Pathomx Documentation

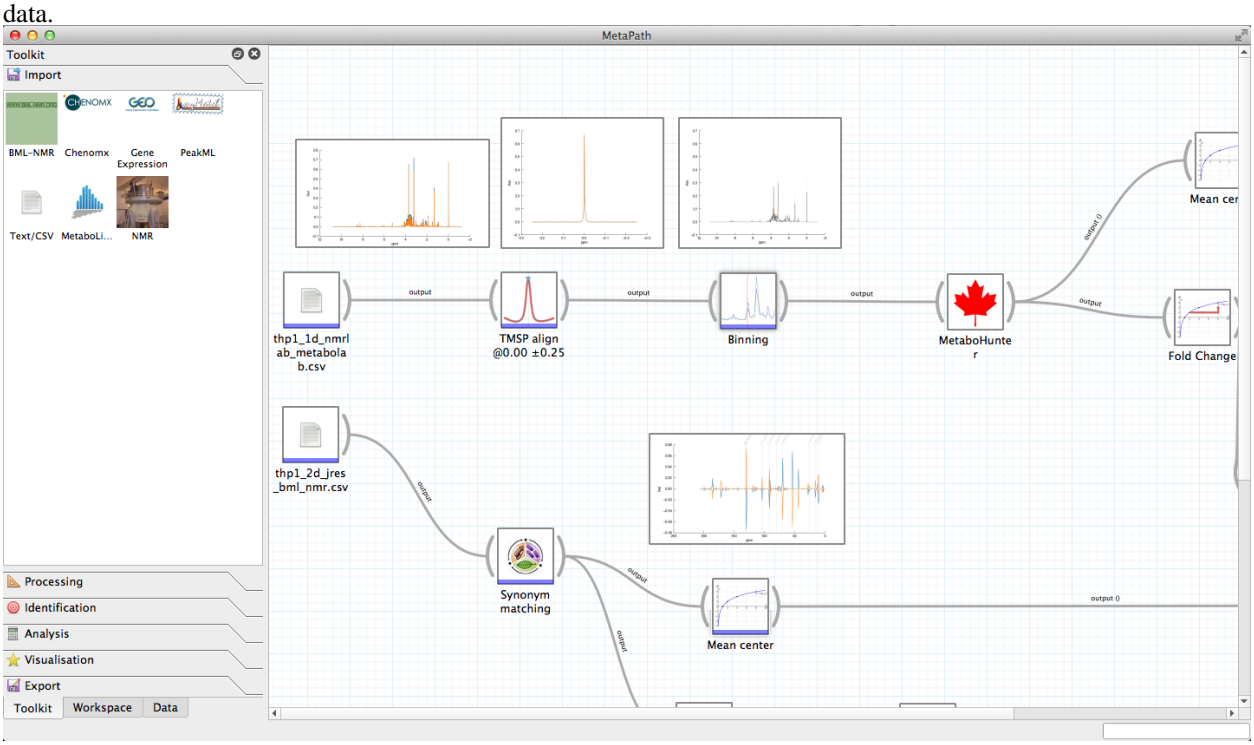
Release 2.2.0

Martin A. Fitzpatrick, Catherine M. McGrath, Stephen P. Young

August 06, 2015

Contents

Pathomx is a workflow-based tool for the analysis of metabolic pathways and visualisation of associated experimental data.



Workflow based data exploration: Workflows can be created, edited and shared using the built-in analysis workflow editor. Set up standardised analysis approaches and apply them to new data consistently. Add-remove tools to test different approaches, and re-apply tried and tested methods to new data.

Metabolic pathway exploration: Browse through the metabolic pathway database, with automated clean rendering of pathways. Add and remove metabolic pathways, show intra-pathway linkages, and map metabolic routes through the system. Browse the in-built database, following links to online resources for further information.

