pySUMO Documentation

Release 1.0

Kent, Konstantin, Philipp, Simon

May 27, 2015

Contents

1 Introduction **1**

2 Structure **2**
- 2.1 Model . . . 2
- 2.2 Controller . . . 2
- 2.3 View . . . 2

3 UML Diagrams: **3**
- 3.1 Sequence diagram . . . 3
- 3.2 Activity diagram . . . 5
- 3.3 Overview: Class Diagram of pySUMO . . . 9

4 Detail: Modules **10**
- 4.1 pysumo package . . . 10
- 4.2 pySUMOQt package . . . 17

5 pySUMO in usage **23**
- 5.1 PySUMO in Use . . . 23

6 Changes in Version 1.0 **27**
- 6.1 pysumo . . . 27
- 6.2 pySUMOQt . . . 28

7 Feature Request **31**
- 7.1 Feature Requests . . . 31

8 Tests **32**
- 8.1 Library . . . 32
- 8.2 GUI . . . 33

9 Test Results **40**
- 9.1 Statement Coverage of pysumo and pySUMOQt . . . 40

10 Errors **42**

11 External Libraries **43**

12 Indices and tables **44**

Python Module Index **45**

CHAPTER 1

Introduction

Welcome to pySUMOs documentation. We hope you will find any information you need to use or develop pySUMO. If you have any questions after reading this, feel free to contact us via email at pysumo@lists.kit.edu.

First, a short excursion to the "Model View Controller" design pattern is made.

Subsequently, our Class Diagram, which is structured according to the Model View Controller (MVC) design.

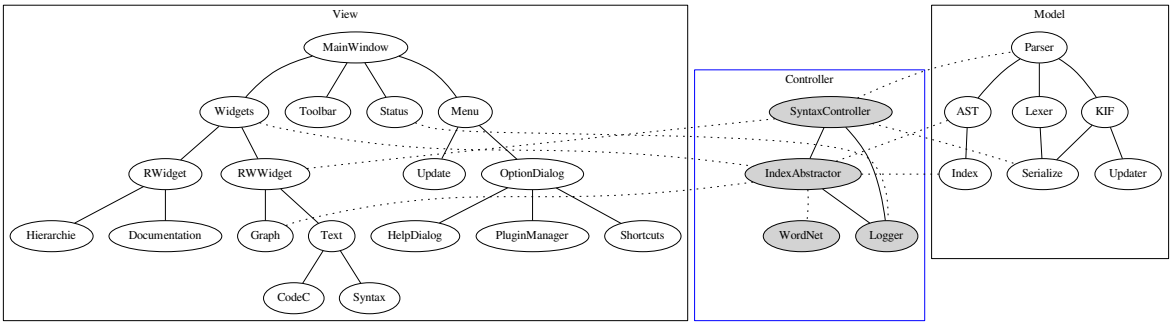
Afterwards, several sequence diagrams which show a timeline of typical actions are presented.

Furthermore, there are a couple of activity diagrams showcasing both a casual pySUMO workflow as well as what pySUMO does in the background to support your workflow.

Thereafter the individual classes are introduced in more detail. This is followed up by a list of errors that pySUMO can throw and a list of external libraries used to develop pySUMO.

CHAPTER 2

Structure



The program is structured around the MVC-architecture. The partitioning into Model, View and Controller makes modification of Program code easier and also allows the lib to be used independently of the GUI.

For more information see

2.1 Model

The model contains both the Parser and the Abstract Syntax Tree (AST) generated by the Parser.
The AST stores the Ontology in memory and allows modifications thereof.
The Parser provides both a kif-parser and serializer as well as a parser for the SUMO-WordNet mapping.

2.2 Controller

The controller contains all base functionality of the program. It mediates all accesses to the AST, creates and maintains an index and handles interactions with WordNet. The controller also handles all other core functionality of pySUMO such as undo and redo.

2.3 View

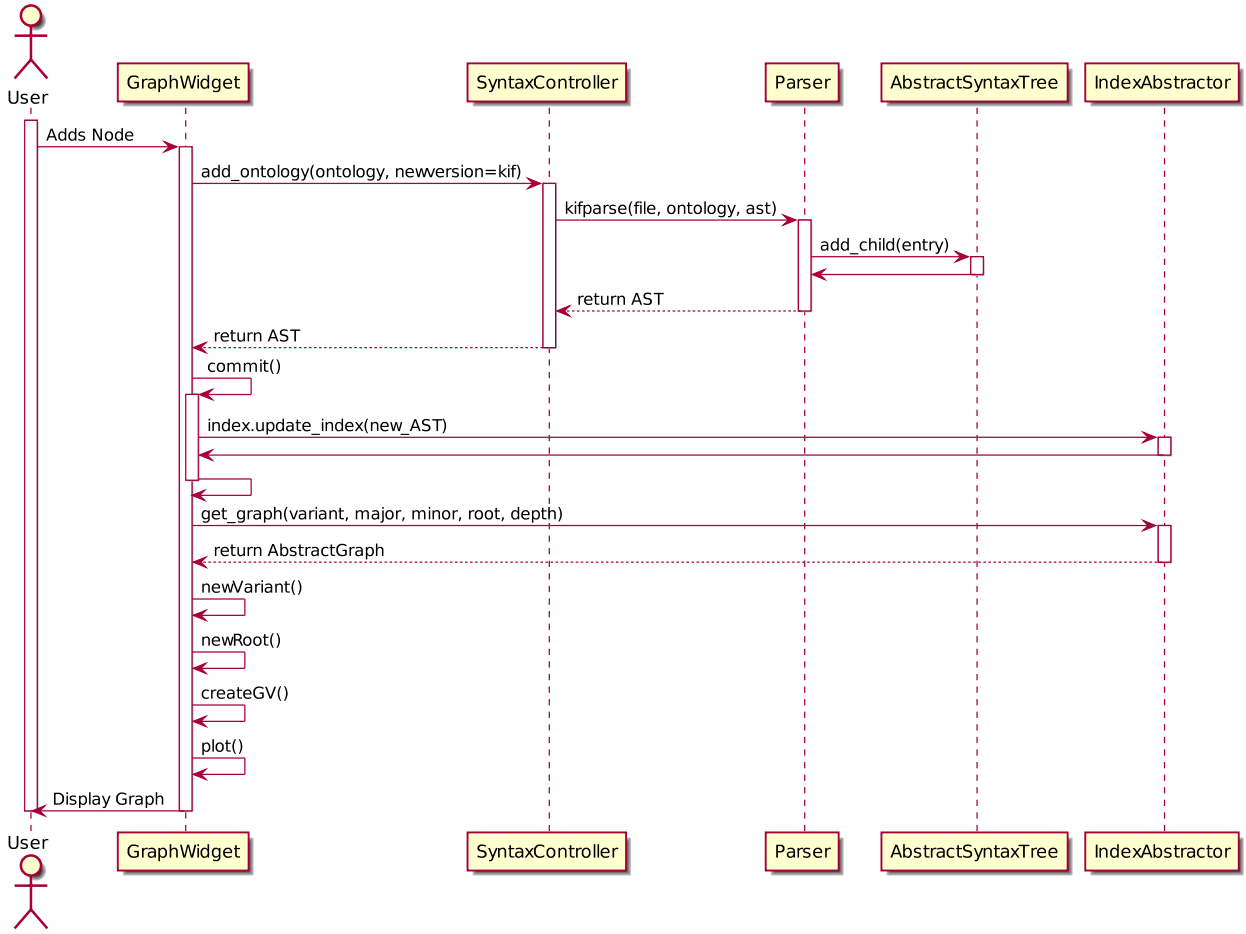
The view consists of all the graphical elements of pySUMO including, but not limited to the Qt GUI.

CHAPTER 3

UML Diagrams:

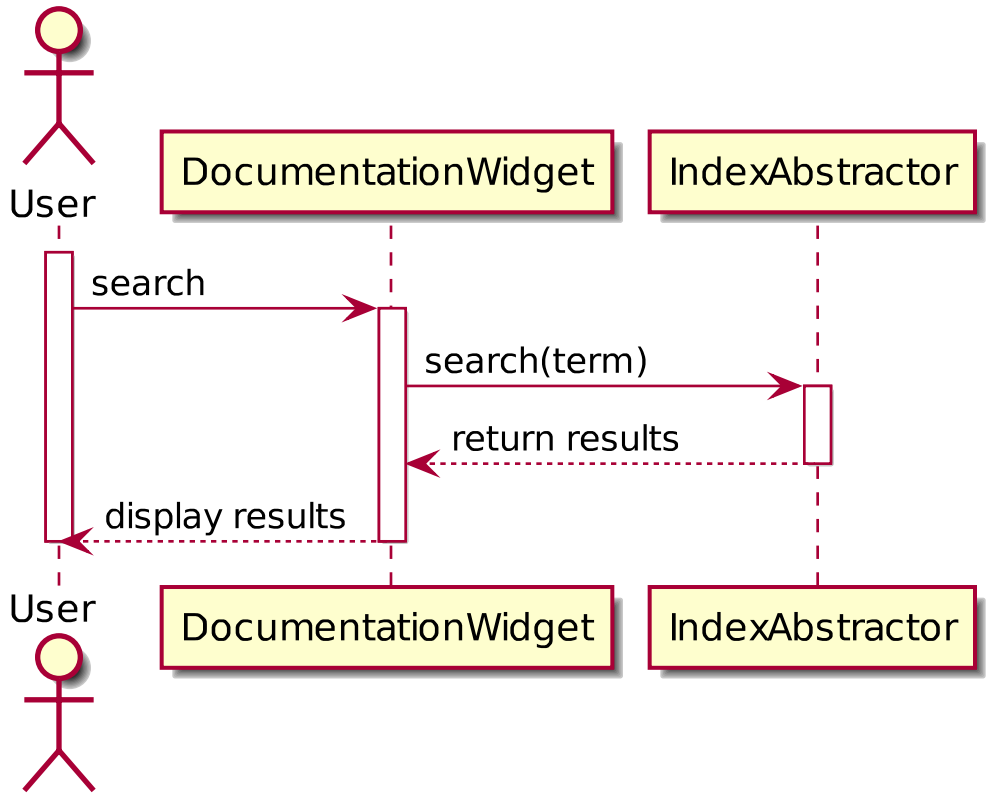
3.1 Sequence diagram

3.1.1 Adding graph node



This diagram displays how the GraphWidget responds to user input and adds new nodes to the Ontology. Note that all operations that might change the Ontology pass through the SyntaxController and all operations that read from the loaded Ontologies pass through the IndexAbstractor.

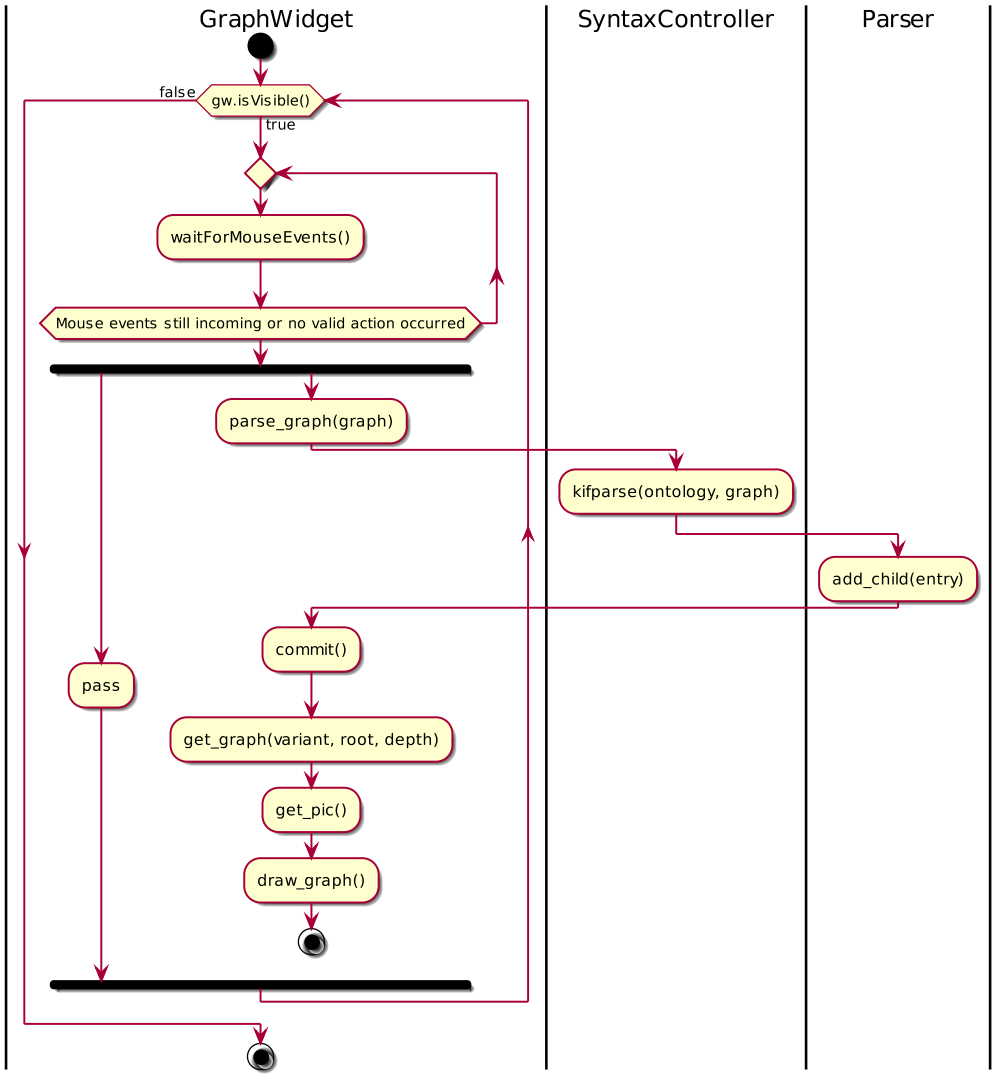
3.1.2 Searching in the index



This diagram displays a simple abstraction of how the DocumentationWidget responds to a user search request. Note that it does not include any IndexAbstractor internal operations such as building and maintaining the Index.

