

# dottoxml

## Table of contents

1 Current version.....	2
2 Contributions.....	2
3 Basic usage.....	2
4 Color options.....	7
5 Other specials.....	8
6 Restrictions.....	9
7 More examples.....	9

While trying to get a deeper understanding for the source of the build system [SCons](#), I actually wanted to see the classes and their dependencies. Googling around I found these two tools that could produce DOT graph files for class or import dependencies from Python source trees.

#### [snakefood](#)

Creates module import dependency graphs. Use the Mercurial command "hg clone <https://hg.furius.ca/public/snakefood>" for a checkout of the current sources.

#### [pyreverse](#)

As part of pylint it analyzes class dependencies.

Unfortunately, the resulting graphs were huge and visualizing them via the `dot` tool did not help. PNG, PS or SVG output, the images got too large and the layout of the nodes and edges left a lot to desire (Example: SCons dependencies in [Postscript](#) and [SVG](#) format).

Then I remembered the [yEd graph editor](#), a great application that can layout and handle even very large datasets...if you find a way to get the data inside. Since it doesn't import DOT files (yet), I wrote this little converter script that outputs yEd's native file format Graphml (XML). Now looking at complicated DOT graphs is a snap...have a try!

## 1 Current version

The sources for this tool are now available at Bitbucket. Please visit <https://bitbucket.org/dirkbaechle/dottoxml> for downloading the latest version.

## 2 Contributions

### 2010-11-13

Daniel Reynaud: error report and patch (copying of attributes for default nodes and edges did not work, due to a copy-paste error).

### 2010-01-27

Martin Bijl-Schwab: error report and patch (`self.sections` did not get initialized in all cases).

## 3 Basic usage

### Note:

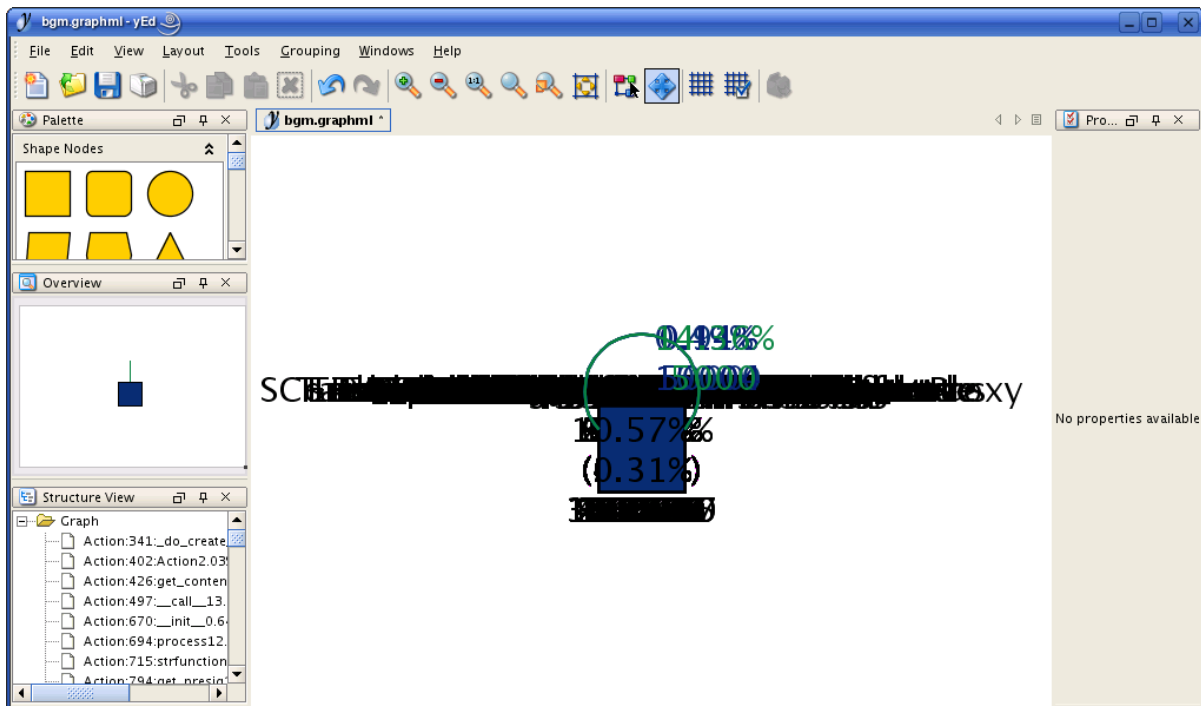
The following screenshots are not from a dependency analysis but a profiling of [SCons](#) at runtime. Using a combination of [Gprof2Dot](#) and [Graphviz DOT](#), Nitro Zark has published the results of his investigations on [his webpage](#). I picked the file `benchgen-full-dry-run.dot` because it uses colors to a great extent, which gives a nice looking graph.