Metabolic data visualisation: Load experimental data gathered by mass-spectroscopy (MS) or nuclear magnetic resonance (NMR) spectroscopy and visualise metabolic changes overlaid on a a map of the sytem. Visualise gene-expression or protein quantity data alongside to explore relationships between enzyme regulation and metabolic processes. Use the built-in "Pathway Mining" tools to select the most up, down, or overall regulated pathways in the given system to identify the key mechanisms at work.

CHAPTER 1

Users

1.1 Installation

We've created installation packages for Pathomx for both Windows and MacOS X. Follow the instructions below to install the software.

1.1.1 Windows (64bit)

Download the latest release as a Windows Installer (.msi). Double-click to start the installation process. You may need to accept the installation. Icons will be added to your Start menu and desktop. Launch as normal from there.

1.1.2 MacOS X

Download the latest release as a Mac Disk Image (.dmg). Double-click to open, and drag the Pathomx application into your Applications folder. Launch as normal.

1.1.3 Linux

Coming soon.

More documentation is coming soon, in the meantime there are demos available on the Pathomx website here.

CHAPTER 2

Developers

Below is API documentation for core/plugin developers. This is a work in progress as documentation is added to the source code. You can submit patches via github.

2.1 Developer Installation

If you would like to help with Pathomx development you will need to install a source version of the code. Note: This is not necessary if you just want to contribute plugins, as these can be developed against the binary installation.

2.1.1 Getting Started

The development code is hosted on Github. To contribute to development you should first create an account on Github (if you don't have one already), then fork the pathomx/pathomx repo so you have a personal copy of the code. If you're not familiar with Github, there is a useful guide available here.

On your version of the repo (should be <username>/pathomx) you will see an url to clone the repo to your desktop. Take this and then from the command line (in a folder where you want the code to live) enter:

```
git clone <repository-url>
```

After a while you will get a folder named pathomx containing the code.

The following sections list platform-specific setup instructions required to make Pathomx run. Follow the instructions from the section and then you should be ready to run from the command line using:

```
python -m pathomx.Pathomx
```

2.1.2 Windows

Install Qt5 (Qt5.2) for Windows. Make the decision at this point whether you're installing 64bit or 32bit and stick to it.

Install Python 2.7.6 Windows installer from the Python download site.

You can get Windows binaries for all required Python libraries from the Pythonlibs library. At a minimum you will need to install NumPy, SciPy, Scikit-Learn, Matplotlib. Make sure that the installed binaries match the architecture (32bit/64bit) of the installed Python.

For NMR data processing, you will need to install NMRGlue binaries.

For the dynamic pathway drawing plugin MetaViz you will also need to install Graphviz.

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python -m pathomx.Pathomx
```

2.1.3 MacOS X

The simplest approach to setting up a development environment is through the MacOS X package manager Homebrew. It should be feasible to build all these tools from source, but I'd strongly suggest you save yourself the bother.

Install Homebrew as follows:

```
ruby -e "$(curl -fsSL https://raw.github.com/Homebrew/homebrew/go/install)"
```

Once that is in place use brew install to install python, PyQt5 (which will automatically install Qt5) and graphviz. Install pip for Python and add the packages numpy, scipy, pydot, nmrglue, gpml2svg, poster, wheezy, sklearn, icoshift, matplotlib. This can be done in a one liner with pip:

```
pip install numpy scipy pydot nmrglue gpml2svg poster wheezy sklearn icoshift matplotlib
```

That should be enough to get Pathomx up and running from the command line. For development a useful tool to install is Total Terminal, which gets you access to the command line via a hotkey.

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python -m pathomx.Pathomx
```

2.1.4 Linux

The development version (available via git) supports Python 3 and so can now be run on Linux (tested on Ubuntu Saucy Salamander). Note: Python 3 PyQt5 is only available from 13.10. To install on earlier releases of Ubuntu you will need to install from source.