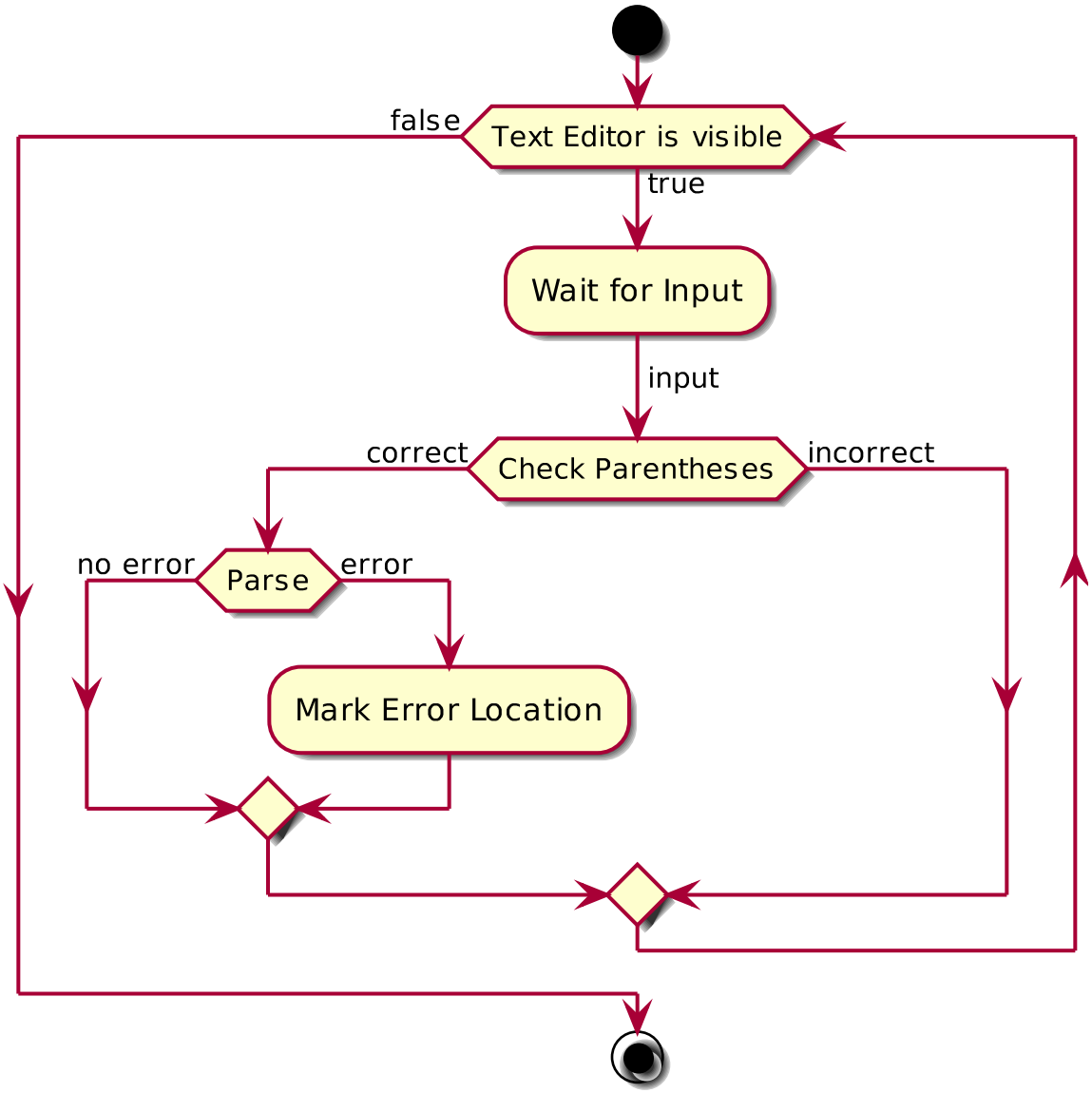
3.2 Activity diagram

3.2.1 Graph widget input loop



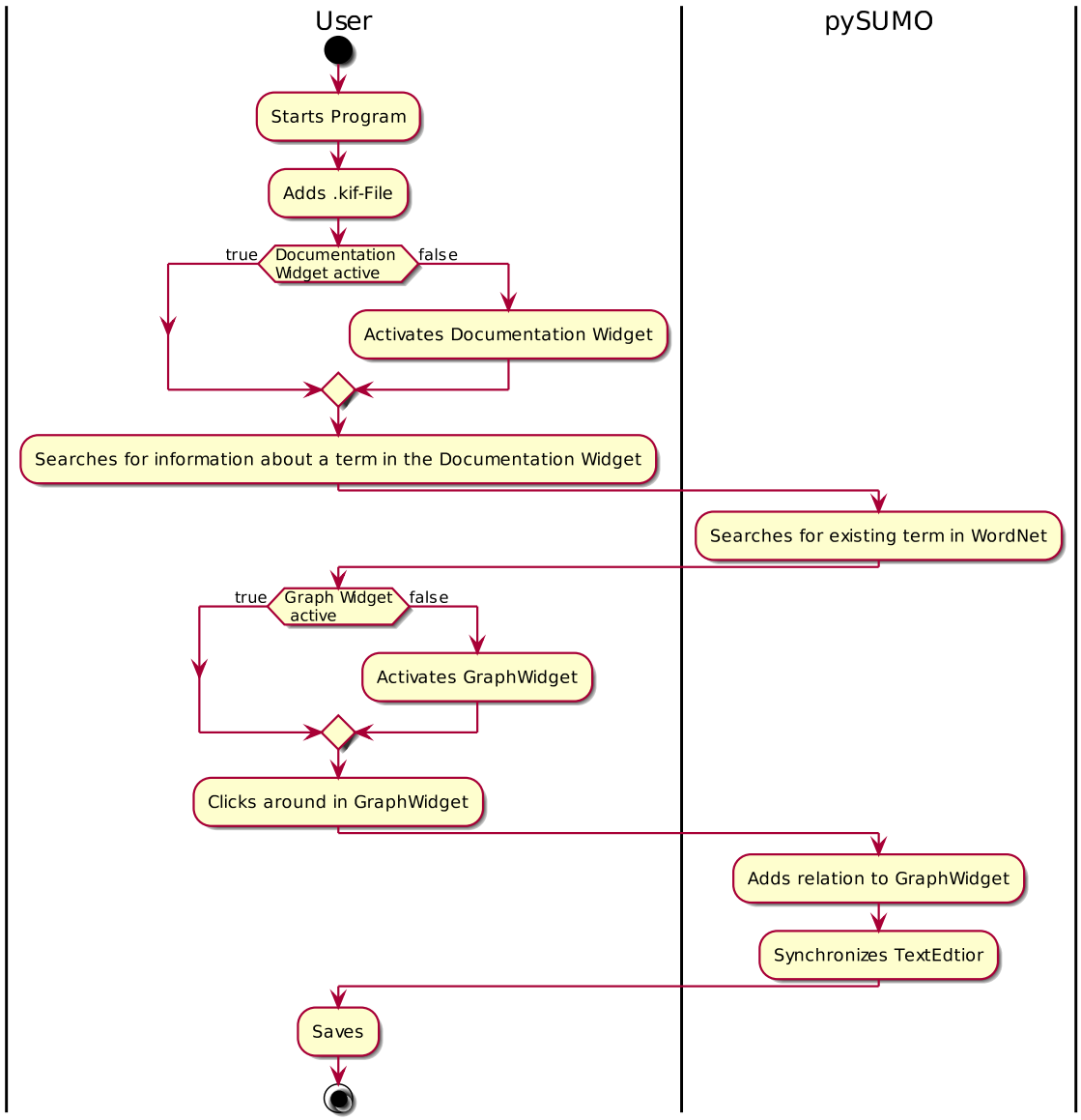
This diagram displays the activity diagram used by the GraphWidget when a user adds a node to the graph. Of special note is that the parsing of the new graph is done in the background so that the user can still interact with the GraphWidget while it is updating the Ontology.

3.2.2 Text editor input loop



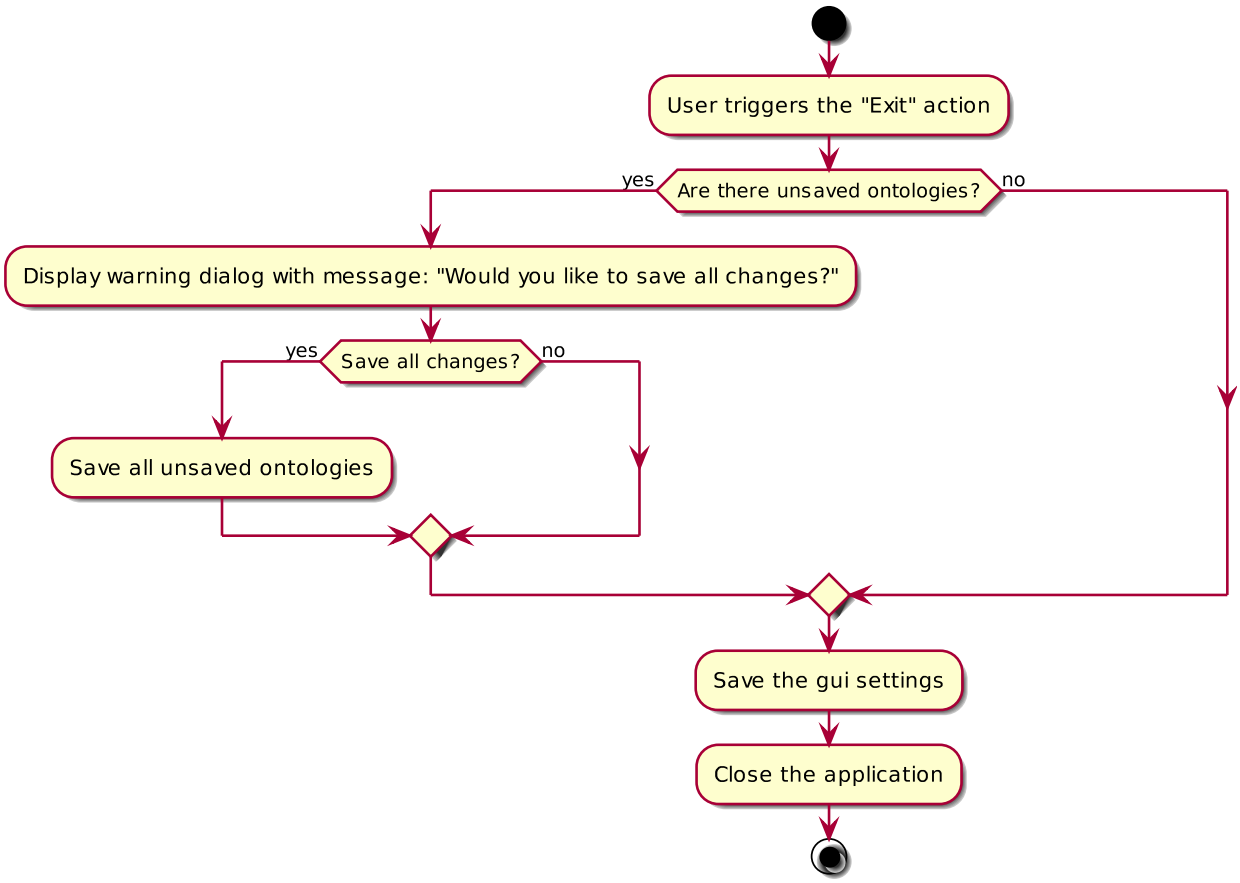
This diagram displays the activity diagram used by the TextWidget while the user is typing. Once the user pauses typing, the text editor checks whether the parenthetical syntax is correct, and if it passes a simple syntax check, the text is passed to the parser. The parser then checks syntax and semantics and returns accordingly. If errors occur during parsing, they are displayed in the editor.

3.2.3 Common workflow



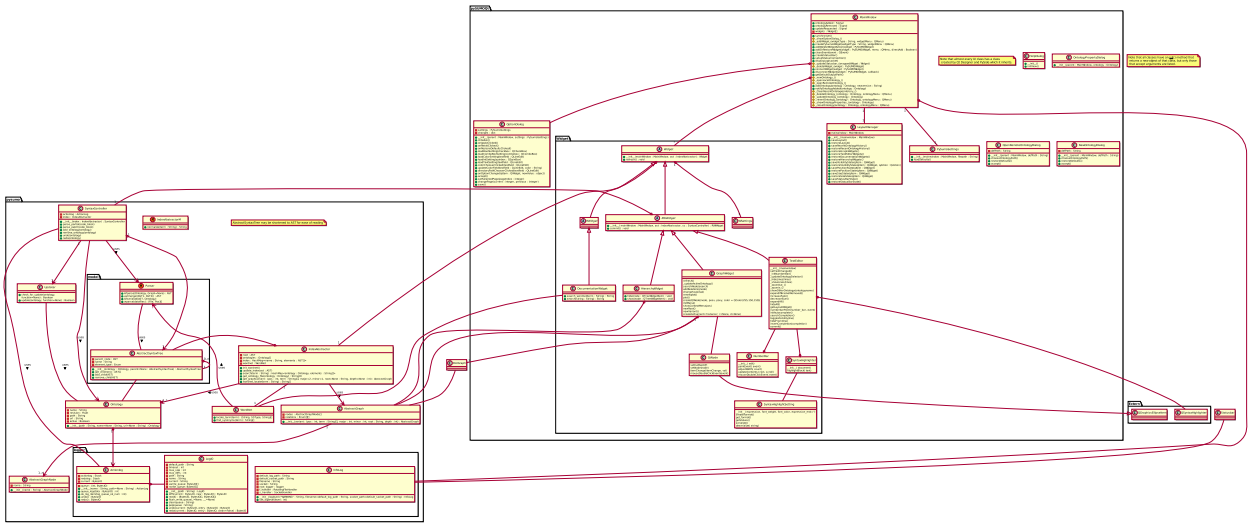
This diagram displays an abstract view of a common User-interaction with the program. The user starts the program adds a .kif-File, searches in WordNet and finally adds a relation through the GraphWidget. PySUMO automatically converts the relation to kif and appends it to the ontology source.

