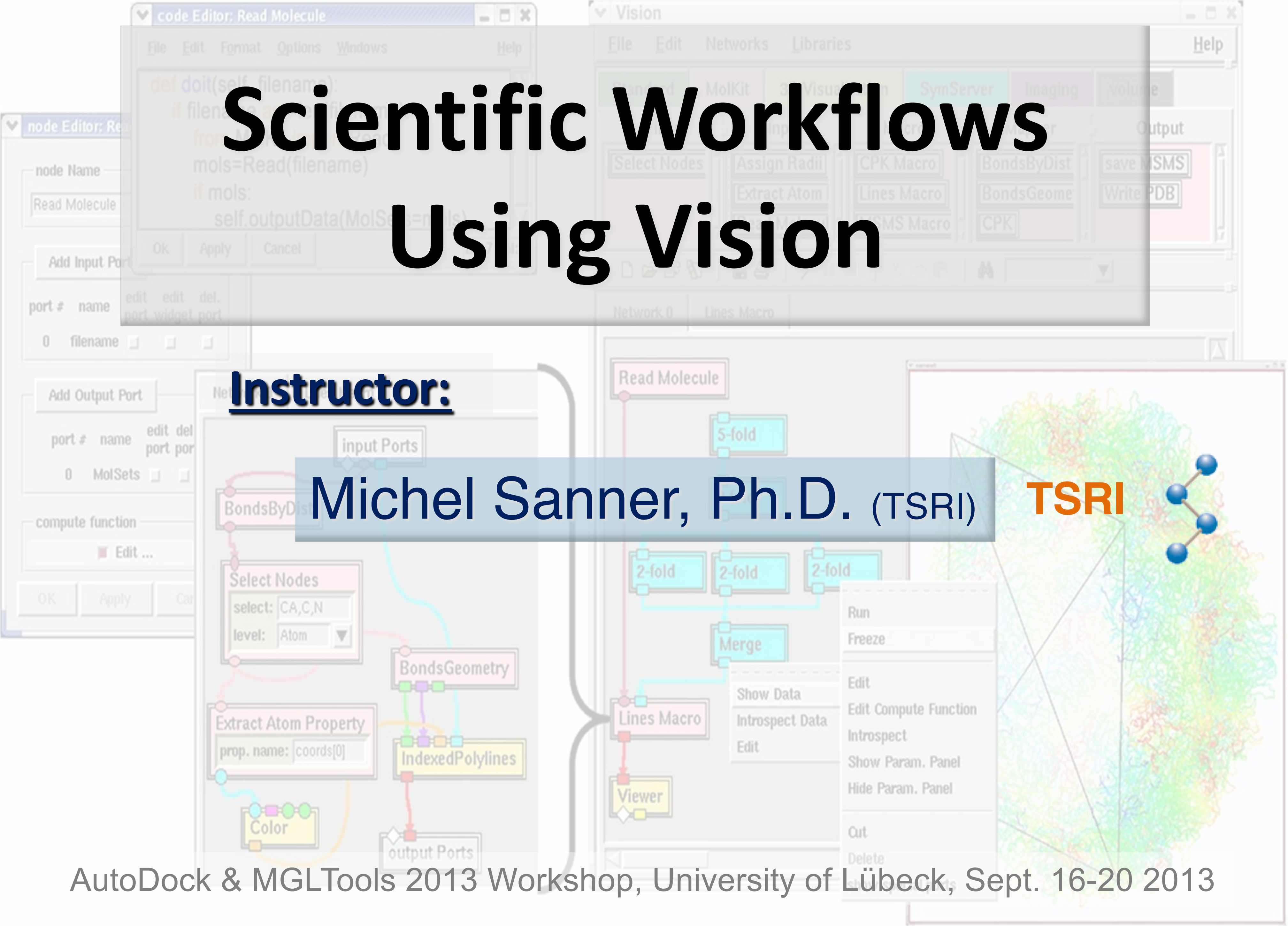
Scientific Workflows Using Vision

Instructor:

Michel Sanner, Ph.D. (TSRI)

TSRI



AutoDock & MGLTools 2013 Workshop, University of Lübeck, Sept. 16-20 2013

Overview

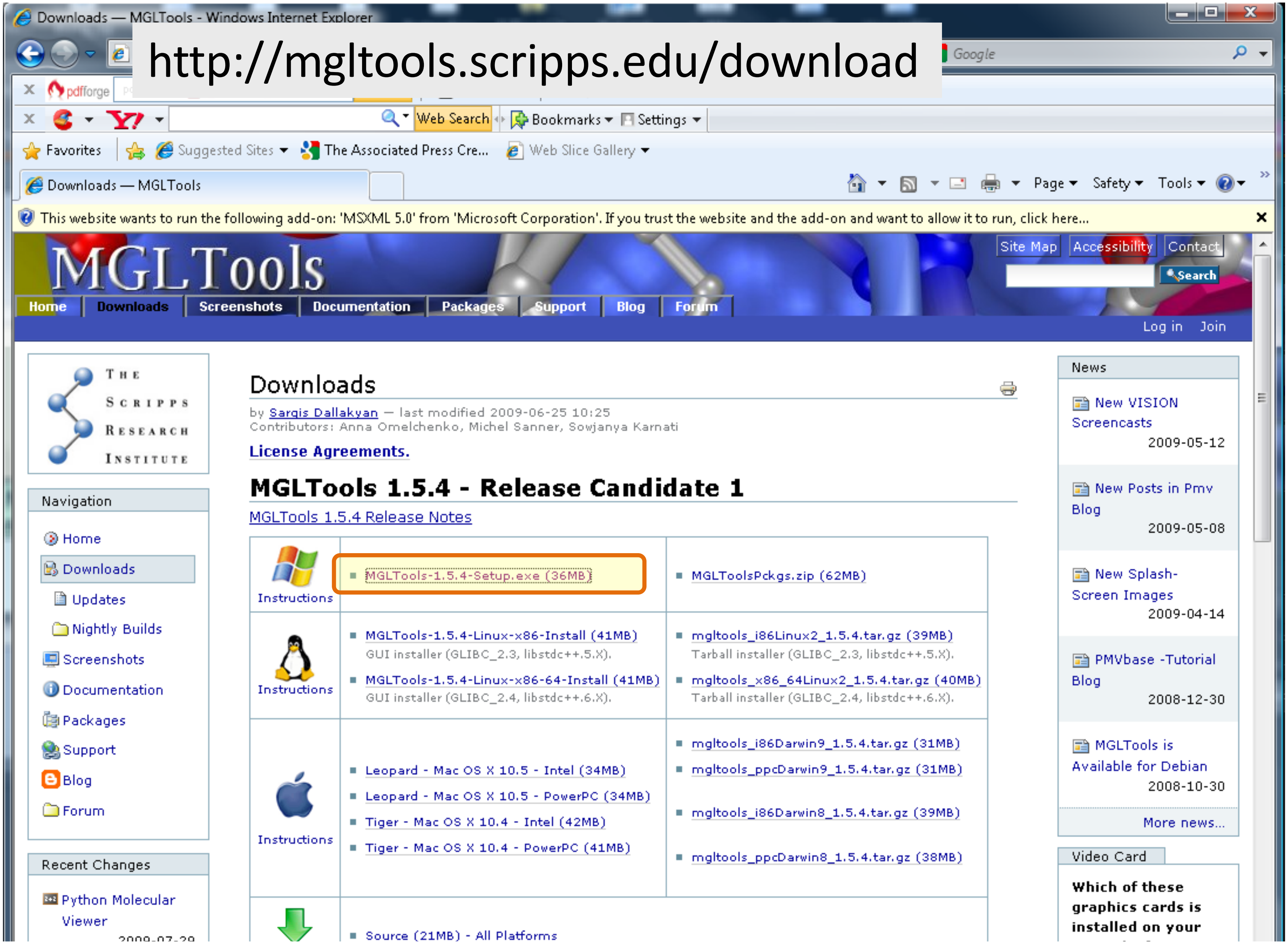
- Part 1: Introduction to Vision (M. Sanner)
  - Basic interactions, basic networks, ImageViewer
- Part 2: Building an application (M. Sanner)
  - Writing nodes, User Library, User panels, moving widgets, noGUI execution
- Part 3: Extending Pmv (M. Sanner)
  - Extending Pmv using Vision, Icosahedral capsid
- Part 4: Advanced topics (M. Sanner)
  - Volume, Student applications, wrap-up

Part 1 Overview

- Installing and starting Vision
- Interacting with the Vision GUI
  - GUI elements
  - Mouse bindings
- Building Networks
  - Image Viewer
  - . . .

Installing Vision

http://mgltools.scripps.edu/download



This website wants to run the following add-on: 'MSXML 5.0' from 'Microsoft Corporation'. If you trust the website and the add-on and want to allow it to run, click here...

Navigation

- Home
- Downloads
- Updates
- Nightly Builds
- Screenshots
- Documentation
- Packages
- Support
- Blog
- Forum

Recent Changes

- Python Molecular Viewer 2009-07-29

Downloads

by Sargis Dallakyan — last modified 2009-06-25 10:25
Contributors: Anna Omelchenko, Michel Sanner, Sowjanya Karnati

License Agreements.

MGLTools 1.5.4 - Release Candidate 1

MGLTools 1.5.4 Release Notes

Instructions	- MGLTools-1.5.4-Setup.exe (36MB)	- MGLToolsPckgs.zip (62MB)
Instructions	- MGLTools-1.5.4-Linux-x86-Install (41MB) GUI installer (GLIBC_2.3, libstdc++.5.X). - MGLTools-1.5.4-Linux-x86-64-Install (41MB) GUI installer (GLIBC_2.4, libstdc++.6.X).	- mgltools_i86Linux2_1.5.4.tar.gz (39MB) Tarball installer (GLIBC_2.3, libstdc++.5.X). - mgltools_x86_64Linux2_1.5.4.tar.gz (40MB) Tarball installer (GLIBC_2.4, libstdc++.6.X).
Instructions	- Leopard - Mac OS X 10.5 - Intel (34MB) - Leopard - Mac OS X 10.5 - PowerPC (34MB) - Tiger - Mac OS X 10.4 - Intel (42MB) - Tiger - Mac OS X 10.4 - PowerPC (41MB)	- mgltools_i86Darwin9_1.5.4.tar.gz (31MB) - mgltools_ppcDarwin9_1.5.4.tar.gz (31MB) - mgltools_i86Darwin8_1.5.4.tar.gz (39MB) - mgltools_ppcDarwin8_1.5.4.tar.gz (38MB)
	- Source (21MB) - All Platforms	

News

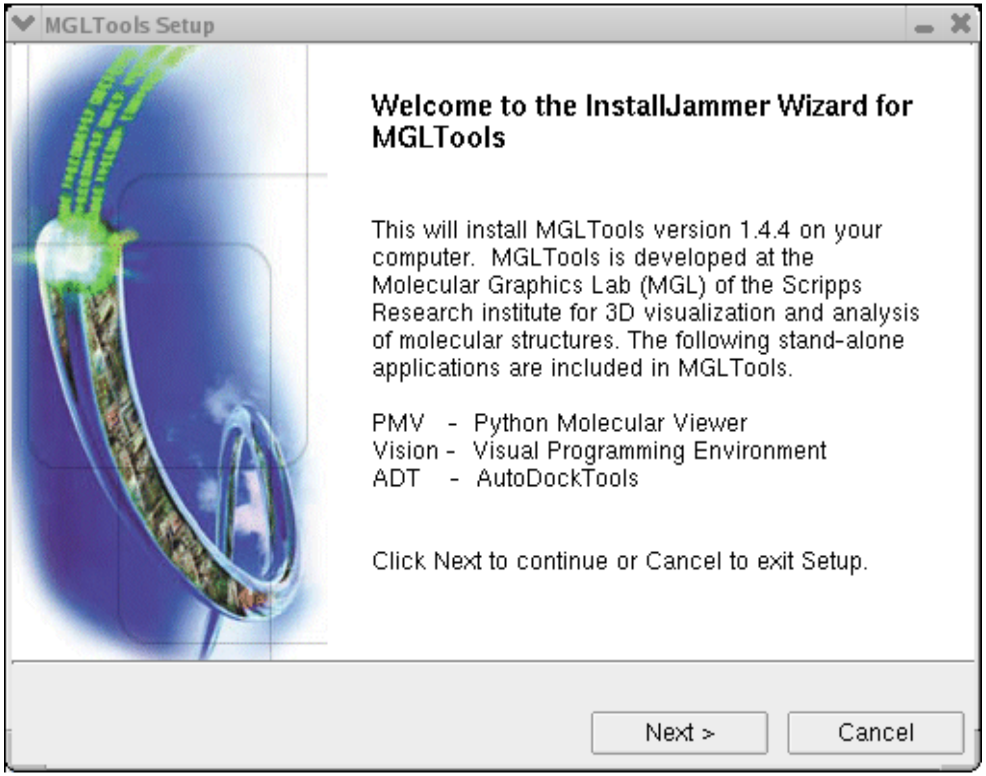
- New VISION Screencasts 2009-05-12
- New Posts in Pmv Blog 2009-05-08
- New Splash-Screen Images 2009-04-14
- PMVbase -Tutorial Blog 2008-12-30
- MGLTools is Available for Debian 2008-10-30

More news...

Video Card

Which of these graphics cards is installed on your

Installing Vision

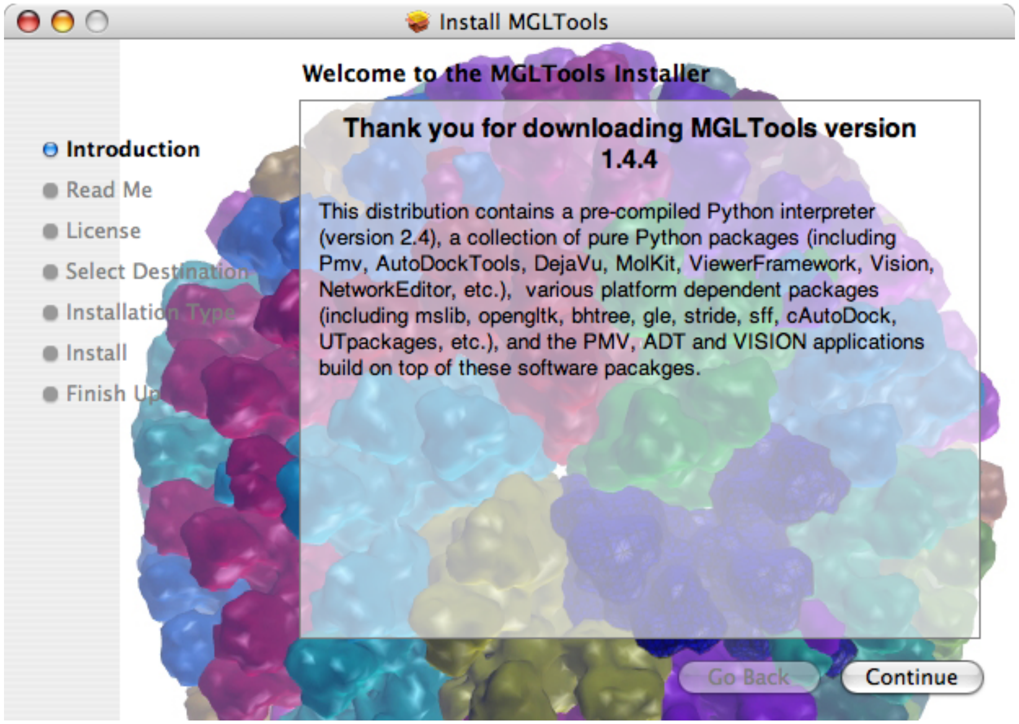


Welcome to the InstallJammer Wizard for MGLTools

This will install MGLTools version 1.4.4 on your computer. MGLTools is developed at the Molecular Graphics Lab (MGL) of the Scripps Research institute for 3D visualization and analysis of molecular structures. The following stand-alone applications are included in MGLTools.

PMV - Python Molecular Viewer
Vision - Visual Programming Environment
ADT - AutoDockTools

Click Next to continue or Cancel to exit Setup.



Welcome to the MGLTools Installer

Thank you for downloading MGLTools version 1.4.4

This distribution contains a pre-compiled Python interpreter (version 2.4), a collection of pure Python packages (including Pmv, AutoDockTools, DejaVu, MolKit, ViewerFramework, Vision, NetworkEditor, etc.), various platform dependent packages (including mslib, opengltk, bhtree, gle, stride, sff, cAutoDock, UTpackages, etc.), and the PMV, ADT and VISION applications build on top of these software pacakges.

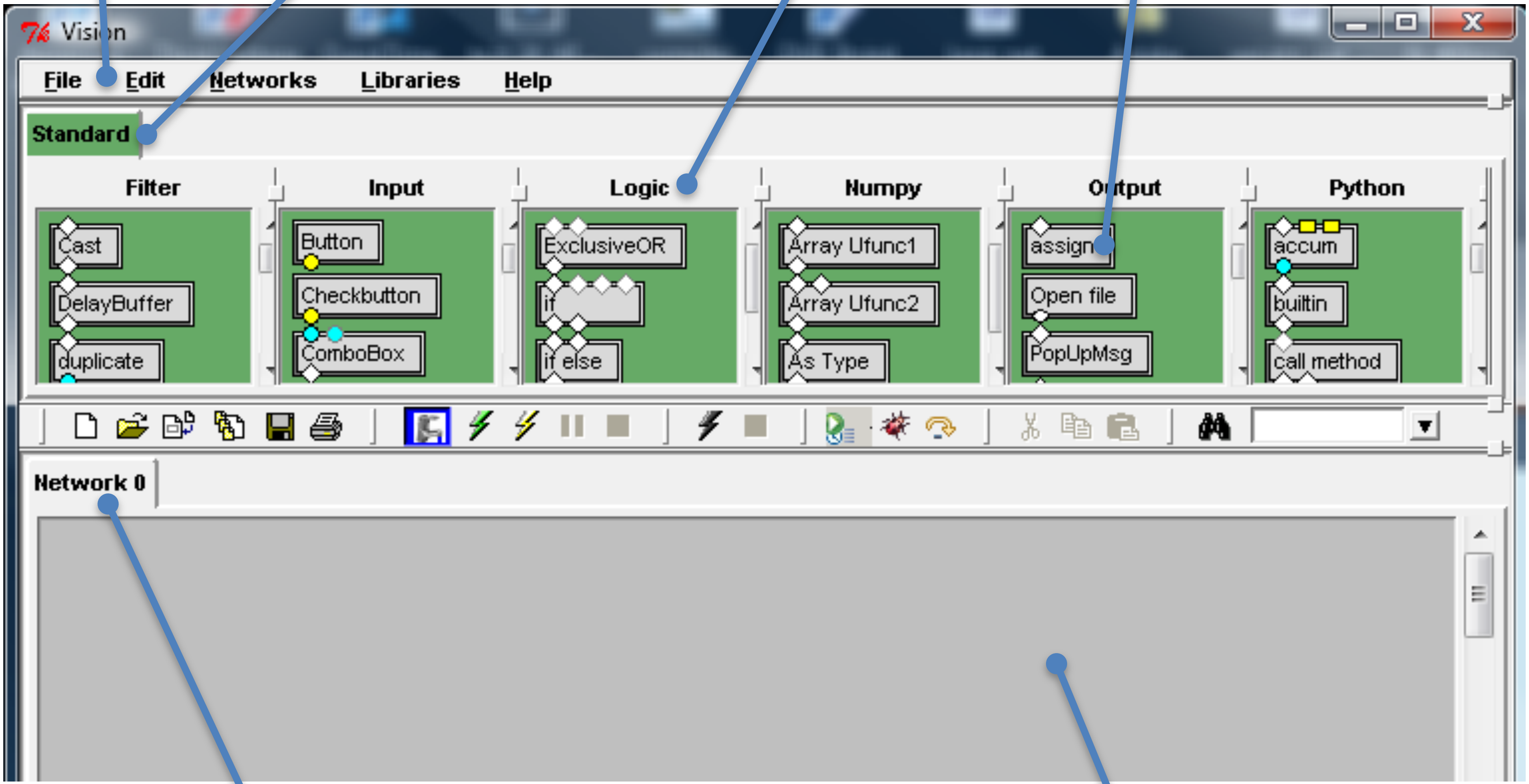
Vision: GUI elements

Menus

Node Library

Node Category

Node

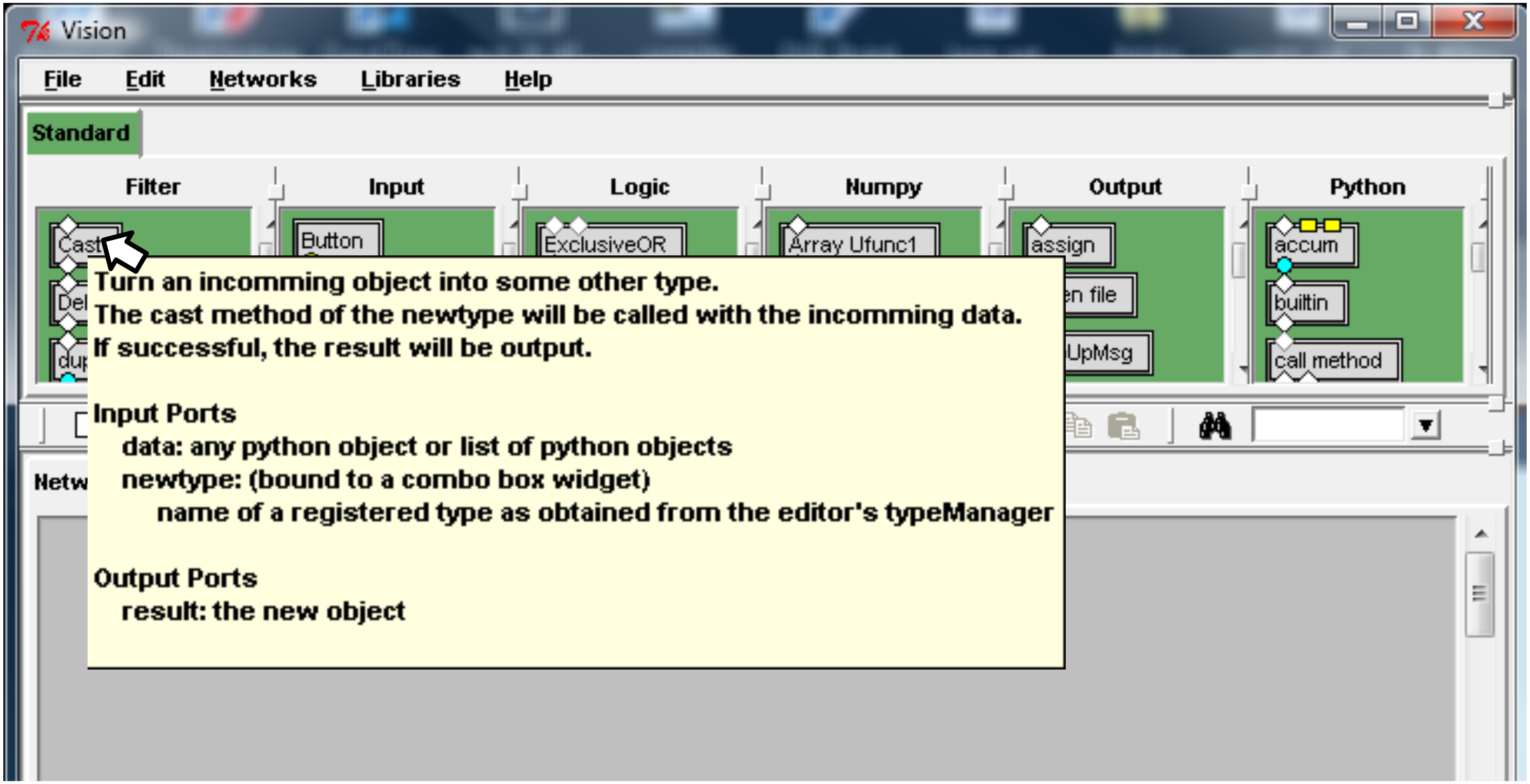


Current network name Network 0

Programming Canvas for Network 0

Node Documentation

NOTE Node documentation displays as tooltips

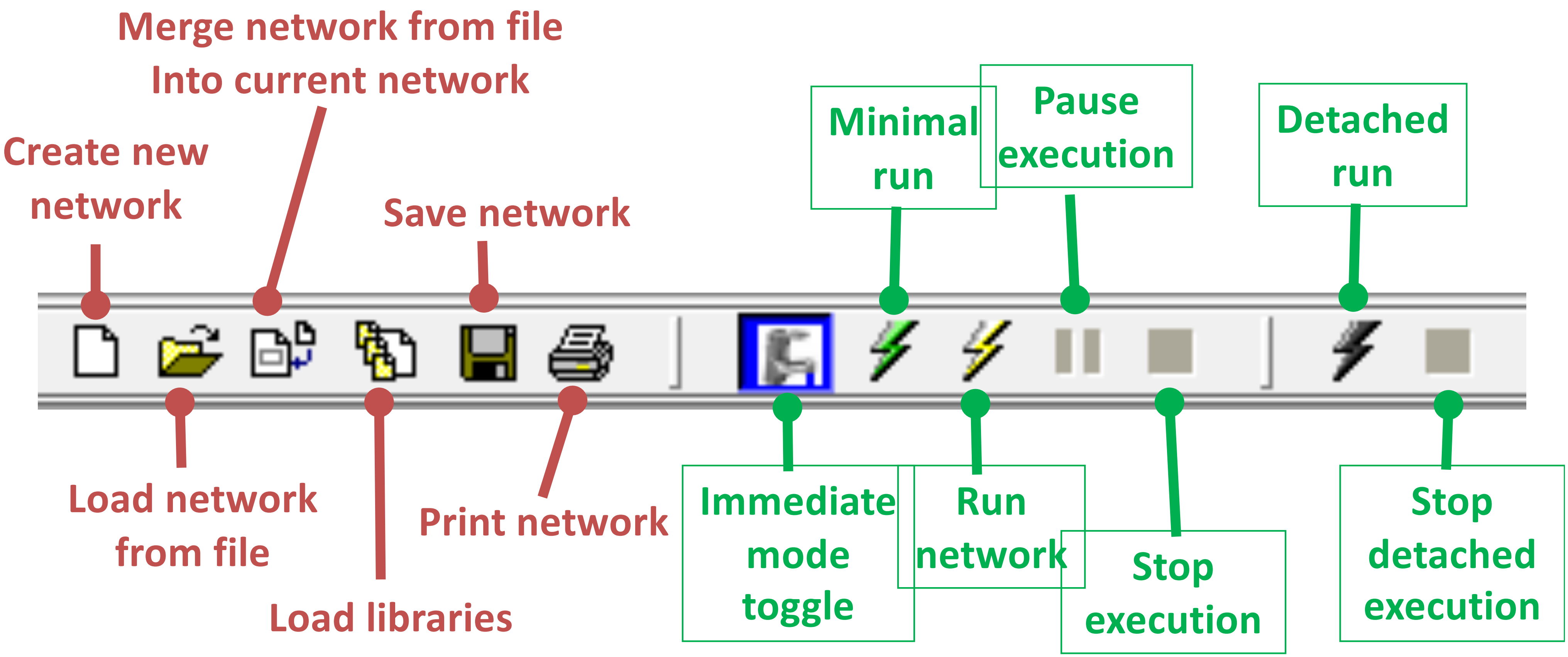


Turn an incomming object into some other type.
The cast method of the newtype will be called with the incomming data.
If successful, the result will be output.

Input Ports
  data: any python object or list of python objects
  newtype: (bound to a combo box widget)
    name of a registered type as obtained from the editor's typeManager

Output Ports
  result: the new object

Vision: GUI elements



Create new network

Merge network from file Into current network

Save network

Load network from file

Load libraries

Print network

Minimal run

Pause execution

Detached run

Immediate mode toggle

Run network

Stop execution

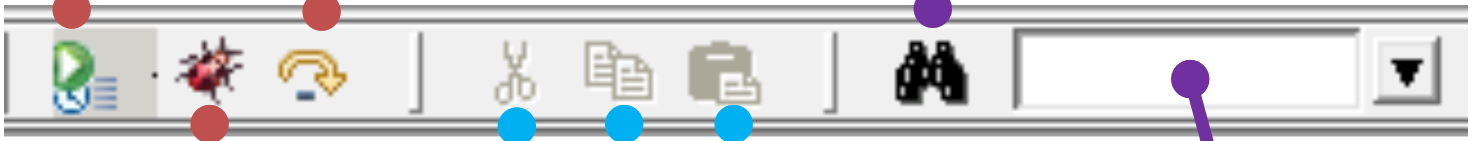
Stop detached execution

Vision: GUI elements

Show execution
Time GANTT chart

Step to
next node

Display advanced
Search interface

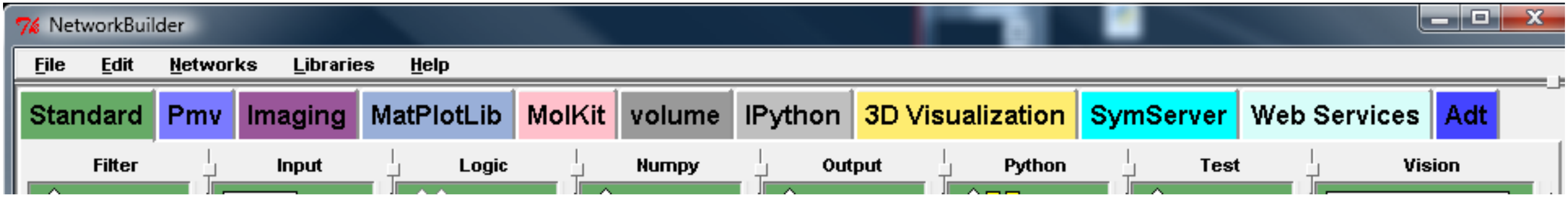


Run new network
In debug mode

Cut, Copy, Paste
Selected nodes

Search nodes by name
in loaded libraries

Libraries



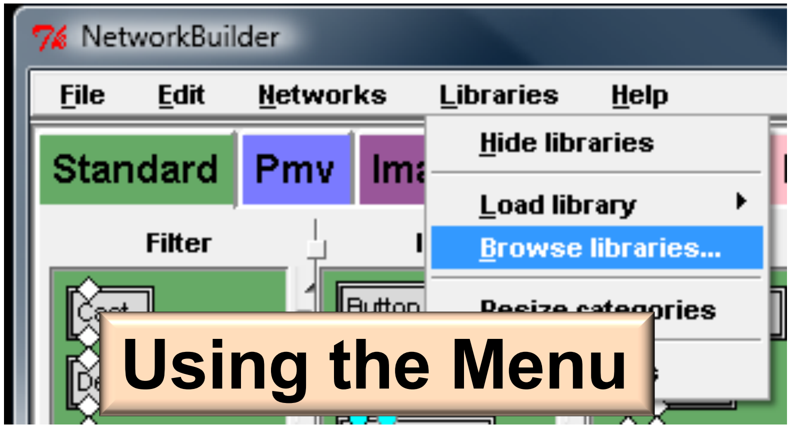
Some Vision node libraries:

- Standard: default Vision node library
- Pmv: Vision interface to Pmv
- Imaging: Interface to the Python Imaging Library (PIL)
- Matplotlib: 2D graphing library
- MolKit: working with biological molecules
- Volume: working with 3D regular grids of scalar data
- Ipython: Vision node for using IPython parallel computing

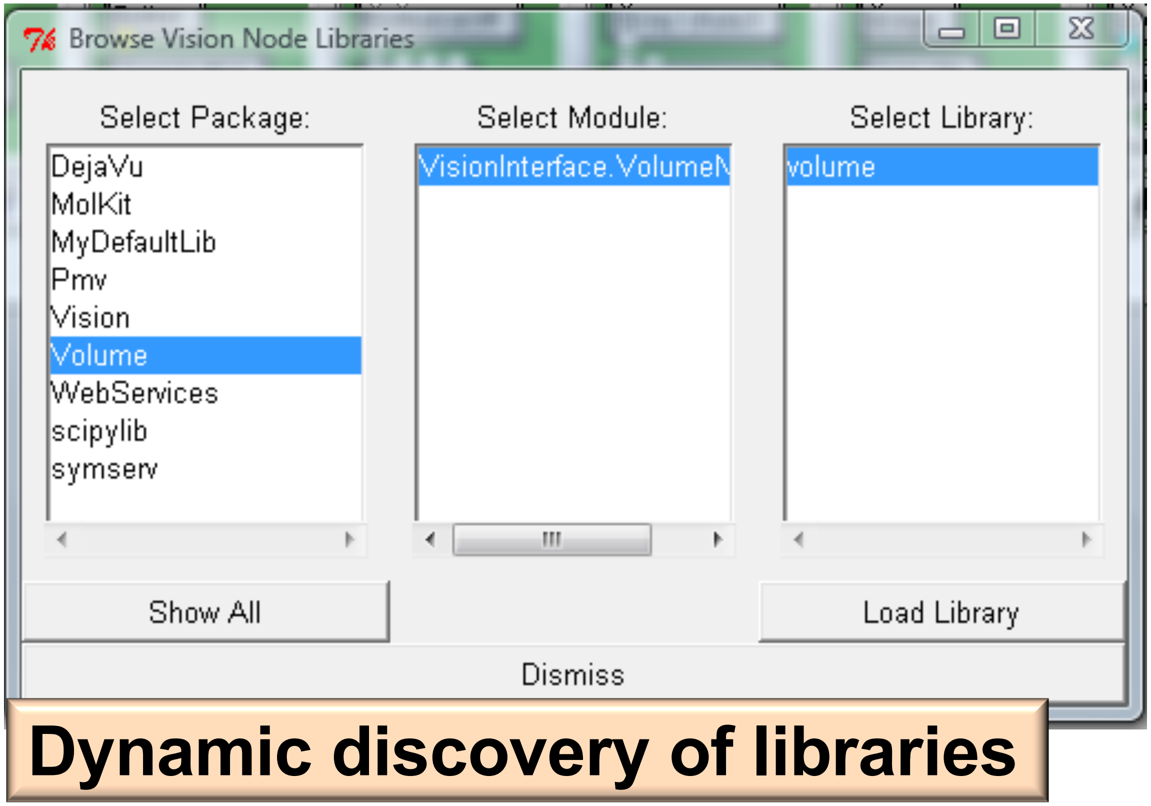
-3D Visualization: Vision interface to DejaVu

- Symserv: working with point symmetry operators
- Web services: Vision nodes for Opal web services servers
- Adt: Vision interface to AutoDockTools (ADT)

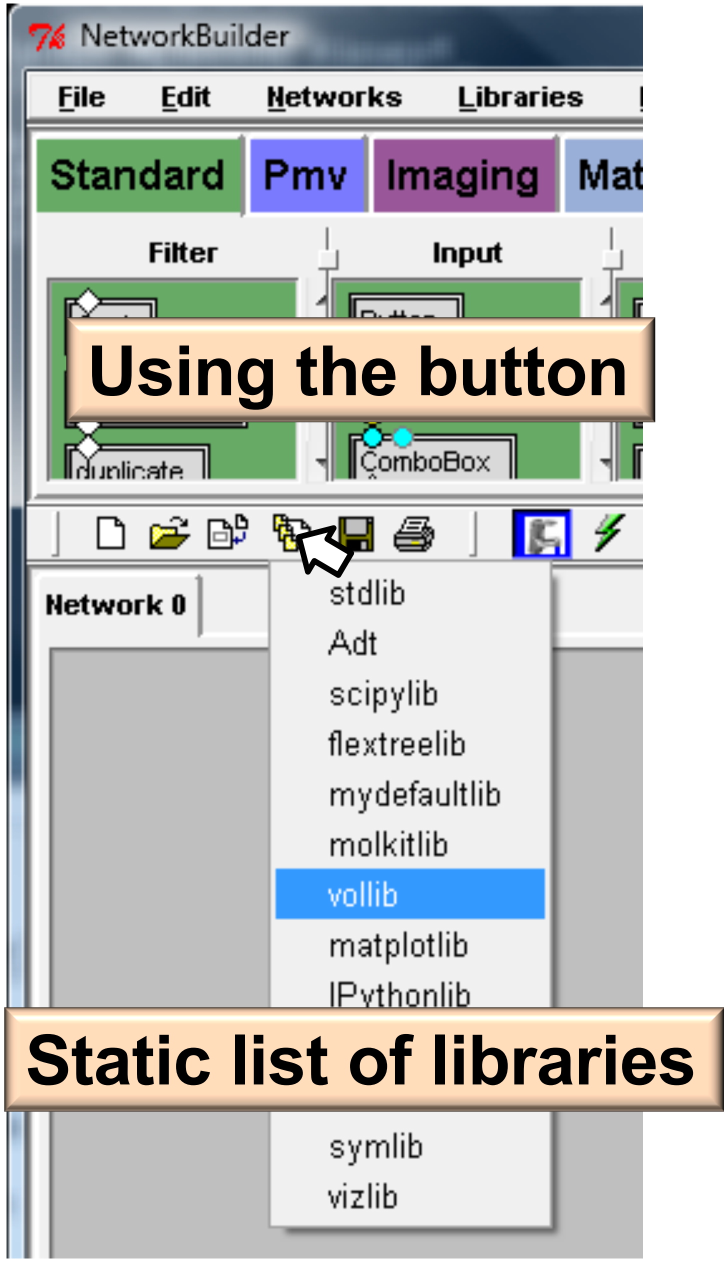
Loading Libraries



Using the Menu



Dynamic discovery of libraries



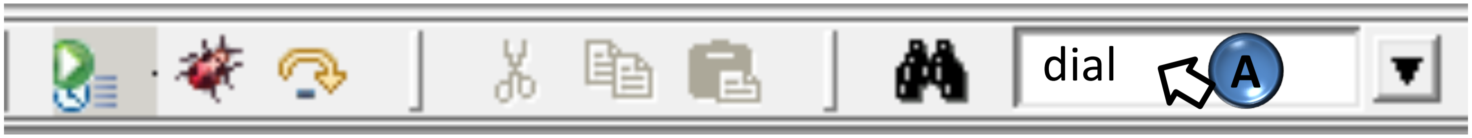
Using the button

Static list of libraries

Exercise: locate a node by name

Task: Find the Dial node in the Standard library

Solution:
1 – left click in the search box (A)
2 – type dial <enter>.
notice how the Dial node is temporarily highlighted in the library
if you missed it, type <enter again>

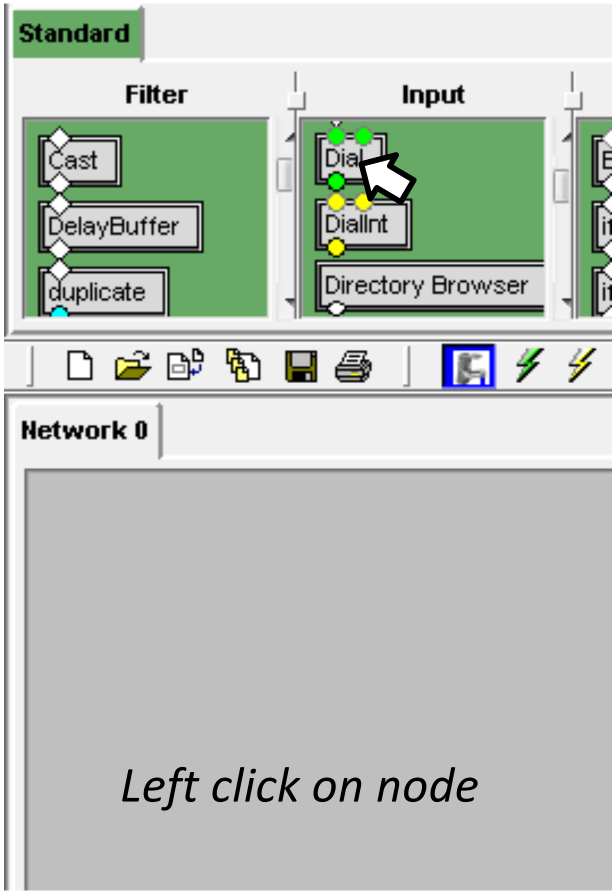


Alternative solution:
1 – left click in the Input category of the Standard Node library
2 – type the letter "d"
notice how the category scrolls to show the Dial node

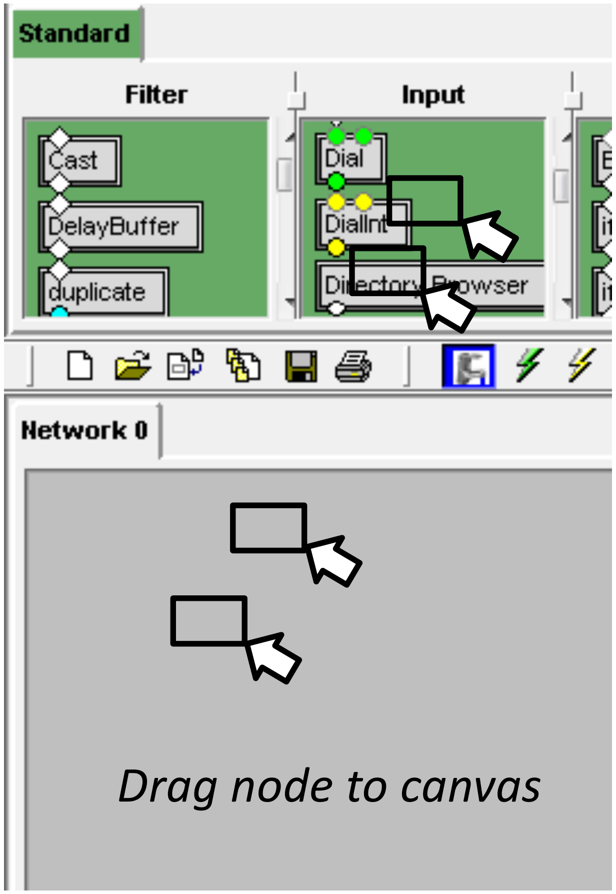
Exercise: drag and drop a node

Task: Drag and drop a Dial node on the programming canvas

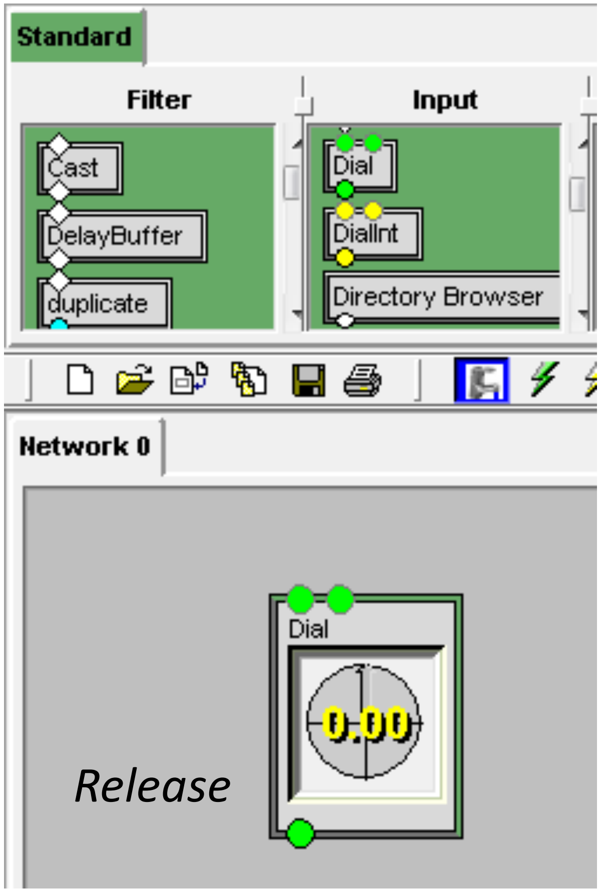
1 – left click on the Dial node in the Input category of the Standard library
2 – drag the mouse to the canvas (without releasing the button)
3 – release the button where you want to place the node



Left click on node

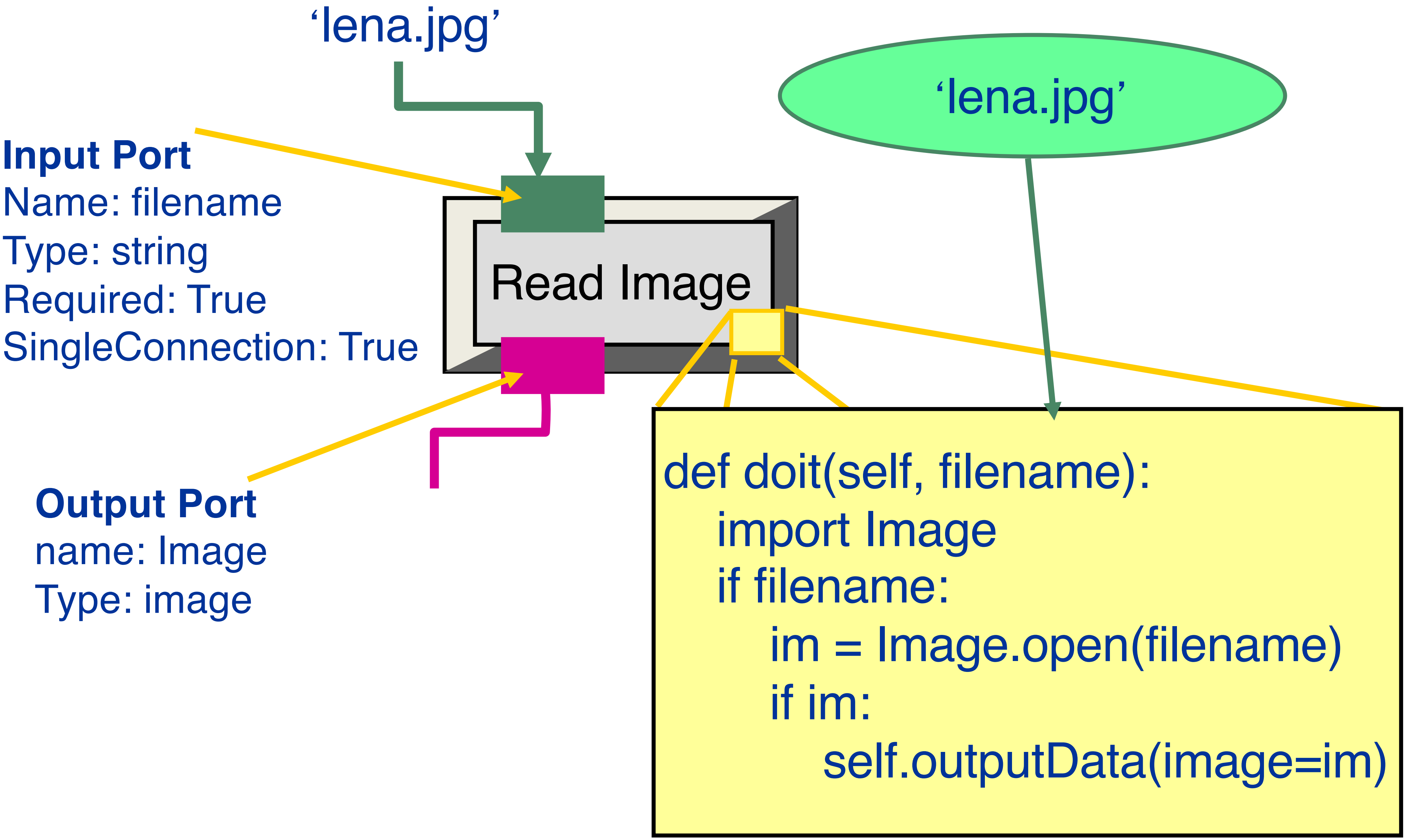


Drag node to canvas



Release

Vision Node



‘lena.jpg’

‘lena.jpg’

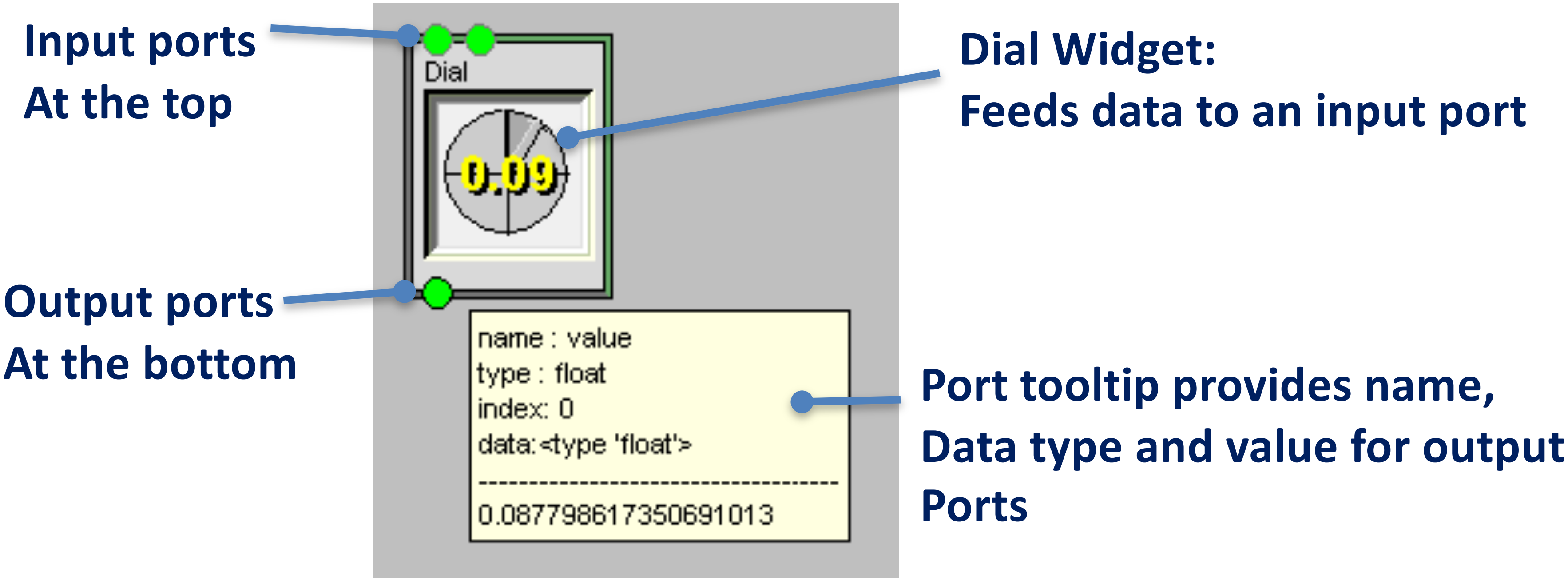
Input Port
Name: filename
Type: string
Required: True
SingleConnection: True

Read Image

Output Port
name: Image
Type: image

```
def doit(self, filename):
    import Image
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
```

Notes



Input ports
At the top

Output ports
At the bottom

Dial Widget:
Feeds data to an input port

```
name : value
type : float
index: 0
data:<type 'float'>
------------------------------------
0.0877986173506910l3
```

Port tooltip provides name,
Data type and value for output
Ports

- Port shape and color indicate data type. Examples of datatypes:
  ⬭ string, 🟡 int, 🟪 image, 🔵 list, 🟨 boolean, …
- Sometimes ports with different data types can be connected

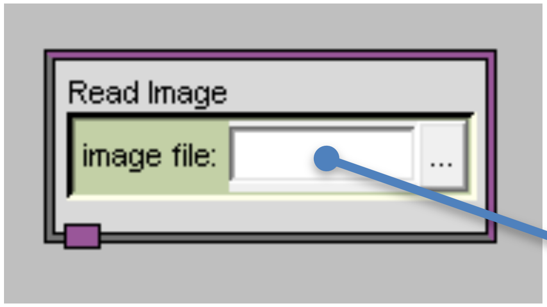
◇ is the None data type, i.e. any Python object is accepted
such ports will "morph" to the incoming data type

- Input ports can be *"required"* or *"optional"*. Valid data is necessary on *"required"* ports for a node to run

Providing Data to Ports

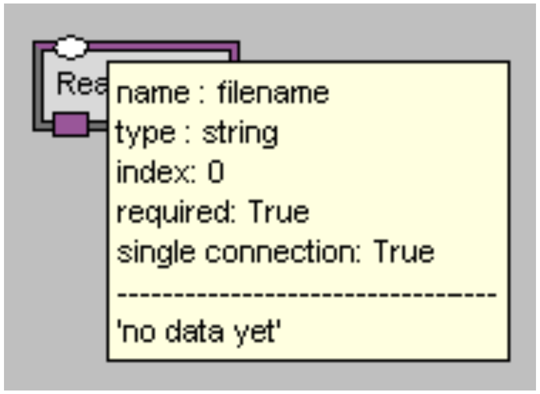
Data can be provided through a connection from an output port of a parent node or from a widget bound to the port.