Start the script with the "-h" or "--help" option and the full set of available commands is displayed.

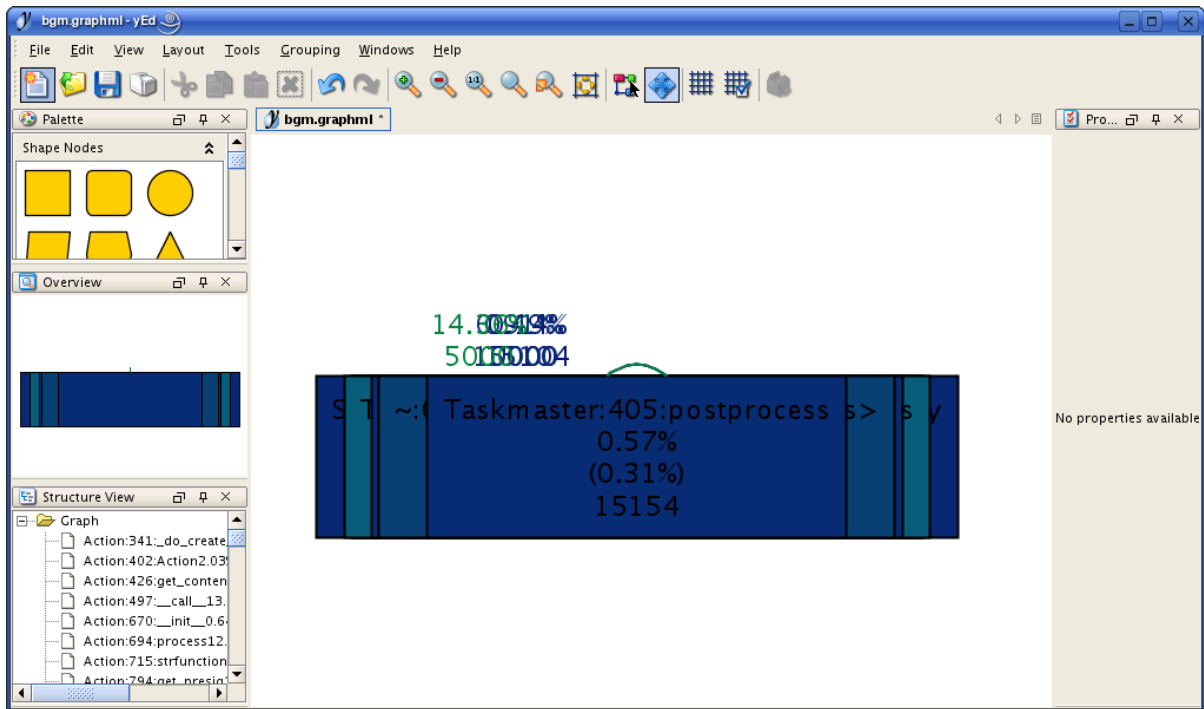
The straightforward way to create a Graphml file out of a DOT is:

```
python dottoxml.py infile.dot outfile.graphml
```