3.2.4 Exit process



This diagram displays an abstract view of the activity diagram when a user wants to exit the program. This is done when the user clicks on the close icon in the title bar of the main window of pySUMO or on 'Exit' in the menu 'File'. The program makes sure that there are no unsaved changes in open ontologies and if there are changes, prompts if they should be saved. If the user doesn't save them, the modifications will be lost. Note that the program also saves session layouting preferences of the GUI so they can be restored in the next pySUMO session. These preferences include, for example, the position of widgets in the main window.

3.3 Overview: Class Diagram of pySUMO

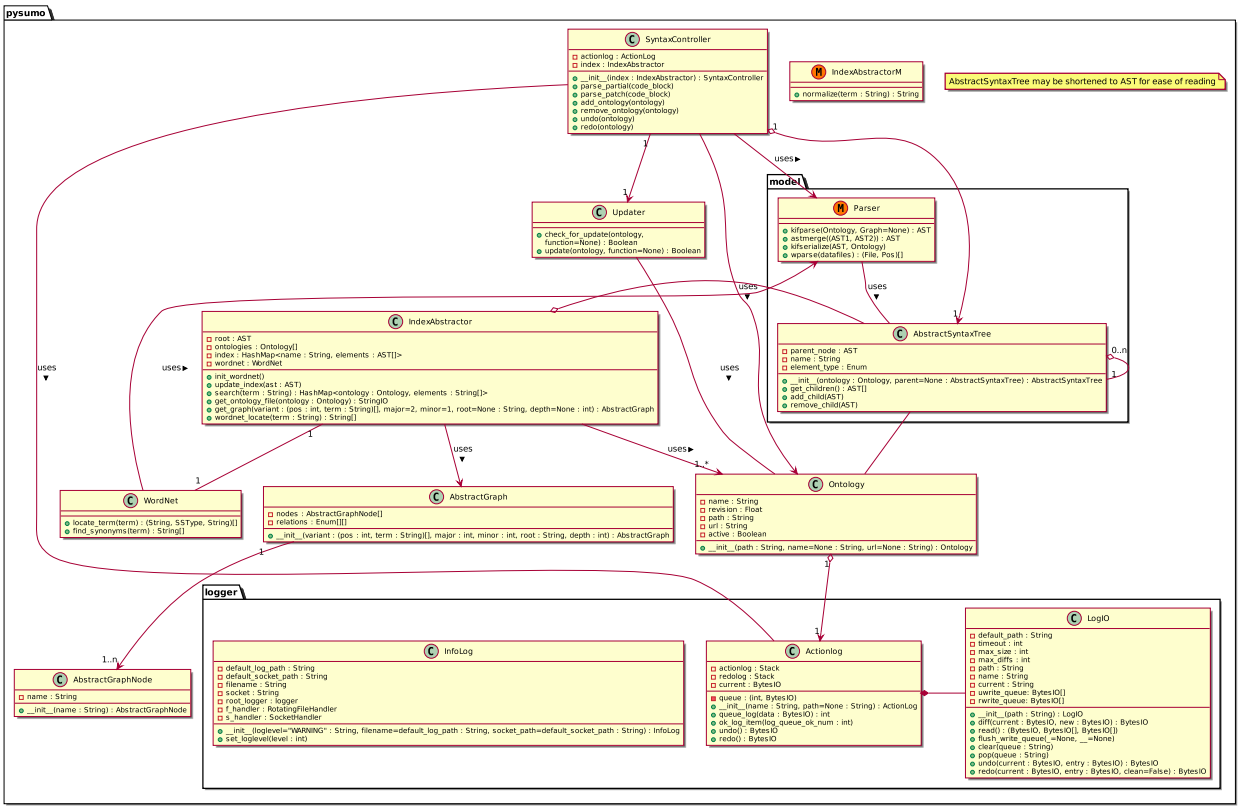


CHAPTER 4

Detail: Modules

4.1 pysumo package

This section describes the backing pysumorary of pySUMO. It contains listings and descriptions of all the modules and classes in the pySUMO core. The pysumo conforms to PEP 008.



4.1.1 Module contents

The pySUMO library package. This package contains the core functionality of pySUMO. It not only provides the GUI with a feature-rich API, but also validates input and keeps the Ontologies in a consistent state.

4.1.2 Subpackages

pysumo.logger package

Module contents

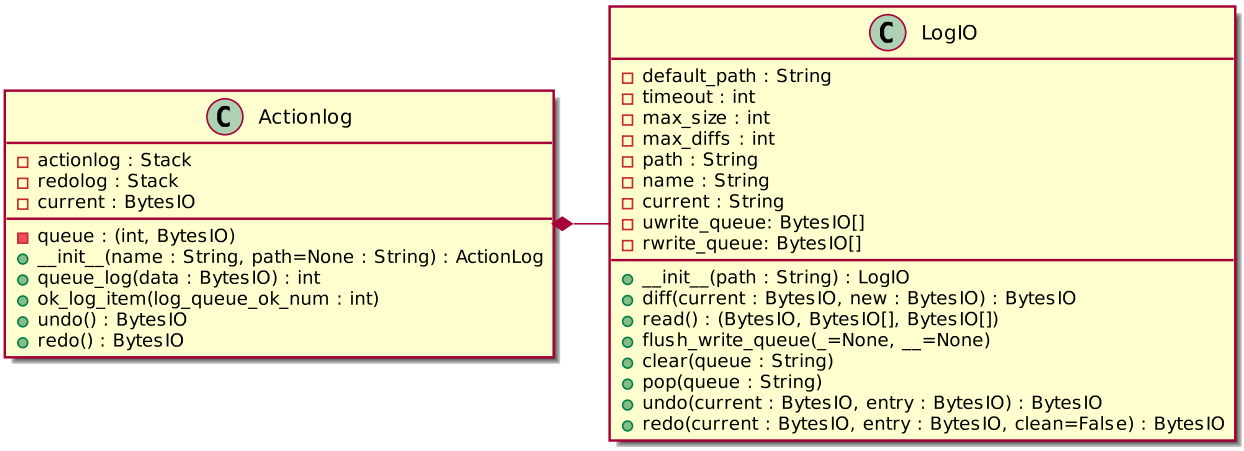
The pySUMO logging interface.

This package contains 2 modules:

- infolog: A singleton called from the entry point to initialize the python logging framework and set the loglevel.
- actionlog: Stores a list of all write operations on the Ontology and provides undo and redo functionality.

Submodules

pysumo.logger.actionlog module



The pySUMO action log handler. This module handles undo and redo operations.

This module contains:

- ActionLog: The action log handler.
- LogIO: The action log io interface.

class `pysumo.logger.actionlog.`**`ActionLog`**(*name*, *path=None*)

The pySUMO action log. The SyntaxController queues a new log entry before every operation that makes changes to an Ontology, if the change is successful it OKs the entry in the log queue and the entry is written out. Log entries that are not OKed time out and are removed from the queue.

Variables:

•log_io: The io object for this log.

•queue: A queue of actions that have not yet successfully completed.

•actionlog: A list of actions that have completed successfully.

•redolog: A list of actions that have been undone successfully.

•current: The current state of the Ontology.

Methods:

•queue_log: Create a log entry and append it to the log queue.

•ok_log_item: Move log entry from log queue to actual log.

•undo: Undoes the last action.

•redo: Redoes the last undone action.

ok_log_item(*log_queue_ok_num*)
Appends the item in self.queue with log_queue_ok_num to self.actionlog and calls self.log_io.append_write_queue on it.

Args:

•log_queue_ok_num: the number of the queue item to okay

Raises:

•KeyError

queue_log(*data*)
Create a log entry and queue it for addition to self.actionlog.

Args:

•data: the data to be placed in the log

Returns:

•int. The log_queue_ok_num

redo()
Redoes the last undone action, appends it to self.undolog and removes it from self.redolog.

undo()
Undoes the last action and appends it to self.redolog.

class pysumo.logger.actionlog.**LogIO**(*name*, *path='/home/docs/.pysumo/actionlog'*)
The IO interface for the pySUMO action log. This class provides a storage backend for the Action Log. Entries in the write queue are written to disk after a timeout, or when the write queue reaches a maximum size.

Variables:

•default_path: The default log path.

•timeout: The time period after which if no new packets have entered the queue, the queue is flushed.

•max_size: The maximum number of actions in the write queue after which when another packet enters the queue, the queue is flushed.

•max_diffs: When the number of stored diffs exceeds max_diffs, old diffs will be deleted.

•path: The log path (defaults to default_path).

•name: The name of the Ontology.

•current: The path to the current state of the Ontology.

•uwrite_queue: The queue in which undo actions are stored before being written to disk.

•rwrite_queue: The queue in which redo actions are stored before being written to disk.

Methods:

•diff: Creates a diff between 2 Files

- read: Instantiates an Action Log with the data in the stored log at path.
- flush_write_queues: Appends all entries in the write queue to the log file.
- clear: Clears a queue in memory and on disk.
- pop: Removes the last entry from a queue.
- undo: Appends an entry to the redo write queue.
- redo: Appends an entry to the undo write queue.

clear (*queue*)
Clears queue both in memory and on disk.

default_path = '/home/docs/.pysumo/actionlog'

diff (*current*, *new*)
Returns a diff between current and new.

flush_write_queues (*_=None*, *__=None*)
Flush self.rwrite_queue and self.uwrite_queue to disk.

max_diffs = 100

max_size = 10

pop (*queue*)
Removes the last entry in queue.

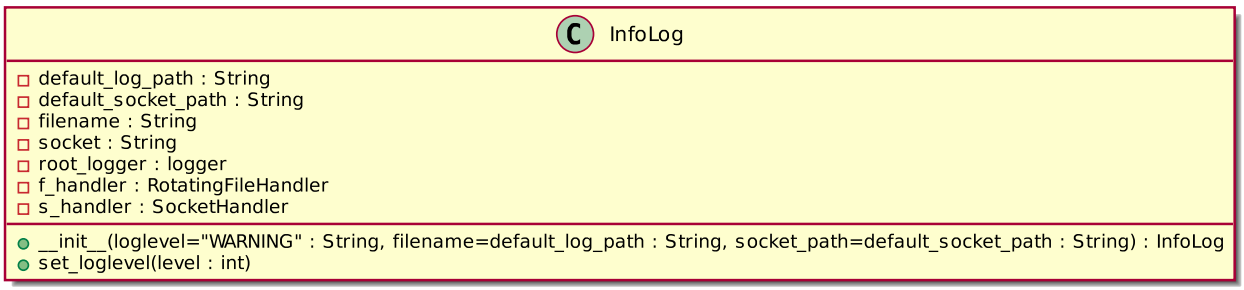
read ()
Reads the log at self.path into log.

redo (*current*, *entry*, *clean=False*)
Append entry to self.uwrite_queue. If clean is True, pop an object from the redo queue.

timeout = 10

undo (*current*, *entry*)
Append entry to self.rwrite_queue.

pysumo.logger.infolog module



The pySUMO informational log handler. Acts as an initializer for the python logging framework and defines several convenience functions for working with it.

This module contains:

- InfoLog: The informational log handler.

class pysumo.logger.infolog.**InfoLog**(*loglevel='WARNING'*, *filename='/home/docs/.pysumo/log'*, *socket_path='/home/docs/.pysumo/status'*)

The informational log handler for pySUMO. Initializes the python logging framework and contains several convenience functions. Instantiated only from the entry point.

Variables:

- default_log_path: The default path where the infolog will be stored.
- default_socket_path: The default socket to which >=INFO logs will be sent.
- filename: The location at which the infolog will be stored.
- root_logger: The root logging object of which all other loggers are children.
- f_handler: Sends messages to a file and rotates it when it becomes too large.
- s_handler: Sends messages to a Unix socket.