Read Image node configured to receive the file name from a a file browser widget



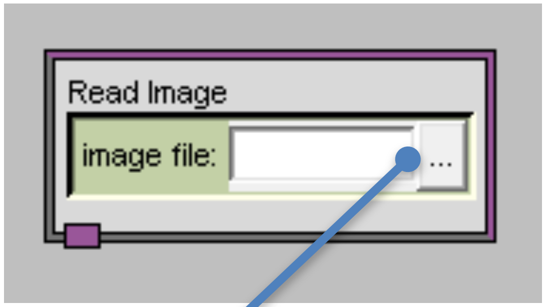
File browser Widget is Bound to the *filename* Input port

Read Image node configured to receive the file name from a parent node

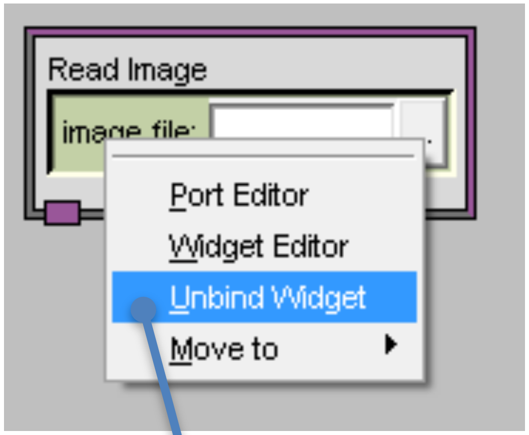


```
name : filename
type : string
index: 0
required: True
single connection: True
--------------------------------
'no data yet'
```

Binding and Unbinding Widgets



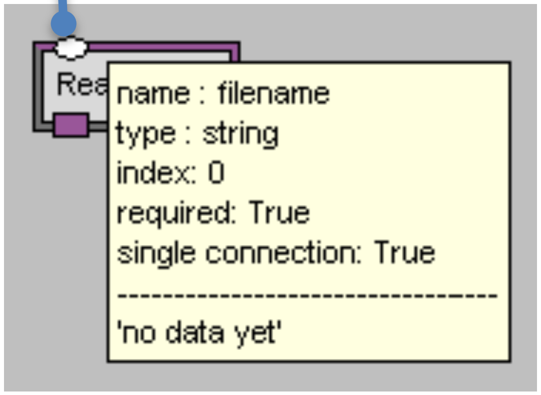
File browser Widget is Bound to the *filename* Input port



Widget can be unbound

- Unbound widget can be rebound
- Widgets are for interactive user input

The *filename* Input port Becomes exposed at the Top of the node, allowing A name computed in the Network to be passed to The node

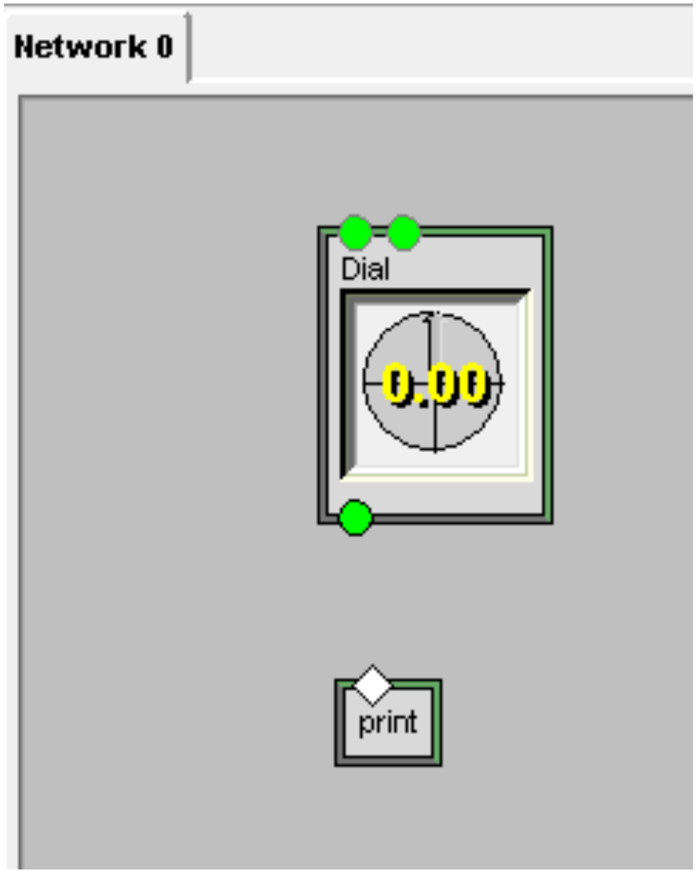


```
name : filename
type : string
index: 0
required: True
single connection: True
--------------------------------
'no data yet'
```

Exercise

Task: Drag and drop a Print node on the programming canvas

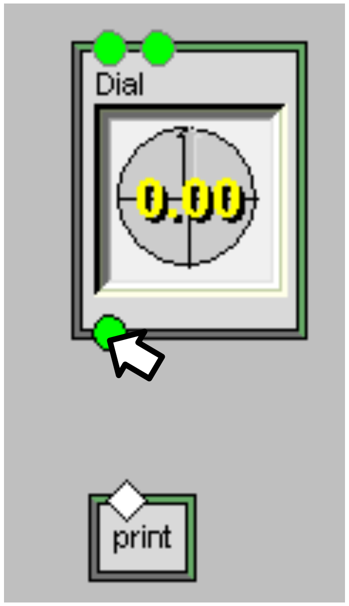
1 – left click on the Output category of the Standard library and type “p”
2 – click on the print Node and drag it to the canvas and release the button



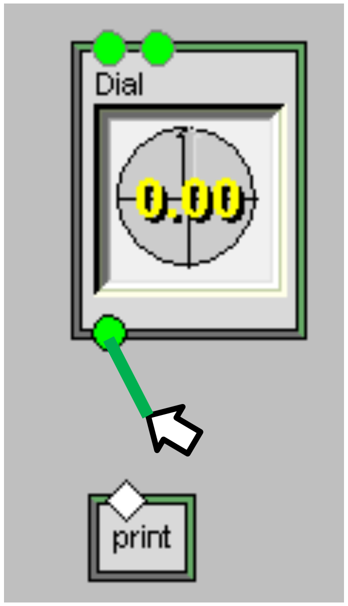
Exercise

Task: Connect the output of Dial to the input port of Print

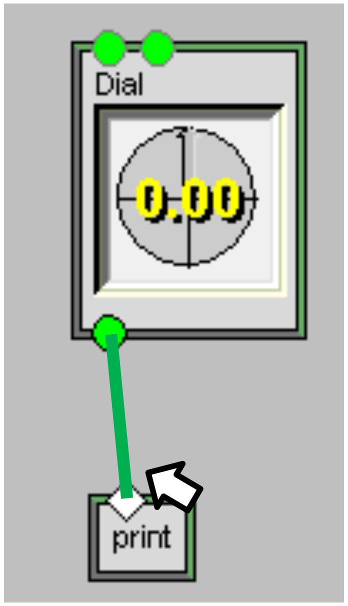
1 – left click on the Output Port of the Dial node (A)
2 – drag the cursor (without releasing mouse button). A green line is drawn.
3 – move cursor close to “print” node’s input port. When you are close the line will snap to the port
4 – release the mouse button to create the connection



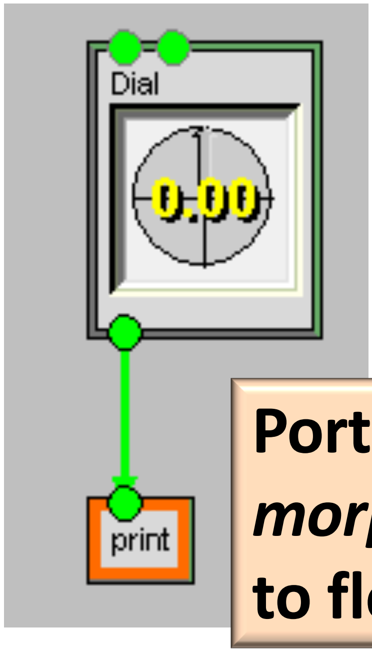
Left click on output port



Drag green line To input port



Green line snaps to input port



Port type *morphed* to float

Release

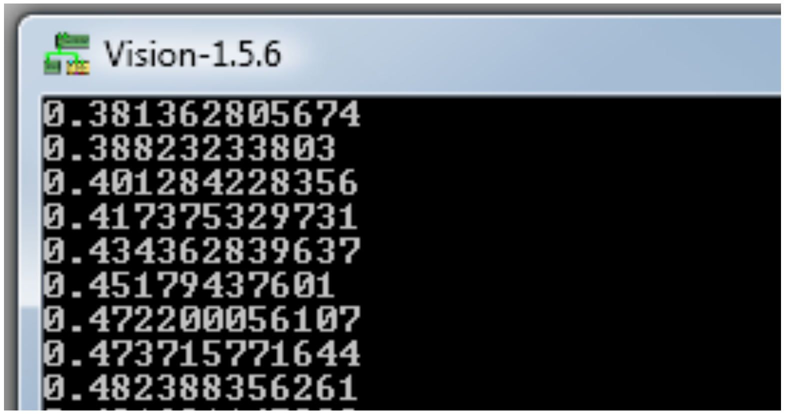
Exercise: modifying parameters

Task: Modify the Dial value

1 – click on the handle and drag the cursor
OR with the cursor over the dial type numbers

Note that:

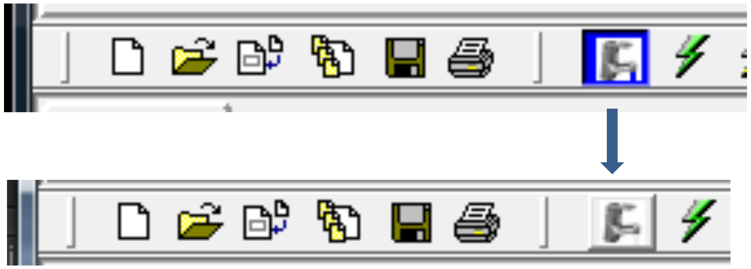
- values are printed to the shell



```
0.381362805674
0.38823233803
0.401284228356
0.417375329731
0.434362839637
0.45179437601
0.472200056107
0.473715771644
0.482388356261
```

Task: Prevent network execution on new data

1 – Toggle the *Toggle Immediate* icon in the toolbar



Modify the Dial value and note that values no longer print to the shell

Notes: node outlines

Node outlines:

- red: running

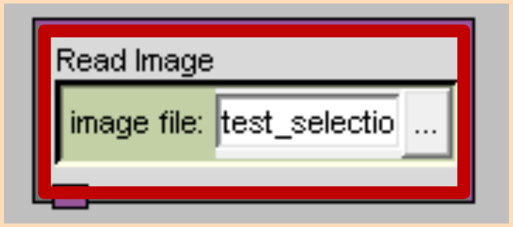
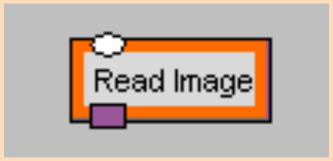


image file: test_selectio ...

- orange: tried to run but missing data



- sick green: node failed

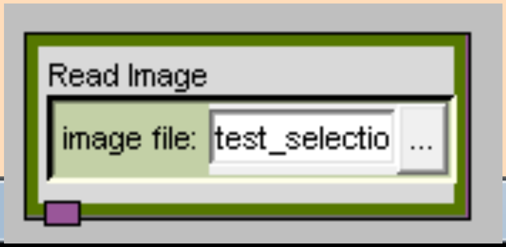
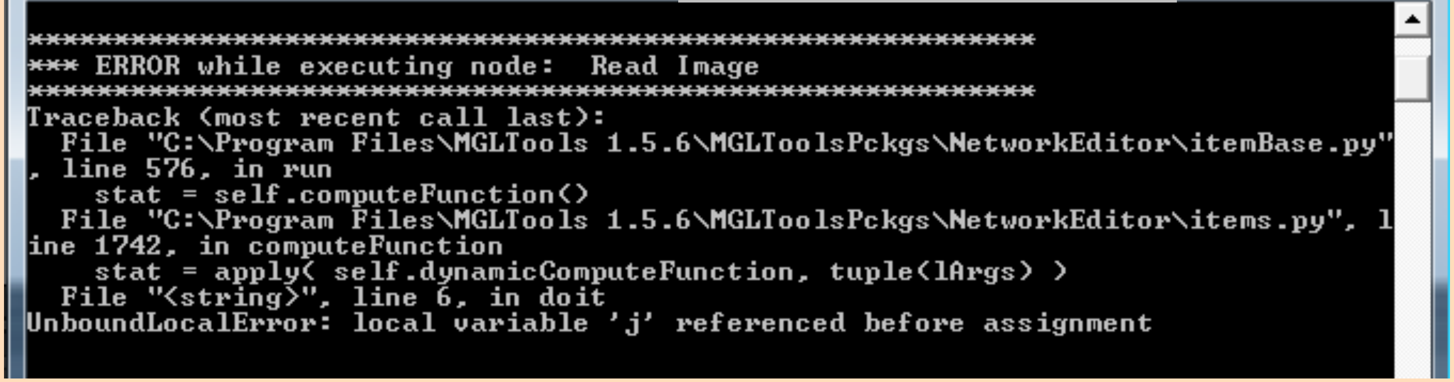


image file: test_selectio ...



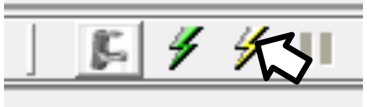
```
***********************************************************
*** ERROR while executing node:  Read Image
***********************************************************
Traceback (most recent call last):
  File "C:\Program Files\MGLTools 1.5.6\MGLToolsPckgs\NetworkEditor\itemBase.py"
, line 576, in run
    stat = self.computeFunction()
  File "C:\Program Files\MGLTools 1.5.6\MGLToolsPckgs\NetworkEditor\items.py", l
ine 1742, in computeFunction
    stat = apply( self.dynamicComputeFunction, tuple(lArgs) )
  File "<string>", line 6, in doit
UnboundLocalError: local variable 'j' referenced before assignment
```

Turn off outlines using Edit-> Flash nodes when run

Exercise: Run Network

Task: Run the whole network, independently of data status

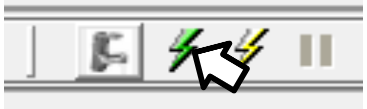
1 – Click on the *Run* button in the tool bar



Note that dial value is printed to the shell each time you run

Task: Make minimal run, i.e. run only nodes that have new input data (and their children)

1 – Click on the *Minimal Run* button in the tool bar

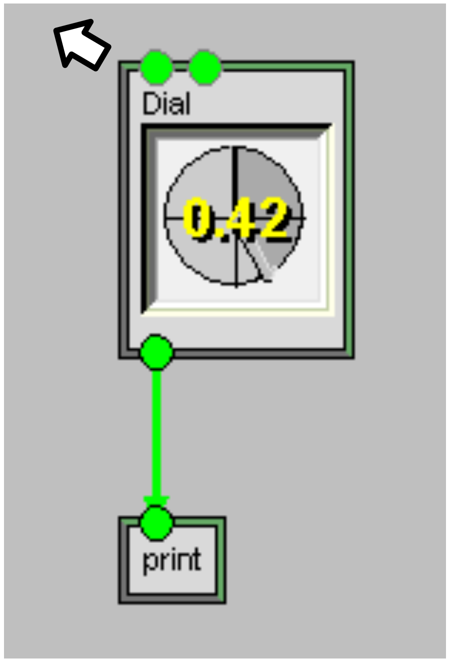


Note that dial value is printed when the dial has a new value but nothing is printed if the value of the dial is unchanged

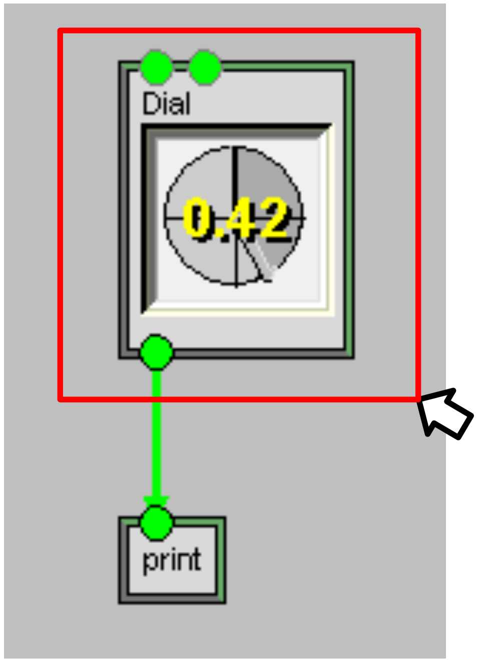
Exercise: selecting

Task: Select the dial node

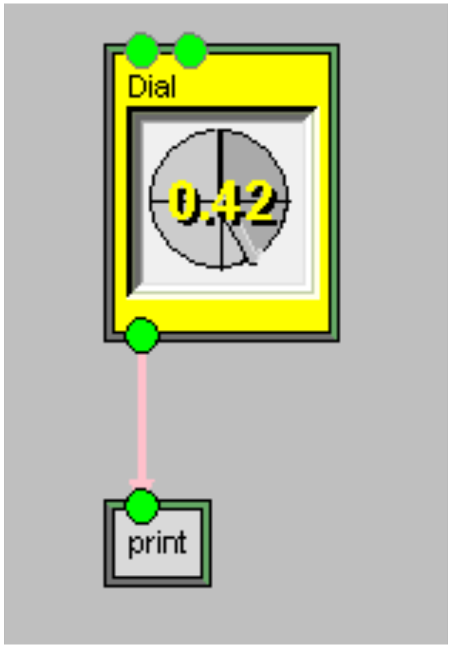
1 – Left click on the canvas background
2 – drag the cursor to draw a box around the Dial node
3 – release the mouse button to toggle nodes between selected and deselected mode



Left click on background



Drag red box around nodes to select



Release to toggle Dial node from deselected to selected

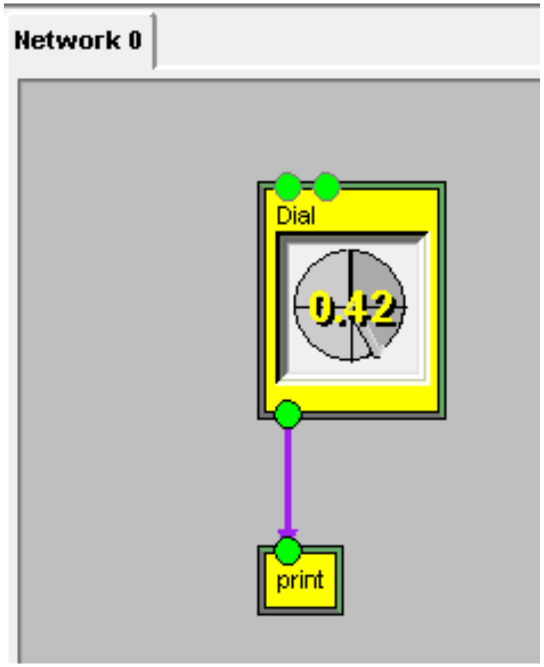
-Left click on background to clear the selection

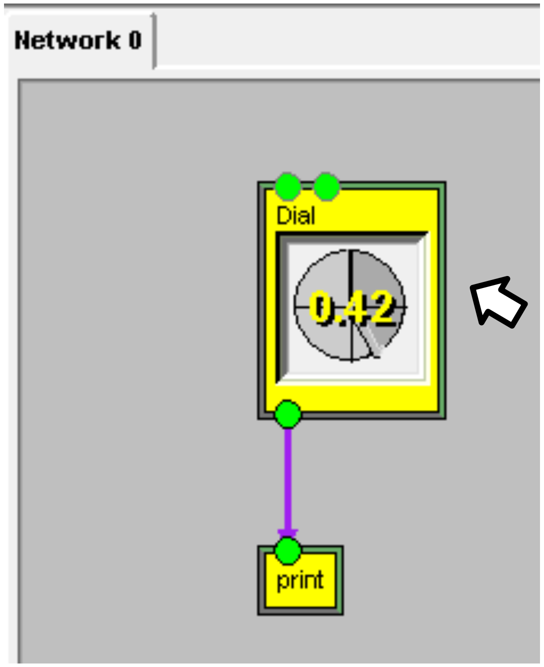
- Ctrl-A selects all

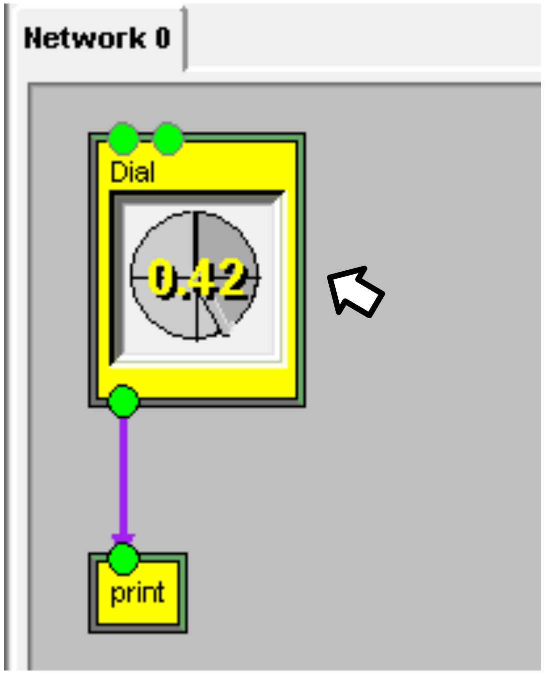
Exercise: move nodes and connections

Task: move the Dial and print nodes together

1 – select nodes to move
2 – middle click on canvas background and drag cursor
3 – release the mouse button to toggle nodes between selected and deselected mode





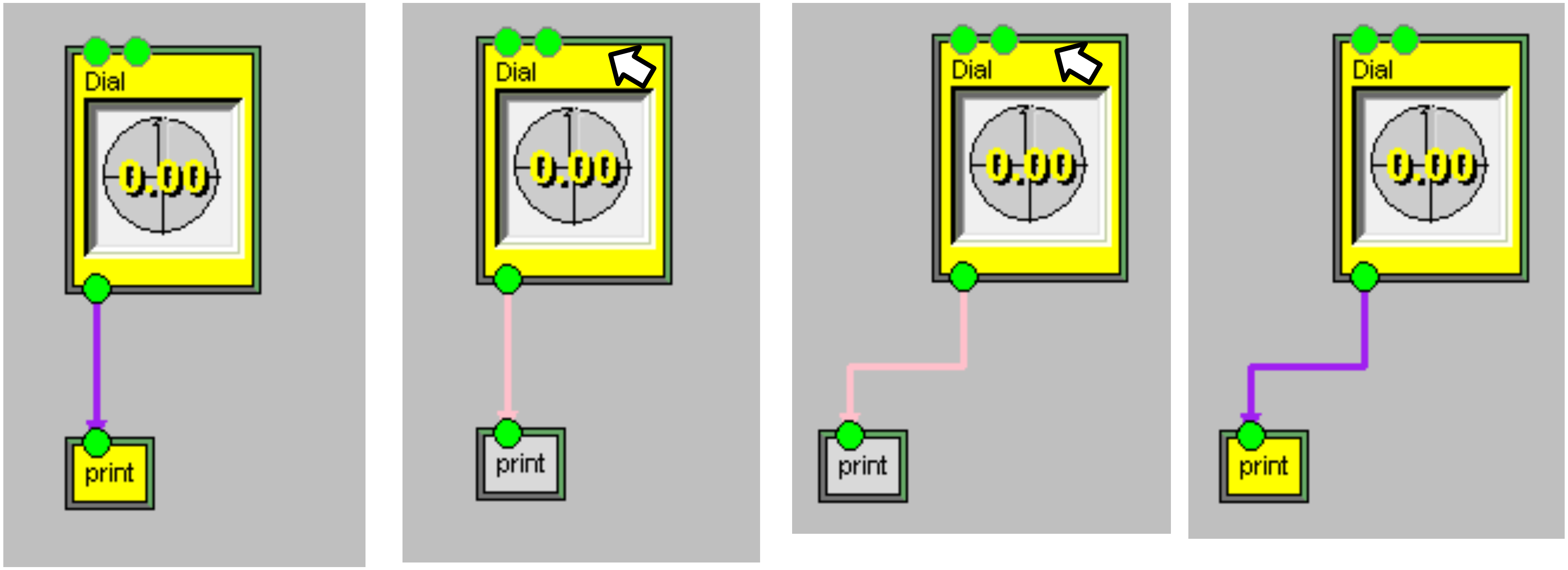


Note: using the middle mouse button on canvas background with no selection scrolls the canvas

Exercise: move a single node

Task: move the Dial node alone

1 – select both nodes
2 – Shift left click on the node background. This node becomes temporarily selected.
3 – drag cursor, the node moves along.
4 – release the mouse button. The previous selection is restored.



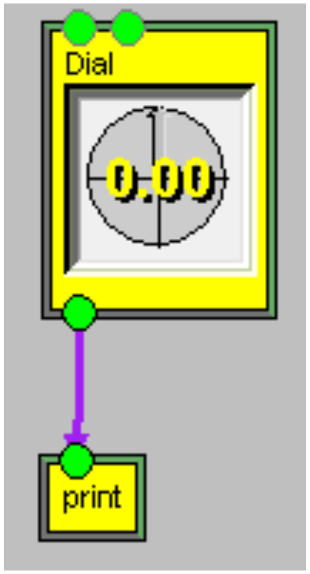
Exercise: copy-paste parts

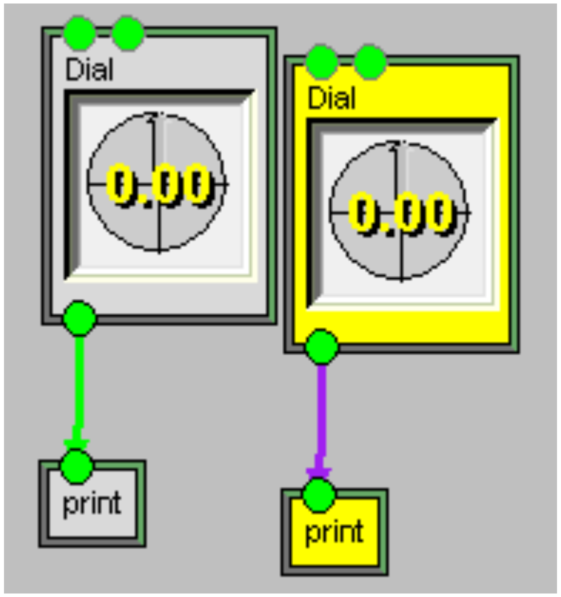
Task: duplicate the dial-print network

1 – select both nodes. Note the copy button in the toolbar becomes active.

2 – click on the copy button. Note the paste button becomes active.

3 – click on the paste button. Note the pasted nodes are now selected and can be moved

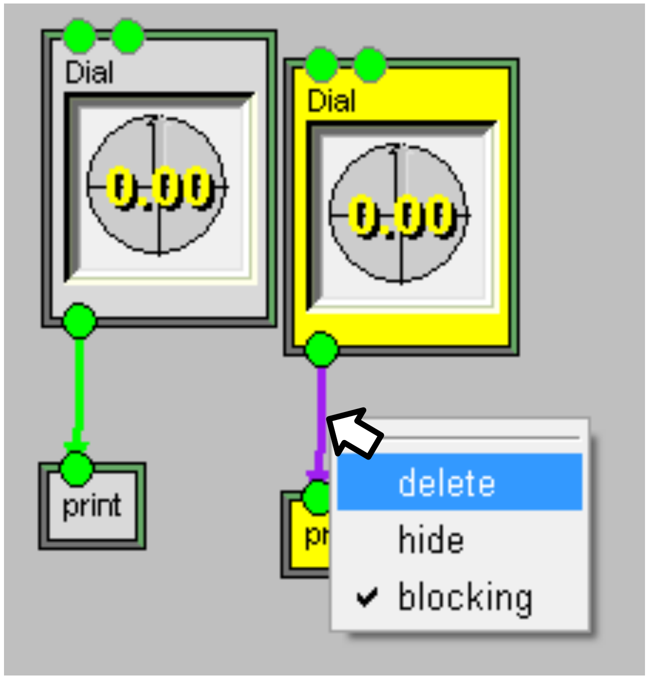


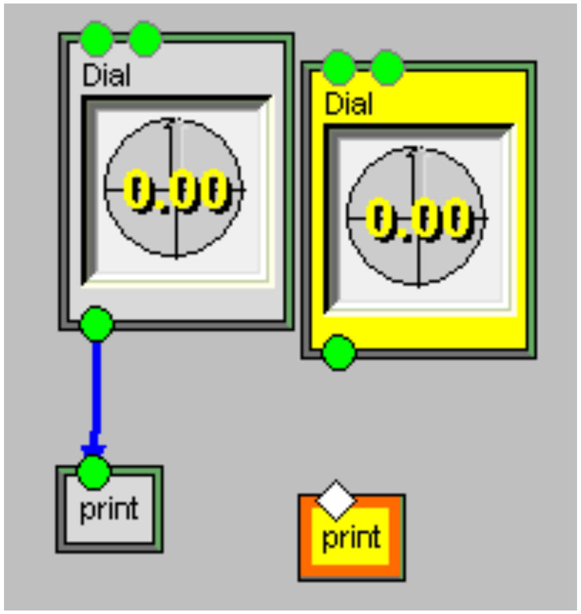


Exercise

Task: delete a connection

1 – right click on the connection between the pasted Dial and print nodes
2 – choose delete in the menu





Exercise: deleting

Task: delete a single node

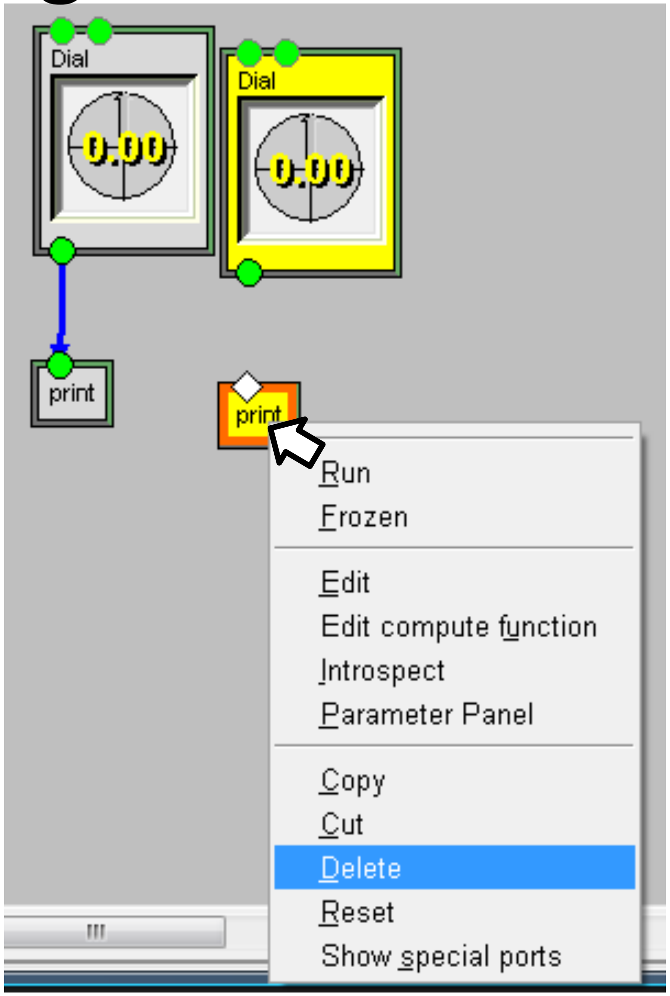
1 – right click on the selected print node
2 – choose “*Delete*” in the menu

Task: cut multiple nodes and connections

1 – select 2 nodes
2 – click on the cut icon in the tool bar ✂

Task: delete all nodes

1 – use Ctrl-A to select everything in the network
2 – right-click on a selected node and chose “Delete”

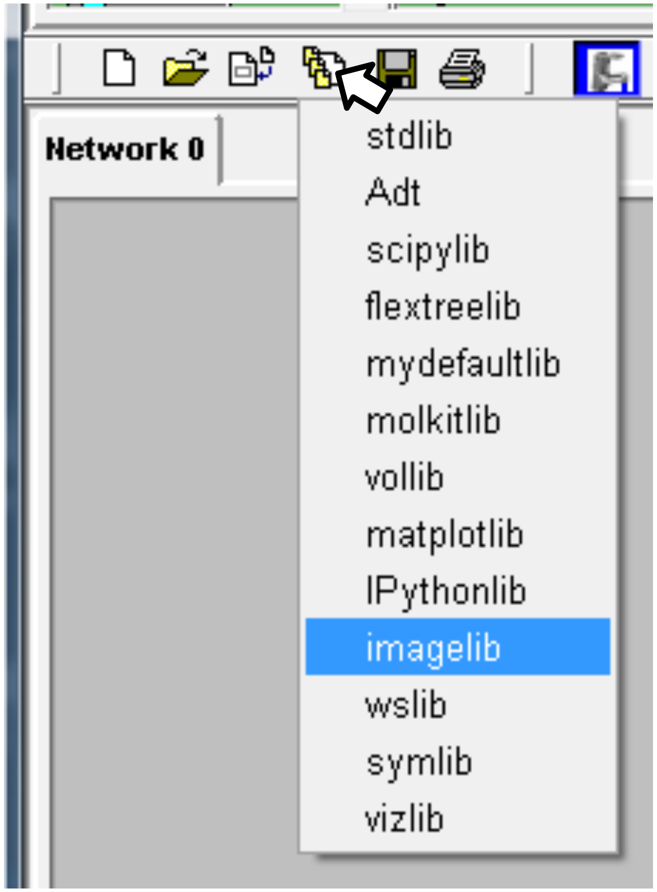


Exercise: loading a library

Task: load the Python Imaging library (PIL)

Using the button bar
1 – click on the "*load library*" icon

2 – select "*imagelib* "

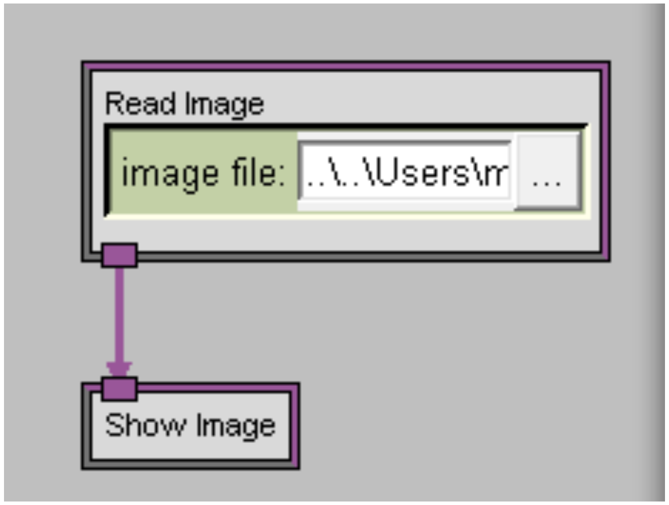


Exercise: Image viewer network

Task: create a network to display an image

1 – locate and instantiate a “*Read Image”*node
2 – locate and instantiate a “*Show Image*” node
Note that a new window is created. It will hold the image.
3 – Connect the output of *Read Image* to the input of *Show Image*
4 – click on the file browser icon I the node …

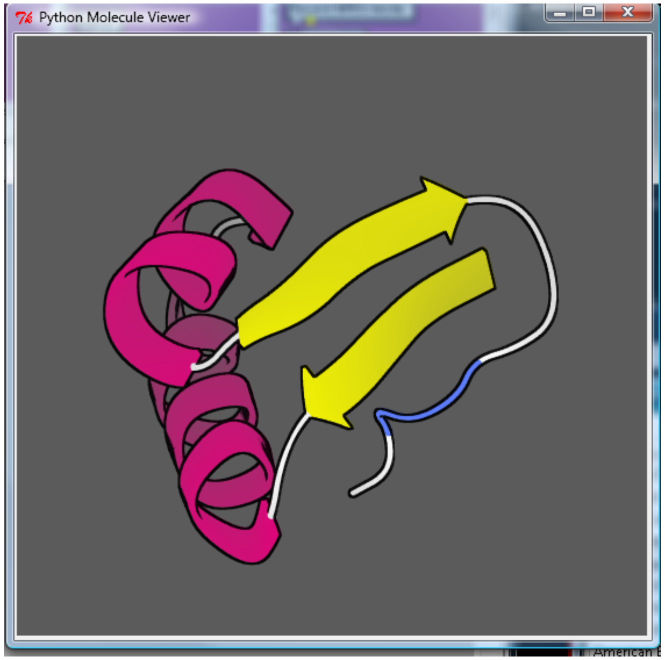
5 – navigate to Desktop/TutorialData/frames/ and select frame0000.png



Read Image

image file: ..\..\Users\m …

Show Image



Exercise: scale image

Task: modify the network to allow scaling the image

1 – locate and instantiate the scale node
2 – delete the connection between *Read Image* and *Show Image*
3 – connect *Read Image* to input of *Scale* and output of *Scale* to *Show Image*
4 – modify scale value

