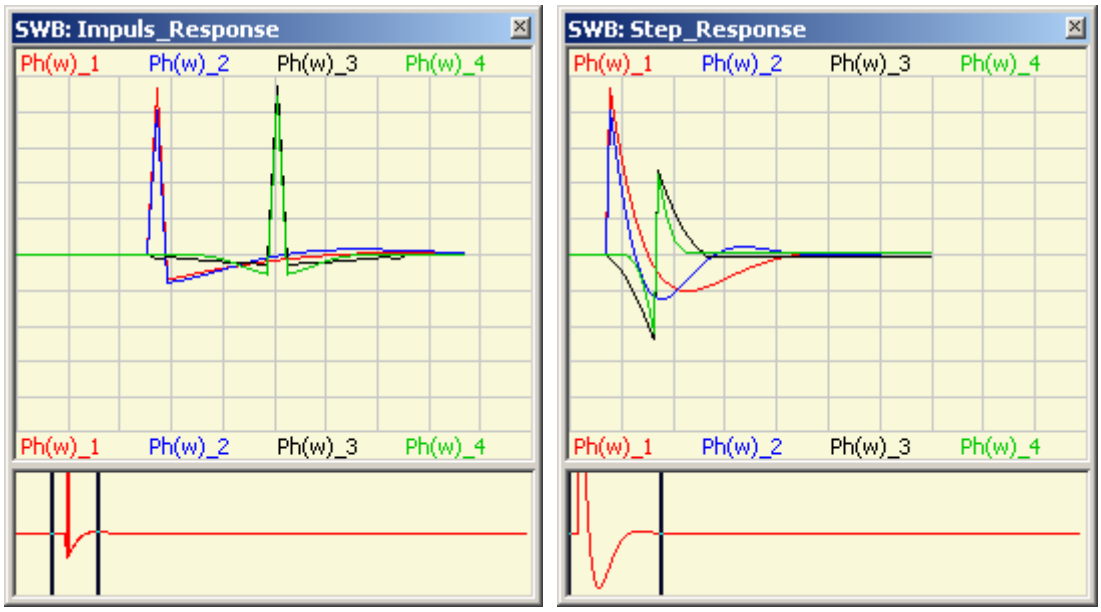
```
************  RUN SCRIPT    Filter_Design_H(w)
======== HighPass Filter ========
Filter-1: IIR (Butterworth), 0.06*fn, -50dB
filter length b[] =  3
filter length a[] =  3    a[0] =  1.0

Filter-2: IRR (Chebychev-1), 0.06*fn, -50dB
filter length b[] =  3
filter length a[] =  3    a[0] =  1.0

Filter-3: FIR (Remez 25), 0.05*fN, -40dB
filter length b[] =  25
filter length a[] =  1    a[0] =  1.0

Filter-4: FIR (Remez 25), 0.2*fN, -60dB
filter length b[] =  25
filter length a[] =  1    a[0] =  1.0

************  RUN SCRIPT    H(w)_passband
************  RUN SCRIPT    H(w)_Phase
************  RUN SCRIPT    Step_Response
************  RUN SCRIPT    Impuls_Response
```

The picture below shows the **amplitude response** (total overview on the left, passband detail on the right) of the 4 filters. The nicest response is from the first (red) filter, but judged from the amplitude response only, they are all not bad.

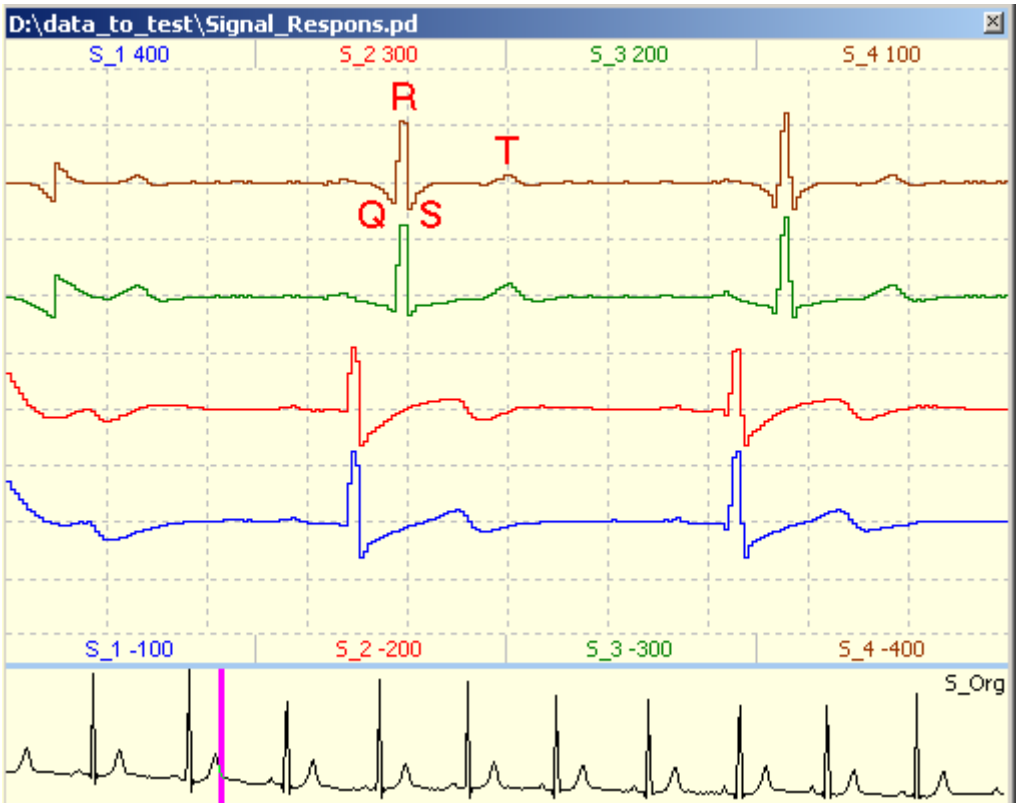


The picture below shows the **phase response** on the left (notice the linear phase characteristic of FIR filters 3 and 4.
The black phase curve, shows an artifact at the start, probably due to bad "unwrappping" of the phase angle.

The picture below shows the **pulse response**, again overall view on the left (time is expanded) and), and the **step response** on the right (time expanded). They all look good, if we discard the extra time-delay of the FIR filters 3 (black) and 4 (green).



And finally the most important view: the **signal response** (time expanded). The black signal at the bottom shows the original input signal. The start of the original signal clearly contains a low frequency component, which we want to get rid off. The filtered signals are time-expanded, so we can take a good look at the first part of the signal.
(this picture is taken from the Delphi version of Signal WorkBench)



The signal is a recording of a human ECG (Electrical signal recorded from the hart), and in this case we're interested in determining the exact locations in time of Q,R,S,T peaks as shown in the brown signal. Because this is a non-stationary signal, both the FIR filters (brown and green) perform much better in preserving the shape of the signal.

FIR, Remez exchange

Filterlength must be odd to get a gain of 0dB at the Nyquist frequency !!

remez(numtaps, bands, desired, weight=None, Hz=1, type='bandpass', maxiter=25, grid_density=16)

Scipy.Signal

The SciPy Signal library contains the following filter design algorithms

Filter design:

```
remez          --  Optimal FIR filter design.
firwin         --- Windowed FIR filter design.
iirdesign      --- IIR filter design given bands and gains
iirfilter      --- IIR filter design given order and critical frequencies
freqs          --- Analog filter frequency response
freqz          --- Digital filter frequency response
```

Matlab-style IIR filter design:

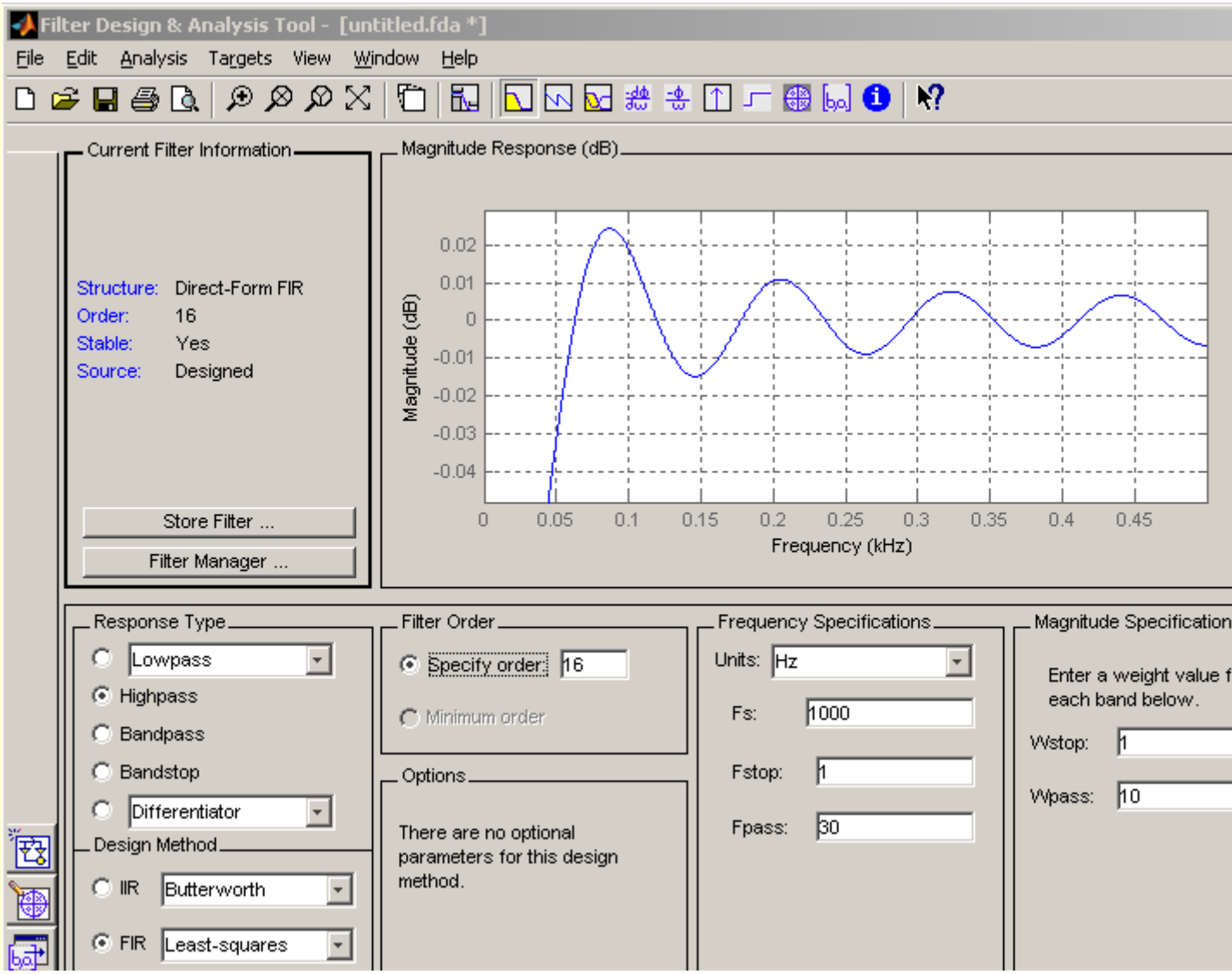
```
butter (buttord)  -- Butterworth
cheby1 (cheb1ord) -- Chebyshev Type I
cheby2 (cheb2ord) -- Chebyshev Type II
ellip (ellipord)  -- Elliptic (Cauer)
bessel            -- Bessel (no order selection available -- try butterod)
```

How we did it with MatLab

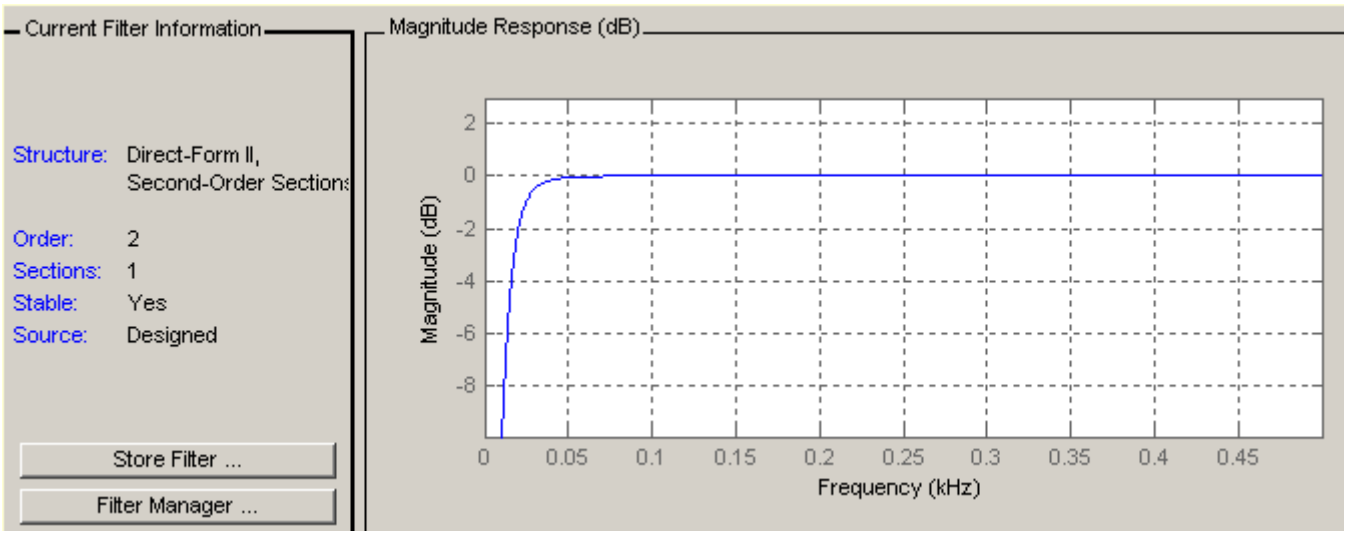
For years we've used MatLab (Signal Toolbox) to design filters, see in the picture below how terrible nice it looks. But although the generated filter coefficients are correct, it's a nightmare to optimize and compare filters with this MatLab tool.

Every time you change the filter type, sample frequency is lost, other edit boxes appear, run button is often dimmed, and for each filter MatLab chooses the scaling of the figure (compare the 2 figures). Storing a set

of coefficients in your own directory, is a crime, MatLab always starts at it's own directory.
Knowing we couldn't change this behavior, we accepted this tool for years.
Now knowing Python, we can influence the behavior and we're very happy in using this new tool..



Comparing it with another filter design, argh MatLab itself chooses it's scale !!



march 2008

FIR Filters 

Introduction

This page just contains my personal notes on filters, using SciPy (coming from MatLab).

FIR filters

+ **Positive features**
+ non recursive FIR filters are inherently stable
+ roundoff noise can easily made small for non recursive FIR filters
+ can have perfect linear phase = constant group-delay = no distortion
- **Negative features**
- needs long filter lengths
- if used with an even number of taps, delay is not an integer number of samples and H(Nyquist)=0 !!

Linear phase

FIR filters having linear phase, can be divided into 4 types, depending on odd/even number of taps and symmetrical/anti-symmetrical, as defined by Rabiner and Gold in "Theory and application of digital signal processing". Some types have restrictions at the boundaries at w=0 and w=Nyquist, which can lead to serious problems when used for the wrong goal.

Type	Ntap	Symmetry	H(0)	H(Nyquist)	Suited for
I	odd	Symmetrical			band filters
II	even	Symmetrical		H(Nyquist) = 0	band filters
III	odd	Anti-Symmetrical	H(0) = 0	H(Nyquist) = 0	diff / Hilbert
IV	even	Anti-Symmetrical	H(0) = 0		diff / Hilbert

Note: MatLab help files and the Signal Toolbox manual contains many errors (about this matter)
Note: MatLab uses the filter-order, which is one less than Ntap

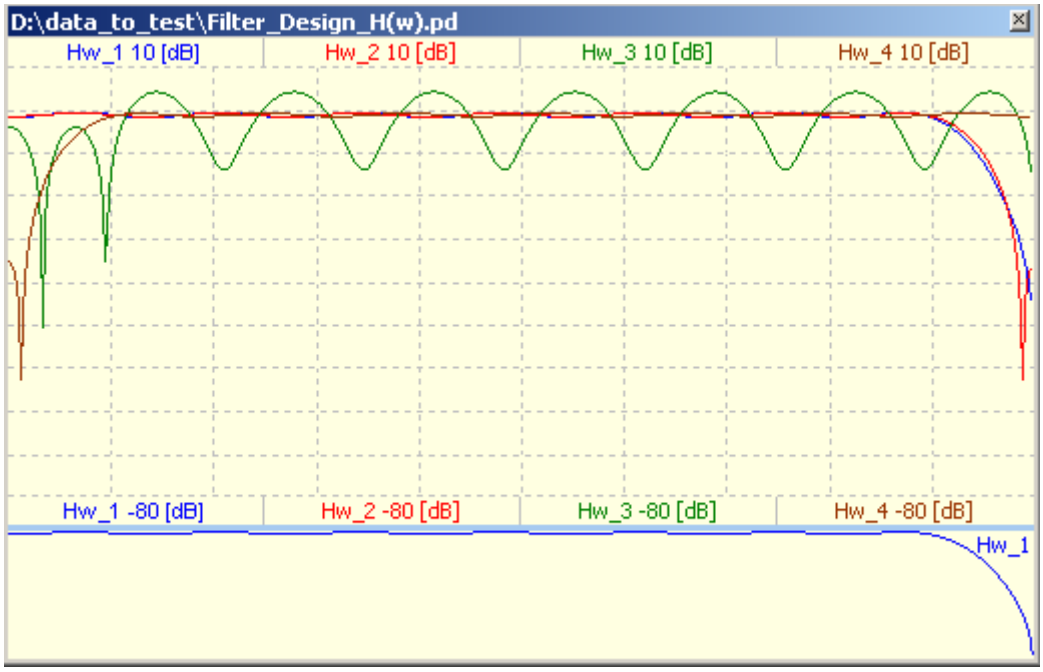
Effect of EVEN number of taps

For bandpass / bandstop filters problems arise when using an even number of taps, while the desired frequency response at Nyquist is not zero. For the differentiator type filter this exact the opposite, an odd number of taps might give problems.

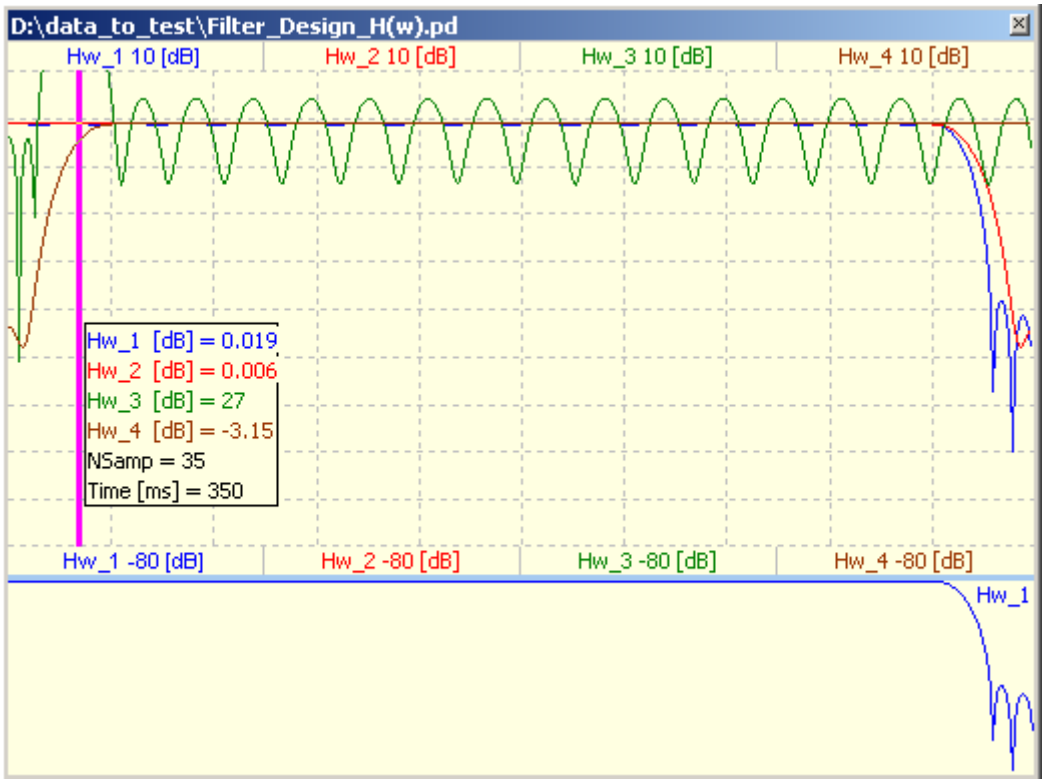
The picture below shows the amplitude of the transfer function (in dB) of 4 different filters, all designed with remez, and having the same band and gain factors.

- filter 1 (blue) : LowPass with 30 taps
- filter 2 (red) : LowPass with 31 taps
- filter 3 (green) : HighPass with 30 taps
- filter 4 (brown) : HighPass with 31 taps

As expected for the LowPass filters 1 and 2 (red and blue), it doesn't matter whether you specify an odd or even number of taps. For the HighPass filter it makes a lot difference, the brown filter with an odd number of taps works as expected, while the green filter, with an even number of taps is very bad.
Note: the red lowpass filter is the exact mirror of the brown highpass filter.



It will even be come worse if we increase the number of taps to 70/71, as can be seen in the picture below. Watch the exorbitant gain of 27 dB at 35 / 512 * fNyquist.



FIR by Remez exchange algorithm

This algorithm was developed and published by E.Ya.Remez in 1934.
T.W. Parks and J.H.McClellan wrote and published a Fortran program using the Remez exchange algorithm in 1973. That's why this algorithm is sometimes called the Parks-McClellan algorithm.
See also: http://mathworld.wolfram.com/RemezAlgorithm.html

remez (Ntap, Bands, Gains, weight=None, Hz=1, type='bandpass', maxiter=25, grid_density=16)

Calculates the minimax optimal linear phase filter (equiripple) using Remez exchange algorithm.
Ntap -- desired number of taps
always use **ODD** number of taps for **type='bandpass'**
always use **EVEN** number of taps for **type='differentiator'**
Bands -- monotonic increasing vector [0 .. 0.5], containing the band edges.
Gains -- vector specifying the gain in stop- and pass-bands (thus half as long as Bands)
weight -- optional vector specifying the weight of every gain-element, used in the optimization procedure
type -- '**bandpass**' : flat response in bands or
'**differentiator**' : frequency proportional response in bands.
maxiter -- maximum number of iterations in the optimization procedure
grid_density --
return -- vector, specifying the numerator of the filter

MatLab: firpm (Ntap-1, Bands, GAIN, weight, type)
Note: in MatLab the filter-order is specified, which is one less than the number of taps !!
Note: in MatLab the GAIN is specified at the band-edges, instead of midpoint of the bands
Note: SciPy doesn't support Hilbert

Examples

```
from signal_workbench import *
import signal_workbench
reload ( signal_workbench )

#Test_Signal = Better_ECG
# Example of Remez HighPass Filter design
fsamp = 100     # sample frequency [Hz]
fs    = 1.0     # stopband corner [Hz]
fp    = 5.0     # passband corner [Hz]
att_s = 40.0    # stopband attenuation [dB]
att_p = 0.0     # passband attenuation [dB]
Ntap  = 70      # number of filter taps
                #   must be ODD for bandpass filters
                #   must be EVEN for differentiators

wp = fp / fsamp
ws = fs / fsamp

gp = power ( 10.0, -( att_p / 20 ) )    # 10.0 is essential for accuracy
gs = power ( 10.0, -( att_s / 20 ) )

bands_lp = ( 0, 0.5-wp, 0.5-ws, 0.5 )
bands_hp = ( 0,      ws,      wp, 0.5 )

gain_lp = ( gp, gs )
gain_hp = ( gs, gp )
```

july 2008

Auto Generated Library Overview 

With the function ***Generate_HTML_Library_Overview*** ('NL', True) in ***PyLab_Works_search_bricks***,

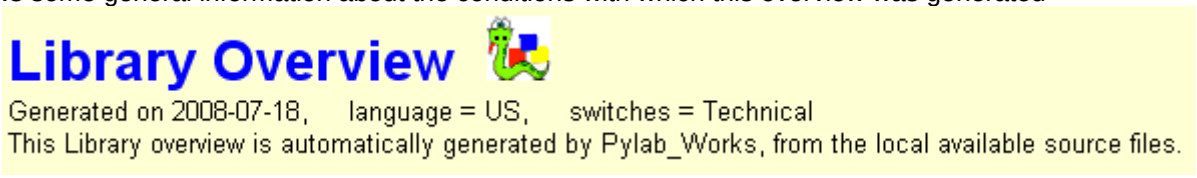
you generate a library overview (in any available language) of all installed Bricks on the local machine. The first parameter should be a valid language ID, the second parameter indicates a technical or more user-like overview (False = default).

This procedure should be implemented in the main menu of PyLab_Works (with a dialog for filename and language).

Below are shown the elementary parts of the generated doc. These images are taken from different doc files, sometimes in English, sometimes in Dutch.

Header

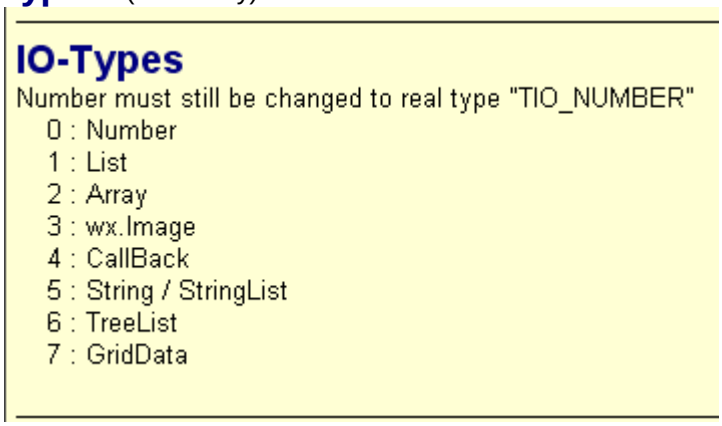
Shows some general information about the conditions with which this overview was generated



Library Files Overview

Shows a list of all library files and for each library the available languages

IO-Types (Tech only)



GUI Controls (Tech only)

Library Description

---

brick_Math

Bibliotheek met basis rekenfuncties
Icon = brick_math_icon.PNG
Color = (200, 0, 200, 255)

Extended Version Information (Tech only)

version: 0.4 date: 10-02-2008 author: Stef Mientki
Test Conditions:
WinXP-SP2, Python 2.4.3, wxPython wx-2.8-msw-ansi
WinXP-SP2, Python 2.5.2, wxPython 2.8.7.1 (msw-unicode)
- scaling and offset is set for each individual signal
- Time history window for 1 signal with min/max display
- set attributes for all signals at once
- border values on top and bottom
- second measurement cursor

version: 0.3 date: 29-01-2008 author: Stef Mientki
Test Conditions:

Technical Library Description (Tech Only)

Technical Description

```
#*****************************************************************************
# control_scope.py : a fast plot library for real-time signals.
# This plot library is more like a oscilloscope than like MatPlot.
# This library should be faster than MatPlot, PyPlot and FloatCanvas.
# This library has some extra features:
# - measurement cursor
# - no scaling in X-axis, so you'll see all samples
# - storage all settings in a ini-file
# - scaling and offset is set for each individual signal
# - Time history window for 1 signal with min/max display
# - set attributes for all signals at once
# - border values on top and bottom
```

Description of Brick

HTML

Shows an html page, that can be used for - course material - instructions - to store answers of students

Extended Description of Brick (Tech only)

addition
Telt 2 getallen bij elkaar op
Ingangen :
- Eerste getal, type = Number
- Tweede getal, type = Number

Uitgangen :
- Som, Number

Methods :
- Generate_Output_Signals

Python Documentation  (november 2008)

Python
basic tutorial from ZetCode
Python by Example
Introduction to Python

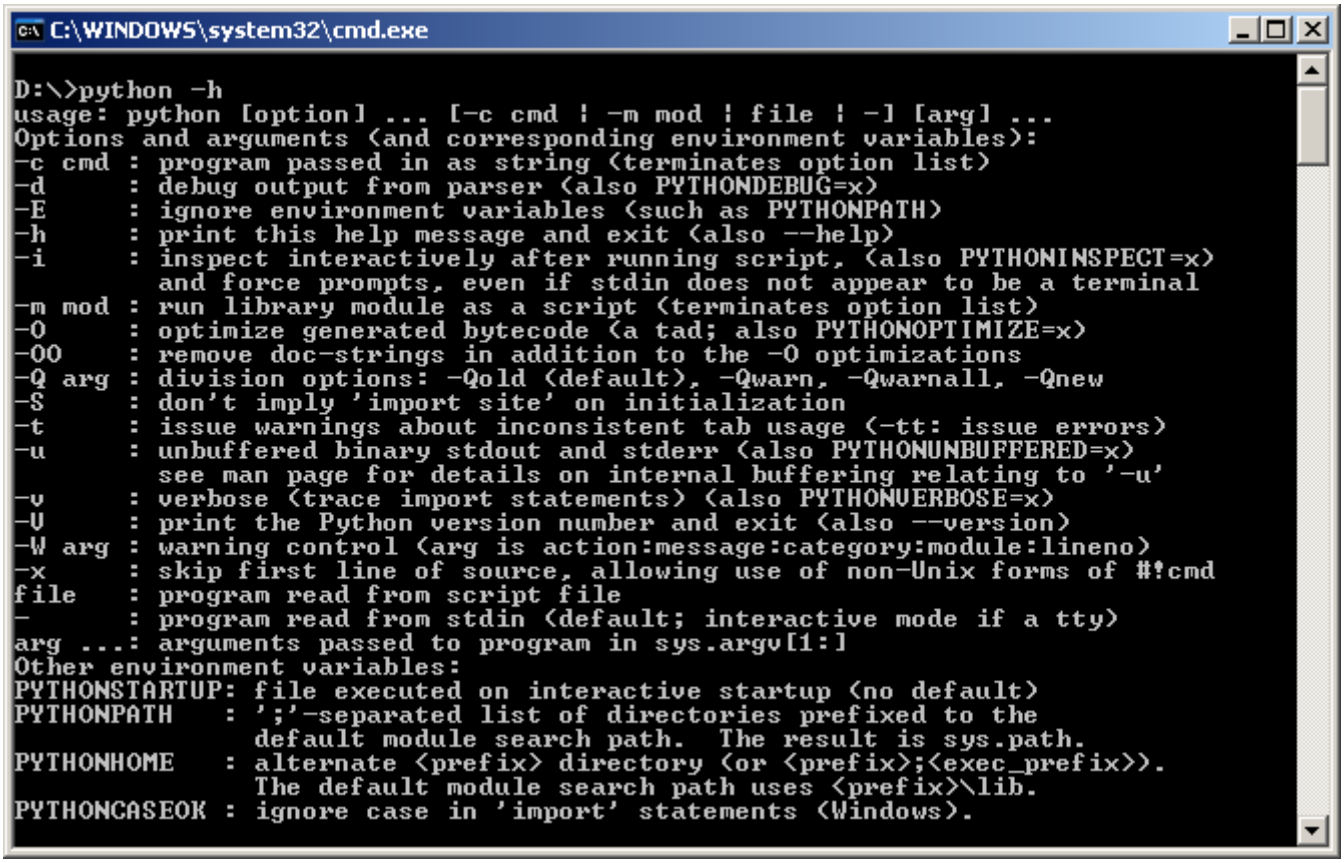
Tim Golden's win32

Scipy / Numpy
MatPlotLib and the gallery

wxPython
basic tutorial from ZetCode

VPython
The famous Ball
Vpython Contributions
GeoScience Models Introductory Geoscience Models - GeoMod
UCT Physics Department

Command line parameters



Coming from MatLab (february 2009)

Application Designer / Domain Expert / Control Designer / Core Developer

Translating MatLab

LiberMate: SourceForge.net LiberMate

One MatLab per Child

Projects An Open-Source MATLAB®-to-Python Compiler

MLabWrap: Revision 2099 trunkmlabwrap

MatLab from other languages: MATLAB

Embedding MatLab

I only embedded MatLab in (realtime) signal analysis is Delphi, which works reasonable well, if you can find the right block size, to optimize the communication speed compared to an accepted delay.

I never used MatLab embedding in Python (why should you :-) , but here are some links.
Revision 2077 trunkmlabwrap
Using the MATLAB Engine to Call MATLAB Software from C and Fortran Programs

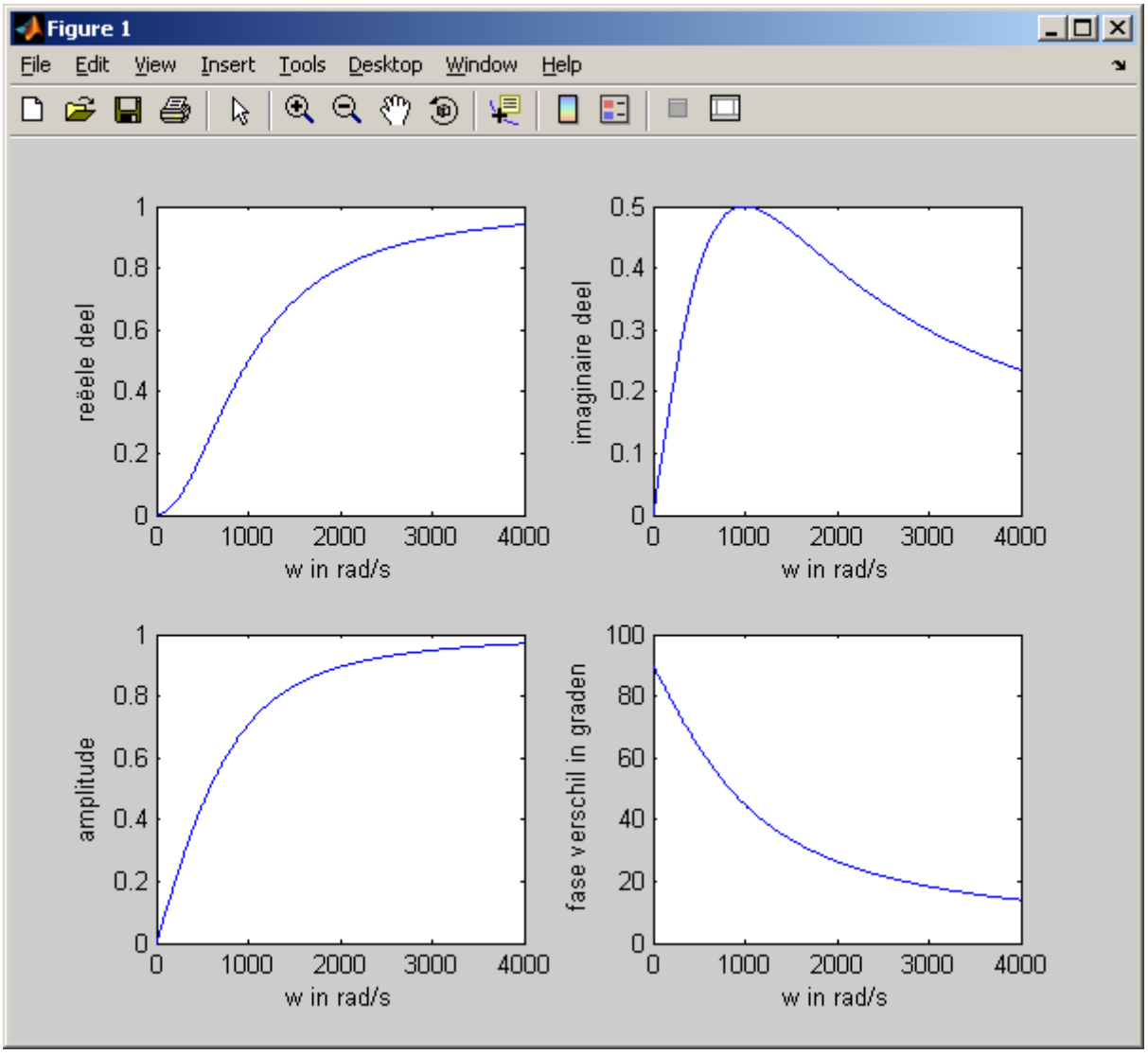
LiberMate (february 2009)

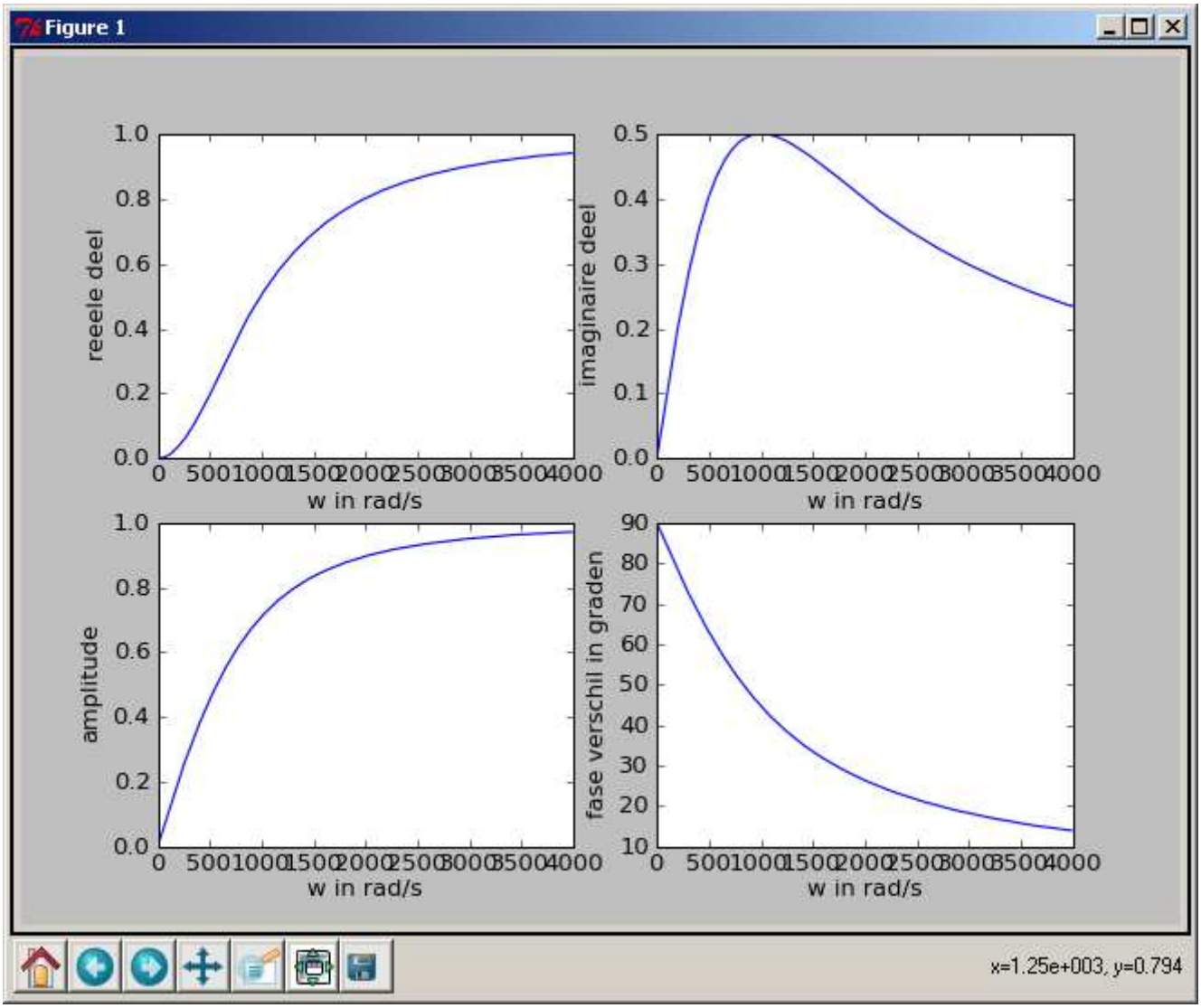
Application Designer / Domain Expert / Control Designer / Core Developer

First test

HPF_graph.m

- show() command is not needed in Matlab.
- Case-Senitive function calls are automatically translated in Matlab
- Complex number i is transformed in a function 1j()
- product of 2 numbers is changed in a (ugly) dot-product.
- can't handle unicode e.g. "reëel"
- matdiv is not found
- graphs are nicer, but axis are uglier
- subplot needs integer values





Installation  (6 april 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

April 2009: The best Python 2.5.4

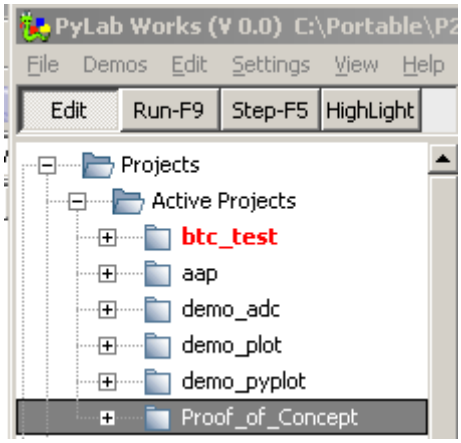
Ubuntu 9.0.4

Has Python 2.6.2 in board.

Ubuntu 8.0.4

Available demos

Has Python 2.5.2 on board.

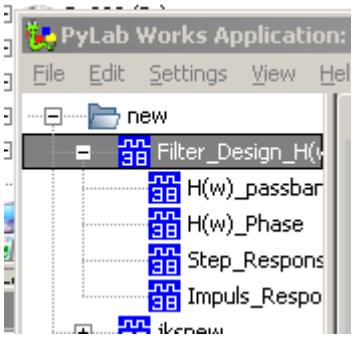


Selecting one of the demos:
- select the desired node in "Projects | Active Projects | ..."
- Right click on it
- select "Set as Active Project"
- press "Run-F9" button

btc_test
works partial, (of course you first must select a valid sound signal) does how the sound signal, but doesn't play the soundfile.
- wave was not included in the build version, fixed
- pygame gives problems with numeric, no idea how to solve this ??? (numpy, numarray works very bad with py2exe)

```
21 """
22 import pygame
23 Traceback (most recent call last):
24   File "PyLab_Works.py", line 33, in ?
25   File "PyLab_Works_Globals.pyc", line 14, in ?
26   File "pygame\__init__.pyc", line 147, in ?
27   File "pygame\surfarray.pyc", line 12, in ?
28   File "pygame\surfarray.pyc", line 10, in __load
29   File "Numeric.pyc", line 93, in ?
30   File "Precision.pyc", line 26, in ?
31   File "Precision.pyc", line 23, in _fill_table
32   File "Precision.pyc", line 18, in _get_precisions
33 TypeError: data type not understood
34 """
```

aap
OK
Press RUN, Select the second node, press shift-F9 and 5 windows should appear.



demo_adc
doesn't work, due to the new scope module, the old scope module is inherited from the new modules, but must be fundamentally be changed !!

demo_plot
OK

demo_pyplot
OK
- background color of image can't be changed, due to a change (bug) in newer version of MatPlotLib

proof_of_Concept
OK, but you have to select a valid image first (could be managed by Inno)
(config file contains absolute paths, don't know yet how (and if) to solve that)

Installation
The installation file contains everything, and you should be able to install wherever you like.

wxPython 2.8.8.1
gives weird problems when closing PyLab_Works, don't use it for the moment

Source Installation 2.5
Items preceding with an asterisk are only needed when certain components are needed.

- install python 2.5.2 msi
- install numpy 1.0.4
- install scipy 0.6.0
- install wxPython + docs-demo 2.8.7.1 unicode
- install MatPlotLib 0.91.2
- install Pygame + docs 1.8.0
- install SendKeys 0.3
- install PyScripter 1.7.2 , updated to 1.9.9.1 !!!
- install Rpyc 2.6
- install ConfigObj 4.5.2 , unpack in sitelibs and run "python setup.py install"
- * install PyODBC 2.0.58 for P2.5, only if ODBC support is needed
- * install LXML 2.1 http://codespeak.net/lxml/index.html , if Babelfish translation is desired
- install winpdb-1.3.8.tar.gz
- install wmi 1.3.2, unpack in sitelibs and run "python setup.py install" (only for core developers in windows)
- pywin32, version 212, just run the executable
- * install psyco 1.6, sourceforge
- install VPython, version 5, if you want VPython
- install PIL version 1.1.6, if you want a good Image library
- PyODE 1.2.0: http://pyode.sourceforge.net/#download
- PyParser 1.5.1: SourceForge.net Python parsing module Files

If needed

- PyRTF 0.45 (creating RTF-files) PyRTF - Rich Text Format (RTF) Document Generation in Python
-
-

possibly, not yet decided

- SourceForge.net Nucular Fielded Full Text Indexing

De-Installation 2.4
This describes how I did the de-installation, realizing that this 2.4 install was a complete mesh !!

- Backup complete python installation

- de-install GTK+ run time environment 2.2.4.1, don't know if this anything todo with Python
- de-install all Python 2.4 ... installation: Boa / EasyDialogs / EpyDoc / IPython / MatPlotLib 0.91.2 / Py2exe 0.6.6 / PyAudio / Pygame 1.8.0 / pySoy 1.0b2/ PythonCard 0.8.2 / SendKeys 0.3 / SPE 0.8.3c / wxPropertyGrid 1.3.2 / wxPython 2.8.7.1 ansi / Vpython 3.2.9 / VisualWx alfa 0.8750 / Femm 4.01 / Cornice 0.3.4 / Python 2.4 Enthought /
- Remove folder P:\Python
- Remove PyScripter
- Clear registry for Python / PyScripter
- Clear Documents and Settings
- Clear Document and Settings / Applications

Source Installation 2.4

The files below are used, to build the program under winXP. As one of the major problems with Python libraries is, the lack of downwards compatibility of the libraries, it's advised not use other versions.

Python + Scipy

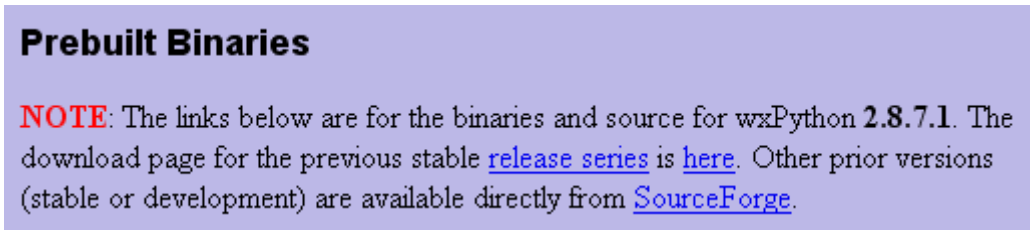
Use the orginal 2.4.3 Enthought Edition, don't use the newer eggs version (they gave me a whole lot of trouble)

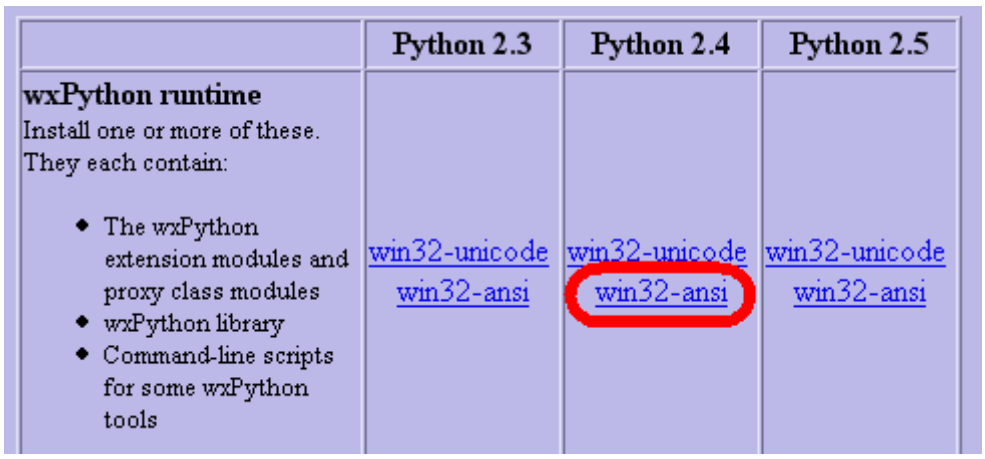
http://code.enthought.com/enthon/

*** Python 2.4.3 - Enthought Edition 1.1.0 (#69, Oct 6 2006, 12:53:45) [MSC v.1310 32 bit (Intel)] on win32. ***

wxPython

http://www.wxpython.org/download.php





CustomTreeCtrl Modified

....

MatPlotLib

http://sourceforge.net/projects/matplotlib

Pygame
Original started with version 1.7.1,
now switched to 1.8

http://www.pygame.org/news.html

VPython = Visual / Soya3D ==> PySoy / OGL
Not decided yet what to choose. Probably in the beginning we don't use a 3D-engine, but just a 2D-engine based on modified OGL-like.

Changes in Python Libraries

- **plot**: around line 1000, some lines added to realize background color setting (including axis font based on grid/background)
- **CustomTreeCtrl** Demo Implementation, !!!!! CHANGES to CustomTreeCtrl: !!!!!!!!!!!! # line 1042 ...
- and much more !!
- **PhysicalQuantities** , revision: 2006-4-28, around line 370, exchange multiply order
-

Ubuntu (16 may 2009)

Application Designer / Domain Expert / Control Designer / **Core Developer**

Should be added to all py-files which can run stand alone.

```
1 #! /usr/bin/env python
```

lxml
After installing lxml, import of lxml succeeds, but import of lxml.html doesn't ??? therefor the Babelfish translation in the languge control doesn't work.