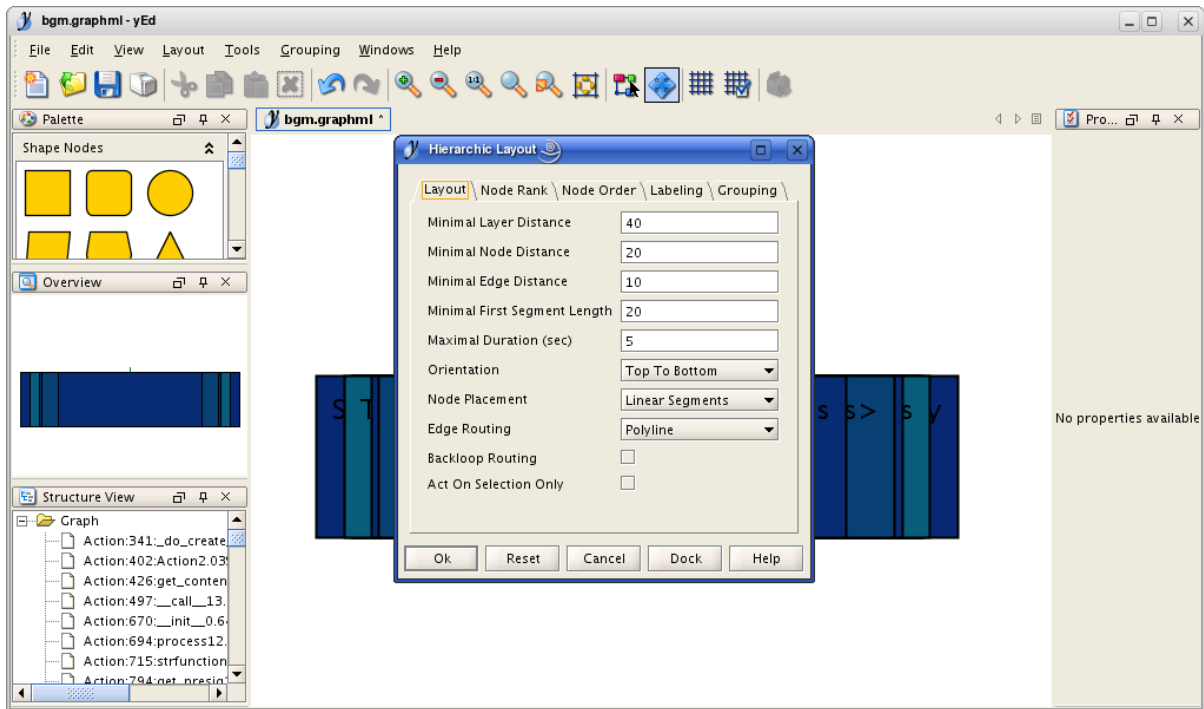
Then open the new Graphml file in the [yEd editor](#). The nodes of the graph are now all centered to the origin and have a standard size of 30x30.



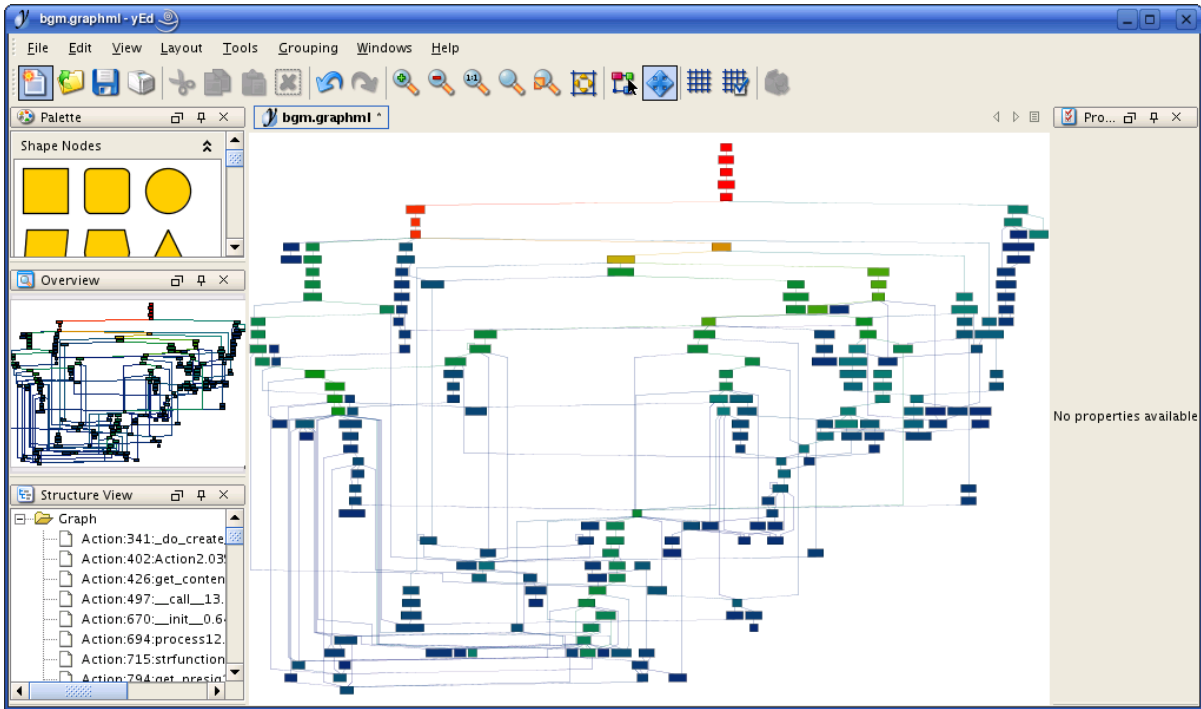
Change the latter by using the "Tools" menu in yEd and select the entry "Fit Nodes to Label". This feature adapts the size of each node to the text that is displayed within.



