# Documentation on PyNSource models.

There are a couple of ways of getting to the PyNSource internals re getting access to the names of the classes etc. that have been imported. There is also the technique of examining the persisted output.

All formats are ‘custom’ and based on arrays, dictionaries and tuples. The ocassional class/object is also used.

## Using the persistence format

{'type':'node', 'id':'CmdBase', 'x':557, 'y':270, 'width':87, 'height':31, 'attrs':'', 'meths':''}

{'type':'node', 'id':'CmdDeselectAllShapes', 'x':650, 'y':14, 'width':230, 'height':83, 'attrs':'', 'meths':'execute|undo'}

{'type':'edge', 'id':'CmdFileSaveWorkspace\_to\_CmdBase', 'source':'CmdFileSaveWorkspace', 'target':'CmdBase', 'uml\_edge\_type':'generalisation'}

{'type':'edge', 'id':'CmdFileLoadWorkspaceFromQuickPrompt\_to\_CmdFileLoadWorkspaceBase', 'source':'CmdFileLoadWorkspaceFromQuickPrompt', 'target':'CmdFileLoadWorkspaceBase', 'uml\_edge\_type':'generalisation'}

### 2019 update, the 1.2 format

Added the comment type e.g.

# PynSource Version 1.2

{'type':'meta', 'info1':'Lorem ipsum dolor sit amet, consectetur adipiscing elit is latin. Comments are saved.'}

{'type':'umlshape', 'id':'MyPrintout', 'x':21, 'y':164, 'width':136, 'height':174, 'attrs': 'canvas|log', 'meths': '\_\_init\_\_|OnBeginDocument|OnPreparePrinting|HasPage|GetPageInfo|OnPrintPage|IncreasePrintAreaSize'}

{'type':'umlshape', 'id':'wx', 'x':74, 'y':30, 'width':27, 'height':24, 'attrs': '', 'meths': ''}

{**'type':'comment'**, 'id':'C8004', 'x':203, 'y':42, 'width':200, 'height':100, 'comment': 'aW5pdGlhbCBjb21tZW50CnRoaXMgaXMgYSBjb21tZW50'}

{'type':'edge', 'id':'MyPrintout\_to\_wx', 'source':'MyPrintout', 'target':'wx', 'uml\_edge\_type': 'generalisation'}

## Accessing the proper models in memory

Unfortunately there are three ‘models’ in use.

1. ParseModel – this is the parse model, after the raw parsing.
2. DisplayModel ~~UmlWorkspace~~/~~UmlModel~~ - for displaying the parse model on the canvas, a mapping of abstract nodes to wxpython OGL shapes
3. UmlGraph – a graph of nodes and edges for layout purposes, self.graph

The parse model "classes" are represented by Graph "nodes" where the node.id is the name of the class, and the node x,y,width,height is abstract positioning for layout purposes. Subclasses of GraphNode have attributes and methods for properly representing the parse model "classes" which have been detected from parsing the source code.

### ParseModel

In class CmdFileImportBase inside filemgmt.py the parsing process begins with a call to new\_parser(f) which returns the parsemodel. The parsemodel structure is documented in dump\_pmodel.py thus:

class AbstractParseModel(object):

def \_\_init\_\_(self):

self.**classlist** = {}

self.**modulemethods** = []

self.**errors** = "" # new as of 2019

where

.classlist {classname:classentry, ...} where classname is a string, classentry is a class containing

.ismodulenotrealclass T/F

.classdependencytuples [(fromclass, toclass), ...]

.classesinheritsfrom [class, ...] # todo should be renamed classinheritsfrom (singular)

.attrs [attrobj, ...]

.attrname

.attrtype [] # todo should be renamed attrtypes plural

.compositedependencies # todo (calculated in real time, should precalc)

.defs [method, ...]

.modulemethods = [method, ...]]

There is a useful

dump\_old\_structure(pmodel)

function in there that you can use to dump the parse model. There is nothing in the gui that can dump the parse model out, but that is easily addresses. For example you could modify filemgmt.py to dump the parse model after a File/Import is done e.g.

class CmdFileImportBase(CmdBase): # BASE

def execute(self):

assert self.files

# these are tuples between class names.

self.context.model.ClearAssociations() # WHY DO WE WANT TO DESTROY THIS VALUABLE INFO?

if self.files:

for f in self.files:

#pmodel, debuginfo = old\_parser(f)

pmodel, debuginfo = new\_parser(f)

from parsing.dump\_pmodel import dump\_old\_structure # ADD THIS

print dump\_old\_structure(pmodel) # ADD THIS

self.context.model.ConvertParseModelToUmlModel(pmodel)

**Tip:** You could thus potentially use **dump\_old\_structure()** as a basis for a different loop that generates e.g. UMI.

### DisplayModel ~~UmlWorkspace~~

The DisplayModel ~~UmlWorkspace Model~~ reads the parse model and creates a different model of Graph **nodes/edges** mapped to wxpython OGL **shapes** on the umlcanvas. It stores these mappings thus:

* self.classnametoshape = {} # dict of classname => shape entries
* self.associations\_generalisation = [] # list of (classname, parentclassname) tuples
* self.associations\_composition = [] # list of (rhs classname, lhs classname) tuples

and also offer access to the 3rd model, self.graph, which contains all the Graph **nodes/edges** which are used for layout purposes.

Each graph node corresponds to a shape in terms of x, y, width, height – but nothing else, because that's all you need for working out layout and overlap. More specific nodes are subclassed from GraphNode e.g. UmlNode and CommentNode. All GraphNodes live inside a Graph. Our specialised UmlGraph inherites from Graph and has extra methods for persistence related stuff – see the diagram comment nodes - UML class-sequence-combo diagram 2019.pdf

### UmlGraph Model

* self.graph = UmlGraph() # graph is full of GraphNode objects, and associations, which are ultimately what we layout.

containing

self.graph.nodes

self.graph.edges

## Getting to the UmlWorkspace Model

This can be dumped at any time in the gui. Just run rungui.bat to ensure you get the console window and hit ‘d’ in the GUI when there is something in the workspace.

If you look at class CmdDumpUmlWorkspace(CmdBase): in diagnostics.py you will see this is called by pressing 'd' whilst the gui is running. It dumps the model using self.context.model.Dump()

Sometimes I see nothing dumped out in generalisation relationship diagnostic dump – not sure why that is – needs to be investigated.

**Tip:** You could potentially change the this dump routine to generate UMI or other formats, as a prototyping technique. Just print to console and hit ‘d’ to see the result – easy development! Later the algorithm can be properly integrated into PyNSource.

### Appendix

Thoughts on the DisplayModel ~~UmlWorkspace Model~~ organisation

- arguably the dict of classname => shape entries is a rendering related mapping which shouldn't be here?

- I think we need the association mappings because the graph object we use only has simple associations not the level of detail we need. Perhaps re-examing this and rely only on a beefed up graph class instead - in order to simplify things?

See [issue 12 at google code](http://code.google.com/p/pynsource/issues/detail?can=1&id=12) containing doco.