ADC
The AD-converter is a windows executable, written in Delphi, so it will not work under Ubuntu.

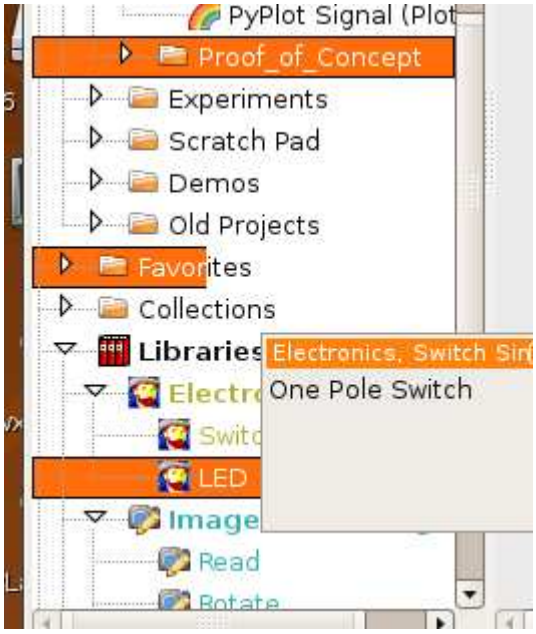
MatPlotLib
For some unknown reason, it's not possible to set the size of an embedded MatPlotLib window smaller than some rather large minimum size.

VPython
Under Ubuntu-8, only VPython-3 is available. Replacing it with VPython is a hell of a job. With VPython-3, most examples work quit well, only the specific VPython-5 demos will not work. Although I couldn't find them, there might be a few instructions which are not valid within VPython-3, in that case it should be easy to extend VPython with the missing methods, implemented as just dummy methods.

Pygame
The Pygame standard available for Ubuntu 8, is a little to old (and uses numeric), but it can easily be replaced by the pygame version of Ubuntu 8.10 (Intrepid).

ODBC
Don't know how to attach ODBC, so for the moment the radiobutton has no real functionality under Linux, and other ODBC selections are blocked.

Tree annoyances
The trees in Ubuntu are sometimes not correctly refreshed, resulting in very ugly images, like these:

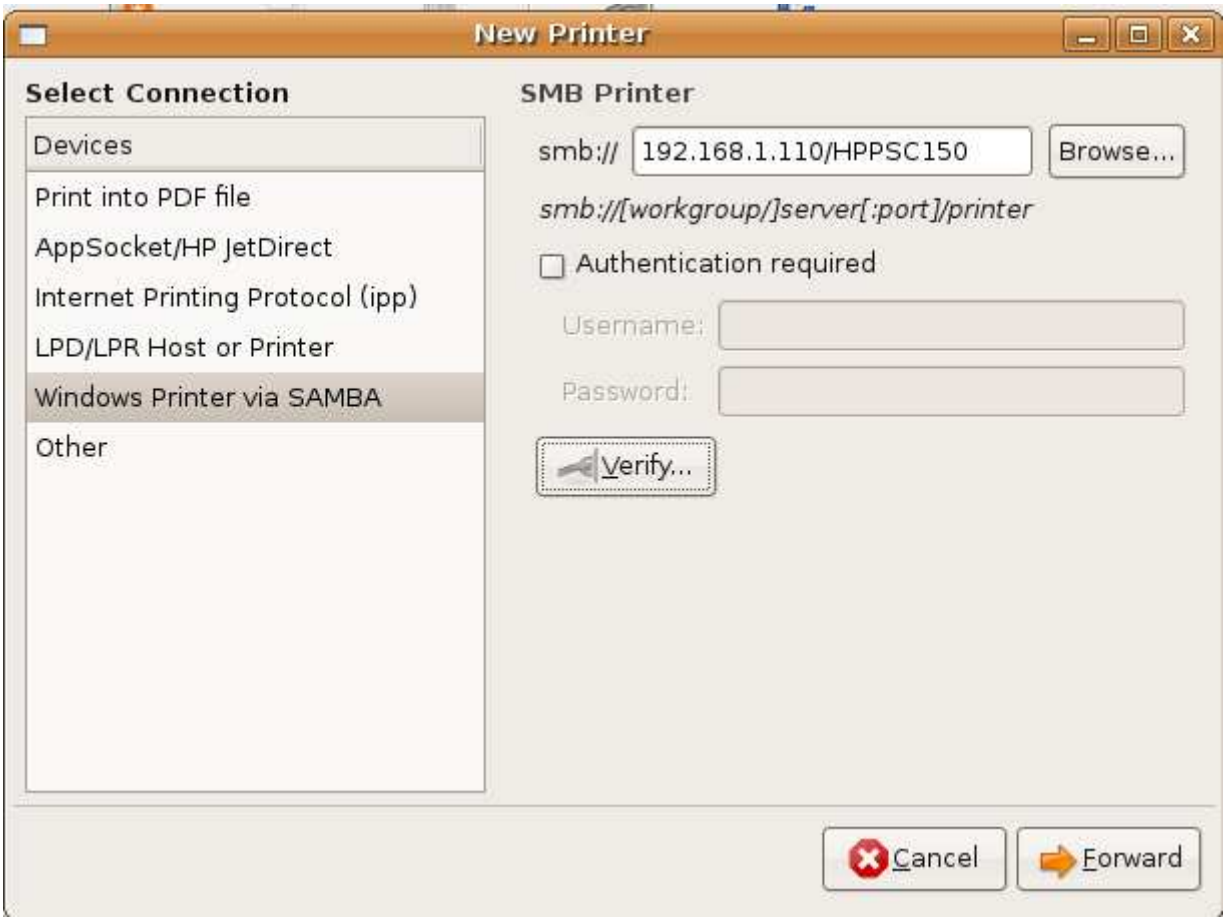


CUPS Printer errors ?

Some weird error, reported many times on the internet with other programs, but no one seems to have a solution or even an explanation for it. Seems to be a known bug in libwxgtk 2.8.

```
** (python:17168): WARNING **: Can't create printer "PDF" because the id "PDF" is already

(python:17168): GnomePrintCupsPlugin-WARNING **: The CUPS printer PDF could not be create


(python:17168): GnomePrintCupsPlugin-WARNING **: The data for the CUPS printer PDF could
```

Installing another printer and setting this new printer as the



PYC

If by accident, pyc files are copied from Windows to Ubuntu, we can get these weird kind of errors:

```
/media/disk/Data_Python_25/PyLab_Works/PyLab_Works.py
************** ALARM *************
New Language = US  Set from : D:\Data_Python_25\PyLab_Works\PyLab_Works_Globals.py
```

TextCtrl

SetLabel doesn't work under Ubuntu (results in a empty textbox), SetValue works both on Ubuntu and Windows.
AFAIK all SetLabel are replaced by SetValue.
The same yields for GetTextLabel ==> GetValue

april 2008

Download / History

contributors:
Lucian Iuga-Popescu
Robbert Mientki
Stef Mientki

ToDo (Bugs and Features)

bugfix: Deleting a Brick that claimed a field in the statusbar, **doesn't release this statusbar field** when the Brick is deleted

- ***bugfix:*** When an **application is finalized**, going back to the edit mode, doesn't realize the fixed build up the GUI
- ***bugfix:*** On some systems the application mode can only be switched between maximized and minimized
- ***bugfix:*** Horizontal Layout is not stored / retrieved ?
- ***bugfix:*** CS_gen bij image show bevat [1] ??
- ***improvement***, replace PythonSTC with Base_STC

V 0.2xx released ....

- ***extension:*** **2D-Scene** added
- ***extension:*** new Brick property: **Outputs_Connected**
- ***extension:*** **BabelFish translation** added to Translation Manager
- ***change:*** Integration of code-editor + code-editor2
- ***bugfix / change:*** **Left-Double-Click in Graphical Code Editor** displayed a bug message, now used for toggling the grouping rubberband

V 0.1 released 14 april 2008

- ***change***, 14 april 2008, **translation error reporting disabled**
- ***bugfix***, 14 april 2008, a Brick containing **no GUI controls** works now
- ***extension***, 14 april 2008, **Diagnostic flag** added to Brick's parent
- ***extension***, 14 april 2008, **Continuous flag** added to Brick's parent
- ***extension***, 14 april 2008, new connection type **TIO_STRING added**
- ***bugfix***, 12 april 2008, Brick = **Code Editor, F9** (send changes to output) didn't work anymore
- ***bugfix***, 12 april 2008, Brick = **Code Editor, Statusbar** started with the wrong text
- ***extension***, 11 april 2008, first version of Multi-Language **Translation-Manager**
- ***improvement***, 10 april 2008, Designer window **Connection lines made rectangular**

V 0.0 released 01-04-2008

- first pre-alpha release

Deployment  (june 2009)

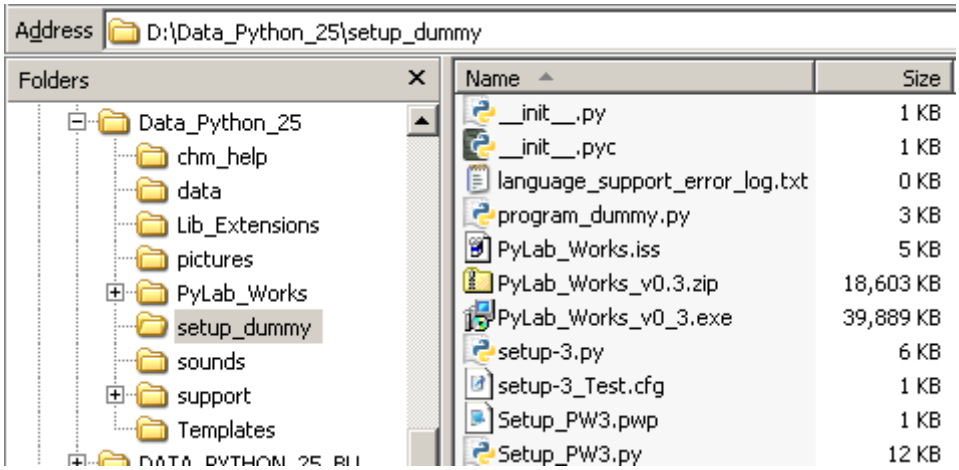
Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

Because PyLab_Works is a very dynamical program, a standard run of py2exe will fail. So we use a trick to get a good windows binary distro: add an extra dummy program that does nothing, but imports all libraries. There's another reason why we don't want py2exe to compile our programs, we want to distribute the sources and not the compiled pyc-files. Therefor we don't let py2exe create exe-files from our programs directly, but we use a small startup python file, that starts the real application dynamically.

Here is the program "_launch_gui_support.py", that will dynamically launch the real program "support/gui_support.py"
This program is generated by Setup_PW3.py.

```
1 import os, sys
2 Base_Path = os.getcwd()
3 My_Globs = {}
4 My_Globs [ '__name__' ] = '__main__'
5 Work_Path = os.path.join ( Base_Path,"support")
6 os.chdir ( Work_Path )
7 sys.path.append ( Work_Path )
8 execfile ( "gui_support.py", My_Globs )
```



setup-3.py

This is the GUI for the distribution, very pre-mature.

Setup_PW3.py

Contains everything to make both the full Inno-installer for windows and a zipped archive with the sources only.

Deploy.py

This program performs the following taks:

- removes all pyc / pyo files, which will give trouble because they might contain absolute paths.
- Guarantees that _Init__ from the root will be copied to all necessary places
- Deletes other temporary files

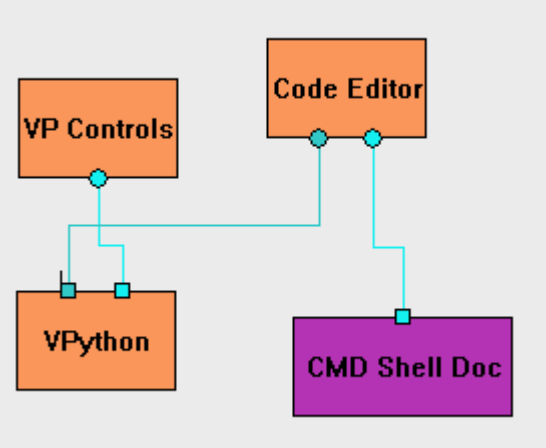
Problems

(may 2009)

Application Designer / Domain Expert / Control Designer / **Core Developer**

Code Editor + VPython

In the case below, I think both Code Editor and VPython are excuting



which is probably the cause of the follwoing error:

```
Traceback (most recent call last):
  File "D:\Data_Python_25\PyLab_Works\bricks\brick_Plotting.py", line 753, in Generate_Ou
    exec ( Code ) #, self.Code_Globals, self.Code_Locals )
  File "<string>", line 85, in <module>
NameError: name 'VPC' is not defined
******** Code Editor, ERROR: *********
```

july 2008

Library Manager

```
1  k=4
2  k
3      ==> 4
4  3+4*k
5      ==> 19
6
7  file: dialog_support
8  def AskFileForOpen ( DefaultLocation = '', DefaultFile = '',
9                       FileTypes = '*.*', Title = '' ) :
10
11 *** Saved and Running : test_IDE.py
12 Traceback (most recent call last):
13   File "D:\Data_Python\P25_PyLab_Works\PyLab_Works_Library_Manager.py", line 627, in
14     execfile ( self.Source_File )
15   File "test_IDE.py", line 5, in <module>
16     cv
17 #NameError: name 'cv' is not defined
18
19 aa=3*k
20 aa
21     ==> 12
22
23 print aa
24 12
```

Introduction

With the Library Manager it's easy to compare / update / rollback your local files with the latest files on the web. The library manager scan your PyLab_works directories and the PyLab_Works Website, compares these two and gives you a nice overview of the differences. The you can decide on an individual file base which files should be updated to the latest version.

Below is image of the Bricks tab, which is a good example of all the other groups.
At the top of the page you can select some general settings, like language (see Brick Math, which is in Dutch), at what details you want to view the information (Techno, ..), ...
Then a number of tables, specifying all local and all available libraries. These files are split in number of logical groups:

- Bricks, the functional building Bricks of PyLab_Works
- Unwanted, Bricks that are available, but you're not interested in (but might be in the future)
- Program, core libraries of PyLab_Works (you need them all)
- Private, libraries found in your directories, but not (yet) part of PyLab_Works
- New, Brick Libraries found on the web and not yet in Bricks or UnWanted. Preferable you should move these files to one of the latter tables.

At the bottom is a general log file (editable), which shows help info, collects all user actions, update information, etc. This information is also stored in a new log file and can be printed.

Through the RM-menu, some extra information is available (e.g. history between current and latest version), which is also included in the log-memo.

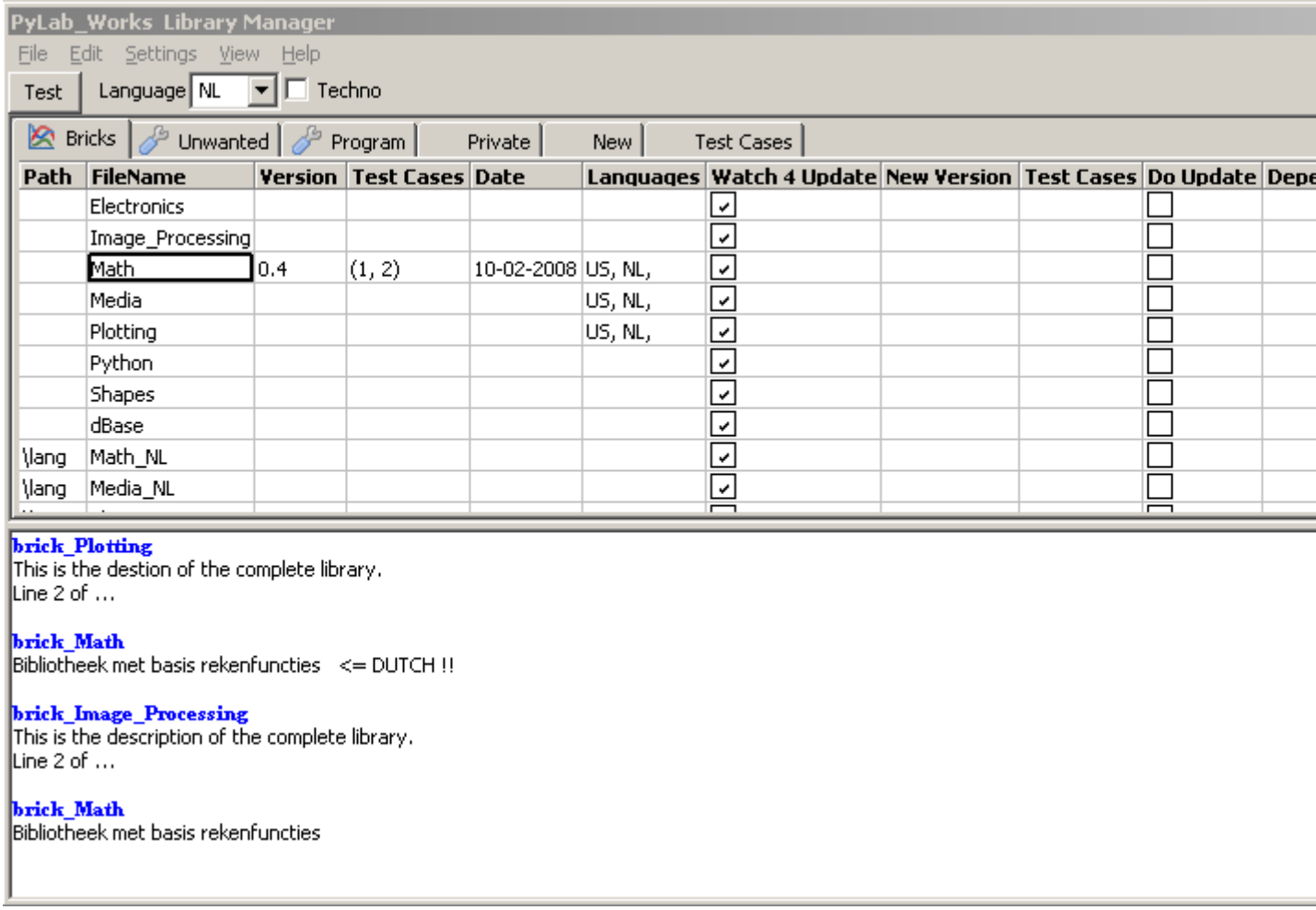


Table Columns

Depending on the selected group of files, the available columns may vary.

Path

The relative (to PyLab_Works) path where the library is located.

FileName

The filename of the library. For Bricks the prefix "brick_" is removed.
Clicking on a cell in this column, adds help information to the memo-log. The information might be a user

description or an extended Technical article including full history.

Version
The version of the current local file.

Test Cases
Under what conditions the current local file is tested.

Date
The release date of the current local file.

Languages
The available languages for this library. If nothing is specified, there are no translation files and only the original language (should be US) is available.

Watch 4 Update
If this checkbox is checked, PyLab_Works will regular check the website for new versions of that library and the program will warn you automatically.

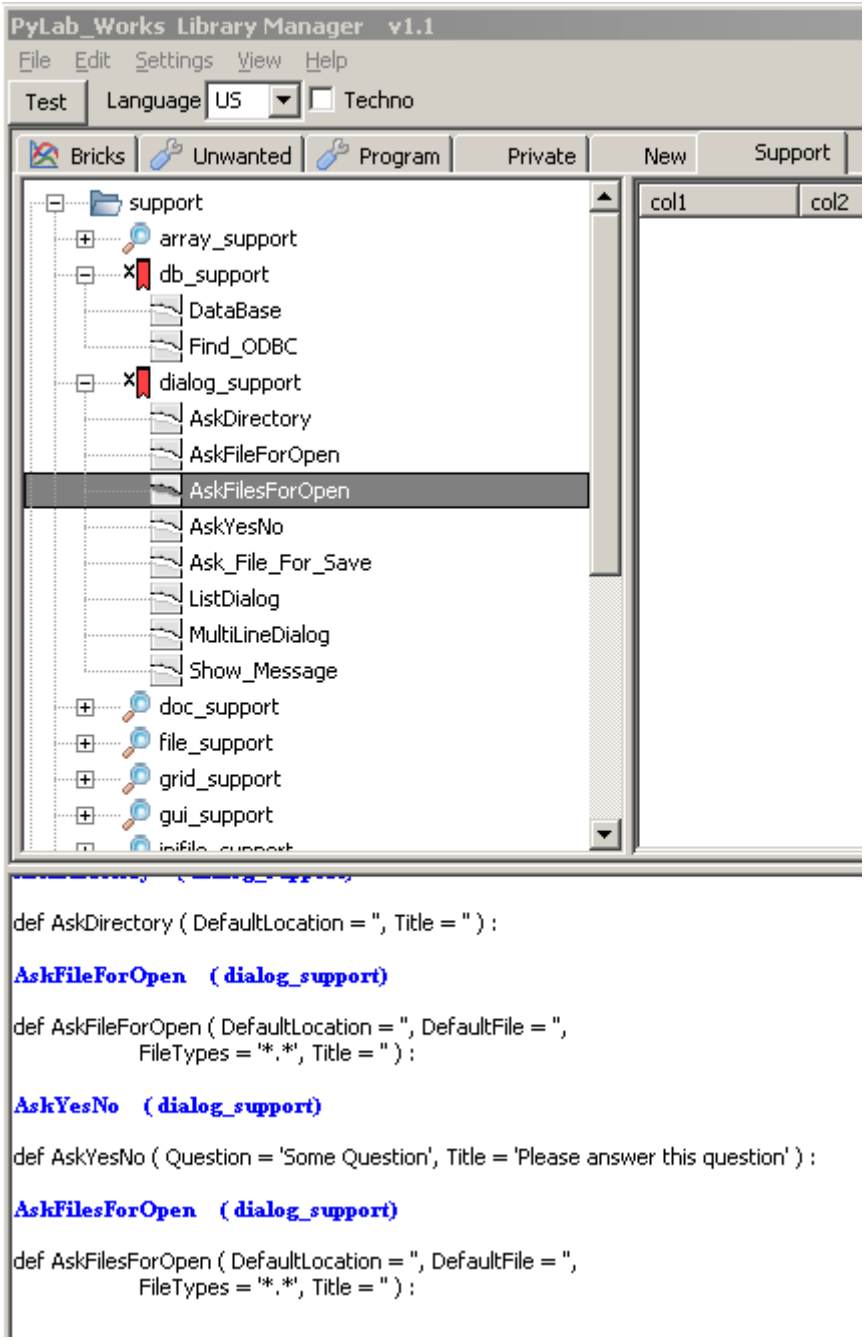
New Version
The version of the latest release.

Test Cases
Under what conditions the newest version is tested.

Do Update
If checked this file will be updated with the latest version.

Dependencies
Not realized yet.
The general idea is that PyLab_Works generates a dependency list and will warn when updating files while you forget to update files which also need to be updated.

Support Tab



RM-menu

ToDo

Bricks tab

When clicking on a cell in the second column, the description of the library is added to the memo-log. (In Techno mode, extended description, technical information and full history is added).
If a newer version is available, the differences between the local file and the file on the web is also added to the memo-log.

Help Menu  (may 2009)

Application Designer / Domain Expert / Control Designer / Core Developer

Because documentation and other information is of great importance for PyLab_Works, it's obvious we make all available information easy accessible. Therefor the help menu is dynamically expanded. Unfortunately this task can not fully be automated, but in the future automatic scan and search should make it more easy.

The help menu is in all programs identical, and consists of the following items:

- "Many Links", point to a web page with many links
- All the chm-files in the chm-help directory are included in the menu list
- links to local files, and webpages can be manually added to the global configuration file

Windows help files (CHM-files) don't work very well on other than Windows OS. Under Ubuntu I got it working, but every time you launch a CHM file, a new instance is opened. A Linux-guru is needed here.

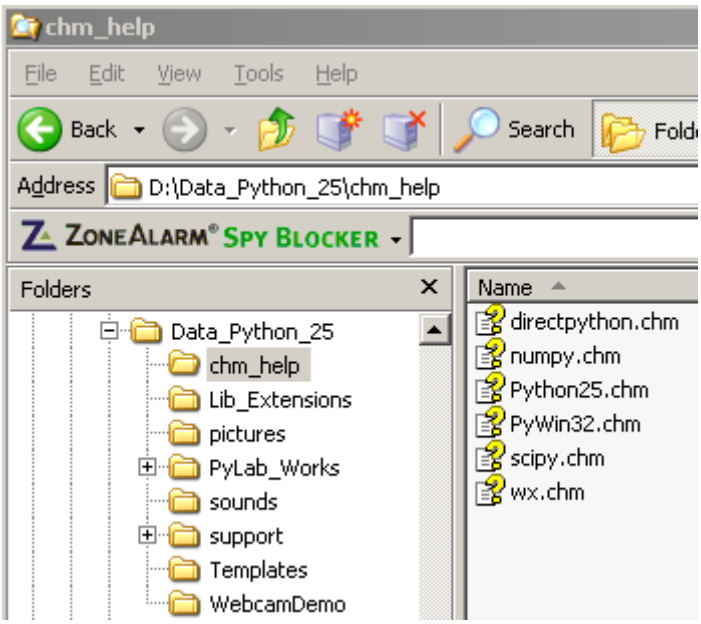
Many Links

This webpage contains a lot of useful links. For the moment I used the first page of my website, I think in the future this might change.

Python by Example Pscyco TraceBack CGI	Numpy Scipy MatPlotLib PyODE Hidden Markov Sage PyAMG Mystic Grace / IDL Modelica	Matlab	wxPython VPython PIL	OpenOffice
PyJamas JavaScript				

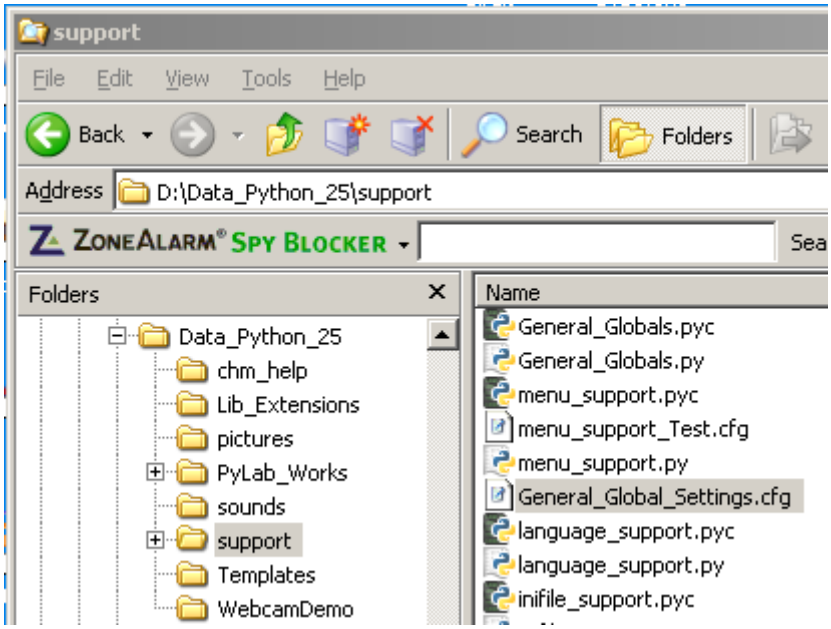
CHM help directory

The chm files in this directory are automatically added to the help menu.



Global Configuration file

In the global configuration file (which yields for all programs), you can define an arbitrary number of local and web-links. For the local files, now only the htm-files and chm-files are recognized, but in the future other files, like doc, pdf should be recognized.
Each user should adapt this file (../support/General_Global_Settings.cfg) !!



The file is simply build as a Windows ini-file (with another extensie, to prevent windows from restoring it) as shown below. At the moment only html (web and local) and CHM-files are supported, but in the bear future this will be extended with pdf, doc, ...).

```
[Help Paths]
&wxPython = '../chm_help/wx.chm'
&Numpy = '../chm_help/numpy.chm'
&Scipy = '../chm_help/scipy.chm'
DirectX = '../chm_help/directpython.chm'
Win32 = '../chm_help/PyWin32.chm'

wxPython-ZetCode = 'http://www.zetcode.com/wxpython/'
wxPython-Web = 'http://wxpython.org/docs/api/'
wxPython-New = 'http://wxpython.org/onlinedocs.php'
```

Future Python Versions  (april 2009)

Python Version 3

The following new features are known:

print changes from statement to function
From now on, print statements are implermented as 2 functions "v3print" and "exprint",
so they can easily be replaced by a simple text substitute.

exec changes from statement to function
All exec statements are already used as a function (through tupple parameter list)

integer division ...
not yet decided if we'll import this from __future__

Bricks  (jue 2009)

Application Designer / **Domain Expert** / Control Designer / Core Developer

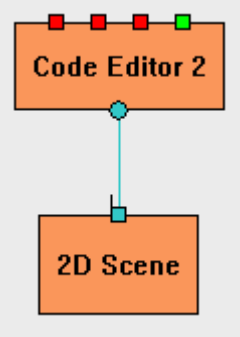
todo

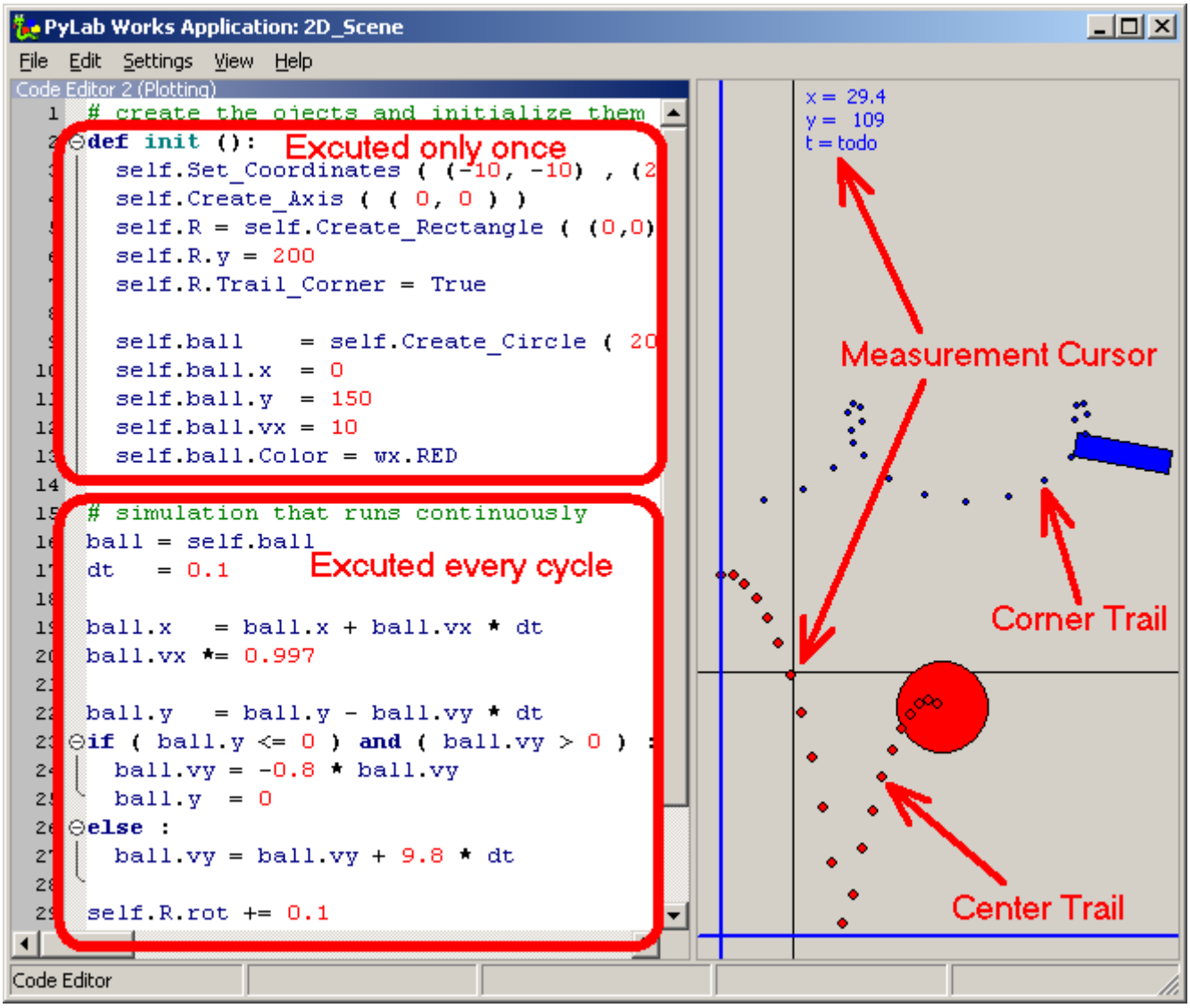
Brick 2D Scene  (april 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

This Brick (consisting of just one control: tScene_2D) presents a 2D space in which you can create 2D animations. In combination with a Code-Editor Brick you can easily simulate physics dynamics and even the special relativity theory of Einstein..





All positions are done in the coordinate system of the user. Most shapes have all common properties, can be dragged around with the mouse and have 2 trails (can be turned on/off) that can be used to measure speed. There are a few special shapes that perform special tasks, e.g. button-shape toggles it's value that can be read by the program code.
This is just a first version and not all issues are decided yet. At this moment the following issues are open for discussion:

- should a shape be sizeable by the mouse (how to inform the program, or does it not need to be informed) ?
- should a shape be rotatable by the mouse (inform the program ?) ?
- if a shape is dragged by the mouse, should the program be informed ?
- should there be a hidden initialization ?
- should there be a slider shape, or should this be done by a normal control ?

The measurement cursor is toggled by the middle mouse button, and can dragged with the middle mouse button. Of course the values in the labels are in the user coordinate system.

Future Extensions

- making an animated recording of the canvas
- time measurement with the cursor

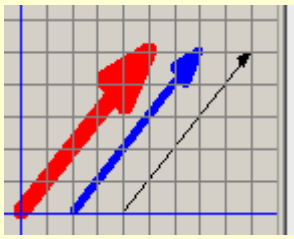
Common Properties

Color	
Line_Width	All shapes use this to determine the width of the lines, e.g. the Text_Shape uses this property to determine the fontsize
x,y	The position of the center of the shape
vx,vy	The speed of the shape
Trail	If True, the Center Trail is drawn
Trail_Corner	If True, the Corner Trail is drawn

Specifying the size of an object

Whenever possible, objects have an universal way of specifying their size. The origin of the object is always defined as the left- bottom (LB) point. The user can choose between specifying width-height (WH), right-top (RT) and a vector from the origin length-angle (RPhi). Here are the 3 ways of generating an identical (but shifted) arrow: (the Line_Width setting is not shown in the code)

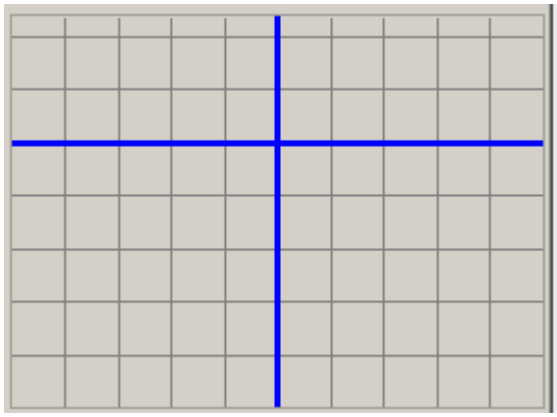
```
200     self.Create_Arrow ( ( 0,  0),       ( 50, 50 ) )
201     self.Create_Arrow ( ( 20, 0), RT = ( 70, 50 ) )
202     self.Create_Arrow ( ( 40, 0), RPhi = ( sqrt(5000), pi/4 ) )
```

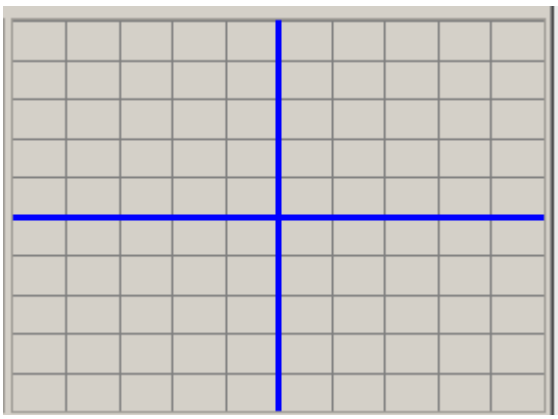


Coordinate System

All coordinates are specified in the user coordinate system, i.e. the user specifies the world value (user value) of the left-bottom / width-height or right-top of the 2D-space. And all object positions should be specified in the user values. If the top value is omitted, the scaling of the Y-axis will be the same as the scaling of the X-axis, so the top value will be depending on the width-height ratio of the visible window. The top picture on the right shows the uniform coordinate system when height / top is not specified. The bottom picture on the right shows the non-uniform coordinate system, where top or height is specified.

```
123     # uniform coordinate system with WH
124     self.Set_Coordinates (
125         ( -100, -100 ), ( 200, None ) )
126
127     # uniform coordinate system with RT
128     self.Set_Coordinates (
129         ( -100, -100 ), RT = ( 100, None ) )
130
131     # non-uniform coordinate system with WH
132     self.Set_Coordinates (
133         ( -100, -100 ), ( 200, 200 ) )
```





Defaults:

- Left_Bottom = (-100, -100)
- Right_Top = (100, None)

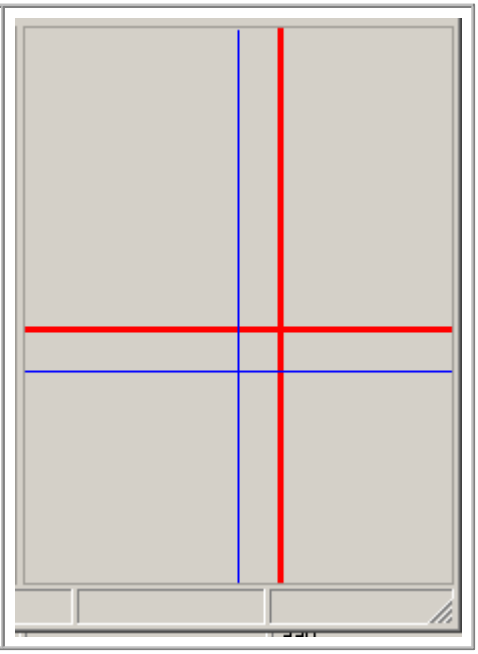
Axis_Shape

You can have multiple axis. The Axis_Shape can either be specified with x,y values or by one xy-tuple.

```
137     shape = self.Create_Axis ( 0,  0 )
138     shape = self.Create_Axis ( ( 20, 20 ) )
139     shape.Color = wx.RED
140     shape.Line_Width = 3
```

Defaults:

- x,y = 0, 0
- Line_Width = 1
- Color = Blue



Grid_Shape

The Grid_Shape is just like any other shape, on creating, the tick interval is specified. After creation you can change the standard properties.

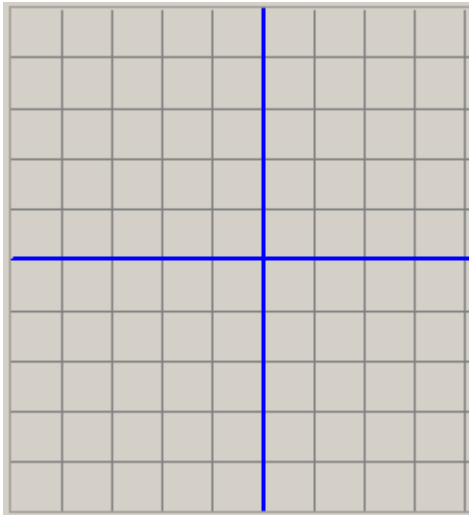
```
139     shape = self.Create_Axis ( 0,  0 )
140     shape.Line_Width = 2
141     shape = self.Create_Grid ( 20 )
```

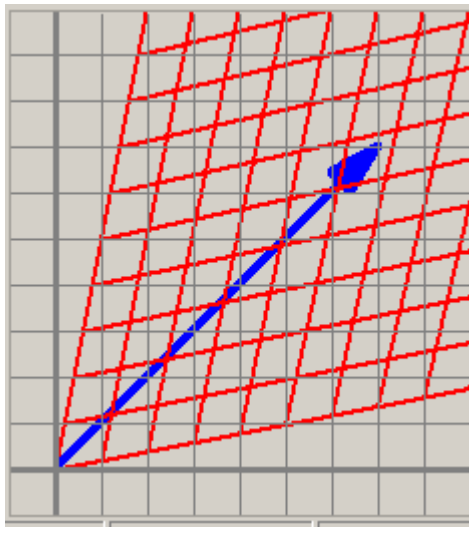
Although the grid can't be rotated, you can specify the property rot, which can already be set on creation as the second parameter. If rotation is not zero, a special grid is created (the red grid in the lower figure on the right), to visualize the time-space in special relativity theory.

```
145     self.Set_Coordinates (
146         ( -10, -10 ), RT = ( 100, None ) )
147     shape = self.Create_Axis ( 0,  0 )
148     shape.Color = wx.GREY_PEN.GetColour()
149     shape.Line_Width = 3
150
151     shape = self.Create_Grid ( 10 )
152     shape = self.Create_Grid ( 10, arctan (0.2) )
153     shape.Color = wx.RED
154     shape.Line_Width = 2
155     shape = self.Create_Arrow ( ( 0,  0), ( 70, 70 ) )
156     shape.Color = wx.BLUE
```

Defaults:

- Color = Grey
- Line_Width = 1
- rot = 0

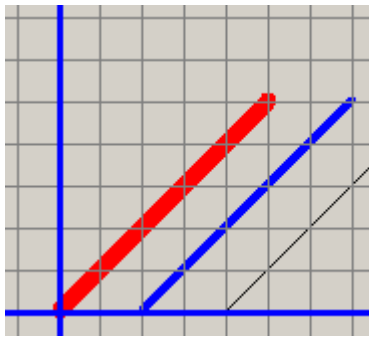




Line_Shape

Defined by an Origin (left bottom) and with either WH (default), RT or RPhi.
Rotation is not done around the center, but around the origin !

```
132     shape = self.Create_Arrow ( ( 0,  0),       ( 50, 50 ) )
133     shape.Line_Width = 8
134     shape = self.Create_Arrow ( ( 20, 0), RT = ( 70, 50 ) )
135     shape.Color = wx.BLUE
136     shape = self.Create_Arrow ( ( 40, 0),
137                                 RPhi = ( sqrt(5000), pi/4 ) )
138     shape.Color = wx.BLACK
139     shape.Line_Width = 1
```



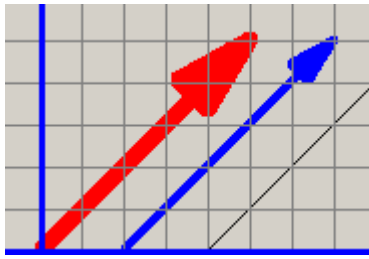
Defaults:

- Color = Red
- Line_Width = 4

Arrow_Shape

Identical to Line_Shape, except that the second point is an arrow-head. The size of the arrow-head is determined on the base of the Line_Width.

```
132     shape = self.Create_Arrow ( ( 0,  0),       ( 50, 50 ) )
133     shape.Line_Width = 8
134     shape = self.Create_Arrow ( ( 20, 0), RT = ( 70, 50 ) )
135     shape.Color = wx.BLUE
136     shape = self.Create_Arrow ( ( 40, 0),
137                                 RPhi = ( sqrt(5000), pi/4 ) )
138     shape.Color = wx.BLACK
139     shape.Line_Width = 1
```



Defaults: see Line_Shape

Text_Shape

Text_Shape is a normal shape that can be moved around and rotated. Fontsize is determined on the base of the standard property Line_Width.

```
146     shape = self.Create_Text ( 'test1', (30,20))
147     shape.Line_Width = 1
148     shape = self.Create_Text ( 'test2', (30,40))
149     shape.Line_Width = 3
150     shape.Color = wx.BLUE
151     shape = self.Create_Text ( 'test2', (30,60))
152     shape.Line_Width = 5
153     shape.rot = pi / 4
```

Defaults:

- Color = Blue
- Line_Width = 1
- rot = 0

Circle_Shape

On creation you can (but don't need to) specify the Radius and the Center position.

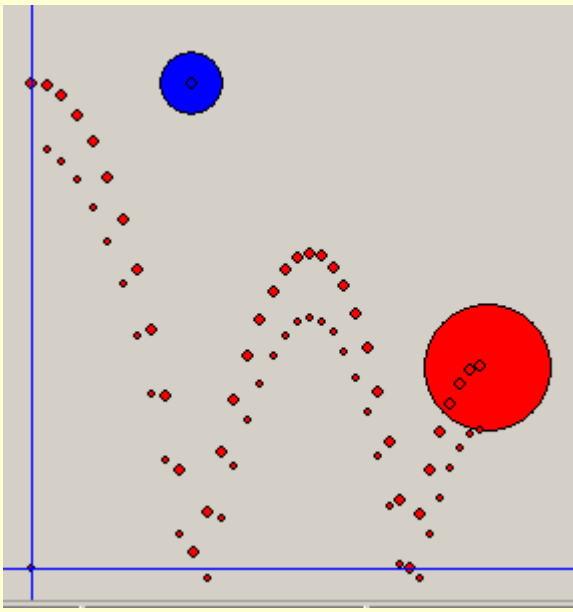
```
177     shape = self.Create_Circle ( 10, [50,150] )
178     shape.Color = wx.BLUE
```

The code below generates the Red Ball in the picture on the right, that has both trails on. The corner trail is at the bottom of the ball (as long as the ball is not rotated.

```
 7
 8    self.ball    = self.Create_Circle ( 20 )
 9    self.ball.x  = 0
10    self.ball.y  = 150
11    self.ball.vx = 10
12    self.ball.Color = wx.RED
13    self.ball.Trail_Corner = True
```

Defaults:

- Color =
- Trail = True
- Trail_Corner = False

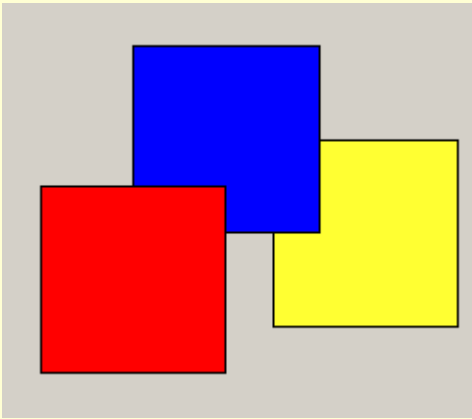


Rectangle_Shape

Normal shape, can be moved, rotated, has both center trail and left-bottom corner trail, can be specified withe the universal parameters. So both the code pieces below (color settings not shown), will generate the same figure, as shown on the right.

```
•186     self.Create_Rectangle ( (0, 0) , (40,40) )
•187     self.Create_Rectangle ( (20,30), (40,40) )
•188     self.Create_Rectangle ( (50,10), (40,40) )
```

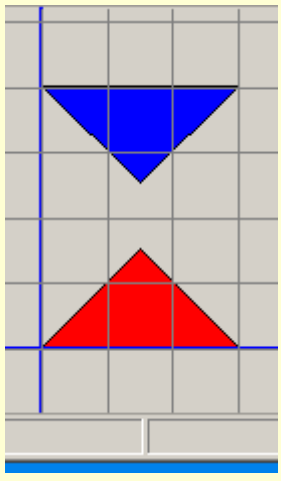
```
•186     self.Create_Rectangle ( (0,0)   , (40,40) )
•187     self.Create_Rectangle ( (20,30), RT = (60,70) )
•188     self.Create_Rectangle ( (50,10),
•189                         RPhi = ( sqrt(3200), pi/4 )  )
```



Free_Shape

Free_Shape is some shapes defined by an unlimited series of points.

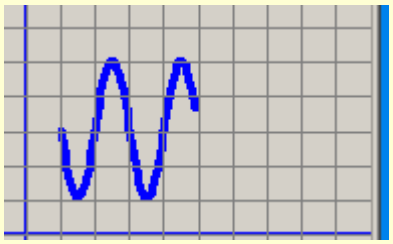
```
•179     shape = self.Create_Free_Shape (
•180         ( (0,0) , (30,0), (15, 30*sin(pi/6) ) ) )
•181     shape.Color = wx.RED
 182
•183     shape = self.Create_Free_Shape (
•184         ( (0,0) , (30,0), (15, 30*sin(pi/6) ) ) )
•185     shape.x   = 30
•186     shape.y   = 40
•187     shape.rot = pi
```



Function_Shape

Put in a function (as a string) and a range (may be floats), and this Shape will draw the curve.

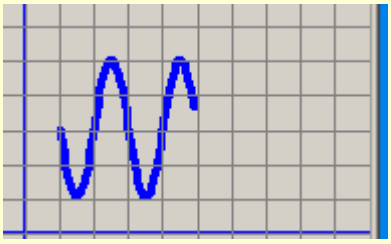
```
197     shape = self.Create_Function (
198       '30 + 20 * sin ( pi * x / 10 )',
199       ( 10,50,1 ) )
200     shape.Line_Width = 4
```



Points_Shape

Same as Function_Shape, but now you've to specify series of xy points.

