

This program is released under the terms of the GNU General Public License. Do what you want with it.

See the History.rtf file to see what this program is all about.

To run, launch it from the Workspace and have it read in a .tree file (format is in ascii, as described in History.rtf). The tree will be displayed graphically. Click on a button to activate a node. When a node gets sent an activation message, it prints it's contents to stdout, which means that you will see feedback in the WorkSpace Console.

And that's all there is. This program mainly serves as a programming example and a way to make available a generic Tree class. The Tree class is particularly useful for building parse trees, and subclasses could easily be made to create a recursive pretty-printer automatically. (I have used the Tree class in an optimizing compiler which deals with a subset of Pascal, and the Tree did pretty printing and three-address code generation automatically by just sending the appropriate message to the root node. It worked out quite nicely. Each subclass corresponded to a different type of programming construct<sup>1/4</sup>) Also of note is the way I use a List as a display List for the line objects. Any type of object that can render itself could be placed in that list, which is part of how my gamekit deals with multiple types of sprites... and in the file parsing is an example of how a List object can be used as a stack. (It could be used as a queue, too, if you think about it. Quite a nifty object IMHO.)

Well I hope that somebody finds this somewhat useful. It only took me an evening to do, so I don't expect it to be particularly useful.

If you have any questions, requests, whatever, feel free to contact me.

Don\_Yacktman@byu.edu

*This is **not** a polished program! Here are some interface things that I would want to clean up if I had the time:*

- Draw the lines differently. Rather than middle of one button to the middle of the next, make the lines go horizontally and connect to a vertical line between tree levels. This should be very easy to do; I leave it as an exercise to the reader.

- Make the window and ScrollViews limit their size so that you can't make them any bigger than the underlying TreeView is.
- Make the buttons not disappear when scrolling is done. This happens because `-drawSelf::` redraws the *entire* view instead of only the portions its asked to redraw. The View machinery doesn't know that I zapped those areas, so it doesn't redraw those buttons; only the buttons in the redraw area get re-rendered. This is why some of the buttons appear and some don't. If I had done a proper `drawSelf::` you wouldn't see this problem because the View's display machinery wouldn't be fooled by my violation it's basic assumptions. (See the docs on `-drawSelf::` and the `-display` methods of View for more on this.)
- Change drawing approach to only draw what's visible.
- Limit the length of the strings used in the button titles so that long titles don't look ugly. Another approach would be to vary the size of the buttons, but that's harder to do...
- Make a proper B&W icon. Amongst other little things¼