Methods:

- set_loglevel: Sets the loglevel above which to log messages.

default_log_path = '/home/docs/.pysumo/log'

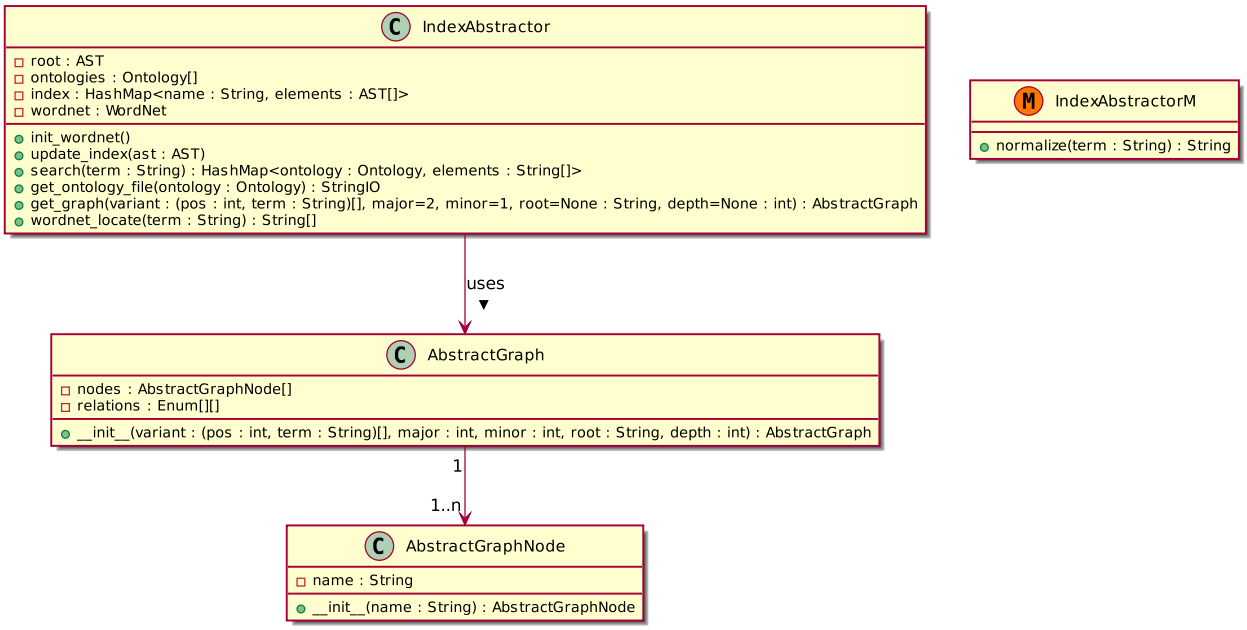
default_socket_path = '/home/docs/.pysumo/status'

set_loglevel(*loglevel*)

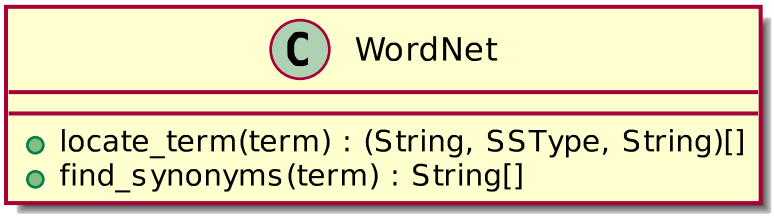
Sets the loglevel above which to log messages.

4.1.3 Submodules

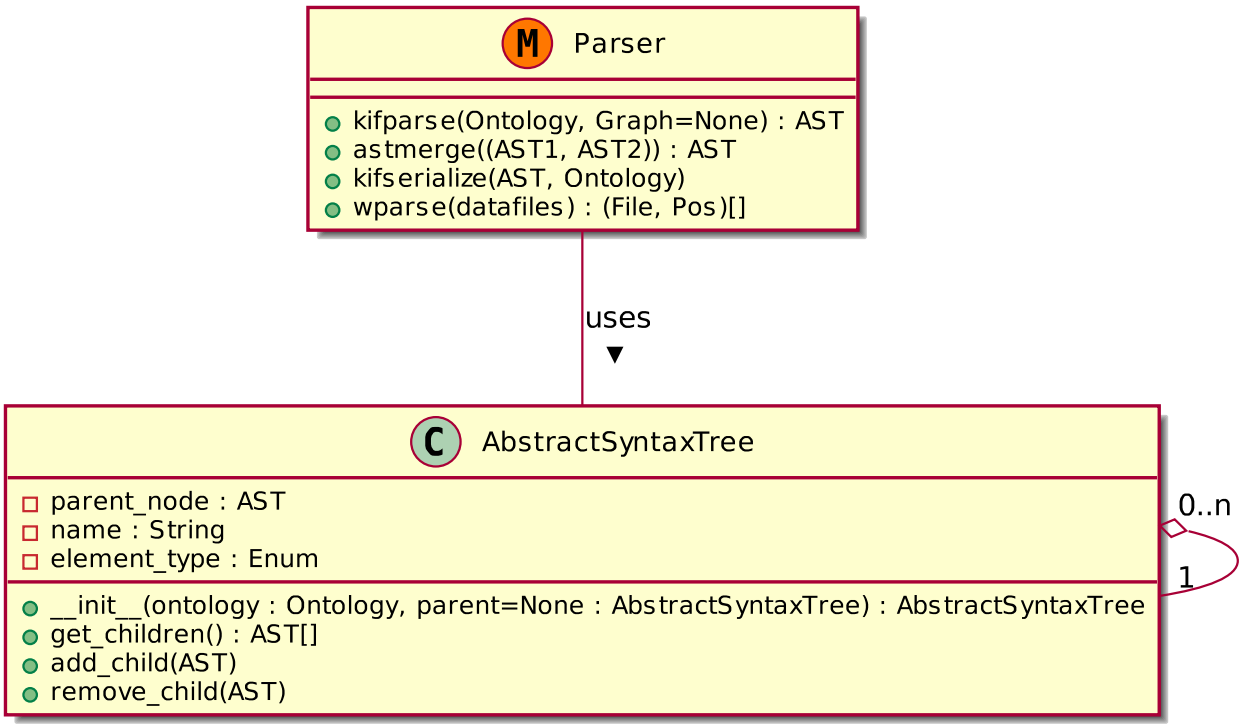
4.1.4 pysumo.indexabstractor module



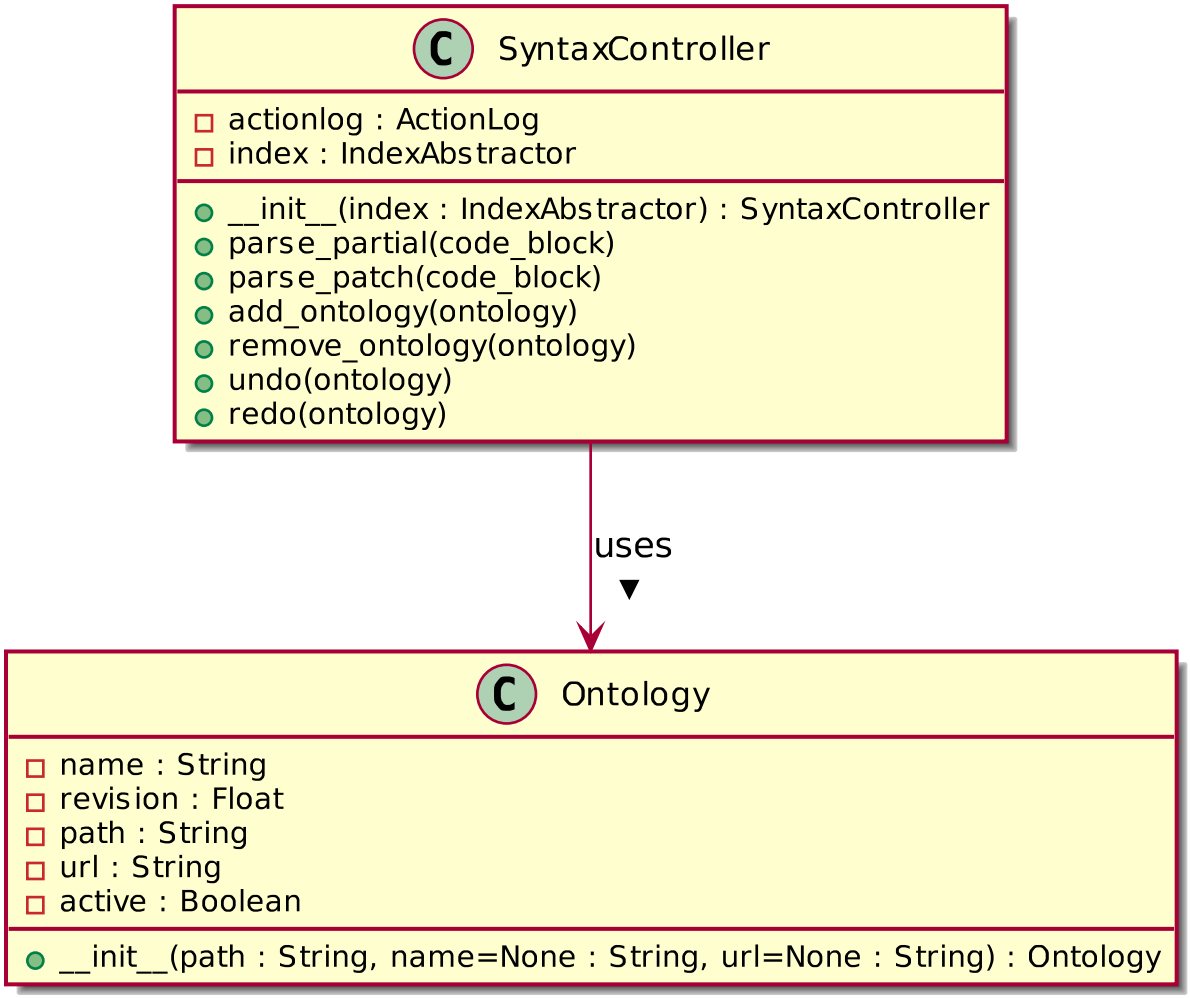
4.1.5 pysumo.wordnet module



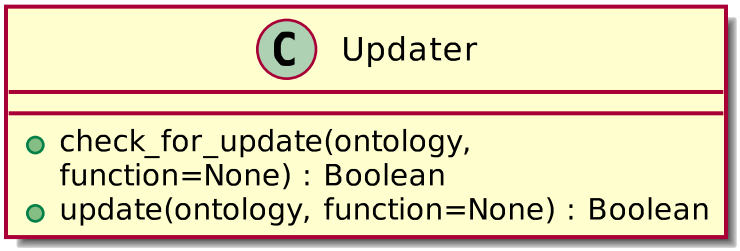
4.1.6 pysumo.parser module



4.1.7 pysumo.syntaxcontroller module

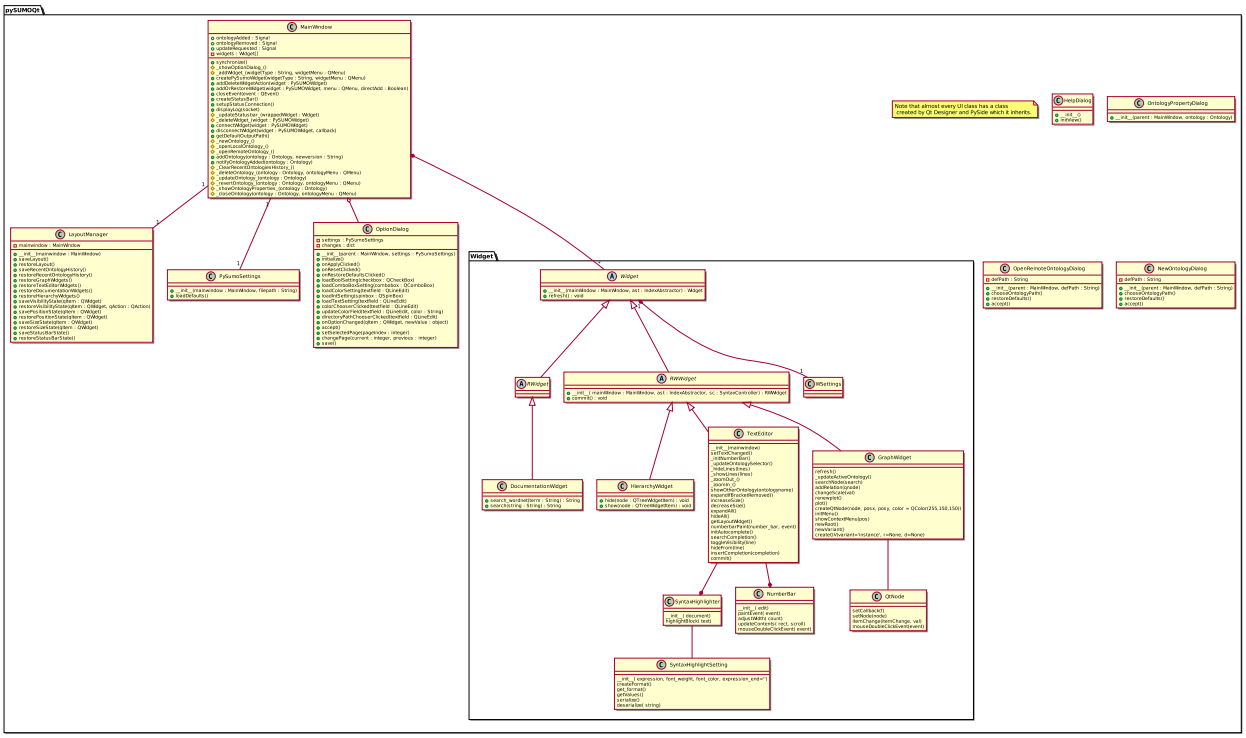


4.1.8 pysumo.updater module



4.2 pySUMOQt package

This section describes the graphical user interface of pySUMO. It will list all modules and classes which are necessary to connect the functionality of the pySUMO's lib with the view and usability of a great editor. For details on this concept see the section about MVC-Structure. Note that the core work is done by Qt and the Qt Designer, so almost all classes will inherit a class designed by Qt Designer and compiled by PySide.



4.2.1 Module contents

The pySUMO Qt-GUI package. This package contains all classes and widgets needed to display the pySUMO editor. There are also classes created by Qt Designer which are inherited from in the 'coded' classes.

