

NAME

tcldot – graph manipulation in tcl

SYNOPSIS

```
#!/usr/local/bin/tclsh
package require Tcldot
```

USAGE

Requires the dynamic loading facilities of tcl7.6 or later.

INTRODUCTION

tcldot is a tcl dynamically loaded extension that incorporates the directed graph facilities of **dot(1)**, and the undirected graph facilities of **neato(1)**, into tcl and provides a set of commands to control those facilities. **tcldot** converts **dot** and **neato** from batch processing tools to an interpreted and, if needed, interactive set of graph manipulation facilities.

COMMANDS

tcldot initially adds only three commands to tcl, namely **dotnew**, **dotread**, and **dotstring**. These commands return a handle for the graph that has just been created and that handle can then be used as a command for further actions on the graph.

All other "commands" are of the form:

handle <**method**> *parameters*

Many of the methods return further handles of graphs, nodes of edges, which are themselves registered as commands.

The methods are described in detail below, but in summary:

Graph methods are:

addedge, addnode, addsubgraph, countedges, countnodes, layout, listattributes, listedgeat-
tributes, listnodeattributes, listedges, listnodes, listnodesrev, listsubgraphs, render, rendergd,
queryattributes, queryedgeattributes, querynodeattributes, queryattributevalues,
queryedgeattribute

uteva
es,

10

0 uteTj-4autSet

*tcl*dot(3tcl)

*tcl*dot(3tcl)

number of attribute name/value pairs for the graph. Certain special graph attributes and permitted values are described in **dot(1)**, but t

graphHandle **write** *fileHandle* *format* ?*dot*|*neato*|*circo*|*twopi*|*fdp*|*nop*?

Write a graph to the open file represented by *fileHandle* in a specific *format*. Possible *formats* are: "ps" "mif" "plain" "dot" "gif" "ismap" If the layout hasn't been already done, then it will be done as part of this operation using the same rules for selecting the layout engine as for the layout command.

graphHandle **rendergd** *gdHandle*

Generates a rendering of a graph to a new or existing gifImage structure (see **gdTcl(1)**). Returns the *gdHandle* of the image. If the layout hasn't been already done, then it will be done as part of this operation using the same rules for selecting the layout engine as for the layout command.

graphHandle **render** ?*canvas* ?*dot*|*neato*|*circo*|*twopi*|*fdp*|*nop*??

If no *canvas* argument is provided then **render** returns a string of commands which, when evaluated, will render the graph to a **Tk** canvas whose *canvasHandle* is available in variable **\$c**

If a *canvas* argument is provided then **render** produces a set of commands for *canvas* instead of **\$c**.

If the layout hasn't been already done, then it will be done as part of this operation using the same rules for selecting the layout engine as for the layout command.

```
#!/usr/local/bin/wish
package require Tcldot
set c [canvas .c]
pack $c
set g [dotnew digraph rankdir LR]
$g setnodeattribute style filled color white
[$g addnode Hello] addedge [$g addnode World!]
$g layout
if {[info exists debug]} {
    puts [$g render]      ;# see what render produces
}
eval [$g render]
```

Render generates a series of canvas commands for each graph element, for example a node typically consist of two items on the canvas, one for the shape and the other for the label. The canvas items are automatically *tagged* (See **canvas(n)**) by the commands generated by render. The tags take one of two forms: text items are tagged with 0<handle> and shapes and lines are rendered with 1<handle>.

The tagging can be used to recognize when a user wants to interact with a graph element using the mouse. See the script in *examples/disp* of the tcldot distribution for a demonstration of this facility.

BUGS

Still batch-oriented. It would be nice if the layout was maintained incrementally. (The intent is to address this limitation in graphviz_2_0.)