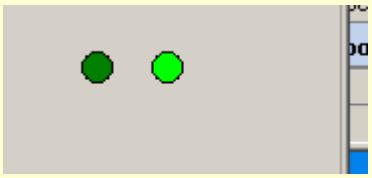
```
202     x = arange ( 10, 50, 1 )
203     y = 30 + 20 * sin ( pi * x / 10 )
204     Points = []
205     for i, xi in enumerate ( x ) :
206       Points.append ( (xi, y[i] ) )
207     shape = self.Create_Points (Points)
208     shape.Line_Width = 4
```



Button_Shape

This is not really a shape, it can not be selected, nor moved. But clicking on it will change it's value and toggles the color, so this can be used to control the program.

```
211     shape = self.Create_Button ( 5, (50,70))
212     shape = self.Create_Button ( 5, (30,70))
```

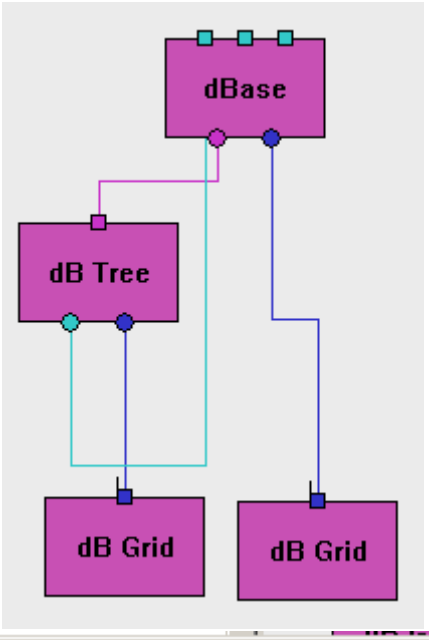


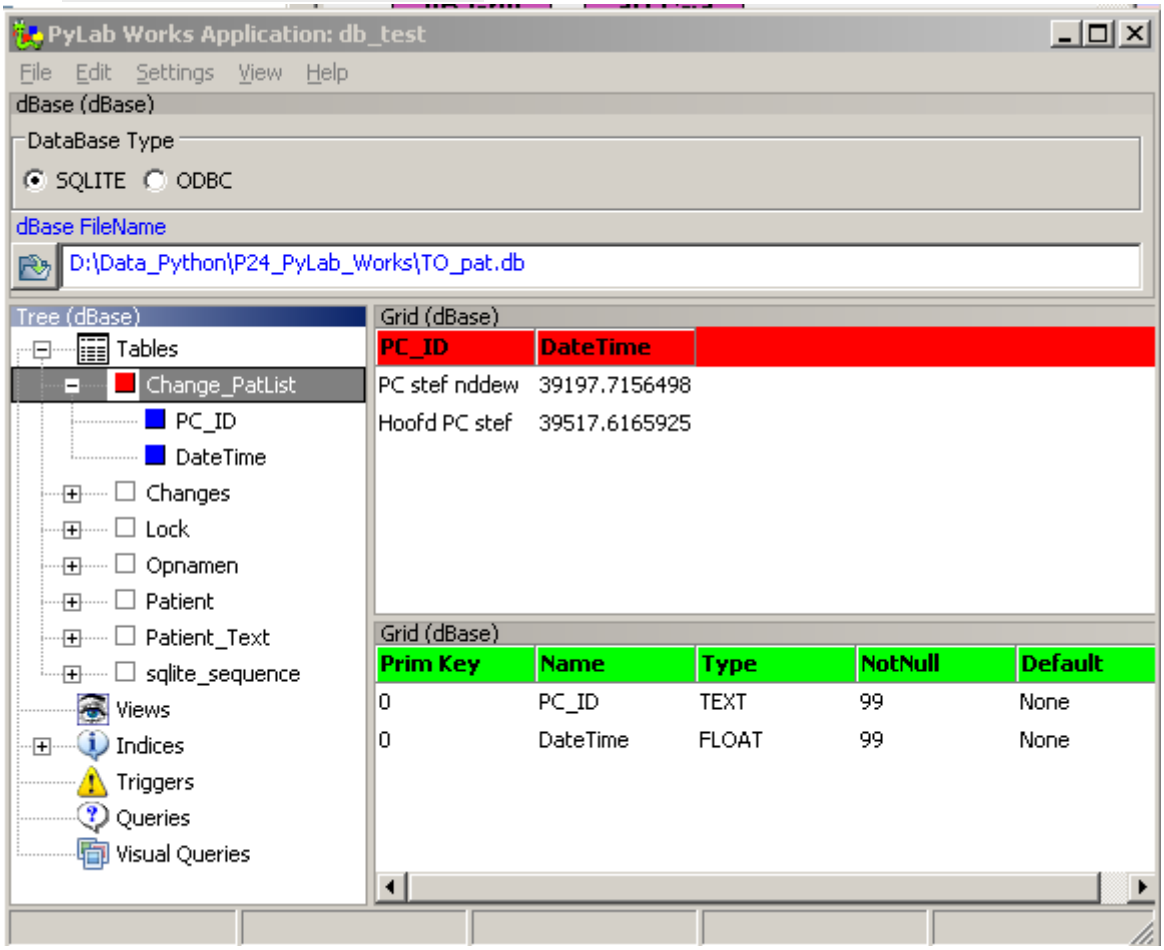
16 may 2008

SQLlite

Application Designer / Domain Expert / Control Designer / Core Developer

Besides the special DataBase bricks, you can use the normal Grid to display tables, use the normal code editor to generate SQL code.

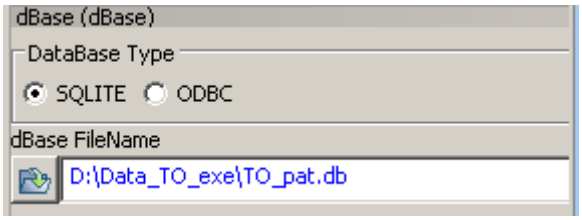


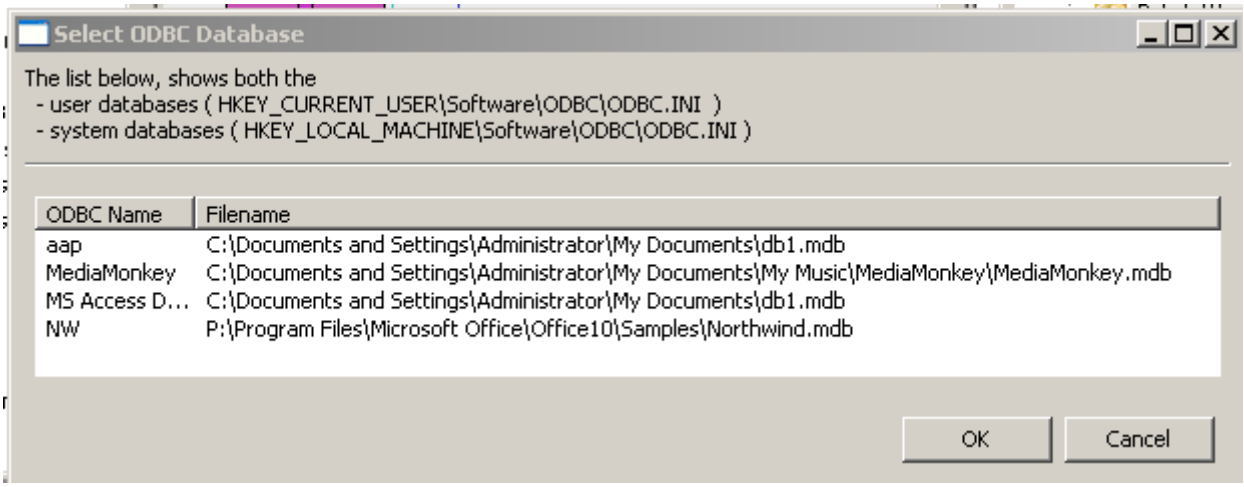


dBase

The database Brick has a buildin open database dialog, which can either connect to a SQLITE3 database or to an ODBC database (MS-Access, Sybase, MySQL, etc.). For ODBC databases it will show both the system databases and the user databases.
The Brick has 2 outputs. The first output gives the metadata of the connected database (tables, views, etc) and can for example be connected to a Tree or a DB_Tree brick. The second output gives the results of a query, in the general form of a grid, so you can connect it to a Grid or DB_Grid.
The Brick has 3 inputs, which are identical and except an SQL string that will be performed on the connected database. If the first input changes, only the SQL statement of the first input will be executed. If the first input didn't change, but the second input changed, the SQL statement of the second imput will

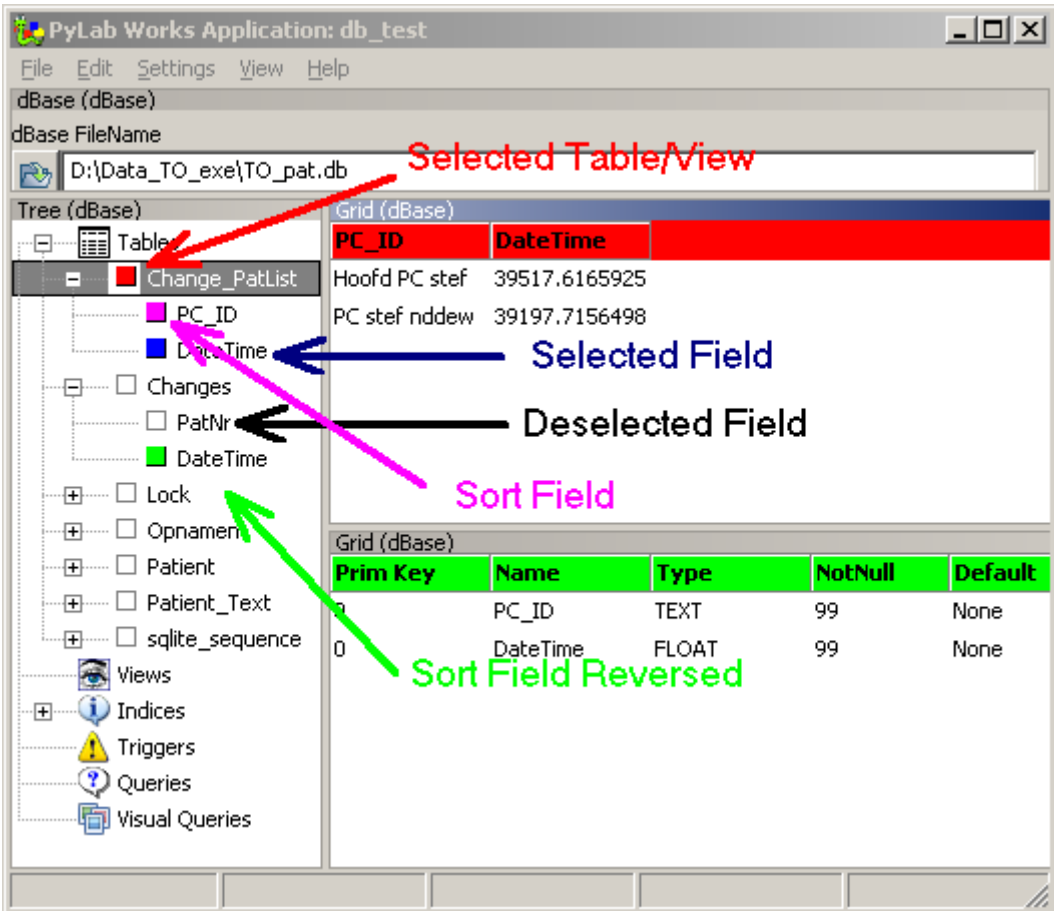
executed and so on. For example you can connect the first input to the SQL output of a dB_Grid, the second input to a general text editor and the third to a visual SQL builder.





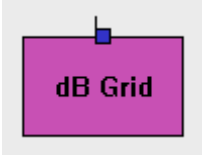
dB_Tree

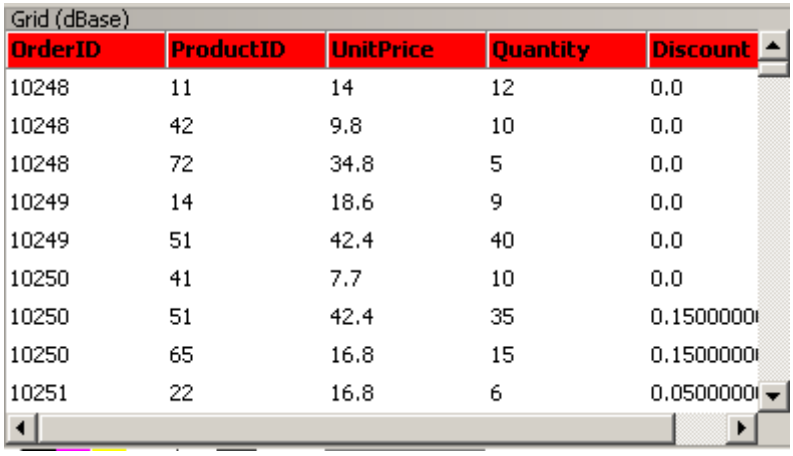
The dB_Tree is a tree with some extra features, to handle the meta-data of a database.



db_Grid

This is a normal gridd !!

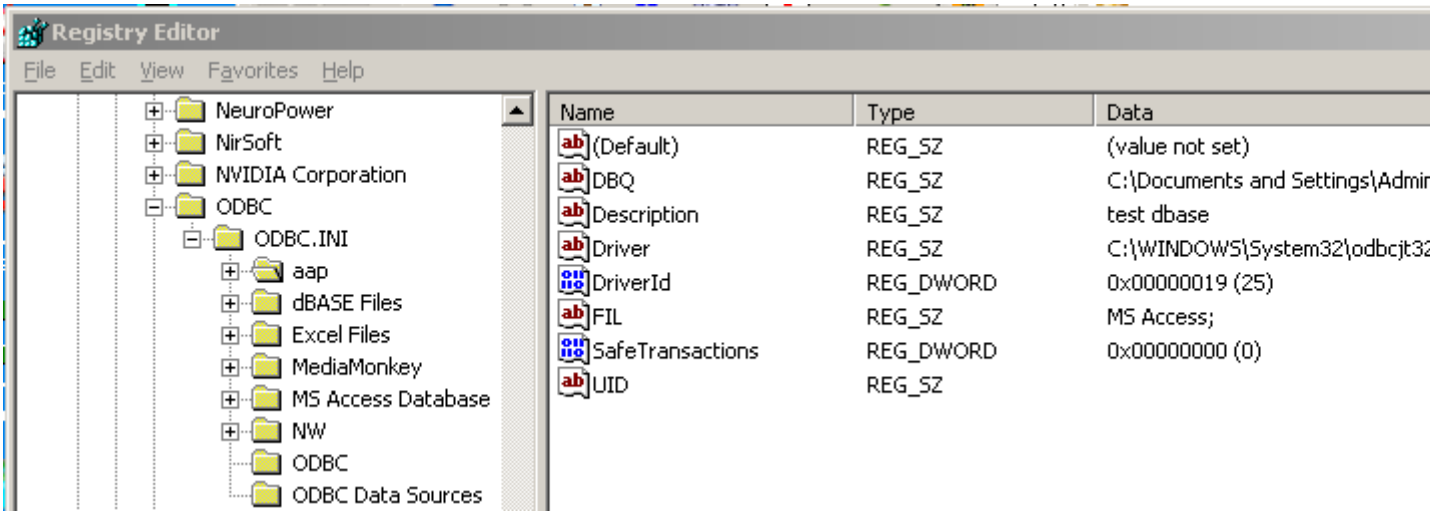




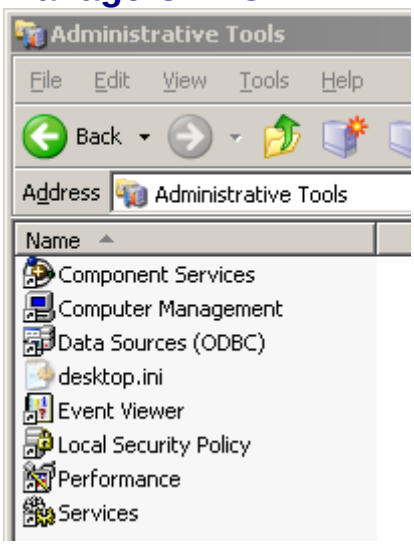
Background Information

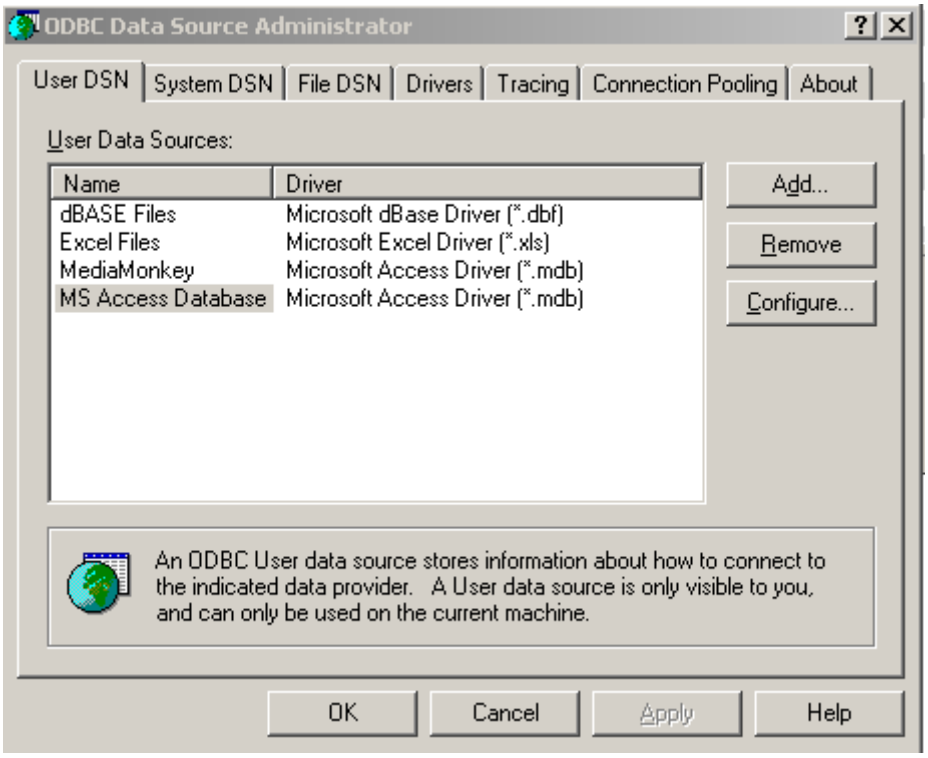
SQLite ODBC Driver

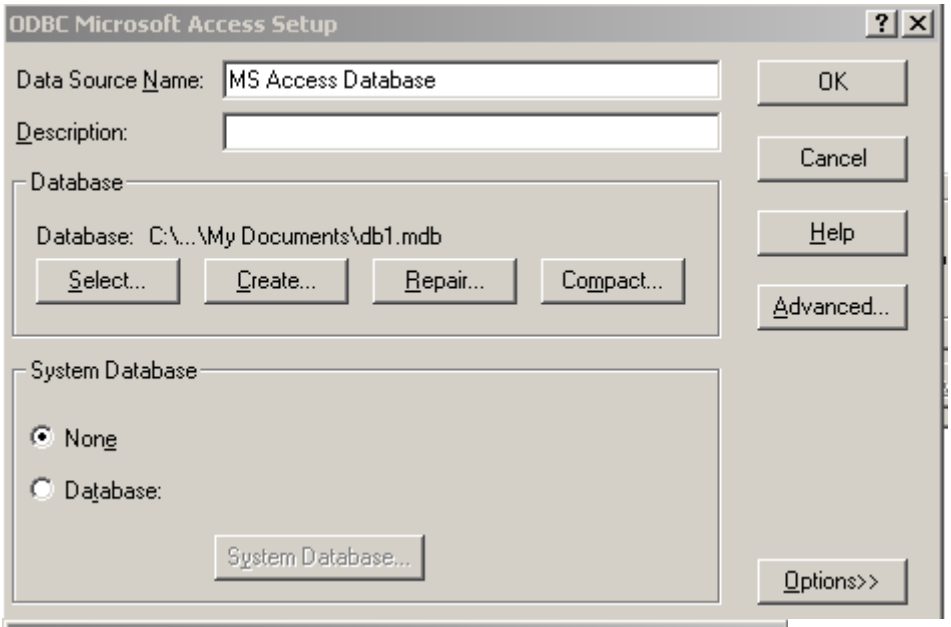
User DSNs are stored in HKEY_CURRENT_USER\Software\ODBC\ODBC.INI.
see :
CodeProject An ODBC (DSNDriver) Manager DLL written in C# (Version - I). Free source code and programming help

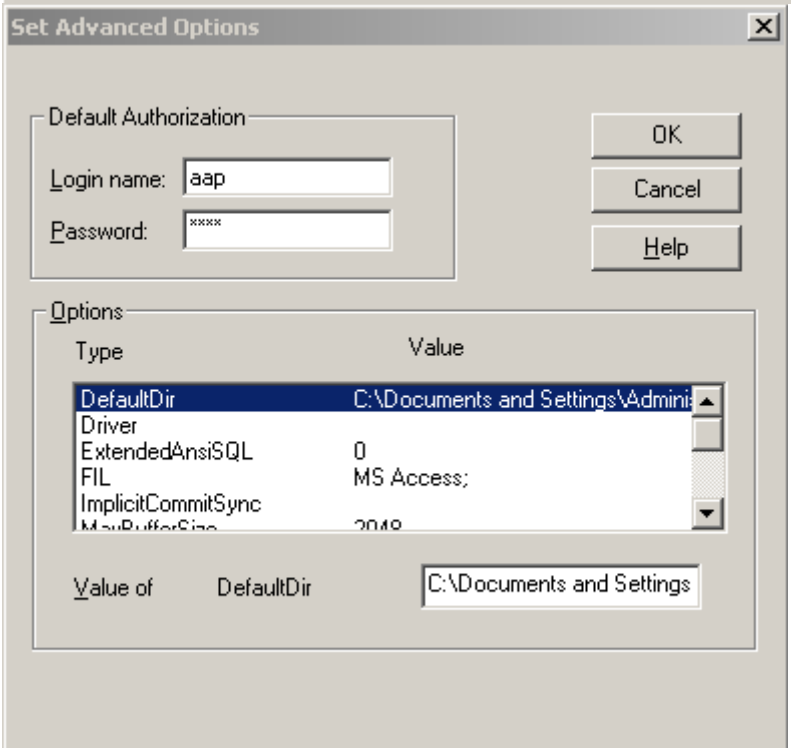

Registry Editor
File Edit View Favorites Help
NeuroPower
NirSoft
NVIDIA Corporation
ODBC
ODBC.INI
aap
dBASE Files
Excel Files
MediaMonkey
MS Access Database
NW
ODBC
ODBC Data Sources
Name Type Data
(Default) REG_SZ (value not set)
DBQ REG_SZ C:\Documents and Settings\Admir
Description REG_SZ test dbase
Driver REG_SZ C:\WINDOWS\System32\odbcjt32
DriverId REG_DWORD 0x00000019 (25)
FIL REG_SZ MS Access;
SafeTransactions REG_DWORD 0x00000000 (0)
UID REG_SZ

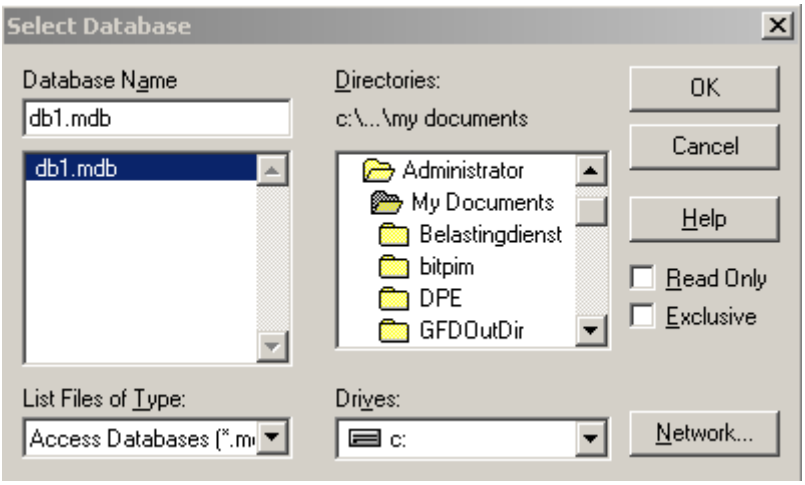
Manage ODBC


Administrative Tools
File Edit View Tools Help
Back
Address Administrative Tools
Name
Component Services
Computer Management
Data Sources (ODBC)
desktop.ini
Event Viewer
Local Security Policy
Performance
Services


ODBC Data Source Administrator
User DSN System DSN File DSN Drivers Tracing Connection Pooling About
User Data Sources:
Name Driver
dBASE Files Microsoft dBase Driver (*.dbf)
Excel Files Microsoft Excel Driver (*.xls)
MediaMonkey Microsoft Access Driver (*.mdb)
MS Access Database Microsoft Access Driver (*.mdb)
Add...
Remove
Configure...
An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.
OK
Cancel
Apply
Help


ODBC Microsoft Access Setup
Data Source Name:
MS Access Database
Description:
Database
Database: C:\...\My Documents\db1.mdb
Select...
Create...
Repair...
Compact...
System Database
None
Database:
System Database...
OK
Cancel
Help
Advanced...
Options>>


Set Advanced Options
Default Authorization
Login name:
aap
Password:
****
OK
Cancel
Help
Options
Type
Value
DefaultDir
C:\Documents and Settings\Admini
Driver
ExtendedAnsiSQL
0
FIL
MS Access;
ImplicitCommitSync
Value of
DefaultDir
C:\Documents and Settings


Select Database
Database Name
db1.mdb
db1.mdb
Directories:
c:\...\my documents
Administrator
My Documents
Belastingdienst
bitpim
DPE
GFDOutDir
OK
Cancel
Help
Read Only
Exclusive
List Files of Type:
Access Databases (*.m
Drives:
c:
Network...

pyodbc
pyodbc is a Python module that allows you to access ODBC databases. It implements the Python Database API Specification v2.0.
Some notable features include:
* The library is free for commercial and personal use.
* It conforms to the DB API standard.
* No 3rd party libraries are required. Only native Python datatypes are used, such as decimal and datetime.
* It requires Python 2.4 or higher, since the decimal type was added in 2.4. (We are open to suggestions regarding versions for earlier Python builds.)
* Additional features have been added to simplify database programming with Python.

Mark J. Nenadov - HowTo Use ODBC with the Python DBI..

ODBC Module

eGenix.com Products Python mxODBC - ODBC Database Interface for Python

ASPN Python Cookbook Odbc connection on win32

Python ODBC Stuff- Peter's Blog

SourceForge.net PyODB
PyODB is an ODBC Python module to provide an quick an easy way to work with databases. It provides a small set of simplified bindings to the unixODBC API and has been developed using SWIG.
Almost no activity, latest 2005

Untitled RealPyODBC
Welcome to RealPyODBC
Version 0.1 beta
This class help you to connect your python script with ODBC engine.
I need at least ctypes 0.9.2 for work.
This class is not db-api 2.0 compatible. If you want to help me to do it
please modify it and send me an e-mail with your work!
All the comunity will thanks you.
Please send bugs and reports to michele.petrazzo@unipex.it
TO-DO
Make compatibility with db-api 2.0, so add:
apilevel, theadsafety, paramstyle, cursor, exceptions, ...
This software if released with MIT Licence

Python ODBC - Summary [Savannah]
Python ODBC - Summary
This project is not part of the GNU Project.
Le objectivo de iste projecto es le realisation de un modulo ODBC pro Python, con licentia GPL, que es usabile in le major numero possibile de systemas operative.
Iste projecto es basate super le modulo ODBC que es parte del pacchetto win32all (que infortunatemente functiona solmente in Windows), ma jam actualmente illo functiona in Linux.
Pro utilisar iste modulo in Linux on necessita un installation functionante de unixODBC o de iODBC.
Registration Date: Sunday 11/23/2003 at 15:18 UTC
License: GNU General Public License v2 or later
Development Status: 2 - Pre-Alph
No activity since 2005

iODBC.org
iODBC has been ported to numerous platforms, including:

Linux (x86, Itanium, Alpha, Mips, and StrongArm), Solaris (Sparc & x86), AIX, HP-UX (PA-RISC & Itanium), Digital UNIX, Dynix, Generic UNIX 5.4, FreeBSD, MacOS 9, MacOS X, DG-UX, and OpenVMS.

unixODBC
The unixODBC Project goals are to develop and promote unixODBC to be the definitive standard for ODBC on non MS Windows platforms. This is to include GUI support for both KDE and GNOME.

Html / pdf / ... Viewer  (december 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

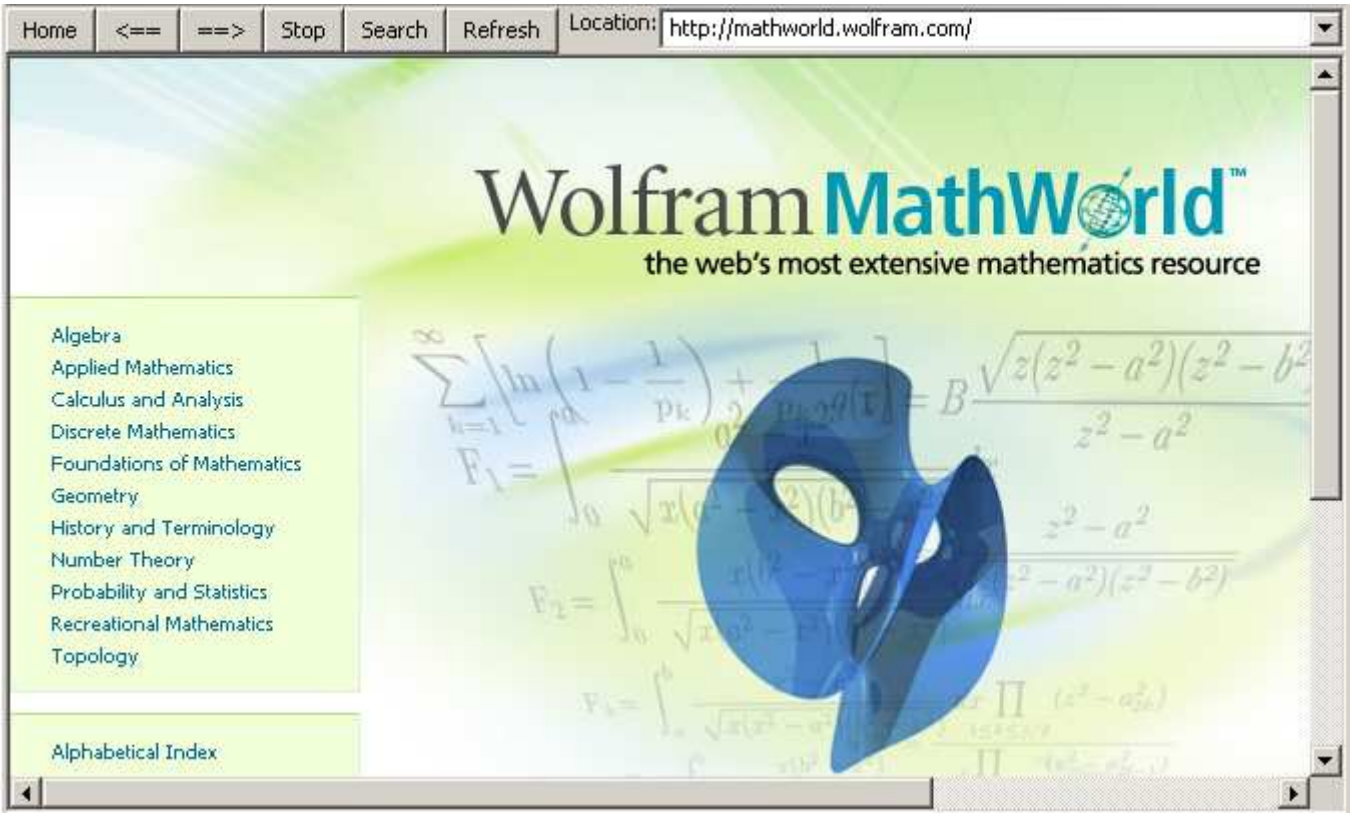
Introduction

Making a general viewer, that can view several standard documents, like web-pages / local html-files / pdf-files / chm-files, is very easy under Windows, but almost impossible under other Operating Systems. Besides that we want to view full blown html-pages (that can't be viewed with wx.html.HtmlWindow) and we want to view html pages with widgets (which can only be viewed with wx.html.HtmlWindow).
The problem might be largely solved, as soon as webkit is integrated in wxPython (at this moment I read that there are already alpha versions available).

The basic solution chosen at this moment is, we place both a wx.html.HtmlWindow and a activeX-IE component together on one panel. Make them both full blown, and use one of the two, depending on the contents, while making the other invisible. For non-Windows OS, we try to display as much as possible on wx.html.HtmlWindow, and everything that doesn't fit, will be sent to the default browser. For non-Windows OS, we could improve the behavior by adding more (external) controls for other document types, like pdf-files and chm-files.

Top Level: URL_Viewer

At the top level, we have a viewer with navigation buttons, that can display a whole range of files: html (+ frames, +CSS), html-widgets, websites, pdf, MS-office-documents, pdf, ... at least this yields for the Windows Operating System. When used in another OS, it will use the default browser for files that can't be displayed with wx.html.HtmlWindow. So in that case the capabilities of the default browser will determine if a document can be viewed or not. The combobox keeps track of the history and is automatically saved and restored.



Low Level

At the low level, we've for all OS's two different viewers available, which on Windows both points to iewin.IEHtmlWindow, but for other OS's points either to the internal viewer or to the external default browser.

```
100 if Win_PlatForm :
101   import  wx.lib.iewin  as iewin
102   IEHtmlWindow     = iewin.IEHtmlWindow
103   IEHtmlWindow_Ext = iewin.IEHtmlWindow
104 else :
105   print '       Contact the developer of this program'
106   IEHtmlWindow     = _My_IEHtmlWindow
107   IEHtmlWindow_Ext = _My_IEHtmlWindow_Ext
```

By always using IEHtmlWindow or IEHtmlWindow_Ext, we can make the program OS independent.

Brick Command-Shell  (december 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

From the user point of view, it consists of two tab pages, an interactive shell, which can perform general IO and catch error messages, where you can perform calculations, keep notes and interact with the active environment. The second tab contains a general viewer, organized as a browser, where you can view all kinds of documents and webpages.

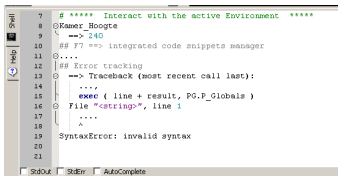
ToDo

Although there seems still a lot to be done, the command-shell is already a very powerful brick.

- special color settings for the editor, so that triple quoted strings (very handy for notes) are clearly visible
- disabling autocompletion doesn't work yet
- ugly empty line, when executing code, with empty lines at the end
- replacing IE-webbrowser by webkit
- including CHMlib as a viewer component (preferable both for Linux and Windows)
- including some kind of pdf viewer, or does webkit has a plugin ?
- click on linenumber in error message, jump into correct location in the editor (if any available)
- better management of visited sites and forward / backward buttons (new instead of append ?)
- let the user remove items from the history list
- better shell icon
- browser status in statusbar
- in error message replace "<string>" by "<editor>" or even the filename if already known
- Stepping through previous commands should be done better (popup window ?)
- during writing this, a bug in correct displaying wxp-widgets is encountered
- grouping tCMD_Shell_Doc = tCMD_Shell + tDoc_Viewer

Interactive Command Shell

In fact the command shell is build around a standard Scintilla editor, so thereby it inherits all Scintilla's features, including automatic save and restore.



StdOut, StdErr

When the checkboxes are checked, program output and error messages are caught by this window. When the brick is closed, the message handlers are restored to the previous ones.

Autocompletion

If the checkbox is checked, all text you type in the command shell, will get (brute force) autocompletion and parameter suggestion features. When typing large amounts of comment this can be annoying, so you can disable the autocompletion with the checkbox.

Execute Code

When pressing enter after the last line, the command-shell grabs all code, starting at the end and going up until a line with zero indent is found. This code is executed in the environment's namespace. So depending on the program, you can often really interact with the running code. With **Ctrl-Up** en **Ctrl-Down**, you can step through previous given commands and select one.

Commands

If the last line starts with an ">" (without the quotes) at the very first location and has no indent, the rest of the line is send as a command to the main application. It's up to the main application what to do with the command, e.g. the application can send it to a debugger to control it's behavior.

Code Snippet

F7 invokes the standard code snippet manager, which is fully integrated.

Help

help can invoked in the standard Python way, but also with a starting question mark "?". So the following 2 code lines are identical

help (wxPython)
?wxPython

General Viewer

The viewer with typical navigation buttons, can display a whole range of files: html (+ frames, +CSS), html-widgets, websites, pdf, MS-office-documents, pdf, ... at least this yields for the Windows Operating System. When used in another OS, it will use the default browser for files that can't be displayed with wx.html.HtmlWindow. So in that case the capabilities of the default browser will determine if a document can be viewed or not. The combobox keeps track of the history and is automatically saved and restored. Another important feature of this browser, is the capability to use and create shortcuts (with or without search keyword).

General

You can just type an URL in the combobox. The prefix "http://" may be left out. Instead of an URL you can also type a local filename or use the browse button to locate a local file.

?

Typing just 1 simple question mark in the combobox, generates an actual information page, explaining all features, displaying all system and user settings and listing of visited pages.

Shortcuts

Here is an example of a shortcut list. The Wikipedia shortcuts are builtin and can not be changed. The shortcuts may look a bit cryptic, but it's important to realize that the user determines the shortcuts. Shortcuts are application wide, so creating them in own application, will make them available in other PyLab_Works applications.

```
These shortcuts are read from the program itself, so they always represent the current state.
("%%%%" will be replaced by the query)
?                   generate and show this page
wp.pendulum         search "pendulum" in wikipedia
wpen.pendulum       search "pendulum" in the English wikipedia, "en" may be any valid language

g       http://www.google.com/search?q=%%%%
gc      http://www.google.com/codesearch?q=%%%%
gp      http://www.google.com/search?q=python+%%%%
lv      http://zone.ni.com/reference/en-XX/help/371361E-01/
ml      http://www.mathworks.com/access/helpdesk/help/helpdesk.html
ml_     http://www.mathworks.com/cgi-bin/texis/webinator/search/?db=MSS&prox=page&rorder=750&
```

For explaining the capabilities, we'll use the example of above. All commands are activated by pressing the Enter-key.

Using Shortcuts

To start a new Google search window, without a keyword search, we just type **g**
To search for wxPython with Google, we type **g.wxPython**
In the list above we see that getting pages from MathWorks (MatLab) we have 2 entries, one for opening the webpage without a keyword search **ml** and one shortcut for searching with a keyword **ml_** .
So simply going to MathWorks, is done by **ml**
And searching the MathWorks site, is done by **ml_.pendulum**

Adding Shortcuts

Adding (or replacing) shortcuts is very simple, just use a two points and type the full URL, and if needed with a placeholder for the search term. So if we want to create a new shortcut, **p**, that performs a search on the python site, we should type:

p..http://google.com/search?domains=www.python.org&sitesearch=www.python.org&sourceid=google-search&q=%%%%
And now we search the python site, with **p.dictionary**

Deleting Shortcuts

To remove the above created python search shortcut, simply type **p..**

Handling File-Types

URL / file type	Windows	Linux
local html	HtmlWindow	HtmlWindow
local html with wxWidgets	HtmlWindow	HtmlWindow
local html with CSS	HtmlWindow	HtmlWindow
http://	embedded IE	external default browser should become WebKit
txt, dat	embedded IE	?
doc, xls, ppt	embedded IE	N.A.
swf	embedded IE	?
pdf	embedded IE	?
chm	external	? should become chmlib

N.A. = Not Available
embedded IE, should be replaced by webkit
No solution found yet to embed CHM under windows.

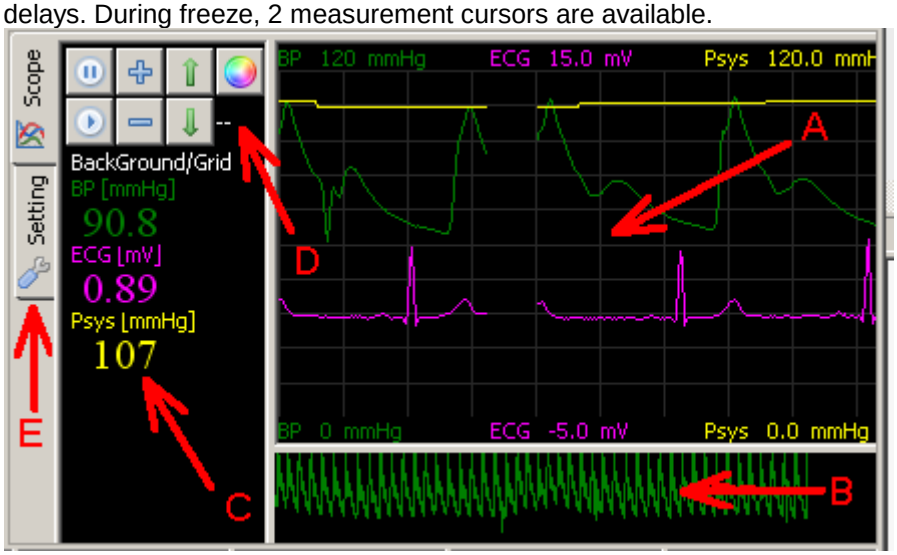
march 2008

ScopePlot 

Introduction

This is an **interactive** display, special designed for **realtime** display and control. Some or all of the recorded signals can be shown in the main signal window. Settings of the signals can be done through the labels on top and bottom, with the **speed-buttons** on the left and through an extensive grid on the second page. On the bottom you see an **history window** where one of the signals can be displayed in min/max mode for the **total recording** (auto compression). This window can also be used to **look back in time**, while recording continuous. Although not strictly necessary, this control will often be the only control in a brick.
The picture shows a realtime ECG and bloodpressure signal with **online analysis** of the bloodpressure (yellow line is the calculated Systolic Pressure). Each signal can be **delayed to compensate** for realtime

filter delays. During freeze, 2 measurement cursors are available.



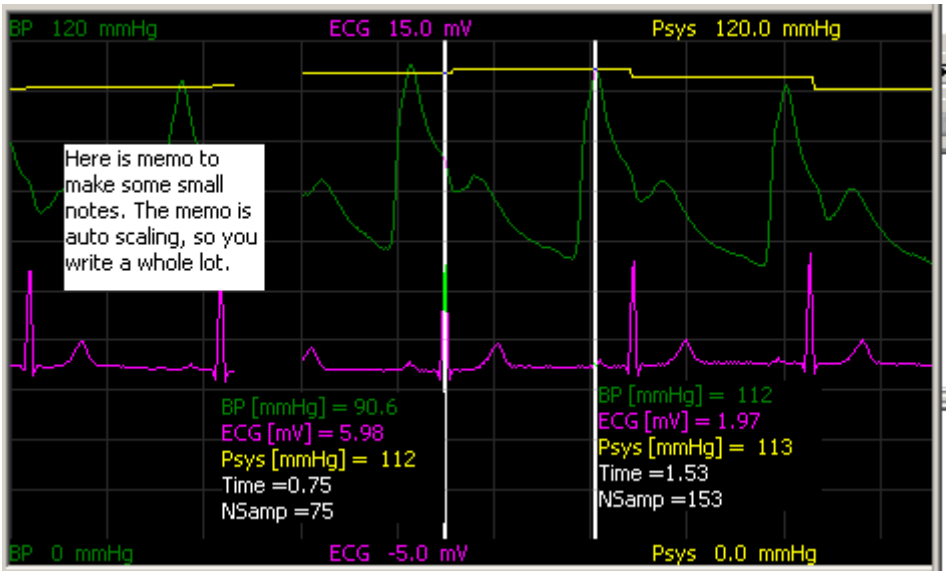
The main parts of the scope display:

- **A: main graph display**, displaying either the actual signal or a part of the already recorded history
- **B: history display**, displaying the whole history in some kind of min/max mode with auto time compression
- **C: numerical display**, display the current value of some of the signals
- **D: speed buttons**, for a quick change of the signals setings in the graph display
- **E: extended settings**, let's you layout the complete display in more detail

In general a brick will only contains this control, in which case it might look like this:
The brick has a number of inputs, each can handle multipe vectors and one control output (callback), that can be used to start / stop an AD-converter.

Main Graph Display

The main graph display is used to show either the actual signals or some part of the already recorded signals. Scaling and shigting can be done in numerous ways: clicking bottom / top labels, speed-buttons and extended settings. In the extended settings you can also change the delay of an individual signal, AC/DC setting, colors etc. In freeze mode you can popup 2 measurement cursors for detailed measurement. Through the RM-menu a memo can be popped up, to make some notes in case you want to save the total image. The memo is placed at the position of the cursor and can be toggled (and thus moved) through the RM-menu.



History Graph

The history graph can display 1 signal over the whole recording session in strip-chart mode (all samples are shown). The amplitude is automatically scaled according to some mean/SD algorithm, with auto compression on the time base. Depending on the type of signal you choose, it might be a valuable way to get an overall view of the experiment, in generel you will be able to see special events during the session. Clicking into this window will show the history (if recorded) in the main graph menu.
In playback mode the history graph can used to cut some parts from the recording.

Numerical Display

This part shows the actual values of the selected signals (refresh rate half second)

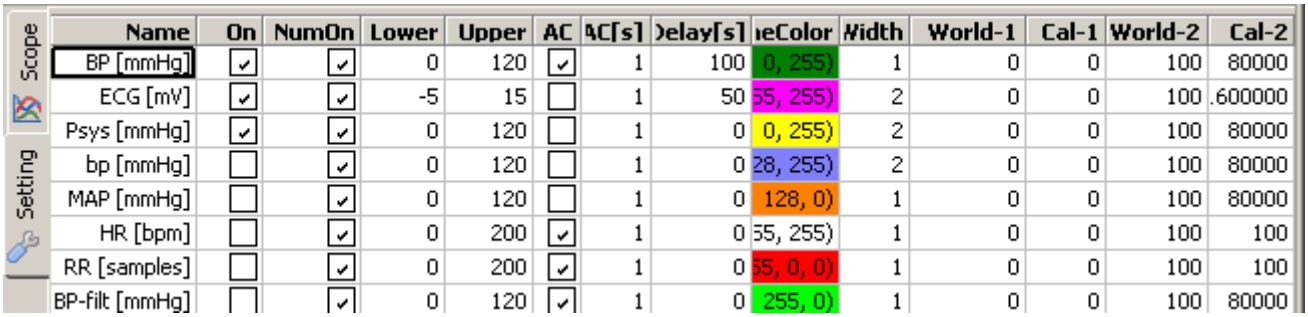
Speed Buttons

By either clicking on a numerical signal label or by clicking on the top or bottom label in the main graph window, the signal is selected to be handled by the speed buttons.

	**stop AD** converter (or recording) **start AD** converter
	**increase the gain** of the selected signal with a factor 2 **decrease the gain** of the selected signal with a factor 2
	shift the selected signal **half a screen up** shift the selected signal **half a screen down**
	**change the color** of the selected signal **the selected signal**

Extended Settings

In the extended settings panel, all the details of the graphical and numerical display can be changed.