4.2.2 Subpackages

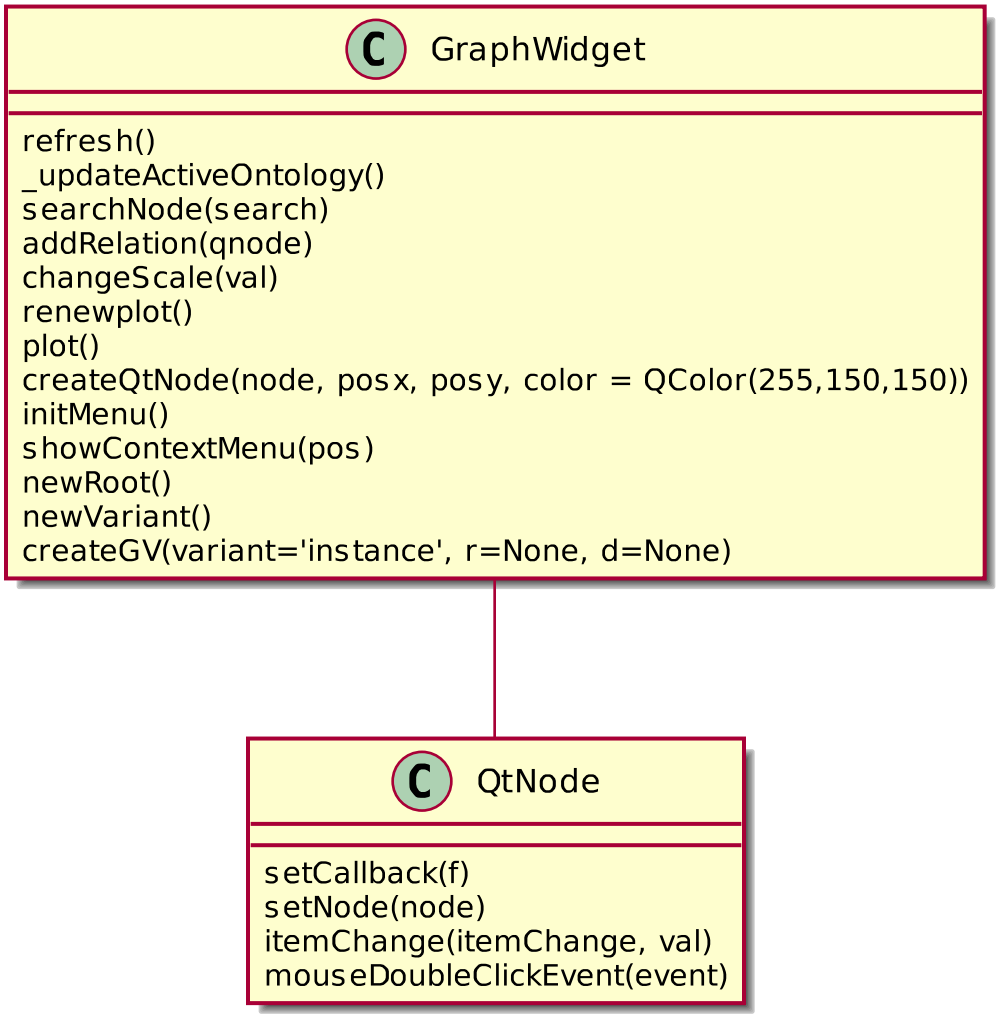
pySUMOQt.Widget package

Module contents

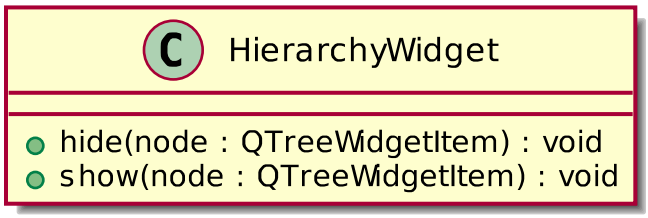
The pySUMO package containing all widgets that interface with the Ontology through the IndexAbstractor and/or the SyntaxController.

Submodules

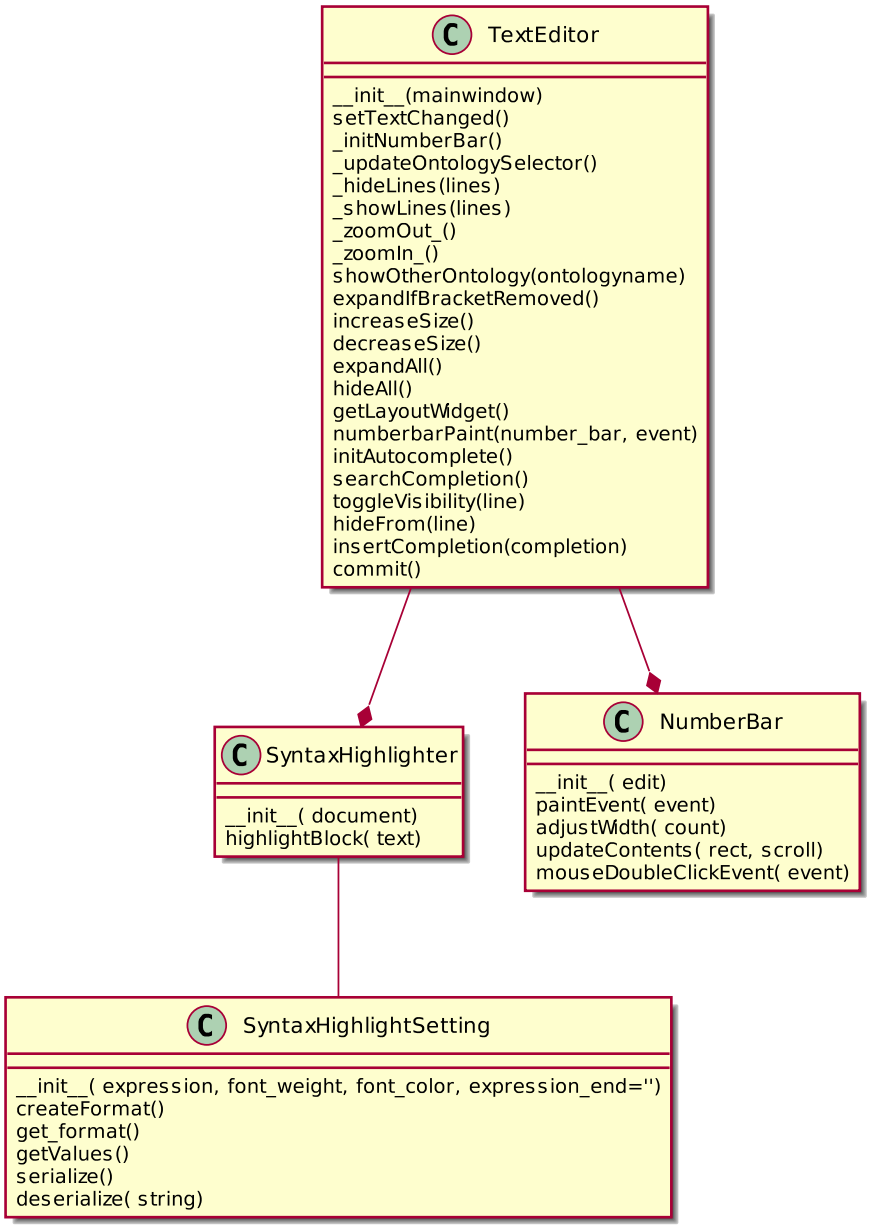
pySUMOQt.Widget.GraphWidget module



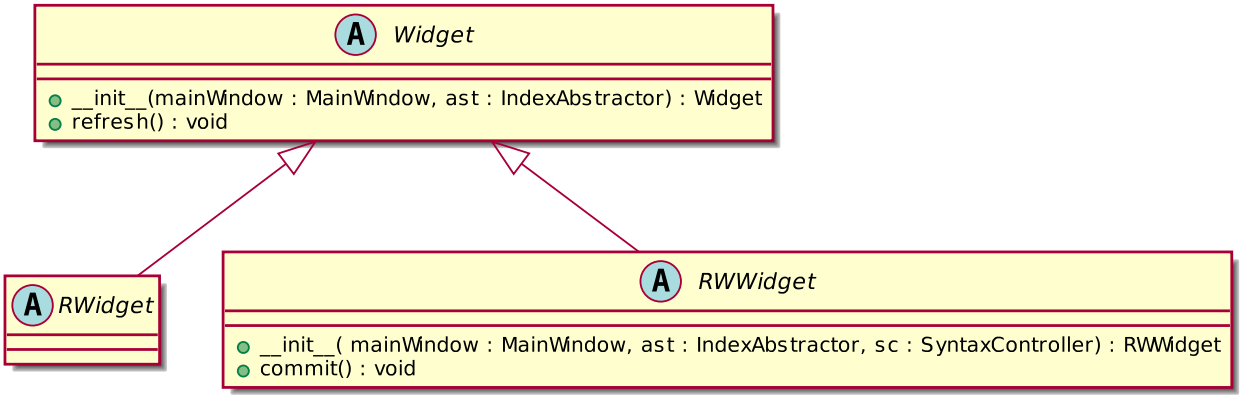
pySUMOQt.Widget.HierarchyWidget module



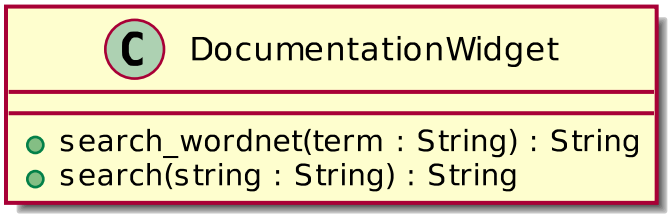
pySUMOQt.Widget.TextEditor module



pySUMOQt.Widget.Widget module

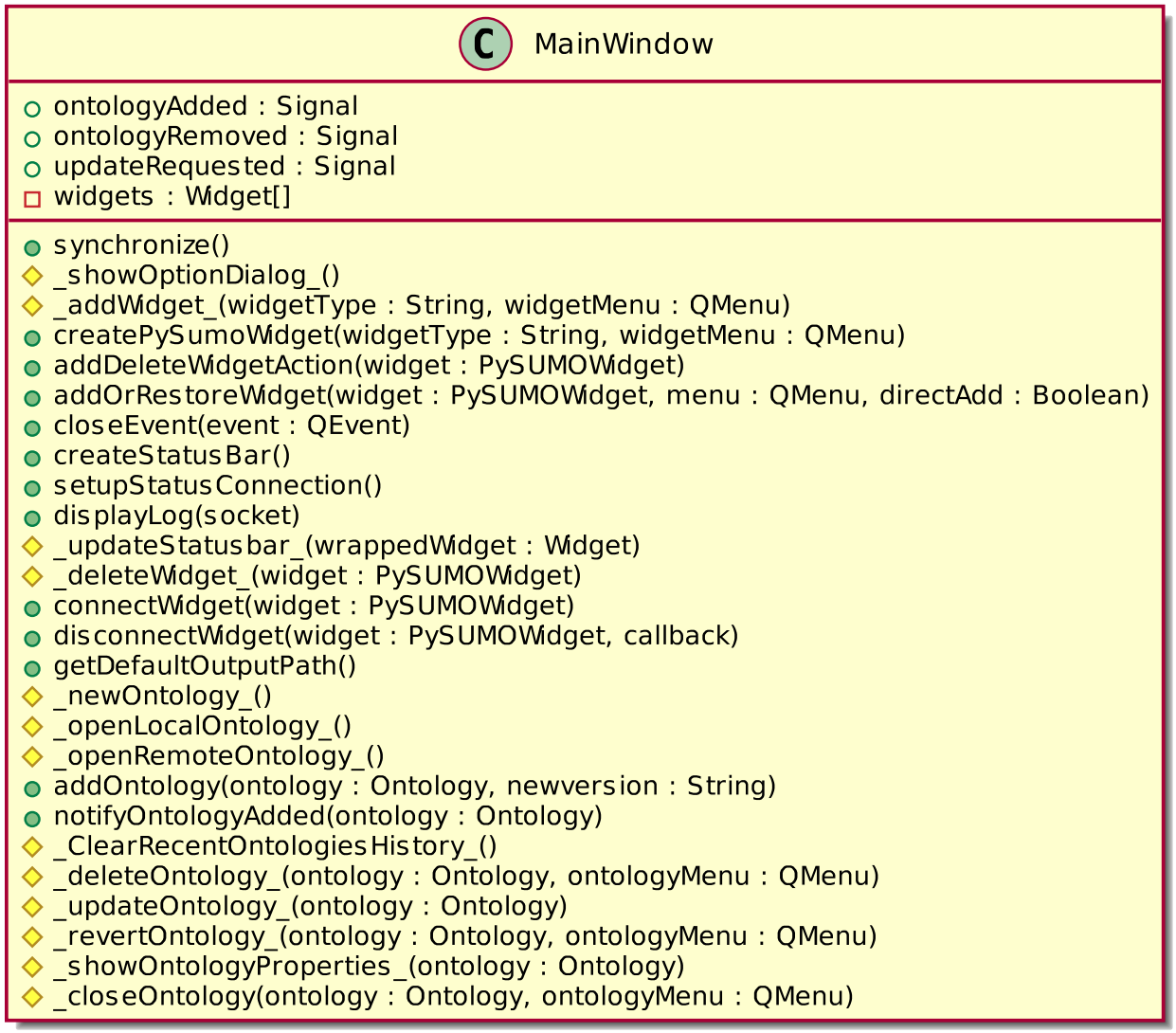


pySUMOQt.Widget.DocumentationWidget module

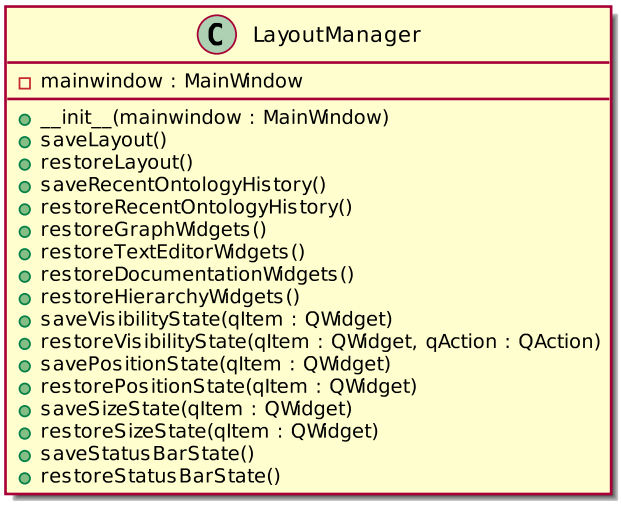


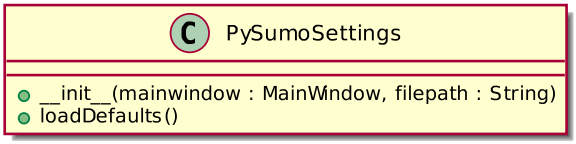
4.2.3 Submodules

4.2.4 pySUMOQt.MainWindow module



4.2.5 pySUMOQt.Settings module





CHAPTER 5

pySUMO in usage

For screenshots of pysumo while using it, go here

5.1 PySUMO in Use

Some screenshots while testing functionality of pySUMO or just klicking around.