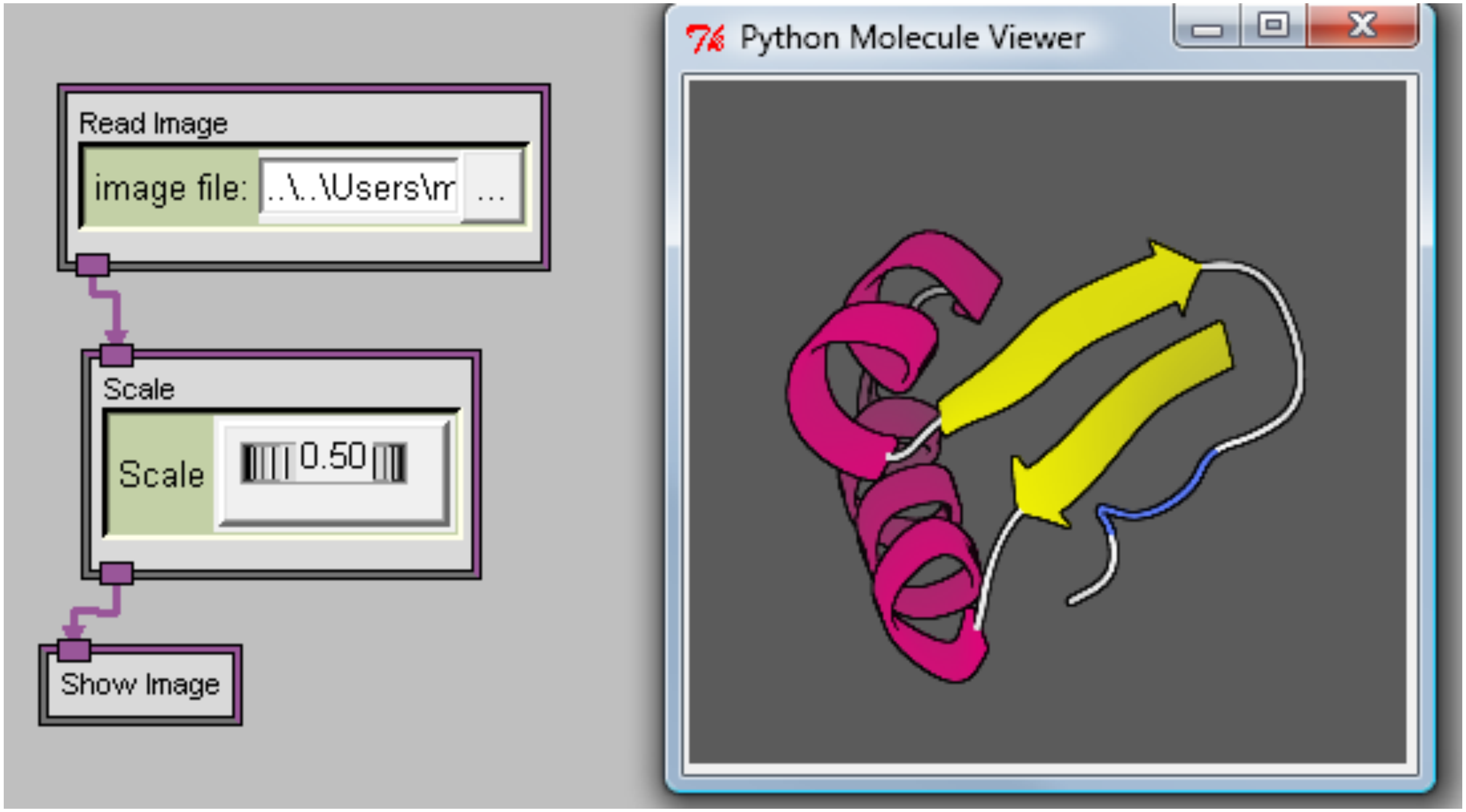
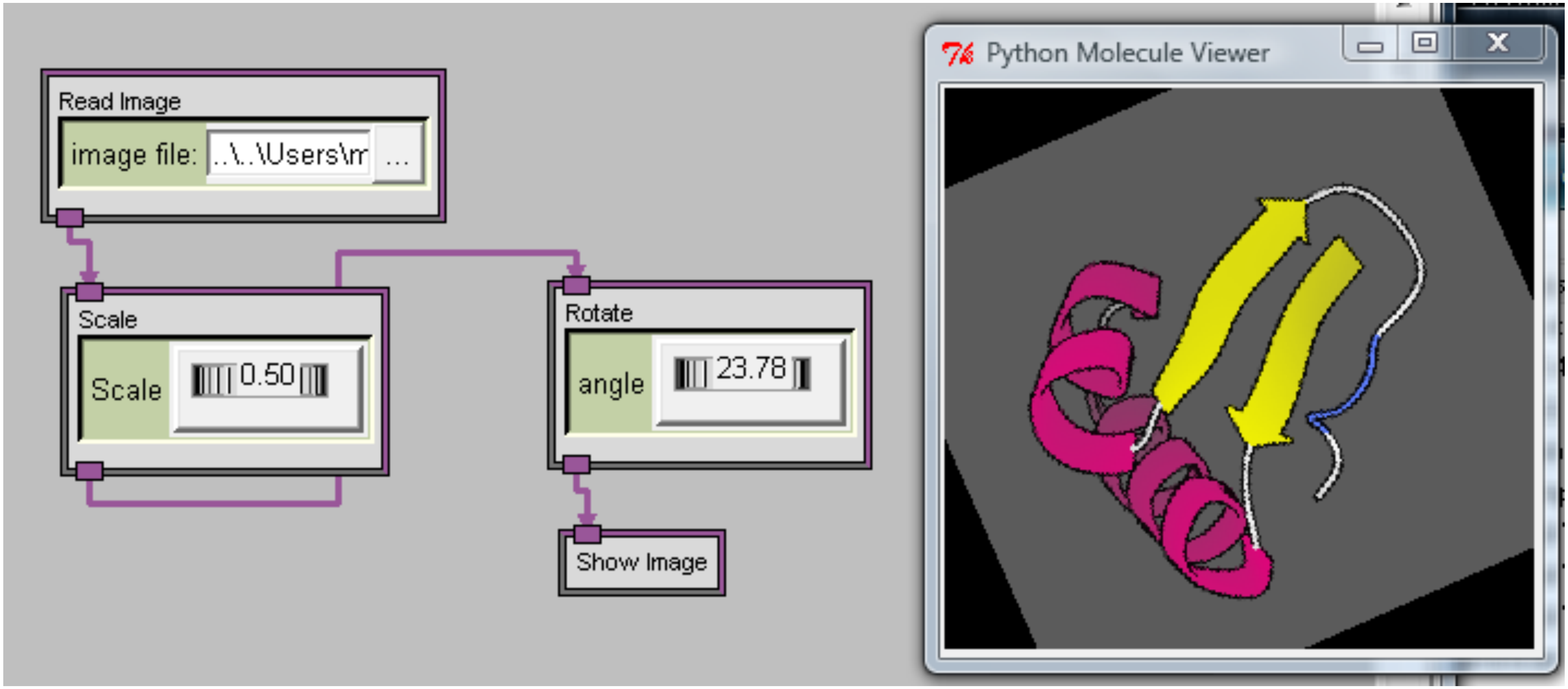


image file: ..\..\Users\m

Scale 0.50

Exercise: rotate image

Task: modify the network to allow rotating the image



Note how sharp edges become jagged

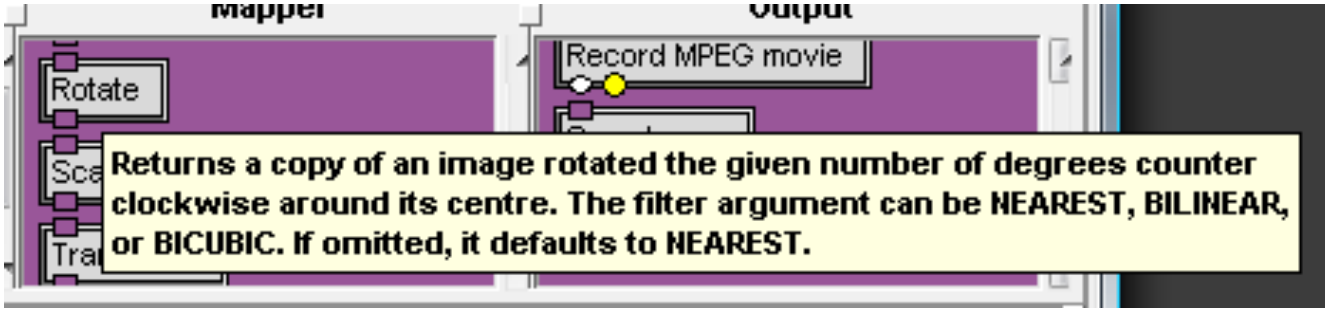
Exercise: rotate image

Task: fix image edges using rotation interpolation filter

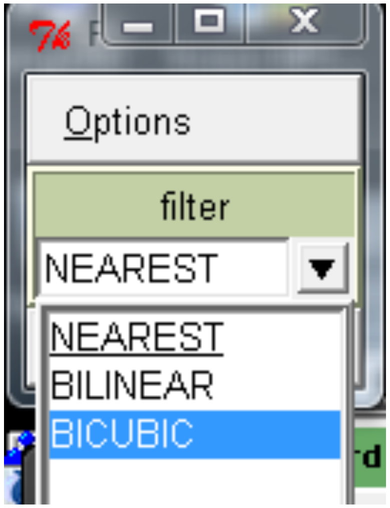
1 – read documentation string of the *Rotate* node. Notice the mention of a filter. Since the filter port is not visible it must be bound to a widget. Since the widget for filter is not visible in the node it must be in the node's Parameter Panel

2 – right-click on the rotate node and select Parameter Panel.

4 – in the parameter panel choose BICUBIC

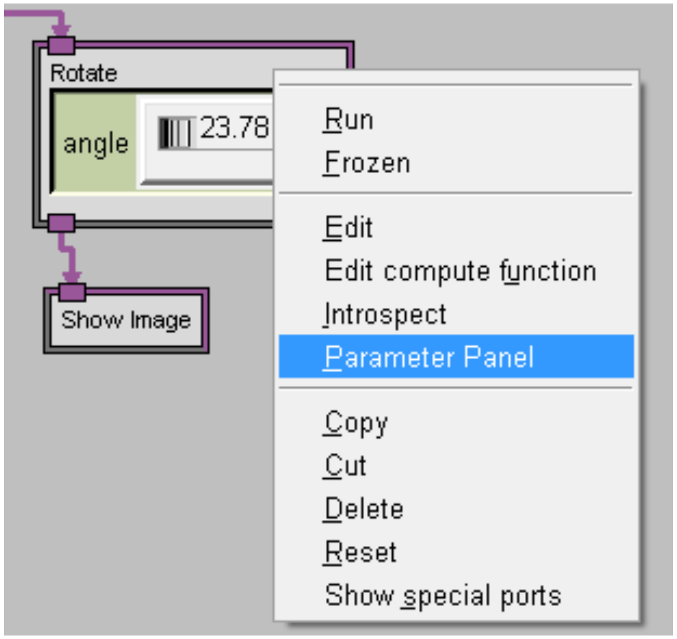


Returns a copy of an image rotated the given number of degrees counter clockwise around its centre. The filter argument can be NEAREST, BILINEAR, or BICUBIC. If omitted, it defaults to NEAREST.

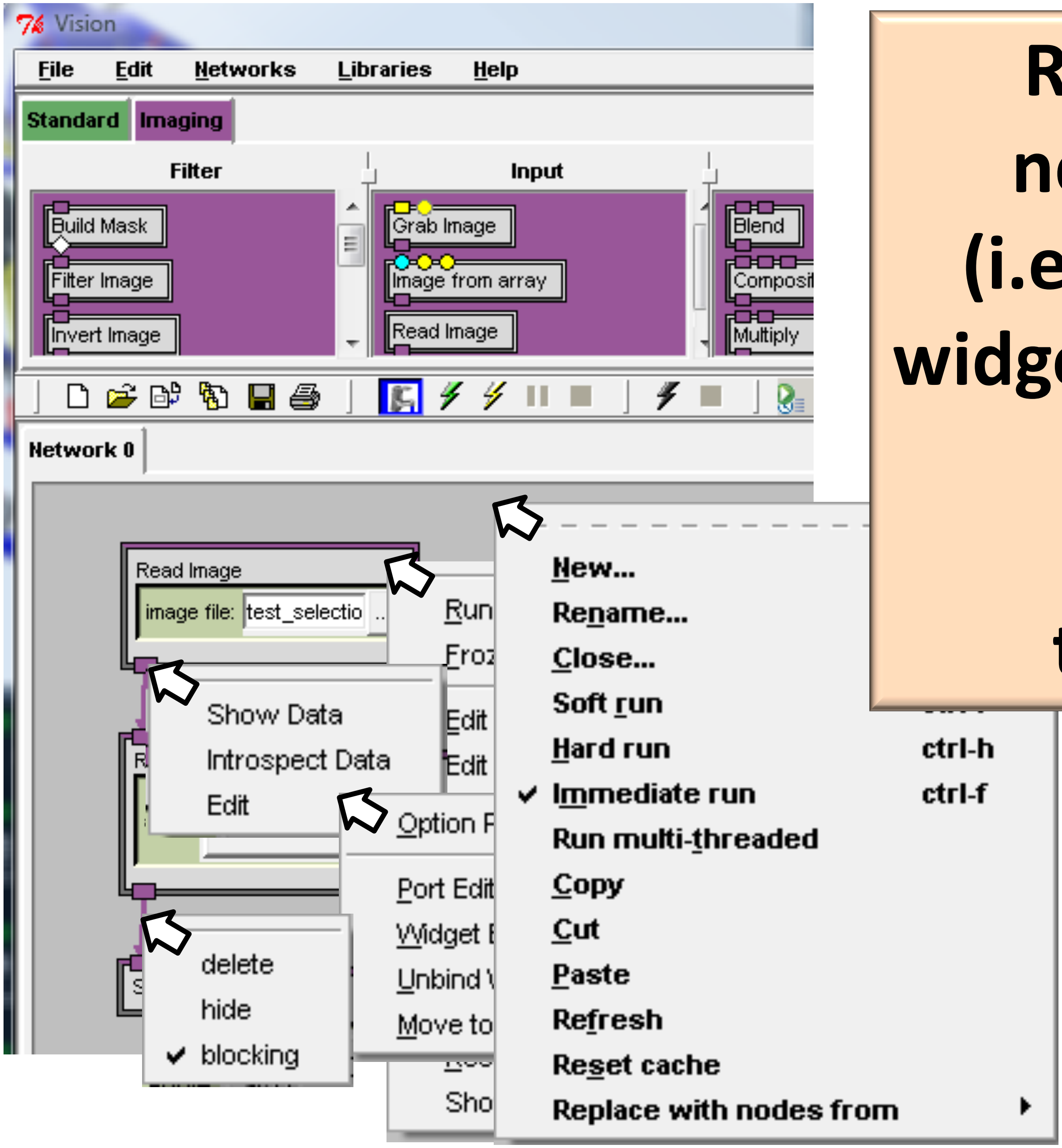


filter

NEAREST

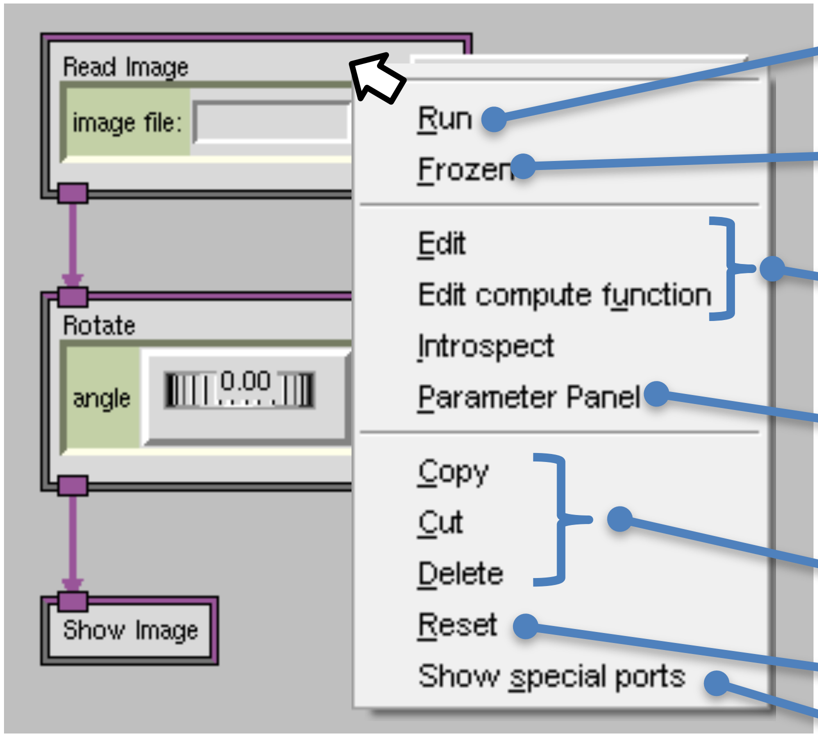


Network Items Menus



Right click on network items (i.e. nodes, ports, widgets, connections, Networks) to display their menus

Node Menu



Run this node and its children

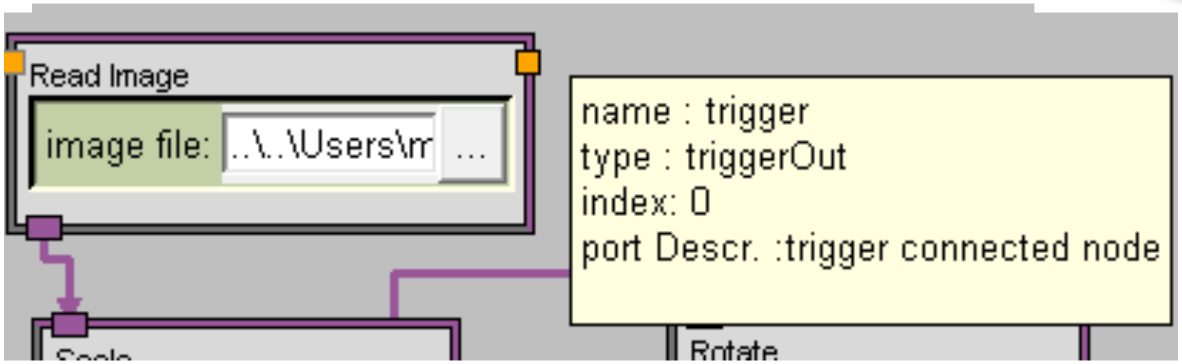
Freeze this node.
Frozen nodes do not run

Start the node editor

Displays a window containing widgets associated with this node

Copy, Cut, Delete this node

Replace node with a fresh copy



name : trigger
type : triggerOut
index: 0
port Descr. :trigger connected node

Displays a *run* port on the right side of the node and a *trigger* port on the left side

Port Menu

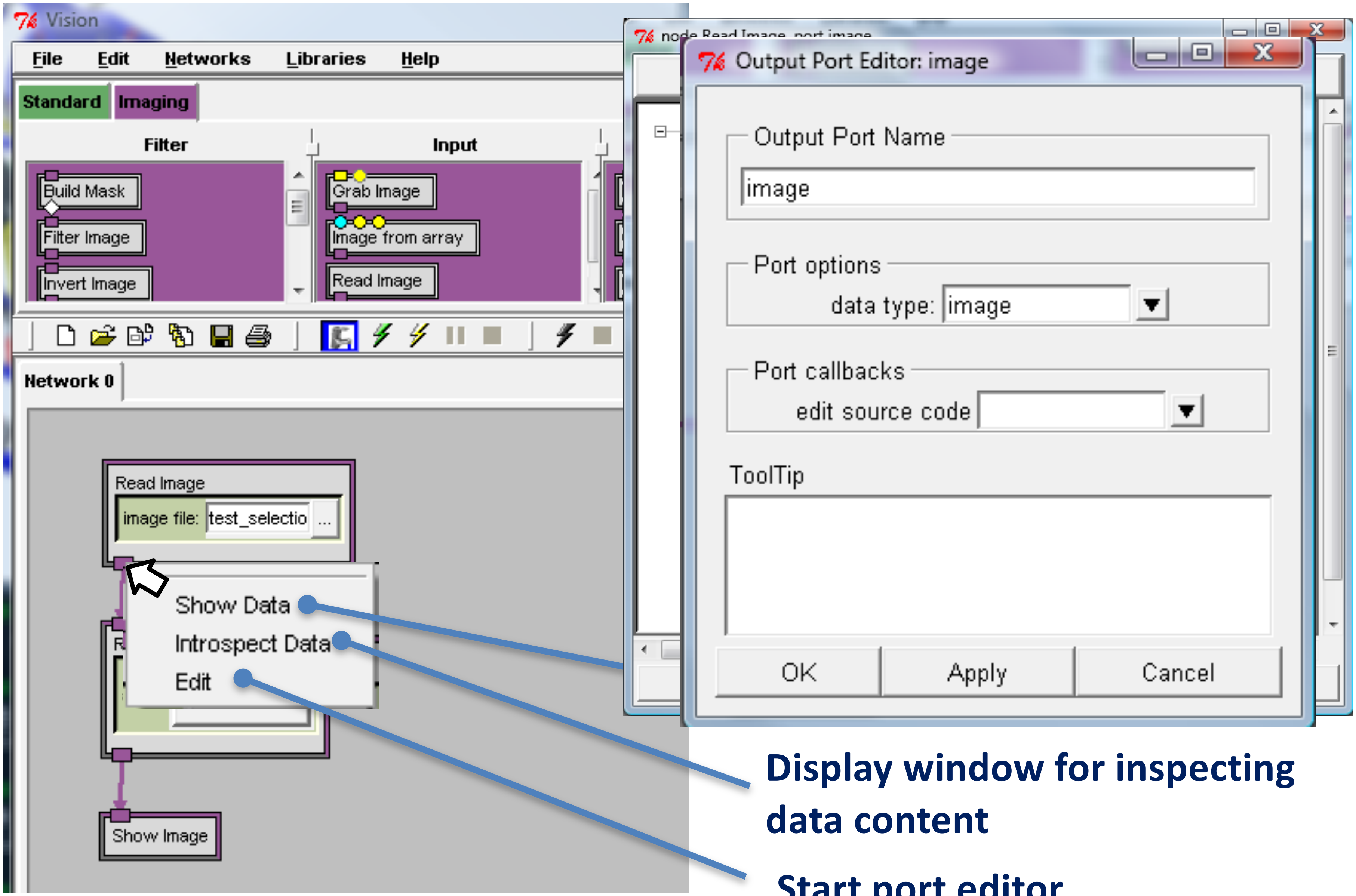


image file: test_selectio ...

Output Port Name: image

data type: image

Display window for inspecting data content

Start port editor

Widget Menu

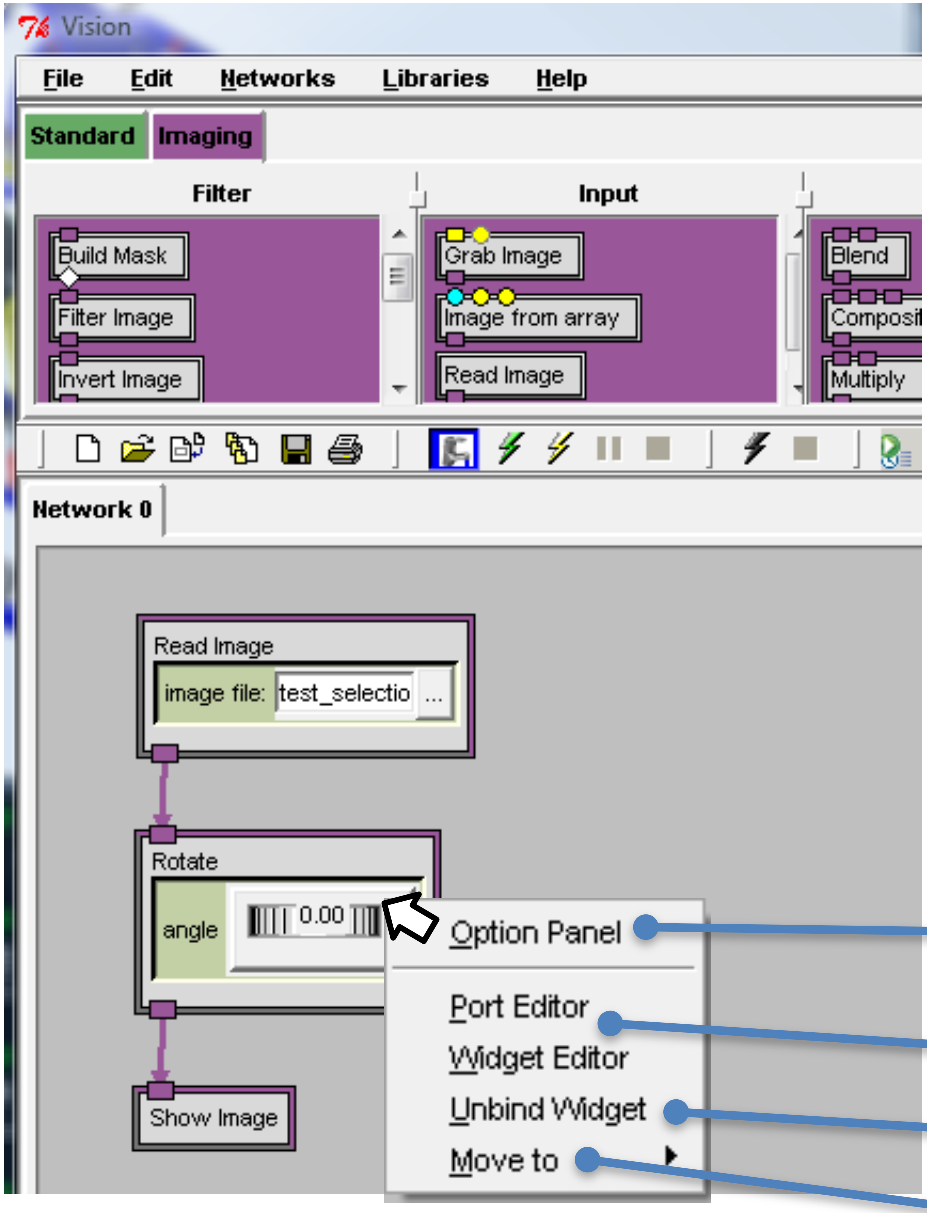
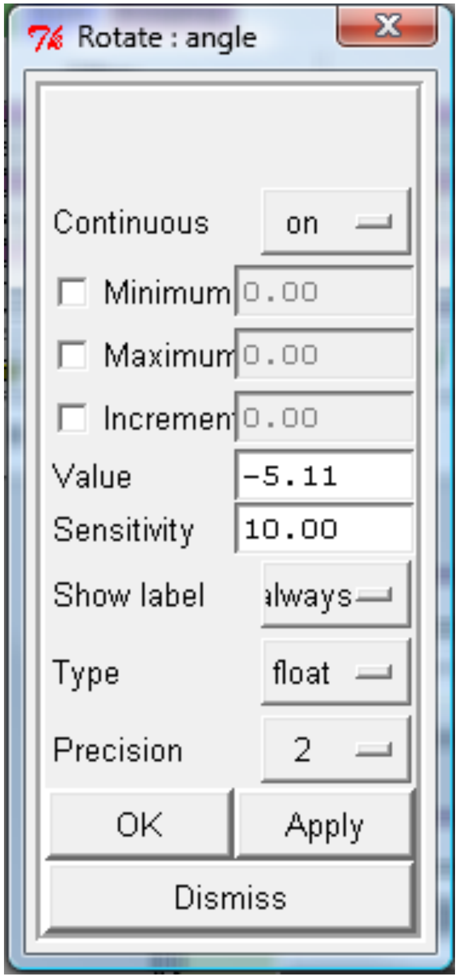


image file: test_selectio

angle 0.00


Rotate : angle

Continuous	on
□ Minimum	0.00
□ Maximum	0.00
□ Increment	0.00
Value	-5.11
Sensitivity	10.00
Show label	always
Type	float
Precision	2

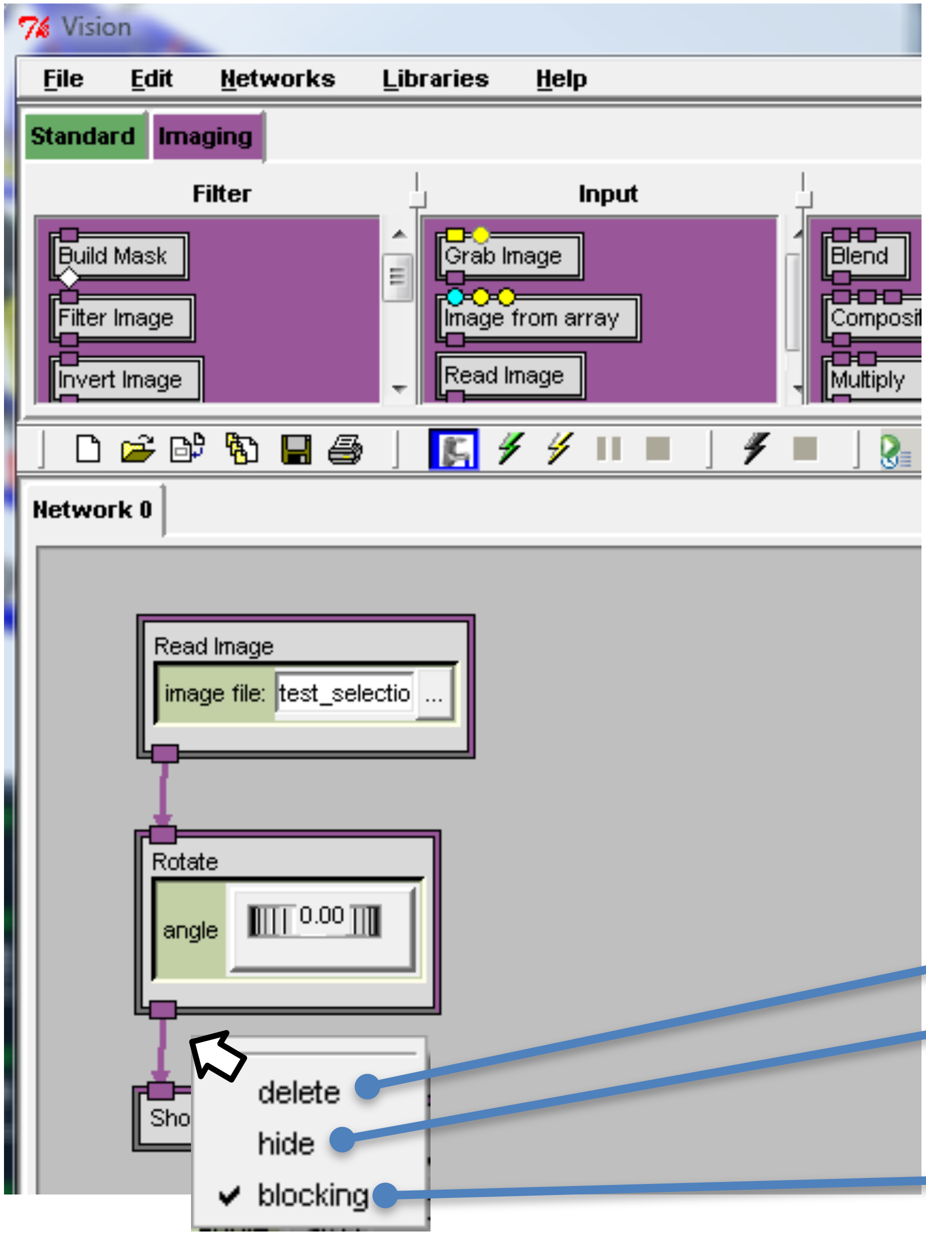
Display option panel
Only for Dial and Thumbwheel

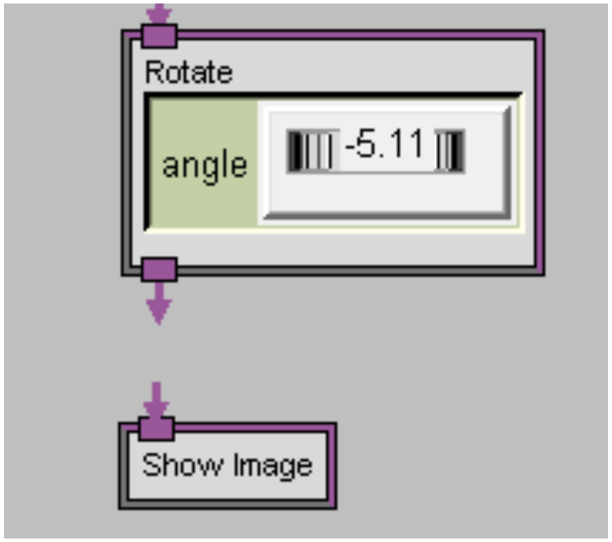
Start widget and port editor

Show port at top of node

Move widget to node or panel

Connection Menu



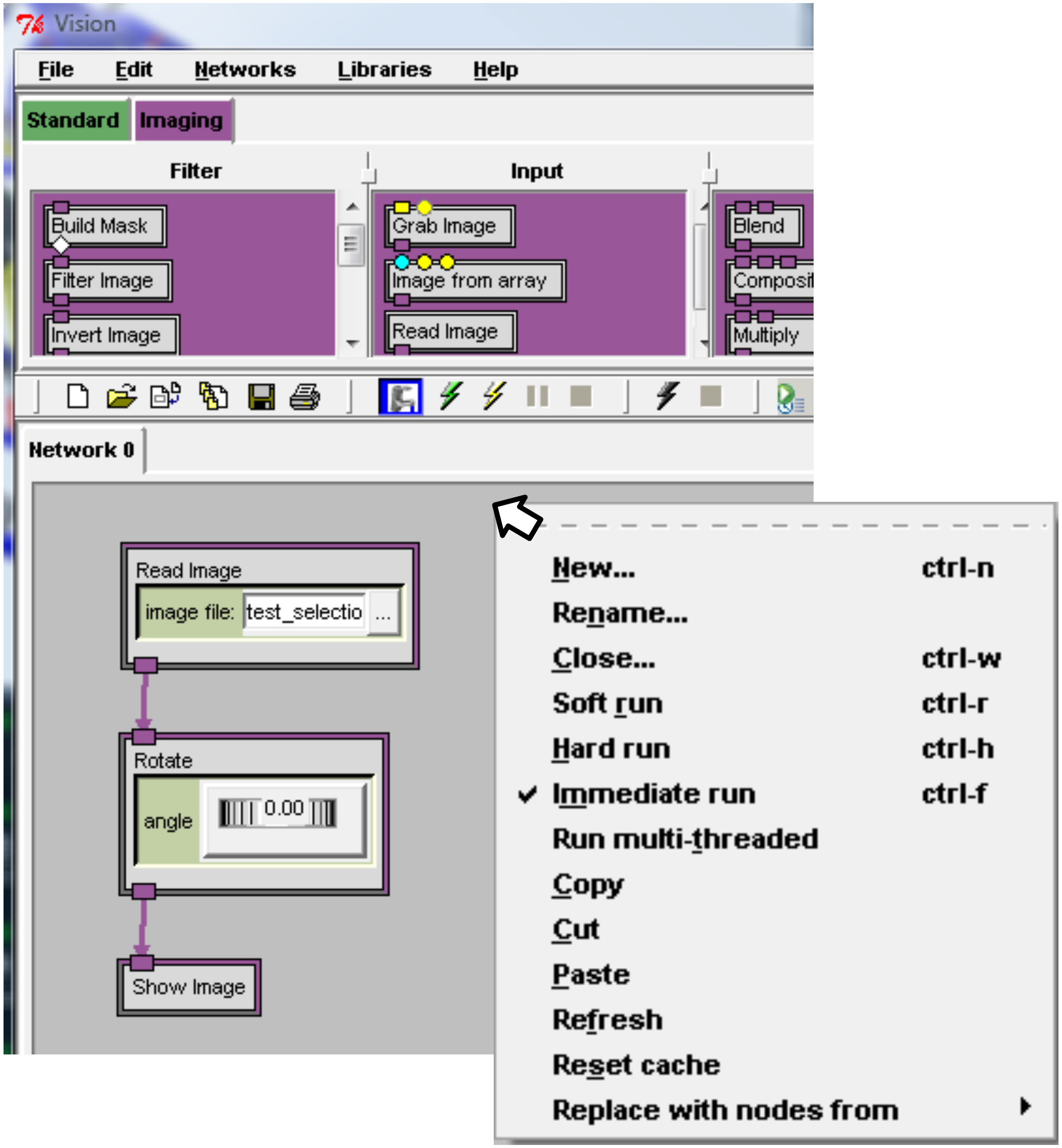


Delete connection

Hide connection

Toggle connection's blocking flag. Non blocking means a child can execute before the parent has

Network Menu



Exercise: browser 1

Task: Build an image browser to look at the images located in the movieFrame directory on the desktop

0 – delete content of network

1 – locate and instantiate a *NumberedName* node

2 – set the directory to *Desktop/TutorialData/frames*

3 – set the base name to '*frame*'

4 – set padding to 4

5 – locate and instantiate a *ThumbwheelInt* node outputting an integer

6 – connect the output of *ThumbwheelInt* to the *number* input port of *NumberedName*

7 – locate and instantiate a *Read Image* node form the imaging library

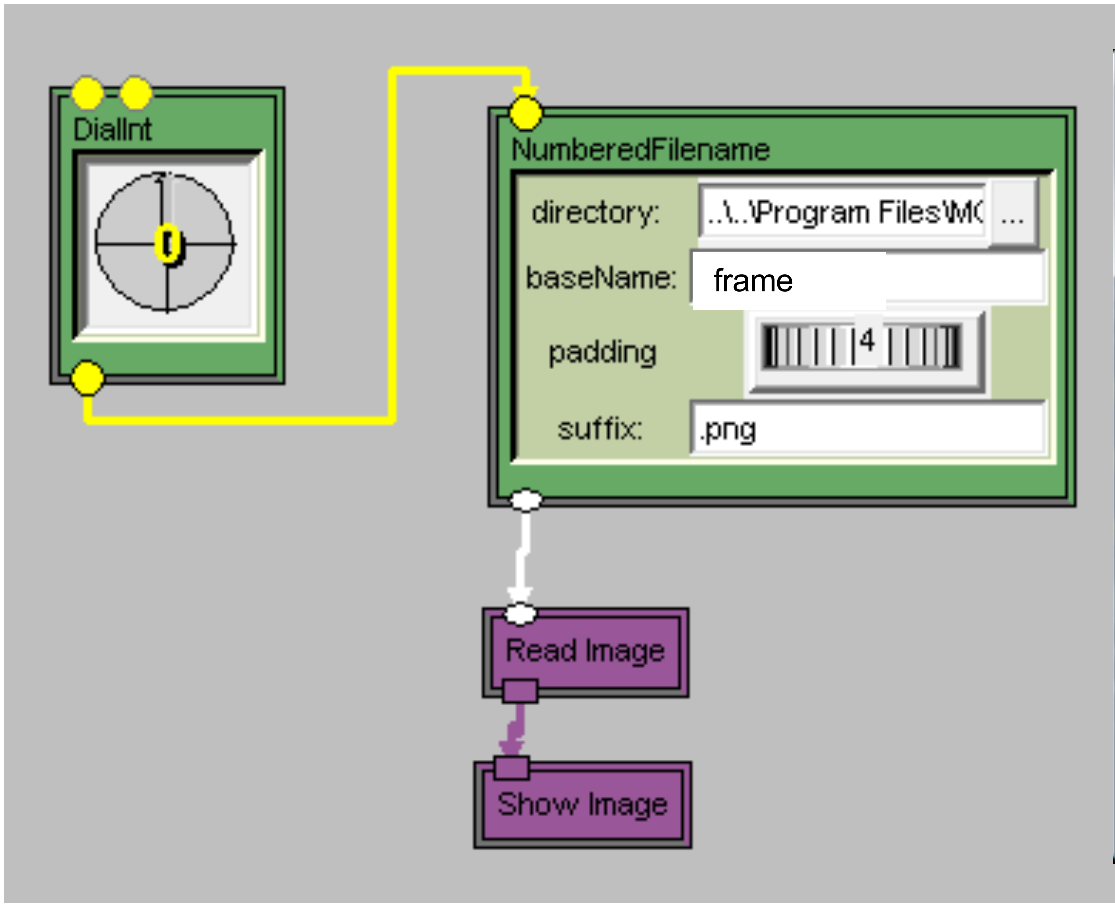
8 – unbind the file browser widget from the *filename* input port in *Read Image*

9 – connect the *filename* output of the *NumberedName* node to the *filename* input port of the read Image node

10 – add the *show Image* node and connect the output of *Read Image* to the output of *Show Image*

Exercise: browser1

Task: Build an image browser to look at the images located in the tutorialData/frames directory on the desktop

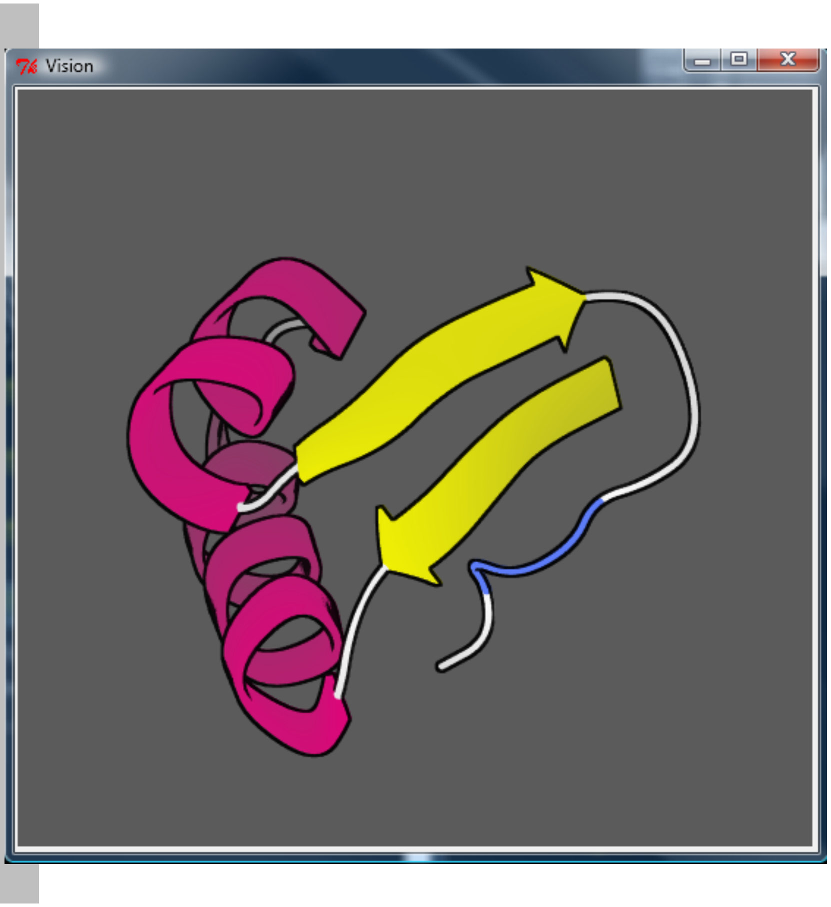


directory: ..\..\Program Files\M

baseName: frame

padding: 4

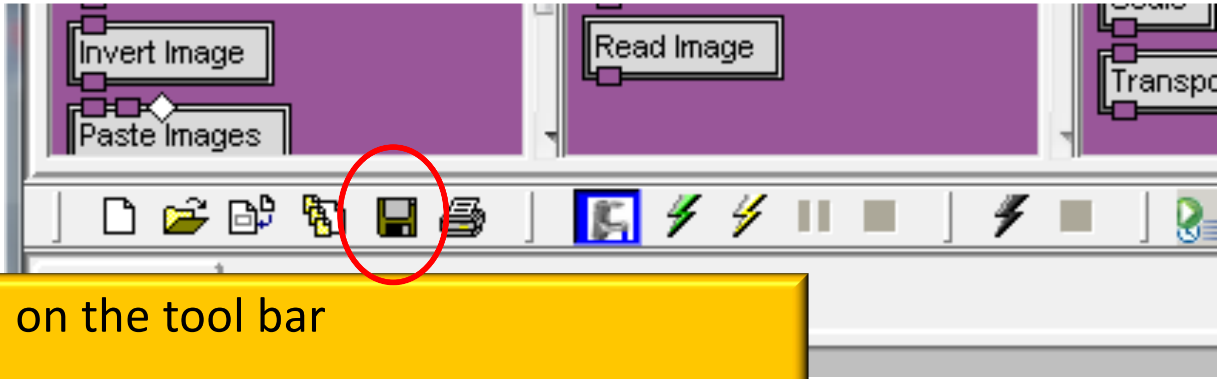
suffix: .png



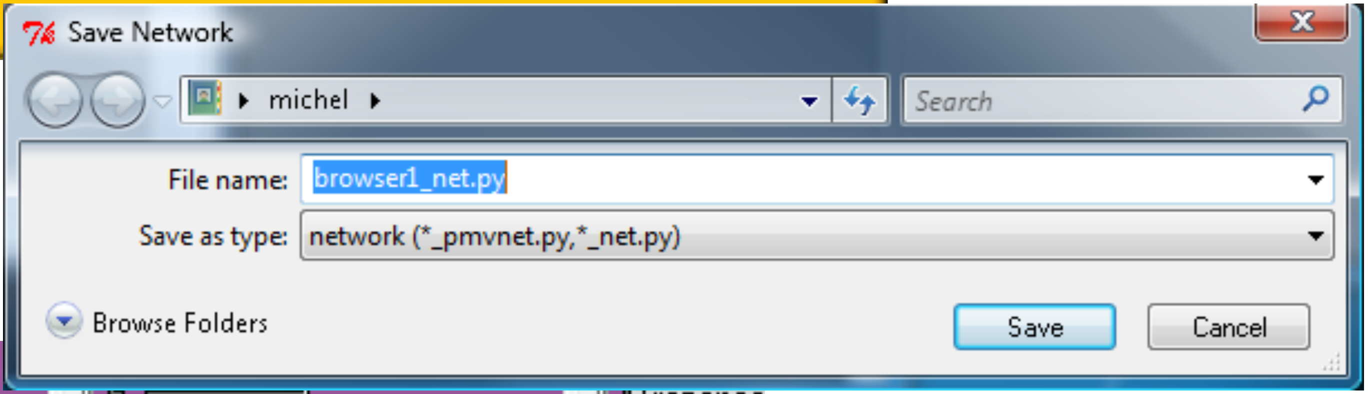
Use Edit -> Color node by library to get node colors

Exercise: save network

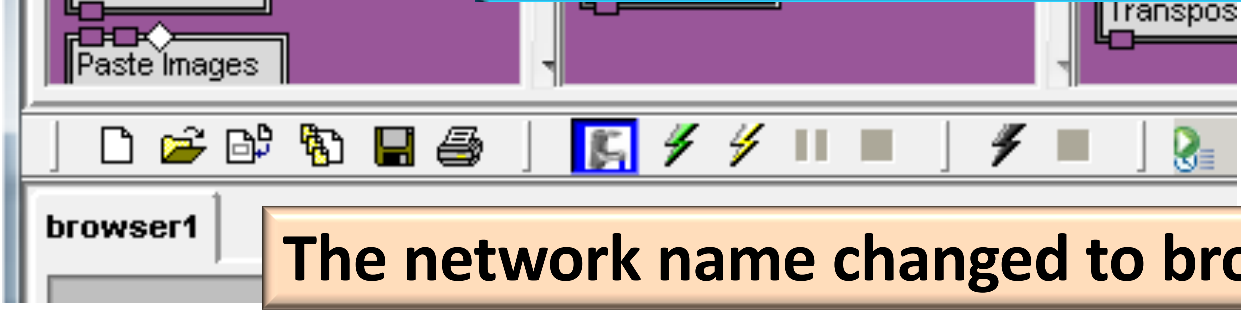
Task:
Save the network



1 – Click on the save button on the tool bar
2 – navigate to Desktop
2 – replace Network0_net.py by browser1_net.py in the browser
3 – click on save



File name: browser1_net.py
Save as type: network (*_pmvnet.py,*_net.py)



The network name changed to browser1

Exercise: iterate over images

Task: make the network iterate over the file numbers

1 – delete the thumbwheel node

2 – locate the *range* node, read its documentation and instantiate one to create a list of numbers ranging from 0 to 100

3 – locate and instantiate an *iterate* node

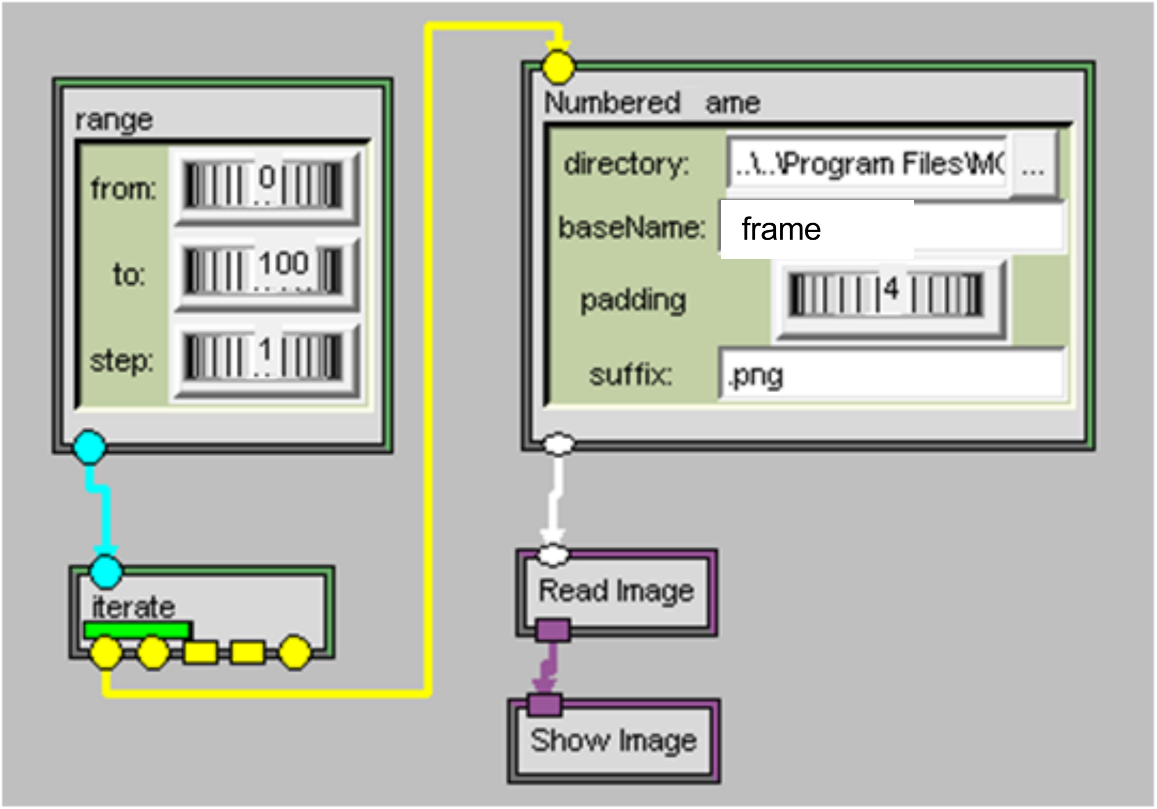
4 – send the list created by *range* into the *iterate* node

5 – connect the *oneItem* output port of *iterate* to the *number* input port for the *NumberedName* node

6 – Run the *iterate* node

Task:

Save the network as "browser2_net.py"



from: 0
to: 100
step: 1

directory: ..\..\Program Files\MC ...
baseName: frame
padding 4
suffix: .png

Exercise: iterate over images

Task: make the network iterate over the file numbers

1 – delete the thumbwheel node

2 – locate the *range* node, read its documentation and instantiate one to create a list of numbers ranging from 0 to 100

3 – locate and instantiate an *iterate* node

4 – send the list created by *range* into the *iterate* node

5 – connect the *oneItem* output port of *iterate* to the *number* input port for the *NumberedName* node

6 – Run the *iterate* node

The frame are not found !
Once a the network is saved, relative paths in the network will be interpreted relative to the SAVED location.