There are a number of packages that need to be installed first:

```
sudo apt-get install python3-pyqt5 python3-matplotlib python3-requests python3-numpy python3-scipy py

pip3 install scikit-learn
```

Note that aside from python3-pyqt5 you can also install the other packages using pip3 (the names on PyPi are the same as for the packages minus the python3- prefix). Once installation of the above has completed you're ready to go.

To run Pathomx from the command line, change to the cloned git folder and then enter:

```
python -m pathomx.Pathomx
```

2.2 API Reference

The API reference is intended for developers of Pathomx and associated plugins. It lists the interfaces that are available to developers to create views and behaviours. Most complex stuff (e.g. processing thread generation) is handled by the core application and exposed as simplified interfaces for use.

Developing a plugin? Use one of the core plugins as a starting point and use this reference to interpret what it's doing.

2.2.1 Colors

2.2.2 Config

class pathomx.config.**ConfigManager**(*defaults={}*, **args*, ***kwargs*)

add_handler(*key*, *handler*, *mapper=(<function <lambda> at 0x7f5a73f6cd70>, <function <lambda> at 0x7f5a73f6cde8>)*)
Add a handler (UI element) for a given config key.

The supplied handler should be a QWidget or QAction through which the user can change the config setting. An automatic getter, setter and change-event handler is attached which will keep the widget and config in sync. The attached handler will default to the correct value from the current config.

An optional mapper may also be provider to handler translation from the values shown in the UI and those saved/loaded from the config.

add_handlers(*keyhandlers*)

get(*key*)
Get config value for a given key from the config manager.

Returns the value that matches the supplied key. If the value is not set a default value will be returned as set by set_defaults.

Parameters **key** (*str*) – The configuration key to return a config value for

Return type Any supported (str, int, bool, list-of-supported-types)

getXMLConfig(*root*)

remove_handler(*key*)

replace(*keyvalues*, *trigger_update=True*)
Completely reset the config with a set of key values.

Note that this does not wipe handlers or triggers (see reset), it simply replaces the values in the config entirely. It is the equivalent of unsetting all keys, followed by a set_many. Anything not in the supplied keyvalues will revert to default.

Parameters

- **keyvalues** (*dict*) – A dictionary of keys and values to set as defaults
- **trigger_update** (*bool*) – Flag whether to trigger a config update (+recalculation) after all values are set.

reset()
Reset the config manager to it's initialised state.

This clears all values, unsets all defaults and removes all handlers, maps, and hooks.

set(*key*, *value*, *trigger_handler=True*, *trigger_update=True*)
Set config value for a given key in the config manager.

Set key to value. The optional trigger_update determines whether event hooks will fire for this key (and so re-calculation). It is useful to suppress these when updating multiple values for example.

Parameters

- **key** (*str*) – The configuration key to set
- **value** (*Any supported (str, int, bool, list-of-supported-types)*) – The value to set the configuration key to

Return type bool (success)

setXMLConfig(*root*)

set_default(*key*, *value*, *eventhook=1*)
Set the default value for a given key.

This will be returned if the value is not set in the current config. It is important to include defaults for all possible config values for backward compatibility with earlier versions of a plugin.

Parameters

- **key** (*str*) – The configuration key to set
- **value** (*Any supported (str, int, bool, list-of-supported-types)*) – The value to set the configuration key to
- **eventhook** (*int RECALCULATE_ALL, RECALCULATE_VIEWS*) – Attach either a full recalculation trigger (default), or a view-only recalculation trigger to these values.

set_defaults(*keyvalues*, *eventhook=1*)
Set the default value for a set of keys.

These will be returned if the value is not set in the current config. It is important to include defaults for all possible config values for backward compatibility with earlier versions of a plugin.

Parameters

- **keyvalues** – A dictionary of keys and values to set as defaults
- **eventhook** (*int RECALCULATE_ALL, RECALCULATE_VIEWS*) – Attach either a full recalculation trigger (default), or a view-only recalculation trigger to these values.

set_many(*keyvalues*, *trigger_update=True*)
Set the value of multiple config settings simultaneously.