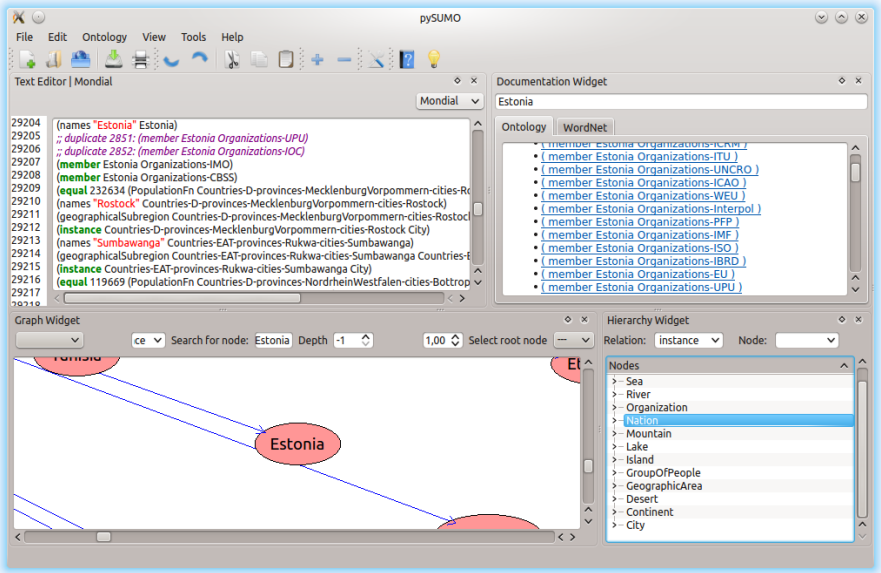


Fig. 5.1: Widgets ordering.

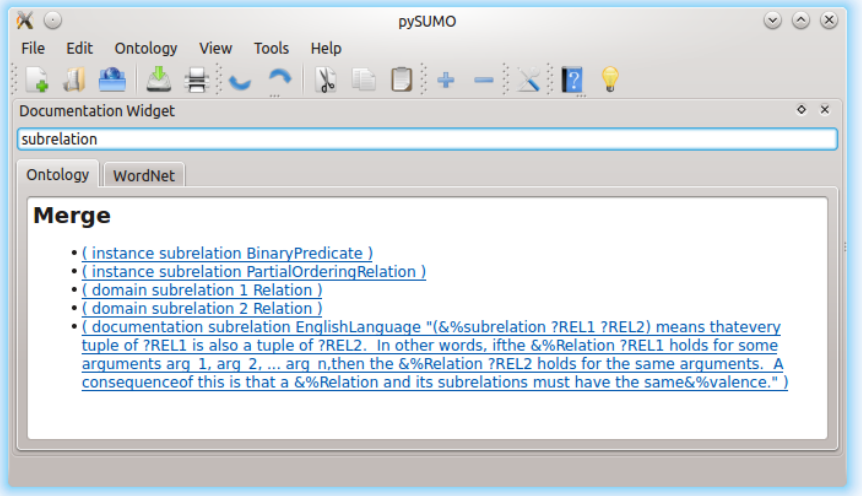


Fig. 5.2: Documentation widget after step 6.

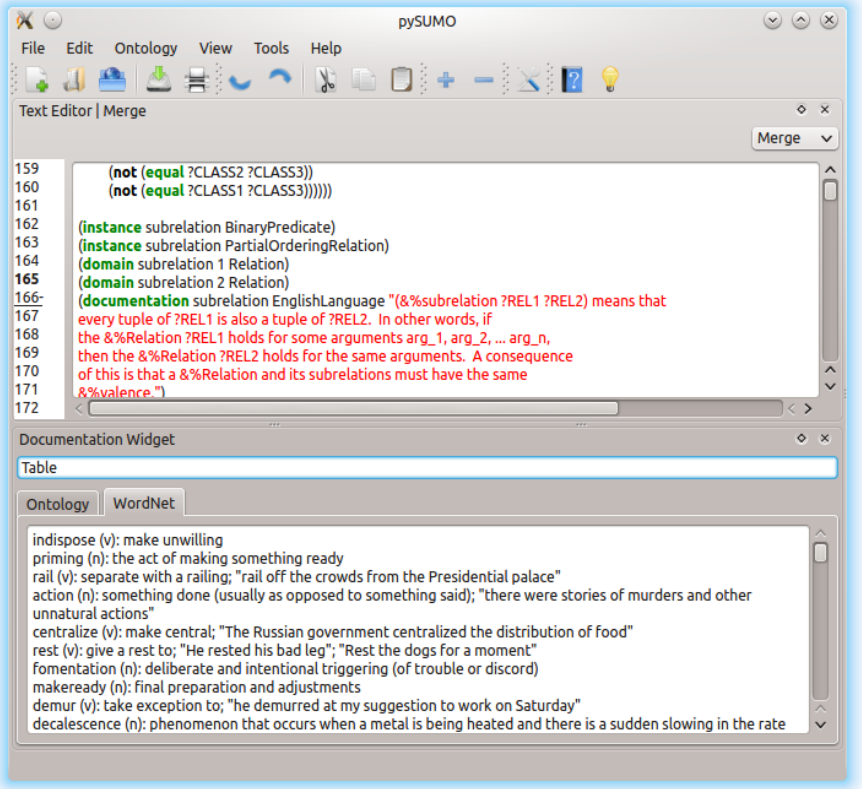


Fig. 5.3: Docwidget after doing test.

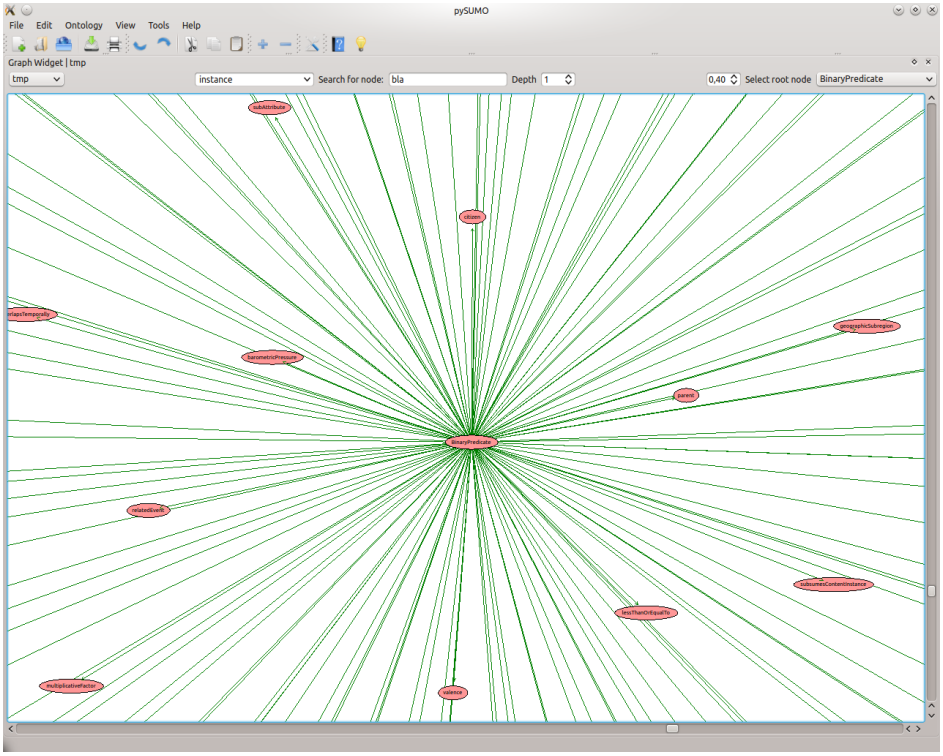


Fig. 5.4: Graphwidget after step 6.

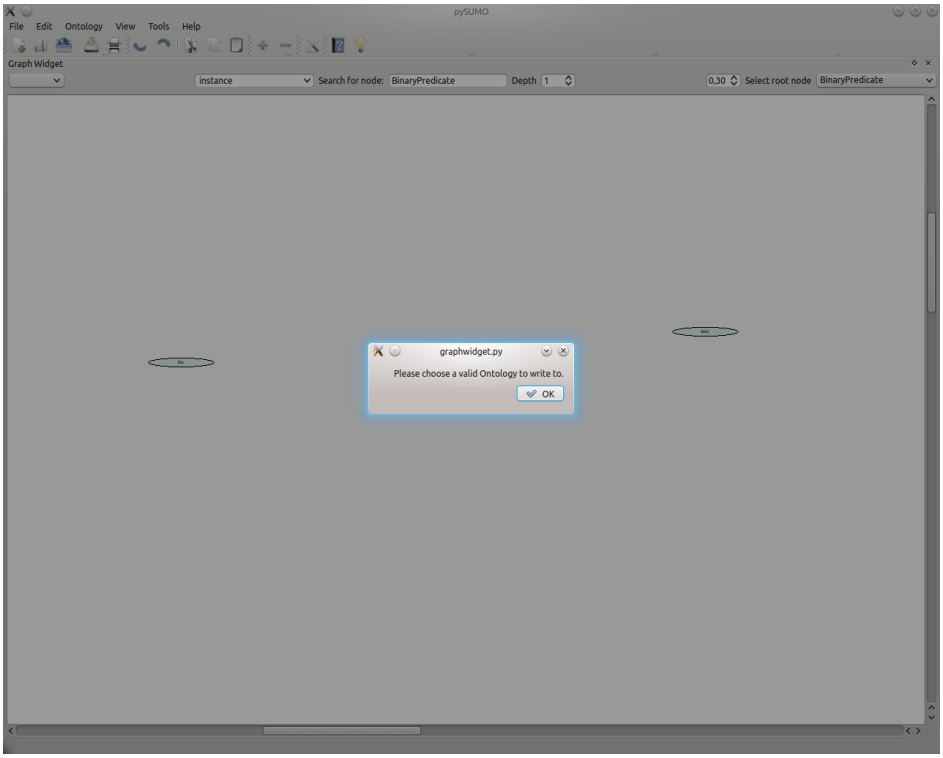


Fig. 5.5: Graphwidget after step 11.

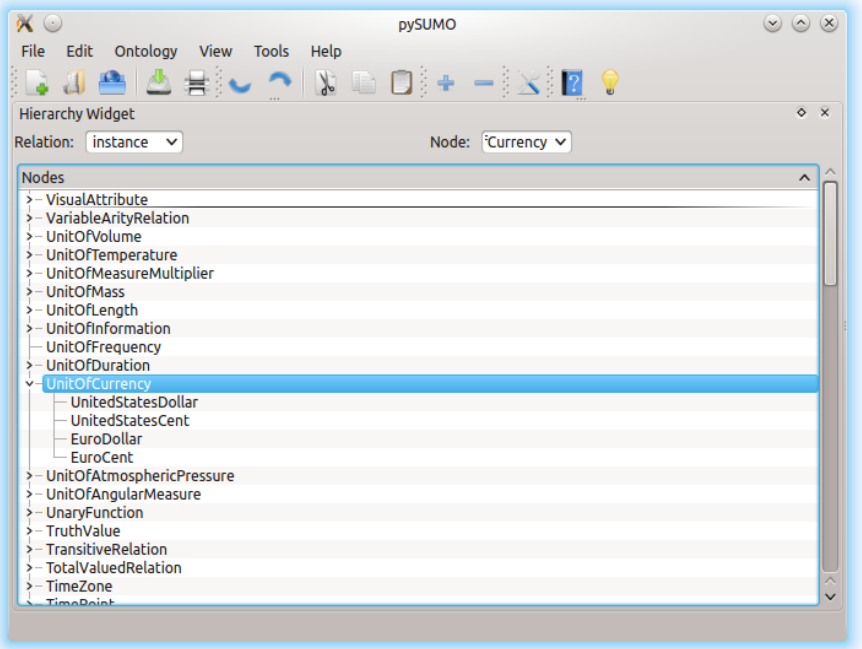


Fig. 5.6: Hierarchywidget after doing test.

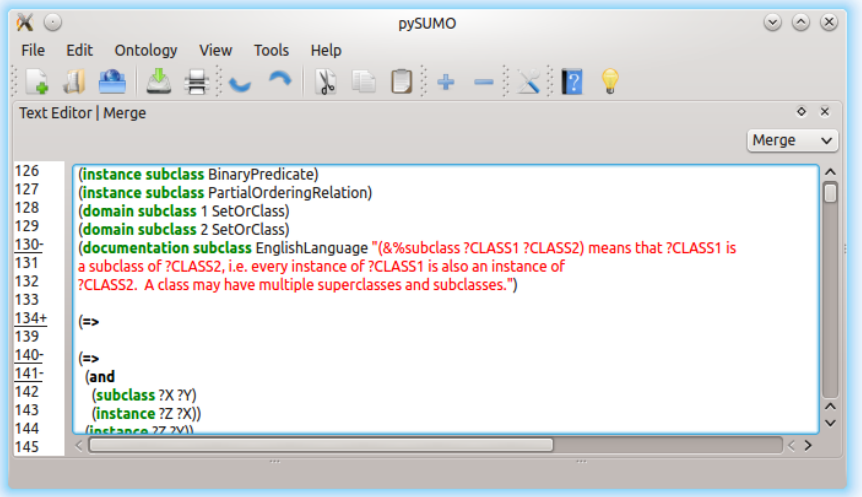


Fig. 5.7: Texteditor after doing test.