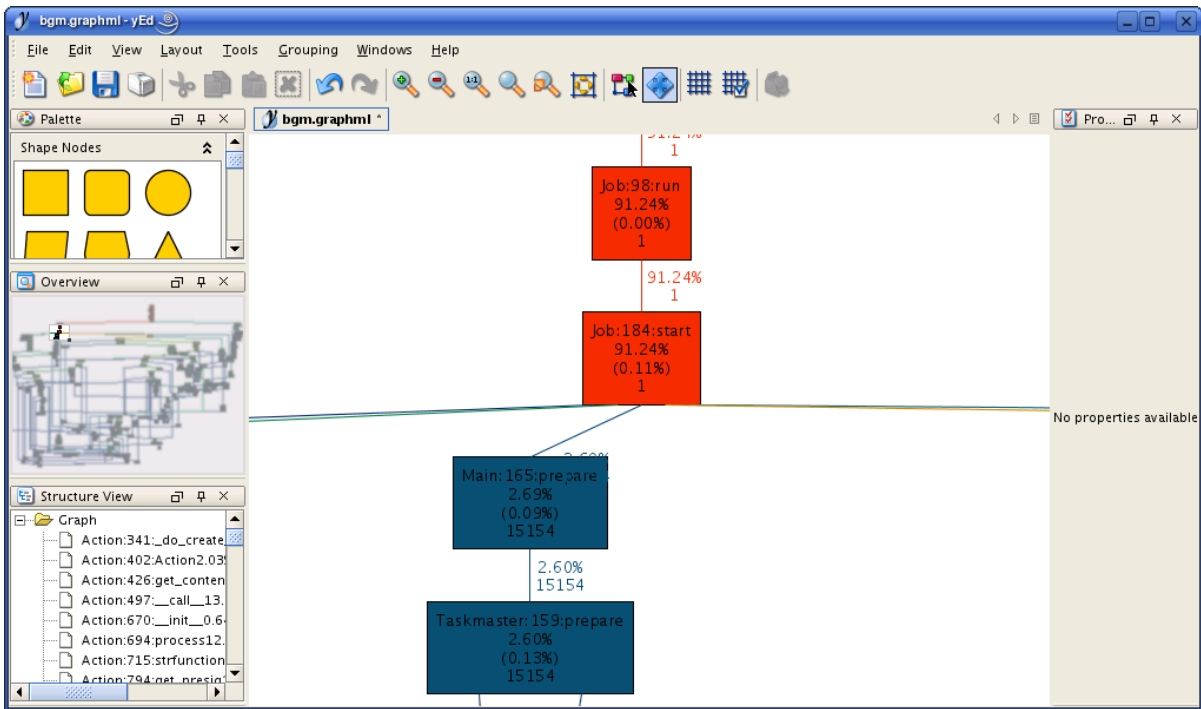
Now you can select one of the automatic layout strategies from the "Layout" menu. Often, one or two dialogs with special options appear.