This postpones the triggering of the update signal until all values are set to prevent excess signals. The trigger_update option can be set to False to prevent any update at all.

Parameters

- **keyvalues** – A dictionary of keys and values to set.
- **trigger_update** (*bool*) – Flag whether to trigger a config update (+recalculation) after all values are set.

updated

class pathomx.config.**QSettingsManager**(*defaults={}*, **args*, ***kwargs*)

reset()
Reset the config manager to it's initialised state.

This initialises QSettings, unsets all defaults and removes all handlers, maps, and hooks.

pathomx.config.**build_dict_mapper**(*mdict*)
Build a map function pair for forward and reverse mapping from a specified dict

Mapping requires both a forward and reverse (get, set) mapping function. This function is used to automatically convert a supplied dict to a forward and reverse paired lambda.

Parameters **mdict** (*dict*) – A dictionary of display values (keys) and stored values (values)

Return type 2-tuple of lambdas that perform forward and reverse map

pathomx.config.**build_tuple_mapper**(*mlist*)
Build a map function pair for forward and reverse mapping from a specified list of tuples

Parameters **mlist** (*list-of-tuples*) – A list of tuples of display values (keys) and stored values (values)

Return type 2-tuple of lambdas that perform forward and reverse map

pathomx.config.**types_MethodType**(*fn*, *handler*, *handler_class*)

2.2.3 Custom Exceptions

exception pathomx.custom_exceptions.**PathomxExternalResourceTimeoutException**

exception pathomx.custom_exceptions.**PathomxExternalResourceUnavailableException**

exception pathomx.custom_exceptions.**PathomxIncorrectFileFormatException**

exception pathomx.custom_exceptions.**PathomxIncorrectFileStructureException**

2.2.4 Data

class pathomx.data.**DataDefinition**(*target*, *definition*, *title=None*, **args*, ***kwargs*)

can_consume(*data*)

cmp_map = {u'>=': <built-in function ge>, u'=': <built-in function eq>, u'<=': <built-in function le>, u'aloeic': <function

get_cmp_fn(*s*)

class pathomx.data.**DataManager**(*parent*, *view*, **args*, ***kwargs*)

add_input(*interface*)

add_output(*interface*, *dso=None*, *is_public=True*)

can_consume(*data*, *consumer_defs=None*)

can_consume_which_of(*datalist*, *consumer_defs=None*)

consume(*data*)

consume_any_of(*data_l*)

consume_with(*data*, *consumer_def*)

consumed

get(*interface*)

geto(*interface*)

has_consumable(*manager*)

interfaces_changed

notify_watchers(*interface*)

provide(*target*)

put(*interface*, *dso*, *update_consumers=True*)

refresh_consumed_data()

remove_input(*interface*)

remove_output(*interface*)

reset()

source_updated

stop_consuming(*target*)

stop_providing(*data*)

unconsumed

unget(*interface*)

unput(*interface*)

class pathomx.data.**DataSet**(*manager=None*, *size=(0,)*, *name=u''*, *description=u''*, **args*, ***kwargs*)

annotations_from_XML(*et*, *obj*)

annotations_to_XML(*et*, *obj*, *anno*)

as_class_groups(*d=0*, *fn=<Mock object>*, *classes=None*)

as_copy()

as_filtered(*dim=1*, *scales=None*, *classes=None*, *labels=None*, *entities=None*)

as_summary(*fn=<Mock object>, dim=1, match_attribs=[u'classes', u'labels', u'entities']*)

classes_l

classes_n

classes_t

crop(*shape*)

dimensions

empty(*size=(0,)*)

entities_l

entities_n

entities_t

from_XML()

import_data(*dso*)

is_empty

labels_l

labels_n

labels_t

refresh_interfaces()
Refresh all interfaces, e.g. as_table

register_interface(*interface_name*, *interface*)