CHAPTER 6

Changes in Version 1.0

6.1 pysumo

- Rename lib to pysumo.

6.1.1 IndexAbstractor

- Delete DotGraph.
- Make root and depth keyword arguments defaulting to None
- Turn index into a simple dict. Makes index access easier and faster.
- Add init_wordnet() to initialize the WordNet mapping
- Add update_index() to build the index
- wordnet_locate() returns a list of strings
- IA now also stores the root AST node and a list of ontologies
- Change get_graph arguments to allow for more complex matching.
- Add normalize method for normalizing string arguments.
- Add function to return list of possible completions.

AbstractGraph

- Change init to reflect new get_graph API.
- Change variant to String from Enum and root to String from AbstractGraphNode
- Add info argument to __init__. Allows passing the Index/List of Ontologies

6.1.2 WordNet

- change argument of wparse() from a mapping file to a directory containing the mapping files and the rest of the wordnet database - avoid merging mapping files
- completely get rid of all dependencies on NLToolkit - extending upon it is a nightmare
- locate_term() returns a tuple of String, SSType, String

6.1.3 SyntaxController

- Add an optional argument newversion to add_ontology()
- Remove parse_graph(), the user now has to edit the file self
- Remove parse_add(), we can use add_ontology() now
- Moved Ontology to syntaxcontroller module

6.1.4 Logger

ActionLog

- Add name argument to init.
- undo/redo now return self.current.
- Make ActionLog work with objects that provide a buffer interface via the getbuffer() function.
- Add functions for pop and clear.

LogIO

- Add default_path variable.

InfoLog

- Add socket interface for StatusBar
- Add default_{log_path,socket_path} variables.
- init now receives a string detailing the loglevel instead of an int.

6.2 pySUMOQt

- Rename ui to pySUMOQt.

6.2.1 TextEditor

- remove show autocomplete – Qt does this on its own
- **Added**
  - setTextChanged()
  - _initNumberBar()
  - _updateOntologySelector()
  - _hideLines(lines)
  - _showLines(lines)
  - _zoomOut_()
  - _zoomIn_()

  - showOtherOntology(ontologyname)
  - expandIfBracketRemoved()
  - increaseSize()
  - decreaseSize()
  - expandAll()
  - hideAll()
  - getLayoutWidget()
  - numberbarPaint(number_bar, event)
  - initAutocomplete()
  - searchCompletion()
  - toggleVisibility(line)
  - hideFrom(line)
  - insertCompletion(completion)
  - commit()

Introduce SyntaxHighlightSetting to handle user defined Syntax Highlight Rules

class SyntaxHighlightSetting() __init__(expression, font_weight, font_color, expression_end='') createFormat() get_format() getValues() serialize() deserialize(string)

Introduce class SyntaxHighlighter class SyntaxHighlighter __init__(document) highlightBlock(text)

Introduce Numberbar because Qt does not do this on his own class NumberBar(QWidget) __init__(edit) paintEvent(event) adjustWidth(count) updateContents(rect, scroll) mouseDoubleClickEvent(event)

6.2.2 GraphWidget

Komplette Änderung der API, um an pygraphviz anzupassen

6.2.3 MainWindow

- **Added**
  - _showOptionDialog_()
  - _addWidget_(widgetType, widgetMenu)
  - createPySumoWidget(widgetType, widgetMenu)
  - addDeleteWidgetAction(widget)
  - addOrRestoreWidget(widget, menu, directAdd=False)
  - closeEvent(event)
  - createStatusBar()
  - setupStatusConnection()
  - displayLog(socket)
  - _updateStatusbar_(wrappedWidget=None)

  - _deleteWidget_(widget)
  - connectWidget(widget)
  - disconnectWidget(widget, callback=None)
  - getDefaultOutputPath()
  - _newOntology_()
  - _openLocalOntology_()
  - _openRemoteOntology_()
  - addOntology(ontology, newversion=None)
  - notifyOntologyAdded(ontology)
  - _ClearRecentOntologiesHistory_()
  - _deleteOntology_(ontology)
  - _updateOntology_(ontology)
  - _revertOntology_(ontology)
  - _showOntologyProperties_(ontology)
  - _closeOntology(ontology)
- Added quit_handler(signum, frame) to capture SIGINT signal.
- Introduced class PySUMOWidget which wrappes the application widgets.
- Removed class Statusbar, it became useless towards createStatusBar in MainWindow.
- Removed class Menubar, because the menu bar is already created by the designer.
- Removed class Toolbar, because the tool bar is already created by the designer.
- Moved class HelpDialog to module Dialog

6.2.4 Settings

- Introduced class LayoutManager.
- Introduced class PySumoSettings.
- Removed class PluginManager.
- Removed class WSettings.
- Moved class OptionDialog to module Dialog.

6.2.5 OptionDialog

- Remove createView() and load(path) methods.
- Added other methods to the OptionDialog.

CHAPTER 7

Feature Request

7.1 Feature Requests

- Search in Text Editor
- Hyperlinks to TextEditor from GraphWidget, Hierachy, Documentation
- Better position for GraphNode
- Documentation Widget to Graph Widget Links
- GraphWidget zoom to fit
- Documentation strings links
- Remote ontology: choose name automatically
- GraphWidget keep root node after refresh (e.g. after adding a node)
- GraphWidget set to new father or children

CHAPTER 8

Tests

A list of your tests, to make sure pysumo is correct working. If you find a feature that is not tested by us, or you found a bug feel free to contact us.

8.1 Library

8.1.1 ActionLog

- Queue the addition of an ontology to the A0ctionLog * Initialize an ActionLog with an ontology * Modify an ontology and store the changes * Assert that undoing the last action returns the previous state * Assert that undo/redo is invariant * Make sure the queue auto-flushes when exceeding size * Check that addition queue is overwritten when new changes are added and the old changes have not been OK'd. * Assert the invariance of excessive undoes * Assert the invariance of excessive redoes
- Check that ActionLog copes with high workloads

8.1.2 IndexAbstractor

- Assert than normalize() normalizes terms correctly.
- Assert that the index is built correctly.
- Test search on various terms
- Test WordNet search and self-initialization.
- Build several AbstractGraphs and check their contents
- Test that get_ontology_file returns the correct kif
- Test that adding multiple ontologies works
- Test that get_completions returns a correct list of terms

8.1.3 Parser

wParse

- Check that Tokenizer works on single lines
- Check that whole WordNet parses successfully

kifParse

- Check invariance of kifparse/kifserialize
- Test that parsing Government.kif works

8.1.4 SyntaxController

- Assert that add_ontology() adds an AST equivalent to the one produced by kifparse()
- Assert the invariance of redundant adds
- Assert that adding and then deleting an ontology is invariant
- Check that adding two ontologies works correctly
- Assert that get_ontologies() lists the correct ontologies
- Assert that modifying the ontology works correctly
- Assert that parsing diffs works correctly
- Assert that undo/redo work correctl

8.1.5 WordNet

- Check functionality of locate_term
- Check functionality of find_synonym

8.2 GUI

8.2.1 Text Editor

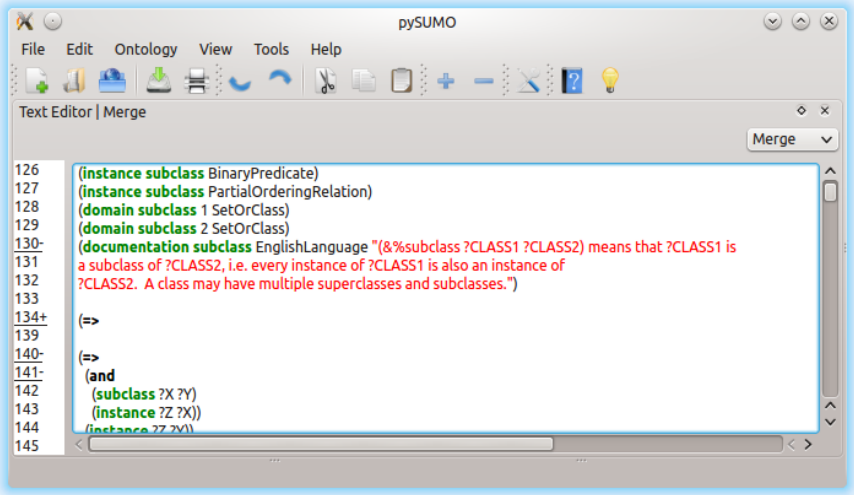


Fig. 8.1: Texteditor after doing test.

1. Open pySUMO
2. Open TextEditor
3. Open Merge.kif
4. Choose Merge.kif
5. Collapse Line 288
6. Collapse Line 136
7. Collapse Line 134
8. Uncollapse Line 134
9. Uncollapse Line 288
10. Collapse all
11. Expand all

8.2.2 Graphical Settings

1. Open pySUMO
2. Open TextEditor
3. Open Merge.kif
4. Choose Merge.kif
5. Open GraphWidget
6. Open DocumentationWidget
7. Open Hierarchy Widget
8. Open TextEdiorWidget

8.2.3 Hierarchy Widget

8.2.4 Graph Widget

1. Open pySUMO
2. Open GraphWidget
3. Open Merge.kif
4. Select instance on Variant selector
5. Select a root node
6. Select a depth (1)
7. Open a new ontology to write temporary content to.
8. Add a node “bla” in Graph Widget
9. Add a node “bla2”
10. Add a node “bla” (error)
11. Add a relation instance between “bla” and “bla2” (error)

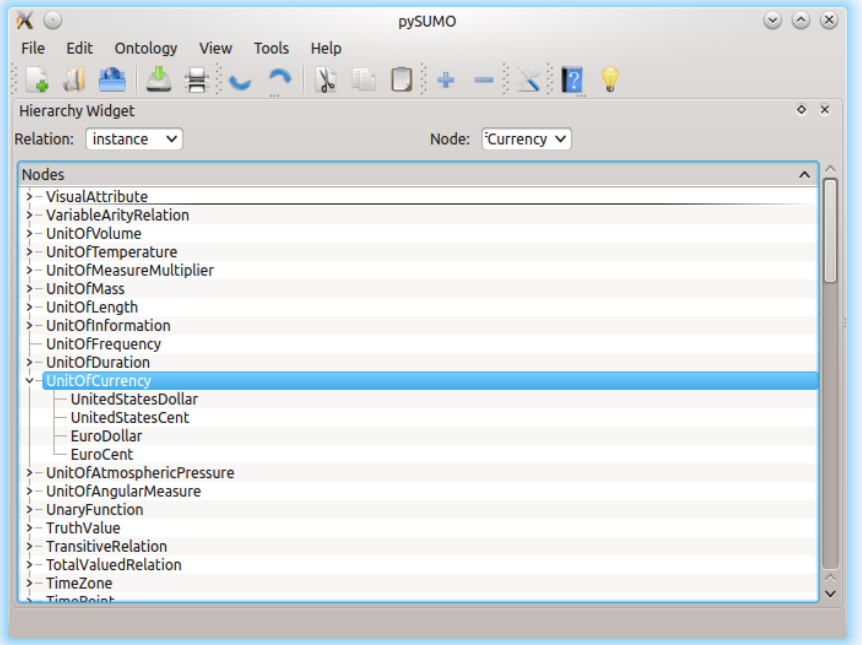


Fig. 8.2: Hierarchywidget after doing test.

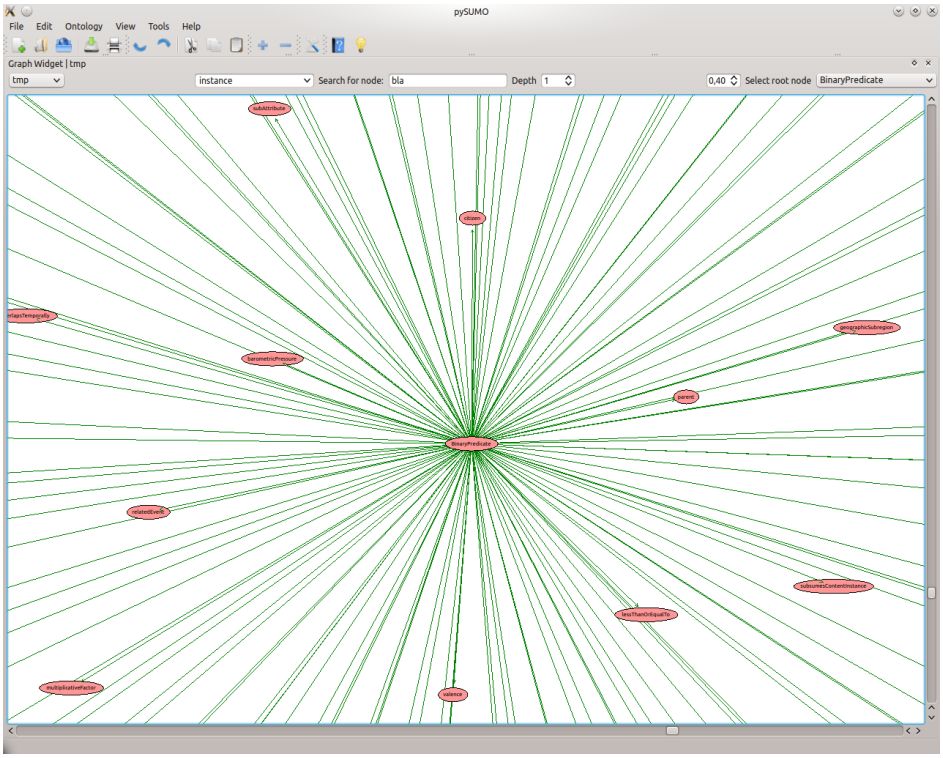


Fig. 8.3: Graphwidget after step 6.