6 – set the directory in *NumberedFrame* to TutorialDat/frames

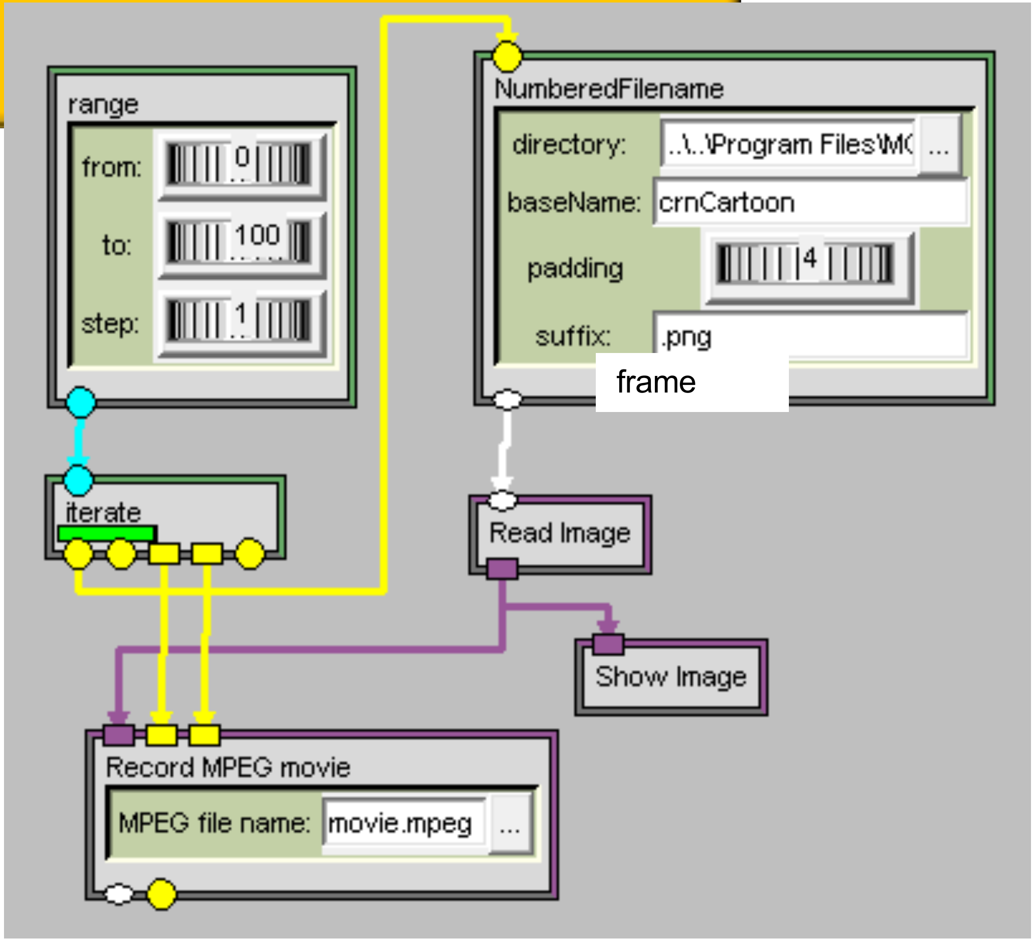
7 – run the network

Exercise: save MPEG

Task: save the sequence of images as an MPEG movie

1 – locate and instantiate Record MPEG Movie
2 – feed the image to the first input port
3 – connect the *begin* and *end* output ports of iterate to the *begin* and *end* input ports of Record MPEG Movie
4 – run the iterate node

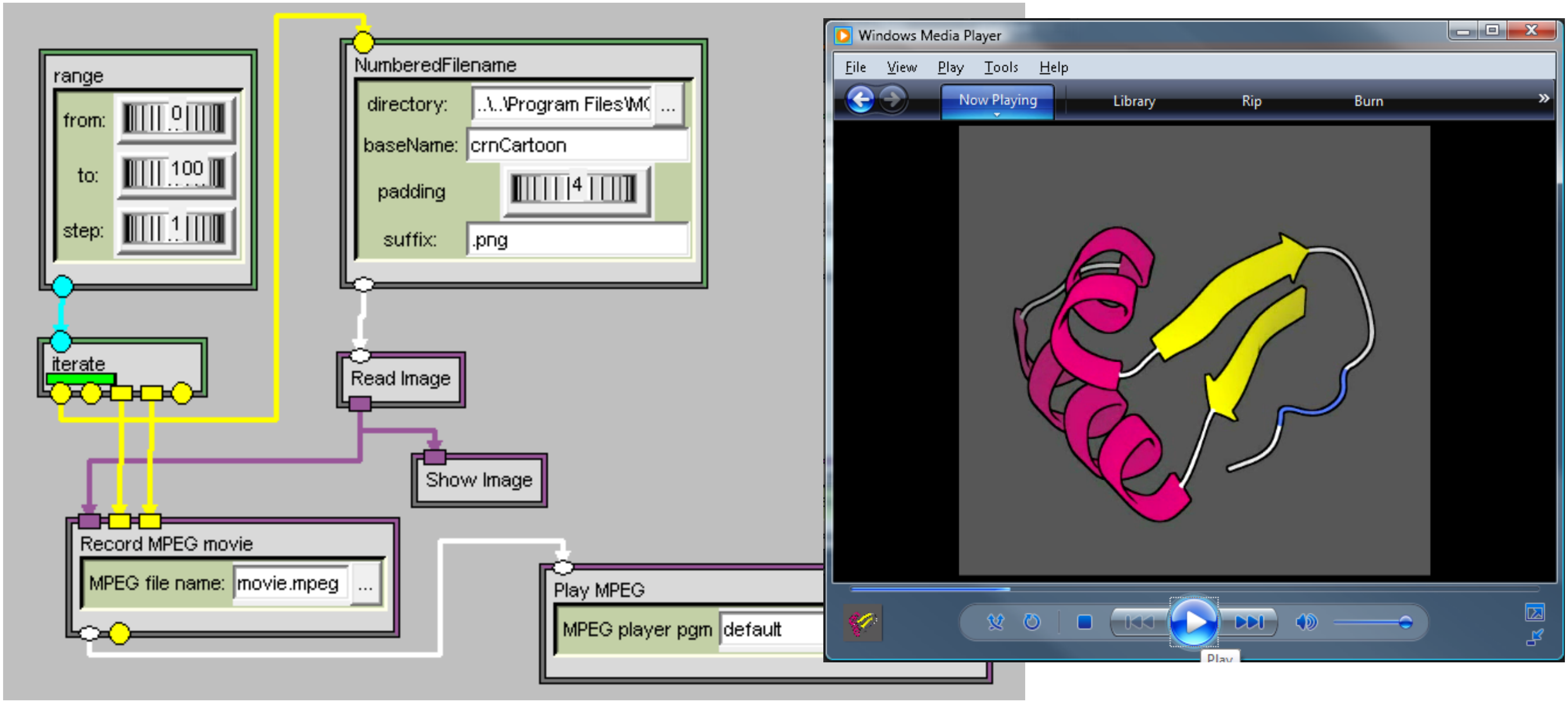
Task:
Save the network as "saveMPEG_net.py"



Exercise: Play back movie

Task: play the MPEG movie

1 – locate and instantiate *Play MPEG* node
2 – unbind the widget from its *movieFileName* port
3 – feed the movie name from the *Record MPEG* node into the *Play MPEG* node
4 – kill the MPEG player to end the *Play MPEG* node's execution



range
from: 0
to: 100
step: 1

NumberedFilename
directory: ..\..\Program Files\MG
baseName: crnCartoon
padding 4
suffix: .png

Record MPEG movie
MPEG file name: movie.mpeg

Play MPEG
MPEG player pgm default

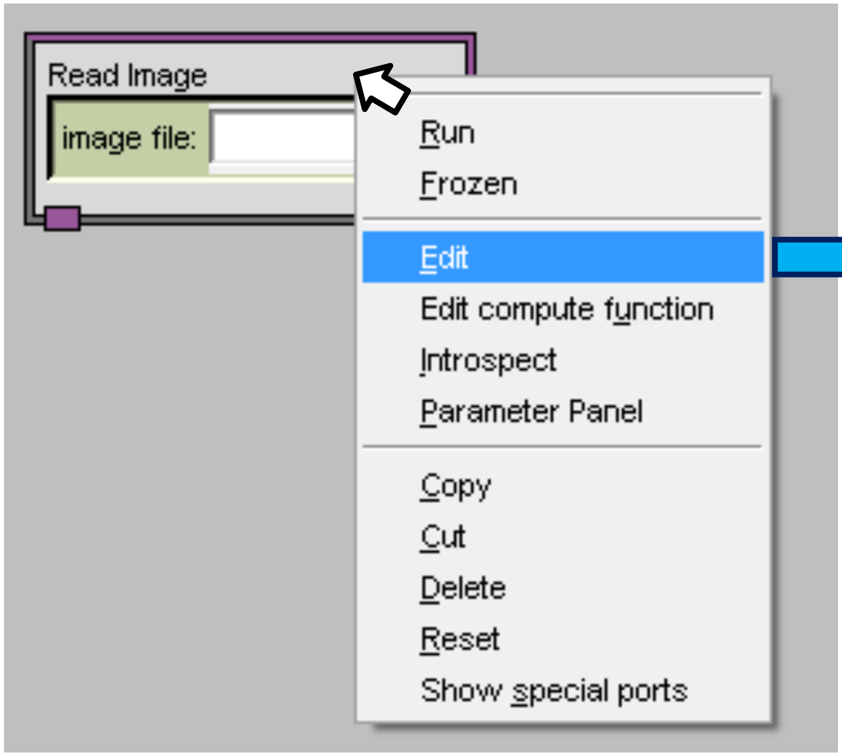
Section 2: Overview

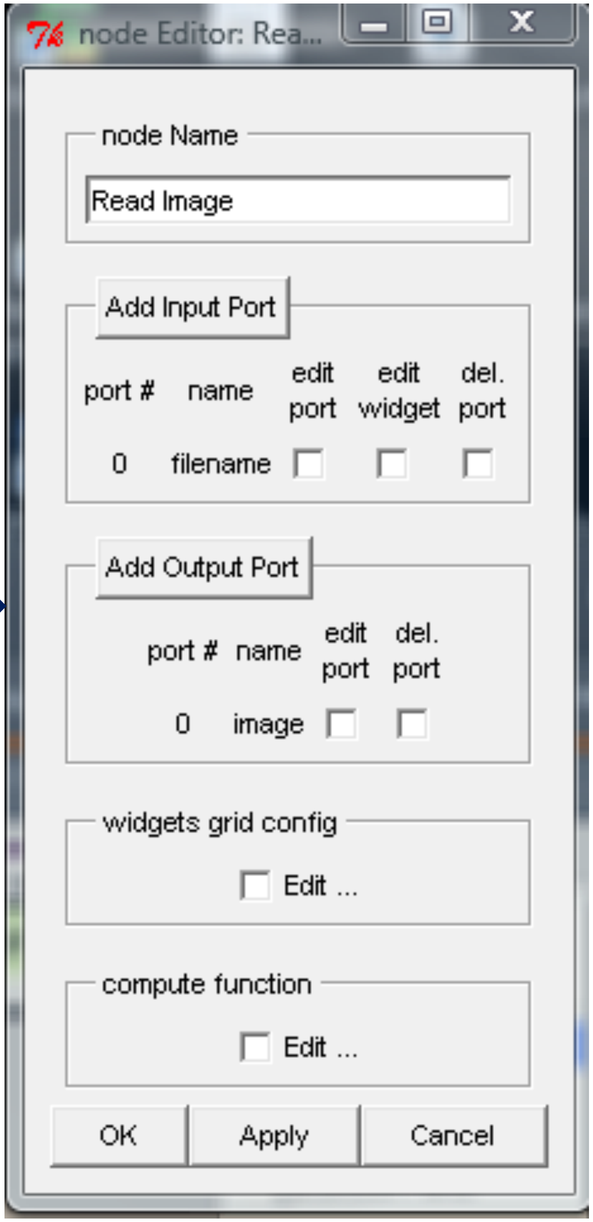
- Look inside a node and modify it
- User library of nodes
- Writing a new node
- User panels and moving widgets
- Command line execution
- Detached execution
- Macro nodes

Exercise: Node editor

Task: start the node editor on the *Read Image* node

1 – Start Vision
2 – Load the imaging library
2 – locate and instantiate a *Read Image* node
3 – Right click on the node’s background and select “*Edit*”





node Name: Read Image

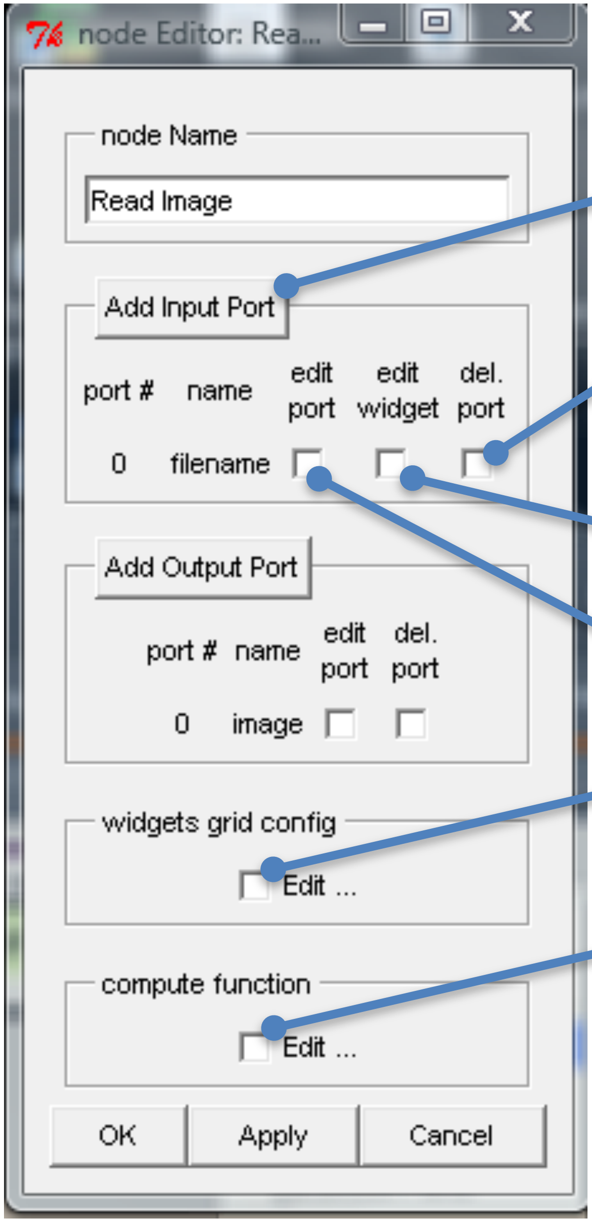
port #	name	edit port	edit widget	del. port
0	filename	□	□	□

port #	name	edit port	del. port
0	image	□	□

widgets grid config: □ Edit ...

compute function: □ Edit ...

Notes: Node editor



node Name: Read Image

port #	name	edit port	edit widget	del. port
0	filename	□	□	□

port #	name	edit port	del. port
0	image	□	□

widgets grid config: □ Edit ...

compute function: □ Edit ...

Edit node name

Add/delete/edit Input Ports

Add/delete/edit Output ports

Edit widget gridding

Edit node' s function

Add port

Delete port

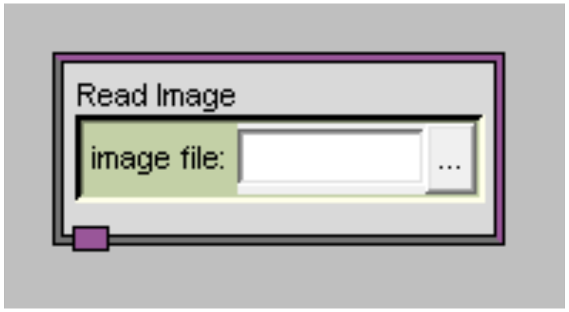
Start widget editor

Start port editor

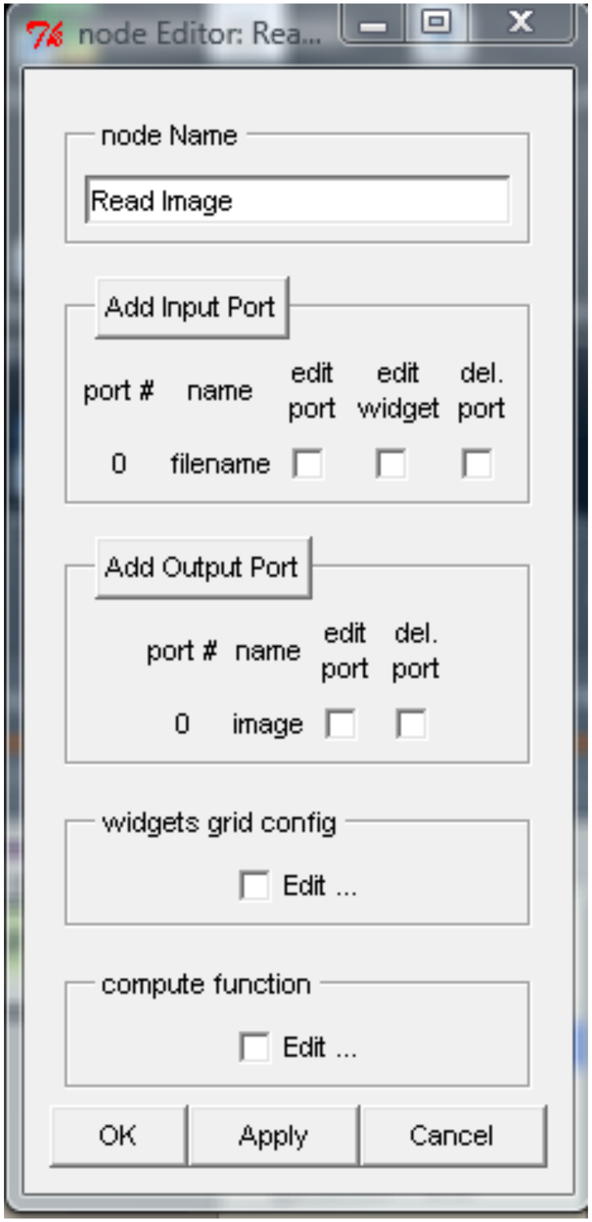
Start widget placement editor

Start code editor

Notes: Node editor



- **The node name is Read Image**
- **1 input port called filename**
- **1 output port called image**
- **Note the input port is not visible at the top of the node because it gets its data from a widget which is placed Inside the node.**
- **Double clicking on the node toggles showing/hiding widgets in the node**



node Name: Read Image

port #	name	edit port	edit widget	del. port
0	filename	□	□	□

port #	name	edit port	del. port
0	image	□	□

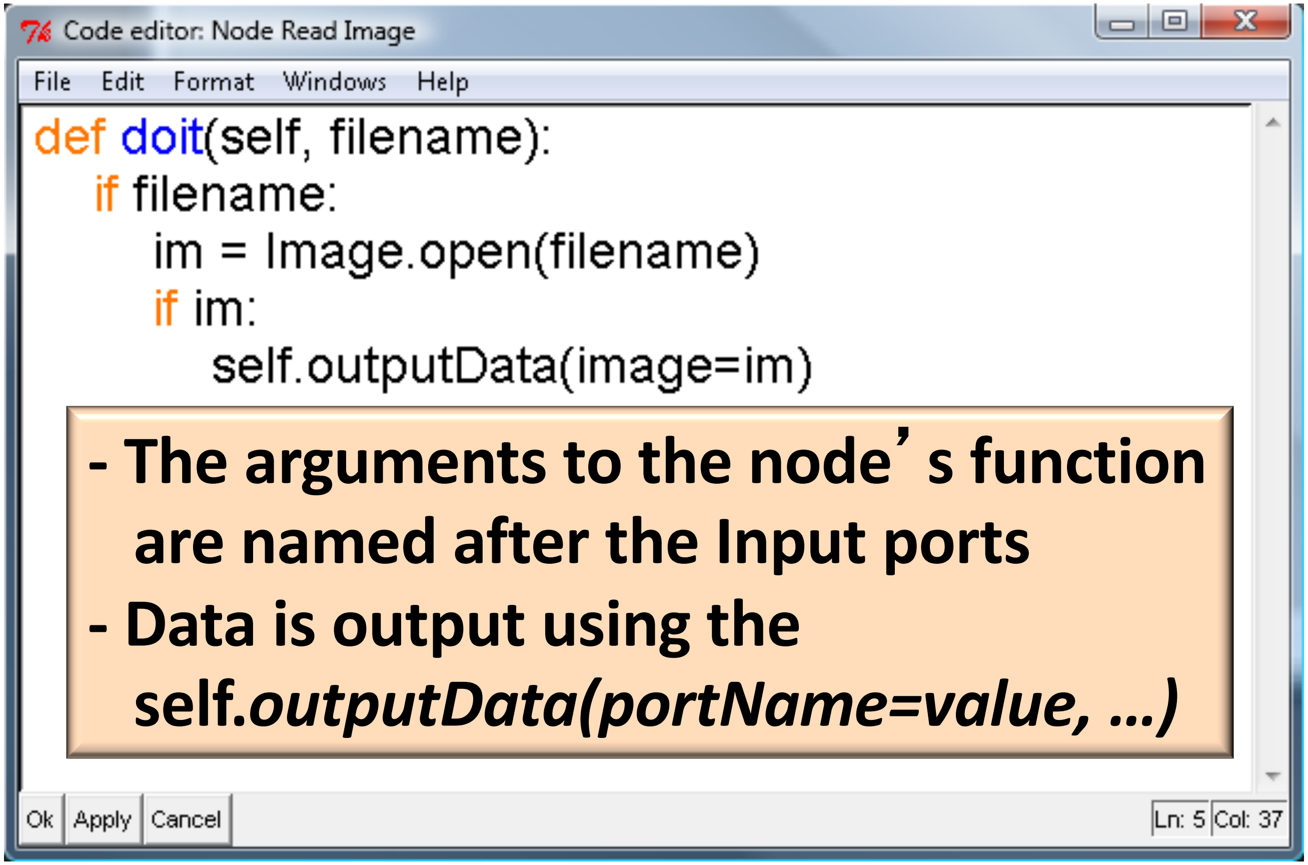
widgets grid config: □ Edit ...

compute function: □ Edit ...

Exercise: Code editor

Task: start the code editor on the *Read Image* node

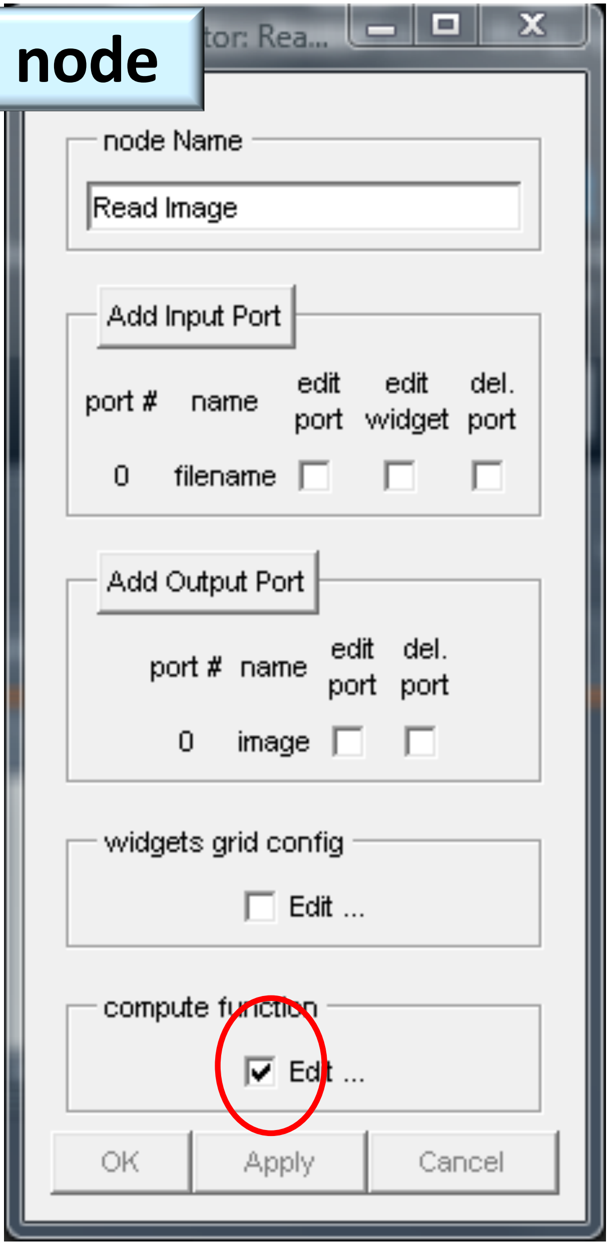
1 – check the Edit … check button in the node editor



```python
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
```

- The arguments to the node's function are named after the Input ports

- Data is output using the self.*outputData(portName=value, …)*



node Name: Read Image

port #	name	edit port	edit widget	del. port
0	filename	□	□	□

port #	name	edit port	del. port
0	image	□	□

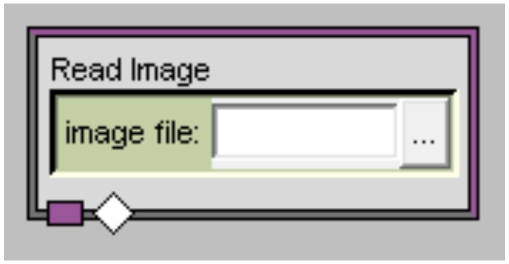
widgets grid config: □ Edit ...

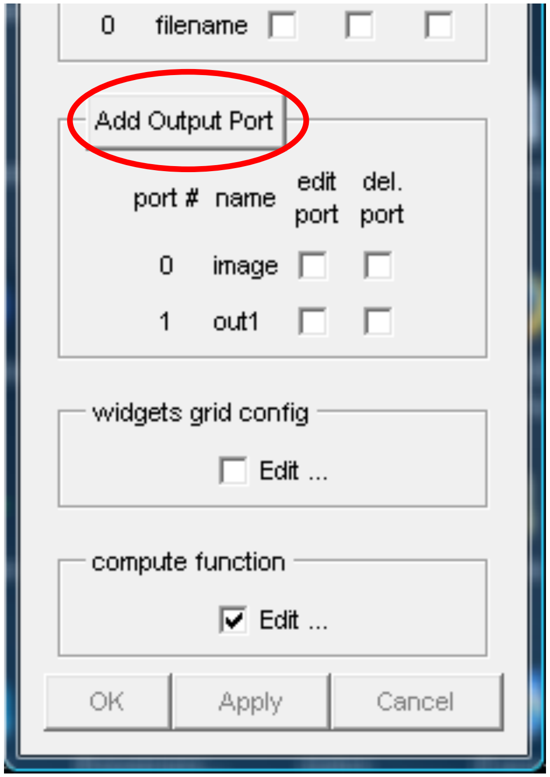
compute function: ☑ Edit ...

Exercise: Node editor

Task: add an output port that will provide the image size

1 – click on “Add Output Port”





0	filename	☐	☐	☐

Add Output Port

port #	name	edit port	del. port
0	image	☐	☐
1	out1	☐	☐

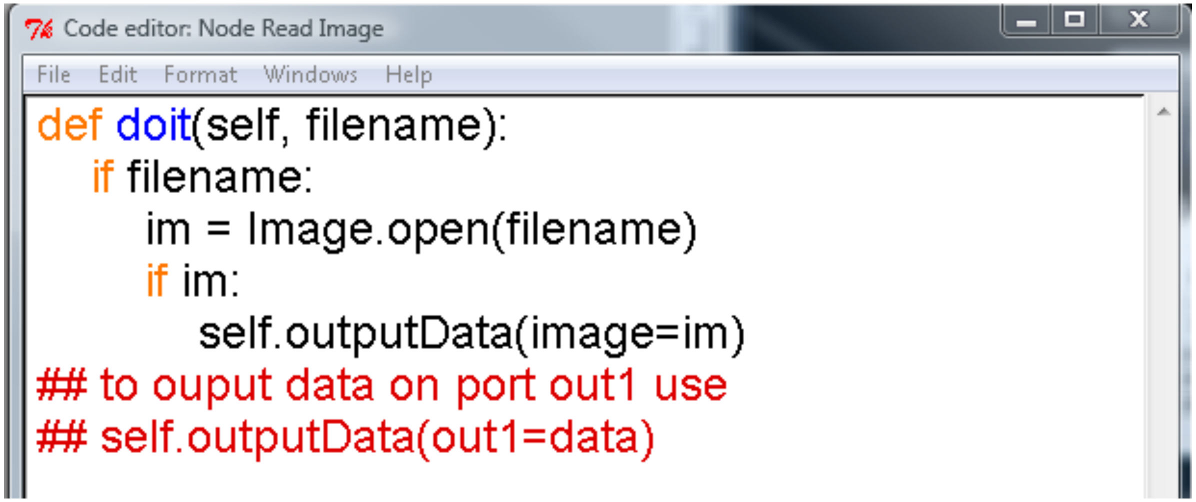
widgets grid config

☐ Edit ...

compute function

☑ Edit ...

- An output port called “out1” is created
- The port appears on the node
- Code is added to the function showing how to output data on the port

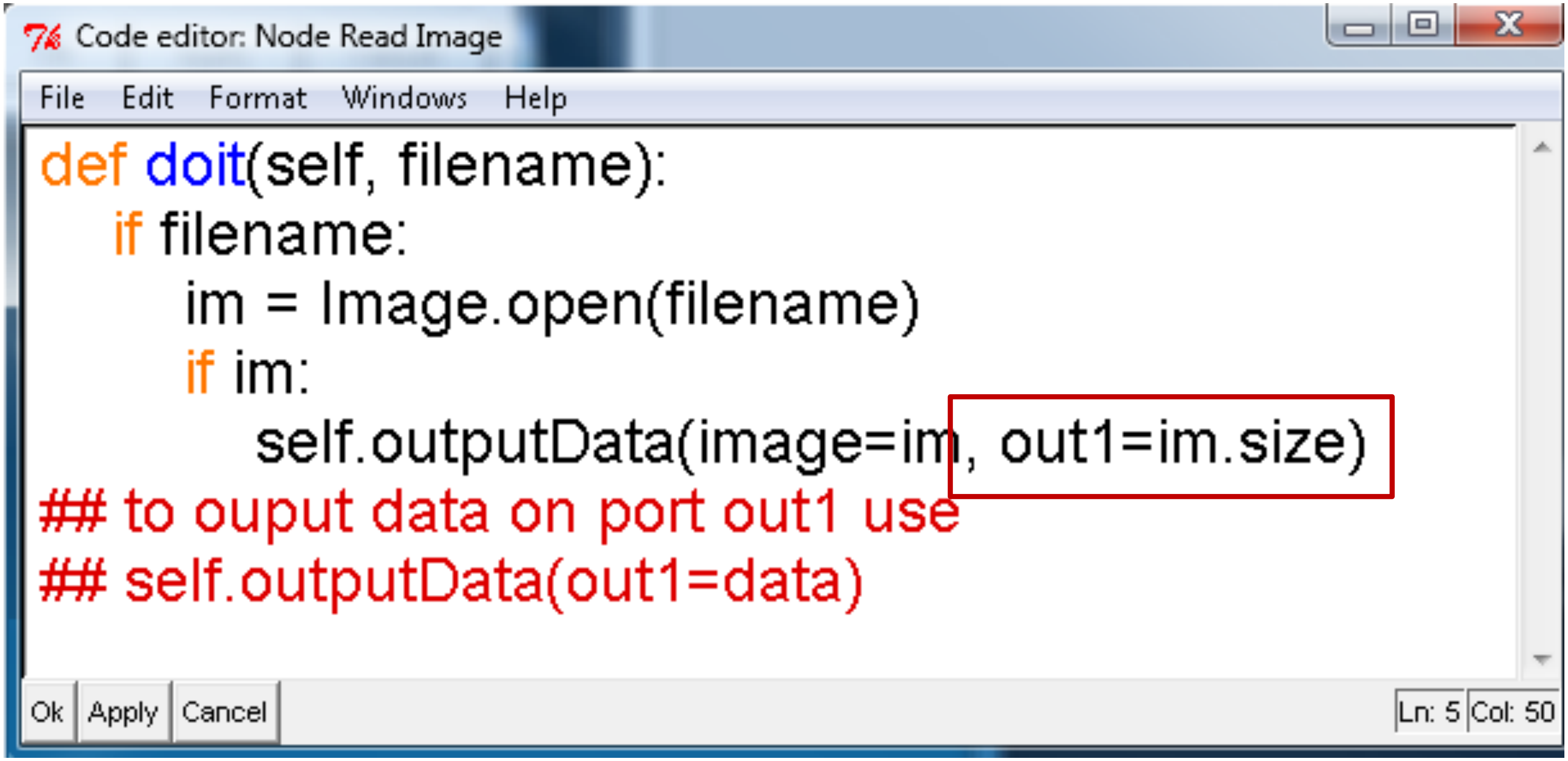


```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
## to ouput data on port out1 use
## self.outputData(out1=data)
```

Exercise: Node editor

Task: add an output port that will provide the image size

2 – modify the function to output “im.size” on the new output port

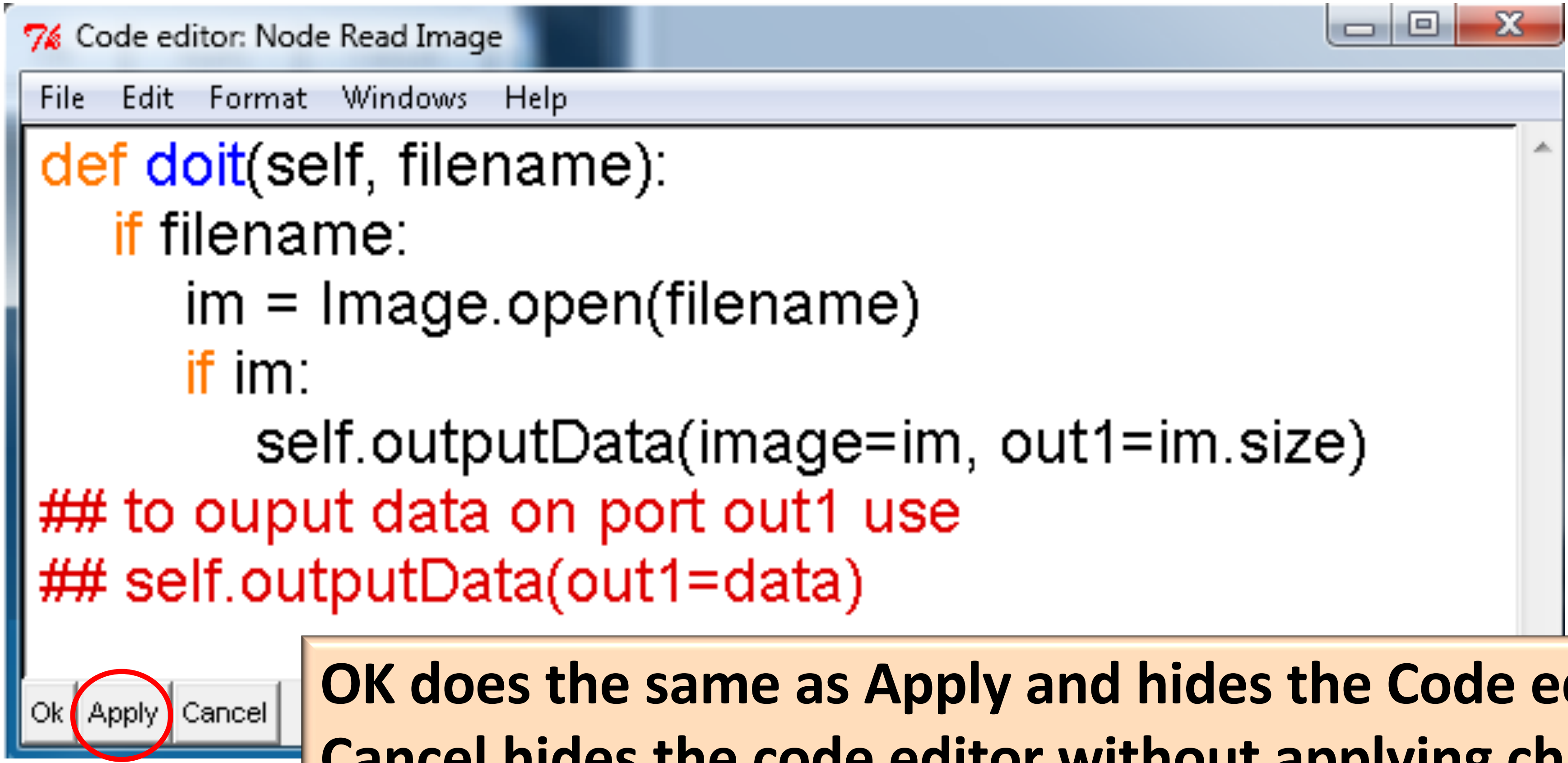


```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, out1=im.size)
## to ouput data on port out1 use
## self.outputData(out1=data)
```

Exercise: Node editor

Task: add an output port that will provide the image size

3 – Click Apply to set the function



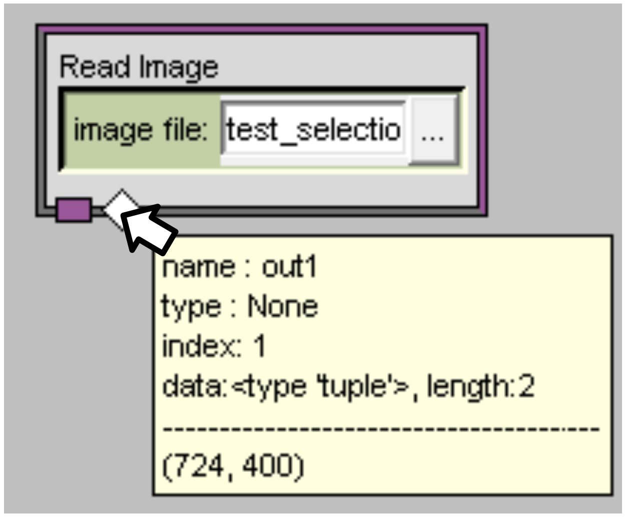
```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, out1=im.size)
## to ouput data on port out1 use
## self.outputData(out1=data)
```

OK does the same as Apply and hides the Code editor
Cancel hides the code editor without applying changes

Exercise: Node editor

Task: read an image and verify that the port outputs the dimensions

1 – read an image
2 – use the output port tooltip to look at the size



test_selectio

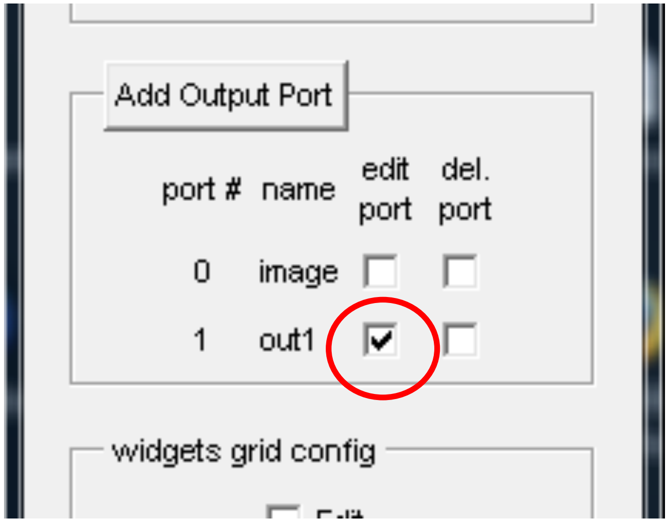
```
name : out1
type : None
index: 1
data:<type 'tuple'>, length:2
----------------------------------------
(724, 400)
```

Note that the data type is None

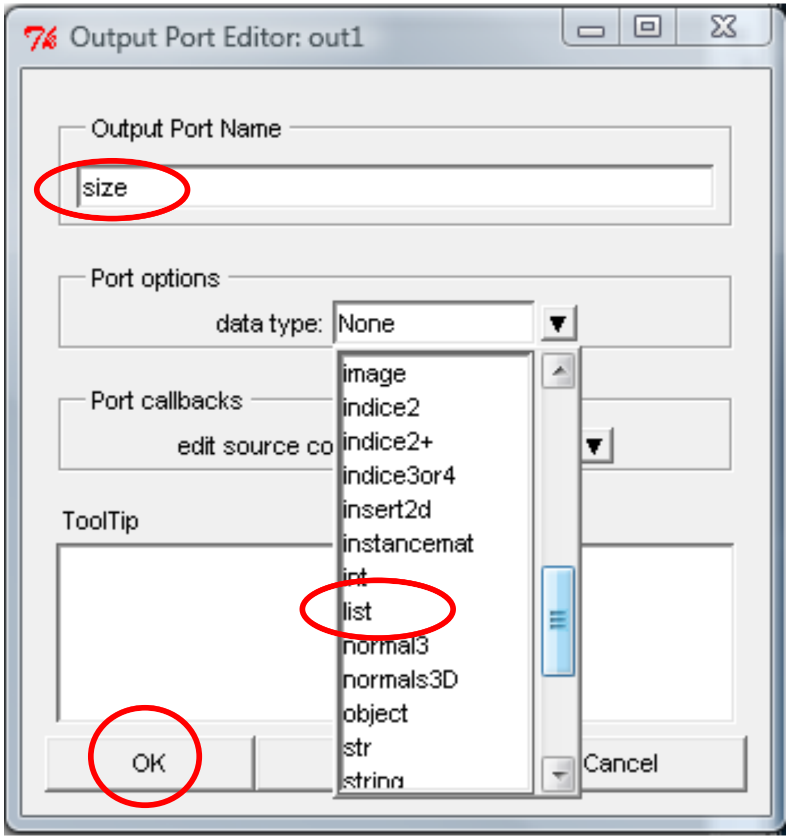
Exercise: Node editor

Task: rename the new port ‘size and change its data type to ‘list’

1 – Click ‘*edit port*’ checkbutton for ‘*out1*’
2 – change the name in the port editor
3 – set the type to ‘*list*’
4 – click OK



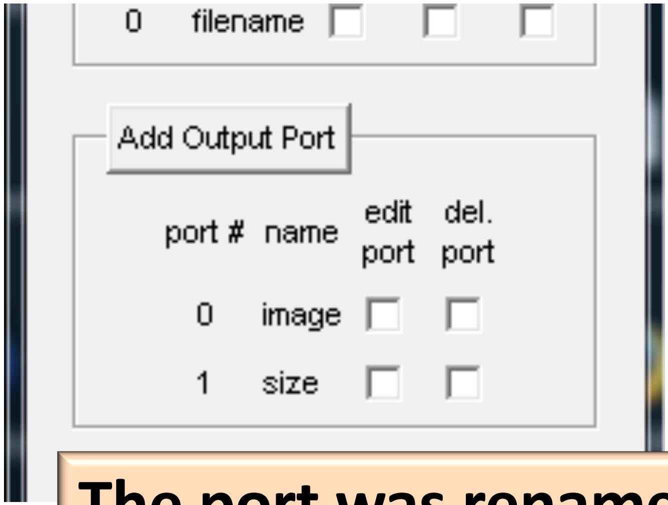
port #	name	edit port	del. port
0	image	□	□
1	out1	☑	□