Name	On	NumOn	Lower	Upper	AC	AC[s]	Delay[s]	LineColor	Width	World-1	Cal-1	World-2	Cal-2
BP [mmHg]	☑	☑	0	120	☑	1	100	0, 255)	1	0	0	100	80000
ECG [mV]	☑	☑	-5	15	☐	1	50	55, 255)	2	0	0	100	.600000
Psys [mmHg]	☑	☑	0	120	☐	1	0	0, 255)	2	0	0	100	80000
bp [mmHg]	☐	☑	0	120	☐	1	0	28, 255)	2	0	0	100	80000
MAP [mmHg]	☐	☑	0	120	☐	1	0	128, 0)	1	0	0	100	80000
HR [bpm]	☐	☑	0	200	☑	1	0	55, 255)	1	0	0	100	100
RR [samples]	☐	☑	0	200	☑	1	0	55, 0, 0)	1	0	0	100	100
BP-filt [mmHg]	☐	☑	0	120	☑	1	0	255, 0)	1	0	0	100	80000

name	The name of the signal and the units in which it is measured. The units must be specified in square brackets. if no signal name is available, the signal will not be displayed.
On	Select, to show the signal in the main graph window
NumOn	Select, to show the signal in the numerical window
Lower	The lower border value (in world coordinates) in the main graph window
Upper	The upper border value (in world coordinates) in the main graph window.
AC	Select, if the signal should be drawn in AC-mode in the main graph window
AC[s]	The time constant of the AC-setting
Delay[s]	The delay of the signal in the main graph window. If you're using real-time filters (which always will have a time delay), you can also delay the original signal synchronous.
LineColor	The color of the signal in both main graph window and numerical window.
LineWidth	The linewidth in the main graph window
World-1	For 2-point calibration, the world value which will correspond to the AD-value "Cal-1"
Cal-1	AD-value for world value World-1
World-2	For 2-point calibration, the world value which will correspond to the AD-value "Cal-2"
Cal-2	AD-value for world value World-2

Code Editor  (march 2008)

developer

SciTE uses the default key bindings defined in Scintilla, so keyboard commands in SciTE mostly follow common Windows and GTK+ conventions. All move keys (arrows, page up/down, home and end) allows to extend or reduce the stream selection when holding the Shift key, and the rectangular selection when holding the Shift and Alt keys. Some keys may not be available with some national keyboards or because they are taken by the system such as by a window manager on GTK+. The user.shortcuts setting may be used to assign a key to a function. Note that Home key behaviour is changed by the vc.home.key option. Keyboard equivalents of menu commands are listed in the menus. Some less common commands with no menu equivalent are:

Magnify text size.	Ctrl+Keypad+
Reduce text size.	Ctrl+Keypad-
Restore text size to normal.	Ctrl+Keypad/
Cycle through recent files.	Ctrl+Tab
Indent block.	Tab
Dedent block.	Shift+Tab
Delete to start of word.	Ctrl+BackSpace

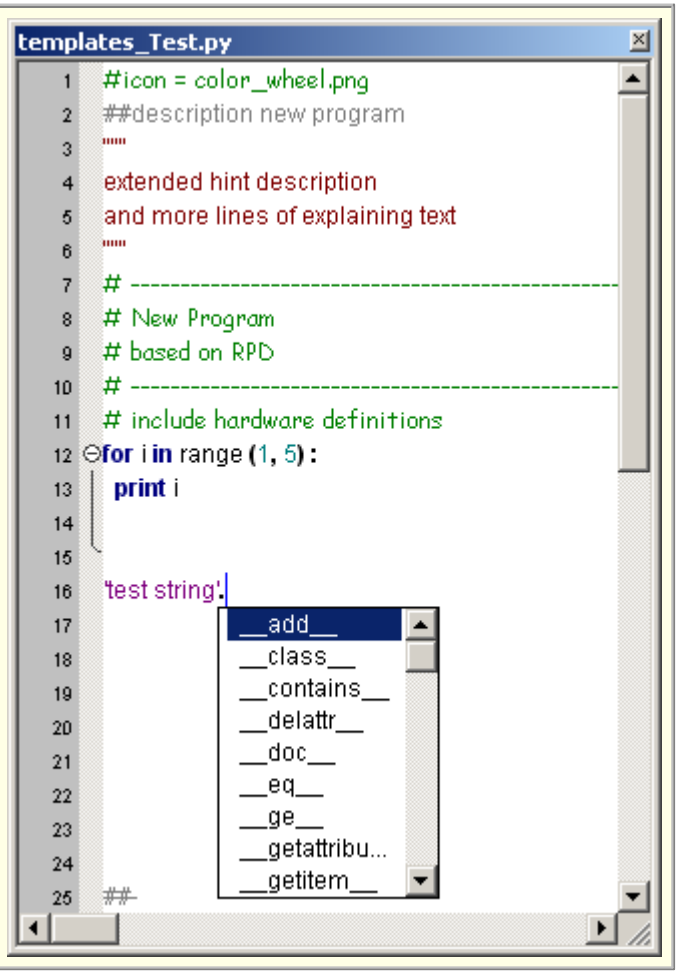
Delete to end of word.	Ctrl+Delete
Delete to start of line.	Ctrl+Shift+BackSpace
Delete to end of line.	Ctrl+Shift+Delete
Go to start of document.	Ctrl+Home
Extend selection to start of document.	Ctrl+Shift+Home
Go to start of display line.	Alt+Home
Extend selection to start of display line.	Alt+Shift+Home
Go to end of document.	Ctrl+End
Extend selection to end of document.	Ctrl+Shift+End
Go to end of display line.	Alt+End
Extend selection to end of display line.	Alt+Shift+End
Expand or contract a fold point.	Ctrl+Keypad*
Create or delete a bookmark.	Ctrl+F2
Go to next bookmark.	F2
Select to next bookmark.	Alt+F2
Find selection.	Ctrl+F3
Find selection backwards.	Ctrl+Shift+F3
Scroll up.	Ctrl+Up
Scroll down.	Ctrl+Down
Line cut.	Ctrl+L
Line copy.	Ctrl+Shift+T
Line delete.	Ctrl+Shift+L
Line transpose with previous.	Ctrl+T
Selection duplicate.	Ctrl+D
Find matching preprocessor conditional, skipping nested ones.	Ctrl+K
Select to matching preprocessor conditional.	Ctrl+Shift+K
Find matching preprocessor conditional backwards, skipping nested ones.	Ctrl+J
Select to matching preprocessor conditional backwards.	Ctrl+Shift+J
Previous paragraph. Shift extends selection.	Ctrl+[
Next paragraph. Shift extends selection.	Ctrl+]
Previous word. Shift extends selection.	Ctrl+Left
Next word. Shift extends selection.	Ctrl+Right
Previous word part. Shift extends selection	Ctrl+/
Next word part. Shift extends selection.	Ctrl+\

Code Editor

Here you can enter code that will be executed, each time this Brick is passed.

The code editor is based on Scintilla. The editor has code highlight support, autocompletion and many more features.

Another nice feature of this code editor is the code-snippet manager, of which the visibility is toggled by F7. And of course editing of the code templates is done in a separate but identical code editor.

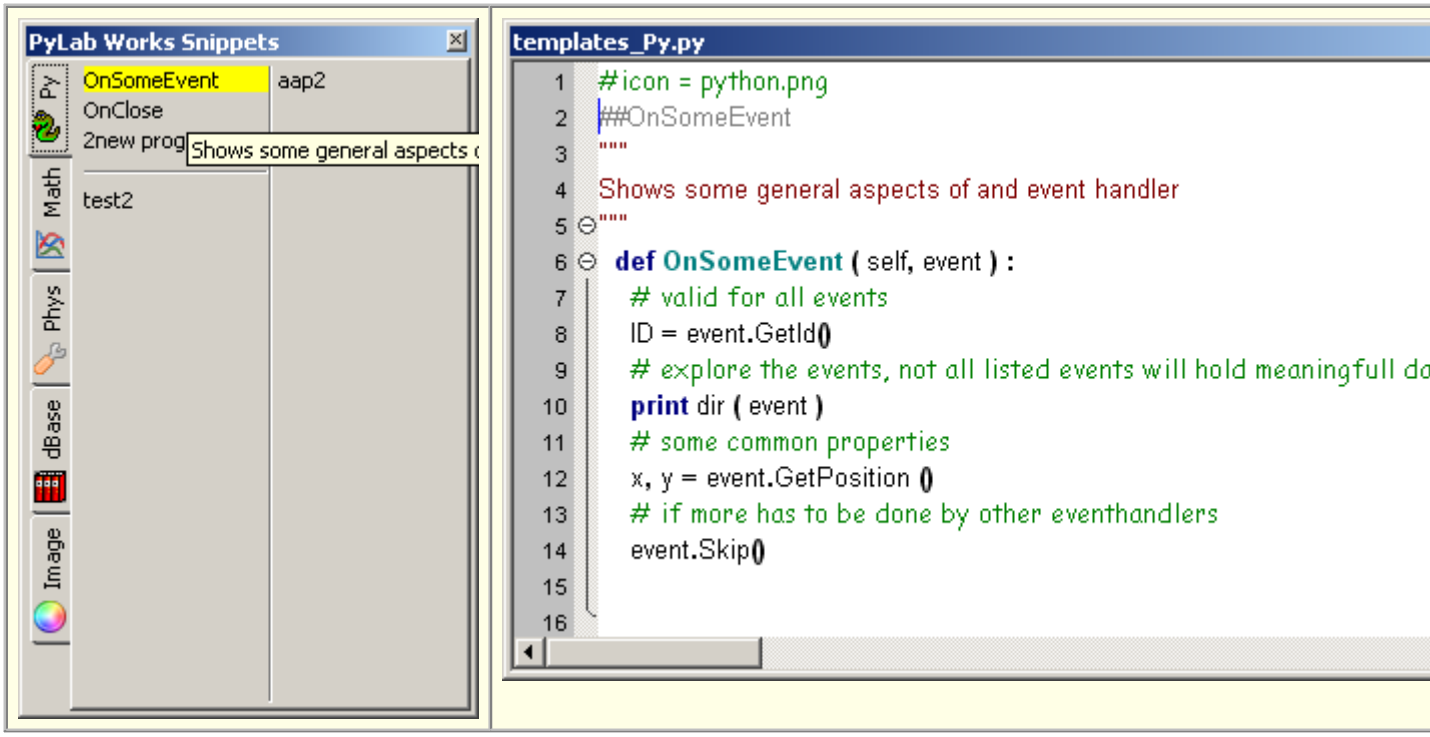


Code Snippets Manager

A code snippet is a (small) pieces of code that can be inserted at the caret position in the editor. The insertion is done is such a way that the indentation (if the caret is not at position 0) is preserved. Organization of a code snippet is very easy. The code snippets manager is made visible (as a stay on top form) by pressing F7. When you hoover over the name of a code snippet, the name will highlight and a hint appears. Hovering while the template file is open, the position in the template file will follow. Clicking on a name will insert the code snippet.

Hoover: highlight + hint + follow code in template window
Left Click: insert in parent or copy to clipboard
Middle Click: nothing at the moment
Right Click: open template editor at this location

Code snippets can be organized in different files, sited in the main program directory, and named "templates_XXXX.py", where XXXX is a short name (otherwise tabs gets too large). The code snippet manager searches for all files of the above mentioned type, will read them and place them in the form. Normally the order of the detected files is random, or at least will depend on the operating system, but you can add a small text file "templates_.txt" in the main directory and list the desired order of the template files. Files that are not in that ordered list, will be placed at the bottom tabs.



The code snippet files can be edited, by right-clicking on a name, the correct file will be opened, at the location of the clicked name. An example of a code snippet file is given in the middle picture above. When the code snippet editor is closed, the code snippet manager is reloaded and the changes made will be visible. What's in the code snippet file from the picture above:
 # icon = , (line 1), a picture that will be displayed on the tab. This picture definition must be on the first line en the picture must be a 16*16 image.
##OnSomeEvent (line 2), the name as it will appear in the snippet window
""" extended explanation """ (line 3..5), lots of text that will used as the hint when the mouse hoovers over the table
the rest (line 6 ..14), the code (with a good indentation) that will be inserted
##- (not shown), insert a horizontal line in the table
##| (not shown), start a new column in the table

Autocompletion

As soon as you type the first character of a new word, the code editor will show you a list of known words (autocompletion-list), with only words that matches the character(s) typed. Continuing to type, will shrink the list dynamically, to only the words that matches all the characters of the word. If a match no longer exists, the autocompletion-list will disappear. Although you can select a word in the list with a mouse click, it's far more convenient to type a few more characters, or select the item with arrow keys. If the desired word is highlighted (and on top) of the autocompletion-list, the easiest way to complete the word is by either typing a space (" ") or a point (".") whichever you need. This will have the effect that the word is completed and the space of point is added. In case of a point, a new autocompletion-list will be opened with suggestions for the word after the point. By pressing the enter key, the word is inserted, and the cursor is placed directly behind the word (no extra insertions). With the Esc-key you can hide the autocompletion-list.

" " = insert the selected word and add a space (" ")
"." = insert the selected word and add a point (".") and start a new autocompletion-list
Enter = insert the selected word and put the cursor behind the selected word
Esc = hide the autocompletion-list

Special Keys

F1	
F2	

F7	Show / focus template windows
F9	Activate changes / Compile
^B ^ScrollWheel Down	increase font size
^N ^ScrollWheel Up	decrease font size

august 2008

Editor / Debugger / IDE 

ToDo:

- variable history must not be shown as strings
- variable overview, add the type of the variable ?
- filenames in multi-page editor, display in correct case
- color the tab of the main file
- ... bug in commands
- breakpoint synchronization after user changes the source
- find the location of winpdb (or let the user set it)
- transporting breakpoints to external winpdb
-

Syntax Check

The py_compile module
30.8 py_compile -- Compile Python source files
This module allows you to explicitly compile Python modules to bytecode. It behaves like Python's import statement, but takes a file name, not a module name. Example: Using the py_compile module
# File: py-compile-example-1.py
import py_compile
# explicitly compile this module
py_compile.compile("py-compile-example-1.py")
The compileall module can be used to compile all Python files in an entire directory tree.

DivmodPyflakes - Divmod - Trac
Pyflakes is a simple program which checks Python source files for errors. It is similar to PyChecker in scope, but differs in that it does not execute the modules to check them.

PyChecker a python source code checking tool
PyChecker is a tool for finding bugs in python source code. It finds problems that are typically caught by a compiler for less dynamic languages, like C and C++. It is similar to lint. Because of the dynamic nature of python, some warnings may be incorrect; however, spurious warnings should be fairly infrequent.

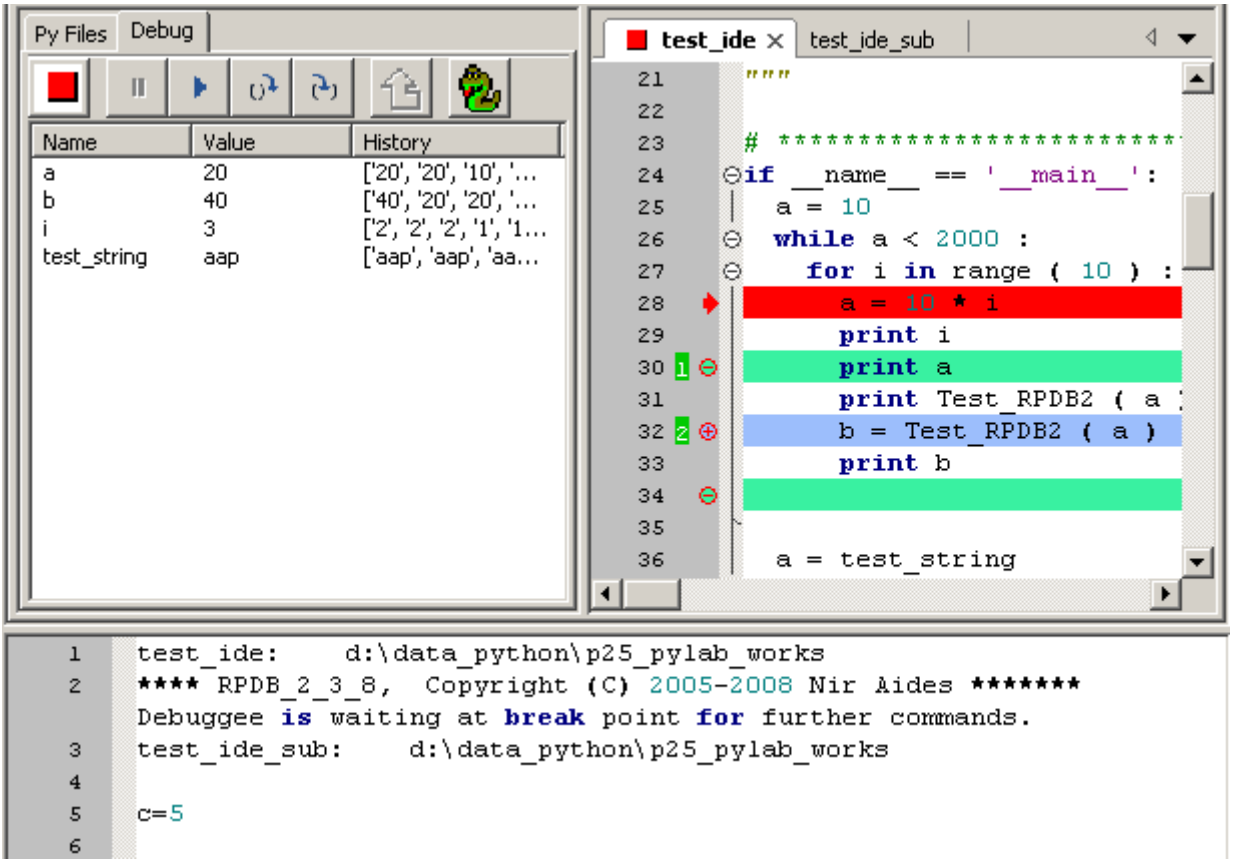
pylint (analyzes Python source code looking for bugs and signs of poor quality.) (Logilab.org)
analyzes Python source code looking for bugs and signs of poor quality. Pylint is a python tool that checks if a module satisfies a coding standard. Pylint can be seen as another PyChecker since nearly all tests you can do with PyChecker can also be done with Pylint. But Pylint offers some more features, like checking

line-code's length, checking if variable names are well-formed according to your coding standard, or checking if declared interfaces are truly implemented, and much more (see the complete check list).
The big advantage with Pylint is that it is highly configurable, customizable, and you can easily write a small plugin to add a personal feature.

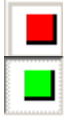
Pygments — Python syntax highlighter

Editor, Debugger

You can choose between a simple internal debugger and the full blown external debugger: Winpdb. Both debuggers are based the wonderful debugger rpdb2, created by Nir Aides (who also wrote Winpdb). Below the simple builtin debugger is shown.
On the left-top the flow control buttons (which all have accelerator keys, shown in the hint). Further on the left an overview of the normal local variables, their current value and their history. On the right the multi-tab editor, shown with bookmarks, breakpoints and current line. During breakpoints you can edit the source files, but the changes will only become active after a restart of the debugging process. On the bottom the standard log-command window. The log-command window is fully editable and be stored and reloaded. The log-command window accepts simple commands just like MatLab, e.g. "5+3", but also accepts full rpdb2 commands. Changes made to local variables will effect the current debug session immediately.



Controls and accelerator keys

		Enable / Disable all Breakpoints. Disabling, first saves the status of all Breakpoints and then disables all the Breakpoints.Enabling restores the previous situation (which is slightly different from winpdb, who enables all Breakpoints, regardless of they were on or off before disabling all Breakpoints)

	F9	Stops a running debug-process.
	F9	Starts a breaked debug-process.
	F8	Single step the debug-process, not going into procedure calls.
	Ctrl-F8	Single step the debug-process, and do enter into procedure calls
	Shift-F9	Restart
	Alt-F9	Start external winpdb. Before winpdb is launched (with the main-file), all modified editors are saved and the current working directory is set to the path of the main-file. You can have many external debug processes and one internal debug proces at the same time (with different versions of the source code).

Autocompletion

AutoCompletion List starts upon

A .. Z, a .. z
0 .. 9
_
Left Arrow
Right Arrow
BackSpace

AutoCompletion List DOESN'T starts upon

Up Arrow
Down Arrow

AutoCompletion is executed upon

.	AutoCompletion + start next AutoCompletion List
(	AutoCompletion + insert (
Enter	AutoCompletion + insert space instead of EOL
Dbl-Click	AutoCompletion

AutoCompletion is NOT executed upon

Space
Tab

Interactive Shell

The Log-Command window can be used as an interactive shell. Text entered on the last line in the Log-Command window will always be evaluated as a command. Other text is assumed to be comment. Simple commands are evaluated in the namespace of the program debugged. If the internal debugger is not running, simple commands are evaluated in the session namespace (starts empty on start of the program).

```
3  9 - 3
4    ==> 6
5
6  dd = 12 * 12
7    dd  = 144
8
9  dd + 6
10   ==> 150
```

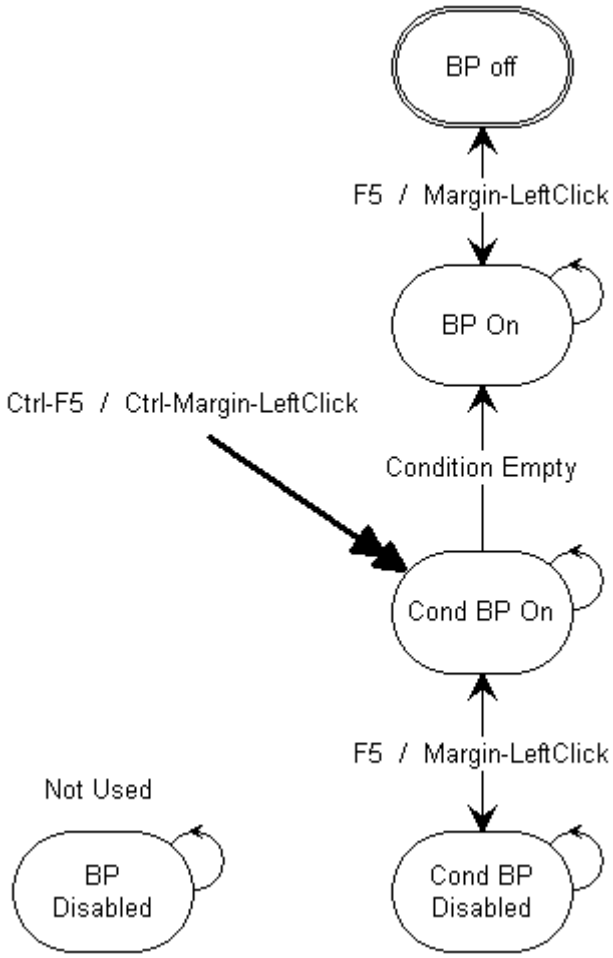
Multi-line commands, which will have auto-indent feature, are evaluated after an empty line is added (press Enter twice).

```
26 ta = []
27   ta  = []
28 for i in range(5) :
29   ta.append ( i)
30 ta
31   ==> [0, 1, 2, 3, 4]
32
```

Debugger commands can be launched by starting the last line with ">":

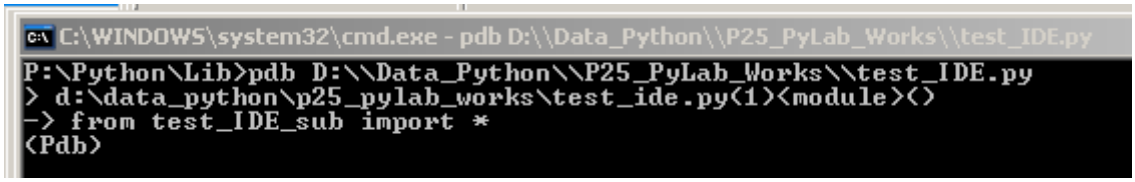
```
11  >bl
12 List of breakpoints:
13
14   Id  State      Line  Filename-Scope-Condition-Encoding
15 ------------------------------------------------------------
16   0  enabled       3  ...\p25_pylab_works\test_ide_sub.py
17                       Test_RPDB2
18
19   1  enabled       9  ...\p25_pylab_works\test_ide_sub.py
20                       Test_RPDB2
```

Breakpoint Statemachine

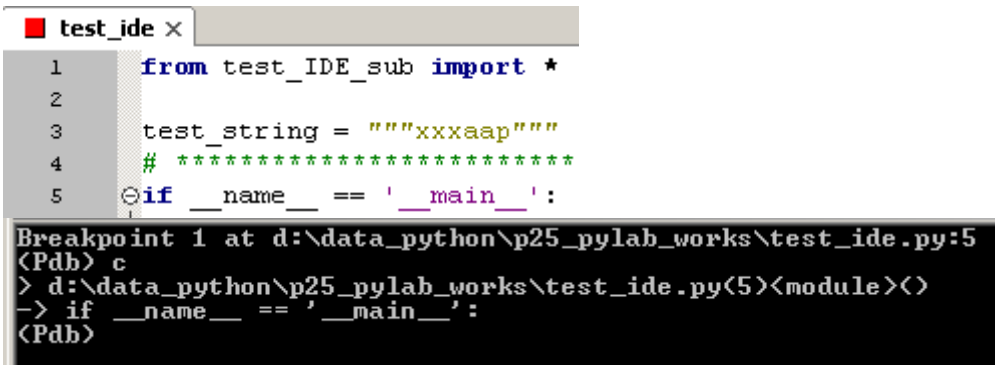


PDB Notes

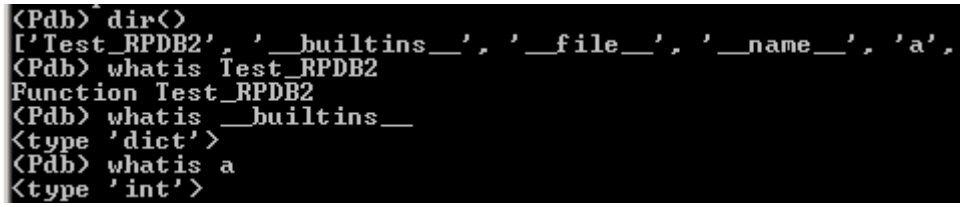
Starting pdb, the debugger is launched and will stop at a temporary breakpoint on the first line of the source file:



Breakpoints use human line numbering, starting at 1.
The line at the breakpoint is not yet executed.



Undocumented "whatis"



Random Notes

Python debugger: pdb, based on bdb + cmd
WinPDB: much nicer / faster

Ctrl-1..9, 0 Shft-Ctrl-1..9,0	Goto BookMark 1..9, 0 Toggle BookMark 1..9, 0 (or click on BookMark Margin)

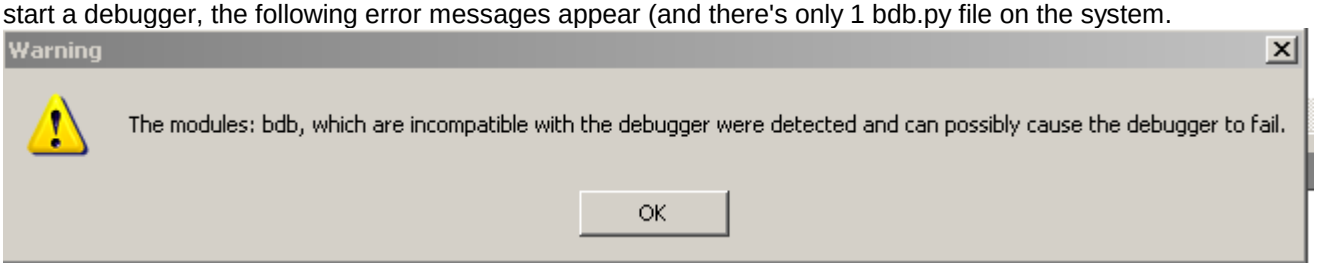
F1	NOT YET: context sensitive help
F2	
F3 Shift+F3 Ctrl + F3	NOT YET: Find Next (from search buffer) shift-F3 = Find Previous Jump to first error
F4 Shift+F4	NOT YET: Find Next from word under cursor shift+F4 = Find Previous This is very handy, when tracing code in the High-Level-language <--> assembler listing.
F5 Ctrl-F5 Shift-F5	Toggle BreakPoint (also Margin-LeftClick) Change BreakPoint Condition (also Ctrl-Margin-LeftClick) Toggle Bookmark (also Shift-Margin_LeftClick)
F6	
F7	NOT YET: Toggle template window visibility
F8	NOT YET: Jump to next field (mainly from template insertion)

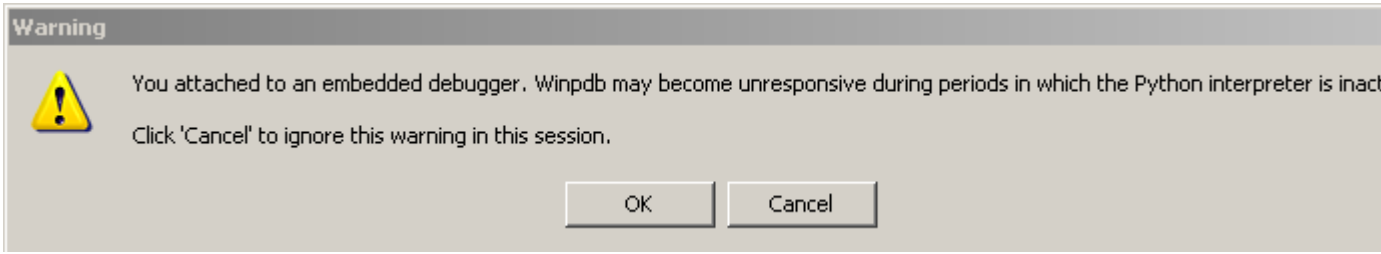
F9 Ctrl-F9 Alt-F9	run NOT YET: run in debugger NOT YET: run in external WinPDB debugger
F10	
F11	
F12	

WinPDB, remarks

started 12 august 2008

- breakpoint condition: counter
- storage of breakpoints in the source directory, so I can carry it around
- step into: only my own source files
- when the to debug is started with: import rpdb2; rpdb2.start_embedded_debugger('aap'), then start a debugger, the following error messages appear (and there's only 1 bdb.py file on the system.





- why not multiple tab source viewers ?
- why not let the user change the source files ?
- how to prevent the "ugly" black command window (DOS box). Commenting line 10391 in rpd2 doesn't help: if python_exec.endswith('w.exe'): python_exec = python_exec[:-5] + '.exe'
- rpd2 takes over the import !!
-

Other IDE

(may 2009)

Application Designer / Domain Expert / Control Designer / **Core Developer**

PyScripter (1.9.9.1)

This is best IDE for Python I've seen so far. It still has some disadvantages:

- only available under windows
- written in Delphi
- crashes about once an hour
- needs RPYC to run programs that contains wx, ...
- Code completion sometimes works, sometimes not, after 2 years I still don't know when
- BreakPoints sometimes works, sometimes not
- much too many unused buttons / info / tabs / etc

- having many files open, difficult to navigate
- Ctrl-F (search in document) sometimes keep focus in editor
-

SPE

SPE looks like a Chinese copy of PyScripter. Doesn't remember most settings, all kinds of unnecessary questions, doesn't jump to lines, etc. As the difference between PyScripter and SPE was too large, I looked for another IDE, that could be used in both Windows and Ubuntu, and that brought me to Editra.

Editra

Serious omissions

- !!! no autosave on run !!!
- no auto checker during typing
- search on selection / word under cursor is missing
- you can't search in the output window

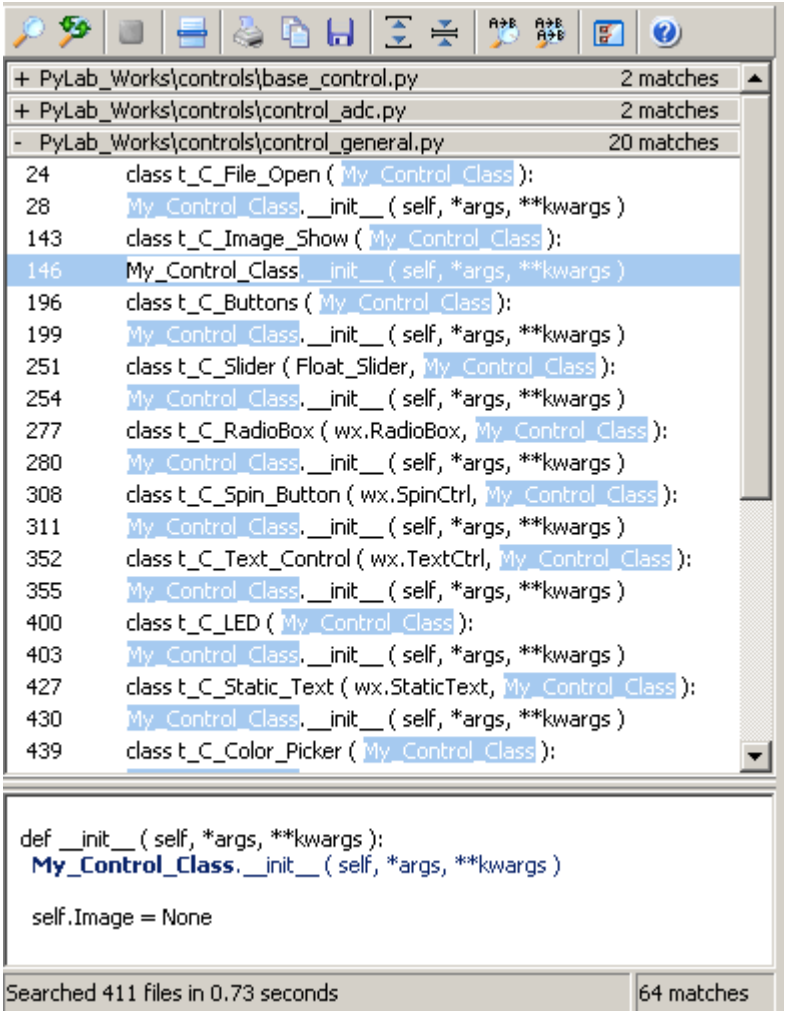
Small omissions

- missing autocompletion of quotes, brackets, ...
- missing (integrated) debugger
- Transparency setting of the main form, sound totally weird to me
- "self" is handled as a keyword ???
- quick search bar: no "whole words" switch, sometimes cursor flickers weird in search window, keys are not always (maybe never) focused back to active editor window, doesn't always remembers history
- you can't select a main file that will be run, when a different editor tab is open or even if the main file is not in an editor tab at all
- checkbox "clear buffers between runs" might be very handy
- hoover info is missing
- search in files forgets the directory
- search in files: up /down ?
- bug: when have the extended search window open, minimize the application, restore the application: main window doesn't show up
- when there are lines longer than the width of the screen, annoying jumping of text
-

Search

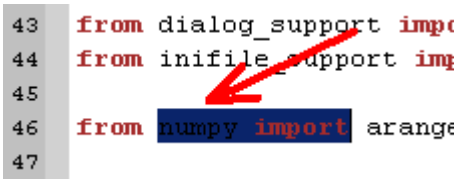
Search should start with the **selection** within the current file and if no selection is available, the **current word** should be used.
Case Match, Whole words, Search in comments should all be selectable.
The search strings should be maintained in a **history list**.
In the results the **search string** should be **highlighted**.
A **preview** of a (selectable) number of lines is sometimes very handy.

Here an example of PyScripter, quit good, but it contains a lot of **non information**: buttons, number of files searched, searchtime.



Selection

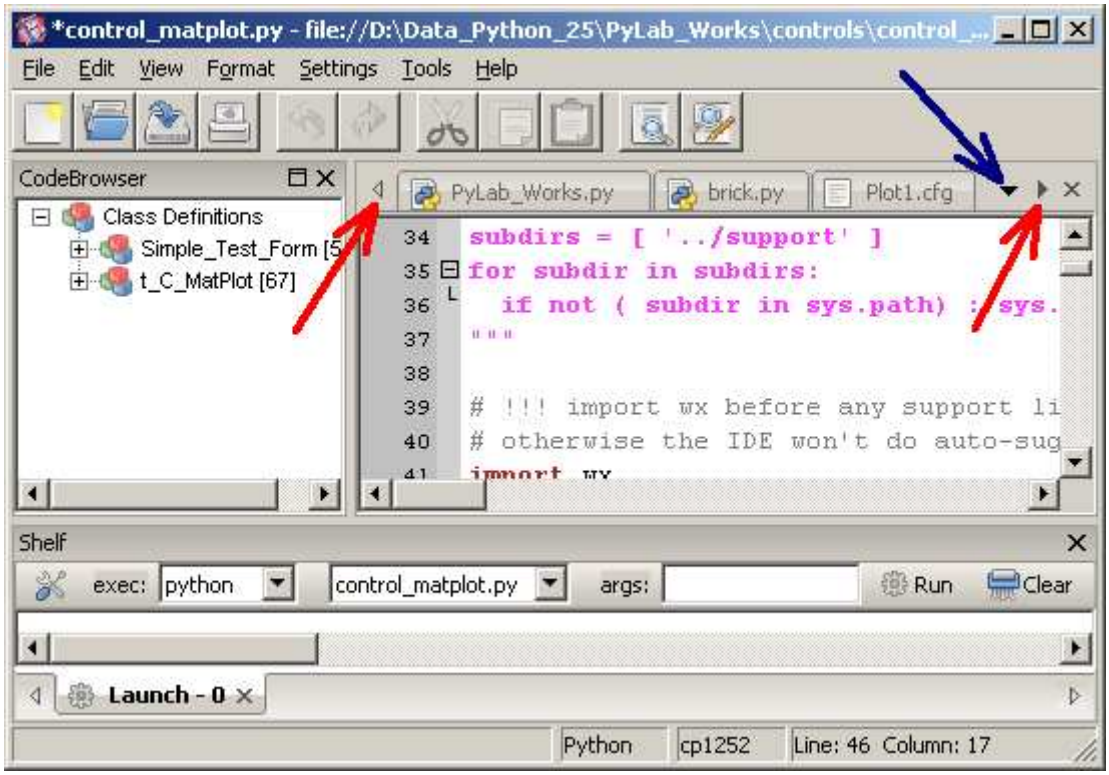
In Editra, selected words become invisible:



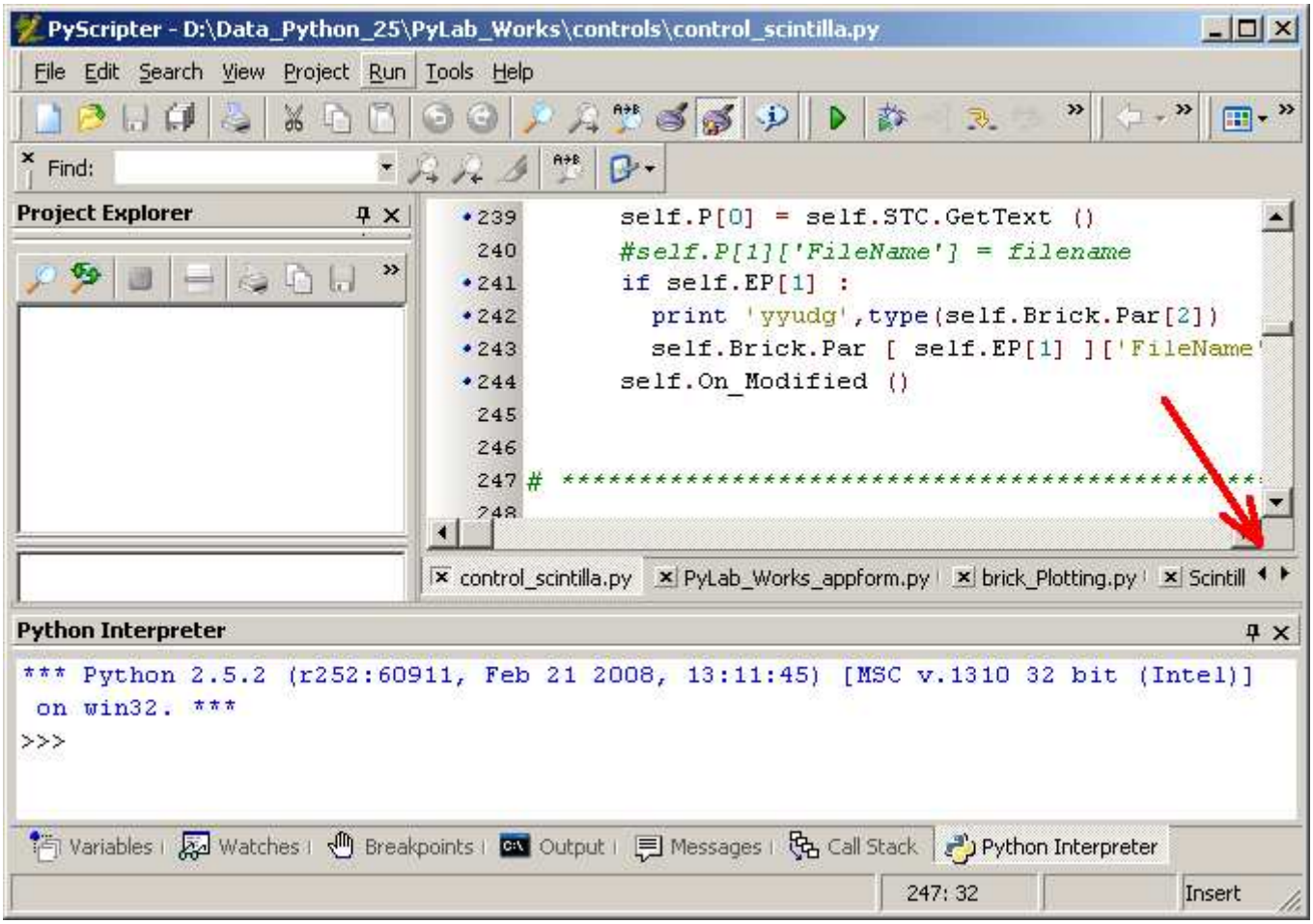
Many open files

Having many files open, it's difficult to navigate between the files.

Editra at least has the advantage that on the right there is an dropdown list (blue arrow).



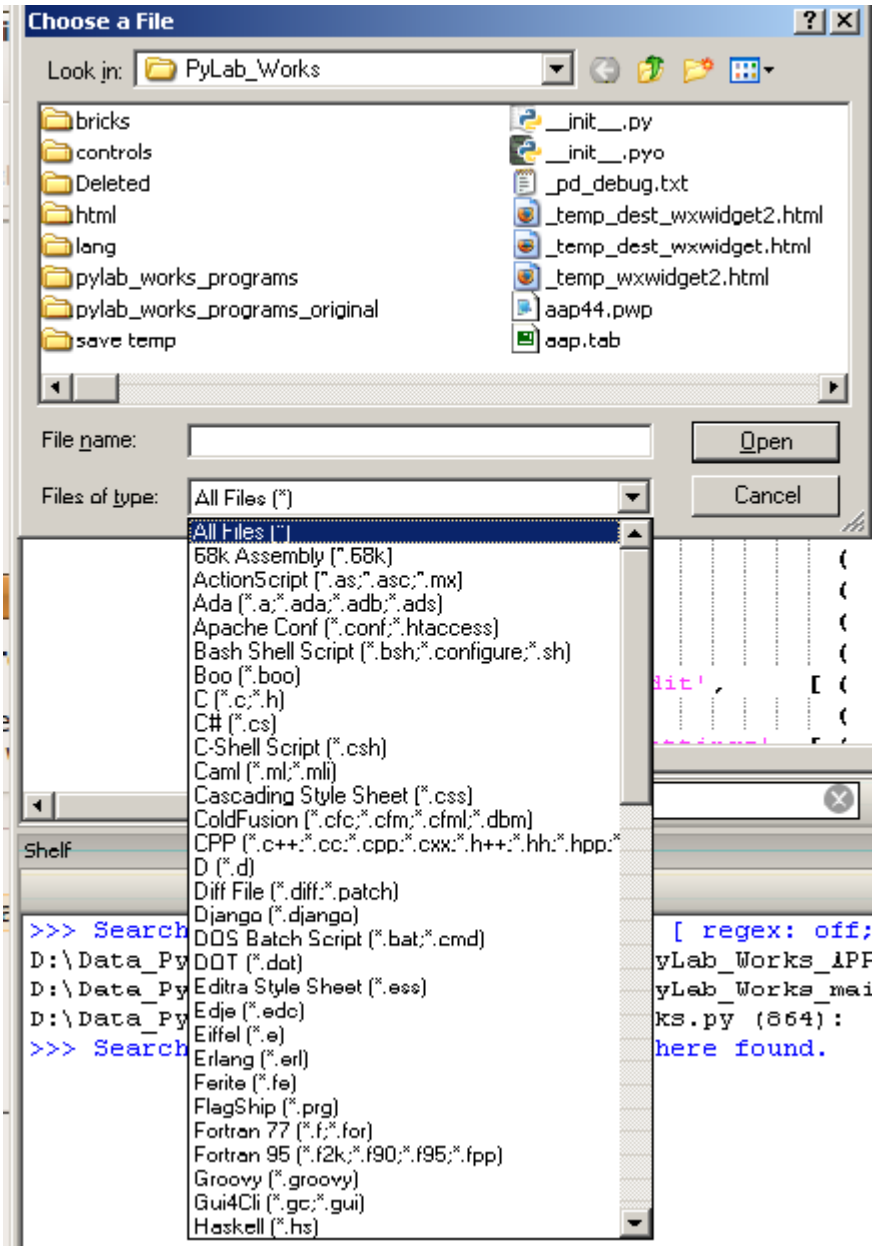
PyScripter has only controls on the right side.
Closing files, don't scroll in the files from the left.



Run Args

args are not saved, args field could be filling up the complete space

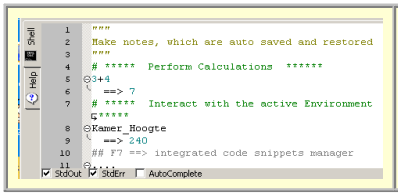
Other Non-information

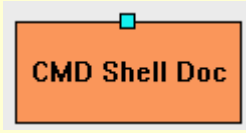


Library Python

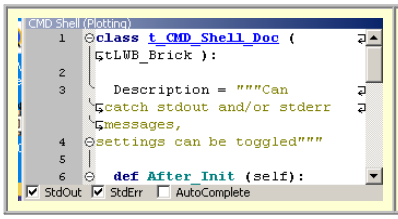
(december 2008)

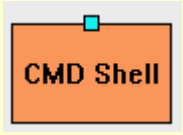
Application Designer / Domain Expert / Control Designer / Core Developer



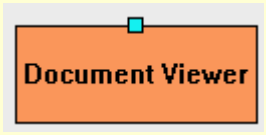


CMD_Shell_Doc = CMD_Shell + Document_Viewer
Interactive command shell combined with a general purpose document viewer.



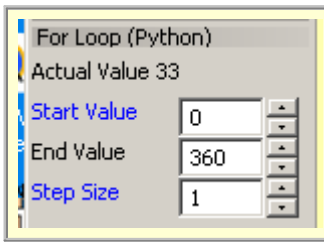


CMD_Shell
Interactive command shell.



Document_Viewer
General purpose document viewer.

does this need an input ?



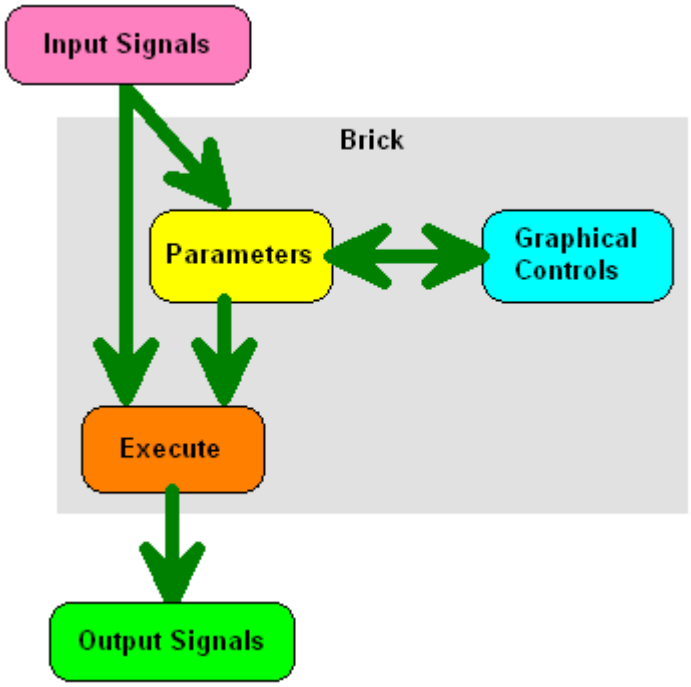
For_Loop

PyLab Works Bricks Library (march 2008)

Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

We have chosen for a human indexing, so all indices start at 1 (whereas Python normally starts at 0).
For the descriptions below, we've used the Rotate Brick from the Image Library.



Creating New Brick

The easiest way to create a new brick is to start with the template "Brick Template", correct the Inputs, Outputs and Controls. In this template, only the "After_Init" method is used and the "Generate_Output_Signals" is replaced by a dummy, that will print all available variables when the method is called. In this way, you can use the Brick in an application, connect it to other Bricks, see the visual presentation and test the user interface including most events.

```
3   ## Brick Template
4   # ****************************************************************
5   # ****************************************************************
6   class t_MYNAME ( tLWB_Brick ) :
7
8     Description = """Extended Description of this Brick"""
9
10    def After_Init (self):
11      self.Caption = 'My Caption'
12
13      # Define the input pins
14      # <Pin-Name>, <Data_Type>, <Required>, <Description>
15      self.Inputs [1] = [ 'SQL', TIO_STRING, False, 'SQL query' ]
16
17      # Define the output pins
18      # <Pin-Name>, <Data_Type>, <Description>
19      self.Outputs [1] = [ 'Meta Data', TIO_TREE, 'Tables, etc' ]
20
21      # Create the GUI controls
22      C = self.Create_New_Control ( 'TO_pat.db' )
23      C [ 'Type'     ] = CT_FILEOPEN
24      C [ 'Range'    ] = FT_DBASE_FILES
25      C [ 'Caption' ] = 'dBase FileName'
26
27      # Temporary use the Debug Output Generator
28      self.Generate_Output_Signals = \
29        self.Generate_Output_Signals_Debug
```

When the above part works, remove the dummy "Generate_Output_Signals" and replace by a real

implementation:

```
31    # ********************************************************************
32    # ********************************************************************
33    def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
34      # The actions of this Brick
35      pass
```

Internationalization & Localization

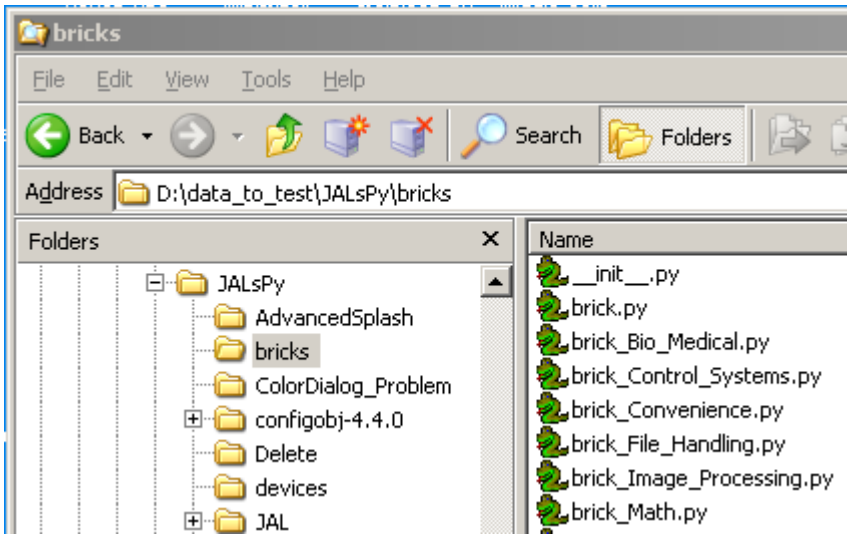
If we want to make this program available for many different users, localization is an important issue. I looked at IL8N / IL10N, which seems to be the de-facto standard, but found it much too complex, especially very difficult to incorporate the system with an inplace editor. Therefor we use for the time being a much simpler (but less complete) translation model.

