Here's why this thing exists.   You'll notice from the dates that it's taken me quite a while to get around to actually doing this; it ended up being quite simple to get it put together. (What took me so long is the fact that in the meantime I had to write a simple optimizing compiler for one of my classes, and only had three weeks to do it in.)   The tree object just happens to be an enhancement of the object I used/subclassed to build the parse tree in the aforementioned compiler, so I know that the Tree class can be useful, at any rate.

The other classes in here won't be particularly useful to most folks, but there are examples of (1) multiple .nib files, (2) simple use of multiple documents±notice that I don't track multiple instances except as I open them, so the Windows menu and the user's mouse clicks do all the work here±I didn't really have to do anything special to get it to work, in other words, since NeXTSTEP does this for me.   There's also examples of using an Open Panel and of getting the WorkSpace to open double-clicked documents.   All this stuff is easy to do anyway, but I suppose that this program could serve as an example for the beginner.

As you'll notice from below, the main thing to show is how to use PostScript to draw lines and how to display the tree.   For most folks, the Tree object±and not this part±will be useful, but it's all in there with everything else for you to look at.   You'll notice that I took the simplest and least efficient route to implement the drawing, thus it should make sense to a beginner.   For better performance, other methods (described in the Adobe Purple book primarily) could be used.   I figured that simply calling the PS() functions would be good enough here, but ideally a PS wrap or better would be used.   Also, I redraw the entire View   every time even though I should really check the arguments to -drawSelf:: so that only the necessary parts get redrawn.   Although that would improve performance, it wasn't worth bothering with for this program.

My notes about the program are included in the message below in *italics*.

```
From weiner@pts.mot.com Thu Mar 18 17:00 MST 1993
Received: from motgate.mot.com by maine.et.byu.edu; Thu, 18 Mar 93 17:00:13 -0700
Return-Path: <weiner@pts.mot.com>
Received: from pobox.mot.com ([129.188.137.100]) by motgate.mot.com with SMTP
(5.65c/IDA-1.4.4/MOT-2.13 for <yackd@maine.et.byu.edu>)
          id AA23402; Thu, 18 Mar 1993 17:57:16 -0600
Received: from pts.mot.com ([145.4.3.2]) by pobox.mot.com with SMTP (5.65c/IDA-1.4.4/MOT-2.12 for
<yackd@maine.et.byu.edu>)
          id AA13790; Thu, 18 Mar 1993 17:57:51 -0600
Received: from info. ([145.4.25.12]) by pts.mot.com (4.1/SMI-4.1)
      id AA18452; Thu, 18 Mar 93 18:53:25 EST
Received: by info. (NX5.67c/NX3.0S)
      id AA02300; Thu, 18 Mar 93 19:01:39 -0500
Date: Thu, 18 Mar 93 19:01:39 -0500
From: Bob Weiner <weiner@pts.mot.com>
Message-Id: <9303190001.AA02300@info.>
To: yackd@maine.et.byu.edu
In-Reply-To: Don Yacktman's message of Thu, 18 Mar 93 16:34:14 -0700
<9303182334.AA29304@maine.et.byu.edu>
Subject: Re: Anyone have a tree display class that can read a tree from a text file?
Status: R
```

> X-Delivered: at request of weiner on infocomm
> Date: Thu, 18 Mar 93 16:34:14 -0700
> From: yackd@maine.et.byu.edu (Don Yacktman)
>
> > I guess if would be easy to write if I knew how to draw lines between buttons
> > (nodes) to connect them but this will be my first NeXTSTEP program and I
> > don't know how to do anything with DPS yet.
>
> > Also, if it is so easy to do, it is surprising that no one else has an object
> > to do this.
>
> Good point...I think it would be easy though, so if you tell me what you
> want, I bet I could whip something up quick and then put it on the net
> as a GNU copylefted thing...

Great.   Here's what I would want.   The tree class could read in a tree from a file in textual form.   The first line is the name of the tree.
In textual form, each node in the tree is given by its label, which can be any text, including spaces on a single line.   The node's position in the tree is given by its preceding indentation, normally using 3 spaces of indentation per level (the amount can be figured out by examining the first indented label and counting its spaces.   An example:

```
Tree Name
Root
   Node 1
      Node 1.1
         Node 1.1.1
   Node 2 has this longer label.
```

I read in the first two lines with the assumption that there is no indentation.   On the third line, I use the number of spaces to determine the indent.   Do not use tabs; I didn't bother to write code which can deal with them.   Be sure that the file ends with ".tree" so that the app will load it.   If you make the file incorrectly, you will cause weird things to happen, since I didn't bother to do much error checking while parsing.   Obviously, for this program to be more "useful" that ought to be rectified.

After the tree is read in, it is displayed within a new view.   Each node is created as a selectable button with the appropriate label and lines connecting it to its parent and children.   (Let's assume only one button can be selected at a time.)

For simplicity's sake, all buttons are the same size.   If   the node's label is really long or really short, this might cause a problem.   The View would have to do tricky calculations in

order to do better, and I demmed this a "not worth bothering with" thing.

When a button is selected, a string consisting of the tree name and the node selected is sent to an output stream.   (stdout is fine as a default.), e.g.

Tree Name^^Node 1.1

could be output with ^^ indicating separation of the two arguments.

This is what I do.   If launched from Workspace, look for the text in the Console window, which is where stdout goes to.   It would probably be better to have a UI "Console" window for this output and allow the user to select a file to "save" to, but it wasn't worth the effort to this this here, easy though it is.

That's it.   Then my other program takes this standard output and displays text associated with the selected node.