Output Port Editor: out1

Output Port Name: size

Port options
data type: None

Notes: Node editor



0	filename	□	□	□

port #	name	edit port	del. port
0	image	□	□
1	size	□	□

The port was renamed

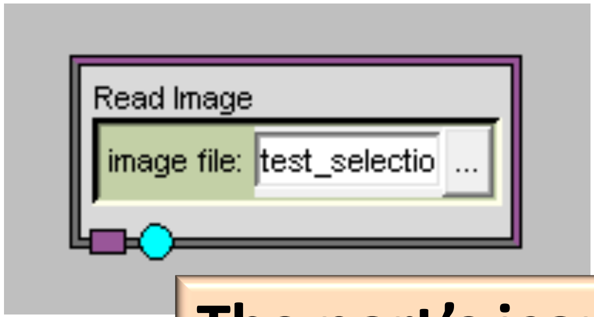
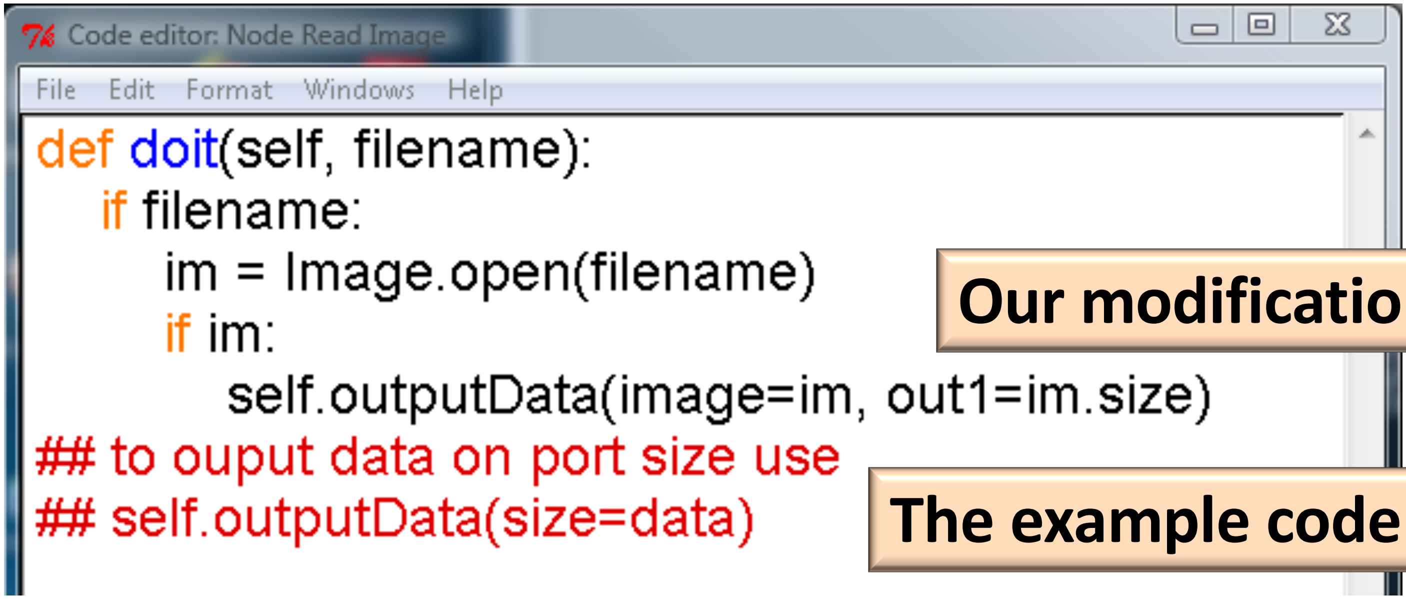


image file: test_selectio

The port’s icon has changed reflecting the list data type



```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, out1=im.size)
## to ouput data on port size use
## self.outputData(size=data)
```

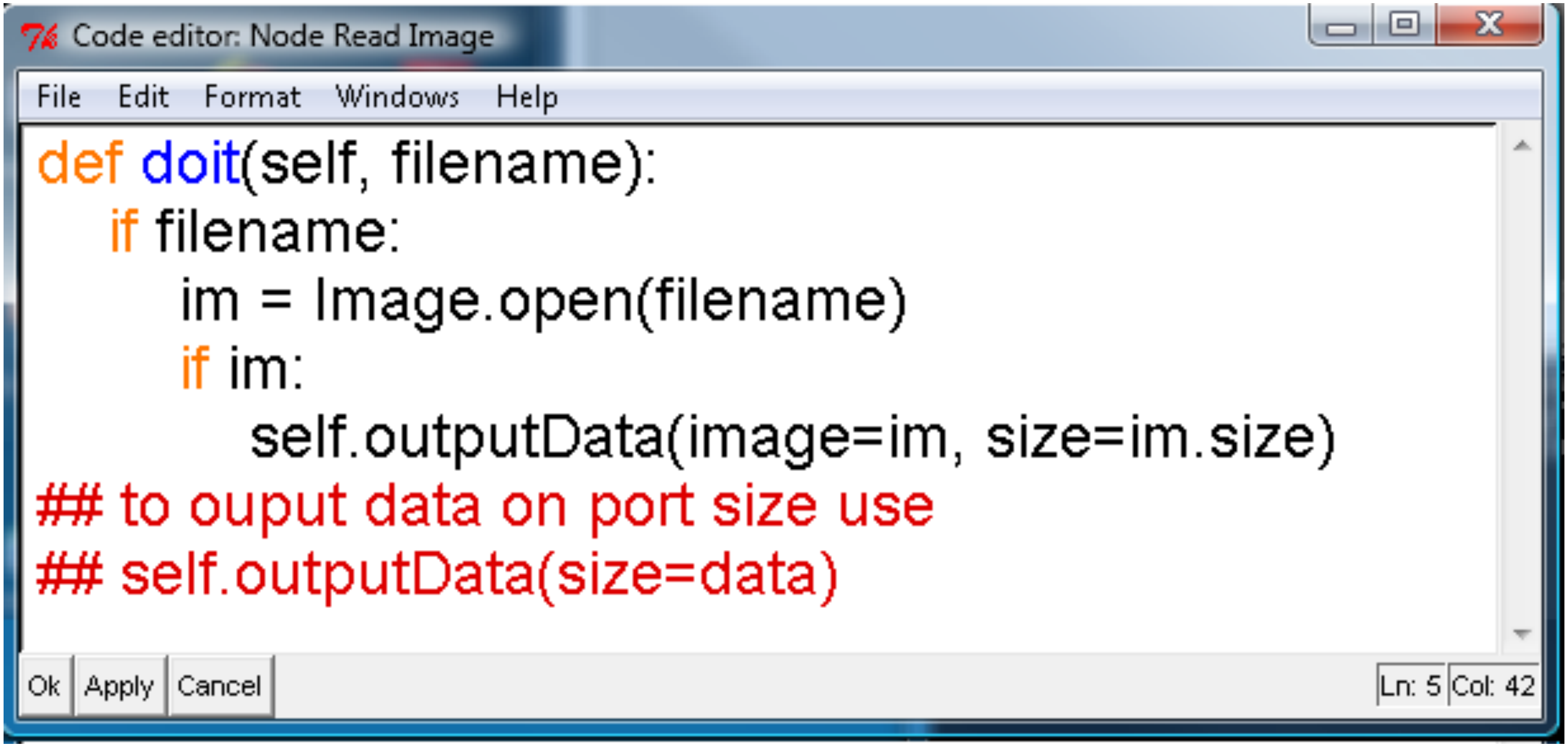
Our modification is unchanged

The example code was modified

Exercise: Node editor

Task: fix the function after renaming the 'out1' port

1 – replace '*out1*' by '*size*' in the function
2 – click Apply
3 – use port's tooltip to verify that the port name has changed



```
def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, size=im.size)
## to ouput data on port size use
## self.outputData(size=data)
```

Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon request

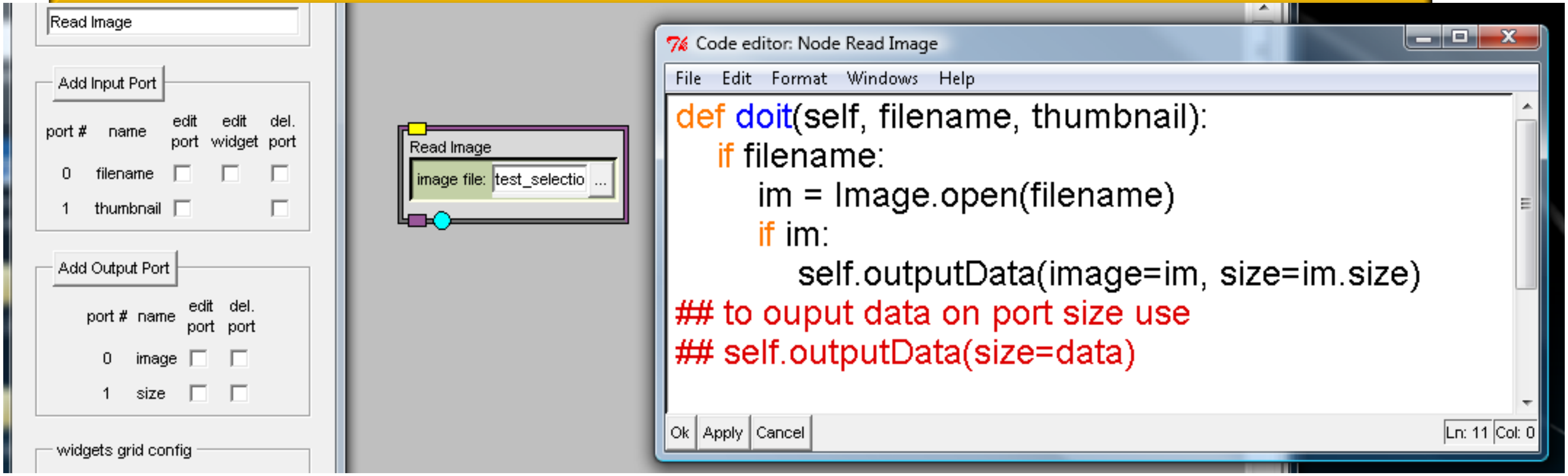
1 - Add an input port allowing to specify whether a thumbnail is wanted or not:

1.a – click on ‘Add Input Port’ (Note the new argument in the code editor, and the new port on the node’s icon)

1.b – click on ‘edit port’ for the newly added port

1.c – change the name to ‘thumbnail’ and the data type to ‘boolean’ in the port editor

1.d – click OK



port #	name	edit port	edit widget	del. port
0	filename	□	□	□
1	thumbnail	□		□

port #	name	edit port	del. port
0	image	□	□
1	size	□	□

```
def doit(self, filename, thumbnail):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im, size=im.size)
## to ouput data on port size use
## self.outputData(size=data)
```

Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon upon request

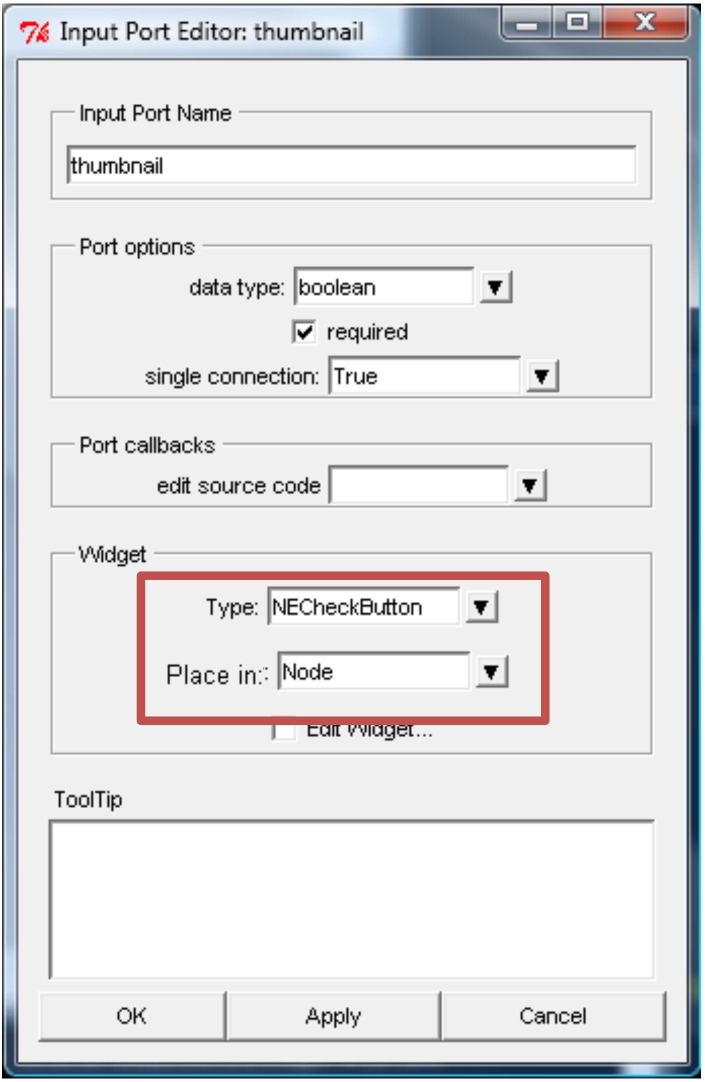
2 – Bind a check button widget to the new input port

2.a – click on ‘*edit port*’ for the ‘*thumbnail*’ input port

2.b – select NECheckButton for the type of widget

2.c – select ‘*node*’ for where to place the widget

2.d – click OK



Input Port Editor: thumbnail

Input Port Name

thumbnail

Port options

data type: boolean

☑ required

single connection: True

Port callbacks

edit source code

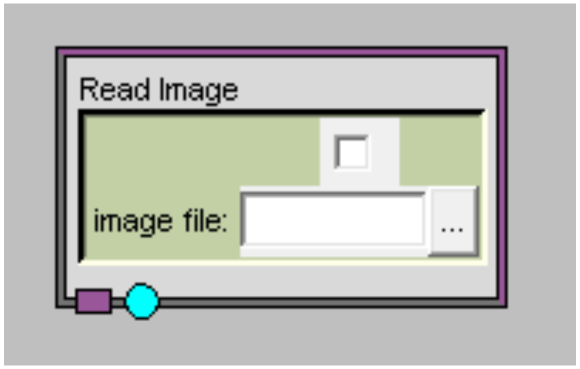
Widget

Type: NECheckButton

Place in:: Node

□ Edit Widget...

ToolTip



Exercise: Node editor

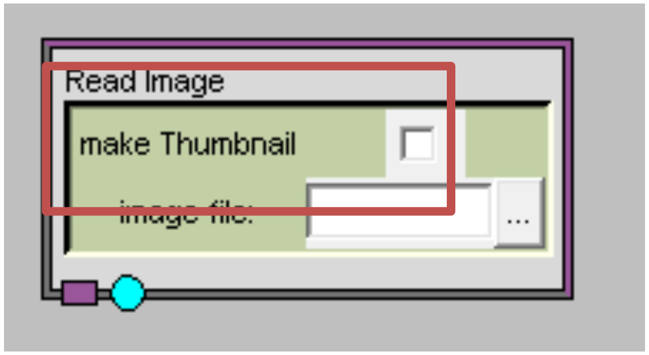
Task: add the ability to output a thumbnail of the image upon upon request

3 - Name the widget

3.a – Click on ‘*widget grid config*’ in the node editor

3.b – Type “*make Thumbnail*” in the name field

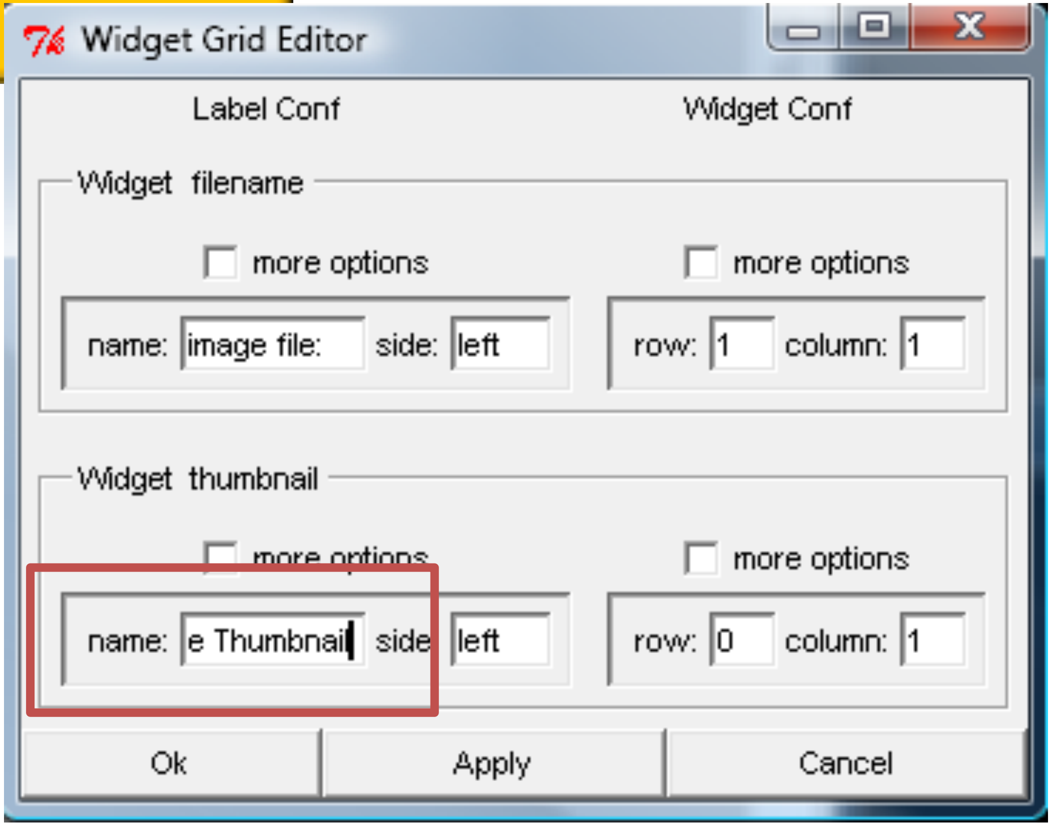
3.c – click OK



Read Image

make Thumbnail □

image file: ...



Widget Grid Editor

Label Conf | Widget Conf

Widget filename

□ more options | □ more options

name: image file: side: left | row: 1 column: 1

Widget thumbnail

□ more options | □ more options

name: e Thumbnail side: left | row: 0 column: 1

Exercise: Node editor

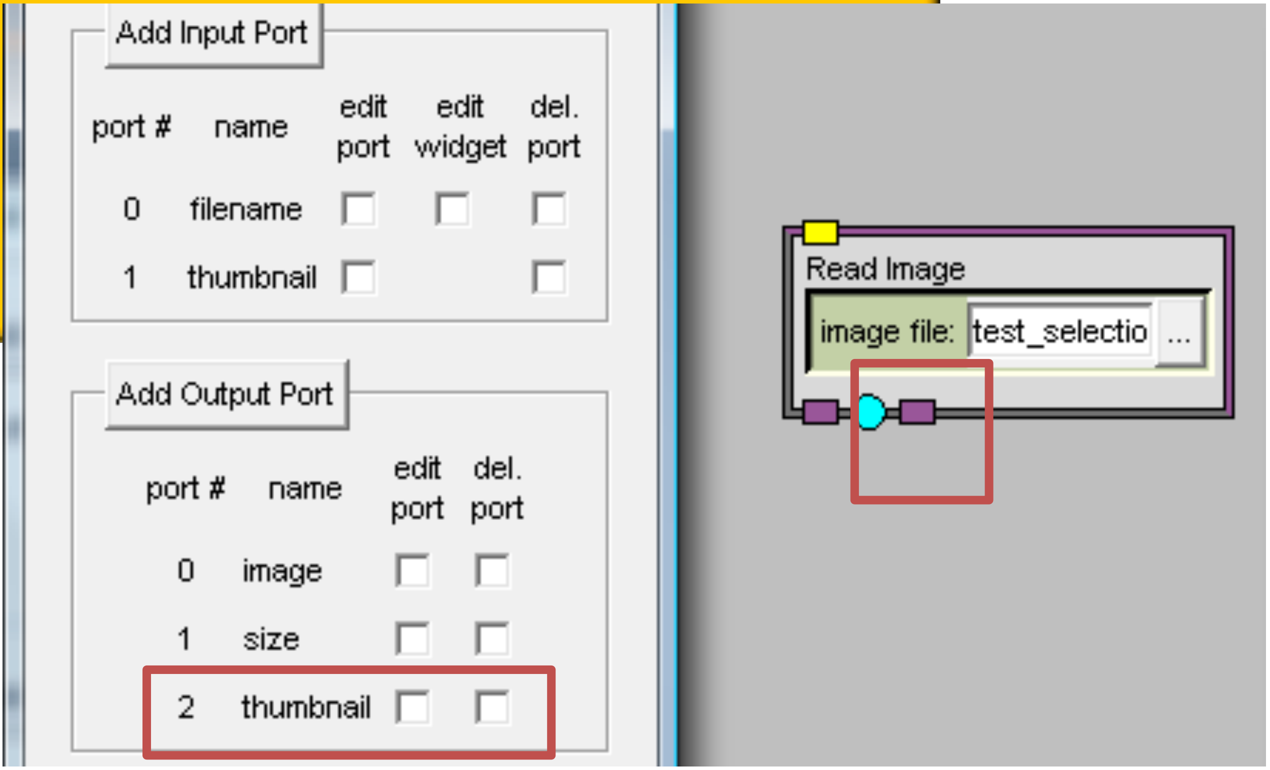
Task: add the ability to output a thumbnail of the image upon upon request

4 – add an output port for the thumbnail image

4.a – Click on ‘*Add Output Port*’ (Note new output port on the node’s icon)

4.b – Click on ‘*edit port*’ for the newly added port

4.c – change the name to ‘*thumbnail*’ and the data type to ‘*image*’ in the port editor



port #	name	edit port	edit widget	del. port
0	filename	□	□	□
1	thumbnail	□		□

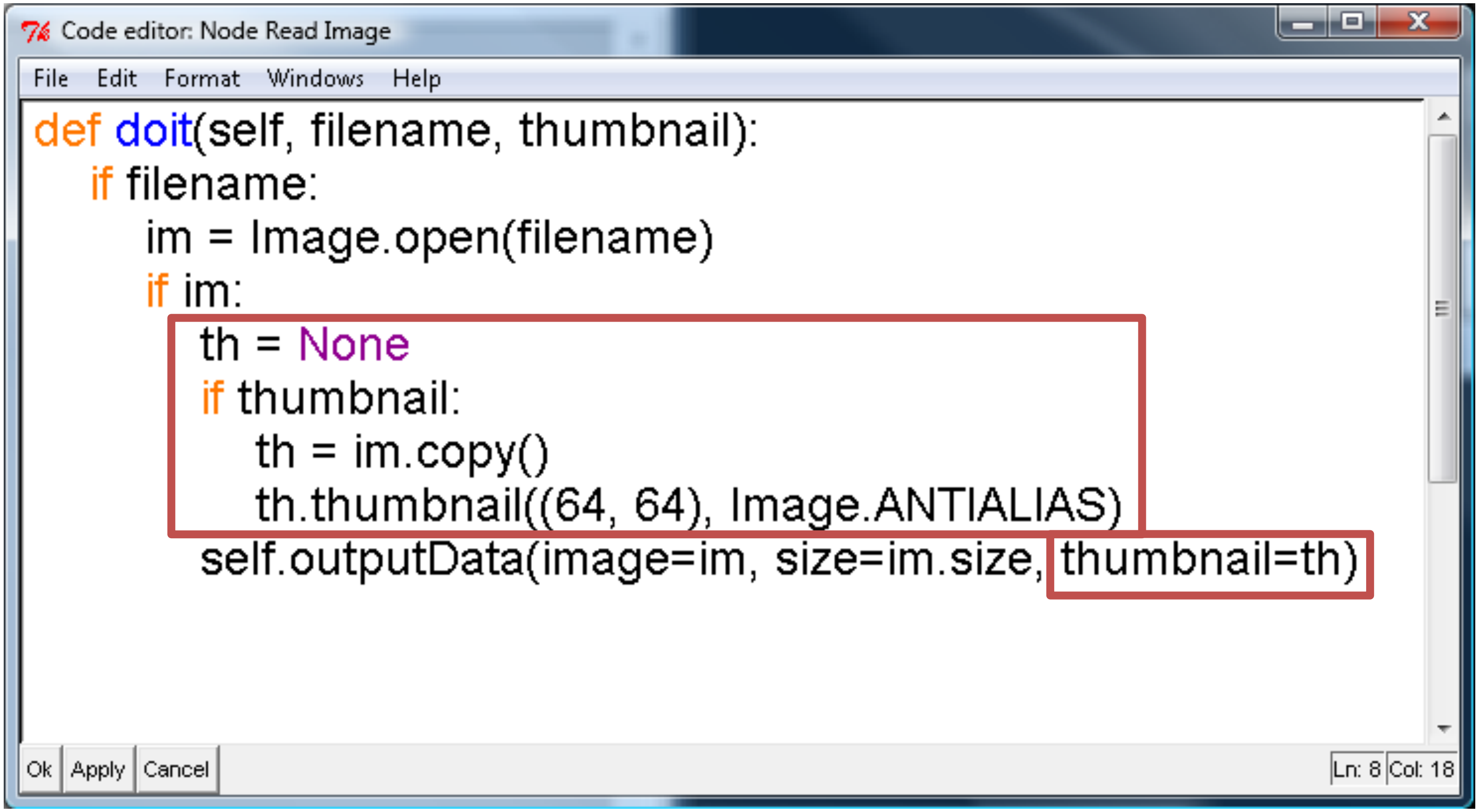
port #	name	edit port	del. port
0	image	□	□
1	size	□	□
2	thumbnail	□	□

image file: test_selectio

Exercise: Node editor

Task: add the ability to output a thumbnail of the image upon upon request

5 - Edit the function to use the thumbnail argument and click Apply
6 – use show image to look at thumbnail



```
def doit(self, filename, thumbnail):
    if filename:
        im = Image.open(filename)
        if im:
            th = None
            if thumbnail:
                th = im.copy()
                th.thumbnail((64, 64), Image.ANTIALIAS)
            self.outputData(image=im, size=im.size, thumbnail=th)
```

Exercise: Node editor

Task: save network with modified node

1 - use File -> Save …
2 - save as ‘modif_net.py’

Task: reload network and verify all modifications

1 – click on the load network icon in the tool bar 📂
2 – load the network called modif_net.py
3 – verify that all added input and output ports are restored
4 – verify the node’s function has the modifications

The loaded network as a trailing ‘1’ in its name to make the name unique

Exercise: Node editor

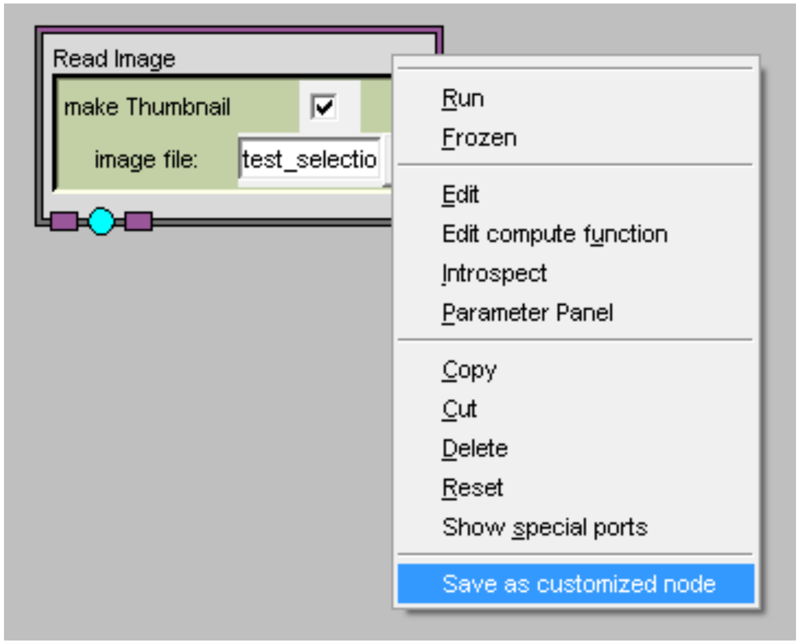
Task: make the thumbnail size a parameter controlled by a thumbwheel widget

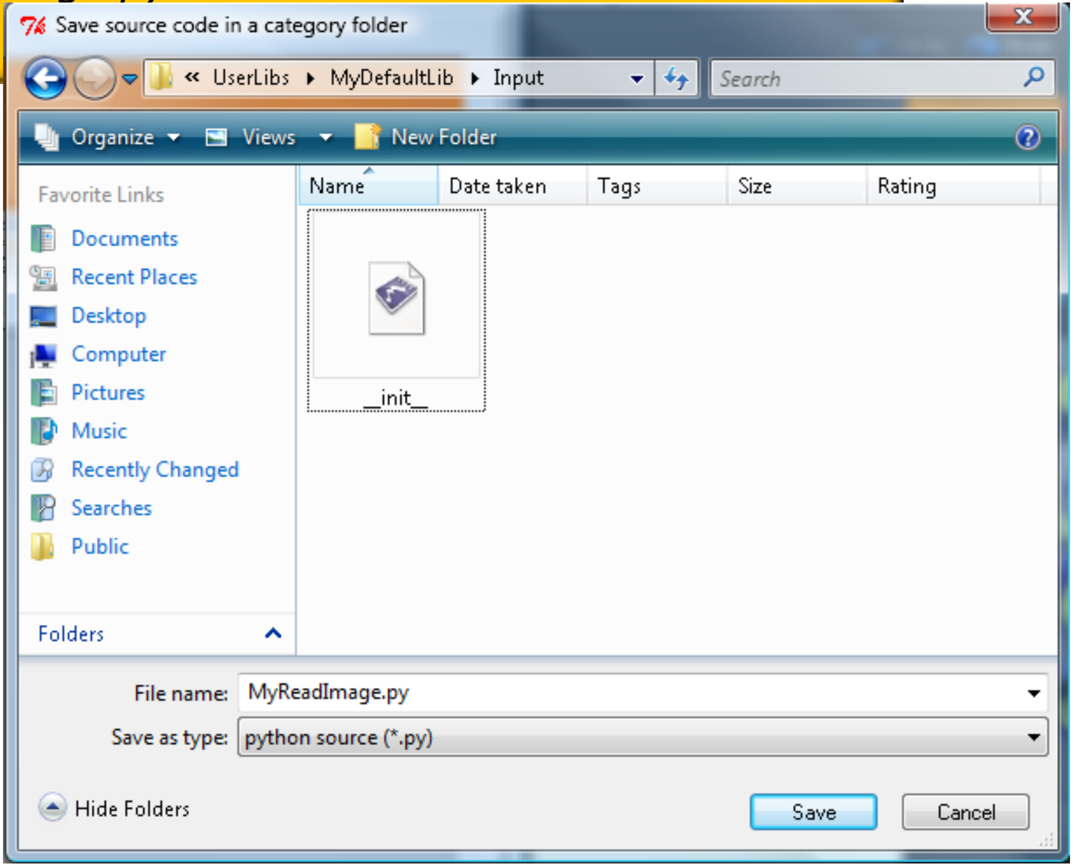
1 – add input port with good name and type integer
2 – bind a thumbwheel and place it in the node
3 – modify the function to use the value provided by the port

Exercise: User library

Task: save the modified Read Image node in our own library

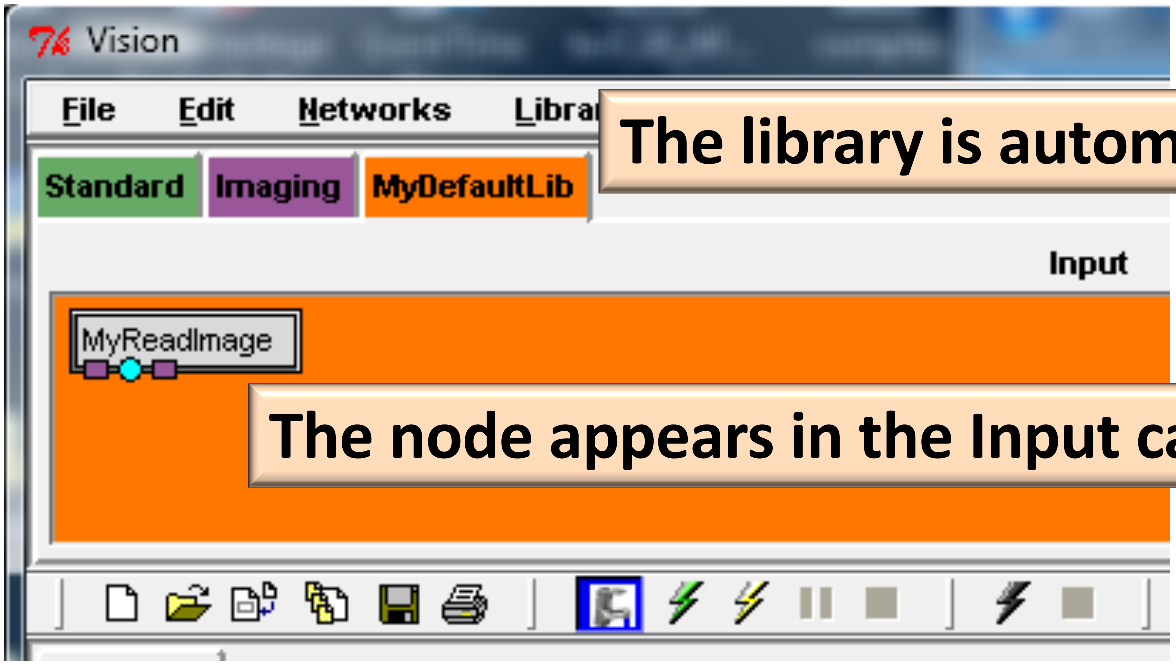
1 – delete modif1_net.py and modif_net.py
2 – right-click on the node's background and select "*save as customized node*"
2 – navigate the file browser to the Input folder of "*MyDefaultLib*"
3 – edit the file name to be "*MyReadImage.py"*
4 – click on Save





File name: MyReadImage.py
Save as type: python source (*.py)

Notes: User library



The library is automatically loaded

The node appears in the Input category

Exercise: Writing a node

Task: write the Read Image node starting from a node template

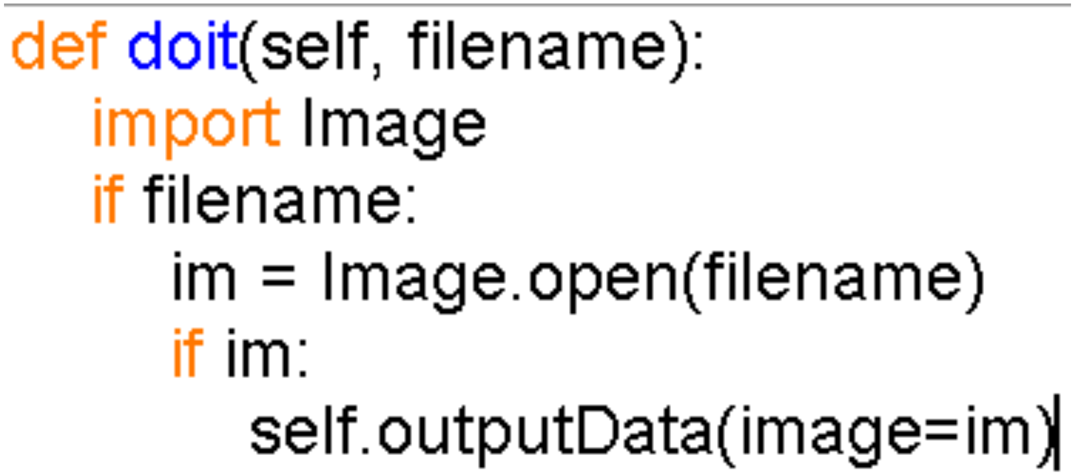
1 – locate and instantiate the *Generic* node

2 – use the node editor to re-create the *Read Image* node

- add an input port called '*filename*' of type string
- bind an NEEntryWithFileBrowser widget and place it in the node
- add the name '*filename*' to the widget
- add an output port called '*image*' of type image
- complete the node's function

3 – save node in MyDefaultLib - Input

The original Read Image node uses Image.open
Image is imported in the file defining the node
In your node you will have to import Image



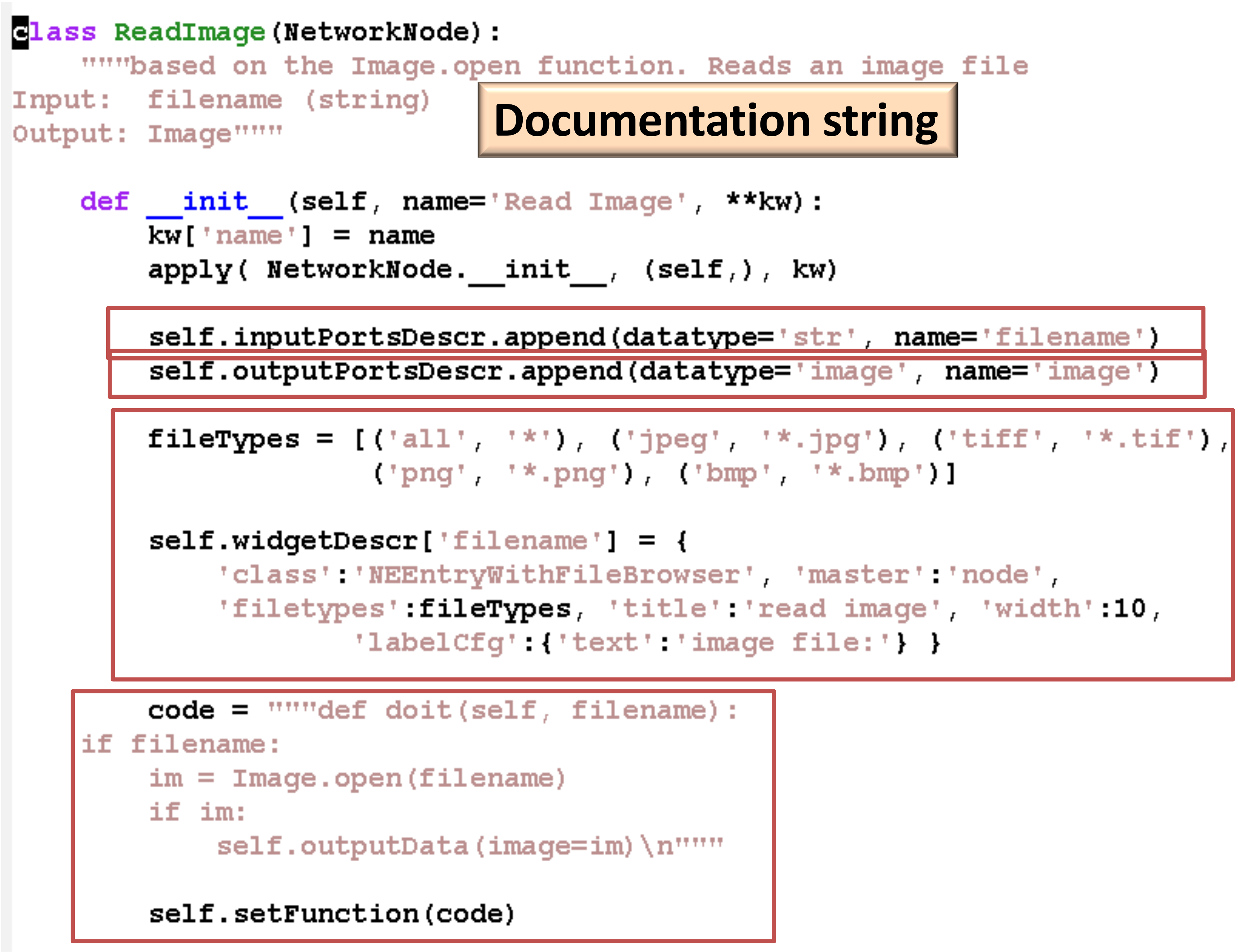
```
def doit(self, filename):
    import Image
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)
```

Exercise: Looking at the source code

Task: study a source code of the node we saved in the user library

1 – navigate to
C:\Documents and Settings\rctraining\.mgltools\latest\Vision\UserLibs\MyDefaultLib\Input
2 – right-click on MyReadImage.py and select open with IDLE

1 – navigate to
~/.mgltools/latest/Vision/UserLibs/MyDefaultLib\Input
2 – right-click on MyReadImage.py and select open with IDLE



Documentation string

```
class ReadImage(NetworkNode):
    """based on the Image.open function. Reads an image file
Input:  filename (string)
Output: Image"""

    def __init__(self, name='Read Image', **kw):
        kw['name'] = name
        apply( NetworkNode.__init__, (self,), kw)

        self.inputPortsDescr.append(datatype='str', name='filename')
        self.outputPortsDescr.append(datatype='image', name='image')

        fileTypes = [('all', '*'), ('jpeg', '*.jpg'), ('tiff', '*.tif'),
                     ('png', '*.png'), ('bmp', '*.bmp')]

        self.widgetDescr['filename'] = {
            'class':'NEEntryWithFileBrowser', 'master':'node',
            'filetypes':fileTypes, 'title':'read image', 'width':10,
                    'labelCfg':{'text':'image file:'} }

        code = """def doit(self, filename):
    if filename:
        im = Image.open(filename)
        if im:
            self.outputData(image=im)\n"""

        self.setFunction(code)
```

Exercise: User Panel

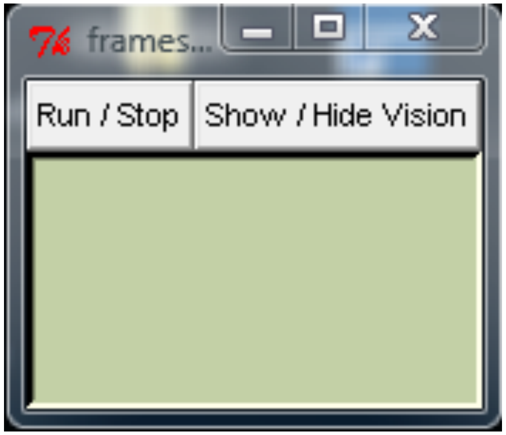
Task: create a panel that provides selected widgets from a network

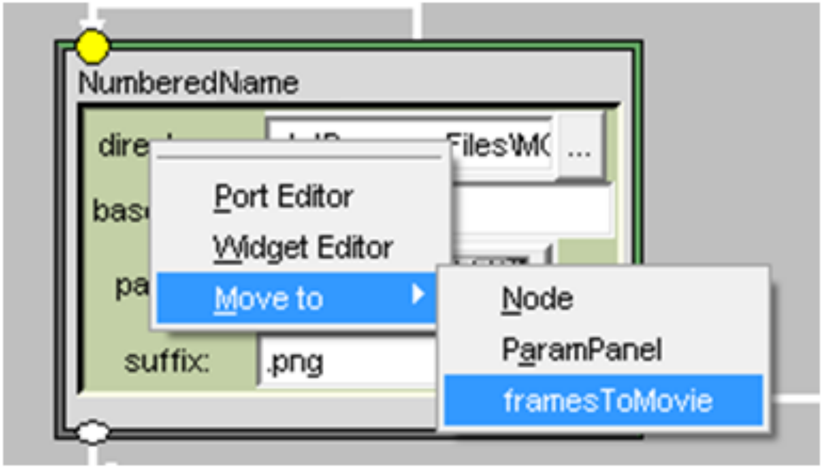
1 – load the saved network “*saveMPEG_net.py*”

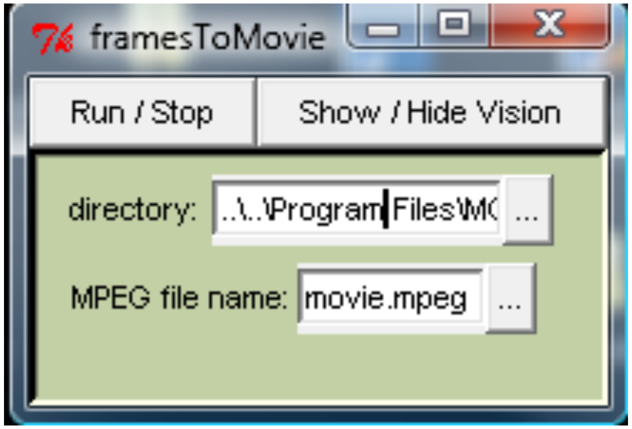
2 – create a user Panel using *Edit -> Create user panel*
(name the panel FramesToMovie)

3 – use the widget menu to move the following widgets to the panel
directory from NumberedName node
MPEG file name from the Record MPEG movie node

4 – middle-click on widget in panel and drag to move it around



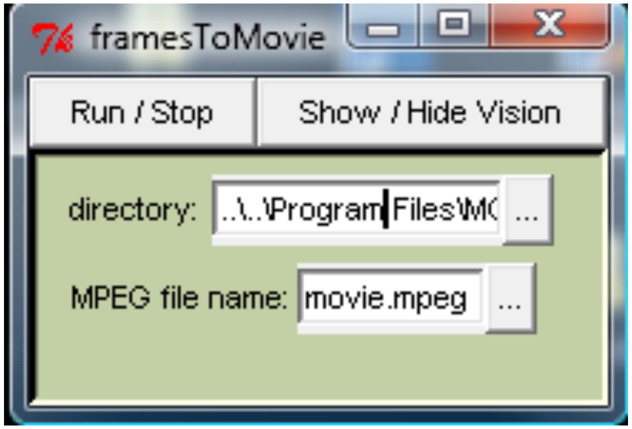




directory: ..\..\Program Files\MC

MPEG file name: movie.mpeg

Notes: User Panel



directory: ..\..\Program Files\MC

MPEG file name: movie.mpeg

The panels allows:
- running and stopping the execution
- showing/hiding the network

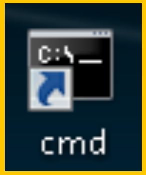
Task: save network as saveMPEGPanel_net.py to Desktop

Exercise: command line execution

Task: run the network from the command line using Vision

1 – save the network with the panel as “*saveMPEGPanel_net.py*”

2 – start a DOS command



3 – In the DOS window type:

```
vision  ..\saveMPEGPanel_net.py
```

- **Use the Tab key for automatic completion**
- **Note that the network comes up with the Vision GUI**

Exercise: Network Execution

Task: run a network from the command line as a program

1 – In the DOS window type:

```
..\saveMPEGPanel_net.py  - - help or –h
```

The help message display info about command line options including parameters that can be set From the command line.