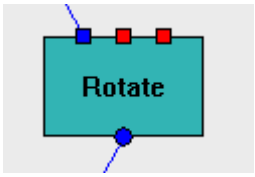
Library files

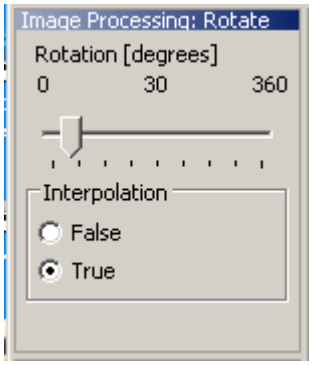
Library files are stored in the subdirectory "bricks\" and each library file should start with "brick_".
If you start a new library file, be sure to give it the right name with the right upper/lower-case, because it's difficult to change e.g. letter-case afterwards (at least under windows).
The easiest way to start a new library is to make a copy of an existing library. The library "brick-Image_Processing.py" is a good example, it contains an only-output brick (t_Read), and only-input brick (t-Show) and a brick with both input and output and parameters (controls) interfering with some of the inputs (t_Rotate).

Special tricks

A control can claim a field in the statusbar.







Brick Definition / Description

```
88 # ****************************************************************************
89 # ****************************************************************************
90 class t_Rotate ( tBrick_Image_Processing ):
91
92   Description = """Rotates an Image counter clockwise.
93 The rotation angle (in degrees) and the interpolation are either taken from
94 the internal GUI controls, or from the external signals,
95 whichever has been changed last."""
96
97   def After_Init (self):
```

Brick Inputs

```
 99     # Define the input pins
100     # <Pin-Name>, <Data_Type>, <Required>, <Description>
101     self.Inputs [1] = \
102       ['Image', TIO_IMAGE, True,
103        'Accepts images of the type wx.Image:\n']
104     self.Inputs [2] = \
105       ['Rotation [degrees]', TIO_NUMBER, False,
106        'The rotation angle is taken modulo 360 degrees.\n'
107        'If this input signal is not present,\n'
108        'the rotation angle is taken from  the internal GUI.']
109     self.Inputs [3] = \
110       ['Interpolation', TIO_NUMBER, False,
111        'If False, faster calculation,\n'
112        'If True, better quality.']
```

Brick Outputs

```
114     # Define the output pins
115     # <Pin-Name>, <Data_Type>, <Description>
116     self.Outputs [1] = \
117       ['Image', TIO_IMAGE,
118        'Rotated image in the same format as the input.']
```

Brick Edit-Shape

```
 90     # Create the shape and set the caption
 91     self.After_Init_Default ( 'Rotate' )
```

Brick Controls / Parameters

```
 93     # Create the GUI controls
 94     C = self.Create_New_Control ( 30 )
 95     C [ 'Type' ]          = CT_SLIDER
 96     C [ 'Range' ]         = [0,360]
 97     C [ 'Caption' ]       = 'Rotation [degrees]'
 98     C [ 'Input Channel' ] = 2
 99     self.Controls.append ( C )
100
101     C = self.Create_New_Control ( True )
102     C [ 'Type' ]          = CT_RADIO
103     C [ 'NCol']           = 1
104     C [ 'Range' ]         = ['False', 'True']
105     C [ 'Caption' ]       = 'Interpolation'
106     C [ 'Input Channel' ] = 3
107     self.Controls.append ( C )
```

Brick Generate Output Signals

This function calculates the new outputs on the base of changed inputs and / or changed parameters. If the brick doesn't generate output signals, no execute function is required. The ancestor of brick will check if input values and or parameters were changed and will decide if the execute function should be called or not.

```
109   # ************************************************************************
110   # Procedure only called when inputs and/or parameters have changed
111   # ************************************************************************
112   def Generate_Output_Signals ( self ) :
113     self.Output_Value[1] = \
114       self.Input_Value[1].Rotate ( pi * self.Params[1] / 180,
115                                    (0,0), self.Params[2] )
116 # ************************************************************************
```

Using Bricks from other libraries

Suppose you've an Audio library, that has a brick with a good working wav-file player, and now you want to design a Media-library. In that case you don't want to copy the functionality already available, but just want to add a name to the Media Library. The user will see the wav-player in the media library, even if the Audio Library is not visible to the user. The implementation is simply:

```
784 from brick_Media import t_Play_Sound
785 class t__Play_Sound ( t_Play_Sound ) :
786   pass
```

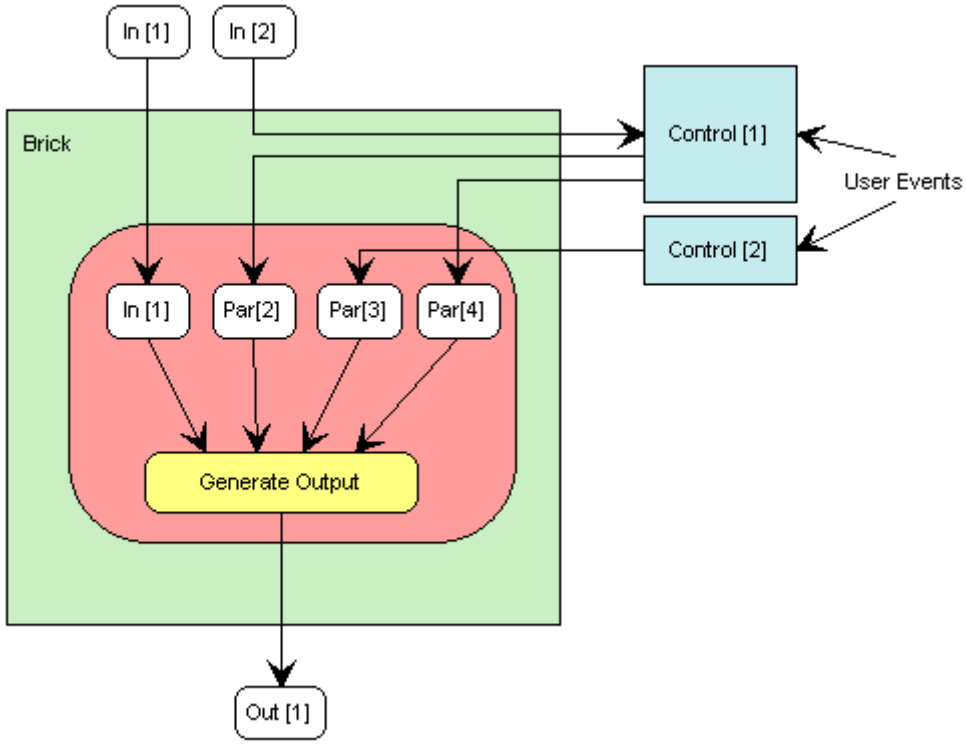
Brick Controls (inside)  (january 2009)

Application Designer / Domain Expert / **Control Designer / Core Developer**

Introduction

The general task of a Brick is to generate output, i.e. visual feedback to the user and / or new output signals

for the machine, based on a combination of inputs and / or user events. In the schematic below this task of a virtual Brick is visualized. In practice it's a bit more complicated, because inputs can also serve as outputs and vise-versa, but that's explained later on.



Notice that all indices starts at 1. This is mainly done to make numbering more human. As an extra advantage we now can use elements 0, for special purposes.

For each Input signal a parameter is created. A Control has at least 1 parameter, but might have more. It's up to the Brick, whether it will use these parameters or not.

There are direct inputs, like In[1], which have no connections with controls, and are only used in the calculation of the output signal.

Other inputs, like In[2], are tightly coupled to controls. In this case In[2] is coupled to Control[1]. The output generate function can't use In[2] for it's calculations, instead it must use Par[2], the result coming from the control. In that way we can prevent race conditions, assure that the latest action, either from the input or from the user is honored. Also the range of input signal can be limited within the control, and the input might even be omitted (is not required anymore). Notice that the index of the Par and the Input signal are equal. In some cases there isn't even a feedback from the control to the par, like in the Brick "Show Image", in which case also the generate output function isn't needed anymore, and thus the input sends its signal directly to the control.

Finally there are controls, like Control[2], that only react on user events, and thus has no connections with input signals.

Brick Definition

Goal is to make the creation of a brick as simple as possible. A second requirement is that the code running in the virtual machineshould be as clean as possible, without too many tricks.
Until now, this is the simplest solution found (the code below is approximately equal to visual Brick shown above) :

```
66 # ************************************************************************
67 # ************************************************************************
68 class t_Dummy ( tLWB_Brick ) :
69
70   Description = """Extended Discription of this Brick"""
71
72   def After_Init (self):
73     self.Caption = 'My Caption'
74
75     # Define the input pins
76     # <Pin-Name>, <Data_Type>, <Required>, <Description>
77     self.Inputs [1] = ['Pin-1 Name', TIO_NUMBER, True, 'Description' ]
78     self.Inputs [2] = ['Pin-2 Name', TIO_NUMBER, True, 'Description' ]
79
80     self.Outputs [1] = ['PinOut-1 Name', TIO_NUMBER, 'Description' ]
81
82     # Create the GUI controls
83     C = self.Create_New_Control ()
84     C [ 'Type' ]          = CT_SOMETHING
85     C [ 'Input Channel'] = 2
86
87     # Create the second GUI controls
88     C = self.Create_New_Control ()
89     C [ 'Type' ]          = CT_SOMETHING
90
91   def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
92     Out[1] = 2 * In[1] + Par[2] + Par[3]
93 # ************************************************************************
```

line 68: This class is automatically detected as a Brick, because its name starts with "t_". The class name without the leading "t_" is used as the name in the Libraries list. All Bricks should be derived from tLWB_Brick.

line 70: An extended description of this brick, which is shown to the user when the user hoovers over the library list or over the Brick.

line 72: The method After_Init is always required, and here the inputs / outputs and controls are defined.

line 73: The caption is used to give the brick a visual name. While we could use the class name for the caption, we've decided not to do so, because in that case our internationalization of the visual name is gone (For simplicity, the internationalization is left out in this example).

line 77,78: Definitions of the inputs. The first parameter is a short name for visual feedback to the user. The last parameter is an extended description used as feedback to user when he hoovers over the Brick. The second parameter is the type of the signal. The third parameter tells if this input needs to be present or not before a calculation will be performed.

line 80: Definitions of the outputs. The parameters are the same as for the inputs, except that the "Required" flag is missing.

line 83-85: Definition of a control, that is connected between In[2] and the calculations.

line 88,89: Definition of a control, that is not connected to an input. Depending on the complexity of the control, more parameters might be needed.

line 91: The output generation method. This method is only needed when new output signals need to be generated. The function parameters are all variables of the class itself, so it looks a bit overdone, but the advantage is that the formula becomes much clearer, because we don't need the "self" prefix.

line 92: Here the new outputs are calculated.

Loop Code

The code executed in the loop of the virtual machine is very clean and straight forward , and is very well suited for single stepping or breakpoint debugging.

```
1809     Loop_Code = 'for Brick in PG.Bricks: Brick.Exec()\n'
```

Communication

Both In and Out signals (and even Par) are derived from an intelligent list:

```
177 class IOP_List ( list ) :
```

This list detects changes in the values and has some extra properties and methods for an easy communication.

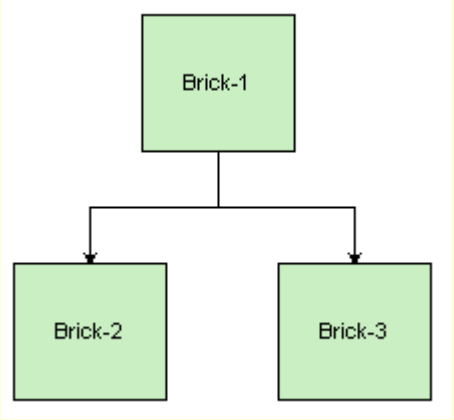
```
186     self.Brick = Brick
187     self.Modified  = leng * [ False ]
188     self.Receivers = leng * [{}]
189     self.IO_Par    = leng * [ None ]
```

Receivers = list of dictionaries, for each pin a dictionary. When a connection is made, any brick (possible even the brick itself) that wants to receive this signal, add itself and the relevant pin to this dictionary. This list is used by the transmission function, to send a new value to all the receivers. This list is quit static and only when a connection is removed, one or more (in case the output brick is removed) bricks are removed from this list.

In the case on the right, the Receivers of the Input signals of Brick_2 and Brick_3 will be empty. The Receiver list of Out[1] of Brick_1 will contain the inputs of Brick_2 and Brick_3:

```
 97 Brick_1.Out.Receivers[1] =
 98 {
 99   Brick_2 : ( 'In', 1 ),
100   Brick_3 : ( 'In', 1 )
101 }
```

After the Exec method of Brick_1 is performed, the brick will send all changes to the Receivers (which are also of type IOP_List and thus will detect changes).



Bi-Directional IO

When in the above figure, Brick_3 wants to send information over the pin In[1], it has to use the same receiver list as the Receiver list of Out[1] of Brick[1], so it simply assigns the receiver list

```
104 Brick_3.In[1].Receivers = Brick_1.Out[1].Receivers
```

The other side of the story, if Brick_1 wants to receive information on it's Out[1], it has to add itself to the receiver list

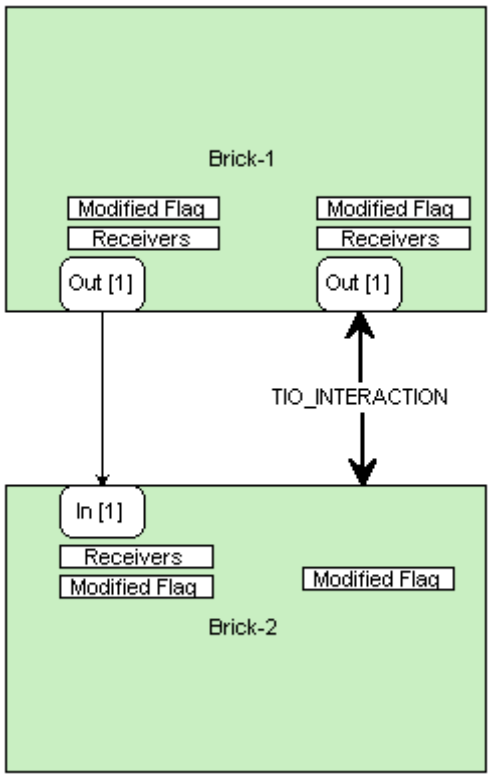
```
106 Brick_1.Out.Receivers[1] [Brick_1] = ( 'Out', 1 )
```

and now the Receiver list will look like

```
 97 Brick_1.Out.Receivers[1] =
 98 {
 99   Brick_1 : ( 'Out', 1 ),
100   Brick_2 : ( 'In', 1 ),
101   Brick_3 : ( 'In', 1 )
102 }
```

The Transmit_Changes method will send changes to items in the dictionary, except if an item is the sender.

```
110   def Exec (self):
111     XIn  = copy.copy ( self.In. Modified )
112     XOut = copy.copy ( self.Out.Modified )
113     self.In. Clear_Modified ()
114     self.Out.Clear_Modified ()
115
116     # Transport Input Parameters to the control
117     if XIn[0] and (len ( self.Controls ) > 0 ):
118       self.Control_Pane.Brick_2_Control ( self.In )
119
120     # Check for changes
121     # and Generate output signals
122     if XIn[0] or XOut[0] or self.Par.Modified [0] :
123       self.Generate_Output_Signals (
124         self.In, self.Out, self.Par,
125         XIn, XOut, self.Par.Modified )
126       self.Par.Clear_Modified ()
127
128     # Transport changed values to receivers
129     self.In.Transmit_Changes  ()
130     self.Out.Transmit_Changes ()
```



IO Definitions  (November 2008)

Application Designer / Domain Expert / Control Designer / **Core Developer**

For the moment these are declared in PyLab_Works_Globals, don't know if this is the best place ??

```
159 TIO_NAMES = {
160   TIO_NUMBER      : 'Number',
161   TIO_LIST        : 'List',
162   TIO_ARRAY       : 'Array',
163   TIO_IMAGE       : 'wx.Image',
164   TIO_CALLBACK    : 'CallBack',
165   TIO_STRING      : 'String / StringList',
166   TIO_TREE        : 'TreeList',
167   TIO_GRID        : 'GridData',
168   TIO_INTERACTION : 'IO Communication',
```

Colors are defined in OGL_Like

```
global TIO_Brush
TIO_Brush = [
    wx.Brush ( ( 255, 0,   0   ) ),
```

TIO_ARRAY

This variable type is suited to pass both array data and meta data, in a very free and flexible form.

Calibrate	Either a 2 points calibration (x1, y1, x2, y2) or a gain / offset tuple (gain, offset)
Channel	Zero based channel number
DisplayRange	(Bottom,Top) value of the display (in world units). Might also have the value "True" for autoscaling.
Frequency	Sample frequency of the signals [Hz]
LineColor	Color of the line used to draw the signal
LineWidth	Line width used to draw the signal [pixels]
SignalName	Name of the signal, possible followed with world units in square brackets, e.g. "BloodPressure [mmHg]"
Units	World Units, without square brackets (this will overrule the units specified in the SignalName)

TIO_INTERACTION

The type of this IO is a dictionary, so it can contain any information and it works bi-directional.

AutoCompletion_List	Extra autocompletion elements
FileName	
CodeFile_To_Open	Request from a secondary control, e.g. Cmd_Shell_Doc, to open this code file. Cmd_Shell_Doc contains a gridpage with files (ordered into groups) in the projects file directory. Clicking on a cell in this grid, will send a request to open this code file. The project VP2 has an example of this use.
CodeText	The complete text from a code editor
Dont_Execute	

CodeFile_To_Open

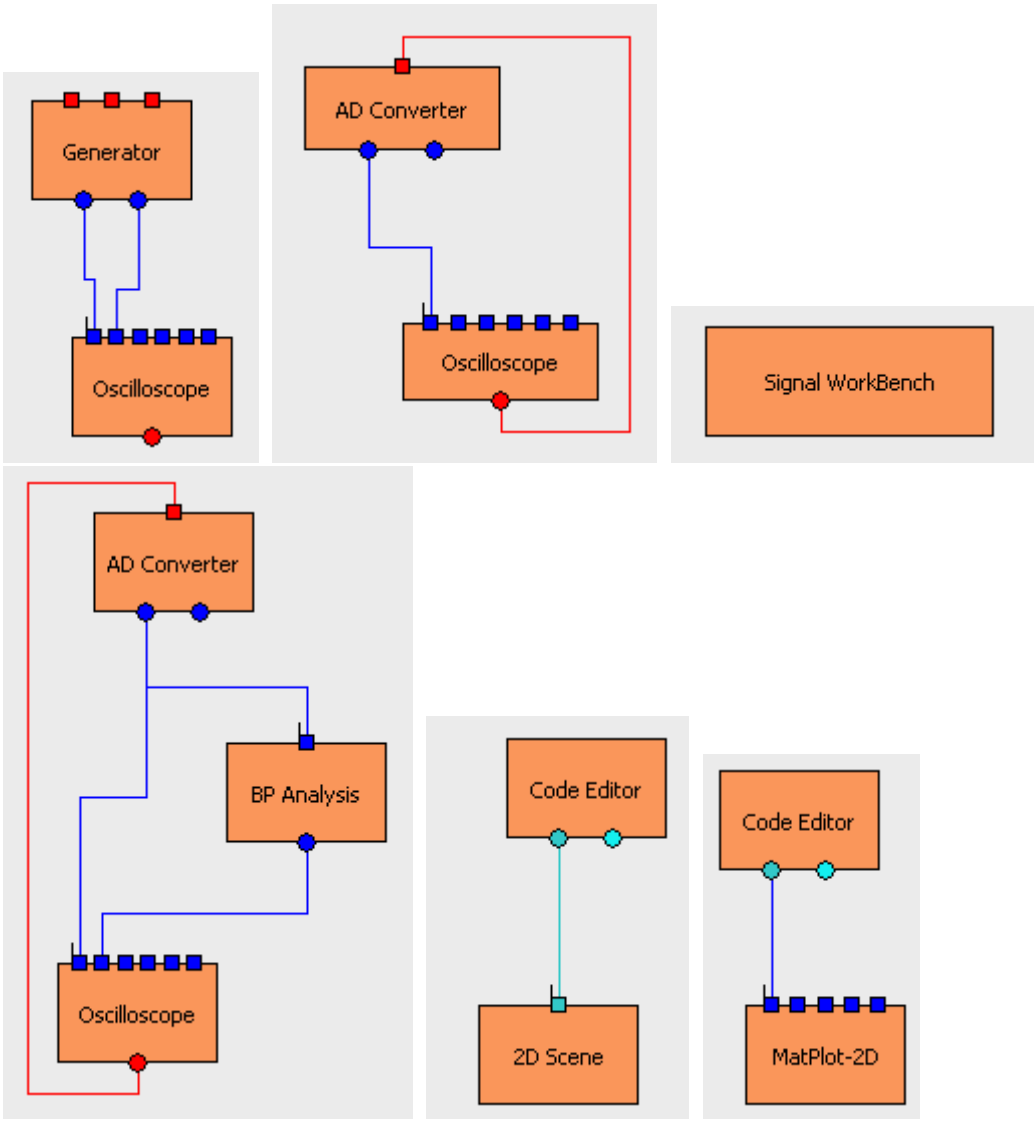
In control_output_viewer.py the class Cmd_Shel_Doc we detect a cell click, which in fact is a file selection, and there we change the params of the parent Brick:

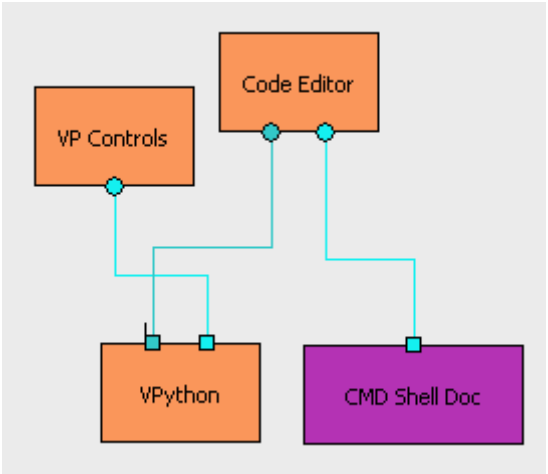
```
358     self.Grid.Bind ( gridlib.EVT_GRID_SELECT_CELL, self.OnSelectCell )
359
360   # ************************************************************
361   # ************************************************************
362   def OnSelectCell(self, evt):
363     col = evt.GetCol ()
364     row = evt.GetRow ()
365     value = self.Grid.GetCellValue ( row, col )
366     # If we realy selected a file,
367     # pass it to the Brick
368     if value :
369       prefix = self.Grid.GetColLabelValue ( col )
370       filename = prefix + '_' + value + '.py'
371       path = os.path.join ( Application.Dir, PG.Active_Project_Filename )
372       self.Brick.Params [0] = os.path.join ( path, filename )
373     evt.Skip()
```

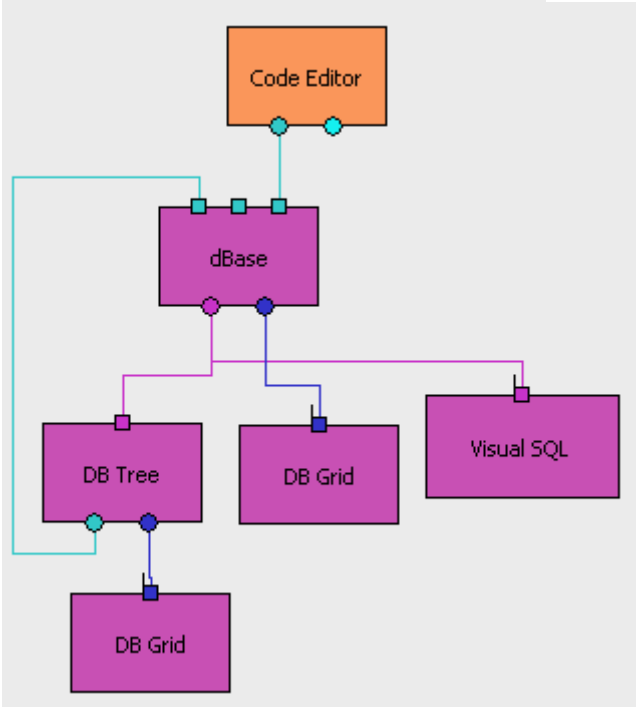
In the Brick t_Code_Editor the change of params is detected and the signal is passed to the TIO_INTERACTION signal

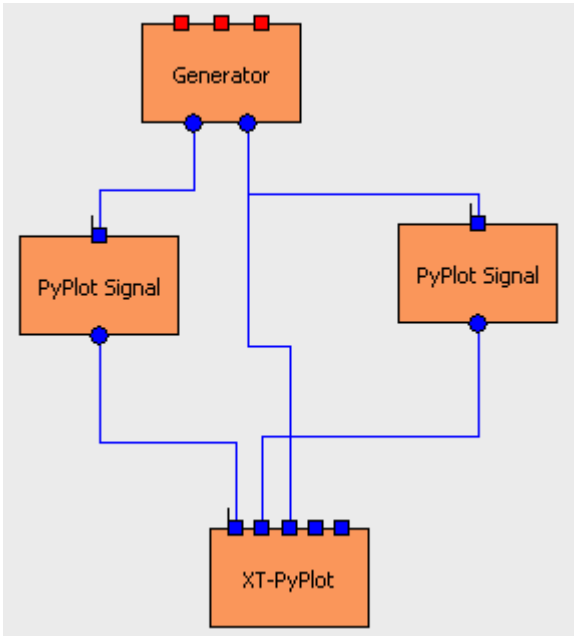
```
5  # ********************************************
6  class t_CMD_Shell_Doc ( tLWB_Brick ):
7
8    def After_Init (self):
9     ...
10
11   # ********************************************
12   # Procedure only called when inputs and/or para
13   # ********************************************
14   def Generate_Output_Signals ( self ) :
15     #print 'CMD SHELL IN',self.Input_Value
16     IO = self.Input_Value [1]
17
18     if self.Params[0] != self.Params_Old[0] :
19       print 'Open another file',self.Params[0]
20       IO [ 'CodeFile_To_Open' ] = self.Params[0]
```

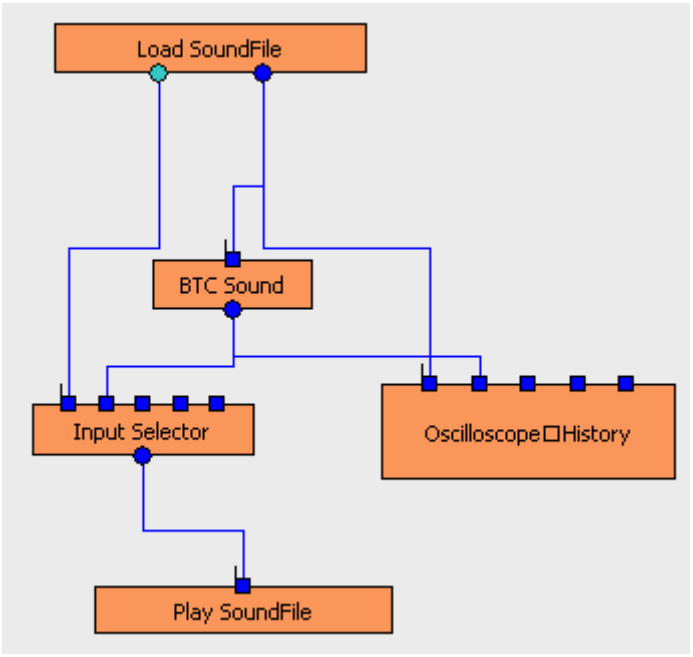
Now if we've a code editor attached to it

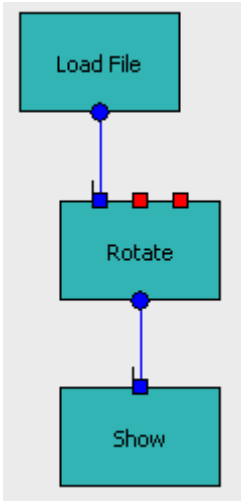



Code Editor
VP Controls
VPython
CMD Shell Doc


Code Editor
dBase
DB Tree
DB Grid
Visual SQL
DB Grid


Generator
PyPlot Signal
PyPlot Signal
XT-PyPlot


Load SoundFile
BTC Sound
Input Selector
Oscilloscope☐History
Play SoundFile


Load File
Rotate
Show

april 2008

Bricks, create Numeric Display 

Introduction

Suppose we want to display a numerical value and there's no numerical display control yet. Using an editbox as the control doesn't work, because an editbox requires a string as input, so it will fail when numbers are used as input.

The first thing to do is to claim an **unique ID in bricks.py**:

```
31 CT_SLIDER            = 1
32 CT_RADIO_BUTTON      = 2
33 CT_FILEOPEN          = 3
34 CT_CHECKBOX          = 4
35 CT_SPIN_BUTTON       = 5
36 CT_STATIC            = 6
37 CT_BUTTONS           = 7
38 CT_LED               = 8
39 CT_COLOR_PICKER      = 9
40 CT_EDIT              = 10
41 CT_EDITOR            = 11   # Scintilla Code Editor
42 CT_SIGNAL_WORKBENCH  = 12
43 CT_STD_VIEWER        = 13   # Viewer for STDOUT and/or STDERR
44 CT_NUM_DISPLAY       = 14   # Numerical Display
```

As wxPython doesn't have a numerical display yet, we've to create one. One of the simplest way to do this, is to derive the new control from wx.StaticText. And the only method we've to add/override is the SetValue method. So our new control might look like this:

```
16 # ***************************************
17 # A component to display a numerical value
18 # ***************************************
19 class Static_Float ( wx.StaticText ):
20   def SetValue ( self, Value ) :
21     self.SetLabel ( str ( Value ) )
22 # ***************************************
```

As this component is so simple, we put in directly in PyLab_Works_appform.py, where also the control will be created and handled. Now we're going to use this component, to create the new control:

```
326       # ********************************************************
327       # NUMERICAL DISPLAY
328       # ********************************************************
329       elif self.C[ 'Type' ] == CT_NUM_DISPLAY :
330         X = self.x + 5
331         Y = self.y + 2
332         X, Y = self.Display_Label ( X, Y )
333
334         Static = Static_Float ( self, -1, 'XXX', pos = ( X, self.y + 2) )
335         if odd : Static.SetForegroundColour ( AC )
336         Control_Pars = self.Create_Control_Instance ( Static, Dont_Scale )
```

Now we have the control, we can make a Brick, containing this component:

```
55 # ************************************************************
56 # ************************************************************
57 class t_Numeric_Display( tLWB_Brick ):
58   Description = _(0, 'Display a number' )
59
60   def After_Init(self):
61     # Define the input pins
62     self.Inputs [1] = [ _(0, 'Number to display'), TIO_NUMBER ]
63
64     # Create the shape
65     self.After_Init_Default ( _(0, 'DisplayNum' ) )
66
67     # Create the GUI controls
68     C = self.Create_New_Control ()
69     C [ 'Type'          ] = CT_NUM_DISPLAY
70     C [ 'Caption'       ] = _(0, 'Displayed Number' )
71     C [ 'Input Channel'] = 1
72 # ************************************************************
```

The above Brick is very straight forward, except line 71:

- Input Channel 1, will control Params[1], and as Params are processed automatically, we don't need any further action to let the control work.

You can spruce up the component, giving it another color, a larger font etc., like this ...

```
67     # Create the GUI controls
68     C = self.Create_New_Control ()
69     C [ 'Type'          ] = CT_NUM_DISPLAY
70     C [ 'Caption'       ] = _(0, 'Displayed Number' )
71     C [ 'Input Channel'] = 1
72
73     C [ 'Color']          = ( 217, 200, 111 )
74     C [ 'FontSize']       = 22
```

... but then of course you have to expand the Numerical Display control to act on this issues.

Scope Internals  (april 2009)

These are the preferred Plot elements, both for static and dynamic display of xt-data. It has the following features:

- it's fast
- it accepts both static and dynamic data
- it's very flexible
- it's actively maintained

t_Scope_Display (Brick with extended Scope)

t_C_Scope_Display (Control with extended Scope)
|
tBase_Scope_with_History
|
_PlotCanvas
_PlotCanvas_History

t_Scope_Plot (Brick with simple Scope)
|
t_C_Scope_Plot (Control with simple Scope)
|
tBase_Scope_with_History
|
_PlotCanvas
_PlotCanvas_History

tScope_Display_Light (TopWindow with simple Scope. e.g. for Signal WorkBench)
|
tBase_Scope_with_History
|
_PlotCanvas
_PlotCanvas_History

tScope_Display_Light

This class is specially developed for the Signal WorkBench (SWB) and consist of a Normal Scope and a History Scope. This class is based on tBase_Scope and has only 1 method, used by the main program to transport data (and parameters). Data can be any combination of tuples, list, 1-dim arrays, 2-dim arrays, which might even be of different length.

Calculate (Display_Data, Display_Params = None)

This class has 3 callback functions,
one to redraw the history signal when another signal is selected by the user :

def _On_History_Signal_Selection (self, Signal) :

and one to redraw the normal signal, when another selection in the history scope is made

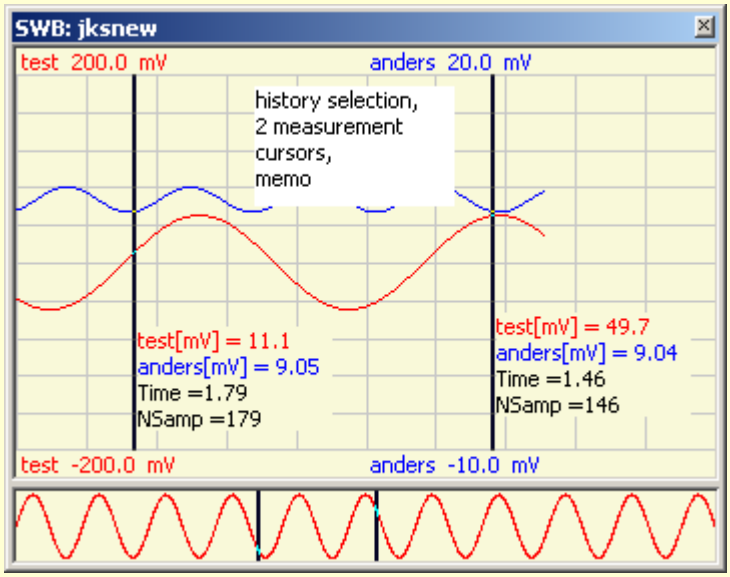
def _On_History_Cursor_Selection (self, x0, x1) :

and one to get autoscale parameters

def _Get_MinMax (self, chan) :

tBase_Scope

```
# ********************************
#  Plots_Dock
#        |___ Panel
#        |        |___ Panel_Top
#        |        |___ Scope_Canvas
#        |        |___ Panel_Bottom
#        |
#        |___ Scope_History
```



- __init__
- On_Cursor_Selection (CallBack from history display)

PlotCanvas

_ __init__
- Add_Data

- Get_DataSet
- Get_NCurve
- Add_Channel
- Set_Canvas
- Set_Channel
- Draw don't know why this is necessary

PlotCanvas2

_ __init__
- Add_Data

- Set_Channel
- Set_Canvas

Brick PyJamas

(march 2009)

Application Designer / Domain Expert / Control Designer / Core Developer

Introduction

Having huge troubles getting V 0.5 working, I decided to stick to 0.4p1 for the moment.

Javascript debugging

Mozilla

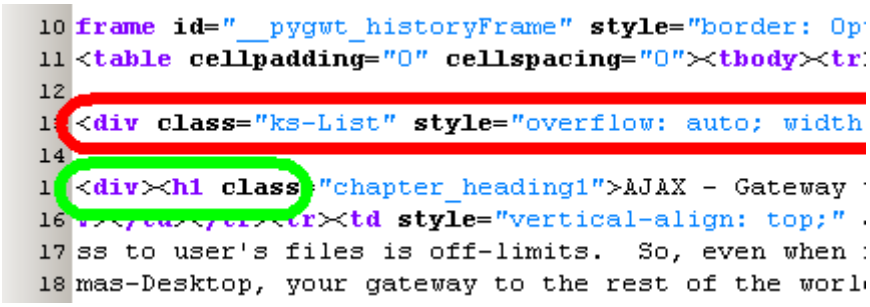
- download and install Venkman-javascript debugger
- Install FireBug
- start

with switch "-venkman"
http://www.mozilla.org/projects/venkman/venkman-walkthrough.html

IE7

- Download / install script debugger Debugging Javascript in IE7 Darren Beale
- Donwload / install IE developers toolbar Debugging Javascript in IE7 Darren Beale

Printing the Book



GUI Controls Overview

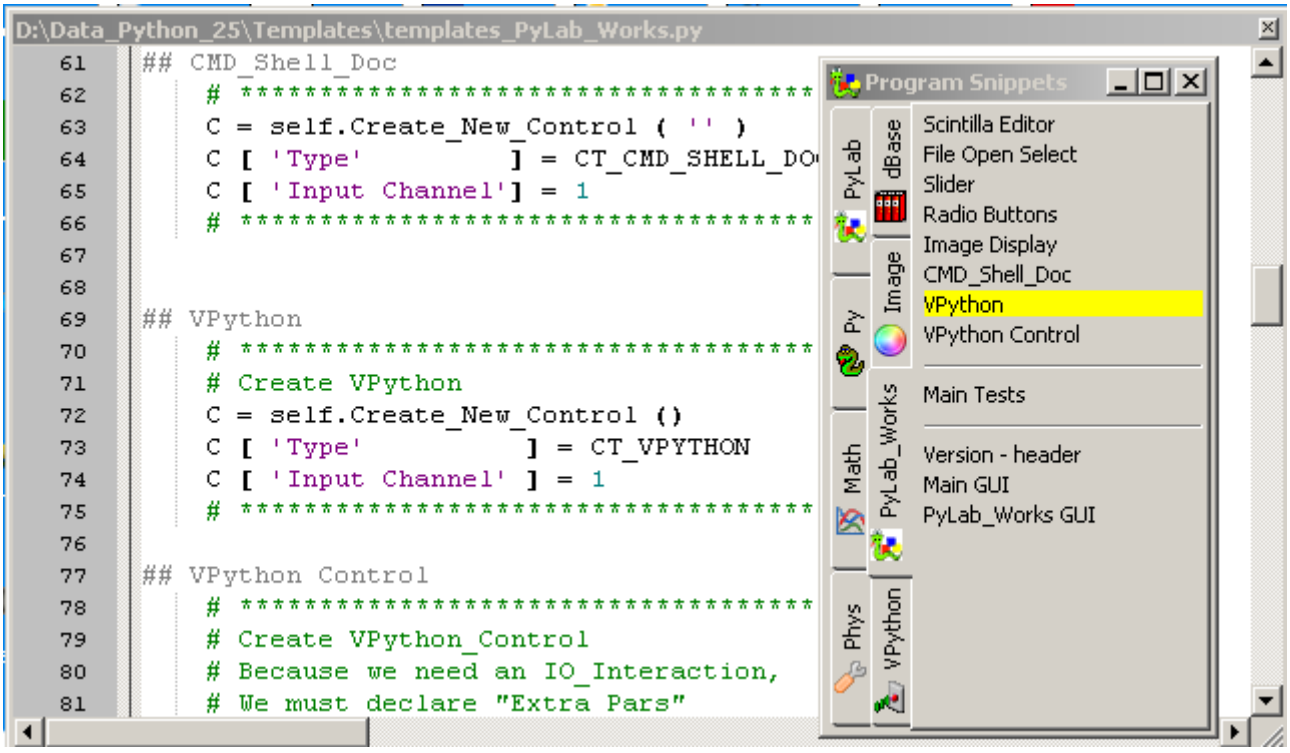
(march 2009)

Application Designer / Domain Expert / **Control Designer / Core Developer**

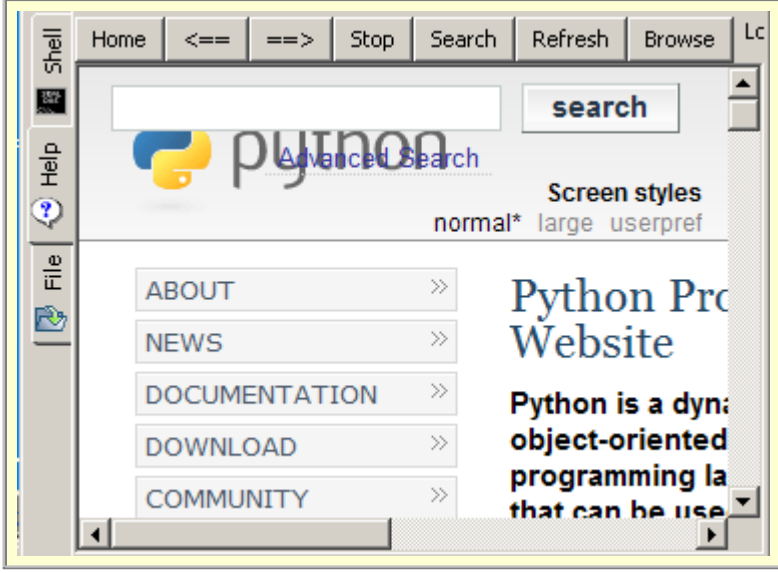
Introduction

Dialogs	Selection / Value	Interactive	Display	Graphics	Misc
File Open Color Picker	Radio Button Spin Button Slider Button dBase Tree	Edit Box Code Editor HTML CMD_Shell_Doc CMD_Shel (obs?) Doc Viewer (obs?) Signal WorkBench Visual_SQL	Static Text Grid STD-Viewer (obs?) LED Num Display	Image PyPlot ScopePlot Scope Hist Plot MatPlot 2D Scene VPython VPython_Controls Float Canvas	ADC

The code for using a control in a Brick can of course be fetched from the template manager,



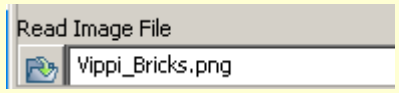
CMD_Shell_Doc



Combines command window, standard / error output, r
and document viewer (including webbrowsing).

```
CD = self.Create_Control ( t_C_Cmd_Shel
CD.Input_Channel = 1
```

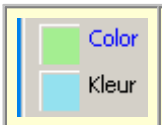
Select File for Open



When a new file is selected, either through the browse button or by an Enter key in the edit box, the filename is returned in Par[1].
Both absolute and relative filenames are accepted.

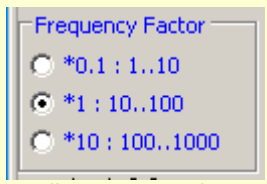
```
# ************************************************
# Create File Select
# Range = one of the constants definied in
#         ../support/dialog_support.py
#       or
#         a general file dialog mask, e.g.
#         'dBase Files(*.db)|*.db|All Files(*.*)|*.*'
CD = self.Create_Control ( t_C_File_Open, 'Read Sound File', De
CD.Range    = '*.wav'
```

Color Picker




```
CD = self.Create_Control ( t_C_Color_Picker, 'Color' )
CD = self.Create_Control ( t_C_Color_Picker, 'Kleur',
                           wx.Color ( 100, 200, 100 ) )
```

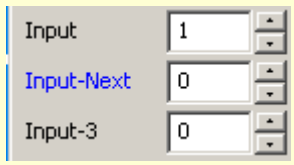
Radio Button



RadioBox occupies full width.

```
CD = self.Create_Control ( t_C_RadioBox, 'Frequency Factor', 1 )
CD.NCol     = 1
CD.Range    = [ '*0.1 : 1..10', '*1 : 10..100', '*10 : 100..1000' ]
```

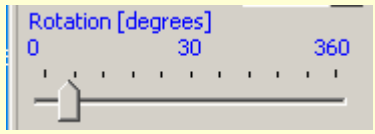
Spin Button



Default range is [-10 .. 10]

```
CD = self.Create_Control ( t_C_Spin_Button, 'Input' )
CD.Range    = [ 1, self.I ]
CD = self.Create_Control ( t_C_Spin_Button, 'Input-Next' )
CD = self.Create_Control ( t_C_Spin_Button, 'Input-3' )
```

Slider



The Slider occupies full width.
The Slider has extra settings:
- Log, set to True for logarithmic
- Format, specify a format string

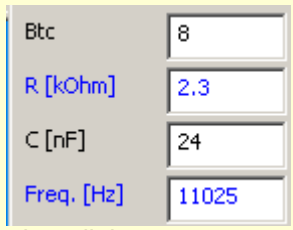
```
CD = self.Create_Control ( t_C_Slider, 'Rotation [degrees]', 30
CD.Range         = [ 0, 360 ]
CD.Input_Channel = 2
```

Button

When a Button is pressed, the ordinal value [0 ..] of the button is returned in the base Par.

```
CD = self.Create_Control ( t_C_Buttons, '', 20 )
CD.Caption = [ 'RePlay', 'Stop' ]
```

Edit Box



The edit boxes are aligned with PG.My_Control_dX2, if the caption texts is smaller than PG.My_Control_dX2. A new value is activated, when the focus is lost or when an Enter key is pressed.

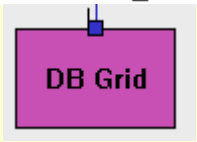
```
CD = self.Create_Control ( t_C_Text_Control, 'Btc'         )
CD = self.Create_Control ( t_C_Text_Control, 'R [kOhm]'    )
CD = self.Create_Control ( t_C_Text_Control, 'C [nF]'      )
CD = self.Create_Control ( t_C_Text_Control, 'Freq. [Hz]' )
```

Grid

Prim Key	Name	Type	NotNull	Default
0	PatNr	TEXT	99	None
0	DateTime	FLOAT	0	None
0	Protocol_Oms	TEXT	0	None
0	Test_Oms	TEXT	0	None
0	Test_Count	INTEGER	0	None

Input is of type GridData, which is a list of rows, where each row contains the column elements for 1 row and where the first row contains the captions of the columns. This grid can also be used to display data or metadata of a database.

```
CD = self.Create_Control ( t_C_Grid )
CD.Input_Channel = 1
```



Code_Editor

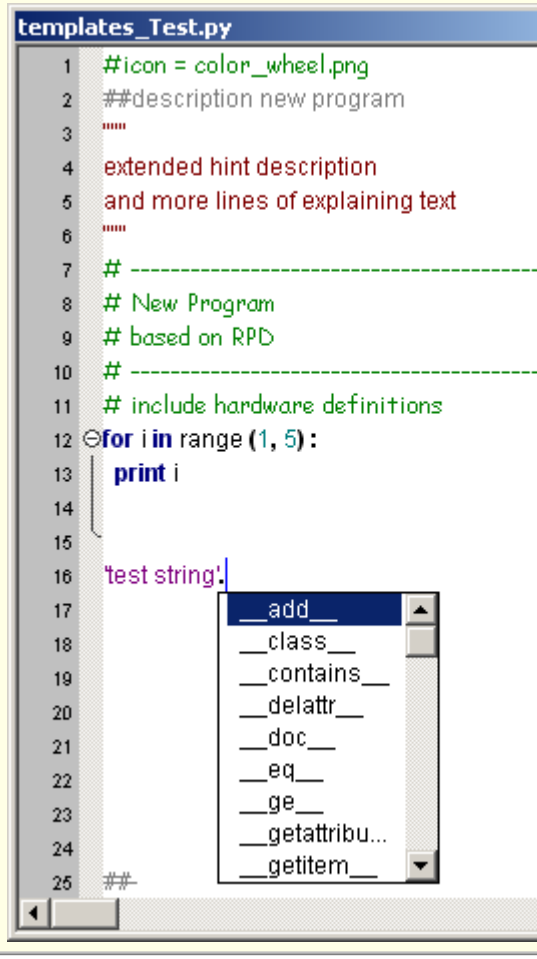
The code editor is based on Scintilla. The editor has code highlight support, autocompletion and many more features. Another nice feature of this code editor is the code-snippet manager, of which the visibility is toggled by F7. And of course editing of the code templates is done in a separate but identical code editor.

For more information see here.