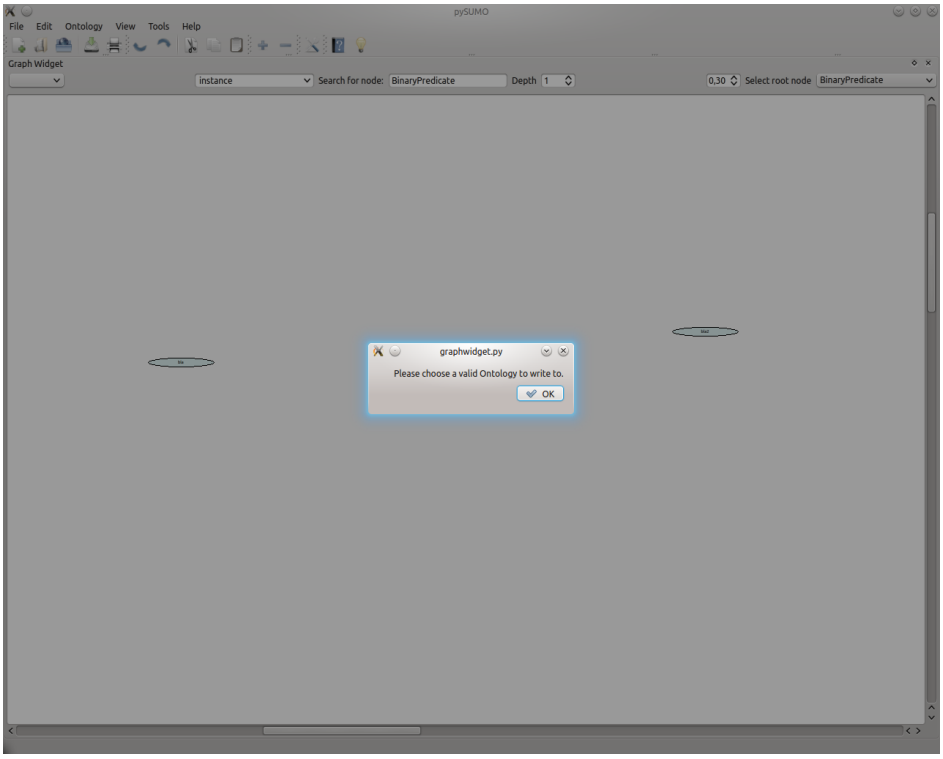


Fig. 8.4: Graphwidget after step 11.

12. Change scale
13. Choose a valid selector (there was a messagebox)
14. Add a relation instance between “bla” and “bla2”
15. Undo
16. Redo

8.2.5 Documentation Widget

1. Open pySUMO
2. Open Merge.kif
3. Open DocumentationWidget
4. Switch to the Ontology tab in the DocumentationWidget
5. Type subrelation into the search field
6. Press Enter
7. Open TextEditor
8. Select Merge.kif in TextEditor
9. Press one of the links listed under “Merge”
10. Switch to the WordNet tab in the DocumentationWidget
11. Search for ‘Object’

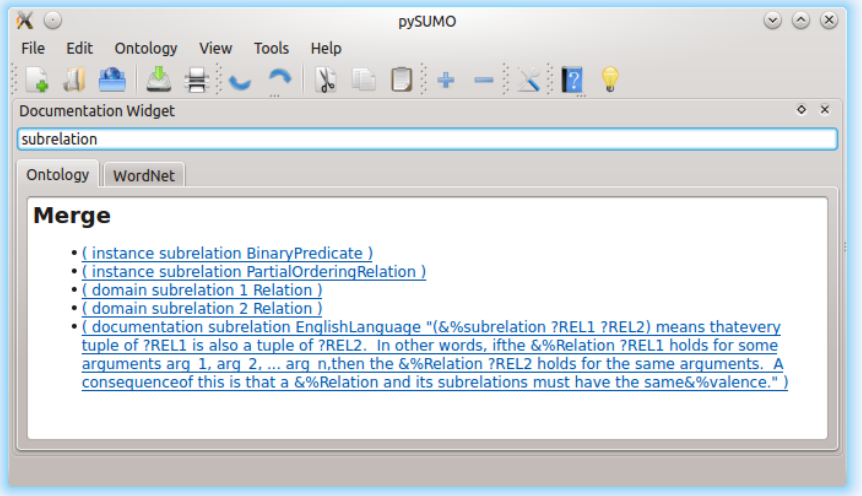


Fig. 8.5: Documentation widget after step 6.

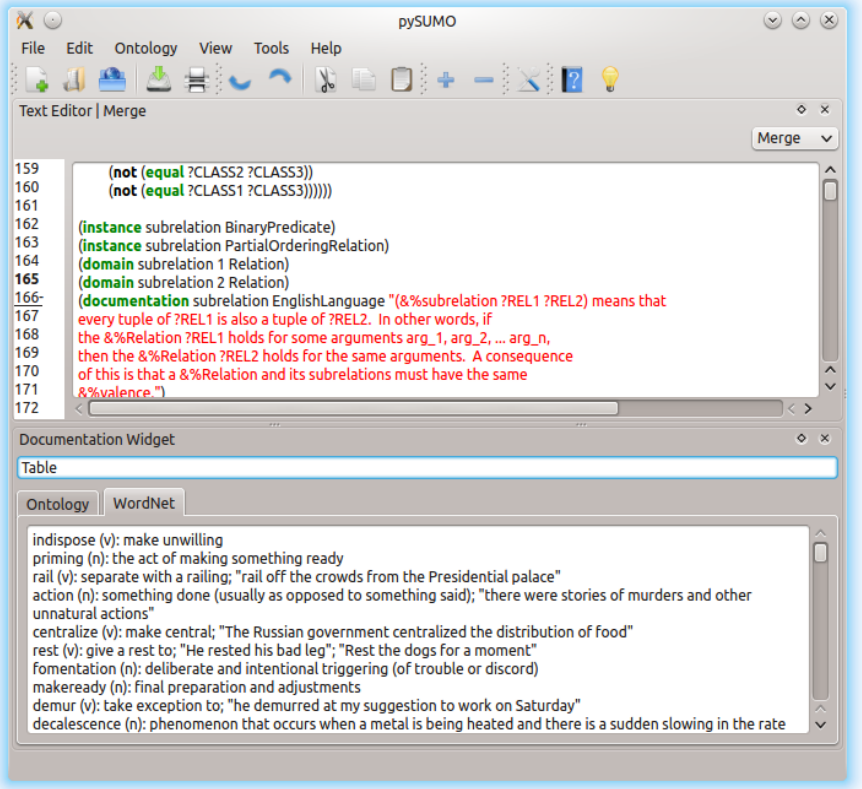


Fig. 8.6: Docwidget after doing test.

12. Search for ‘Table’

8.2.6 MainWindow

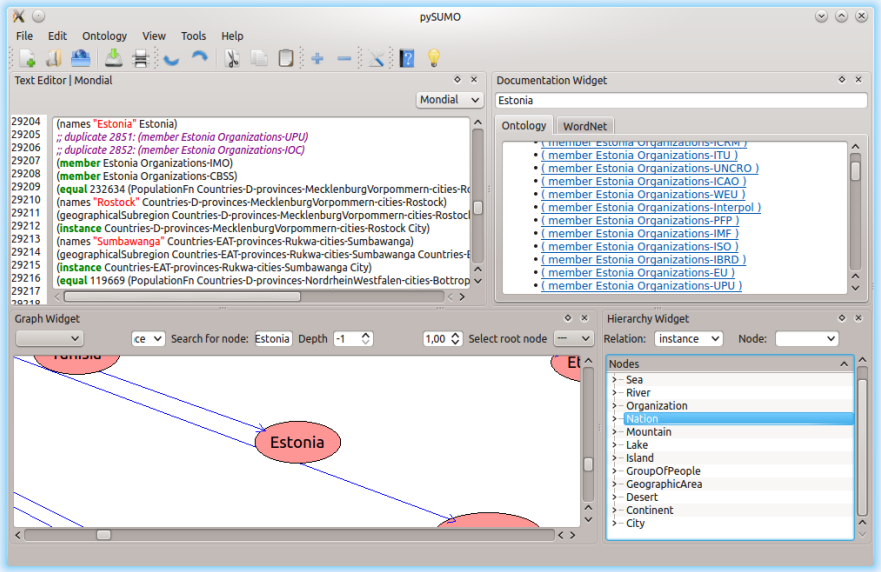


Fig. 8.7: Widgets ordering.

1. Open pySUMO
2. Open a new Ontology named “Mondial”
3. Open a remote ontology named “Mondial” at location: http://sigmakee.cvs.sourceforge.net/viewvc/sigmakee/KBs/mondial.kif
4. Close the ontology named “Mondial” with save and reopen it as local file.
5. Add Text Editor Widget
6. Add Documentation Widget
7. Add Graph Widget
8. Add Hierarchy Widget
9. Reorder Widgets
10. Make a print preview of the ontology “Mondial” from the Text Editor Widget
11. Delete the Text Editor Widget
12. Make a print preview of the ontology “Mondial” from the Graph Widget
13. Add a node and a relation in Graph widget
14. Delete the Graph Widget
15. Try to update the ontology “Mondial”
16. Open ontology properties dialog for “Mondial” and then close it.
17. Open Option Dialog and then close it

18. Revert the ontology “Mondial”
19. Quit pySUMO

CHAPTER 9

Test Results

9.1 Statement Coverage of pysumo and pySUMOQt

Name	Stmts	Miss	Cover
src/pySUMOQt/Designer/DocumentationWidget	48	0	100%
src/pySUMOQt/Designer/GraphWidget	82	0	100%
src/pySUMOQt/Designer/HierarchyWidget	68	0	100%
src/pySUMOQt/Designer/MainWindow	486	0	100%
src/pySUMOQt/Designer/NewOntologyDialog	50	0	100%
src/pySUMOQt/Designer/OntologyPropertyDialog	85	0	100%
src/pySUMOQt/Designer/OpenRemoteOntologyDialog	53	0	100%
src/pySUMOQt/Designer/OptionDialog	689	0	100%
src/pySUMOQt/Designer/TextEditor	30	0	100%
src/pySUMOQt/Designer/__init__	0	0	100%
src/pySUMOQt/Designer/css_rc	9	1	89%
src/pySUMOQt/Designer/gfx_rc	9	1	89%
src/pySUMOQt/Dialog	264	59	78%
src/pySUMOQt/MainWindow	454	48	89%
src/pySUMOQt/Settings	202	8	96%
src/pySUMOQt/Widget/DocumentationWidget	64	10	84%
src/pySUMOQt/Widget/GraphWidget	255	15	94%
src/pySUMOQt/Widget/HierarchyWidget	103	5	95%
src/pySUMOQt/Widget/TextEditor	476	85	82%
src/pySUMOQt/Widget/Widget	68	18	74%
src/pySUMOQt/Widget/__init__	0	0	100%
src/pySUMOQt/__init__	0	0	100%
src/pysumo/__init__	2	0	100%
src/pysumo/indexabstractor	147	9	94%
src/pysumo/logger/__init__	1	0	100%
src/pysumo/logger/actionlog	181	4	98%
src/pysumo/logger/infolog	33	7	79%
src/pysumo/parser	219	4	98%
src/pysumo/syntaxcontroller	118	4	97%
src/pysumo/updater	12	1	92%
src/pysumo/wordnet	19	0	100%
TOTAL	4227	279	93%

For a more detailled view on our testing please visit our github Page

CHAPTER 10

Errors

Errors	Cause[s]
HTTPError	An HTTP connection failed
IOError	An IO operation failed
KeyError	The specified key does not exist in the dict
NoSuchOntologyError	The specified Ontology does not exist
NoSuchTermError	The specified Term does not exist
ParseError	The specified Ontology is not valid

CHAPTER 11

External Libraries

PySide	Python bindings for Qt
PyGraphviz	Python bindings for Graphviz

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

p

pysumo, 10
pysumo.logger, 11
pysumo.logger.actionlog, 11
pysumo.logger.infolog, 13
pySUMOQt, 17
pySUMOQt.Widget, 17

A

ActionLog (class in pysumo.logger.actionlog), 11

C

clear() (pysumo.logger.actionlog.LogIO method), 13

D

default_log_path (pysumo.logger.infolog.InfoLog attribute), 14
default_path (pysumo.logger.actionlog.LogIO attribute), 13
default_socket_path (pysumo.logger.infolog.InfoLog attribute), 14
diff() (pysumo.logger.actionlog.LogIO method), 13

F

flush_write_queues() (pysumo.logger.actionlog.LogIO method), 13

I

InfoLog (class in pysumo.logger.infolog), 13

L

LogIO (class in pysumo.logger.actionlog), 12

M

max_diffs (pysumo.logger.actionlog.LogIO attribute), 13
max_size (pysumo.logger.actionlog.LogIO attribute), 13

O

ok_log_item() (pysumo.logger.actionlog.ActionLog method), 12

P

pop() (pysumo.logger.actionlog.LogIO method), 13
pysumo (module), 10
pysumo.logger (module), 11
pysumo.logger.actionlog (module), 11
pysumo.logger.infolog (module), 13
pySUMOQt (module), 17
pySUMOQt.Widget (module), 17

Q

queue_log() (pysumo.logger.actionlog.ActionLog method), 12

R

read() (pysumo.logger.actionlog.LogIO method), 13
redo() (pysumo.logger.actionlog.ActionLog method), 12
redo() (pysumo.logger.actionlog.LogIO method), 13

S

set_loglevel() (pysumo.logger.infolog.InfoLog method), 14

T

timeout (pysumo.logger.actionlog.LogIO attribute), 13

U

undo() (pysumo.logger.actionlog.ActionLog method), 12
undo() (pysumo.logger.actionlog.LogIO method), 13