2 – In the Dos window type:

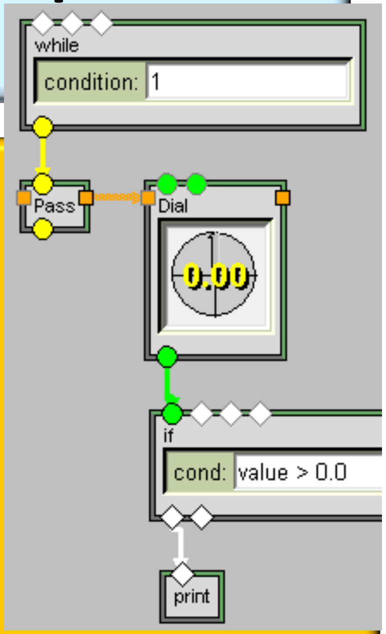
```
..\saveMPEGPanel_net.py
```

Only the parameter panels comes up

Exercise: endless loop

Task: build a network with an endless loop that will print the value of a dial if the value is positive

1 – create a new network using the button bar
2 – turn off immediate mode using the button bar
3 – locate and instantiate a *while* node and set the condition to 1
4 – locate and instantiate a *pass* node
5 – connect the output of *while* to the *pass* node
6 – right click on the *pass* node and select “*show special ports*”
7 – locate and instantiate a *Dial* node
8 – right click on the *Dial* node and select “*show special ports*”
9 – connect the “*trigger*” special port of *Pass* to the “*run*” special port of *Dial*
8 – locate and instantiate an “*If*” node and set the condition to “*value > 0.0*”
10 – connect the *dial* output to the *value* in put port of the *If* node
11 – locate and instantiate a *print* node
12 – connect the _*if* output port of the *If* node to the *print* node
13 – turn immediate mode back on
14 – save network as *endlessPrint_net.py*



condition: 1

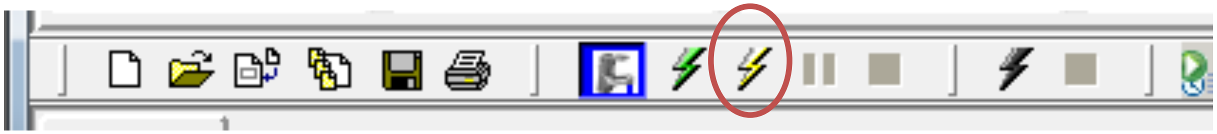
cond: value > 0.0

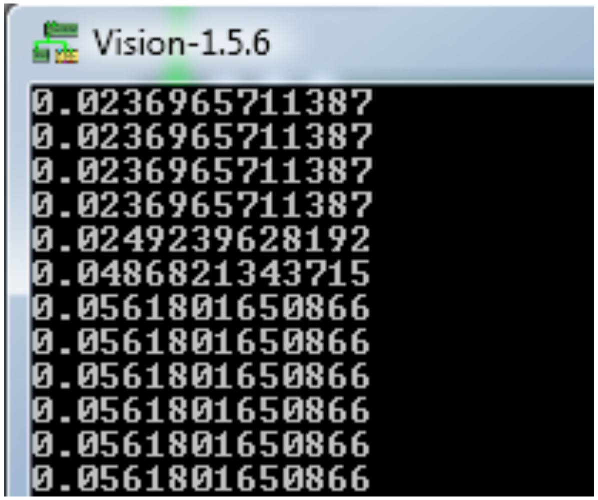
Exercise: endless loop

Task: run the network in Vision

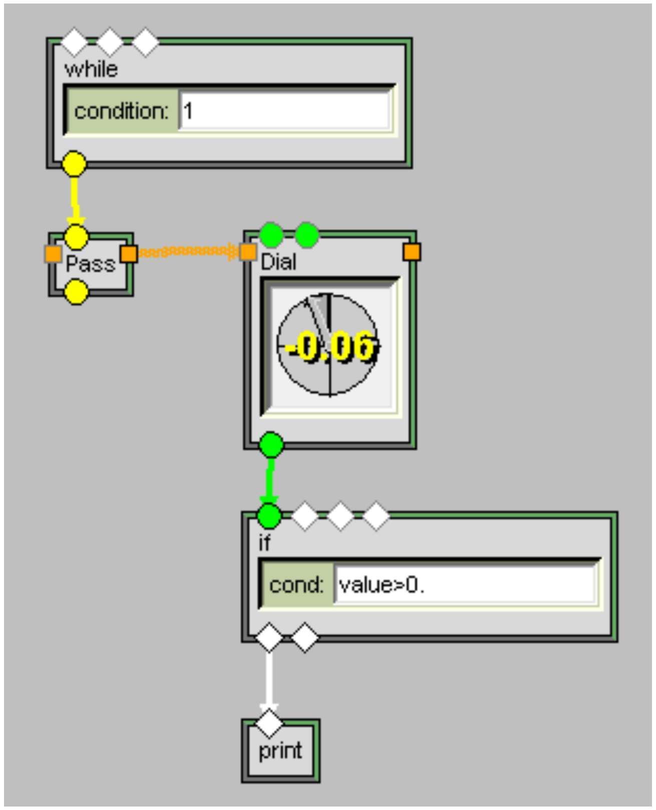
1 – click on the run network button in the tool bar

2 – make the dial positive to print to the shell or negative to stop printing





```
0.0236965711387
0.0236965711387
0.0236965711387
0.0236965711387
0.0249239628192
0.0486821343715
0.0561801650866
0.0561801650866
0.0561801650866
0.0561801650866
0.0561801650866
0.0561801650866
```



condition: 1

cond: value>0.

Exercise: endless loop

Special ports:

- Left side of the node (*run*) receives signal

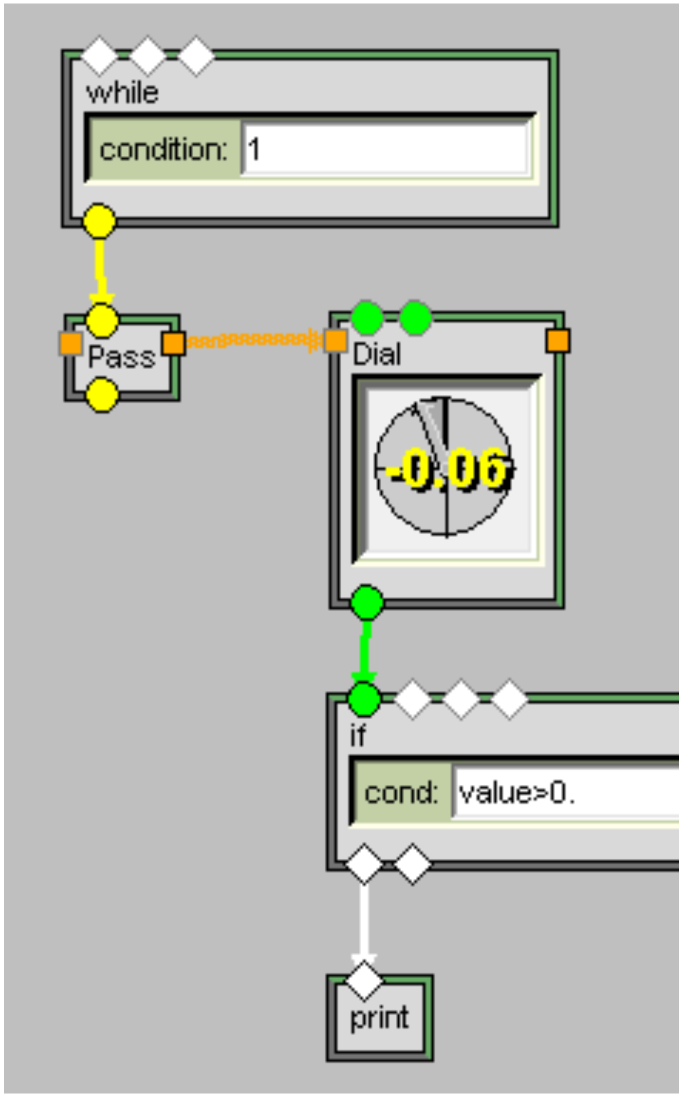
- Right side (*trigger*) sends a signal after running

Allow to trigger Dial execution each Time Pass runs without actually Passing data from Pass to Dial

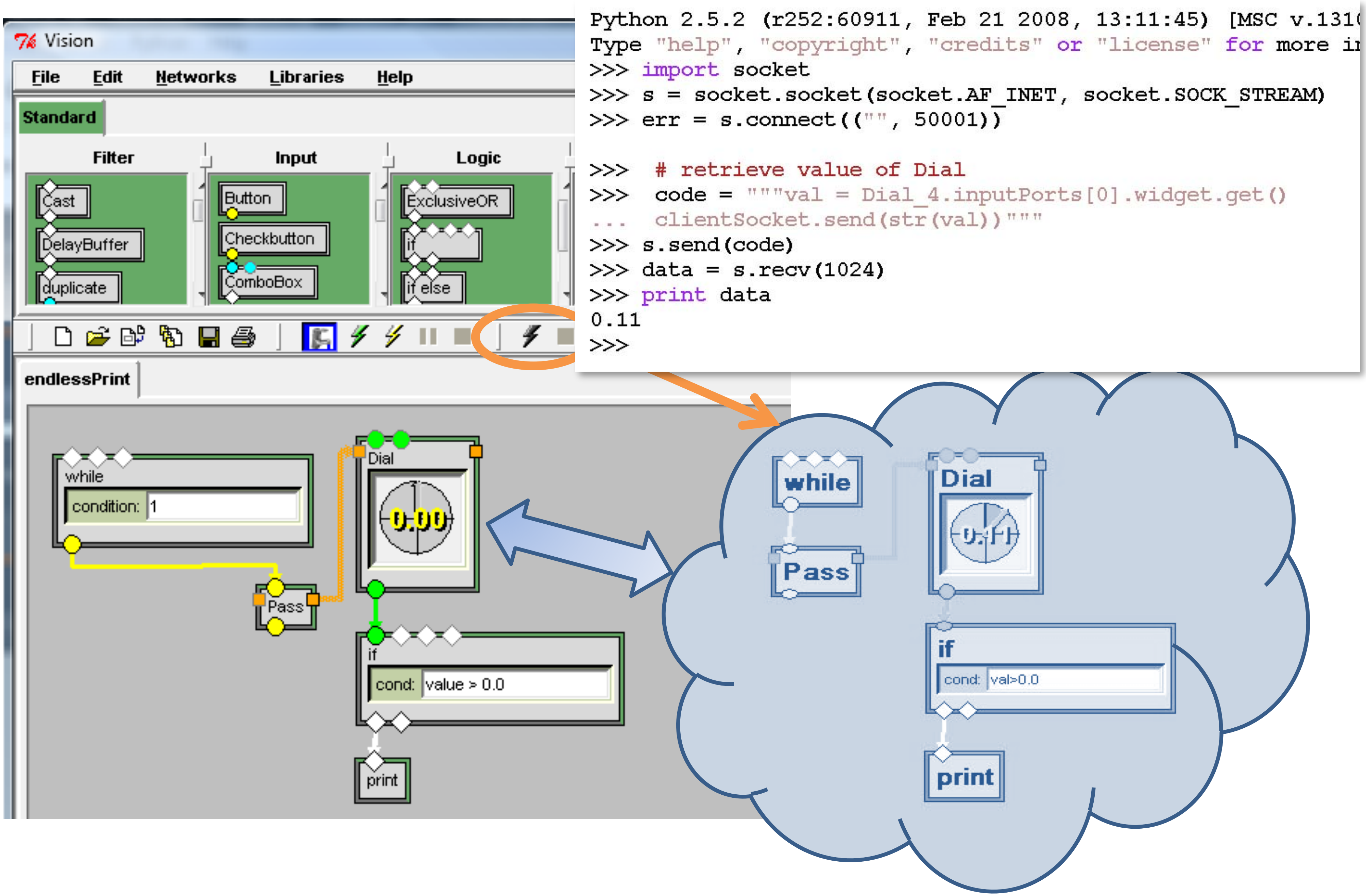
Task: run the while node

When the dial is < 0.0 the print node does not flash, when you move the dial to a positive value it flashes

Task: stop network execution



Notes: detached execution



```
Python 2.5.2 (r252:60911, Feb 21 2008, 13:11:45) [MSC v.131
Type "help", "copyright", "credits" or "license" for more in
>>> import socket
>>> s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> err = s.connect(("", 50001))

>>>  # retrieve value of Dial
>>>  code = """val = Dial_4.inputPorts[0].widget.get()
...  clientSocket.send(str(val))"""
>>> s.send(code)
>>> data = s.recv(1024)
>>> print data
0.11
>>>
```

condition: 1

cond: value > 0.0

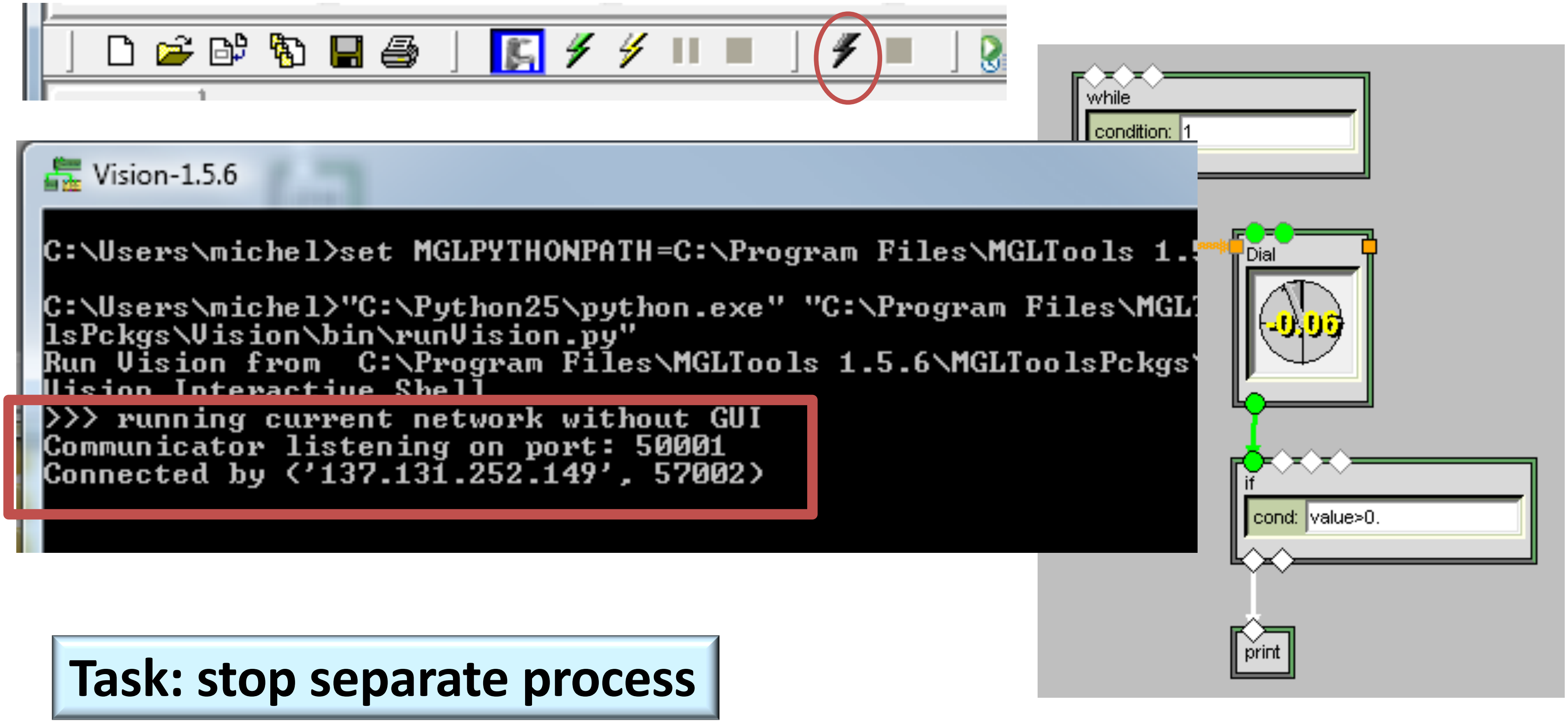
cond: val>0.0

Exercise: detached execution

Task: run the network in a separate process

1 – click on the run detached button in the tool bar ⚡

2 – make the dial positive to print to the shell or negative to stop printing



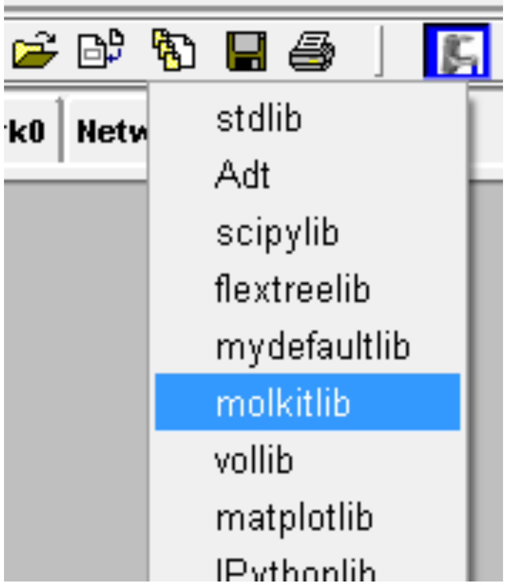
```
C:\Users\michel>set MGLPYTHONPATH=C:\Program Files\MGLTools 1.5

C:\Users\michel>"C:\Python25\python.exe" "C:\Program Files\MGLT
lsPckgs\Vision\bin\runVision.py"
Run Vision from  C:\Program Files\MGLTools 1.5.6\MGLToolsPckgs\
Vision Interactive Shell
>>> running current network without GUI
Communicator listening on port: 50001
Connected by ('137.131.252.149', 57002)
```

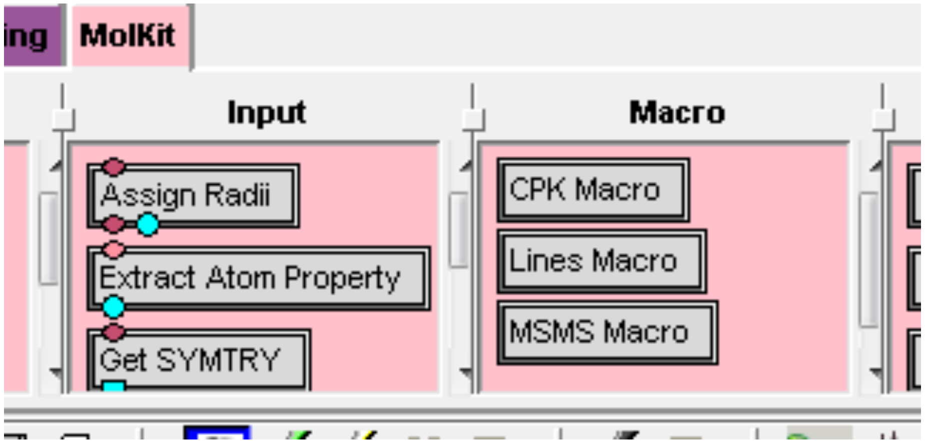
Task: stop separate process

Macro nodes

- Macro nodes represent a network as a single node in a parent network
- Data can be passed into the macro and come out of it
- Macros can be nested
- Macros can be added to libraries of Vision nodes

Task: load the MolKit library





Macro nodes

Task: use a Lines Macro to display a molecule

1 – locate and instantiate a *Read Molecule* node

2 – locate and instantiate a *Lines Macro* node

The macro output date of type geometry defined in the 3D Visualization library which is pulled in automatically

3 – locate and instantiate a *Viewer* node

A window associate to the Viewer node (black window) is created. It can be used to display 3D geometry.

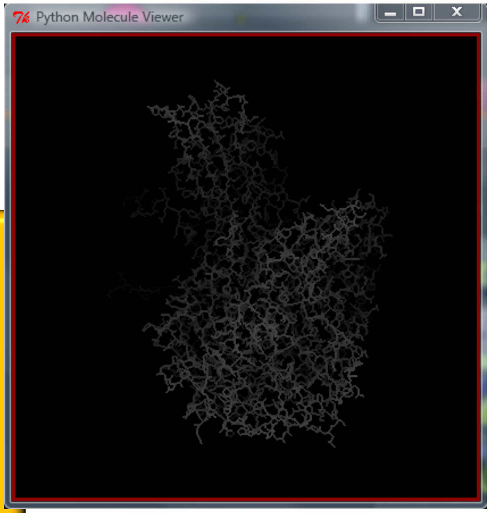
4 – connect the Read molecule output to the macro input

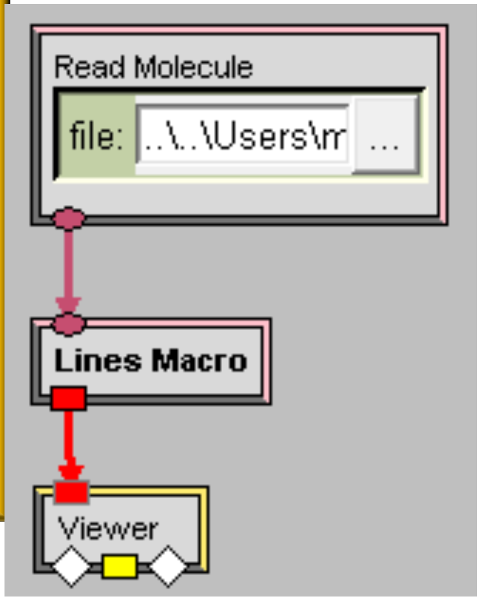
5 – connect the macro output to the *Viewer*

6 – read the molecule TutorialData/2plv.pdb from the desktop

7 – click in the 3D viewer (black window) and type the letters r n c d

8 – use middle mouse button to rotate the molecule





Macro nodes

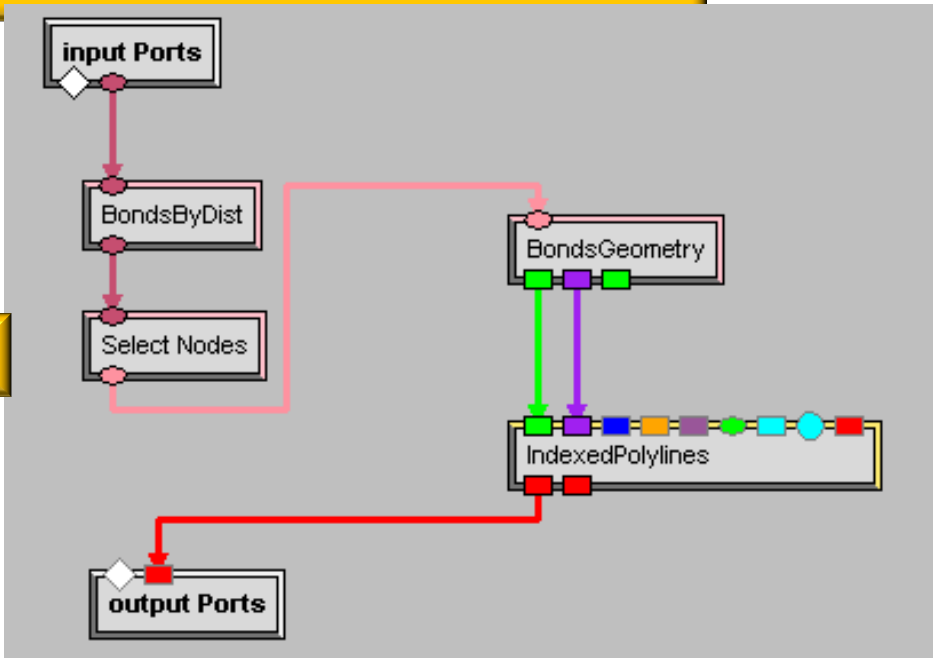
Task: open the macro node

1 – Shift double click on the macro node (or right click and select expand)

A new network called "Lines Macro" is displayed.

2 – double-click on all expanded nodes to collapse them (hide widgets)

Macro networks have 2 special nodes "input Ports" and "output Ports". These nodes allow data to enter and exit the macro.



Task: close the macro node

1 – double click on the *"input Port"* node

Macro nodes

Task: create a new macro node

1 – *Edit -> create macro* or *Ctrl-m*

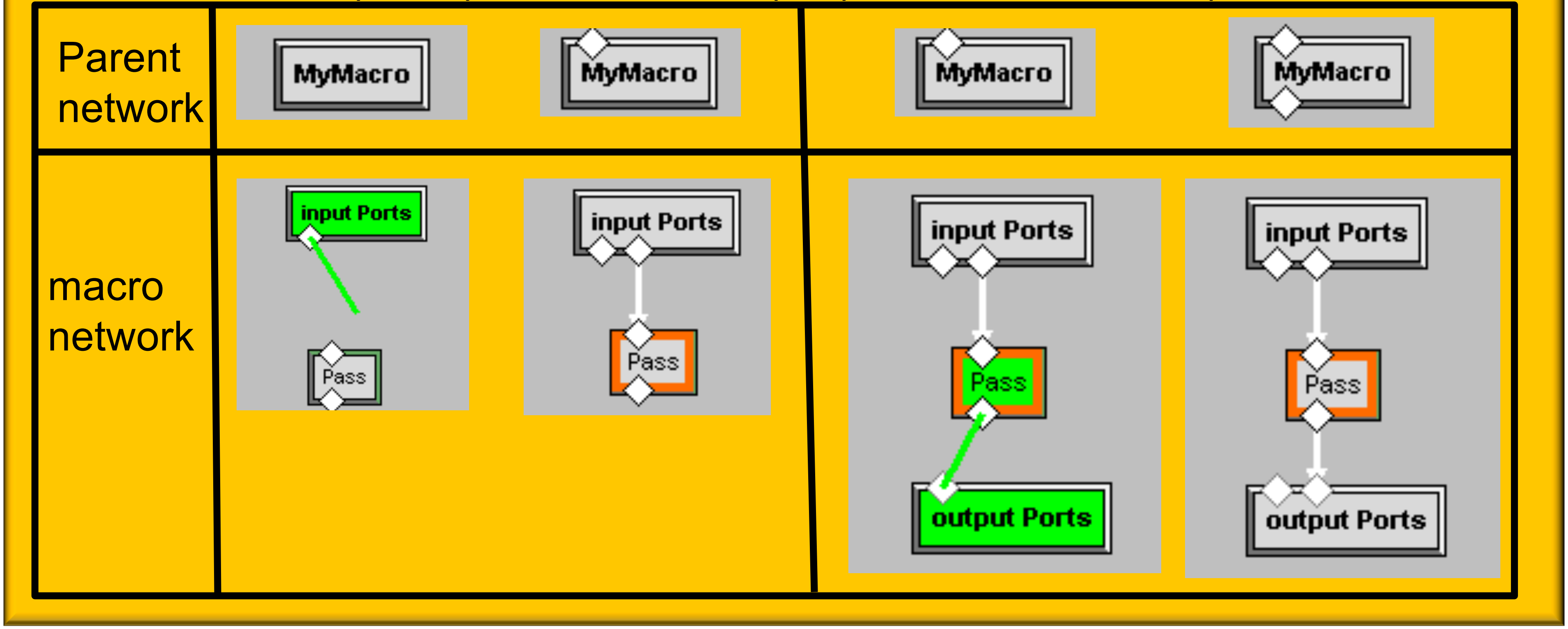
2 – name the macro "*MyMacro*"

You are automatically taken inside the macro network

3 – locate and instantiate a *pass* node inside the macro

4 – connect the first output port of the Macro *input Port* node to the input of *pass*

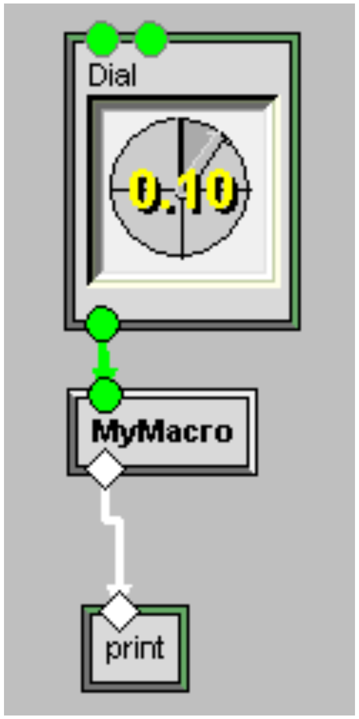
5 – connect the output of *pass* to the first input port of the Macro *output Port* node



Macro nodes

Task: create a new macro node

In the parent network:
1 – instantiate a *Dial* node and send the value into the macro
2 –instantiate a *print* node and connect the output of the macro



Section 3: Overview

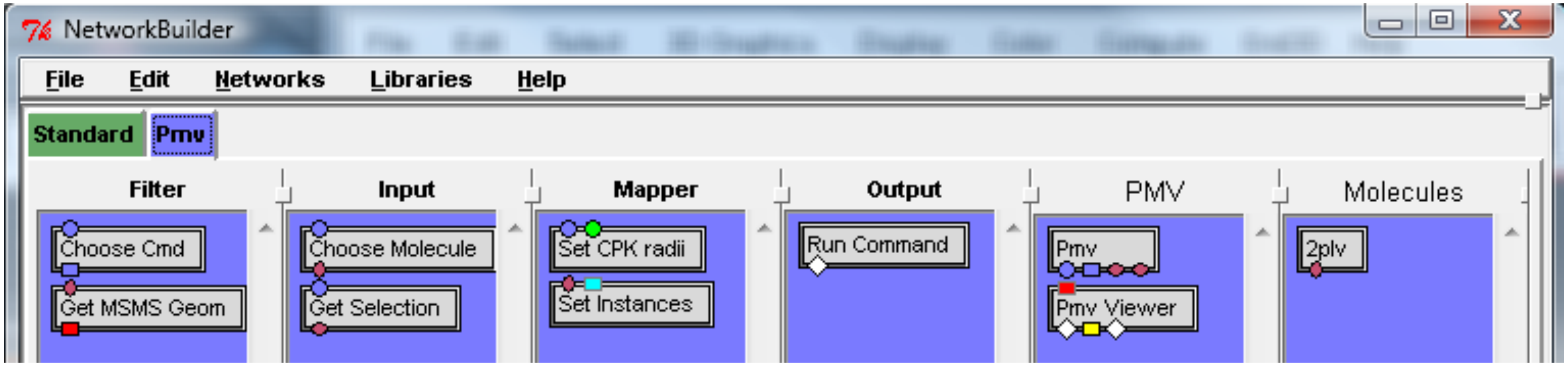
- Introduction to Pmv
- Building a viral capsids
- Running PMV commands in a Vision network
- Creating new PMV commands

Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

1 – Start Pmv
2 – right-click on PMV Molecules and load Desktop/TutorialData/2plv.pdb
3 – select backbone atoms in 2plv
4 – invert selection in 2 plv
5 – undisplay lines for selection
6 – clear the selection
7 – start Vision by clicking on the Vision button in the toolbar

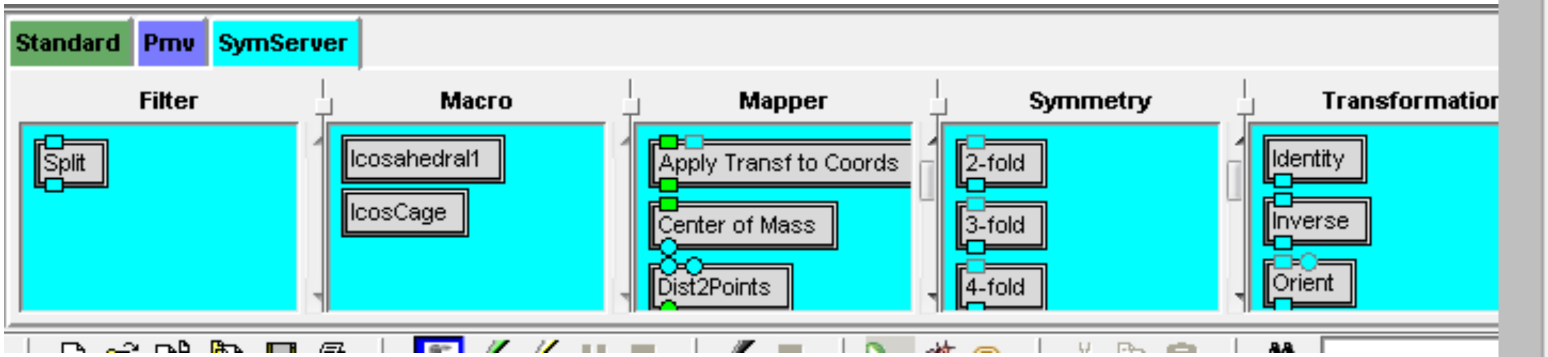
The Pmv node library has Vision nodes specific to PMV

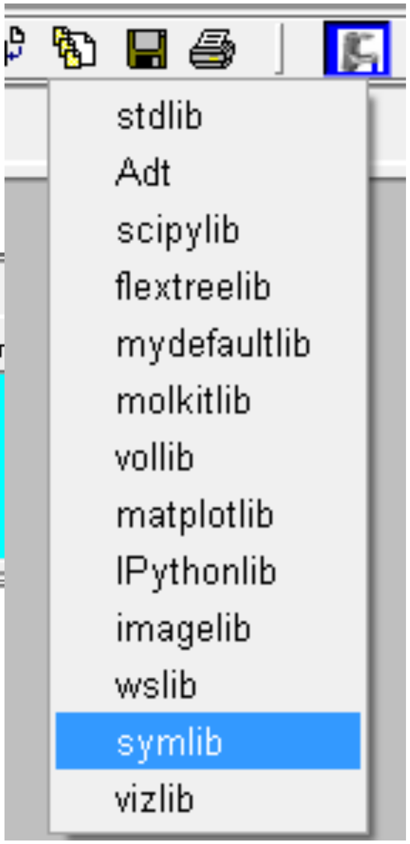


Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

4 – load the symmetry server library of Vision nodes



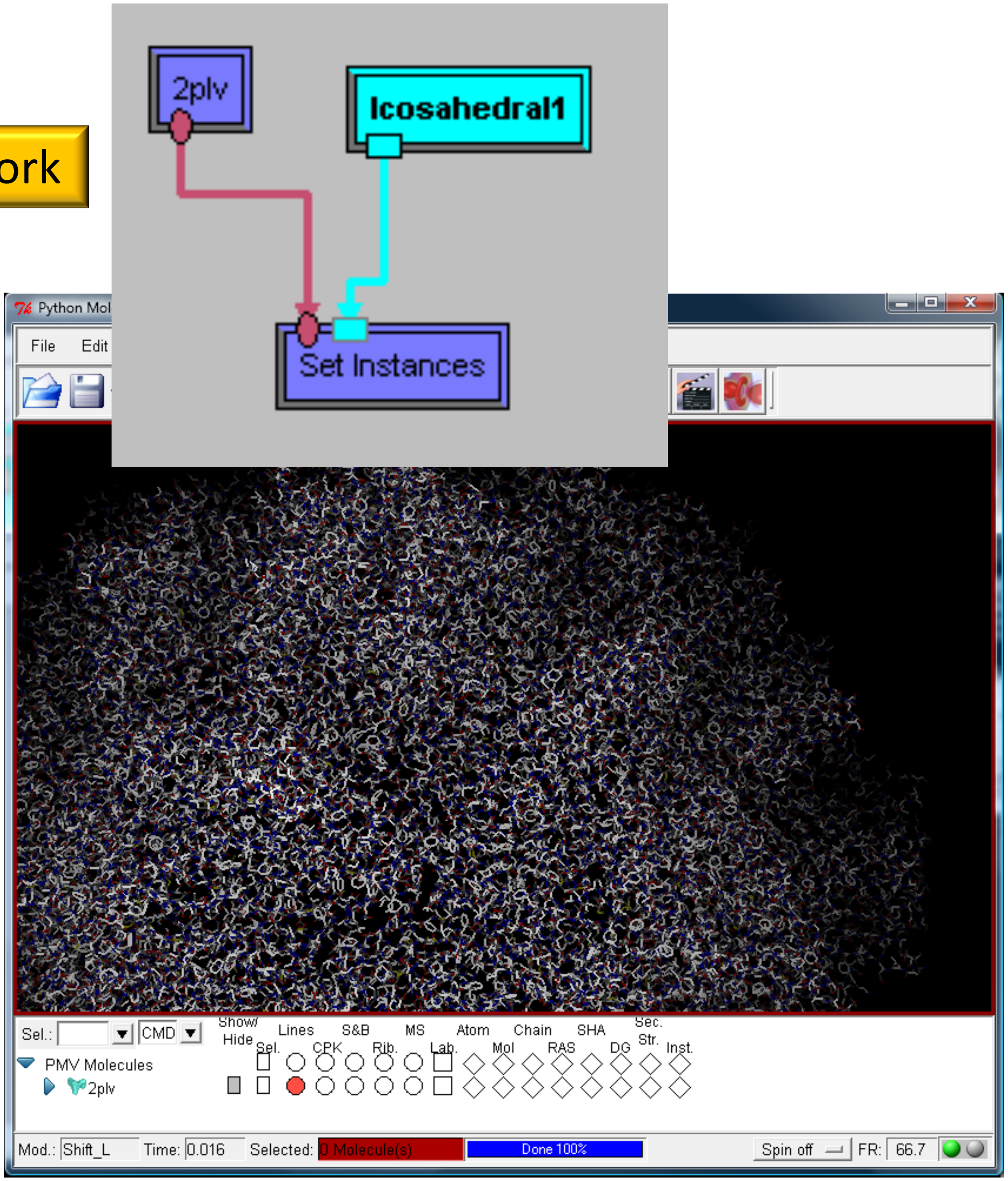


Exercise: Viral capsid

Task: build the viral capsid of Polio Virus

5 – build the following network

6 – run the network



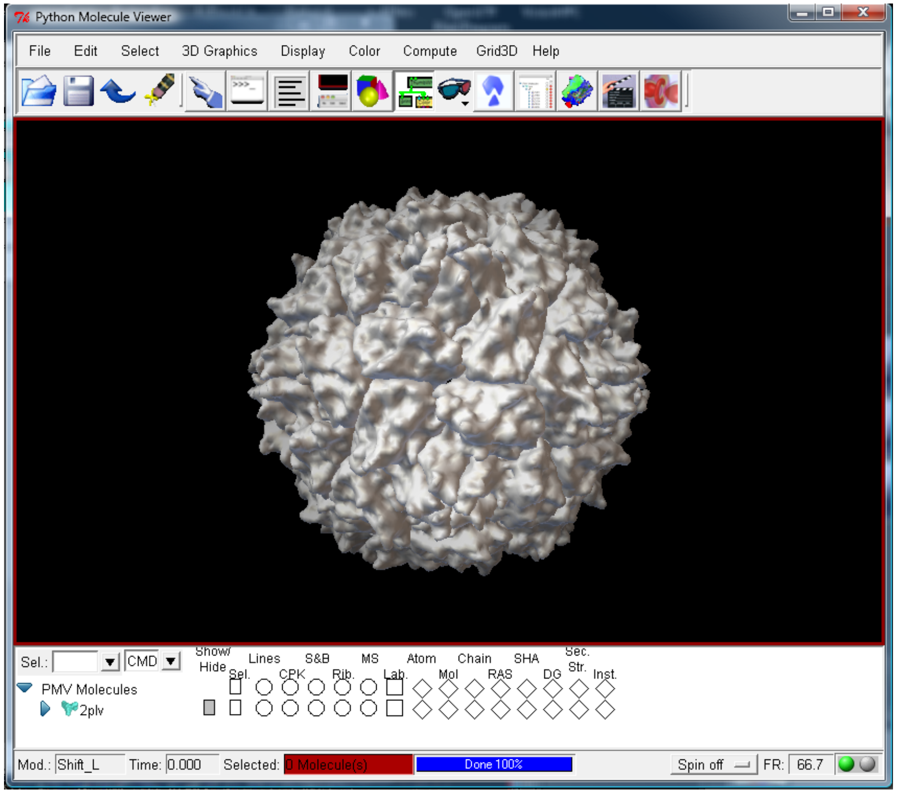
Exercise: Viral capsid

Task: change the representation from lines to a coarse molecular surface

1 – using the dashboard un-display the lines for 2plv

2 – execute the command Compute -> Coarse Molecular Surface from PMV menu using default parameters

3 - place cursor on 3D viewer window and type ‘r’, ‘n’, ‘c’



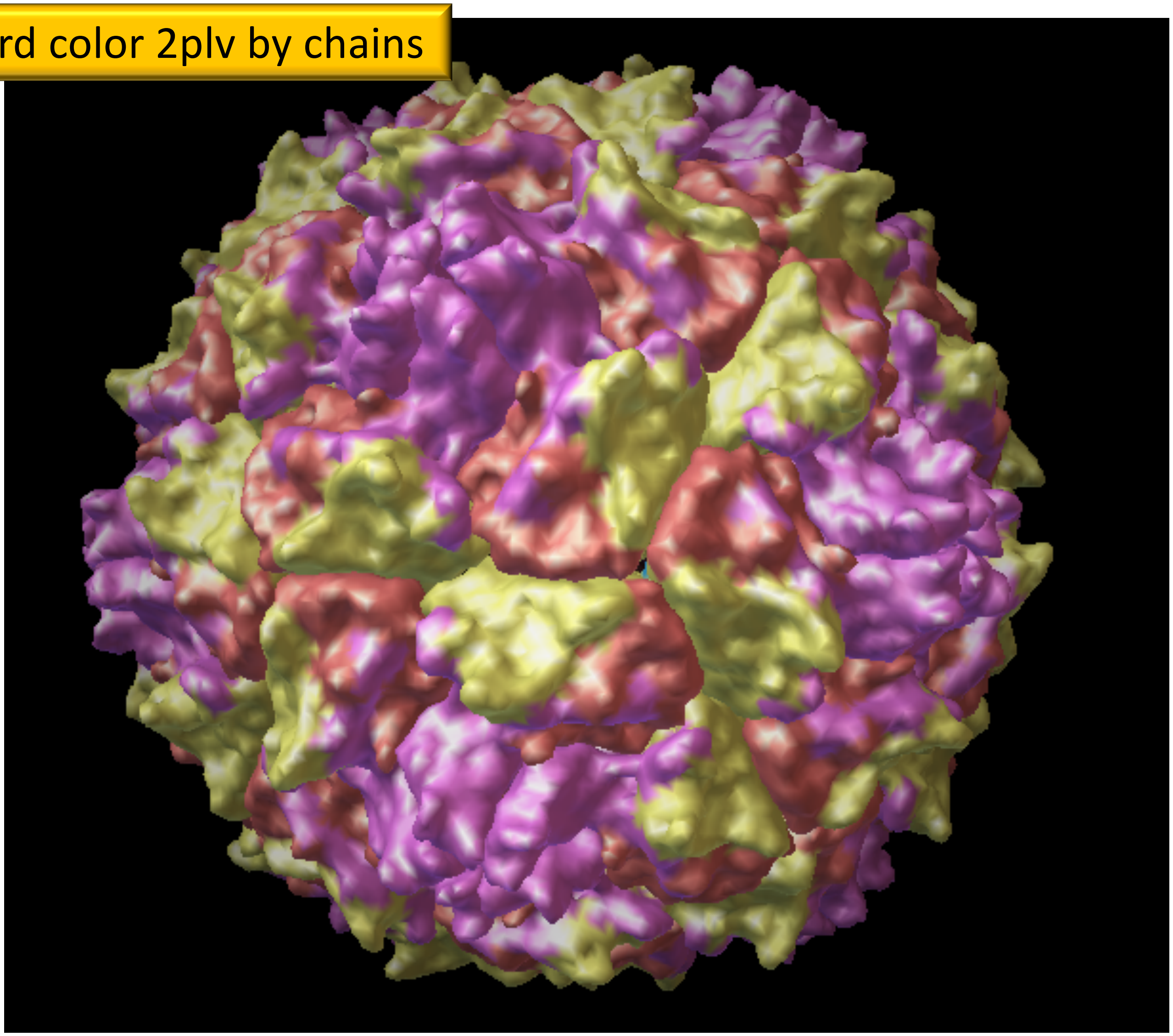
Mod.: Shift_L Time: 0.000 Selected: 0 Molecule(s) Done 100% FR: 66.7

The compute coarse molecular surface command is implemented as a Vision networks that is loaded the first time the command runs