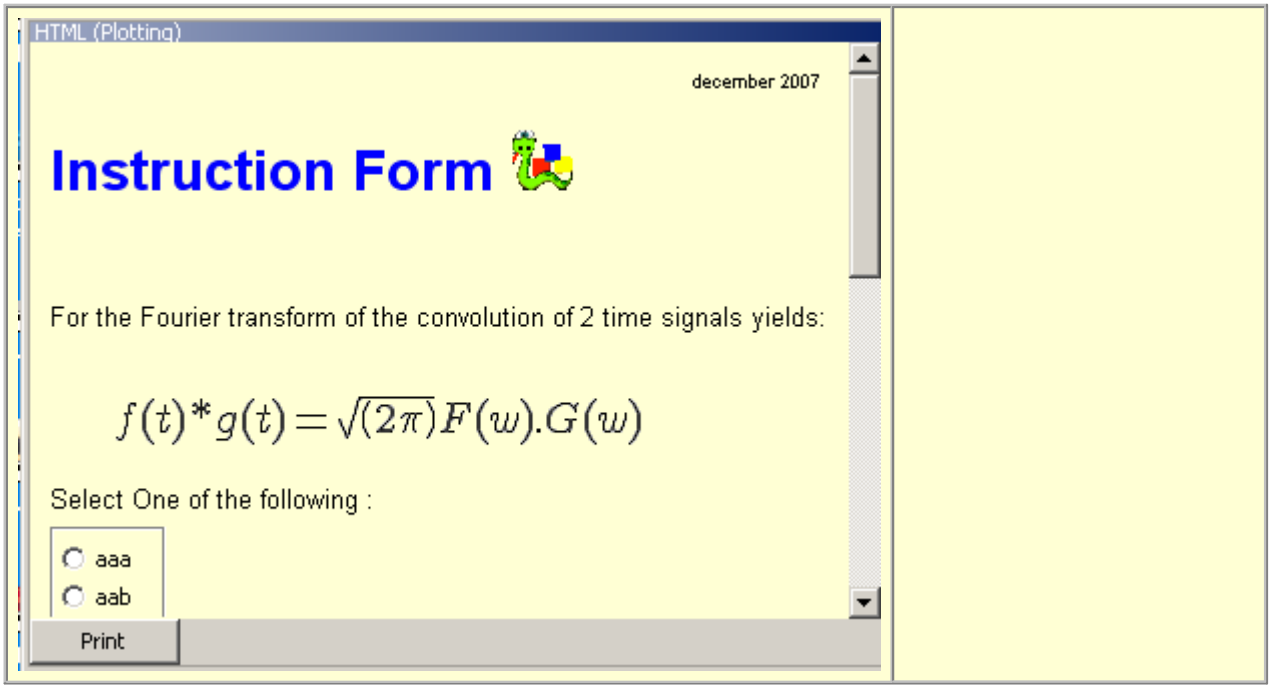
```
# Create the GUI controls
# ***********************************************
# Create the Scintilla editor
# Parameters
#    1 = output, ( <complete code>, <selected code> )
#    2 = IO_Interaction
#          FileName  = output, the loaded code file
#          CMD Shell = input, the local Help Browser
CD = self.Create_Control ( t_C_Code_Editor )
# ***********************************************
```

Because the Editor has no inputs, and we need 2 communication parameters (of which 1 is even an IO_Interaction), we need to declare "Extra_Pars".

The Editor will hook itself to the main-menu of the form.



HTML*



Static Text

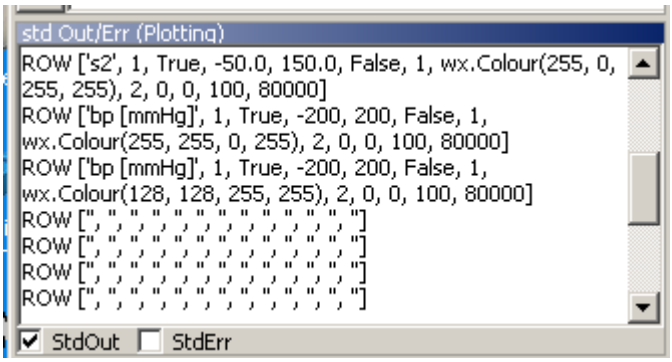
Label 1 Label 2
Label 3

```
CD = self.Create_Control ( t_C_Static_Text, 'Label 1' )
CD.NewLine = False
CD = self.Create_Control ( t_C_Static_Text, 'Label 2' )
CD = self.Create_Control ( t_C_Static_Text, 'Label 3' )
```

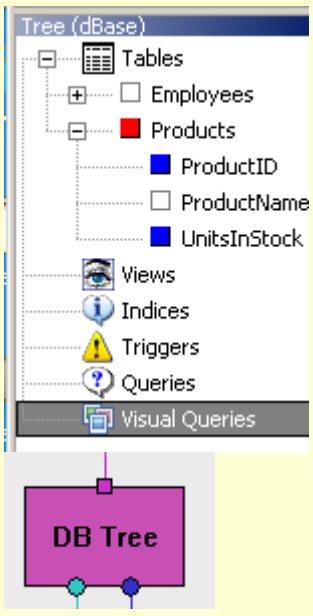
STD-Viewer (obsolete ?)*

STD-Viewer can be used to catch the **stdout** and/or the **stderr** messages. Toggling the viewers redirections can be done dynamically. Select, Select All, Copy are both available through the RM-menu and through the acceleration keys.
The associated brick has no inputs or outputs. There's only one such brick allowed.

Inside:
The control has a special trick, that will redirect the outputs back to the original ports, as soon as the application window is closed.



dBase Tree



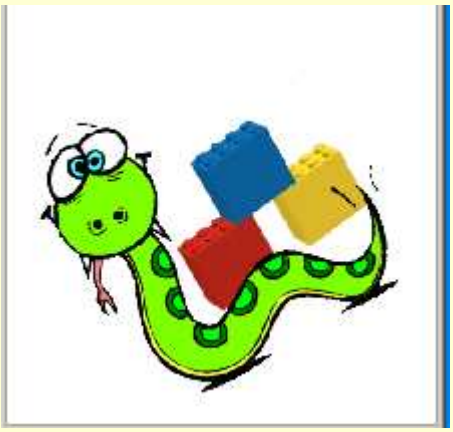
Shows the meta-data from a database as a Tree. Fields in a table / view / etc. can be selected / de-selected or selected for ordering. Every time a change is made, a new SQL statement is generated, which can be sent to the database.

Normally this control will be the only control in the Brick, and the Input of the Brick will be coupled to the control, so Input changes will be redirected to the control. The Brick has to watch for changes of the second and / or third parameter and if found can send the parameter values directly to the outputs.

self.P[0] = Input Connection, dBase meta-data
Output, SQL statement (generated from selection)
self.P[2] = table info (generated from selection)

```
# Create the GUI controls
CD = self.Create_Control ( t_C_DB_Tree )
CD.Input_Channel = 1
```

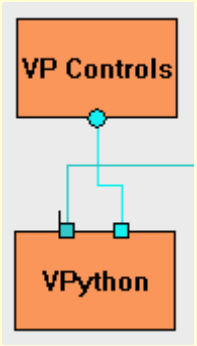
Image_Show



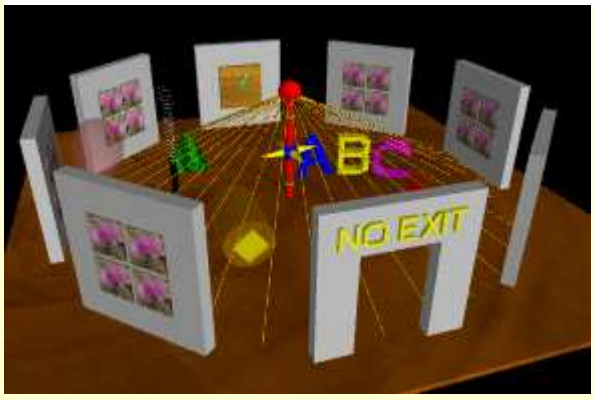
```
CD = self.Create_Control ( t_C_Image_Show )
CD.Input_Channel = 1
```

VPython

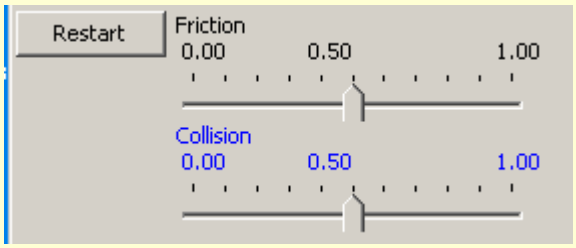
This control is a complete embedding of VPython. Together with VPython_Controls and a editor-control, you get a powerful IDE for developing VPython 3D animations.




```
# Create VPython
CD = self.Create_Control ( t_C_VPython )
CD.Input_Channel = 1
```



VPython_Control



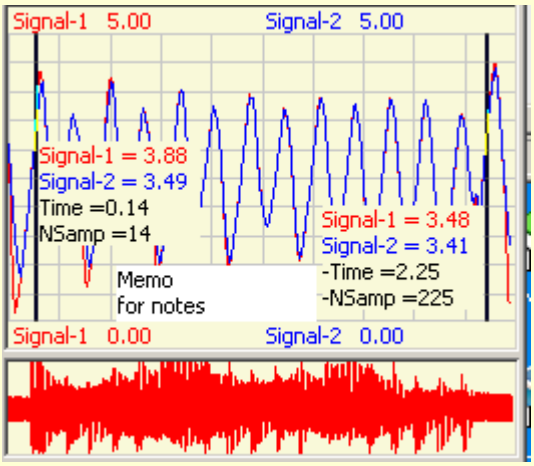

```
# Create VPython_Control
CD = self.Create_Control ( t_C_VPython_Control )
```

This a dynamic control, directly generated by the code from the VPython control.
See VPython control.

Scope_Plot

Very power full display for graphs. It can handle multiple signals, which might be one of the iterable types, like, list, tuple, array, s_list, ...The signals may differ in length. The lower part of the display shows the total length of 1 signal in a auto scaling min-max mode (strip chart recorder) and this part is used to make a selection in time of all the signals. The upper graph shows the selected part of all signals, with easy to set scales (or auto scales), has 2 measurement cursors and free positioning memo field.

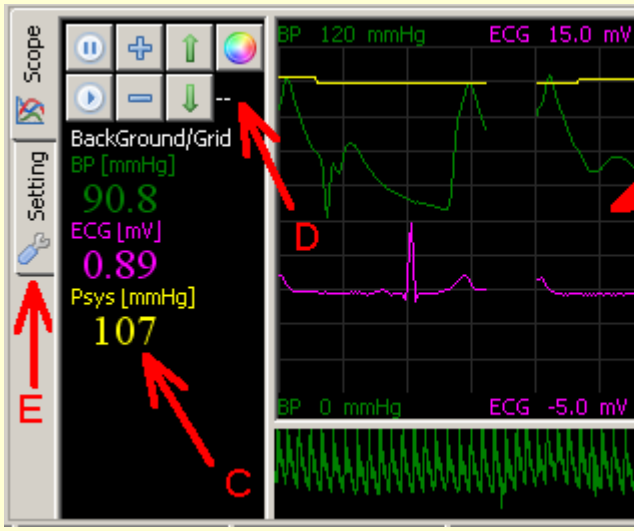

```
CD = self.Create_Control ( t_C_Scope_Plot )
```



Scope_Display

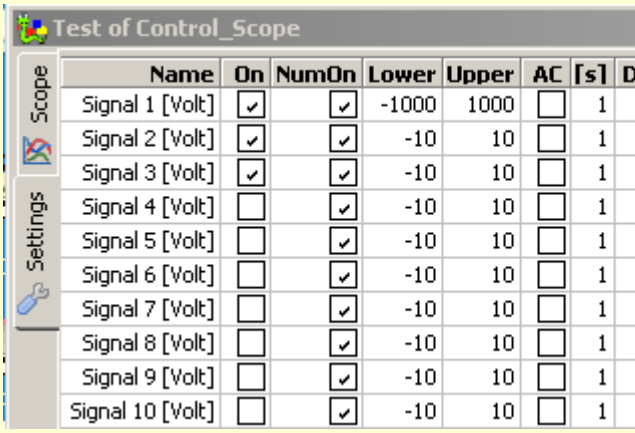
This is an **interactive** display, special designed for **realtime** display and control. Some or all of the recorded signals can be shown in the main signal window. Settings of the signals can be done through the labels on top and bottom, with the **speed-buttons** on the left and through an extensive grid on the second page. On the bottom you see an **history window** where one of the signals can be displayed in min/max mode for the **total recording** (auto compression). This window can also be used to **look back in time**, while recording continuous. Although not strictly necessary, this control will often be the only control in a brick.


```
CD = self.Create_Control ( t_C_Scope_Display )
```



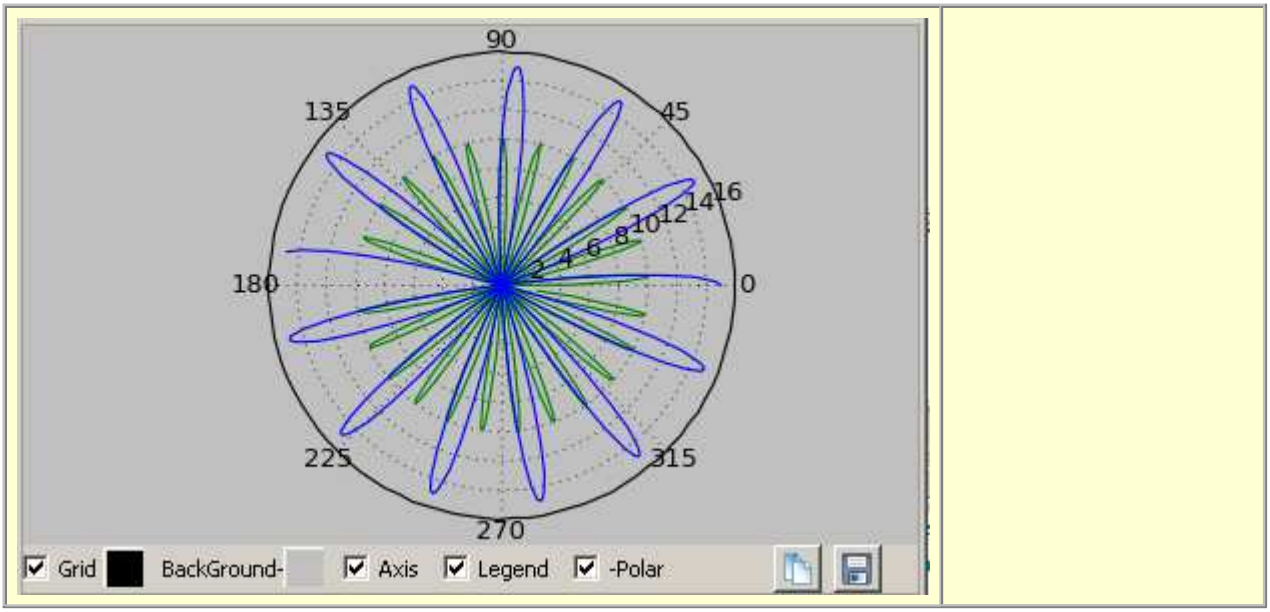
The picture shows a realtime ECG and bloodpressure signal with **online analysis** of the bloodpressure (yellow line is the calculated Systolic Pressure). Each signal can be **delayed to compensate** for realtime filter delays. During freeze, 2 measurement cursors are available.

Accepts the following **MetaData**: SignalName, Units, Calibrate (only the gain/offset tuple for this moment), DisplayRange.

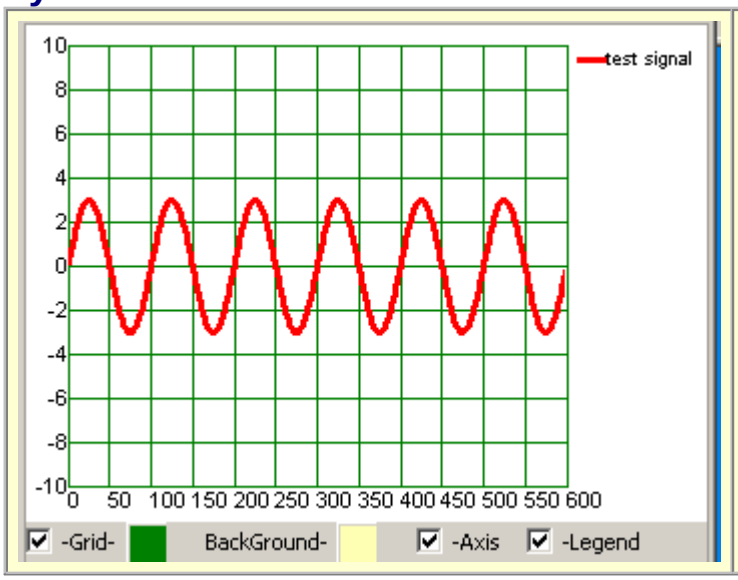


for **more information** see here

MatPlot*



PyPlot



This is a graphical window for displaying (real-time) X-t signals, it should be faster than MatPlotLib, but slower and less flexible that ScopePlot.

```
CD = self.Create_Control ( t_C_PyPlot )
CD.Range         = self.I
CD.Input_Channel = 1
```

Background color doesn't work.

LED

rood1 rood2
rood3

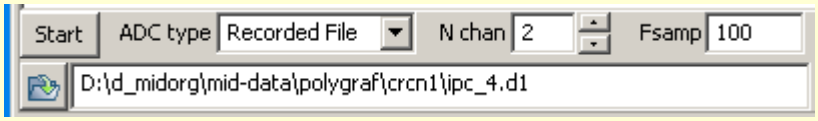
Experimental Control, it should be possible to turn the LED on / off and give it different colors.

```
CD = self.Create_Control ( t_C_LED, 'rood1' )
CD.NewLine = False
CD = self.Create_Control ( t_C_LED, 'rood2' )
CD = self.Create_Control ( t_C_LED, 'rood3' )
```

ADC

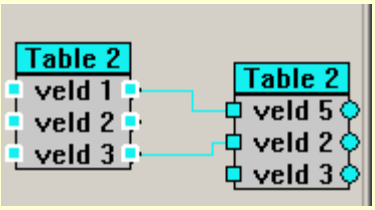
Because this control uses a windows executable for controlling the AD-converter, this control only works on MS-Windows at the moment. The control exports 32 bit integer values and (sometimes) gives the following metadata: Calibrate, SignalName, Units, Frequency.
This control wants to be called in every loop, so the Continuous flag must be set, and the Run_Loop method must be implemented.

```
def Run_Loop ( self ) :
  C = self.GUI_Controls [0]
  N = C.Get_New_Samples ()
  if N > 0 :
```




```
# This device wants to be executed every loop
self.Continuous = True

# Create the GUI controls
CD = self.Create_Control ( t_C_AD_Converter, '', 0 )
```

Visual SQL



This is a beginning of a Visual SQL builder, based on the same canvas as PyLab_Works itself. No real functionality yet.

GUI Controls Intro

(march 2009)

Application Designer / Domain Expert / **Control Designer** / Core Developer

Introduction

Controls are the basic GUI building blocks used in Bricks. Controls transports information from the user to the Brick and vice-versa. In the beginning we shall need to extend the controls somewhat, but after a short time we'll have a complete set of controls and only cosmetic changes will be needed.

In the this part we'll describe how controls can be used to create new Bricks.
In the part "GUI Controls Overview", an overview will be given of all available controls, with enough detail to use them to build new Bricks.
The part "GUI Controls Building", will describe how to build new controls.

Help on Controls

The name of a control class always starts with "t_C_", so by typing this first characters in one of the PyLab_Works editors will popup an auto-completion list with all the available controls. In a PyLab_Works command-shell you can ask help about "t_C_". The chapter GUI Controls Overview, describes all the available controls. And last but not least the template manager (F7 form an editor) has a tab where all controls are available as an example, with all the specific parameters used.

Creating a Control Instance

A control is created by calling the Brick's "Create_Control".

```
763   def Create_Control ( self, Type = None, Caption = '', Default = None ) :
```

Often you only have to specify the Type, because the default values for Caption and Default will suffice:

```
139     CD = self.Create_Control ( t_C_Static_Text )
```

And as an alternative, which might be better readable, you might even specify the type as an attribute:

```
141     CD = self.Create_Control ()
142     CD.Type = t_C_Static_Text
```

That's all !!! Everything else is done behind the scene: layout of controls, saving and restoring settings etc.

More Control Parameters

For some controls you might need to specify some additional parameters, because of your specific needs. e.g. you need a slider with a logarithmic changing value:

```
860       CD = self.Create_Control ( t_C_Slider, 'Frequency', 100 )
861       CD.Range = [1,10000]
862       CD.Log   = True
```

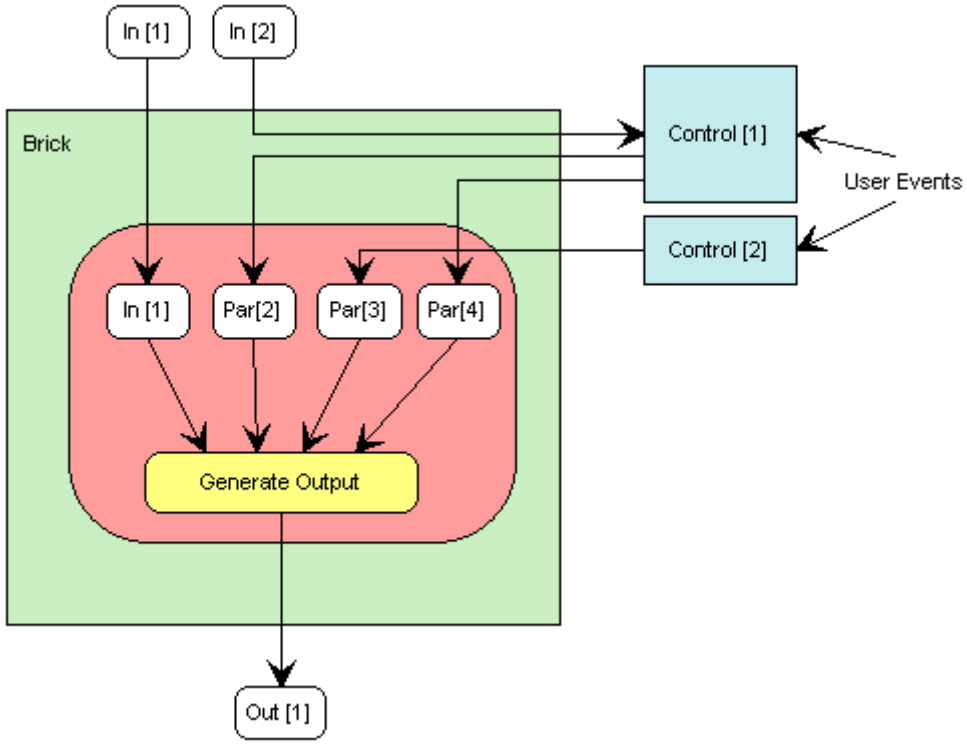
The parameters that are a available for each Control are given in the help about each Control.

If you want a Control to be positioned behind the previous Control on the same line, you can set the following attribute to False (this will not work for all controls)

```
139     CD.NewLine = False
```

Data transport between Brick and Controls

Looking at the schematic below, shows that the communication between a GUI-control at it's parent Brick is always done through the Par-array of the Brick. If a control acts on the same Par as an input (e.g. Par [2] and Control [1]), the input is automatically passed to the control and the control decides what value will be passed to the Brick. A Control can also have extra parameters (e.g.Par [3] - Control [2] or Par[4] - Control [2]), to exchange information between the Brick and the Control. Notice that the numbering starts at "1", which is done to get a human representation in the Brick Definitions. Don't bother with the complex numbering for the controls, because they'll see a zero-based Par-array, only containing the elements that concerns them.



So in your Brick you can simple check any use the Par-array in the following way:

```
879     if XPar [0] :       # check if any Par changed
880       if XPar [1] :     # check if Par [1] changed
881         A = Par [1]     # use Par [1]
882       if XPar [2] :
883         B = Par [2]
```

GUI Controls Building  (march 2009)

Application Designer / Domain Expert / **Control Designer / Core Developer**

Introduction

Controls are the basic GUI building blocks used in Bricks, they perform the interaction between the Brick and the user. Control classes are detected automatically and therefor they should always reside in a file with a name starting with "control_" which should be located in the directory "PyLab_Works/controls/", and the name of a control class should always start with "t_C_". Besides that, control classes should always be a subclass of "PG.My_Control_Class".

Simplest Control

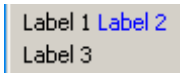
Let's first look at the most simple control currently available, e.g. the Static_Text, just a static text label. The control is derived from wx.Static_Text and PG.My_Control_Class, in this order, so that the properties and the methods of wx.Static_Text will yield (as far as they exists), instead of the (sometimes dummy) properties and methods of PG.My_Control_Class. The first thing to do in the __init__ method, is always to generate the PG.My_Control_Class instance, because this will generate all necessary attributes (like self.Dock, self.X) from the input parameters (see below for all the available properties and methods)

```
425 # ************************************************************
426 # ************************************************************
427 class t_C_Static_Text ( wx.StaticText, PG.My_Control_Class ):
428
429   def __init__ ( self, *args, **kwargs ):
430     PG.My_Control_Class.__init__ ( self, *args, **kwargs )
431     wx.StaticText.__init__ ( self, self.Dock, -1,
432                              self.CD.Caption,
433                              pos = ( self.X, self.Y ) )
434 # ************************************************************
```

Using this Control in a Brick, like this:

```
1047     CD = self.Create_Control ( t_C_Static_Text, 'Label 1' )
1048     CD.NewLine = False
1049     CD = self.Create_Control ( t_C_Static_Text, 'Label 2' )
1050     CD = self.Create_Control ( t_C_Static_Text, 'Label 3' )
```

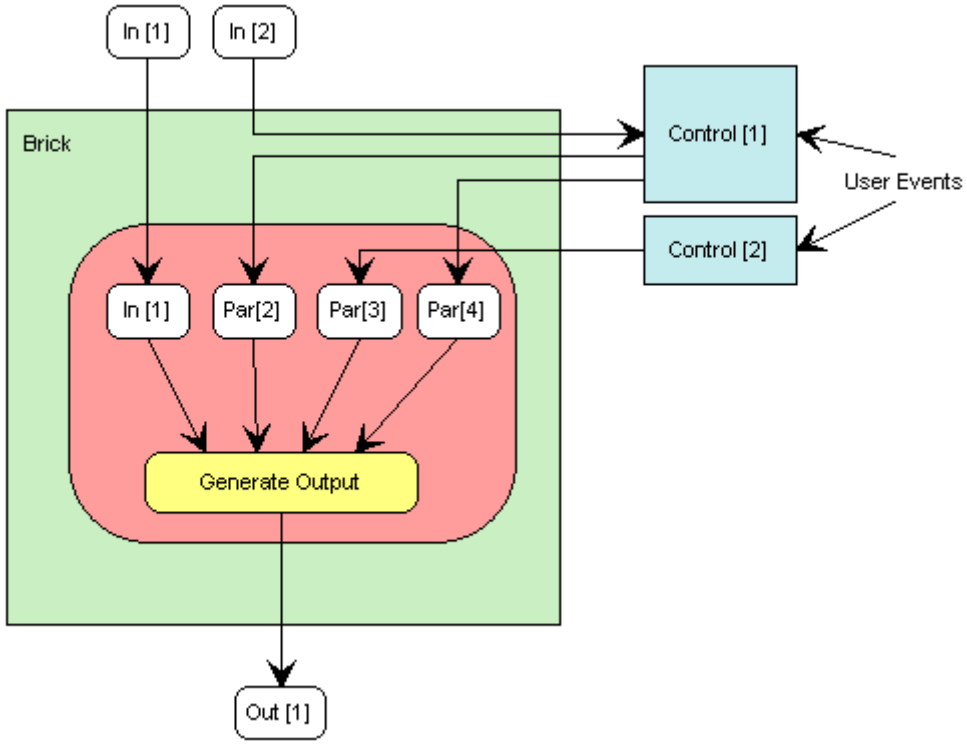
will result in a GUI-interface, like this:



As you can see, positioning, scaling, alternate coloring, etc. are all done by the wrapper methods. Also saving and reloading of the settings is done by the wrappers.

Data transport between Brick and Controls

Looking at the schematic below, shows that the communication between a GUI-control at it's parent Brick is always done through the Par-array of the Brick. If a control acts on the same Par as an input (e.g. Par [2] and Control [1]), the input is automatically passed to the control and the control decides what value will be passed to the Brick. A Control can also have extra parameters (e.g.Par [3] - Control [2] or Par[4] - Control [2]), to exchange information between the Brick and the Control. Notice that the numbering starts at "1", which is done to get a human representation in the Brick Definitions. Don't bother with the complex numbering for the controls, because they'll see a zero-based Par-array, only containing the elements that concerns them.



Each control has at least 1 Par associated, but each control can ask for more Par's (so called Extra-Pars, through _EP_Add) to communicate with the Brick. The control can decide for each control if it's a relative simple parameter or a (special) dictionary. Within the control the Par's are available through the P-array, which is simply zero-based, and has exactly the amount of elements as the number of Par's the control has asked for.

Properties & Methods

In the lists below, for clarity "self" is replaced by the object's base name. The first item in the comment, between brackets is the default value.

Brick

```
Brick.Control_Defs [ CD0, .. ]   # Array of pointers to the definition of each Control
Brick.Control_Pane               # AUI pane, containing the GUI-Controls
Brick.GUI_Controls [ C0, .. ]    # Array of pointers to the GUI-Controls
Brick.Diagnostic_Mode
```

CD[0] = super_object ()

```
CD[0].Brick           # For easy accessing the Brick
CD[0].GUI_Control     # For easy accessing the associated Control

CD[0].Type            # Control-Type
CD[0].Caption         # ('') Label in the upper left corner
                      # In some cases e.g. Buttons, this is a list with the Buttons captions
CD[0].Initial_Value   # Default / Initial Value
CD[0].Input_Channel   # (None) The index of the Brick's input [1..], to which the control is
connected
CD[0].NewLine         # (True) The next control (if any) will be placed on a new line
```

just for some specific controls, in comment some controls that use these. See the specific control for a more extensive explanation.

```
CD[0].Range           # Radio_Buttons, Slider
CD[0].Value           # ???
CD[0].NCol            # Radio_Buttons
CD[0].Log             # Slider: Linear / Logarithmic
CD[0].Format          # Slider
```

GUI_Control

```
    C[0].Brick          #
    C[0].CD             # The Control_Description of this control
    C[0].Dock           # The control Pane on which the control is placed

    C[0].TopFrame       # The application Frame
    C[0].Caption        # (None) Caption of the AUI Frame

    C[0].Test           # (False) If True this Control will run in Test-Mode
    C[0].Ini            #
    C[0].X              # Left-Top of this Control
    C[0].Y              # Left-Top of this Control
    C[0].IniSection     #
    C[0].EP             # [ None ]  Don't use this !!
    C[0].EP_IsDict      # [ False ]  Don't use this !!

    C[0].P              # This is thé array to get into contact with the Bricks Par-array


  methods
    C[0].Load_Settings       #
    C[0].Save_Settings       #
    C[0].Display_Label       # ???
    C[0].Kill                #
    C[0].SetValue            #
    C[0].GetValue            #
    C[0].SetPosition         #
    C[0].SetSize             #
    C[0].GetSize             #
    C[0].SetForegroundColour #
    C[0].SetToolTipString    #

  event handlers
    C[0]._On_Event     # Bounded to all kinds of events
    C[0]._On_Size      #
```

Control Pane

```
    Pane.Brick          #
    Pane.CD             # points to the Bricks Control_Defs
    Pane.GUI_Controls   #
    Pane.x              # during building, the current left top for the next control
    Pane.y              # during building, the current left top for the next control
```

Control Methods

Positioning and Sizing

SetPos, SetSize, GetSize

Override the GetSize method:

```
339   # ****************************************
340   # ****************************************
341   def GetSize ( self ) :
342     """ Return size of the fixed multi-widget component """
343     return self.dX, self.dY
```

Control.SetValue, Control.GetValue

If the base control has a different naming for these functionalities (like ColourPicker) or if special

processing is needed (like validation) these methods should be implemented.

```
75  # ********************************************************
76  # ********************************************************
77  def GetValue ( self ) :
78    filename = self.Text.GetLabelText ()
79    try :
80      return Get_Relative_Path ( filename, Application.Dir )
81    except :
82      return None
```

In case only the name differs, it's often more convenient to just redirect the methods:

```
452     # SetValue, GetValue have different names
453     self.SetValue = self.CP.SetColour
454     self.GetValue = self.CP.GetColour
```

Control.SetForegroundColour

Subsequent controls are alternate colored, so it become's more clear which elements belong to which control (like a label telling the meaning of a slider). If the control has more than 1 widget, you've to implement this method and step through all the controls to give them the right color.

```
375  # ************************************************
376  def SetForegroundColour ( self, color ) :
377    self.Caption.SetForegroundColour (       color )
378    wx.TextCtrl.SetForegroundColour  ( self, color )
379# ************************************************
```

Control.Save_Settings, Control.Load_Settings

If the control settings are completely definied by the SetValue / GetValue values, then there's no need to implement this method. But if the control wants to save / restore some extra settings, than it has to implement this method.
Does it has to store GetValue ????

By defining self.Type (default None),

```
357     self.Type = str
```

SetValue (and maybe other methods in the future) assures the correct type.

```
136       if self.Type :
137         Value = self.Type ( Value )
```

Control.Kill

This method is called when the control is being destroyed. In general there's no need to implement this method. But some controls (like Scene_VPython) need special processing before they are killed, in that case this method might be implemented.

```
491  def Kill ( self ) :
492    self.Timer.Stop ()
493    visual.scene.visible = False
```

Control Test Methods

There's a control manager available, in which you can easily build and test individual controls. For complex controls it's difficult to create the necessary input signals, therefore each control has 3 dummy test methods

(Test1, Test2, Test3), which can be overriden by the control implementation. Here an example that generates a few real-time signal to test the scope plot. The doc test is important because it's assigned to the Test-Keys as a tooltip.

```
612   # ************************************************
613   def Test2 ( self ) :
614     """
615 Generates 3 dynamic signals: sine, square, sawtooth,
616 which will be shown in Real-Time mode.
617     """
618     self.Real_Time = True
619     self.Scope_Normal.Real_Time  = self.Real_Time
620     self.Scope_History.Real_Time = self.Real_Time
621
622     SD0, SD1, SD2 = self._Create_Test_Signals ()
623     self.Add_Data ( SD0, SD1, SD2 )
```

Control Events

In PG.My_Control_Class a lot of events are already bound (see actual code in PG for a complete list). If an event-type is missing, it's best to add it in PG.My_Control_Class, so it will be available for future use.

```
324     self.Dock.Bind ( wx.EVT_SIZE                , self._On_Size  )
325
326     self.Dock.Bind ( wx.EVT_SLIDER               , self._On_Event )
327     self.Dock.Bind ( wx.EVT_RADIOBOX             , self._On_Event )
328     self.Dock.Bind ( wx.EVT_SPINCTRL             , self._On_Event )
329     self.Dock.Bind ( wx.EVT_TEXT_ENTER           , self._On_Event )
330     self.Dock.Bind ( wx.EVT_COLOURPICKER_CHANGED, self._On_Event )
```

Main Menu Bindings

Claiming a statusbar field

It's possible to claim one or more fields in the statusbar of the application. In the creation of a control we can ask for the next free field of the statusbar, like this:

```
# we want a StatusBar Field
self.My_StatusBar_Field = len ( Frame.StatusBar_Controls )
Frame.StatusBar_Controls.append ( self )
```

And if want to change "our" statusbar field:

```
self.Frame.StatusBar.SetStatusText( 'CE modified', self.My_StatusBar_Field )
```

Pane

Brick_2_Control

If an input of a Brick changes, the Brick will call this method to transport the input value to the control.

```
217   # ******************************************************
218   # If the parameters of a Brick have changed due to input signals,
219   # this procedure must be called to correct the visual settings
220   # of the control
221   # ******************************************************
222   def Brick_2_Control ( self, inputs ) :
223     for Control in self.GUI_Controls :
224       CD = Control.CD
225       try :
226         if Control.Brick.Diagnostic_Mode :
227           line='Brick_2_Control: ' + str(inputs[CD.Input_Channel])
228           PG.wbd ( Control.Brick, line )
229
230         if  CD.Input_Channel and \
231            ( inputs [ CD.Input_Channel ] != Null ):
232           Control.SetValue ( inputs [ CD.Input_Channel ] )
233
234       except :
235         pass
```

Controls IDE  (april 2009)

Application Designer / Domain Expert / **Control Designer** / Core Developer

Introduction

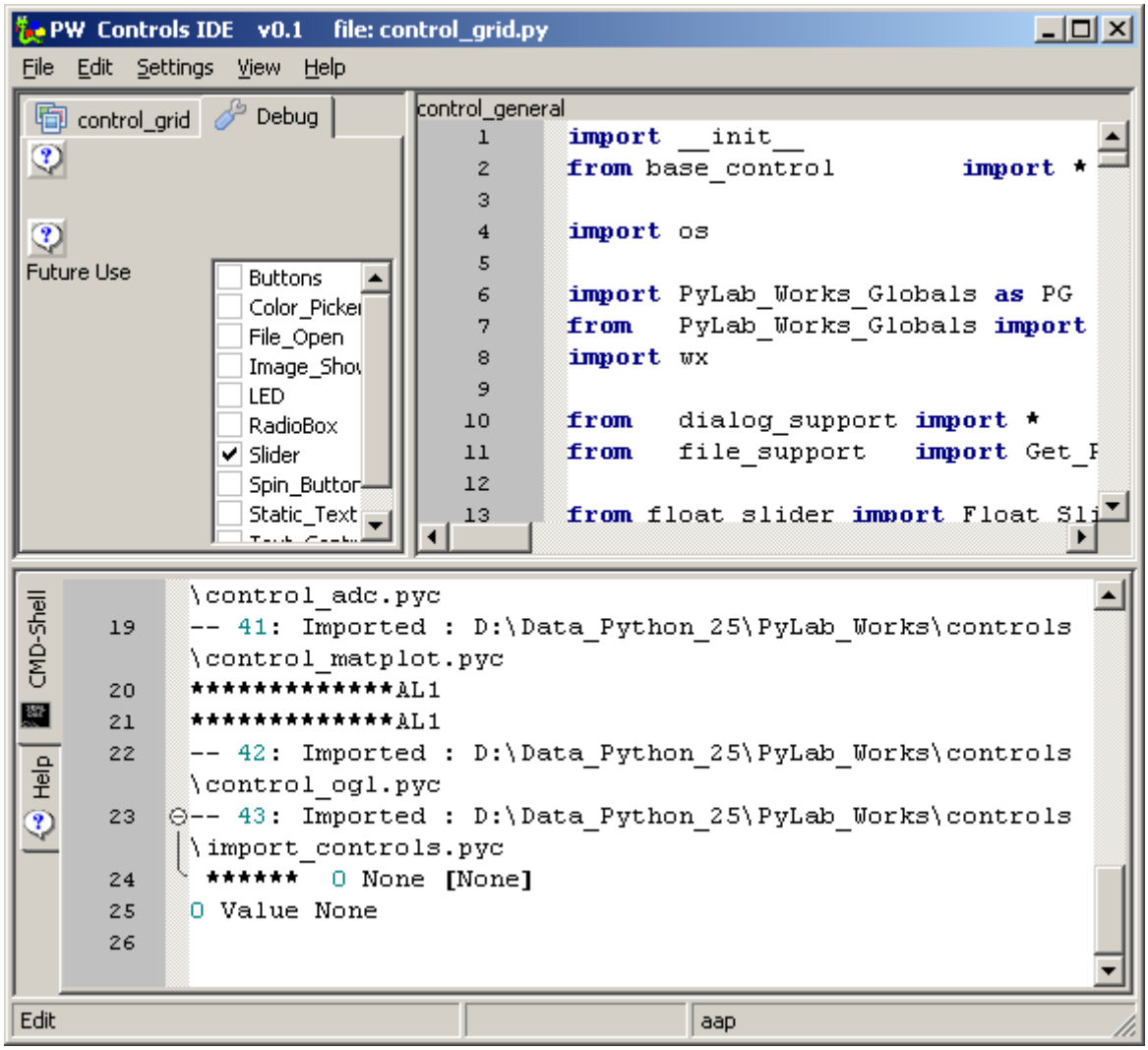
Control = GUI, consisting of 1 or more graphical components, that displays data in some way and / or collects data from the user.
This program is very premature, but nevertheless quite useful.
Instead of making 1 IDE for all tasks, we made a number of specific IDE's, which can be held quit simple, while they are optimized for their specific task. After we've made the different IDE's, we'll make a new decision if we'll combine the different IDE's. For instance, if we want to create / modify a control the following tasks are required:

- edit the source code
- compare the source code with the source code of other controls (examples)
- getting specific help information about controls
- debugging the control
- previewing the control

On the left you see a premature debug tab, and a tab to display another control code file (in read-only mode). Changing either the code file for the model or the code file under construction, right click on the tab and select from the popup-menu.
Important keys are

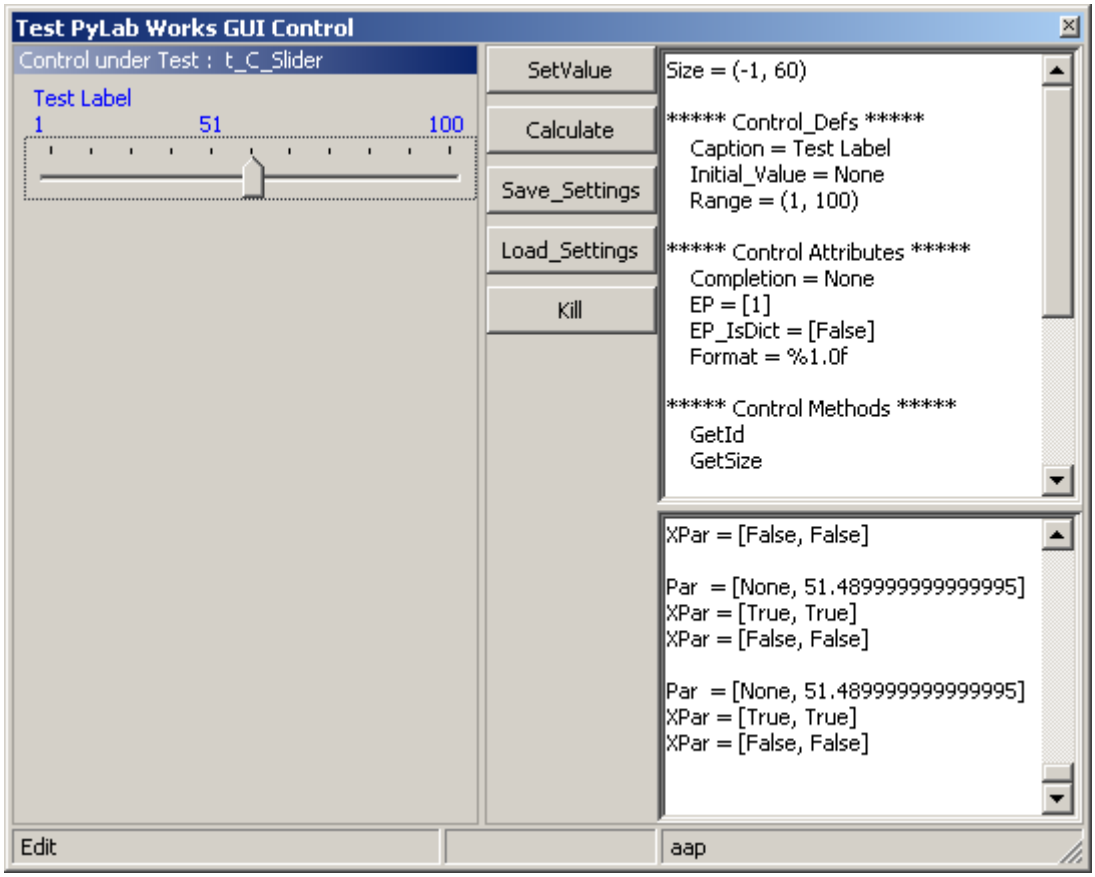
- F7 = Template Manager
- F9 = Preview the selected control

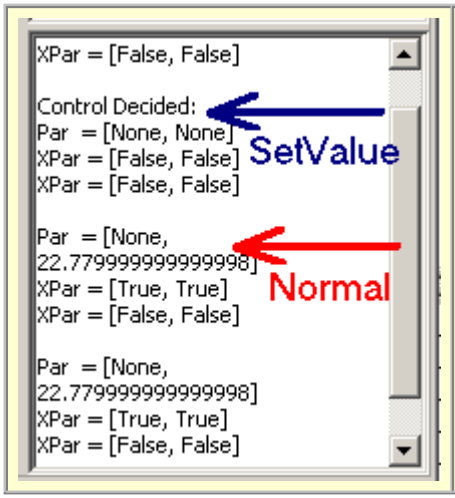
At the moment only the first selected control is used in the preview.
At the bottom we'll find the standaard Shell-Command window with doc-viewer.
The module that contains the control doesn't have to have a main-section (but may have it) and doesn't need to have a test-form (but again may have it). The namespace is cleared before every new run. The namespace is available during and after a control has run in the tester.



Control Preview

The preview is started, by pressing F9 from the code editor. The control is drawn in the left panel and is always drawn in the alternate color (blue), so you can see right away if the function SetForegroundColor works correctly. On the right panel, at the top, the control definitions, attributes with the initial values, and the control methods are listed. On the bottom right the important current values are shown.





Each time the control is changed through the normal GUI, the actual parameters are listed, as shown on the left at the red arrow. If the values are changed by the SetValue key, the parameters are listed, preceded by "Control Decided:", as shown at the blue arrow on the left. It's up to the control to decide what he does with SetValue: only change the GUI or change GUI and the corresponding Par value.

Technical Info
Modules:
- PyLab_Works_Control_Manager.py
- controls_wrapper_dont_touch.py

Debugging & Testing

(march 2009)

Application Designer / **Domain Expert** / **Control Designer / Core Developer**

Introduction

PyLab_Works and it's associated programs have many logging, debugging and testing tools, some are

always present, some must be switched on explicitly by command line flags and a few other mechanisms.

- **exprint**, print procedure with traceback
- **Debug(file) flag**, if set, will display all kinds of intermediate results
- **Main exception capture**, will capture final exceptions not caught internally
- **State Machine debugging**, a special module which gives only detailed information about the running Bricks and their communication
- **Test Runs**, most modules can run as a main file. In general the main file has several test cases, which can be selected by the command line flags. There's also a possibility to run a whole bunch of files. And all tests can be started from the launch center.
- **Control Manager**, to test and debug individual controls
- **Real time debuggers**, most useful as a brick itself. Both pdb and rpdb are supported.
- **Dedicated Brick loggers**, ...

ToDo

- add main exception capture to all main modules
- extend main exception capture to display more deeply nested items
- take over the stdout / stderr in main exception capture
- extend main exception capture with a lot of system information
-

exprint = Print + TraceBack

You can use the exprint procedure just as the print procedure (remember the print statement is obsolete from Python version 2.6), but the exprint procedure will add full traceback information (first caller at the top).

```
2  tprint ( 'Hist PlotAppend_Data' )
3
4
5  ____________ Print Traceback ____________
6  ('Hist PlotAppend_Data',)
7     Called from :   186, scope_plot_hist.py
8     Called from :   316, control_scope_base.py
9     Called from :   133, scope_plot_hist.py
10    Called from :  1178, _windows.py
11    Called from : 14470, _core.py
12    Called from :  1341, _misc.py
13    Called from :  8004, _core.py
14    Called from :  1944, PyLab_Works.py
```

Debug(file) Flag

Through the function pd (info_line) in the module General_Globals, debug information will be shown in the stdout and / or a log file with the name _pd_FileName. There's a special function pd_Module (__file__) which should be placed at the end of every module, to log the order of the imports of PyLab_Works own files.

```
1094 # ********************
1095 # ********************
1096 pd_Module ( __file__ )
```

Main Exception Capture

All main programs have an exception capture on the outer most statement. This exception capture should

never occur, but if it does, it gives detailed information about the exception. from all error lines, specified in the traceback, the exception capture tries to evaluate the values, and in case of list, tuples, dict, etc it tries to calculate the length / keys, before calculating the final value. This is very convenient way to see what goes wrong e.g. in case of an index out of range error.

```
Traceback (innermost last):
*****  File "D:\Data_Python_25\PyLab_Works\PyLab_Works_Library_Manager.py", line 1082,
 in <module>
    Main_Form = my_Test_Form ( None, ini )
      local:  my_Test_Form = <class '__main__.my_Test_Form'>
      local:  keys(ini) = ['empty', 'Library Manager', 'Test Form']
        ini = {'empty': {'empty': 0}, 'Library Manager': {'Pos': (5, 148), 'Size': (615,
 796), 'CS_': [506, u'NL', False, 234, 5, 1], 'CS_Bricks': [49, 98, 48, 80, 62, 67, 98,
....
*****  File "D:\Data_Python_25\PyLab_Works\PyLab_Works_Library_Manager.py", line 396, in
 __init__
    Tree.Add_PyFile_Info ( underscore )
      self:   Tree: <tree_support.Custom_TreeCtrl_Base; proxy of <Swig Object of type
'wxPyScrolledWindow *' at 0x3574c18> >
      local:  underscore = _
*****  File "D:\Data_Python_25\support\tree_support.py", line 634, in Add_PyFile_Info
    CFL = Get_Class_Methods ( filename, F )
      local:  filename = visual_support
      local:  F = varrow
*****  File "D:\Data_Python_25\support\doc_support.py", line 215, in Get_Class_Methods
    if ( my_module in getfile ( C[1] ) ) and \
*****  File "P:\Python\Lib\inspect.py", line 363, in getfile
    raise TypeError('arg is not a module, class, method, '
TypeError: arg is not a module, class, method, function, traceback, frame, or code
 object
```


The main exception capture is inserted with:

```
1092  try :
1093    run_the_main_program
1094  except :
1095    from PyLab_Works_Globals  import format_exception
1096    print format_exception ( globals )
```

december 2007

Bugs / Future Development

Principle Choices still to make

- use of super-list / super-array / traits for all IO, with or without intelligent connection validator
- do we need separate fast / real-time signals
- Add other 3D engines Soya3D / PySoy
- absolute / relative (to what) paths
- faster / smarter main loop ?