Just go with the default settings and click OK. The nodes in the graph are then rearranged, according to your selected strategy. Here we see the layout "Hierarchical, classic":



And another time, in a randomly chosen closeup:

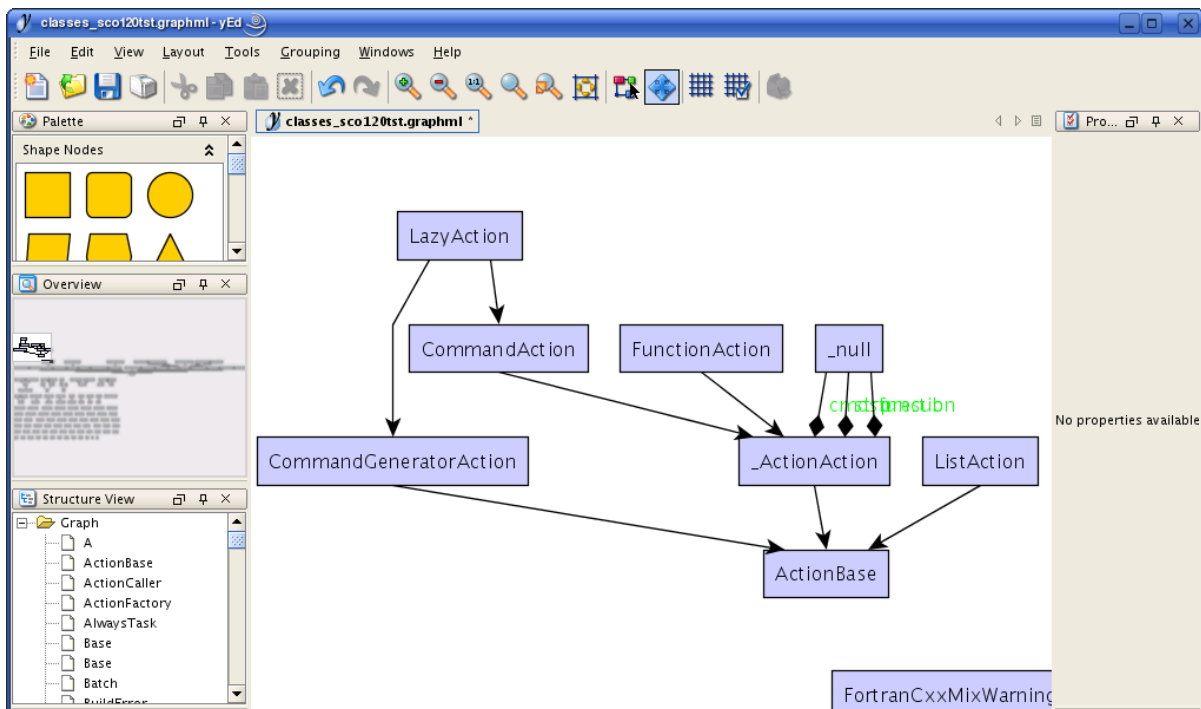


For further advice about the display or editing of graphs in [yEd](#), refer to its manual please.

## 4 Color options

In the screenshots above, the nodes are colored because the DOT file contained the necessary attribute statements. The `dottoxml` script tries to pick up as much information as possible from the input file, not only colors but also arrow shapes ("arrow" vs. "diamond") for example.

If no color information is present in the DOT file, `dottoxml` falls back to its defaults which are: some sort of grey (#CCCCFF) for the node background, and black (#000000) for the outline, the labels and the arrows.