Exercise: Viral capsid

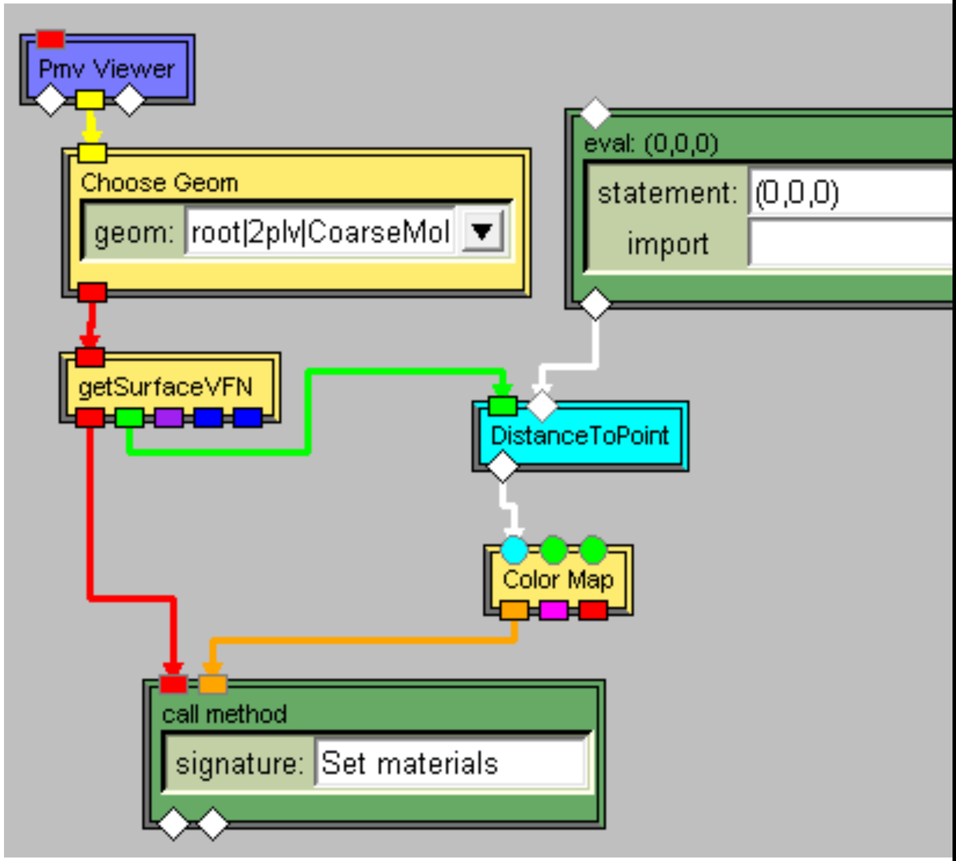
Task: color the surface by chain

1 – using the dashboard color 2plv by chains



Exercise: Viral capsid

Task: color the surface by depth in the capsid

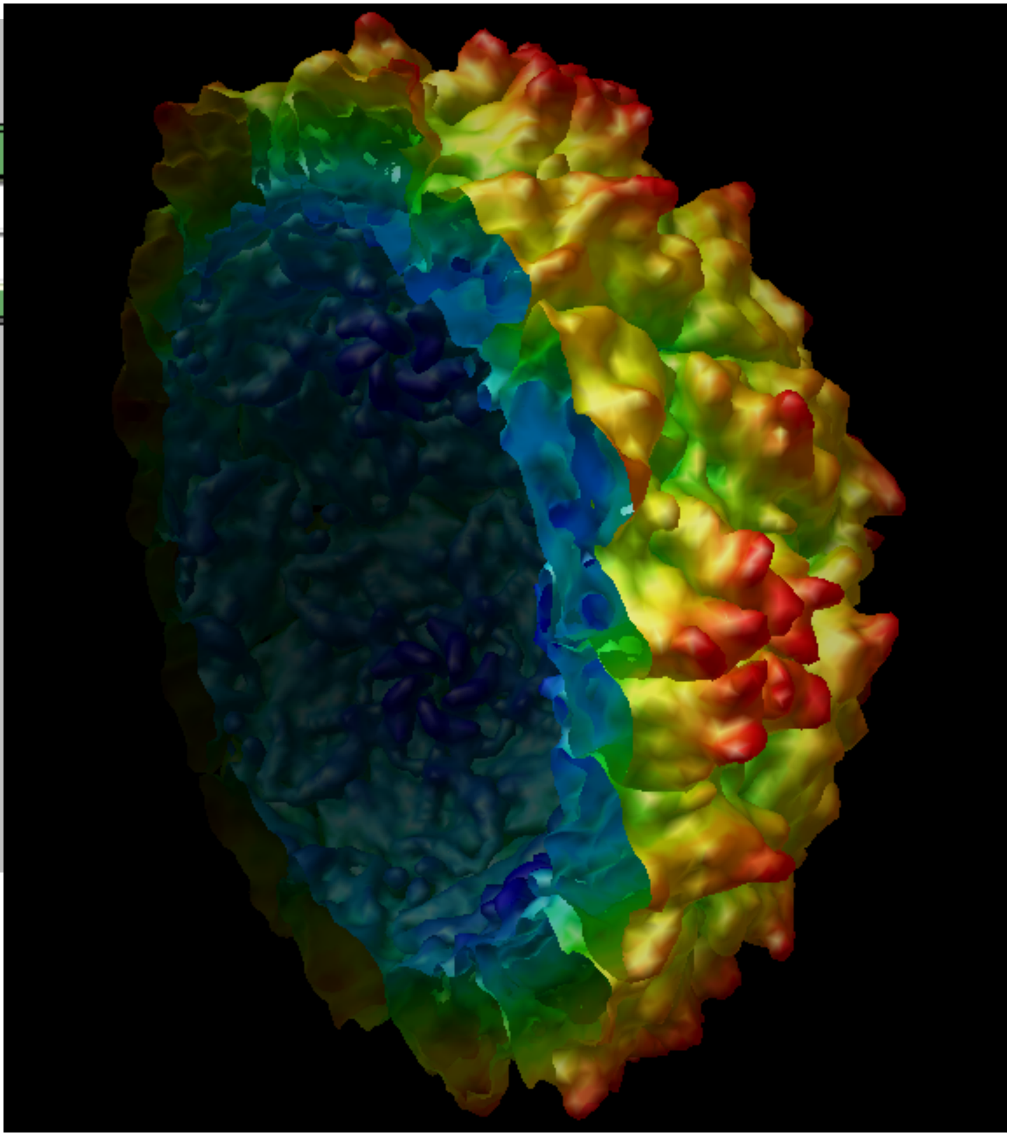


geom: root|2plv|CoarseMol

statement: (0,0,0)

import

signature: Set materials



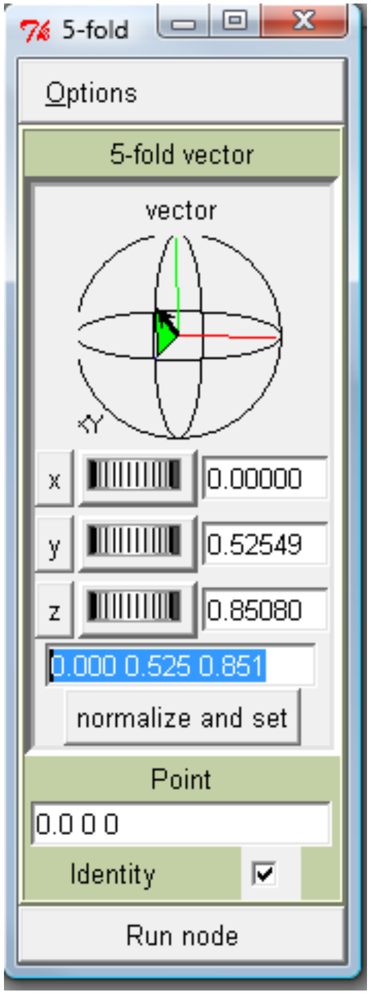
Exercise: Viral capsid

Task: expand the capsid by translating each 5-fold copy along its 5-fold axis

1 – expand the Icosahedral1 macro node
2 – display the 5-fold node's parameter panel
3 – find the 5-fold axis values (0.000 0.525 0.851) in the parameter panel
4 – locate and instantiate a translate node
5 – display the translate node's parameter panel
6 – find the translation vector 1 0 0 and replace with 0.000 0.525 0.851
7 – delete the connection between the 5-fold and the 3-fold nodes
8 – insert the translate node between the 5-fold and 3-fold nodes
9 – modify the translation length in the parameter panel of the translation node (right click on the thumbwheel to increase sensitivity to 10 for better results)
10 – witness the capsid expand

Exercise: Viral capsid

Task: expand the capsid by translating each 5-fold copy along its 5-fold axis



5-fold

Options

5-fold vector

vector

x 0.00000

y 0.52549

z 0.85080

0.000 0.525 0.851

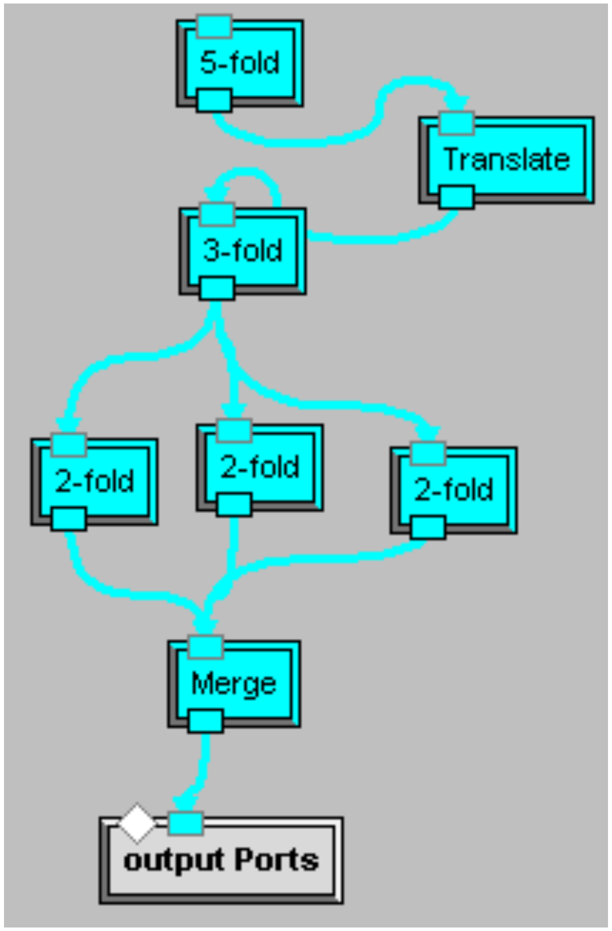
normalize and set

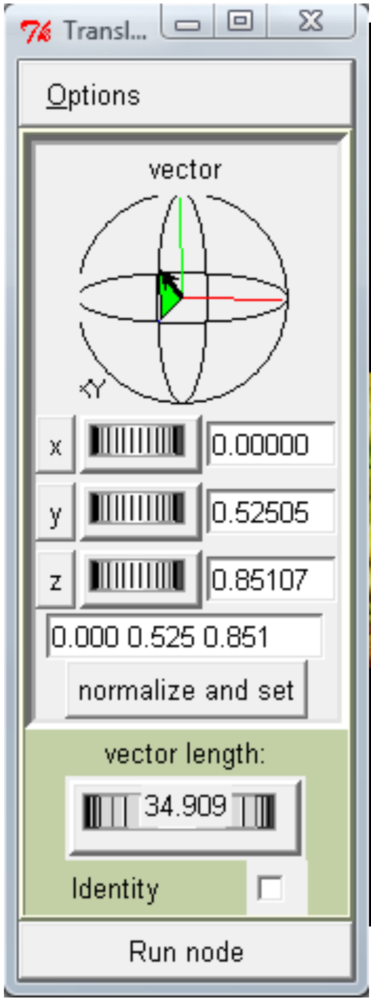
Point

0.0 0 0

Identity ☑

Run node





Transl...

Options

vector

x 0.00000

y 0.52505

z 0.85107

0.000 0.525 0.851

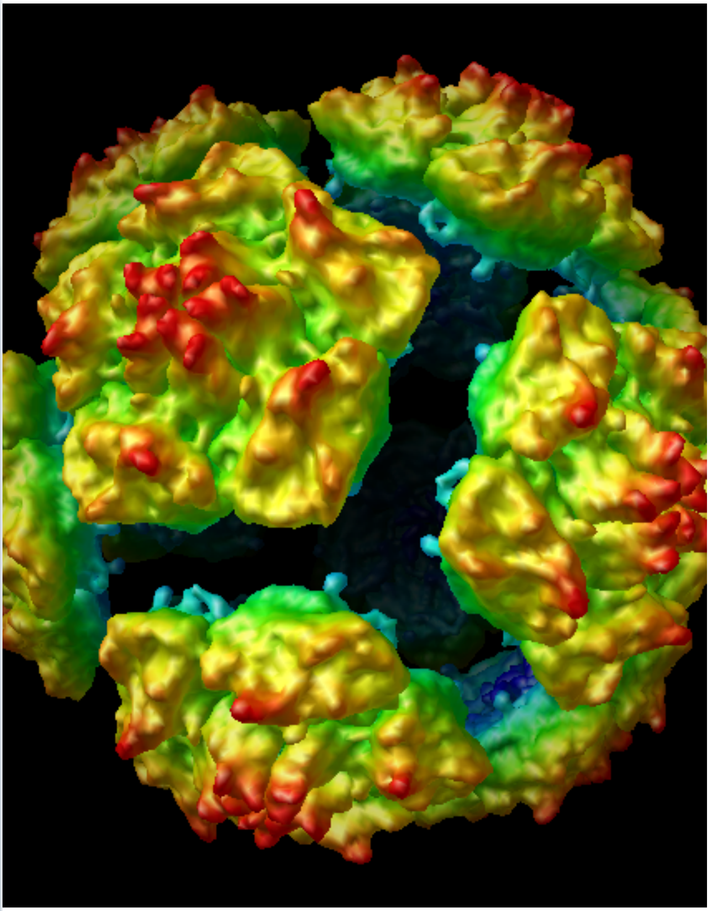
normalize and set

vector length:

34.909

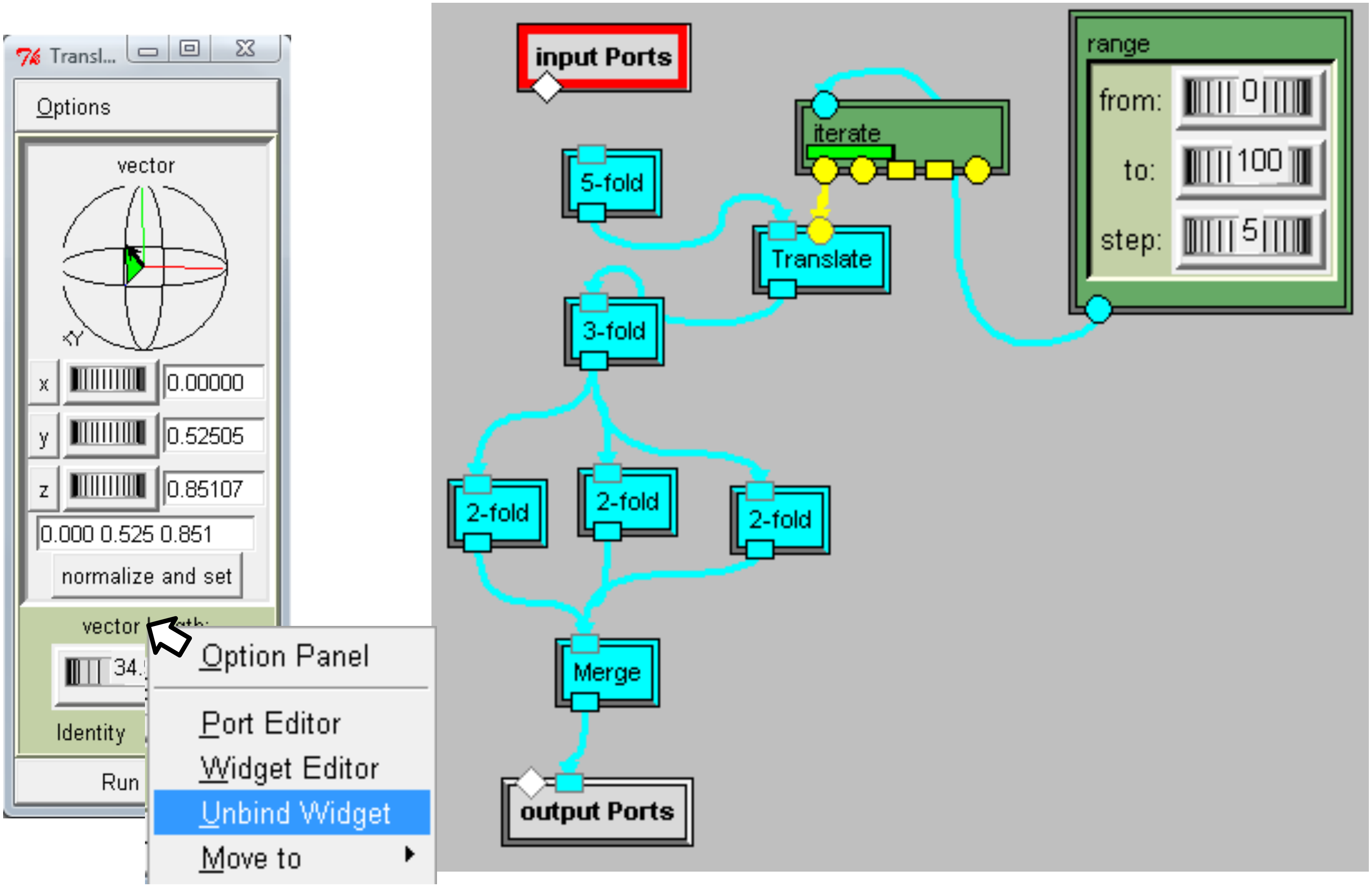
Identity □

Run node



Exercise: Viral capsid

Task: add range and iterate node to automate expansion



x 0.00000

y 0.52505

z 0.85107

0.000 0.525 0.851

from: 0

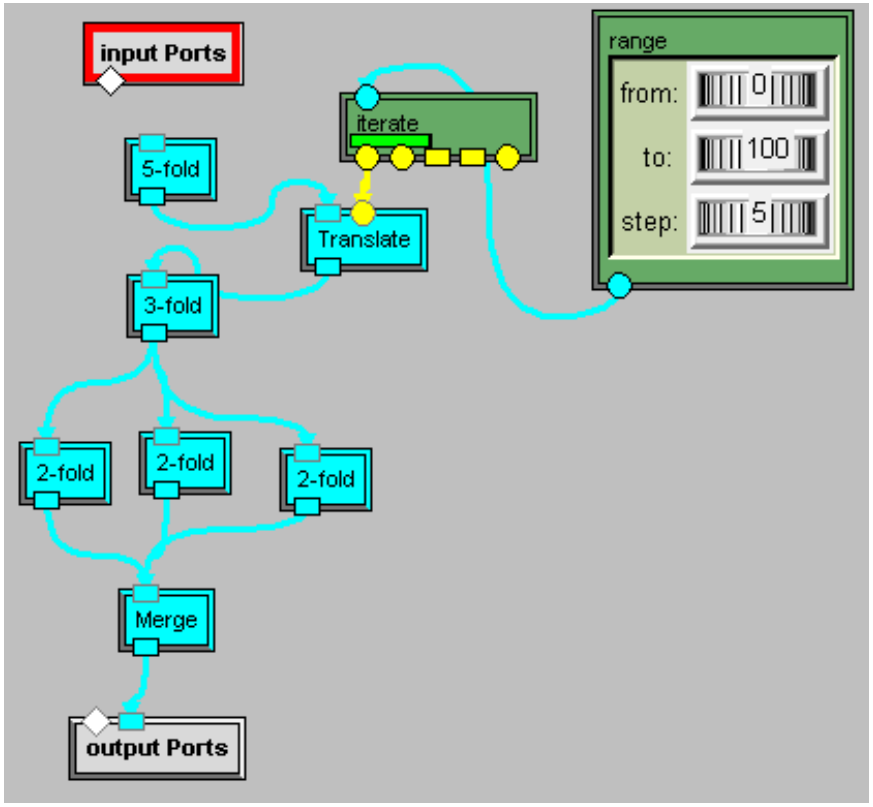
to: 100

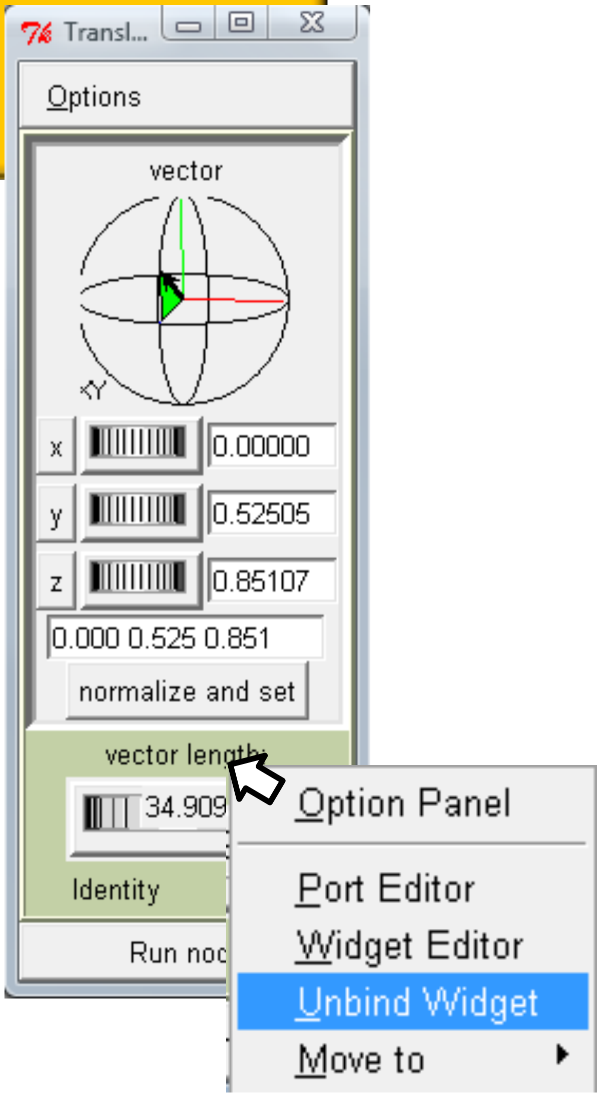
step: 5

Exercise: Viral capsid

Task: add range and iterate node to automate expansion

1 – unbind “vector length” widget in the parameter panel of the translate
2 – add range node from 0 to 100 in steps of 5
3 – add iterate node to iterate over range output
4 – feed value from iteration into “vector length” port





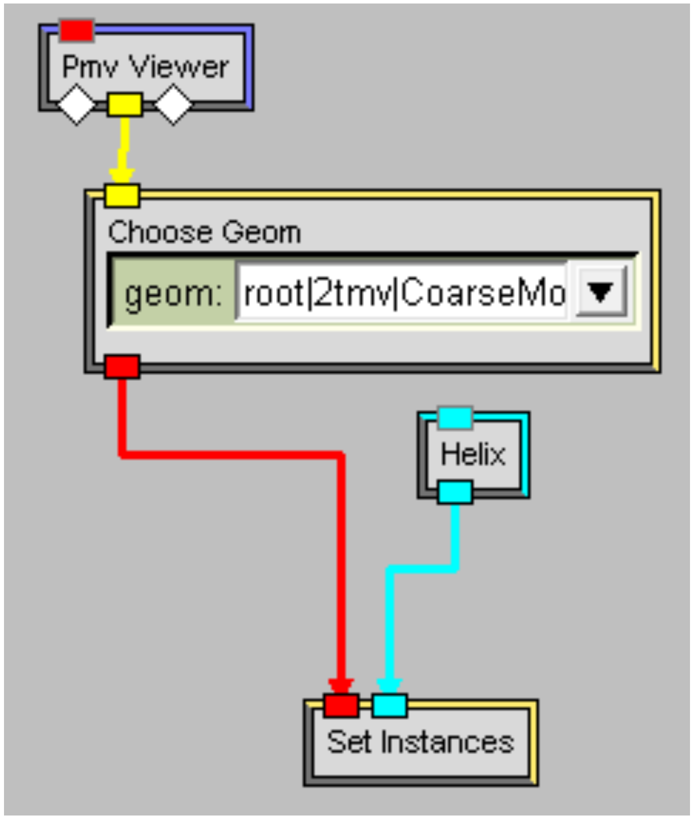
Exercise: Viral capsid

Task: build the TMV capsid

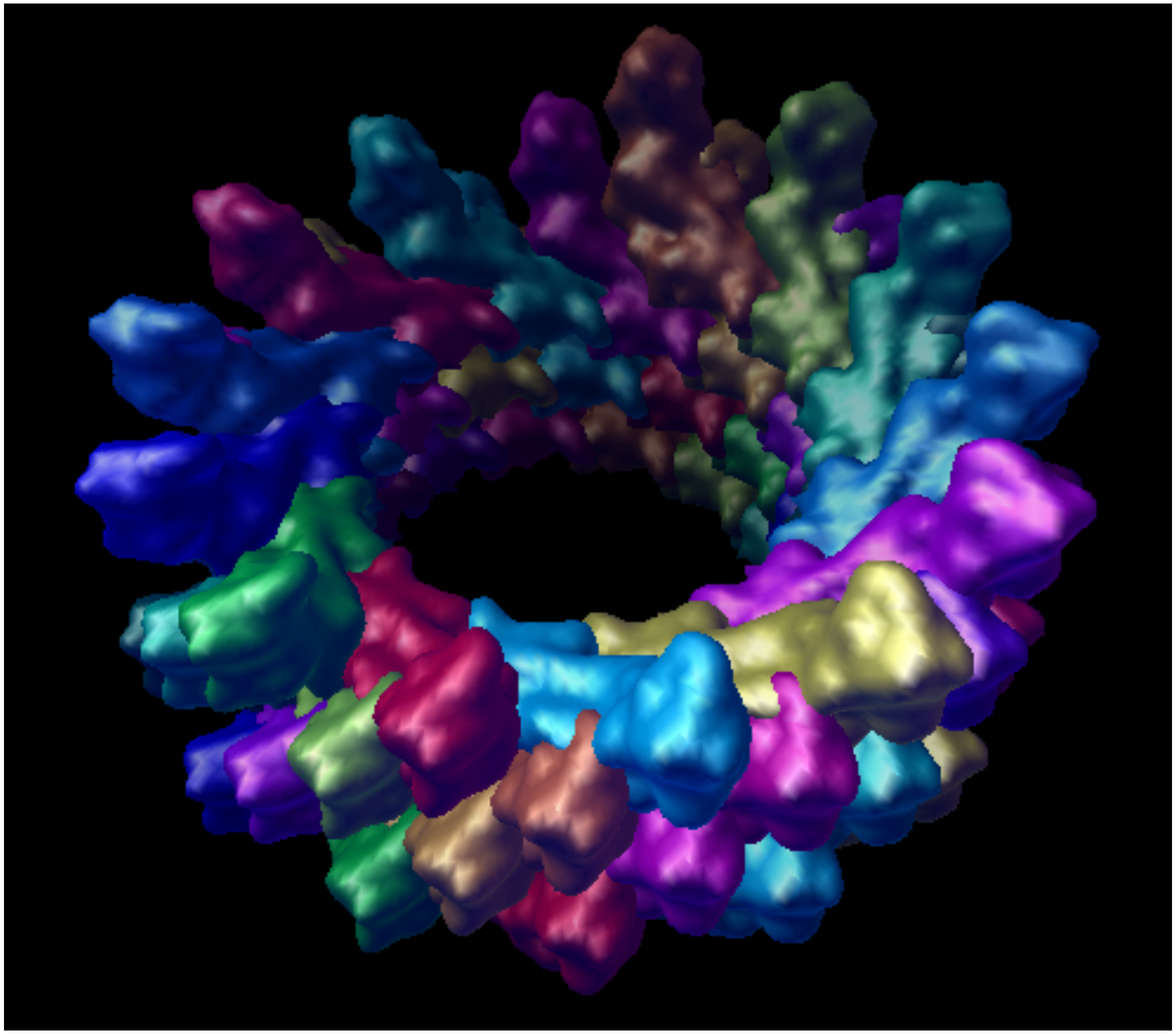
1 – delete the 2plv molecule
2 – load the Desktop/TutorialData/tmv/2tmv.pdb protein
3 – hide the lines and display a coarse molecular surface
4 – instantiate Pmv Viewer node
5 – instantiate a Choose Geom node
5 – connect the viewer to the choose Geom (the combo box will be populated)
6 – select root|2tmv|lines in the choose geom node
8 – Instantiate a Set Instances node (use the one from the 3D vis library)
9 – connect Choose Geom and Helix outputs to Set Instances
10 – double click on helix to display its parameter panel
11 – set copies to 50
12 – start changing the angle of the helix (values around 20 are good)
13 – interact with the molecule in the viewer (r n c and rotate)
14 – start changing the rise of the helix
15 – set the rise to 1.43 and the angle to 22.04
16 – color by Instances (using the dashboard)

Exercise: Viral capsid

Task: build the TMV capsid



geom: root|2tmv|CoarseMo



Exercise: use Pmv cmds in Vision

Task: build a network to run the computeMSMS cmd

Run the MSMS command in Pmv first to see possible arguments

1 – in Pmv delete 2tmv and load tf_1.pdb

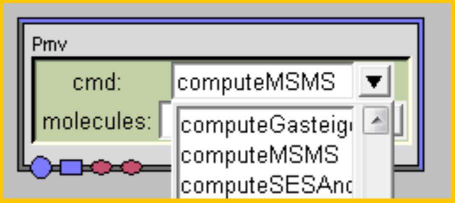
2 – in Pmv use the Compute -> molecular surface command

note the arguments that are possible include: surface name, probe radius, density, per molecule, etc …

3 – click Dismiss

4 – in Vision: create a new network 🗋

5 – instantiate a Pmv node and select computeMSMS for the cmd:



cmd: computeMSMS

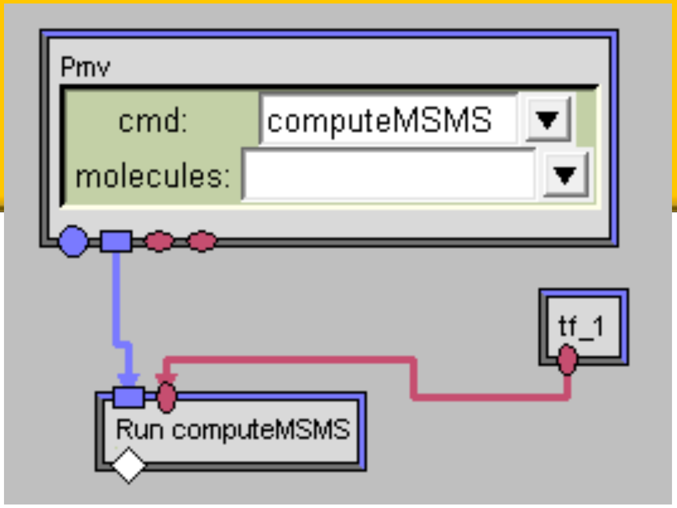
6 – instantiate a Run_command node and display its parameter panel

7 – connect the cmd output port of the Pmv node to the Run_command (note what happens)

8 – instantiate a tf_1 node and connect the molecule to the new input port on Run computeMSMS

9 – the surface gets computed after you connect

10 – vary the probe radius dial to higher values



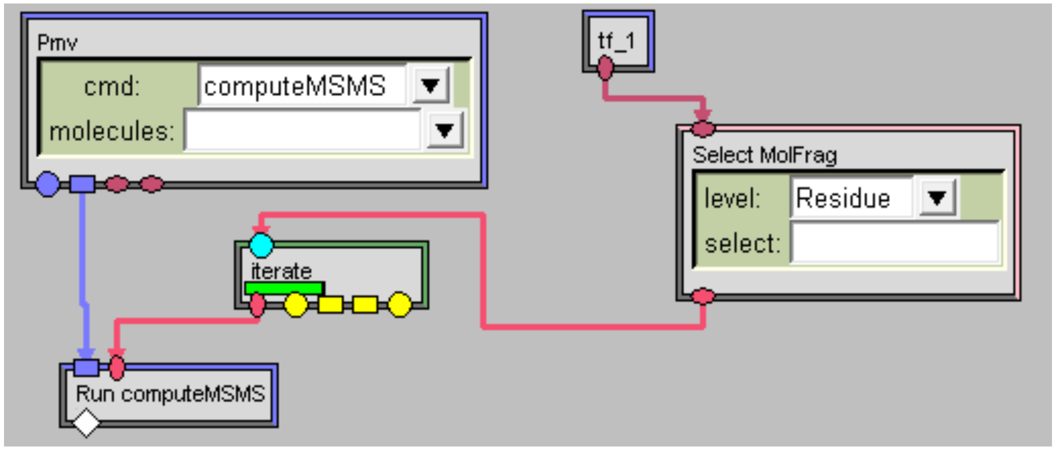
cmd: computeMSMS

molecules:

Exercise: use Pmv cmds in Vision

Task: modify the network to compute a surface for each amino acid

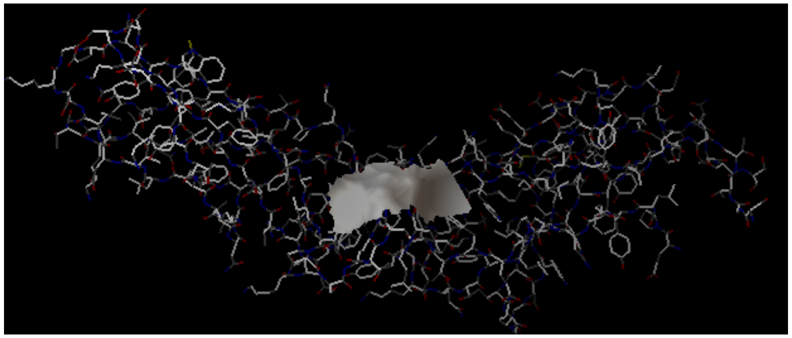
1 – instantiate a select MolFrag node to get a list of residues
2 – send the tf_1 molecule into this node and select Residue for level
3 – add an iterate node to iterate over the list of residues output by this node
4 – connect the oneItem output port of iterate to the compute MSMS node



Pmv
cmd: computeMSMS
molecules:

Select MolFrag
level: Residue
select:

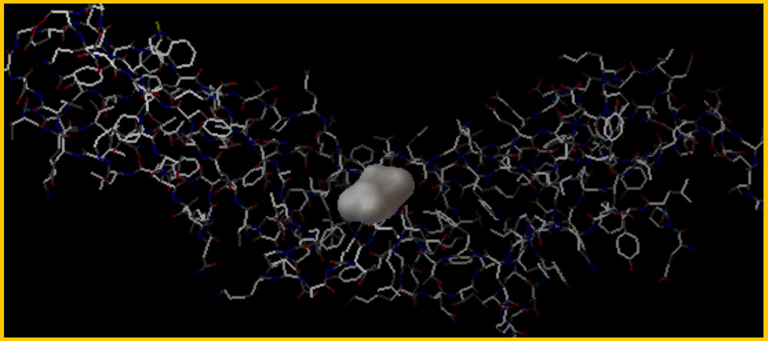
Surface is computed for the whole molecule but only the patch for the last residue is displayed



Exercise: use Pmv cmds in Vision

Task: modify the network to compute a surface for each amino acid

5 – un-check the “perMol” check button in the computeMSMS parameter panel



Surface is now computed for the for the set of atoms in the residue

6 – run the iterate node and watch the surface walk along the chain

Exercise: scale CPK radii by charge

Task: display CPK spheres scales y the atomic charge

1 – delete all molecules and load indinavir

2 – display CPK and color by atom types

3 – create a new vision network

4 – instantiate an indinavir node, Select MolFrag and Extract Atom Property

5 – send the molecule into select MolFrag with level set to Atom

6 – send the resulting AtomSet into Extract Atom Property and set prop. Name to ‘charge’

7 – instantiate a Dial node and set it to 3.0 and an op2 node

8 – in op2 select operator to be ‘mul’ and check apply to elements

9 – connect the list of charges to the first input port of op2 and the dial to the second

10 – instantiate a Pmv Viewer node and a Choose Geom

11 – connect the viewer output to the Choose Geom node and select root|indinavir|cpk

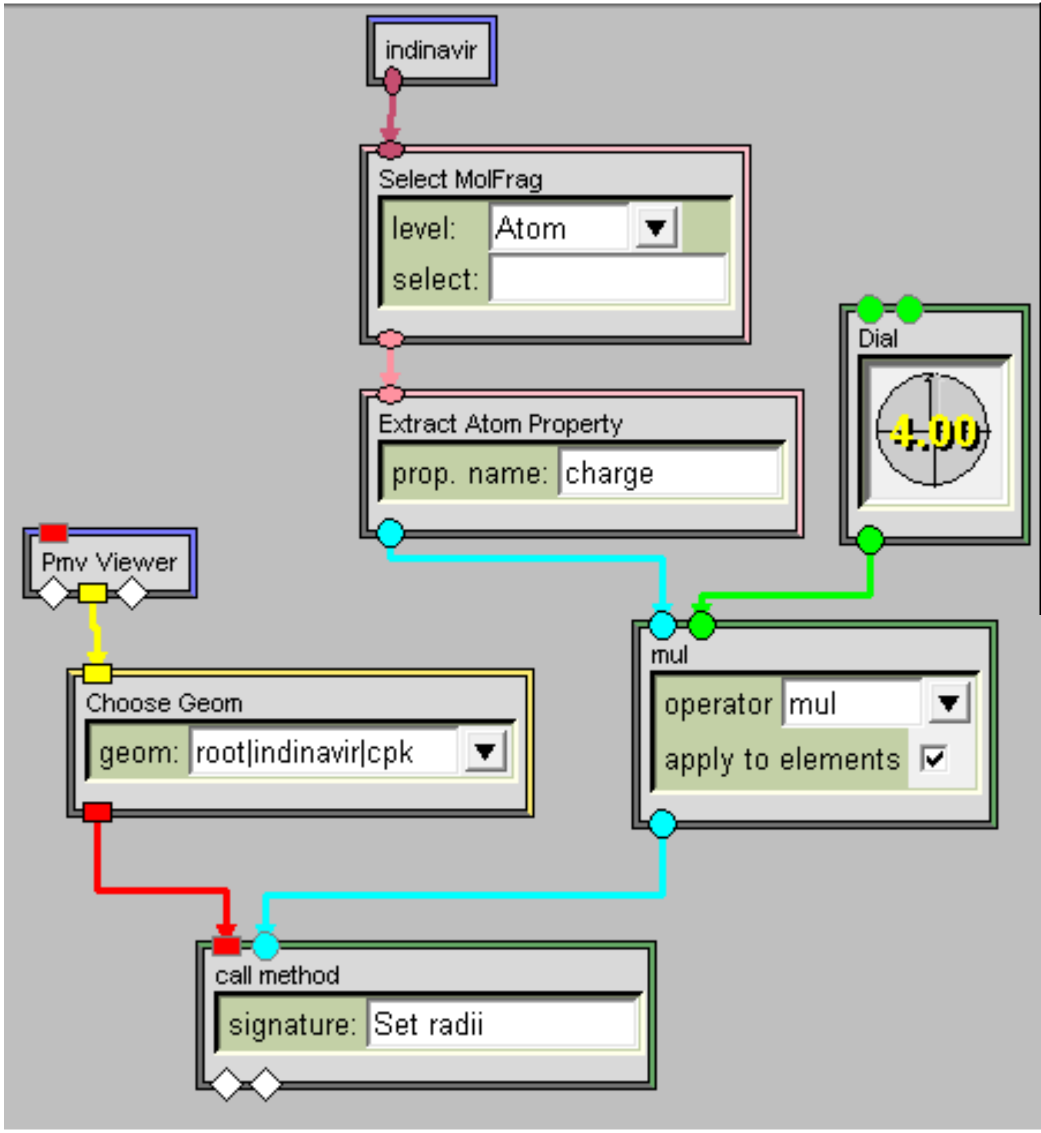
12 – instantiate a Call method node and set the signature to “Set radii” <enter>

13 – connet the output of Choose Geom to the first port of call method

14 – connect the list of scaled charges coming out of mul to the second port

Exercise: scale CPK radii by charge

Task: display CPK spheres scales by the atomic charge



indinavir

Select MolFrag

level: Atom

select:

Extract Atom Property

prop. name: charge

Dial

4.00

Pmv Viewer

Choose Geom

geom: root|indinavir|cpk

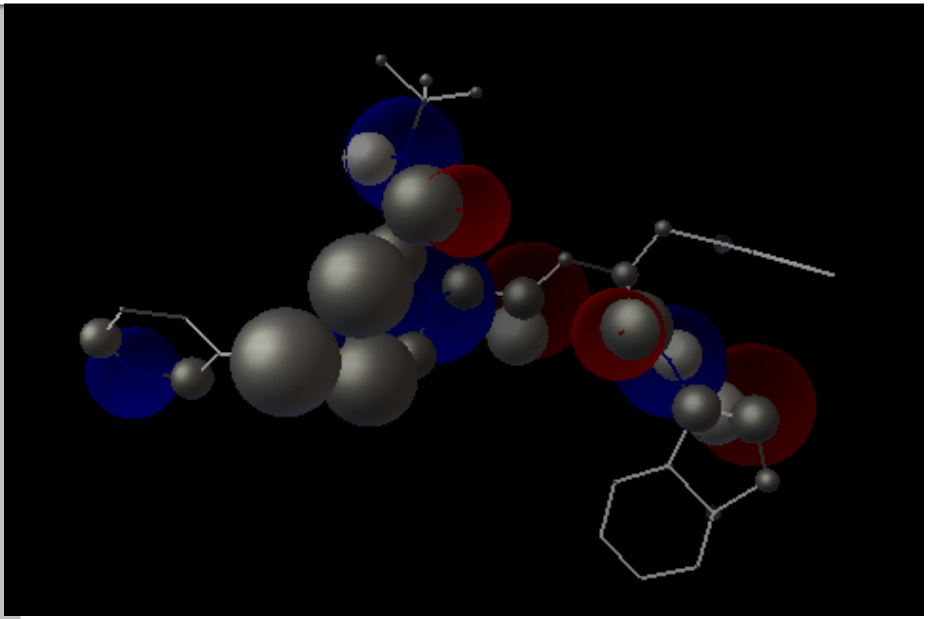
mul

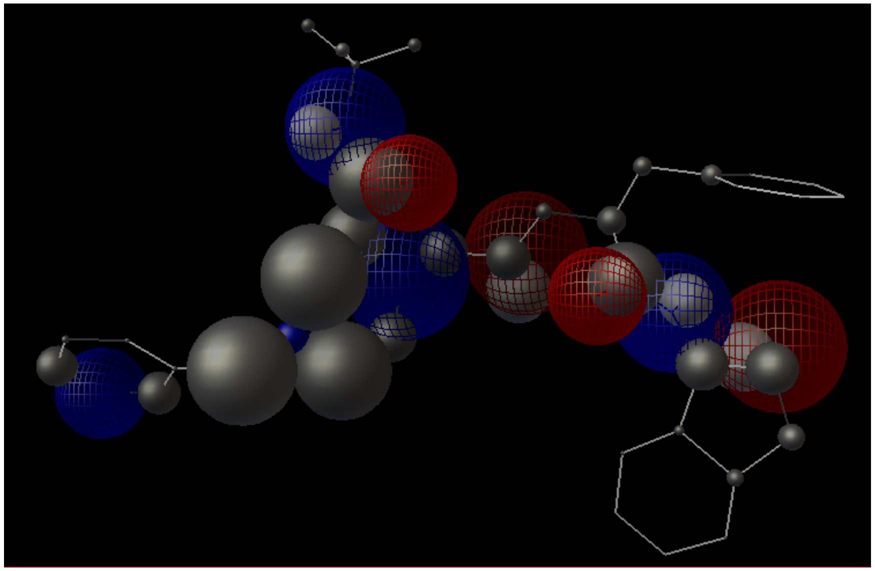
operator mul

apply to elements ☑

call method

signature: Set radii

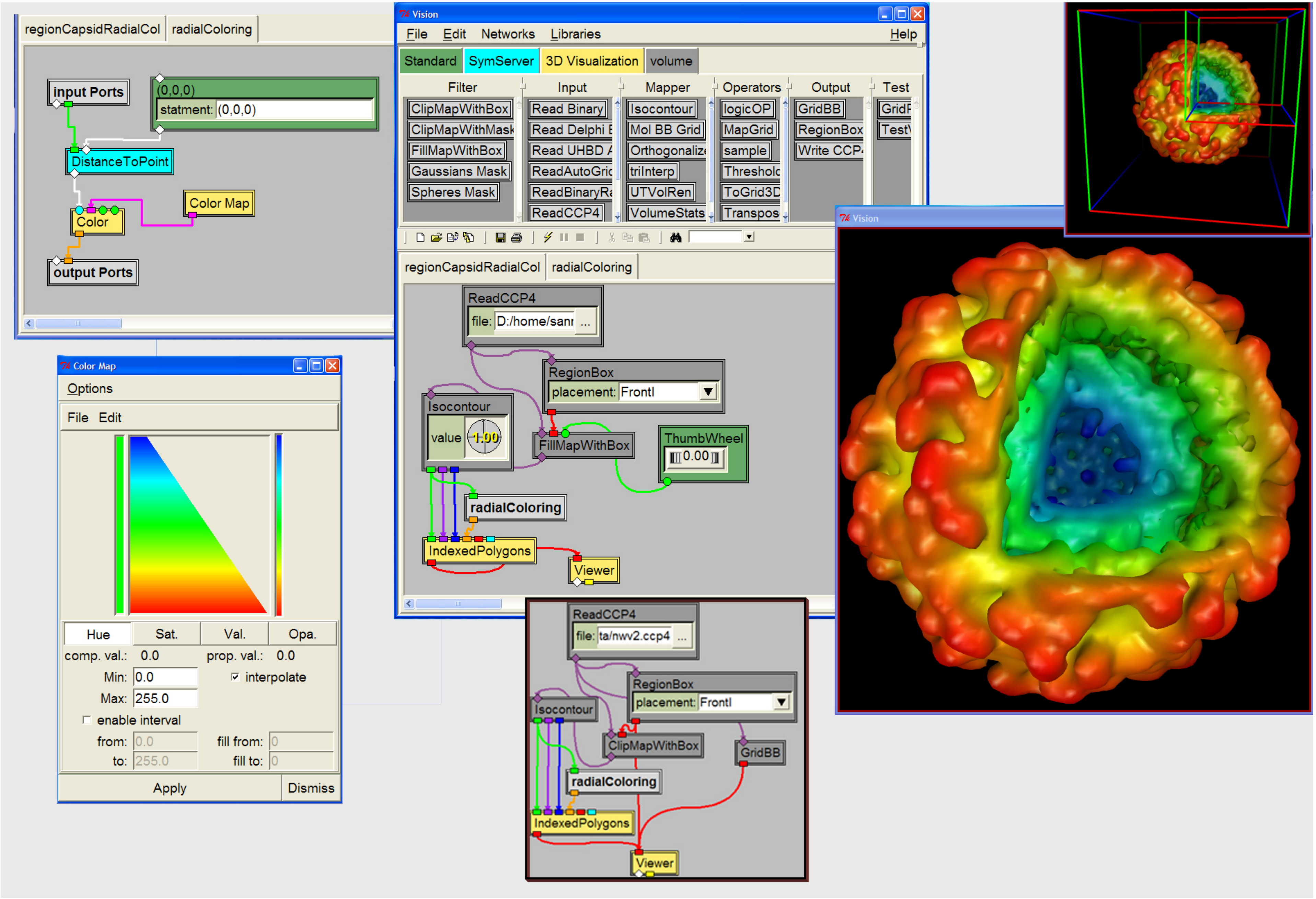




Part 4: Overview

- The Volume library
- The vizlib library
- The matplotlib library
- Student problems
- Wrap up