Extensions

- -- Library manager database / updating

- -- Native hardware control
- -- control system analysis
- -- collection implementation
- -- menubar / toolbar as a standard Brick
- -- merging / splitting of Bricks
-

ToDo

- IconIndex from library to project, will become a mess if libraries changes, because these indices are relative !!
- Fixed sizes of panes in wx.aui, doesn't work, bug / limitation of AUI ? (if so recreate the complete window without AUI)
- Library database / updating / etc.
- Code Editor: RED statusbar if error, Error catching, (print) output catching, extended autocomplete
- MatPlot: measurement in pseudo color, right axis in pseudo color, color/linewidth as inputs add
- Error messages in application mode
- refactoring of scope units

Serious Bugs

- when a new library is added in the tree, the ID numbers of all following libraries are incremented, so the icons of library elements in a project might be wrong
- If an connection is removed, the Input should de set to None
- fundamental bug: if inputs to controls like PyPlot arrive along different paths, from a Generator source, the signal pieces are sometimes repeated
- SOLVED ??? After Delete an object, connections are mangled
- bug in MatPlotLib, rc-params, due to new version
- when run in standalone mode, tree selection is lost
- When a new item is added to the project tree, (and on some other occasions), sometimes this error occures

```
Traceback (most recent call last):
  File "D:\Data_Python_25\Lib_Extensions\customtreectrl_SM.py", line 5229, in OnMouse
    thisItem, flags = self._anchor.HitTest(pt, self, flags, 0)
  File "D:\Data_Python_25\Lib_Extensions\customtreectrl_SM.py", line 1683, in HitTest
    res, flags = child.HitTest(point, theCtrl, flags, level + 1)
  File "D:\Data_Python_25\Lib_Extensions\customtreectrl_SM.py", line 1683, in HitTest
    res, flags = child.HitTest(point, theCtrl, flags, level + 1)
  File "D:\Data_Python_25\Lib_Extensions\customtreectrl_SM.py", line 1683, in HitTest
    res, flags = child.HitTest(point, theCtrl, flags, level + 1)
  File "D:\Data_Python_25\Lib_Extensions\customtreectrl_SM.py", line 1654, in HitTest
    image_w, image_h = theCtrl._imageListNormal.GetSize(self.GetImage())
  File "P:\Python\lib\site-packages\wx-2.8-msw-unicode\wx\_gdi.py", line 6149, in GetSize
    return _gdi_.ImageList_GetSize(*args, **kwargs)
wx._core.PyAssertionError: C++ assertion "m_hImageList" failed at ..\..\src\msw\imaglist.cpp(128) in
wxImageList::GetSize(): invalid image list
```

Small Bugs

- If in the graphical editor a connection line is not connected the first time, it can't be connected anymore (just delete it and start a new connection line)
- Connections can be made, if not in edit mode
- When a control is deleted from the editor, the screen is not always refreshed, so the control stays visible (and sometimes it isn't even removed from the file

-

Solved Bugs

- plot-signal, output connected, but input not connected ??
- 12 april 2008, SM, Brick: Code Editor, F9 (send changes to output) didn't work anymore
- 12 april 2008, SM, Brick: Code Editor, Statusbar started with the wrong text
-

Debug Virtual Machine  (january 2009)

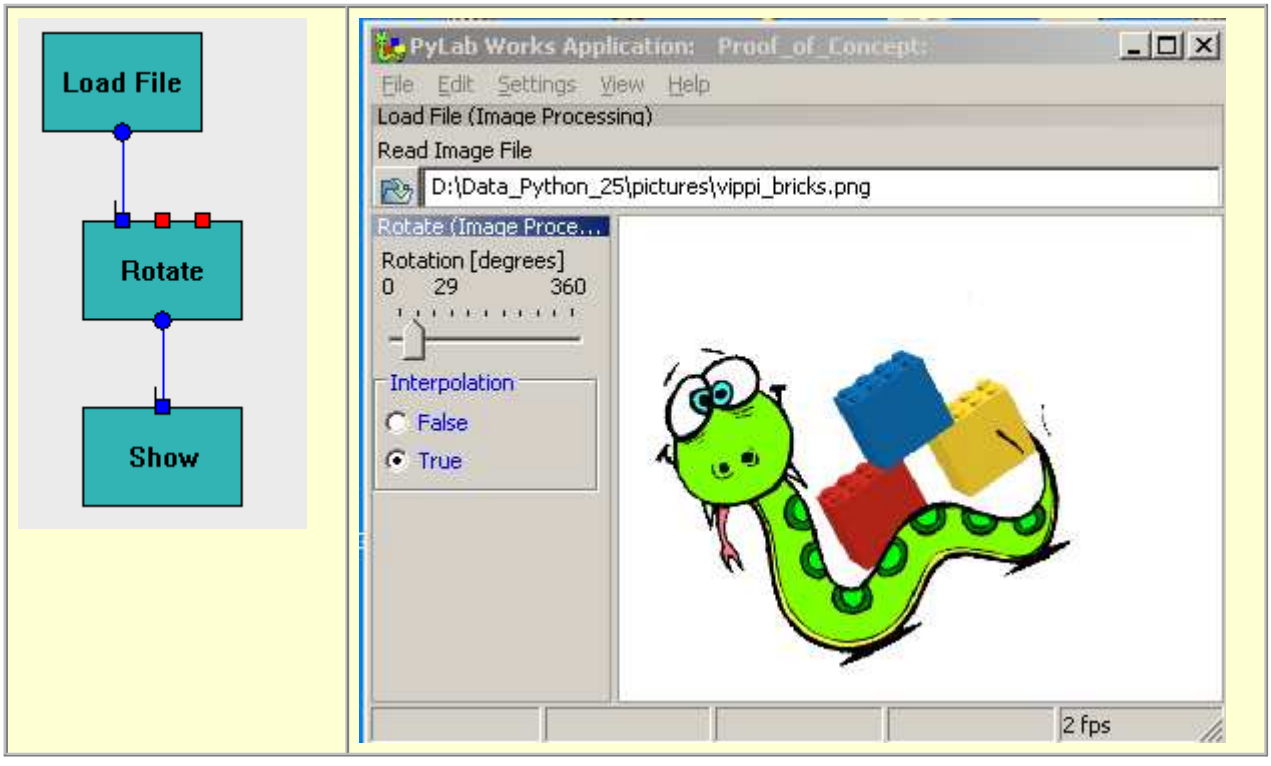
Application Designer / **Domain Expert** / **Control Designer / Core Developer**

Introduction

The Debug_Table is a convenient way to follow the flow of the data in relation to the execution of the virtual machine. For the moment the Debug_Table is created in the form of a html page, that can be viewed afterwards. In some cases it's more convenient to see the realtime data, which can easily be added. The Debug_Table and it's class definition can be found in PyLab_Works_Globals.
The following flags are of special interest for Debug_Table
-debugtable
-vmdelayxxxx, where xxxx is delay in msec

Example

For the example we've chosen the very first example, rotating an image, because it's simple but yet shows most of the features.



Looking at the image below, the meaning of the **columns (name of the Brick)** is shown in the green bars, which are shown at regular intervals. The first column contains two numbers, the first one is just a line

number for references, the **second number is the cycle counter** of the virtual machine.
The red blocks are execute statements. The execute statements are only shown if the inputs or the parameters of the Brick have changed.

line 2: here we see "Load General" which is the automatic loading of the saved parameters of each brick. We also can see that these parameters are send to the Parameter-array. Notice that sometimes the parameter array is loaded twice, which is perfectly normal, because the automatic loading not only send the value to the control, but also the parameter array. Now some controls (but not all) also trigger an event when their value is set. This event sends the value to the parameter. Another strange issue, the Brick Show loads a parameter, but doesn't have a explicit parameter, should be investigated further.

line 3: execute Brick Load_File, it's very logical that this will be the first Brick that will perform some action. We see it calculates an output signal and send the output signal to all receivers, in this case only the Brick Rotate.

line 4: The Brick Rotate receives a new input value. The value of the input signals is deliberately not shown, because it can easily be seen in the line above.

line 8: Here we see the parameter of the Brick Rotate changing, due to movement of the slider by the user. Because in this demo the virtual machine runs slowly, we can get more than 1 parameter change, before the virtual machine can act on these changes.

line 15: when the application closes, all the information stored in the buffer is dumped. Here we see that the automatic saving, saves the settings of each Brick.

PyLab_Works Debug (14-02-2009)

Protocol: pylab_works_programs\Proof_of_Concept

Line	Show / SN4	Load File / SN3	Rotate / SN2
1 / 1			
2 / 1	**Load General** = [2] **Par-1** = 1	**Load General** = [u'../pictures /chart_curve.png'] **Par-1** = ../pictures/chart_curve.png **Par-1** = ../pictures/chart_curve.png	**Load General** = [86.39999999 **Par-1** = 86.4 **Par-2** = 0
3 / 1		**Out-1** = <wx._core.Image; proxy of <Swig Object of type \wxImage *' at 0x452b150> > **Out-1 Send To** = Rotate('In', 1)	
4 / 1			**In-1** =
5 / 2			**Out-1** = <wx._core.Image; prox Object of type \wxImage *' at 0x **Out-1 Send To** = Show('In', 1)
6 / 2	**In-1** =		
7 / 5			
8 / 5			**Par-1** = 86.4 **Par-1** = 201.6
9 / 6			**Out-1** = <wx._core.Image; prox Object of type \wxImage *' at 0x **Out-1 Send To** = Show('In', 1)
10 / 6	**In-1** =		
Line	**Show / SN4**	**Load File / SN3**	**Rotate / SN2**
11 / 6			
12 / 6			**Par-1** = 212.4 **Par-1** = 212.4 **Par-1** = 212.4
13 / 7			**Out-1** = <wx._core.Image; prox Object of type \wxImage *' at 0x **Out-1 Send To** = Show('In', 1)
14 / 7	**In-1** =		
15 / 12	**Save General** = [2]	**Save General** = [u'../pictures /chart_curve.png']	**Save General** = [212.3999999

Current tags

In-3	The value of Input-3 has changed

Load General	Automatic Loading of the parameters, by the parent class Brick
Load ?????	Explicit Loading of the parameters, by the Brick itself
Out-3	The value of Output-3 has changed
Out-3 Send To	Send the changed value of Out-3 to all receivers
Par-2	The value of Par-2 has changed. In most cases this happens when the user interacts with the GUI. It also happens when the initial settings of the Brick are loaded.
Save General	Automatic Saving of the parameters, by the parent class Brick
Save ?????	Explicit Saving of the parameters, by the Brick itself

Adding new tags

```
399   # ********************************************
400   # Writes information to the buffers / file
401   # if Row == True,
402   #   first the execute statement is sent to the file
403   #   then the buffers are sent to the file
404   #   and the buffers are cleared
405   # It's safe to call this method at any time,
406   # because it will just return if Debugging is Off
407   # ********************************************
408   def Write ( self, Brick_Nr, Msg = '', Row = False ) :
409     if not ( self.On ) :
410       return
```

Example 1

too many output changes

Line	*BP Analysis / SN7*	*Oscilloscope / SN6*	*AD Converter*
281 / 345	**Out-1** = 2 **Out-1** = (array([89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647]), [])		
282 / 345		**In-2** = **In-2** =	

caused by setting the output twice, like this:

```
540   def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
541     Out[1] = 2
542     Out[1] = \
543       self.BP.execute ( data [S] )[0], MetaData
```

Line	BP Analysis / SN7	Oscilloscope / SN6	AD Converter
321 / 671	Out-1 = (array([89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647]), []) Out-1 = (array([89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647, 89039.88463647]), []) Par-3 = 180.0 Par-3 = 180.0		
322 / 671		In-2 = In-2 =	

Caused by changing the output and setting the modified flag. The modified flag should only be used when a complex variable like a numpy array (that is changed in place) is changed. In this case a list of numpy arrays is changed !!

```
540   def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
541     Out[1] = \
542       self.BP.execute ( data [S] )[0], MetaData
543     self.Out.Set_Modified ( 1 )
```

Example 2

Generating output signals, while there is no reason for, caused by incorrect or none change checking in the execute function.

Here Par-3 is changed by the user through the GUI. Because the signal generator (AD Converter) is stopped (you don't see any activity there, there shouldn't be any signals OUT-1 in the BP_Analysis.

Line	BP Analysis / SN7	Oscilloscope / SN6	AD Converter
191 / 534	Par-3 = 64.8		
192 / 534			
193 / 535	Out-1 = (array([90439.8781467, 90439.8781467, 90439.8781467, 90439.8781467, 90439.8781467]), []) Par-3 = 61.2		
194 / 535		In-2 =	

And if you look at the code, you see there's no checking at all.

```
541   def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
542       Out[1] = \
543         self.BP.execute ( data [S] )[0], MetaData
```

The correct way is to set the output signal, only if the input signal has changed, like this

```
541   def Generate_Output_Signals ( self, In, Out, Par, XIn, XOut, XPar ) :
542     if XIn [1] :
543       Out[1] = \
544         self.BP.execute ( data [S] )[0], MetaData
```

which will result in the normal patern.

Line	BP Analysis / SN7	Oscilloscope / SN6	AD Converter
281 / 385	**Par-3** = 187.2		
282 / 385			
283 / 386	**Par-3** = 205.2		
284 / 386			
285 / 387	**Par-3** = 219.6 **Par-3** = 230.4		

april 2008

Debugger / Diagnostics 

Application Designer / Domain Expert / **Control Designer / Core Developer**

Debug Options

For debugging we want to control **what information** are we interested in, **what level of detail** we're interested to see and **where** we want to see the information. In **General_Globals.py** a number of debug facilities is located.

To activate debugging, you need to specify the commandline flag **-debug** .

```
45 # ************************************
46 # WHAT TO DEBUG ( if Debug flag on )
47 # Just (un-)comment the following lines
48 # ************************************
49 Debug_What = set ()
50 Debug_What.add ( 'TIO' )
51 Debug_What.add ( 'Brick' )
```

```
52 # ************************************************
53 # And in the following lines, just change the number
54 # if necessary
55 # At the moment we allow:
56 #    0 = not deep
57 #    1 = deeper
58 #    2 = deepest
59 # ************************************************
60 Debug_How_Deep = {}
61 Debug_How_Deep [ 'TIO' ] = 2
62 Debug_How_Deep [ 'Brick' ] = 2
```

```
68 def Debug_Dump ( *args ) :
```

```
77 def Debug_Dump_Trace ( *args ) :
```

For a number of complex objects, the dump routine will automatically use a significant part, so for a inifile it will only use the filename. So we can simply dump a inifile, like this :

```
    # ********************************************************
    # ********************************************************
    def Save_Settings ( self, ini ) :
      if ( 'Load_Save' in Debug_What ) :
        Debug_Dump_Trace (
          'tScintilla_Editor ( My_Control_Class ) Save Settings to :', ini )
```

which will result in:

```
tScintilla_Editor ( My_Control_Class ) Save Settings to : pylab_works_programs\VPython.cf
  Called from : 1354 , PyLab_Works.py
    Called from : 1311 , PyLab Works.py
```

Bricks Diagnostics

By setting the Diagnostics flag of a brick to True, it lot of diagnostic information of both that Brick itself and the used controls in that Brick will be printed.

```
124
125     self.Diagnostic_Mode = True
126
```

```
****** WBD: (SN6) = Code Editor 2 : Brick Running in Diagnostic Mode
        <Name> <Type> <Required>
Input  1 = ['IN[1]', 0, False],  Value = None
Input  2 = ['IN[2]', 0, False],  Value = None
Input  3 = ['IN[3]', 0, False],  Value = None
Input  4 = ['All IN', 1, False],  Value = None

Output 1 = ['OUT[1]', 2],  Value = None,  Changed = False

Param  0 = None$$
Param  1 = $$

Control 1
  Owner = <brick_Plotting.t_Code_Editor_2 object at 0x033BBAF0>
  Input Channel = None
  Type = 11
```

april 2008

Test Suite 

Introduction

The Test-Mode will run a number of application scripts and report output and errors to specific log files. The Test_Mode can be started through menu .... In the future the Test-Mode capabilities could be used to run batch processes, although I don't think that's very attractive for programs relying heavily on GUI.
After starting the Test_Mode (by starting the program with the parameter "-test" or by pressing the menu-item Test), the special program PyLab_Works_Test_Bench is launched and PyLab_Works will close itself. PyLab_Works_Test_Bench will then restart PyLab_Works with each of the specified scripts.
To control the test mode, every application configuration file can contain the special section ....
If this section is omitted, the TestBench will just run the program for a pre-definied number of cycli.

Test commands

Example

Current Test Suite

```
10 # ************************************************
11 # List of all scripts to be invoked in the testrun
12 # ************************************************
13 TestPrograms = []
14 TestPrograms.append ( 'aap'      )
15 TestPrograms.append ( 'btc_test' )
16
17
18 # ************************************************
19 # run all the scripts
20 # ************************************************
21 for Program in TestPrograms :
22   runwait ( [ 'Python', 'PyLab_Works.py',
23               '-TestRun', Program ],
24               shell = True )
```

Default section

```
•1616     if PG.TestRun :
•1617       output ( 1, 'try :' )
•1618       output ( 2, 'TestRun_Timer += 1 ')
•1619       output ( 1, 'except :')
•1620       output ( 2, 'TestRun_Timer = 0')
 1621
•1622       output ( 1, 'if TestRun_Timer == 100 :')
•1623       output ( 2, 'PG.Final_App_Form.OnClose ( None )' )
```

july 2008

Analyze Python 

Inspect

>>> getmembers (file_support, isfunction)
[('File_Delete', <function File_Delete at 0x02071E30>),
('File_Exists', <function File_Exists at 0x02071DF0>),
('Find_Files', <function Find_Files at 0x02071F30>),
('Find_Files_1', <function Find_Files_1 at 0x02071EF0>),
('Force_Dir', <function Force_Dir at 0x02071EB0>),
('Get_Absolute_Path', <function Get_Absolute_Path at 0x02071CF0>),
('Get_Relative_Path', <function Get_Relative_Path at 0x02071D70>),
('Main_Module_Filename', <function Main_Module_Filename at 0x02071DB0>),
('Tree_Delete', <function Tree_Delete at 0x02071E70>),
('_', <function _ at 0x02071670>)]

>>> getmembers (file_support, isclass)

[('test_class', <class 'file_support.test_class'>)]

The ***getmembers***() function retrieves the members of an object such as a class or module. The eleven functions whose names begin with ``is'' are mainly provided as convenient choices for the second argument to getmembers(). They also help you determine when you can expect to find the following special attributes:

Type	Attribute	Description	Notes
module	__doc__	documentation string	
	__file__	filename (missing for built-in modules)	
class	__doc__	documentation string	
	__module__	name of module in which this class was defined	
method	__doc__	documentation string	
	__name__	name with which this method was defined	
	im_class	class object that asked for this method	(1)
	im_func	function object containing implementation of method	
	im_self	instance to which this method is bound, or None	
function	__doc__	documentation string	
	__name__	name with which this function was defined	
	func_code	code object containing compiled function bytecode	
	func_defaults	tuple of any default values for arguments	
	func_doc	(same as __doc__)	
	func_globals	global namespace in which this function was defined	
	func_name	(same as __name__)	
traceback	tb_frame	frame object at this level	
	tb_lasti	index of last attempted instruction in bytecode	
	tb_lineno	current line number in Python source code	
	tb_next	next inner traceback object (called by this level)	
frame	f_back	next outer frame object (this frame's caller)	
	f_builtins	built-in namespace seen by this frame	
	f_code	code object being executed in this frame	
	f_exc_traceback	traceback if raised in this frame, or None	
	f_exc_type	exception type if raised in this frame, or None	
	f_exc_value	exception value if raised in this frame, or None	
	f_globals	global namespace seen by this frame	
	f_lasti	index of last attempted instruction in bytecode	
	f_lineno	current line number in Python source code	
	f_locals	local namespace seen by this frame	
	f_restricted	0 or 1 if frame is in restricted execution mode	

	f_trace	tracing function for this frame, or None
code	co_argcount	number of arguments (not including * or ** args)
	co_code	string of raw compiled bytecode
	co_consts	tuple of constants used in the bytecode
	co_filename	name of file in which this code object was created
	co_firstlineno	number of first line in Python source code
	co_flags	bitmap: 1=optimized | 2=newlocals | 4=*arg | 8=**arg
	co_lnotab	encoded mapping of line numbers to bytecode indices
	co_name	name with which this code object was defined
	co_names	tuple of names of local variables
	co_nlocals	number of local variables
	co_stacksize	virtual machine stack space required
	co_varnames	tuple of names of arguments and local variables
builtin	__doc__	documentation string
	__name__	original name of this function or method
	__self__	instance to which a method is bound, or None

Note:(1)
Changed in version 2.2: im_class used to refer to the class that defined the method.

getmembers(*object*[, *predicate*])

Return all the members of an object in a list of (name, value) pairs sorted by name. If the optional ***predicate*** argument is supplied, only members for which the predicate returns a true value are included.

getmoduleinfo(*path*)

Return a tuple of values that describe how Python will interpret the file identified by ***path*** if it is a module, or None if it would not be identified as a module. The return tuple is (***name***, ***suffix***, ***mode***, ***mtype***), where ***name*** is the name of the module without the name of any enclosing package, ***suffix*** is the trailing part of the file name (which may not be a dot-delimited extension), ***mode*** is the open() mode that would be used ('r' or 'rb'), and ***mtype*** is an integer giving the type of the module. ***mtype*** will have a value which can be compared to the constants defined in the imp module; see the documentation for that module for more information on module types.

getmodulename(*path*)

Return the name of the module named by the file ***path***, without including the names of enclosing packages. This uses the same algorithm as the interpreter uses when searching for modules. If the name cannot be matched according to the interpreter's rules, None is returned.

ismodule(*object*)

Return true if the object is a module.

isclass(*object*)

Return true if the object is a class.

ismethod(*object*)

Return true if the object is a method.

isfunction(*object*)

Return true if the object is a Python function or unnamed (lambda) function.

istraceback(*object*)

Return true if the object is a traceback.

isframe(*object*)

Return true if the object is a frame.

iscode(*object*)

Return true if the object is a code.

isbuiltin(*object*)

Return true if the object is a built-in function.

isroutine(***object***)
Return true if the object is a user-defined or built-in function or method.

ismethoddescriptor(***object***)
Return true if the object is a method descriptor, but not if ismethod() or isclass() or isfunction() are true.
This is new as of Python 2.2, and, for example, is true of int.__add__. An object passing this test has a __get__ attribute but not a __set__ attribute, but beyond that the set of attributes varies. __name__ is usually sensible, and __doc__ often is.
Methods implemented via descriptors that also pass one of the other tests return false from the ismethoddescriptor() test, simply because the other tests promise more - you can, e.g., count on having the im_func attribute (etc) when an object passes ismethod().

isdatadescriptor(***object***)
Return true if the object is a data descriptor.
Data descriptors have both a __get__ and a __set__ attribute. Examples are properties (defined in Python), getsets, and members. The latter two are defined in C and there are more specific tests available for those types, which is robust across Python implementations. Typically, data descriptors will also have __name__ and __doc__ attributes (properties, getsets, and members have both of these attributes), but this is not guaranteed. New in version 2.3.

isgetsetdescriptor(***object***)
Return true if the object is a getset descriptor.
getsets are attributes defined in extension modules via `PyGetSetDef` structures. For Python implementations without such types, this method will always return `False`. New in version 2.5.

ismemberdescriptor(***object***)
Return true if the object is a member descriptor.
Member descriptors are attributes defined in extension modules via `PyMemberDef` structures. For Python implementations without such types, this method will always return `False`. New in version 2.5.

26.10.2 Retrieving source code

getdoc(***object***)
Get the documentation string for an object. All tabs are expanded to spaces. To clean up docstrings that are indented to line up with blocks of code, any whitespace than can be uniformly removed from the second line onwards is removed.

getcomments(***object***)
Return in a single string any lines of comments immediately preceding the object's source code (for a class, function, or method), or at the top of the Python source file (if the object is a module).

getfile(***object***)
Return the name of the (text or binary) file in which an object was defined. This will fail with a TypeError if the object is a built-in module, class, or function.

getmodule(***object***)
Try to guess which module an object was defined in.

getsourcefile(***object***)
Return the name of the Python source file in which an object was defined. This will fail with a TypeError if the object is a built-in module, class, or function.

getsourcelines(***object***)
Return a ***list of source lines*** and ***starting line number*** for an object. The argument may be a module, class, method, function, traceback, frame, or code object. The source code is returned as a list of the lines corresponding to the object and the line number indicates where in the original source file the first line of code was found. An IOError is raised if the source code cannot be retrieved.

getsource(***object***)
Return the text of the source code for an object. The argument may be a module, class, method, function, traceback, frame, or code object. The source code is returned as a single string. An IOError is raised if the source code cannot be retrieved.

april 2008

Internationalization 

Application Designer / **Domain Expert** / Control Designer / Core Developer

Introduction

As text is an important factor in presenting domain knowledge, internationalization is of great importance. The standard way of implementing internationalization (in wxPython) is the use of Il8N, Il10N. After downloading the manual I was surprised how complex it was (200 page manual). As "simplicity" goes before "perfect" in my designs, I developed a much simpler way, not as perfect as Il8N, Il10N, but good enough and much simpler. There's even a big disadvantage in using Il8N, Il10N, as the source string is stored as the key of the database, changing the source string will make it unfindable for all other languages. It's not necessary to write the texts in a source file in US-English (the default standard), but you can write texts in any language, as long as you use only one language per file / module.

History

Future Ideas

- ***new:*** selecting a second preferred language, that will be invoked it the primary language string can not be found.
- ***improvement:*** available languages should not be stored in language_support

V1.1 released 18-07-2008

- ***improvement:*** Translation through Babel Fish done with LXML instead of BeatifulSoup
- ***improvement:*** Translation moved to a separate thread
- ***improvement:*** Autocompletion in translation disabled
- ***improvement:*** Sources and Babel Fish follows Translation, also when navigation with keyboard
- ***improvement:*** Keywords in Translation editor cleared
- ***bugfix:*** Tripple quoted strings on more lines were not correctly read / written

V1.0 released 10-04-2008

- ***orginal release***

Cookbook

There's just one thing to remember: just wrap every string in a function call to underscore, with an extra first parameter of zero. So instead of writing:

```
print  "helllo"
```

write:

```
print   _(0, "hello" )
```

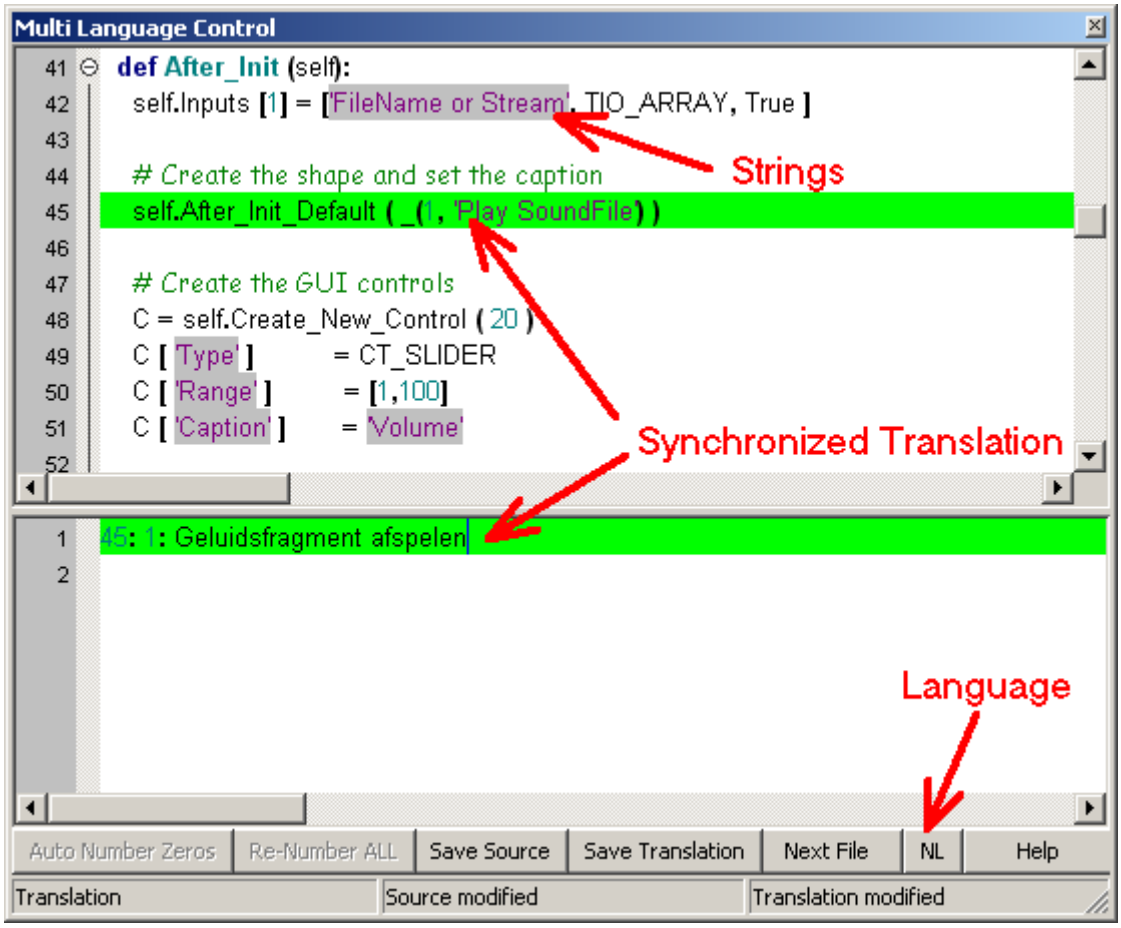
Now PyLab_Works will try to find a translation in the selected language, if not found the original text is used. Translations that are not found, are logged in a log-file. PyLab_Works will also replace the zero with an unique integer number that serves as a reference for the other languages.

Translation Tool

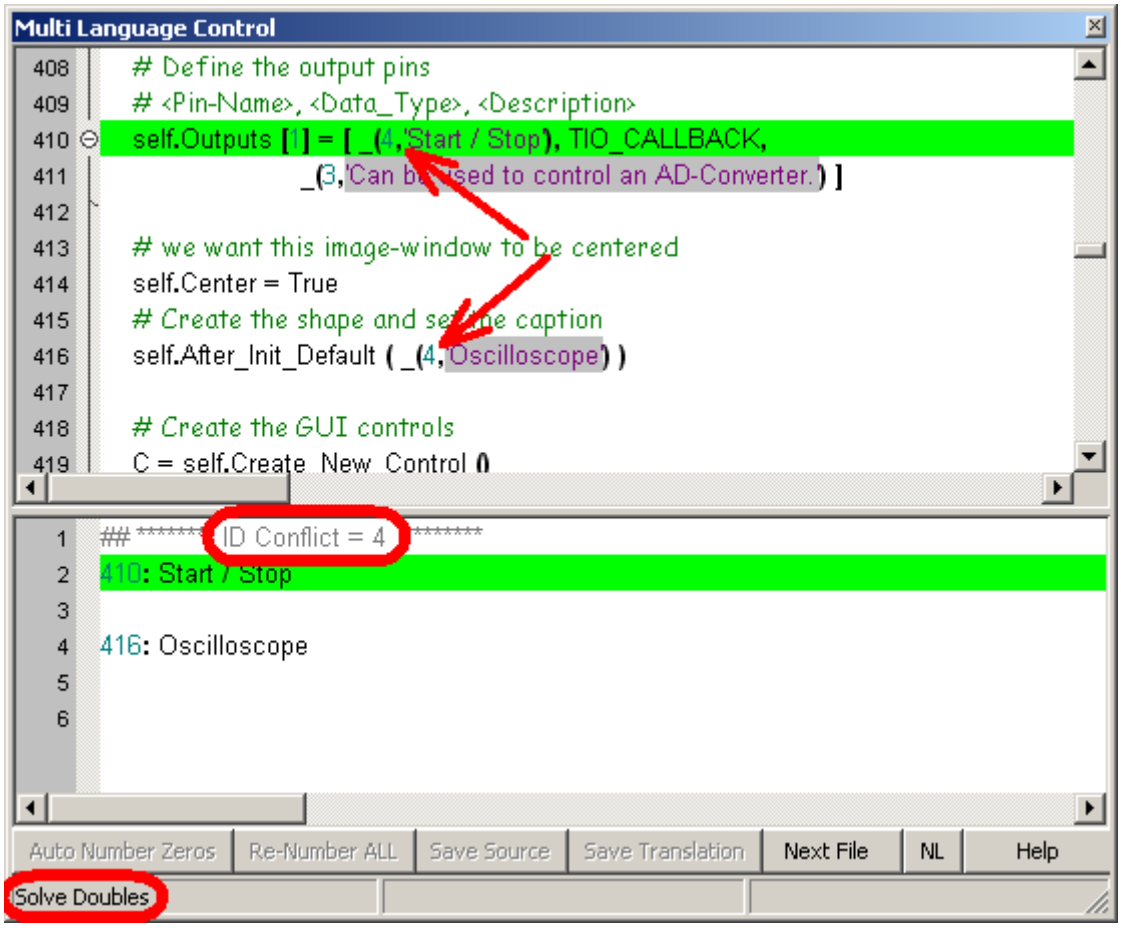
With the Translation tool it's easy to make a translation of the strings in one or more files, and also to test the original files for strings and see if these strings are made multi-lingual.

After opening a Python source-file, the source file is loaded in the top editor, and the translated strings are displayed in the bottom editor. The 2 editors are synchronized, so selecting a line in the translation editor, will show the source string highlighted in the top editor. All strings in the source editor will have a gray background color, so you can easily find strings that are not multi-lingual and modify them into multi-lingual strings. At the bottom are a series of buttons, which will only be enabled if meaningful in that situation. The statusbar displays the state of the translation (Conflicts Detected / Zeros Found / Translation), and the modifications of source file and translated strings. As far as possible, changed files are automatically saved. The language button will popup a list of all available languages (except "US", because that's the base language. If the desired language is not listed, just add the language in the file language_support.py, to

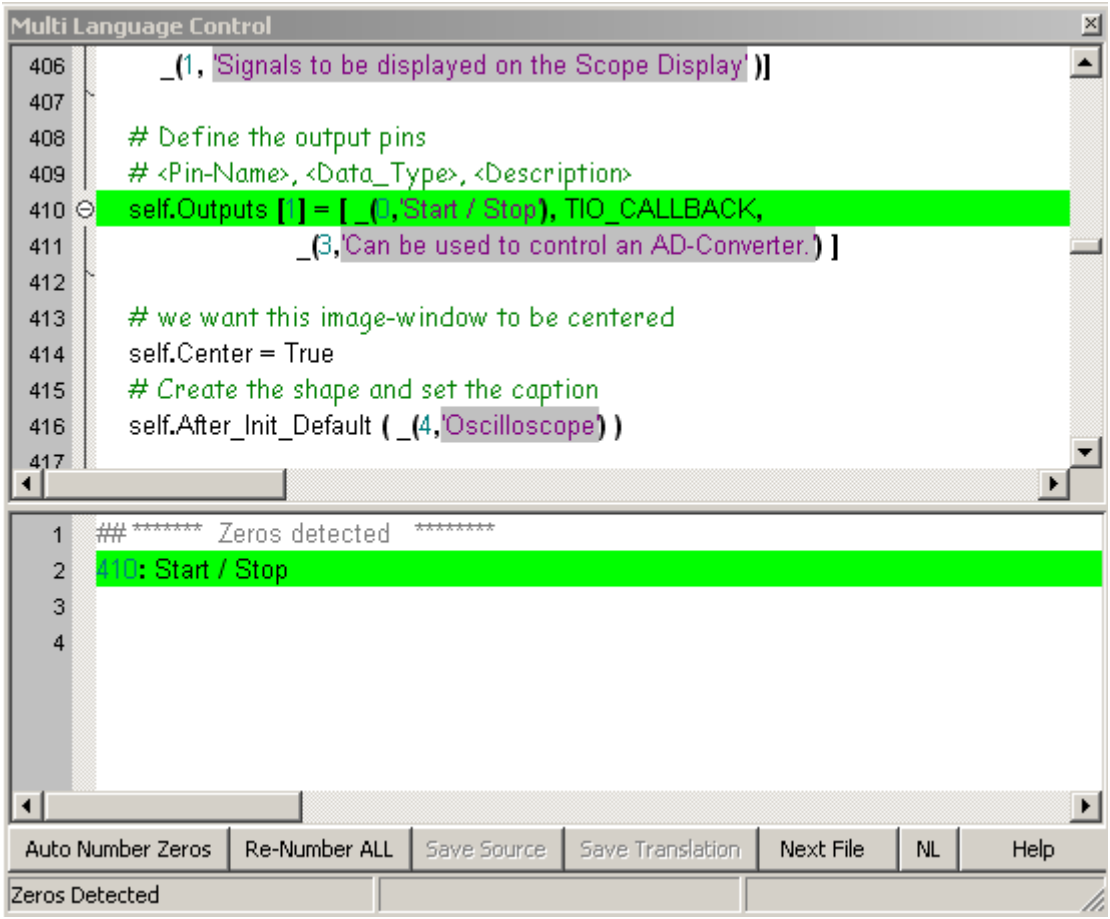
```
20 Language_IDs = [ 'NL', 'RO', 'US']
```



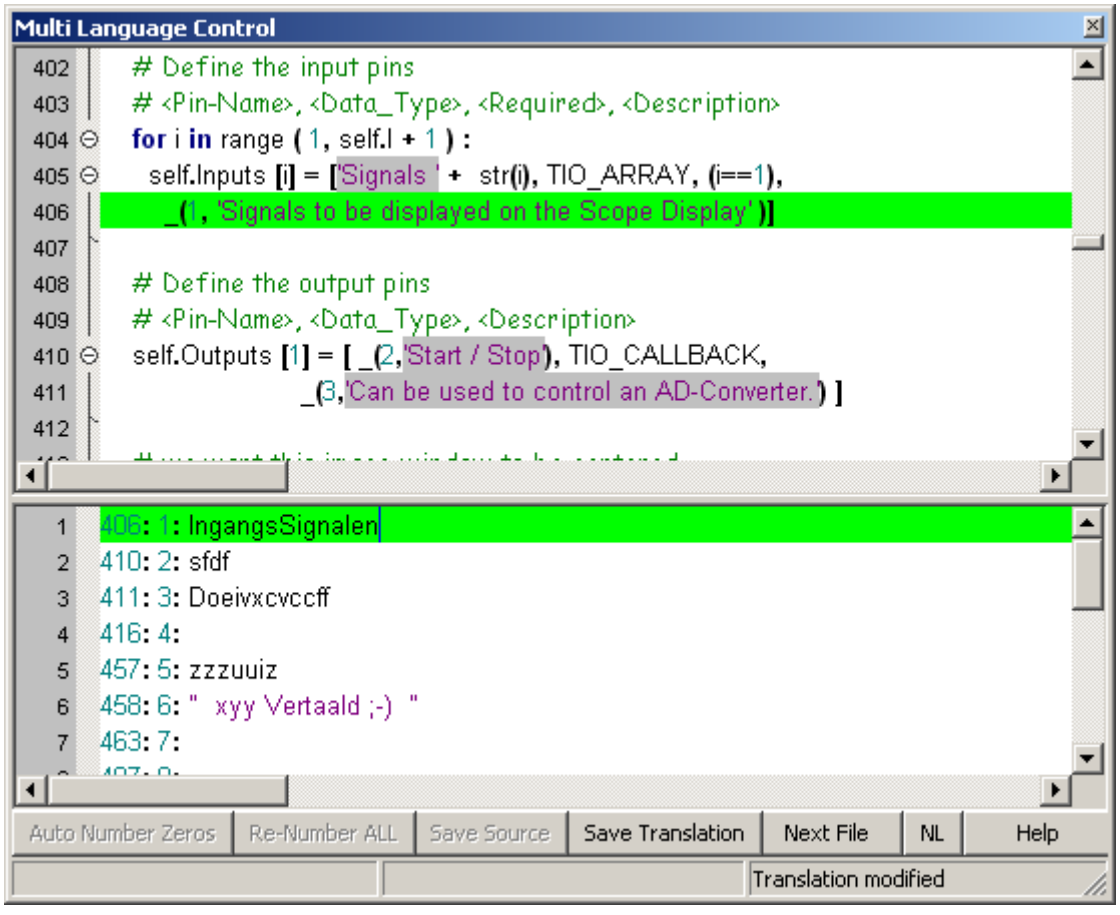
The first action after opening a source file the sourcefile is checked for conflicts, i.e. multi-lingual strings that are different and have the same ID-nr (unequal zero). The conflicts (probably introduced by copy and paste) can't be solved automatically, because the program can't judge which of the 2 strings is the oldest, and therefor may have already translations to other languages. After solving the conflicts, by changing one or more of the conflicting string IDs into "0", press the "Save Source" button, so the source will be saved and the source file will be automatically rechecked.



The next the program performs is the presence of zero IDs. If found they are listed in the second editor and the renumber buttons become available. In general you will press "Auto Number Zeros", which will fill all zero IDs with a unique number, save the modified source file and start a recheck (which of course is OK now).
There is also a possibility, to renumber all multi-ligual strings in the source file, so all strings get an ordered number. But if there are already translations of this source file, they are all lost !! Therefor this action is protected by a confirmation question.

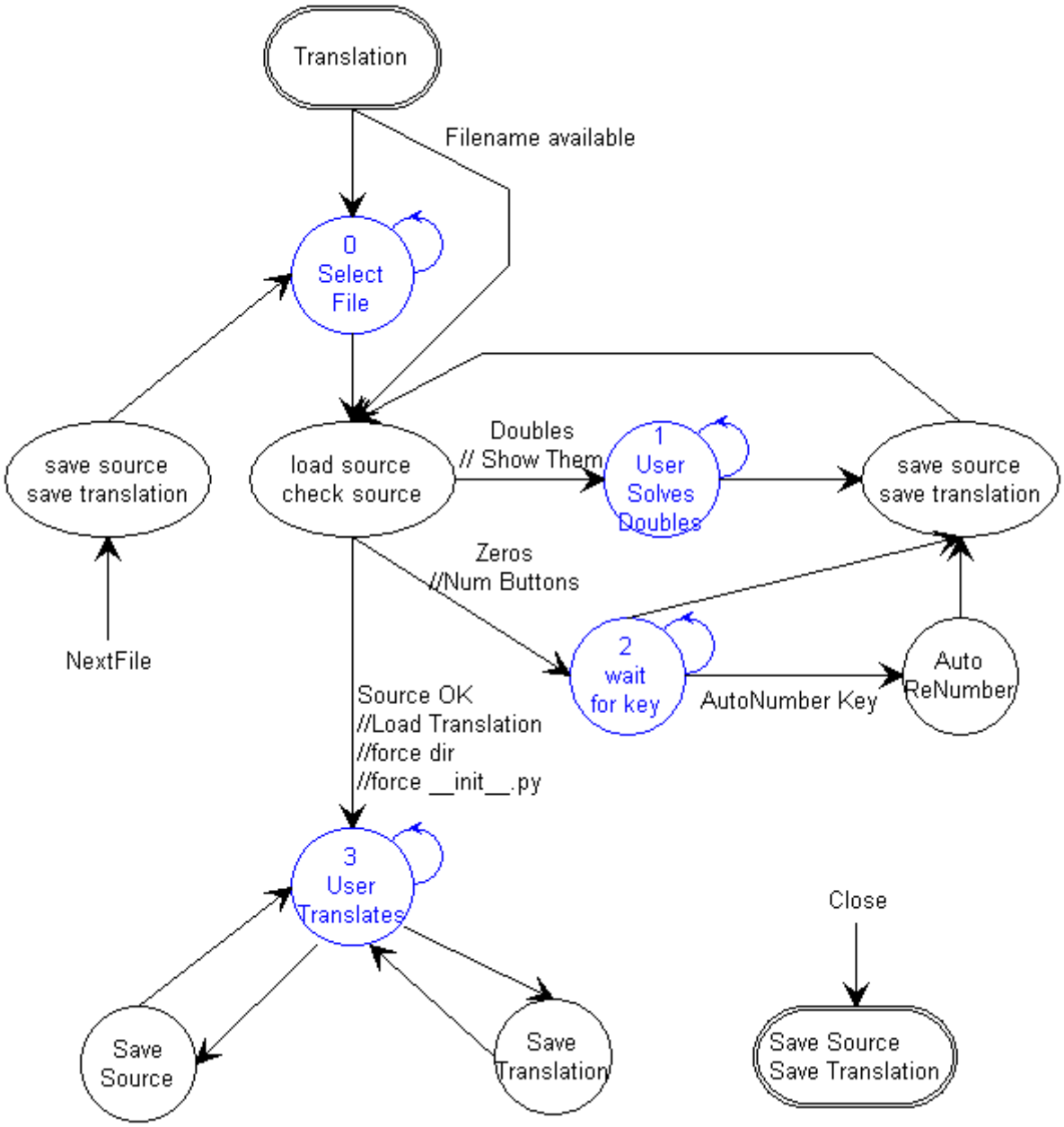


After conflicts and zeros are solved the translation will be loaded. Now just edit all the translation strings.



Behind the scenes

Here is the state machine, used to create the program.



Html / pdf / ... Viewer  (november 2008)

Application Designer / Domain Expert / Control Designer / **Core Developer**

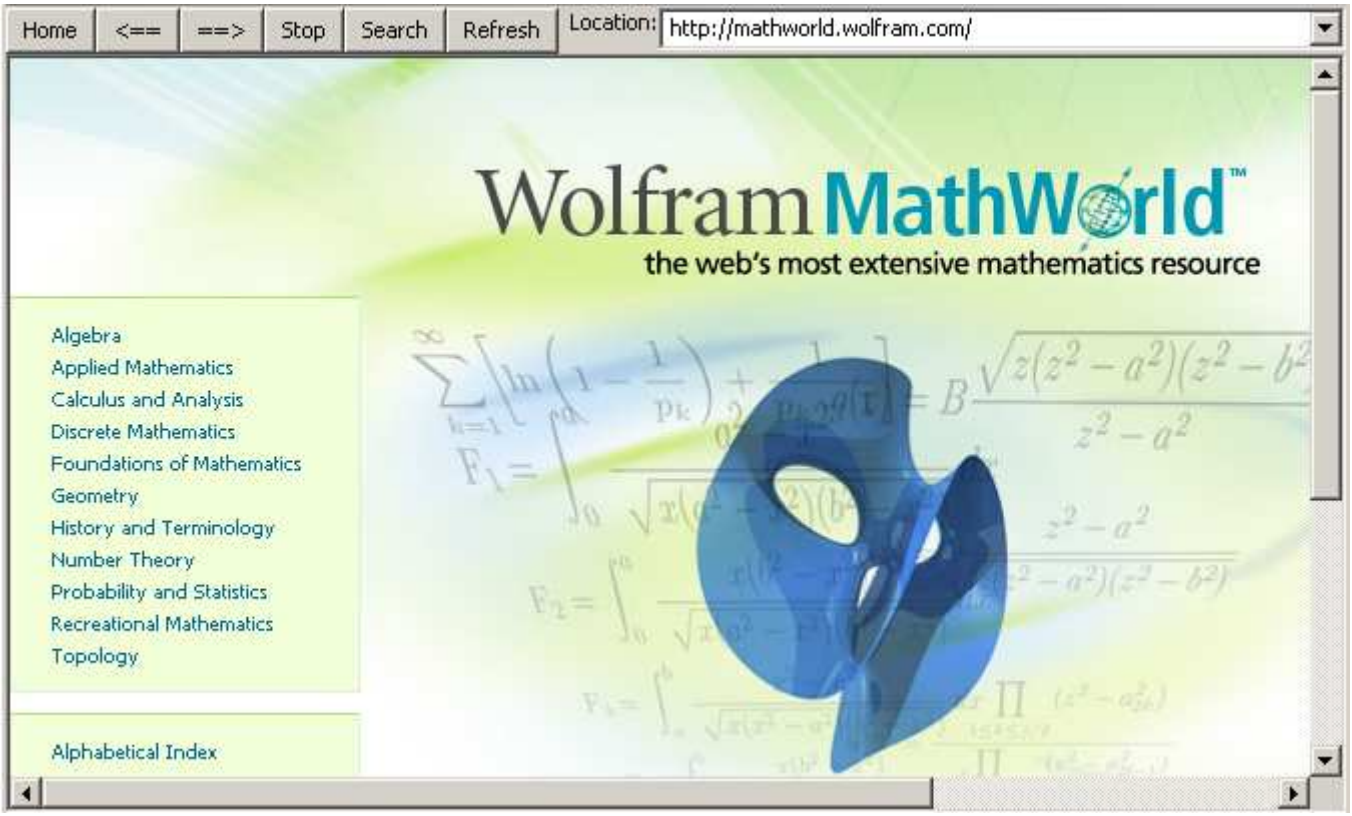
Introduction

Making a general viewer, that can view several standard documents, like web-pages / local html-files / pdf-files / chm-files, is very easy under Windows, but almost impossible under other Operating Systems. Besides that we want to view full blown html-pages (that can't be viewed with wx.html.HtmlWindow) and we want to view html pages with widgets (which can only be viewed with wx.html.HtmlWindow).
The problem might be largely solved, as soon as webkit is integrated in wxPython (at this moment I read that there are already alpha versions available).