remove_invalid_data(*axis=1*)

scales_n

scales_r

scales_t

shape

to_XML()

class pathomx.data.**DataTreeItem**(*dso*, *header*, *parentItem*)
a python object used to return row/column data, and keep note of it's parents and/or children

appendChild(*item*)

child(*row*)

childCount()

columnCount()

data(*column*)

icon()

parent()

row()

class pathomx.data.**DataTreeModel**(*dsos=[]*, *parent=None*)
a model to display a few names, ordered by sex

columnCount(*parent=None*)

data(*index*, *role*)

headerData(*column*, *orientation*, *role*)

index(*row*, *column*, *parent*)

parent(*index*)

refresh()

rowCount(*parent=<class 'QModelIndex'>*)

setupModelData()

class pathomx.data.**TableInterface**(*dso*, **args*, ***kwargs*)

columnCount(*parent*)

data(*index*, *role*)

headerData(*col*, *orientation*, *role*)

refresh()

rowCount(*parent*)

sort(*col*, *order*)
sort table by given column number col

pathomx.data.**at_least_one_element_in_common**(*l1*, *l2*)

2.2.5 DB

class pathomx.db.**Compound**(***entries*)

as_csv()

type = u'compound'

type_name = u'Compound'

url

class pathomx.db.**Gene**(***entries*)

as_csv()

type = u'gene'

type_name = u'Gene'

url

class pathomx.db.**Pathway**(***entries*)

as_csv()

type = u'pathway'

type_name = u'Pathway'

url

class pathomx.db.**Protein**(***entries*)

as_csv()

type = u'protein'

type_name = u'Protein'

url

class pathomx.db.**Reaction**(***entries*)

as_csv()

compounds

secondary_compounds

type = u'reaction'

type_name = u'Reaction'

url

class pathomx.db.**ReactionIntermediate**(***entries*)

name = u'n/a'

type = u'dummy'

type_name = u'n/a'

class pathomx.db.**databaseManager**

add_compound(*id*, *attr*)

add_db_synonyms(*id*, *databases*)

add_gene(*id*, *attr*)

add_identity(*id*, *identity*)

add_pathway(*id*, *attr*)

add_protein(*id*, *attr*)

add_reaction(*id*, *attr*)

add_synonym(*id*, *synonym*)

add_synonyms(*id*, *synonyms*)

extract_db_unification(*db_unification*)

get_via_synonym(*id*)

get_via_unification(*database*, *id*)

load_compounds()

load_genes()

load_gibbs()

load_identities()

load_pathways()

load_proteins()

load_reactions()

load_synonyms(*filename=u'/home/docs/checkouts/readthedocs.org/user_builds/pathomx/checkouts/stable/pathomx/database*

load_xrefs()

save_compounds()

save_genes()

save_pathways()

save_proteins()

save_reactions()

save_synonyms()

2.2.6 Pathomx

2.2.7 Plugins

2.2.8 Qt

2.2.9 Threads

class pathomx.threads.**Worker**(*callback*, **args*, ***kwargs*)

Worker thread

Inherits from QRunnable to handler worker thread setup, signals and wrap-up.

Parameters

- **callback** (*function*) – The function callback to run on this worker thread. Supplied args and kwargs will be passed through to the runner.
- **args** – Arguments to pass to the callback function
- **kwargs** – Keywords to pass to the callback function

process(**args*, ***kwargs*)

run

class pathomx.threads.**WorkerSignals**

Defines the signals available from a running worker thread.

Supported signals are:

finished No data

error *tuple* (exctype, value, traceback.format_exc())

result *dict* data returned from processing

status *str* one of standard status flag message types

error

finished

result

status

2.2.10 Translate

pathomx.translate.**tr**(*s*, **args*, ***kwargs*)

2.2.11 UI

2.2.12 Views

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

p

A

B

C

D

S

T

U

W