Working with volumetric data


statment: (0,0,0)

file: D:/home/sanr

placement: FrontI

value 1.00

ThumbWheel 0.00

file: ta/nwv2.ccp4

placement: FrontI

comp. val.: 0.0

prop. val.: 0.0

Min: 0.0

Max: 255.0

☑ interpolate

☐ enable interval

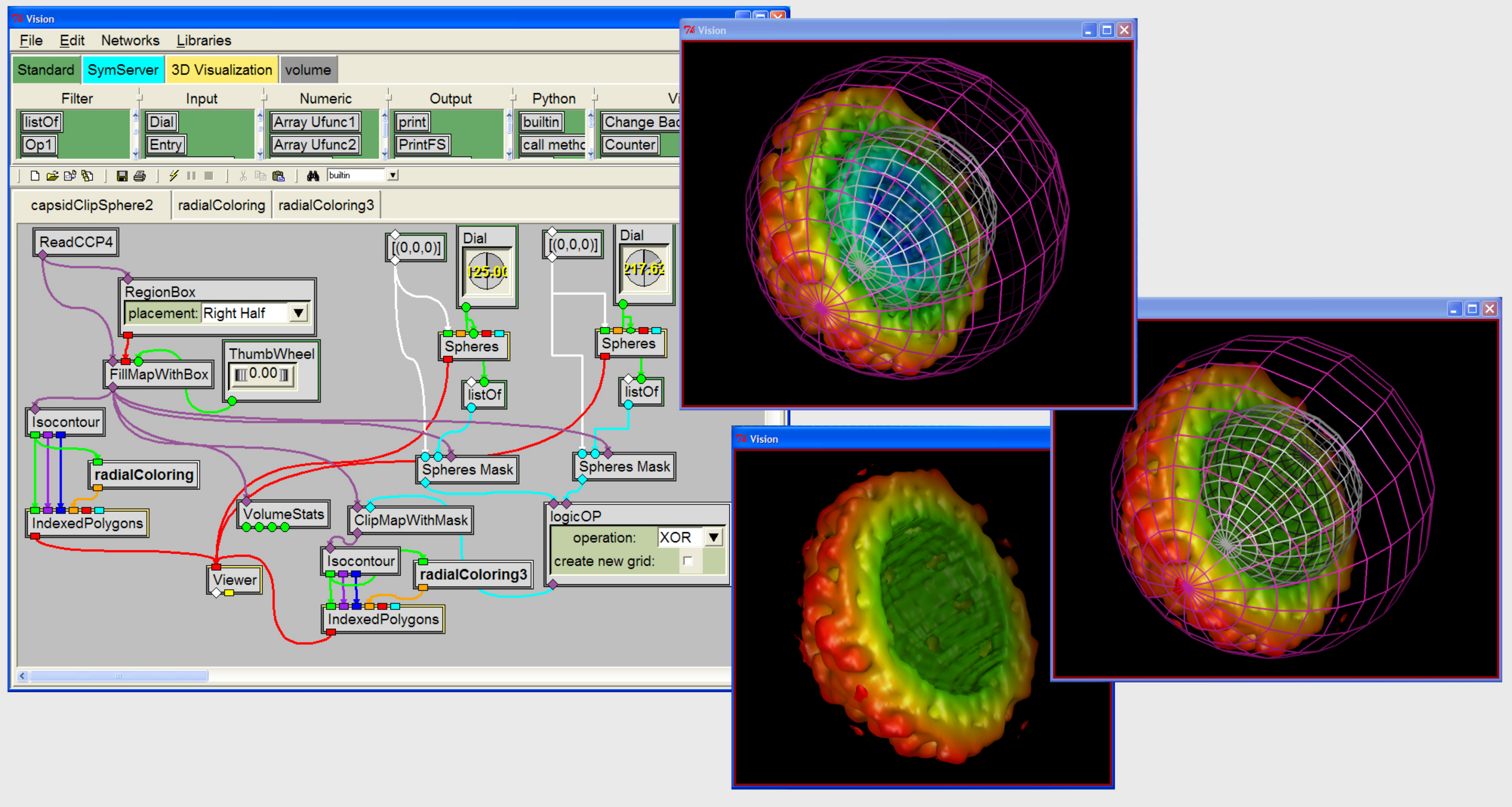
from: 0.0

to: 255.0

fill from: 0

fill to: 0

Working with volumetric data



placement: Right Half

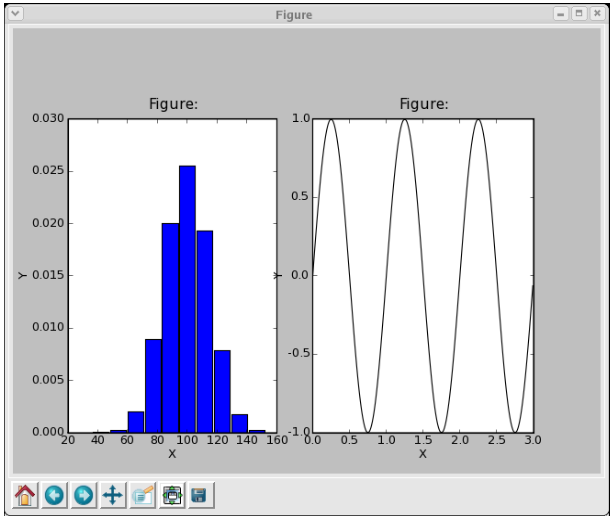
operation: XOR

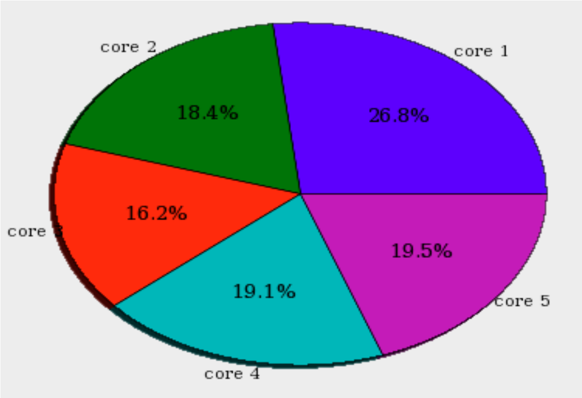
create new grid: □

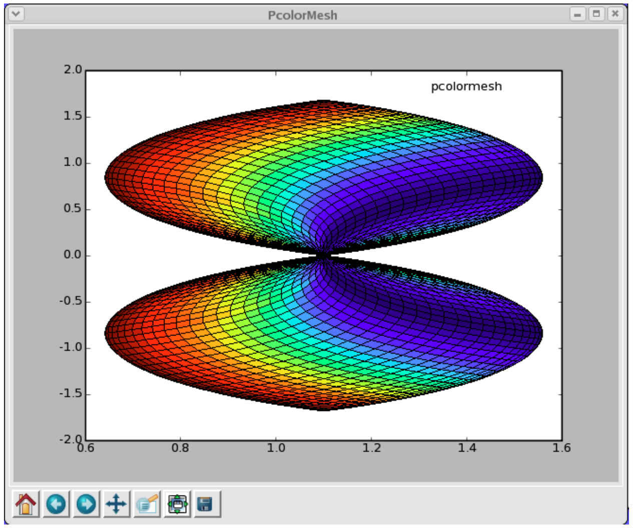
Boolean operation can be performed on masks to create complex masks
Here a spherical slab mask is created by XOR' ing 2 spheres

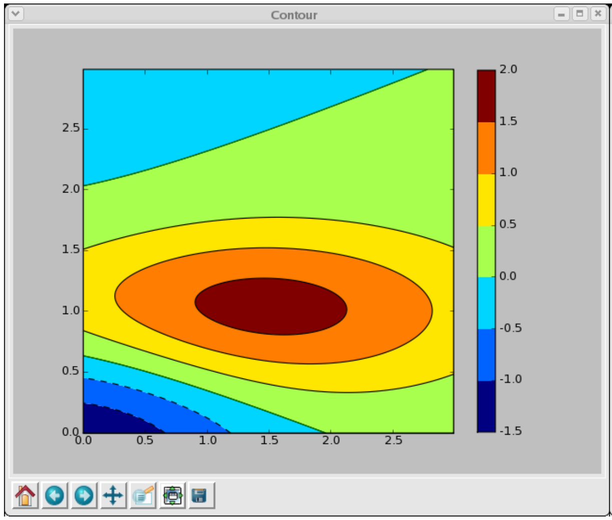
2D plotting

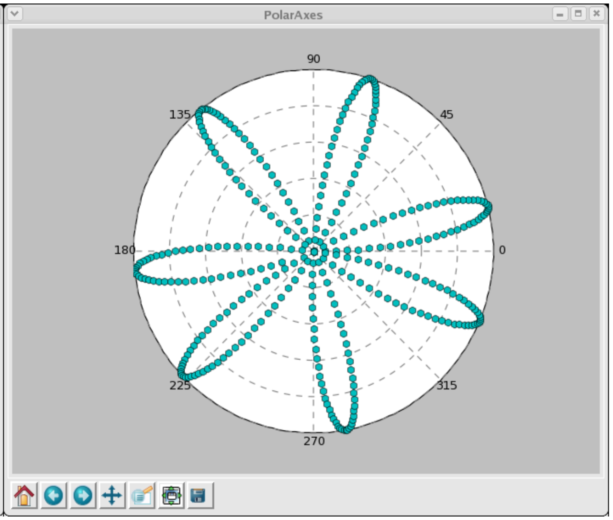
http://mgltools.scripps.edu/packages/vision/matplotlib/vision-networks-for-matplotlib

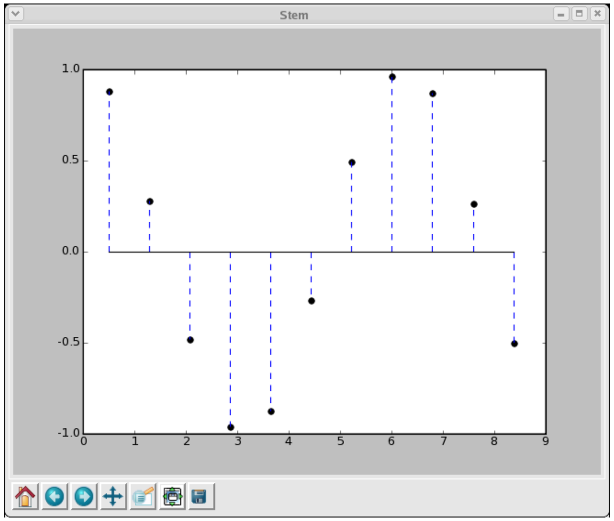




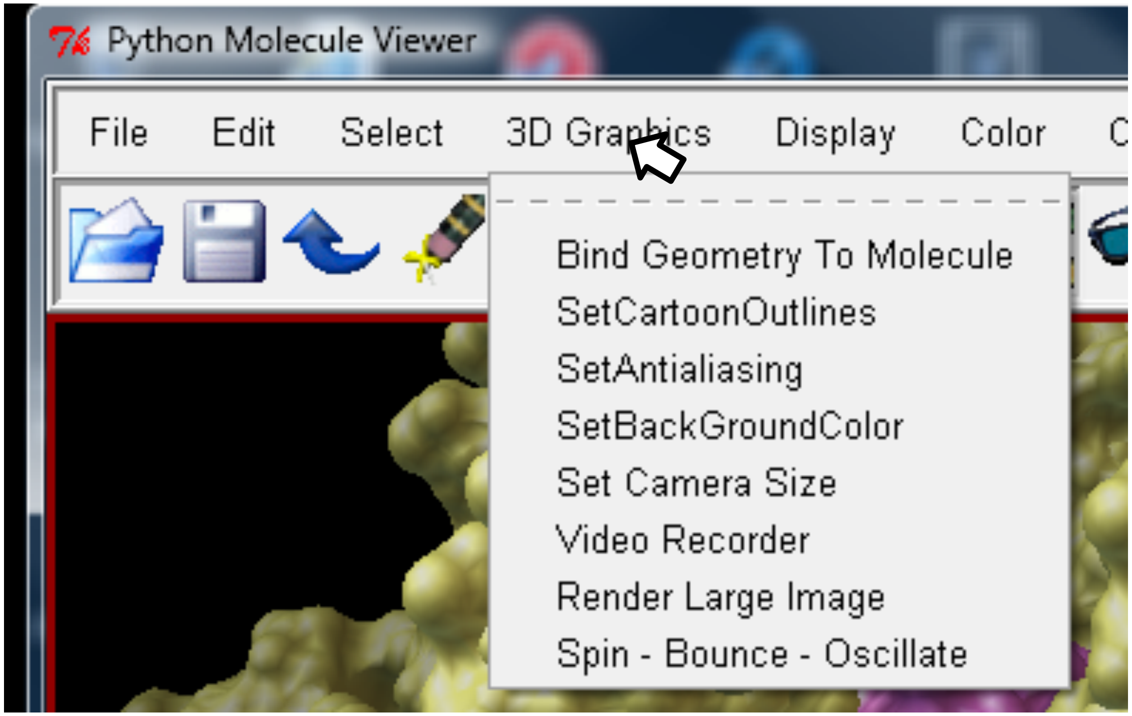




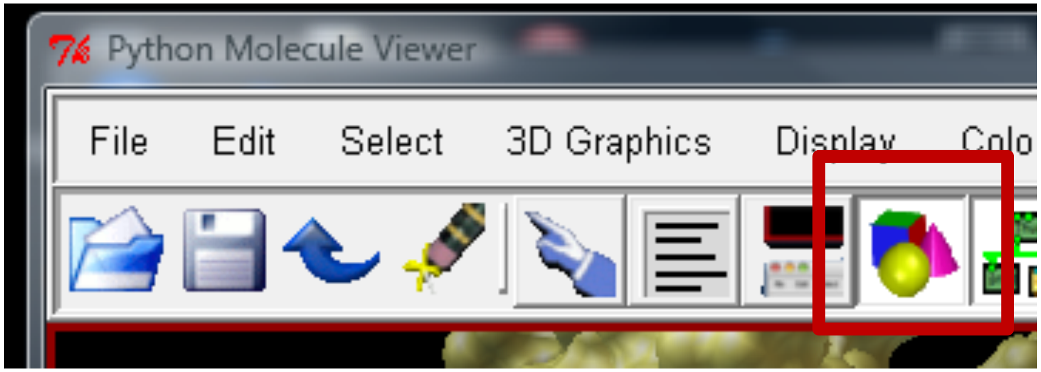


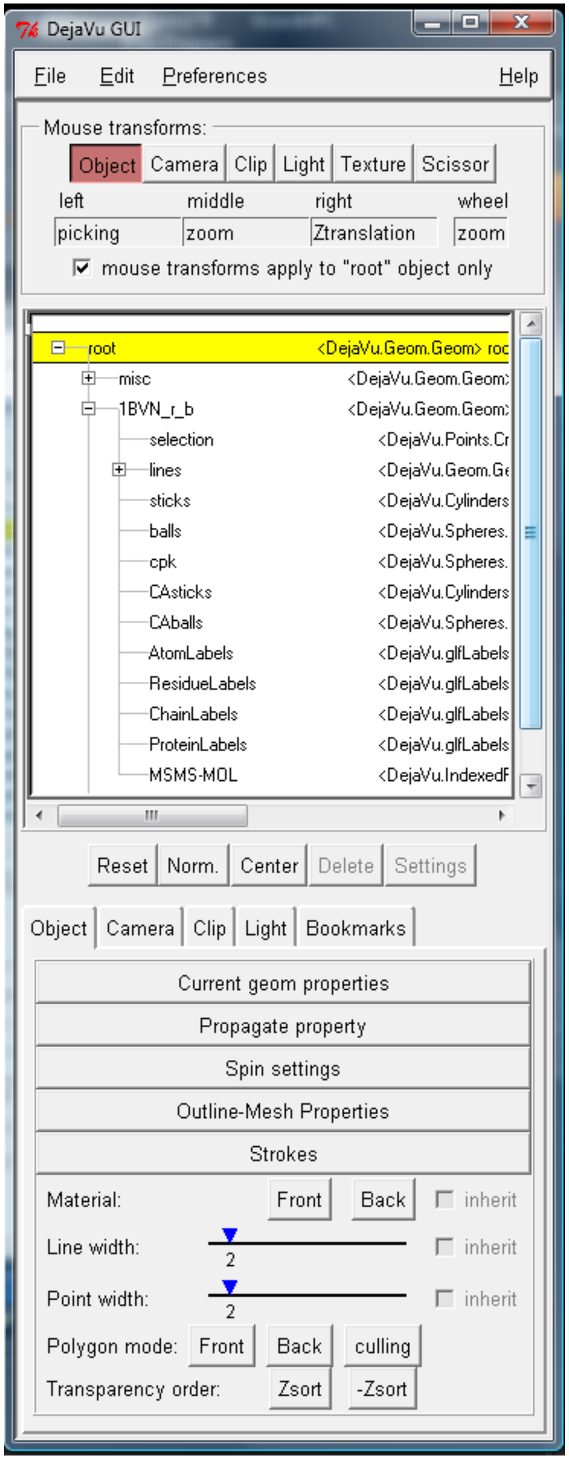


3D Visualization



3D Visualization





left: picking
middle: zoom
right: Ztranslation
wheel: zoom

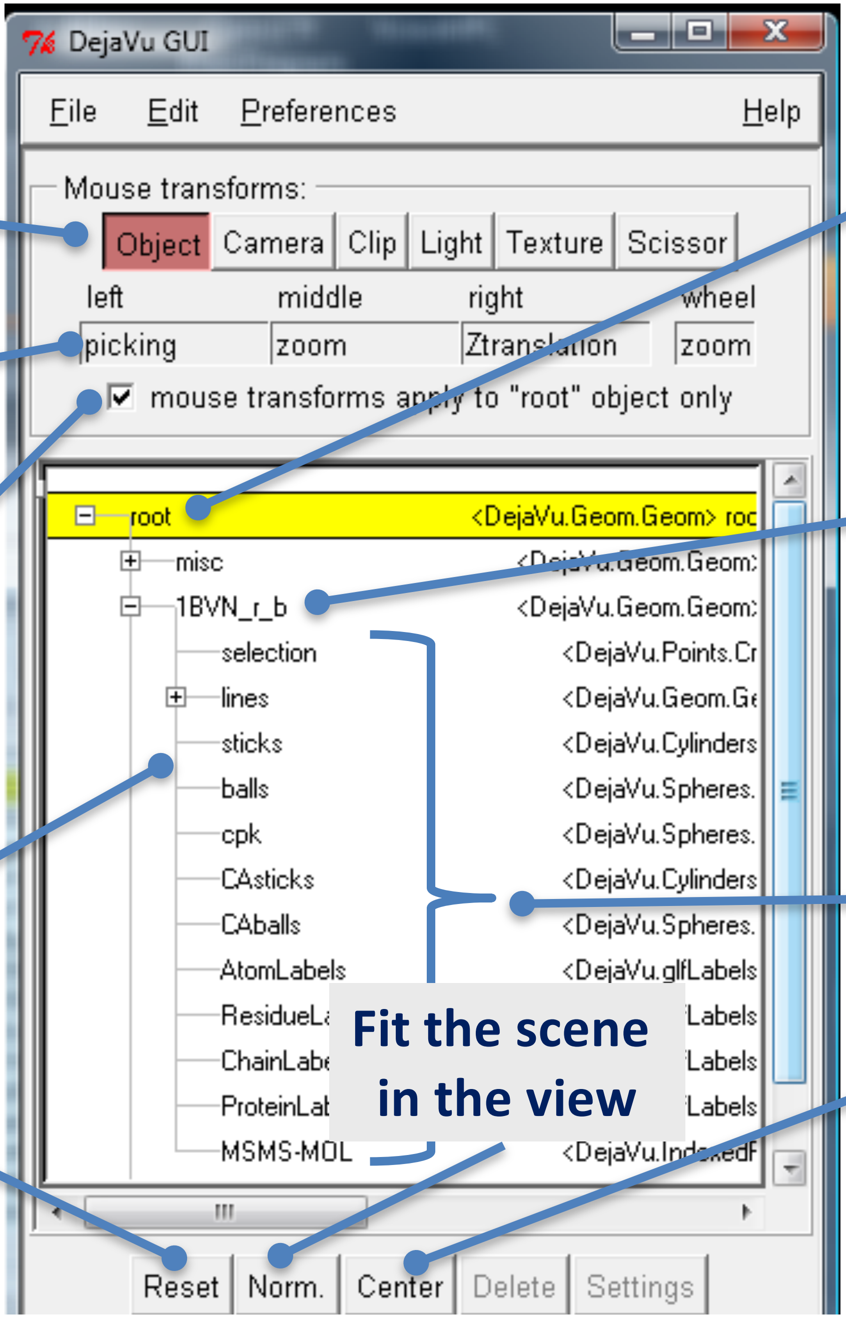
☑ mouse transforms apply to "root" object only

Line width: 2 □ inherit

Point width: 2 □ inherit

Material: □ inherit

3D Visualization: GUI overview



Mouse transforms:

left	middle	right	wheel
picking	zoom	Ztranslation	zoom

☑ mouse transforms apply to "root" object only

```
⊟ root                <DejaVu.Geom.Geom> roc
   ⊞ misc              <DejaVu.Geom.Geom>
   ⊟ 1BVN_r_b          <DejaVu.Geom.Geom>
        selection      <DejaVu.Points.Cr
      ⊞ lines          <DejaVu.Geom.Ge
        sticks         <DejaVu.Cylinders
        balls          <DejaVu.Spheres.
        cpk            <DejaVu.Spheres.
        CAsticks       <DejaVu.Cylinders
        CAballs        <DejaVu.Spheres.
        AtomLabels     <DejaVu.glfLabels
        ResidueLa      Labels
        ChainLabe      Labels
        ProteinLab     Labels
        MSMS-MOL       <DejaVu.IndexedF
```

Bind Mouse to transform

Operations assigned to mouse buttons (changes with modifiers)

When checked 3D Xforms apply to root

Geometry objects hierarchy

Reset Xform of current object

Fit the scene in the view

Root geom parent of all geometries and current object

Master geom for all geoms of a given molecule

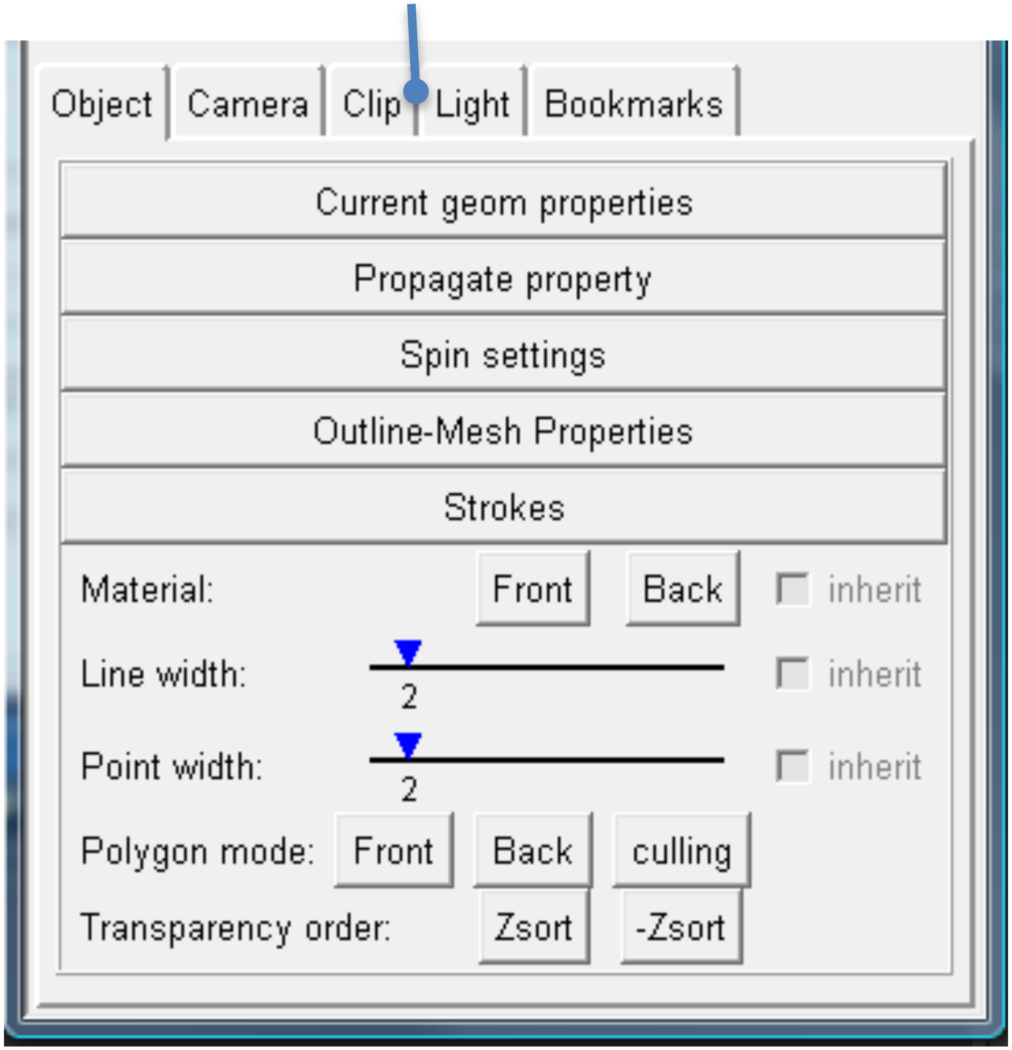
Geoms created by Pmv cmds for that molecule

Set rotation center to center of the scene

3D Visualization: GUI overview

Select property panel to show

Object property panel

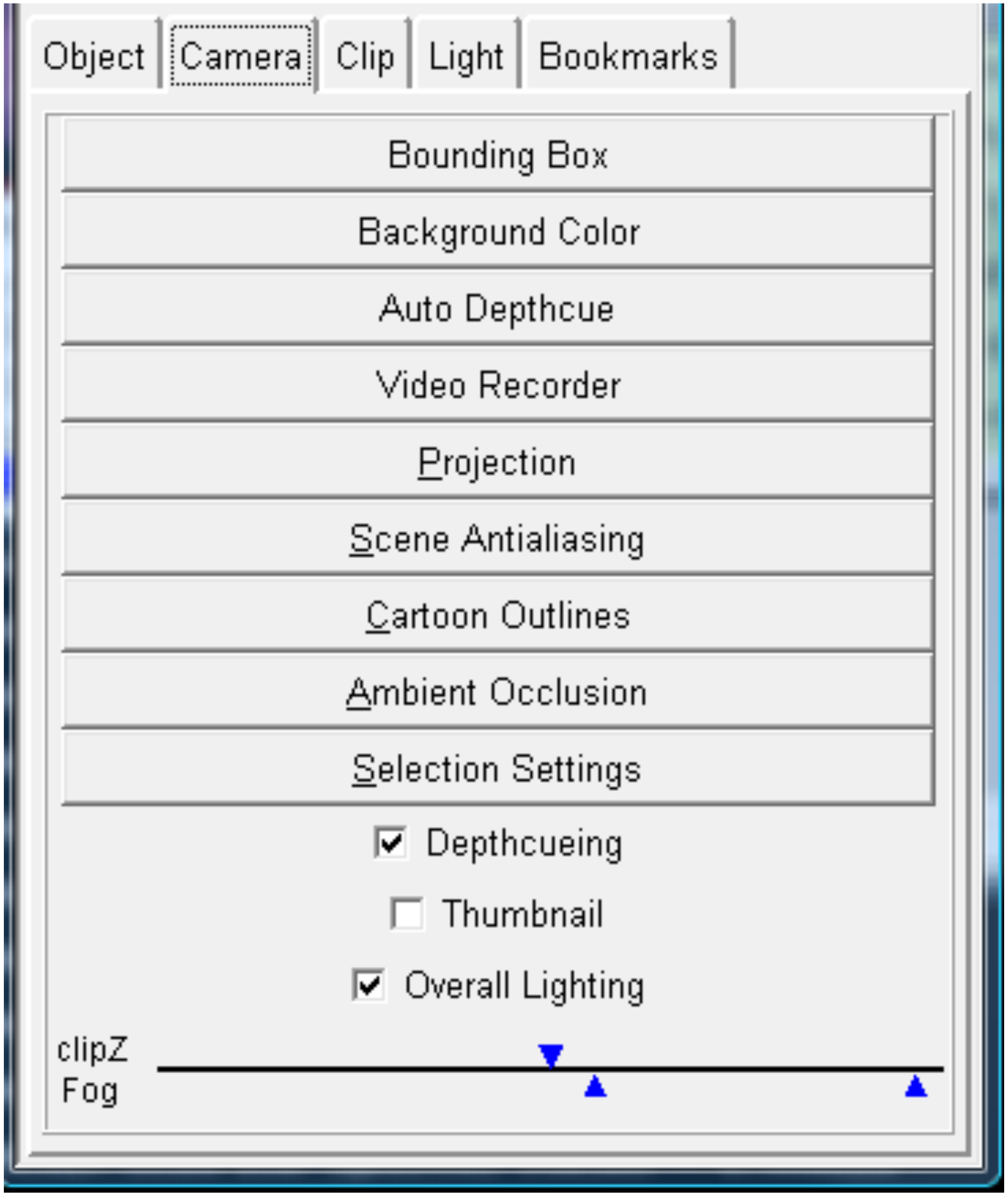


Line width: 2 □ inherit

Point width: 2 □ inherit

3D Visualization: GUI overview

Camera property panel

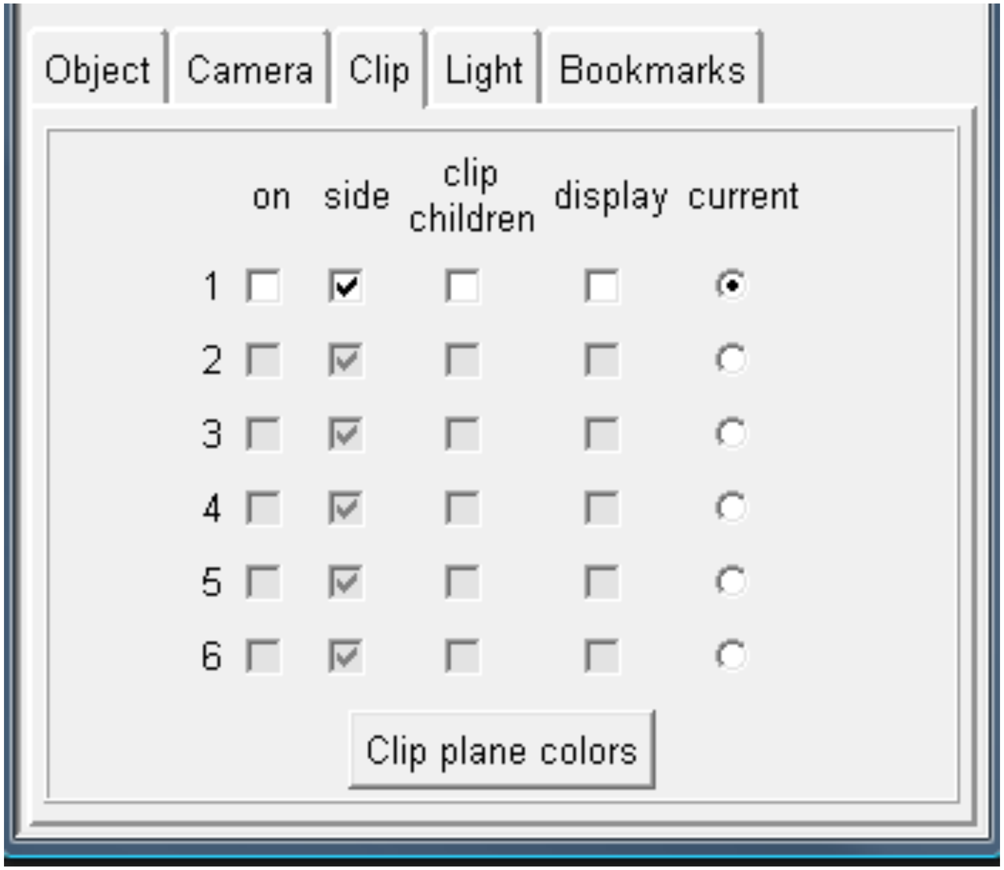


☑ Depthcueing

□ Thumbnail

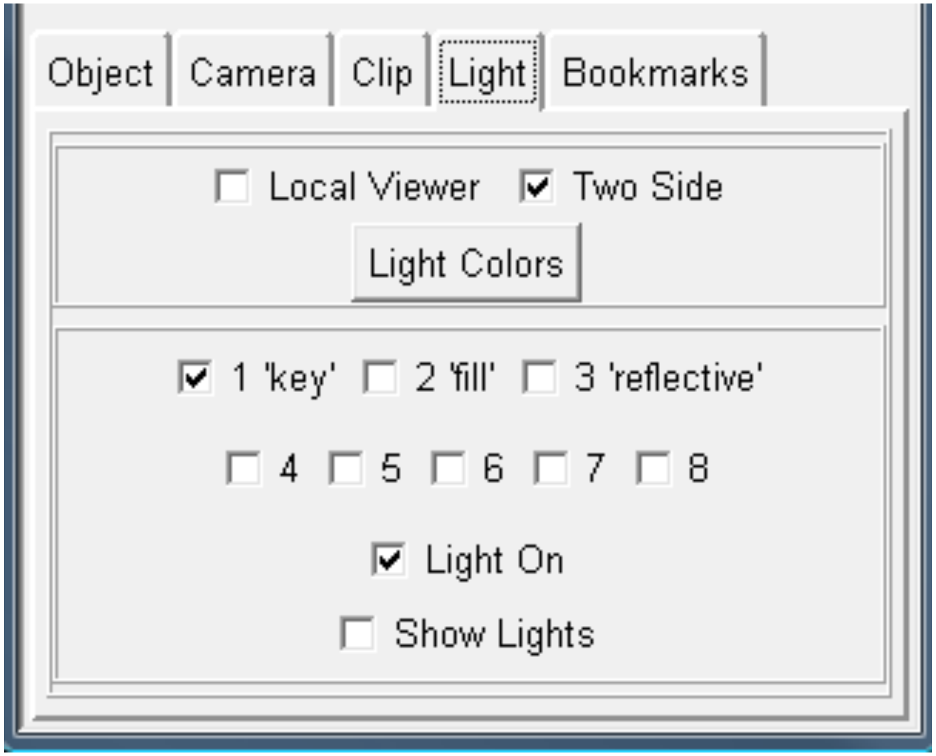
☑ Overall Lighting

3D Visualization: GUI overview



	on	side	clip children	display	current
1	□	☑	□	□	●
2	□	☑	□	□	○
3	□	☑	□	□	○
4	□	☑	□	□	○
5	□	☑	□	□	○
6	□	☑	□	□	○

Clipping planes property panel



□ Local Viewer ☑ Two Side

☑ 1 'key' □ 2 'fill' □ 3 'reflective'

□ 4 □ 5 □ 6 □ 7 □ 8

☑ Light On

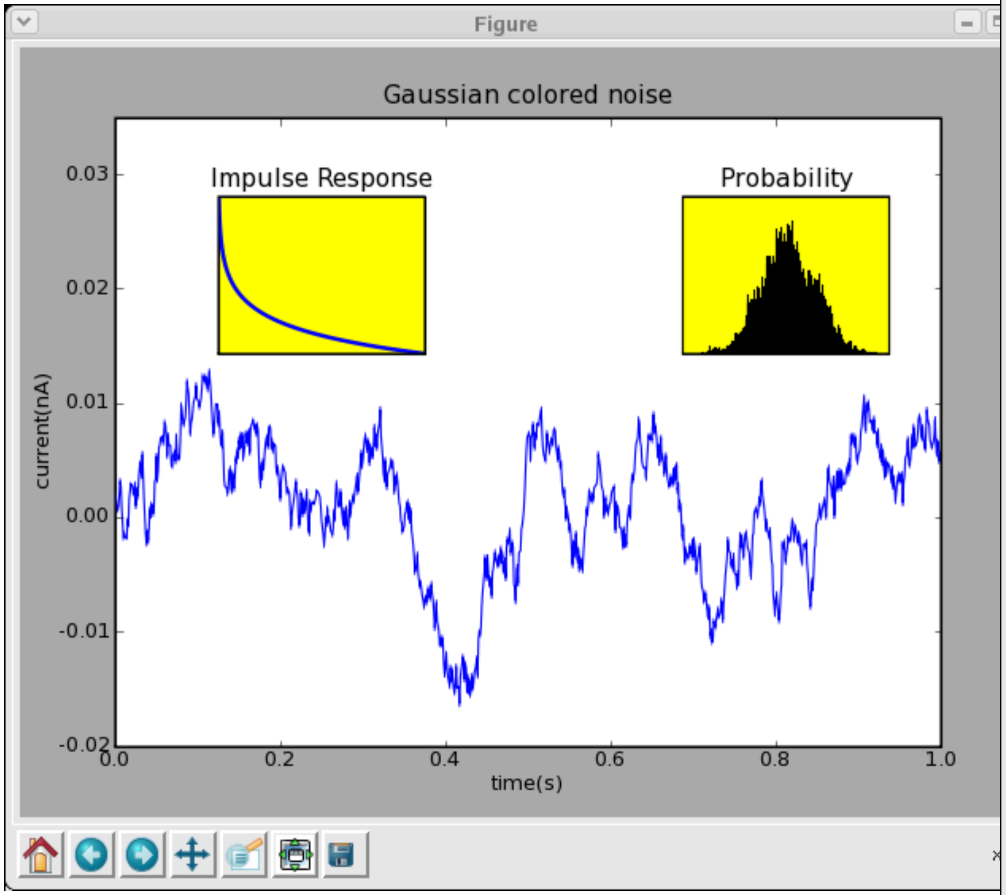
□ Show Lights

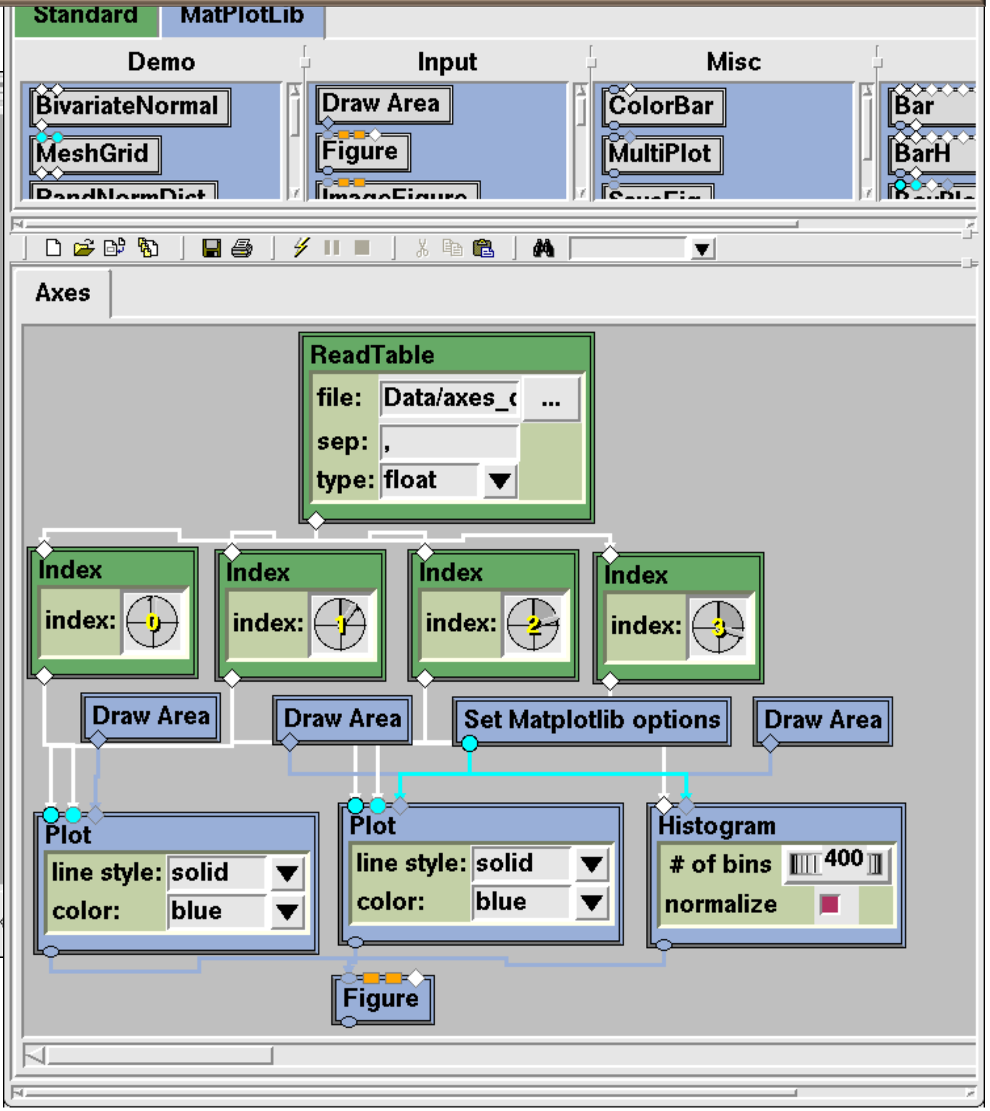
Lights property panel

2D plotting

http://mgltools.scripps.edu/packages/vision/matplotlib/vision-networks-for-matplotlib

Desktop/doc/Examples/matplotlib





ReadTable
file: Data/axes_c
sep: ,
type: float

Index index: 0

Index index: 1

Index index: 2

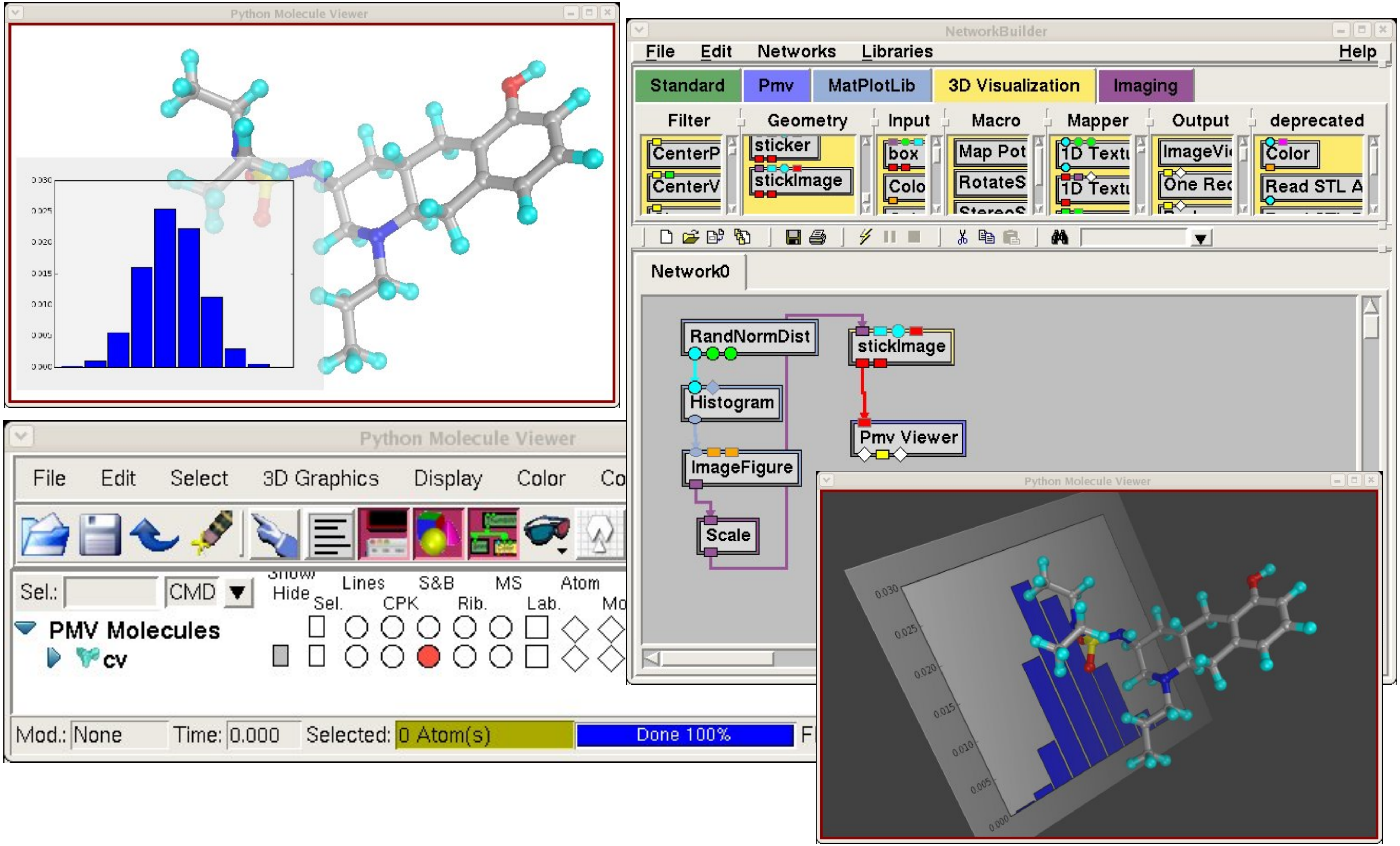
Index index: 3

Plot
line style: solid
color: blue

Plot
line style: solid
color: blue

Histogram
# of bins 400
normalize

Matplotlib in DejaVu



Mod.: None Time: 0.000 Selected: 0 Atom(s) Done 100%