The basic solution chosen at this moment is, we place both a wx.html.HtmlWindow and a activeX-IE component together on one panel. Make them both full blown, and use one of the two, depending on the contents, while making the other invisible. For non-Windows OS, we try to display as much as possible on wx.html.HtmlWindow, and everything that doesn't fit, will be sent to the default browser. For non-Windows OS, we could improve the behavior by adding more (external) controls for other document types, like pdf-files and chm-files.

Top Level: URL_Viewer

At the top level, we have a viewer with navigation buttons, that can display a whole range of files: html (+ frames, +CSS), html-widgets, websites, pdf, MS-office-documents, pdf, ... at least this yields for the Windows Operating System. When used in another OS, it will use the default browser for files that can't be displayed with wx.html.HtmlWindow. So in that case the capabilities of the default browser will determine if a document can be viewed or not. The combobox keeps track of the history and is automatically saved and restored.



Low Level

At the low level, we've for all OS's two different viewers available, which on Windows both points to iewin.IEHtmlWindow, but for other OS's points either to the internal viewer or to the external default browser.

```
100 if Win_PlatForm :
101   import  wx.lib.iewin  as iewin
102   IEHtmlWindow     = iewin.IEHtmlWindow
103   IEHtmlWindow_Ext = iewin.IEHtmlWindow
104 else :
105   print '       Contact the developer of this program'
106   IEHtmlWindow     = _My_IEHtmlWindow
107   IEHtmlWindow_Ext = _My_IEHtmlWindow_Ext
```

By always using IEHtmlWindow or IEHtmlWindow_Ext, we can make the program OS independent.

Menu Support  (november 2008)

Application Designer / Domain Expert / Control Designer / **Core Developer**

Introduction

The module menu_support, has convenience procedures for

- menubar
- popup menu
- toolbar menu

MenuBar

The MenuBar support has the following features:

- Automatic creation of a default menu
- Easy creation of a menubar through a dictionary
- Easy dynamic extension of default or other menu
- Default actions for a number of standard menu-items
- Automatic binding of methods of the form "OnMenu_<menuitem>"
- Easy binding from within controls on the frame
- Automatic enable / disable menu-items based on focused control

The MenuBar, definied as the class **Class_Menus**, has a default menu, which should be preferable used, so all forms will have the same menus. So in general the menu can be created in a frame, like this:

```
137     self.MenuBar = Class_Menus ( self )
```

NOTE: some components that will be placed on the frame, will try to hook them self into the menu. For the moment this is done by looking for a control named "MenuBar" in the Frame. It would be better to look for the control of the type "MenuBar".

Now you can bind actions to existing or new menu items, e.g. to bind a method to the menu **File,** item **Save** :
(If the menu and / of the item doesn't exist yet, it's automatically added)

```
260     # ******************************************
261     # Menu completions
262     # ******************************************
263     MB = self.MenuBar.Bind_MenuItem
264     MB ( 'File', 'Save', self._On_Menu_FileSave )
```

It's not strictly necessary to do the bindings in the frame, often it's even far more convenient to place both the bindings and the completion methods in a component. e.g. it's very logical that a scintilla textcontrol binds actions to printing menu items, which could look something like this (so this piece of code is not from the frame but from the editor class !!) :

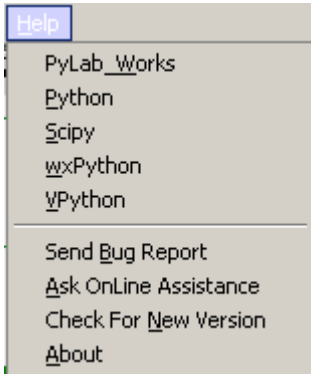
```
451     # ******************************************************
452     # Try to bind to standard menu items
453     # ******************************************************
454     try :
455       MB = self.TopFrame.MenuBar.Bind_MenuItem
456       MB ( 'File', 'Print'         ,self._On_Menu_Print         )
457       MB ( 'File', 'Print Preview' ,self._On_Menu_Print_Preview )
458       MB ( 'File', 'Page Setup'    ,self._On_Menu_Page_Setup    )
```

The menubar will register all the owners of the methods, so it can automate the whole menu handling by

itself. On popup of the menu, the menubar itself will enable / disable the correct items, depending on which control on the form has focus. So when you've 2 editors on the form and you launch a print command, the editor that has focus will be printed.

I don't have a nice feature to enable / disable the menus itself (but I think that's of less importance).

For a number of menu-items, there's a default action bound (but of course you can override it):



For menuitems which are already in the menu at the moment the menubar is created, it's not necessary to bind the event, but instead you can also realize the binding by taking care the method "OnMenu_<item>" exists in the frame.

Main Control or Sub Control

In many situations a control can catch all the events, but often the main control (frame) will want to pre-process and / or even Veto the action.
As this is written in the very premature stage of the concept, we might change it in the future.
Method 1 is the method which performs the above actions, but it can perform this for the standard menu items. So here we already have our first reason to improve this handling in the future.

Goal:

- The event is first captured by the frame
- The event handler of the frame determines if the event handler of the control will be called or not
- The control determines if a menu item is enabled or disabled

In the frame we bind to the default menu-item, by defining a event_handler with the correct name: line 14 in the picture below.

```
4  # ************************************************************
5  # ************************************************************
6  class my_App_Form ( wx.Frame, Menu_Event_Handler):
7    def __init__ ( self, parent )
8      wx.Frame.__init__(self, parent )
9
10     # Create a default menu bar
11     self.MenuBar = Class_Menus ( self )
12
13   # ************************************************
14   def OnMenu_Open ( self, event ) :
15     print 'Main Application : File|Open '
16     # enable the focussed control itself to handle the event
17     event.Skip ()
```

In the control that's placed somewhere on the frame, we bind to the menu-item, with the special method Bind_MenuItem, at line 27 in the picture below. The event handler at line 32 will be called if not captured by the parent frame, or if the parent frame issues an event.Skip().

```
21 # ************************************************************
22 # ************************************************************
23 class Some_Control ( ... ) :
24   def __init__ ( ... ) :
25     try :
26       MB = self.TopFrame.MenuBar.Bind_MenuItem
27       MB ( 'File', 'New/Open'       ,self._On_Menu_File_Open     )
28     except :
29       pass
30
31   # ***********************************************
32   def _On_Menu_File_Open ( self, event ):
33     print 'Scintilla : File|Open '
34     ... handle the event
35     event.Skip ()
```

STD Redirection  (november 2008)

Introduction

For redirection (and restore) of the stdout and stderr, there are functions located in PyLab_Works_Globals.

```
159       PG.    Set_StdOut ( self.Log )
160       PG.Restore_StdOut ( self.Log )
161       PG.    Set_StdErr ( self.Log )
162       PG.Restore_StdOut ( self.Log )
```

Both Set_... and Restore_... are called with the redirection object.
The procedures are fail safe, you may call Set_... more than once, before calling Restore_... more than once.
The redirection is stored as a link list.
A typical call to the redirection might look like this:

```
151   # ********************************
152   # ********************************
153   def Toggle_StdOut ( self, event ) :
154     if self.CB_StdOut.GetValue () :
155       PG.Set_StdOut ( self.Log )
156     else :
157       PG.Restore_StdOut ( self.Log )
```

Docking VPython  (november 2008)

Application Designer / Domain Expert / Control Designer / **Core Developer**

Introduction

We want to use VPython in PyLab_Works to visualize 3D physics. To make VPython an integral, easy to use visualization tool, it's necessary to dock VPython in PyLab_Works.
At first sight docking of VPython seems almost impossible, due to the special structure of the VPython package, the way it handles it's graphical output and the awesome high level of the functions and other

automation. The solution provided here is for MS-Windows only, but the ideas presented here might stimulate an Linux and / or Mac guru to implement something equal for these Operating Systems. The solution found, is one big trick and is more base on trial and error than on pure knowledge, so major improvements / simplifications ought to be possible if re-factored by a specialist.

Docking

The first problem is to dock the VPython display window into the wxPython container. This is basically an easy step, although some mechanism is needed to wait until the VPython display window really exists. The VPython display window can be found by searching for a window with the title "VPython" (of course it's better to give the window in VPython a more unique name). Searching for a specific window type is discouraged, because it relies on the implementation in VPython, that might vary among versions, so we set it to None (=NULL).

```
self.VP = win32gui.FindWindow ( None, 'VPython' )
```

As we know the container where the VPython display should be docked, we can easily retrieve the handle of the container (in this case "self.STC") :

```
PP = self.STC.GetHandle ()
```

And by using the SetParent function, the VPython window will be docked and fixed within our application:

```
win32gui.SetParent ( self.VP, PP )
```

Remove Caption and Frame

For real docking we want to remove the caption and the frame of the VPython window, so the user can't control the buttons and mouse actions on the caption and frame-borders. In windows this is not possible at all, because the frame settings can only be set at the time of creation of the window, of which we've no control at all. But with a trick we can realize the same effect, by the correct positioning and sizing of the window in a container.

Position and Sizing

Of course we want that the container that contains the VPython application also controls the size of the VPython display. By making the Vpython display a little bit larger than the container, we can achieve a situation where caption and frame borders will fall just outside the container and thus will be unreachable for the user. Normally positioning of a window should be simply done with the function SetWindowPos, but for some unknown reason, that doesn't work when the VPython display is docked. It might have a relation with the fact that VPython display properties x,y,width,height are not accessible (not even for reading) while the display is visible. Based on this assumption we found a workable solution by trial and error, which looks like this:
hide the VPython window

```
413         visual.scene.visible = False
```

show the VPython window (this fully recreates the VPython window, because afterwards it's undocked and has another handle).

```
433       visual.scene.visible = True
```

Set the position and size of the VPython window, notice the negative y-position to get rid of the caption. On winXP in teletubie-view and Vista these values should be slightly larger.

```
#flags = win32con.SWP_ASYNCWINDOWPOS or \
#        win32con.SWP_SHOWWINDOW     or \
#        win32con.SWP_FRAMECHANGED
flags = win32con.SWP_SHOWWINDOW or \
        win32con.SWP_FRAMECHANGED
win32gui.SetWindowPos ( self.VP, win32con.HWND_TOPMOST,
                        -4, -22, w+8, h+26, flags )
```

redock the VPython window

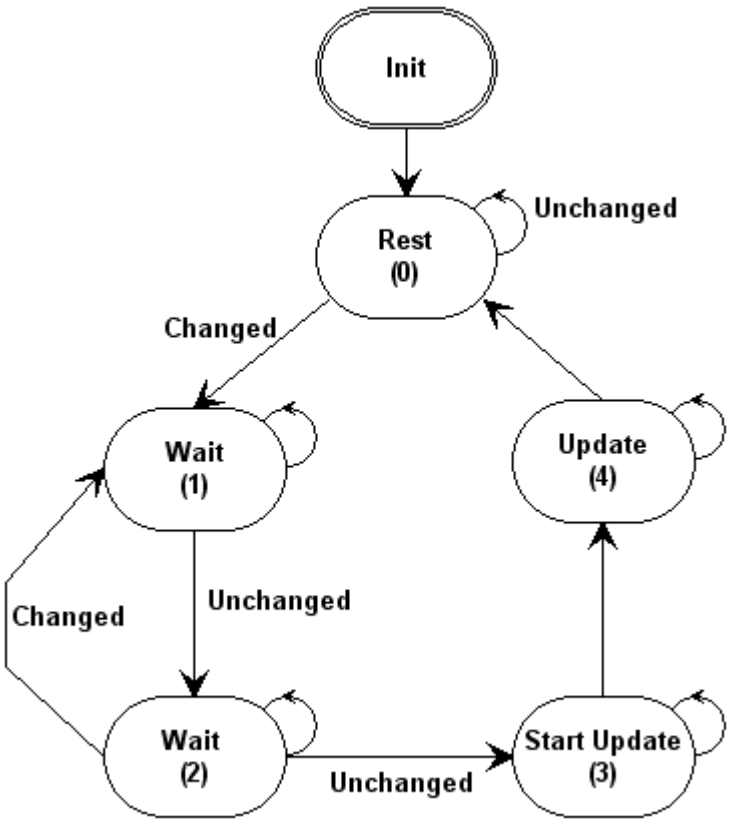
```
win32gui.SetParent ( self.VP, PP )
```

Misc

There's one other thing to remind, visual must be imported as one of the first modules, otherwise the DLL can't be loaded !!??

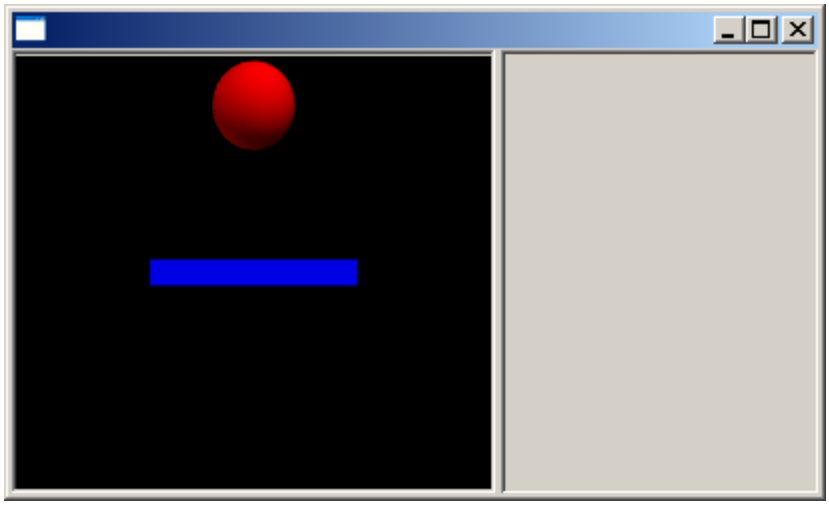
Timing

Timing is the next thing to solve: recreating a VPython window takes a considerable amount of time (a few seconds), while some resize events comes at a high repetition rate (10 per second or even more). Sizers in wxPython behaves quit nice, they only send a resize at the end, but a windows resize sends resize commands continuously. We solved this problem with a finite state-machine, clocked by a wx.Timer. A clock period of 100 msec seems to work quit well (not perfect !!). Maybe it should be better to detect a mouse up event.



Demo

The demo code can be found in PyLab_Works.

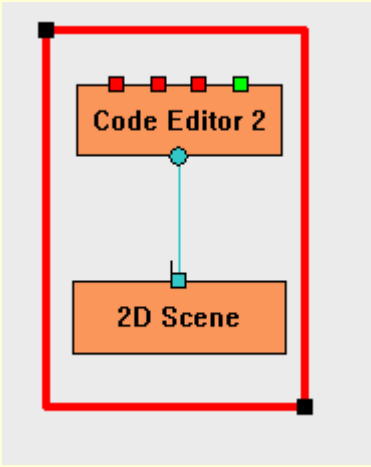


Grouping  (december 2008)

Application Designer / Domain Expert / Control Designer / **Core Developer**

Grouping

By double-clicking on the canvas, a rubberband rectangle will be toggled. This rubberband can be used to group a number of individual Bricks into a new Brick, The new Brick will contain all connections that are available between the bricks in the selection. After the selection is made, the grouping is activated through the RM-menu.



In control_output_viewer.py an attempt is done, to merge 2 controls tCMD_Shell_Doc = tCMD_Shell + tDoc_Viewer.
There are some problems to overwin :

```
********** ERROR in GUI-string *************
1 self.NB=wx.Notebook( self.Dock,style = wx.BK_LEFT)
2 self.CMD_Shell=tCMD_Shell( self.NB)
3 self.Html=tDoc_Viewer( self.NB)
4 name = self.CMD_Shell.GetName()
5 self.NB.AddPage ( self.CMD_Shell, name)
6 name = self.Html.GetName()
7 self.NB.AddPage ( self.Html, name)
8 Sizer = wx.BoxSizer ( )
9 Sizer.Add ( self.NB, 1, wx.EXPAND )
10 self.Dock.SetSizer ( Sizer )
11
**********
Traceback (most recent call last):
  File "D:\Data_Python_25\support\gui_support.py", line 471, in __init__
    exec ( self.code, p_globals, p_locals )
  File "<string>", line 5, in <module>
  File "P:\Python\lib\site-packages\wx-2.8-msw-unicode\wx\_controls.py", line 2965, in Ad
    return _controls_.BookCtrlBase_AddPage(*args, **kwargs)
TypeError: in method 'BookCtrlBase_AddPage', expected argument 2 of type 'wxWindow *'
Error File : D:\Data_Python_25\PyLab_Works\control_output_viewer.py
Error Func : __init__
Error Line : 244
Error Code :     self.wxGUI = PG.Create_wxGUI ( GUI, my_parent = 'self.Dock' )
```

Imports  (november 2008)

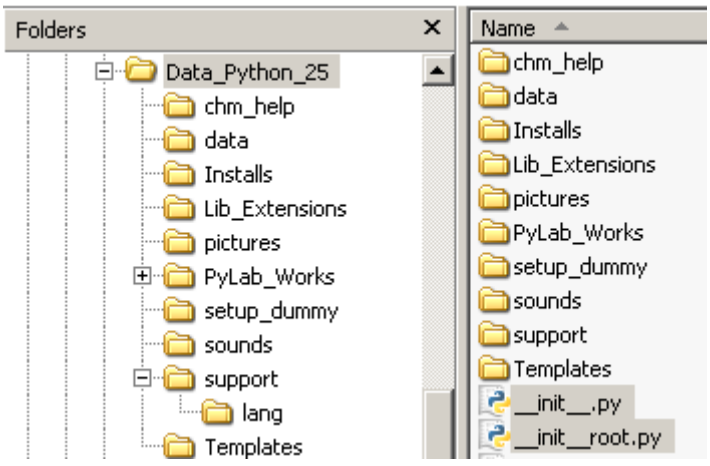
Application Designer / Domain Expert / Control Designer / **Core Developer**

Introduction

Redundancy is one of the worst items in software. You have to maintain and extend code at more than one place, and if you don't realize that some code is redundant, debugging is a hell of a job, certainly in a dynamic language as Python is. Therefor we give higher priority to redundancy removal than to hierarchical structure. As a consequence, this means that every file of the project should be reachable from any other file. So it's essential to have a good PYTHONPATH and a good import strategy. A second demand is that every module should be able to act as a main file by runing it's main section.

Solution

The solution_old worked perfect until I wanted to build a windows binary distro.
The root of all programs, contains a template __init__.py.



By the program Deploy.py (which also performs other tasks), this __init__.py is copied to all necessary locations.
The __init__.py module will search for the module __init__root.py, starting at it's own directory and then walking the tree up. Then it will add the path where __init__root was found to the PythonPath and imports __init__root.

__init__root.py will add all the subdirectories, starting at it's own location walking down, to the PythonPath.

The above procedure works both in the working directory, and also in the windows binary distro.

Solution_old

As every directory must have an __init__ module, we can use the __init__ module to ensure all the paths are reachable. So every module must import __init__ as the first module. As every file knows at what level of the root it's positioned, the __init__ module can easily import a module at the root level. Every __init__ module in every subpath is exactly like this :

```
 1 # This __init__ module should be used
 2 # in every path of the project
 3
 4 # Add my own path to the PYTHONPATH
 5 import os, sys
 6 Path, File = os.path.split ( __file__ )
 7 Path = os.path.join ( Path, '../' )
 8 Path = os.path.normpath ( Path )
 9 if not ( Path in sys.path ) :
10   sys.path.append ( Path )
11
12 #  Get the whole rest of the project
13 import __init__root
```

Line 7, points to the root directory, so this might be different depending on the level of the current directory.
At line 9,10 the root is added to the PythonPath (if it's not already there).
Then at line 13 the module __init__root, located in the root is imported, which will make a complete PythonPath.

The __init__root module, recursive searches for all subdirectories and adds them to the Python Path (if they are not yet there).

```python
import os, sys

# ******************************************************
# Searches for all Paths, starting at path
# returns a sorted list of full path names
# ******************************************************
def Find_Paths ( path, Py_Paths ) :
  files = os.listdir ( path )
  for file in files :
    new_path = os.path.join ( path, file )
    if os.path.isdir ( new_path ) :
      Py_Paths.append ( new_path )
      Find_Paths ( new_path, Py_Paths )
  Py_Paths.sort()
# ******************************************************


# ******************************************************
# extend the system path with ALL paths in the project
# ******************************************************
Py_Paths = []
Path = os.path.split ( __file__ ) [0]
Find_Paths ( Path, Py_Paths )
for path in Py_Paths :
  if not ( path in sys.path ) :
    sys.path.append ( path )
# ******************************************************
```

24 October 2008

GUI support 

Application Designer / Domain Expert / **Control Designer** / Core Developer

ToDo

- if component placed on a Notebook is not a wx.Window, automatically insert a wx.Panel in between
- PreView-wxGUI in AUI-panes

Introduction

This library file is integral part of PyLab_Works and was never meant to be released as a separate package. Therefor the file contains a lot of dependencies which can easily be removed and therefor this page is just a quick and dirty compilation of some screen shots. Although these libraries might be very attractive for newbies to getting started, a firm warning should be placed: using these libraries prevents you from learning (or let you forget) the basic building of a GUI with wx-Python components and specially the use of sizers. Although the program works very well, it's still work in progress and will undergo more changes in the near future).

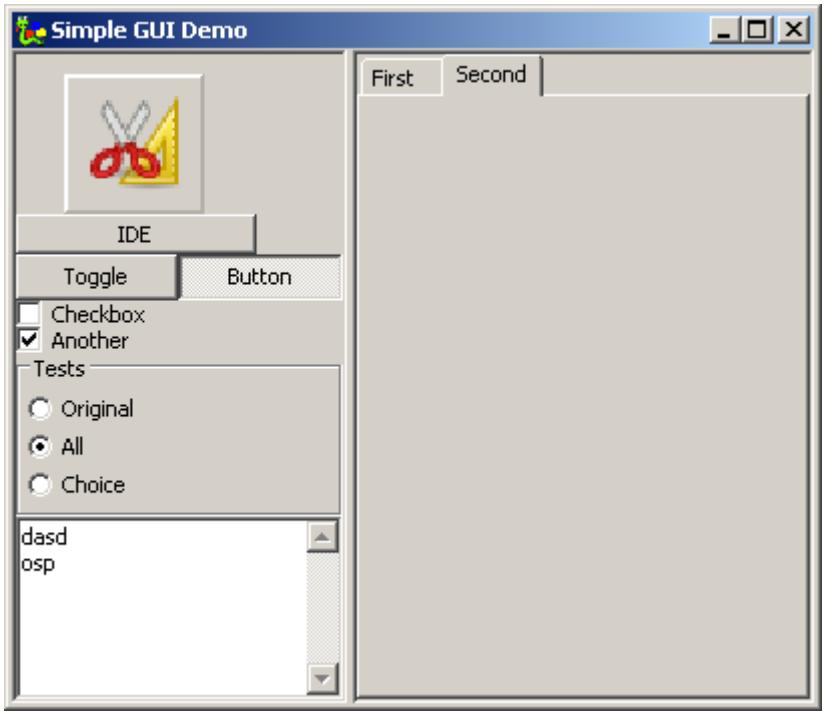
The library is tested under windows-XP, '2.8.7.1 (msw-unicode)' and Ubuntu wx 2.8 uni gtk+, both using Python 2.5

Creating a GUI in wxPython isn't easy. There are several tools to make this design easier, but I couldn't get one of them working. There is one that works (XRC), but I hate it I find XRC a typical example of "Why should we do it easy, as it can be done complex".
Therefor I created a simple solution, which is even more powerful than XRC.
Gui_support is a library, with the following features

- **create a GUI** from a string

- **implicit import** of (special) component types
- **Save and Restore settings** of all GUI-components
- **convenience procedures** to save typing
- **F12 preview** of the GUI within the PW-IDE

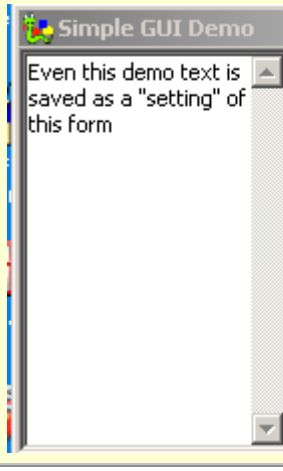
The code for the application below (including saving and restoring the settings) is just 38 lines (see picture and the bottom).



Principle

The principle is to define the structure in a string, where the indentation defines which component is the container for other components. So each line in the string defines 1 component, and has the following **syntax : name, type, arguments**. If we simplify the above example to just the textcontrol at the left bottom and the notebook at the right, but keeping the automatically save and restore settings, the code looks like this (on the right the resulting GUI):

```
29      GUI = """
30      self.Splitter    ,SplitterHor
31        self.Text      ,wx.TextCtrl      ,style = wx.TE_MULTILINE
32        self.NB        ,wx.Notebook
33          p4           ,wx.Panel         ,name = 'First'
34          p5           ,wx.Panel         ,name = 'Second'
35      """
36      self.wxGUI = Create_wxGUI ( GUI, IniName = 'self.Ini_File' )
```



At line 30, we define a SplitterWindow, which is the container for the TextCtrl defined at line 31 and the Notebook defined at line 31.
The Notebook defined at line 32 is the container for the Panels (notebook-pages) p4 and p5.
Finally the GUI is generated by line 36.
Each component definition consists of a name, followed by the type, followed by any keyword-list the

component can accept.
There are created a few new convenience components, like the "SplitterHor" (which produces a Vertically Splitter, weird naming !!) in the above example. But if you don't mind the weird naming and extra typing, you can of course also use the normal wx.SplitterWindow.
Furthermore, the settings are only restored (and can be saved later) if the component's name starts with "self.".
Even for very simple designs it works great, because you don't need to bother about sizers:

```
20      GUI = """
21        self.STC  ,Base_STC
22      """
23      self.wxGUI = Create_wxGUI ( GUI, my_parent = 'self.Dock' )
24
```

Implicit Imports

The library file contains a dictionary, that defines implicit imports. This list can be expanded to your own needs. Whenever a type in the GUI-string is detected, and the type is defined in the dictionary and the import is not yet done, an extra import statement will be added.

```
178 # ************************************************************
179 # Special types for which an automatic import will be done
180 # ************************************************************
181 _Special_Types = {
182 'Base_STC' : [ 'from Scintilla_support import Base_STC', 0 ],
183 }
184 # ************************************************************
```

Error reporting

If an error occurs during the creation of the GUI, the error is caught by the GUI-support library. The GUI-library will print the code generated by the GUI library, where each line will be preceded by a line-number and then print the error message itself. Because the error is caught, the program tries to continue its normal flow (during development this is sometimes very handy).
Below an example of in error in line 9 of the generated code. The error here is caused by a new graphical component "tScintilla_Editor" and we want to place that Editor (direct) on a page of a Notebook. To add a component to a new Notebook page, we need to give it a name and therefor in line 9 the name of the component tScintilla_Editor is asked, but the tScintilla_Editor has no method GetName.

```
********** ERROR in GUI-string *************
1 from tree_support import Custom_TreeCtrl_Base
2 global p2
3 self.SplitV=wx.SplitterWindow( Dock)
4 self.Split=wx.SplitterWindow( self.SplitV)
5 self.Tree=Custom_TreeCtrl_Base( self.Split)
6 self.NB=wx.Notebook( self.Split)
7 self.Editor=tScintilla_Editor( self.NB,Brick=Brick,ini=ini,Test=Test)
8 p2=wx.Panel( self.NB,name = "Test")
9 name = self.Editor.GetName()
10 self.NB.AddPage ( self.Editor, name)
**********
Traceback (most recent call last):
  File "../support\gui_support.py", line 396, in __init__
    exec ( self.code, p_globals, p_locals )
  File "<string>", line 9, in <module>
AttributeError: 'tScintilla_Editor' object has no attribute 'GetName'
********** End ERROR in GUI-string *************
```

To solve the above problem there are many solutions:

- the most easy way is to put a wx.Panel (which does have a name) on the Notebook page first and then put the tScintilla control in that panel. But the disadvantage is that our GUI-string becomes unnecessary longer.
- a better way is to add the GetName method to our newly derived component
- a third method is to extend the gui-generator with all kinds of try/except statements. But that makes the generated code (which we want to see in case of an error) less readable.

So here we'll choose for the second solution, but ...
... this gives an other error. You are only allowed to put wx.Window components on a Notebook page. So for the moment we'll choose the first solution. In a later stage I'll extend the gui-generator so it'll put a panel in between components if necessary.

Here an output of the improved error message (V1.6), which now specifies the file and line number.

```
********** ERROR in GUI-string *************
1 from tree_support import Custom_TreeCtrl_Base
2 global p2
3 self.SplitV=wx.SplitterWindow( Dock)
4 self.Split=wx.SplitterWindow( self.SplitV)
5 self.Tree=Custom_TreeCtrl_Base( self.Split)
6 self.NB=wx.Notebook( self.Split)
7 self.Editor=tScintilla_Editor( self.NB,Brick=Brick,ini=ini,Test=Test)
8 p2=wx.Panel( self.NB,name = "Test")
9 name = self.Editor.GetName()
10 self.NB.AddPage ( self.Editor, name)
11 name = p2.GetName()
**********
Traceback (most recent call last):
  File "../support\gui_support.py", line 399, in __init__
    exec ( self.code, p_globals, p_locals )
  File "<string>", line 10, in <module>
  File "P:\Python\lib\site-packages\wx-2.8-msw-unicode\wx\_controls.py", line 2965, in A
    return _controls_.BookCtrlBase_AddPage(*args, **kwargs)
TypeError: in method 'BookCtrlBase_AddPage', expected argument 2 of type 'wxWindow *'
Error File : D:\Data_Python_25\PyLab_Works\control_scintilla.py
Error Func : __init__
Error Line : 232
Error Code :     self.wxGUI = Create_wxGUI ( GUI, IniName = 'self.Ini', my_parent = 'Doc
********** End ERROR in GUI-string *************
```

Class Create_wxGUI

```
196 class Create_wxGUI ( object ) :
197   def __init__ ( self,
198                  GUI,
199                  IniName    = '',
200                  my_parent  = 'self',
201                  code       = ''   ) :
```

GUI = the string containing the GUI definition
IniName = the string representation of the name of the inifile (if available)
my_parent = string representation of the parent window. Because in general you'll define the outer window as a separate class, "self" will be ok.

code = meant to cascade a number of string definitions

```
43   # ******************************************
44   def On_Close ( self, event ) :
45     if self.Ini_File :
46       self.Ini_File.Section = self.Ini_Section
47       self.wxGUI.Save_Settings ()
48     event.Skip()
```

```
351   def Ready ( self ) :
352     pass
```

This is a dummy method to find the end of the code for the F12 preview.

Convenience Components

See for the latest extensions the source code. At the moment of the first release, the following convenience components were available:

- **SplitterHor**, a wx.SplitterWindow that splits the space in horizontal direction
- **SplitterVer**
- **PanelHor**, a wx.Panel that splits the space in the horizontal direction, the second parameter are the weight factors of components on the panel, ie "0011" means give the first two components the minimum size and divide the remaining space between the third and fourth component
- **PanelVer**
- PageHor, PageVer, the same as PanelHor, PanelVer, I think these will be removed in the future
- **GUI_Notebook**, a flatnotebook, with the style I always use

Auto Save / Restore settings

```
 # ****************************************************************
 # Also a good position to generate code to save / restore settings
 # Components are only added if the following conditions are met:
 #   - must have a name starting with "self." (otherwise we can't save it)
 #   - must be definied in the _Save_Restore dictionary
 # ****************************************************************
```

```
# ************************************************************************
# Here are the definitions for saving and restoring settings
# For each item 2 strings need to be defined:
#   - the string to get the value from the component
#   - the string to set the value of the component
# There are substitues possible:
#   %% will be replaced by the name of the component (as it used in the forms init)
#   %  will be replaced by the value read from the config-file
# ************************************************************************
_Save_Restore = {
  'wx.CheckBox'       : ( 'GetValue()',         '%%.SetValue(%)'       ),
  'wx.Notebook'       : ( 'GetSelection()',     '%%.SetSelection(%)'   ),
  'wx.RadioBox'       : ( 'GetSelection()',     '%%.SetSelection(%)'   ),
  'wx.SplitterWindow' : ( 'GetSashPosition()', 'wx.CallLater ( 100, %%.SetSashPosit:
  'wx.TextCtrl'       : ( 'GetValue()',         '%%.SetValue(str(%))'  ),
  'wx.ToggleButton'   : ( 'GetValue()',         '%%.SetValue(%)'       ),
}
# ************************************************************************
```

Other Convenience functions

These classes are not only meant to prevent a lot of typing, but more over to **prevent redundancy** !!

- **My_Main_Application**, a class defining a simple application
- **My_Frame_Class**, a class defining a simple form with some auto save / restore settings

F12 - Preview

For the preview the code starting with the GUI-string until the Create_wxGUI (line 34) or to the Ready method (line 38) is executed. This code is packed into a frame and completed with an wx-application. The wrapper is defined in the file '../support/gui_template_dont_touch.py'. This file can be changed if other wrapper code is needed.

```
29      GUI = """
30
31      """
32
33
34      self.wxGUI = Create_wxGUI ( GUI )
35
36
37
38      self.wxGUI.Ready ()
```

To use the preview just call the function PreView_wxGUI with the whole text of an editor. Here is the call I use in my editor

```
# ************************************
# F12       is GUI pre-view
# Shift-F12 is GUI in AUI panes  (ToDo)
# ************************************
elif key == wx.WXK_F12 :
  PreView_wxGUI ( self.GetText () )
```

Full demo code

```
from gui_support import *

# ************************************************************************
# ************************************************************************
class Simple_Test_Form ( My_Frame_Class ):
  def __init__ ( self, ini = None ) :
    My_Frame_Class.__init__ ( self, None, 'Simple GUI Demo', ini )
    bmp_IDE    = Get_Image_Resize ( '../pictures/applications-accessories.png', 48 )
    GUI = """
    self.Splitter          ,SplitterHor
      Panel2               ,PanelVer, 01
        p1                 ,wx.Panel
          B_IDE            ,BmpBut          ,bitmap = bmp_IDE   ,pos = (24,10) ,size=(
          B_IDE_Run        ,wx.Button       ,label = "IDE"      ,pos = ( 0,80) ,size=(1
        p2                 ,PanelVer, 00001
          p3               ,PanelHor, 11
            self.B_Toggle  ,wx.ToggleButton ,label = "Toggle"
            self.B_Button  ,wx.ToggleButton ,label = "Button"
          self.CB_CheckBox ,wx.CheckBox     ,label = 'Checkbox'
          self.CB_Another  ,wx.CheckBox     ,label = 'Another'
          self.RB_Test     ,wx.RadioBox     ,label = 'Tests', choices=['Original', 'Al
          self.Text        ,wx.TextCtrl     ,style = wx.TE_MULTILINE
      self.NB              ,wx.Notebook     ,style = wx.NO_BORDER
        p4                 ,wx.Panel        ,name = 'First'
        p5                 ,wx.Panel        ,name = 'Second'
    """
    self.wxGUI = Create_wxGUI ( GUI, IniName = 'self.Ini_File' )

    self.Bind          ( wx.EVT_CLOSE,        self.On_Close    )
    self.B_Toggle.Bind ( wx.EVT_TOGGLEBUTTON, self.On_B_Toggle )
    self.B_Button.Bind ( wx.EVT_TOGGLEBUTTON, self.On_B_Button )

  # ******************************************************
  def On_Close ( self, event ) :
    if self.Ini_File :
      self.Ini_File.Section = self.Ini_Section
      self.wxGUI.Save_Settings ()
    event.Skip()

  # ******************************************************
  def On_B_Toggle ( self, event ) :
    self.B_Button.SetValue ( not ( self.B_Toggle.GetValue() ) )
  def On_B_Button ( self, event ) :
    self.B_Toggle.SetValue ( not ( self.B_Button.GetValue () ) )
# ************************************************************************

if __name__ == '__main__':
  My_Main_Application ( Simple_Test_Form )
```

march 2008

Bind - Skip Events 

Introduction

Despite all explanations in books, on the web and in newsgroups, I again bumped into a problem with event

bindings. So after solving my problem, I decided to write another view on events. I'm not a programmer, just a user and therefor my notes on this page may not be exactly correct, nor may I use the right words. It's also not a complete overview of bindings, e.g. there seems to be two distinct event types: **Basic Events** and **Command Events**, which behave different (but I don't know what the difference is, moreover I don't even know when I deal with one or the other).
There are numerous pages on the web, describing the event handling, and the best ones I found (although not solving my problem) are:

- http://www.zetcode.com/wxpython/events/ (contains also other good tutorials)
- http://wiki.wxpython.org/self.Bind_vs._self.button.Bind
- http://www.wxwidgets.org/manuals/stable/wx_eventhandlingoverview.html#eventhandlingoverview

This document handles the following issues and tries to give a cookbook for event binding:

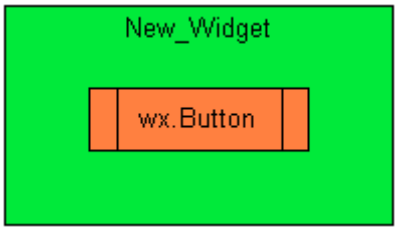
- should the **binding be done to** control or to the panel / frame / window containing that control
- should the **source of the event** be specified or not
- how to get hold of the **right control-ID** in the event handler

For the general binding method:

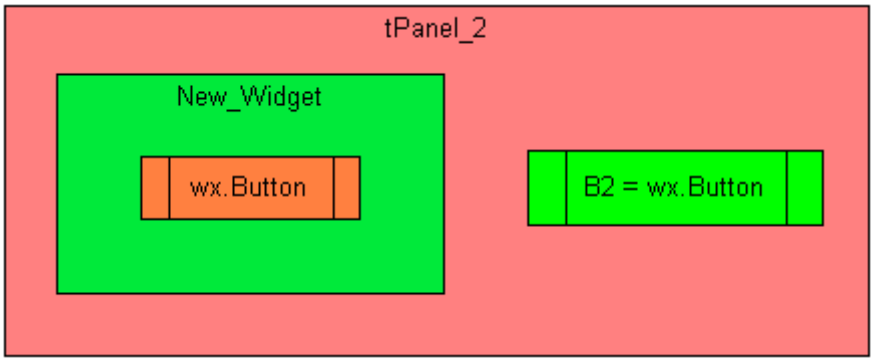
```
<destination>.Bind ( <event-type>, <handler>, [ <event-source> ] )
```

The environment

I needed a float / logarithmic slider, so (with the help of Robin Dunn, thanks !!), I created a new widget in a very simple way. I derived the new widget from a wx.Panel and created the slider in the __init__ of the derived panel. All functionality of the logarithmic slider was placed in the derived panel.
For simplicity I'll use in this document a wx.Button (representing the logarithmic slider) and added no extra functionality to that button. To make small enough images, I used some shortcuts in this document, like "s = self", which is certainly not advised for real programs. So here's what the new widget looks like:

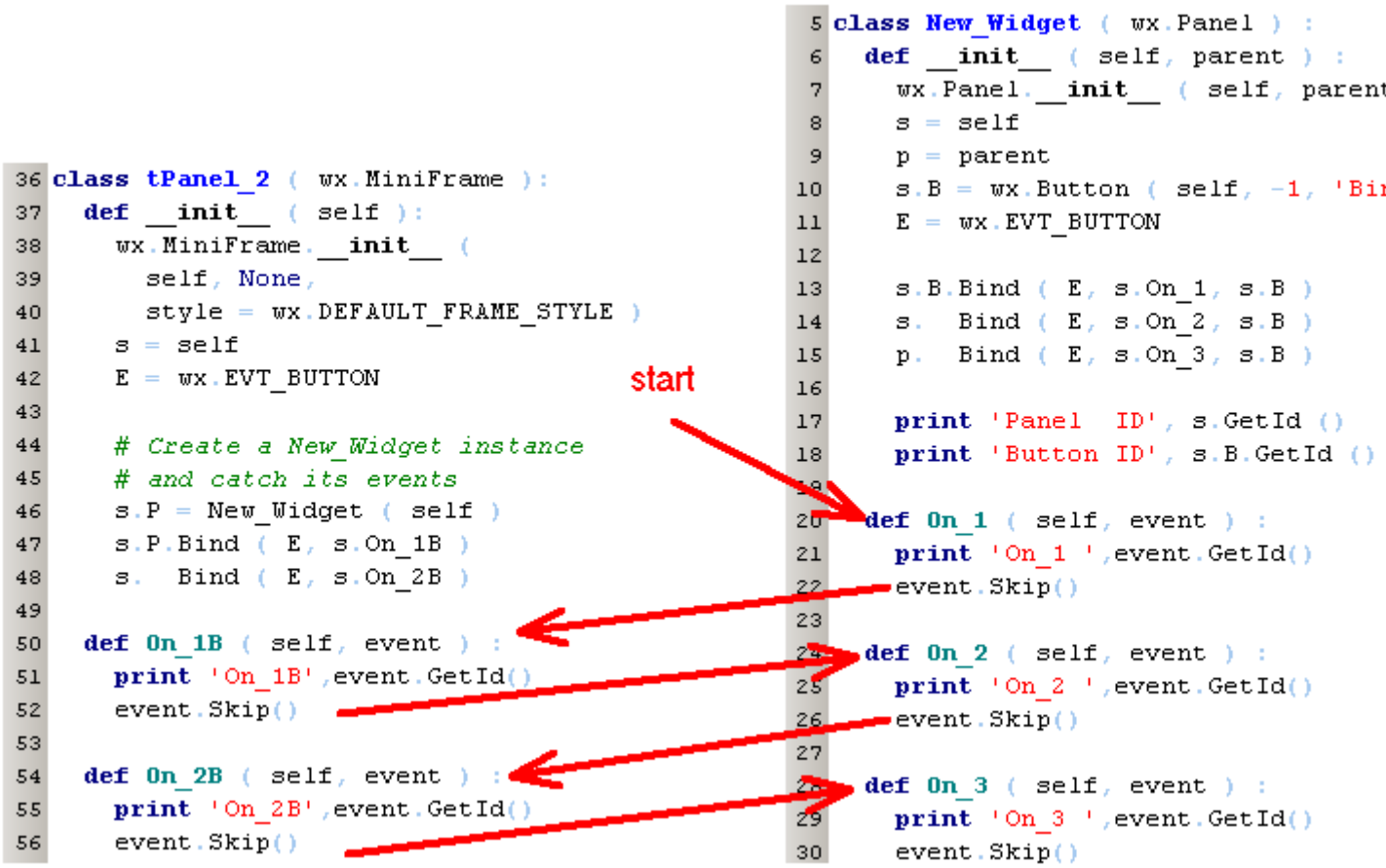


Now I create an application, which contains the new widget and some other widgets. And because that application is created dynamically by another application, I need a general way to handle events, so I just have one event handler in my main program for all events. The program distinguishes between the different sources by maintaining a table of object-ID's. The problem was that although all normal control events were triggered and handled well, the event of the new widget wasn't handled at all.



Knowing the cause of my problem, I also know that the problem has nothing to do with the normal controls, so I leave button B2 out in the further discussion. Let's first look to the path along which the event is evaluated: on the left side you see the code of the upper hierarchical frame, on the right the code of the

new widget.




```
36 class tPanel_2 ( wx.MiniFrame ):
37   def __init__ ( self ):
38     wx.MiniFrame.__init__ (
39       self, None,
40       style = wx.DEFAULT_FRAME_STYLE )
41     s = self
42     E = wx.EVT_BUTTON
43
44     # Create a New_Widget instance
45     # and catch its events
46     s.P = New_Widget ( self )
47     s.P.Bind ( E, s.On_1B )
48     s.   Bind ( E, s.On_2B )
49
50   def On_1B ( self, event ) :
51     print 'On_1B',event.GetId()
52     event.Skip()
53
54   def On_2B ( self, event ) :
55     print 'On_2B',event.GetId()
56     event.Skip()
```

```
5 class New_Widget ( wx.Panel ) :
6   def __init__ ( self, parent ) :
7     wx.Panel.__init__ ( self, parent
8     s = self
9     p = parent
10    s.B = wx.Button ( self, -1, 'Bi
11    E = wx.EVT_BUTTON
12
13    s.B.Bind ( E, s.On_1, s.B )
14    s.   Bind ( E, s.On_2, s.B )
15    p.   Bind ( E, s.On_3, s.B )
16
17    print 'Panel  ID', s.GetId ()
18    print 'Button ID', s.B.GetId ()
19
20  def On_1 ( self, event ) :
21    print 'On_1 ',event.GetId()
22    event.Skip()
23
24  def On_2 ( self, event ) :
25    print 'On_2 ',event.GetId()
26    event.Skip()
27
28  def On_3 ( self, event ) :
29    print 'On_3 ',event.GetId()
30    event.Skip()
```

Leaving out one of the bindings, doesn't break the chain, but just makes it shorter, some examples:

All bindings	Commenting line 13	Commenting line 47	Commenting line 13, 47
On_1 -203 On_1B -203 On_2 -203 On_2B -203 On_3 -203	On_1B -203 On_2 -203 On_2B -203 On_3 -203	On_1 -203 On_2 -203 On_2B -203 On_3 -203	On_2 -203 On_2B -203 On_3 -203

Leaving out one of the event.Skip() calls, does end the chain, some examples :

All bindings	Commenting line 26	Commenting line 52	
On_1 -203 On_1B -203 On_2 -203 On_2B -203 On_3 -203	On_1 -203 On_1B -203 On_2 -203	On_1 -203 On_1B -203	

The problem

Now I always thought that specifying the event-source was a very good idea. So let's change line 47, and specify the event-source:

```
46      s.P = New_Widget ( self )
47      s.P.Bind ( E, s.On_1B, s.P )
```

```
On_1   -203
On_2   -203
On_2B  -203
On_3   -203
```

We see that the event handler "On_1B" is never called !! Why ??
After debugging the program, the answer is very simple and (after all very) obvious: the ID of the New_Widget is not the ID of the Button.

```
Panel  ID -202
Button ID -203
```

Now you can probably say, that the New_Widget is a lousy solution for creating new widgets, but it's a very easy one (certainly for non-programmers). Because PyLab_Works should be as easy as possible and hide as many as possible details from the core, we accept this as perfect valid solution.

The Solution

The solution (or work around) consists of two extensions, one in the core of PyLab_Works and the other into a small extension of the new widget:

Core Extension

```
75    if 'Get_ID' in dir ( Control ) :
76      Control_Pars ['ID'] = Control.Get_ID ()
77    else:
78      Control_Pars ['ID'] = Control.GetId()
```

New Widget Extension

```
32    def Get_ID ( self ):
33      return self.B.GetId()
```

Conclusion:

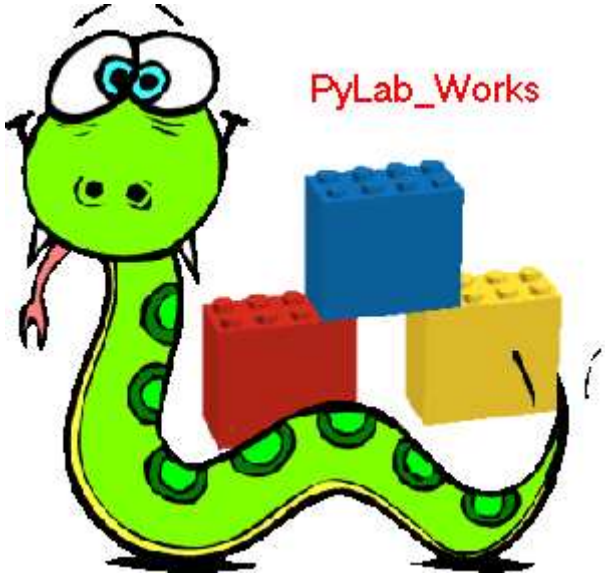
These conclusions might be too simplistic in general cases, but they certainly are valid within the PyLab_Works framework.

- By **choosing the right component binding**, you can determine which handler is invoked first, the object itself or his parent
- **Always use event.Skip()**. By not using event.Skip() you can stop the propagation of the event, so in general you should always use event.Skip(), unless you explicitly want to prevent the higher layers to receive the event.
- **Never specify the event-source**, unless you want to bind this specific event-handler only to be used by this source and we are absolutely sure we use the right source-ID.

ToDo  (March 2009)

Application Designer / Domain Expert / Control Designer / **Core Developer**

- Null can't be stored in inifile. This is simple to solve for the single value Null, but difficult for nested values. Probably both pickling and configobj have problems with this. So for we've some workarounds: writing CS_gen, Null is translated into None.


PyLab_Works