You can override the default colors with the four commandline options:

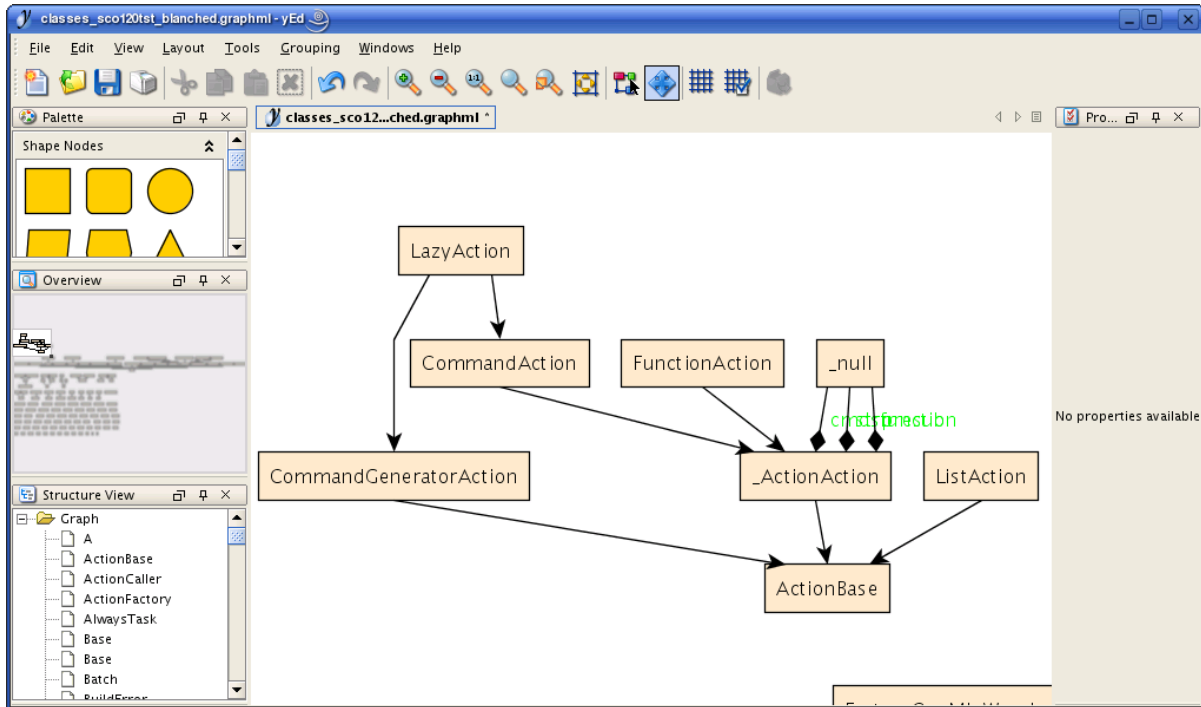
- cn**  
Node background color
- ce**  
Edge color
- cnt**  
Node label color
- cet**  
Edge label color

An example:

```
python dottoxml.py --cn #FF0000 infile.dot outfile.graphml
```

sets the standard node background to a pure "red". Instead of giving RGB triplets, you can also specify an X11 color name like this:

```
python dottoxml.py --cn blanchedalmond infile.dot outfile.graphml
```



## 5 Other specials

Very simple graphs often contain only the node labels and the edge information itself, but no labels for the edges. With the option "--ae" you can enable the "Auto labeling". This means that for every edge that doesn't provide its own label, dottoxml generates one of the form

```
source_node_label -> destination_node_label
```

When analyzing dependencies, there may appear single nodes that are "isolated" from the rest of the graph and have no outgoing or incoming edges at all. If you want to see only "connected" nodes, you can enable the "sweep" option "-s". The script then filters out all single nodes and doesn't output them to the Graphml file.

Finally, a very special option for the work with UML nodes that also contain the names of attributes and methods for a class. If you activate the "Fit Label to Nodes" feature in

yEd, the nodes are expanded only around the class name (=label) but not the methods. As a workaround you can enable the "lumping attributes" option "--la", which collects all the text data for the UML node and puts it into the label. The single sections are divided by separators, built with the "separator char" given by the "--sc" option.

## 6 Restrictions

This script is still under development and far from stable! Please note the following remarks and restrictions:

1. At the moment, the DOT parser is very simple and line-based. It detects only node and edge lines, no subgraphs are handled. Node and edge specifications must be in a single line!
2. I tried to do my best, but the whole encoding part (Unicode support and detection of input encoding) still appears to be a little bit "shaky" to me. Do not expect too much here.

## 7 More examples

More examples and screenshots can be found at the [VisualizeDependencies](#) page in the [SCons Wiki...](#)