Project 1: transfig

Version 1, 17 September 2004

1.1 General overview of projects

The three projects have a common thematic focus: tools for creating graphics. We seek to emulate certain existing tools rather than inventing our own. However, proposals for extensions and improvements are encouraged.

In the Unix/X11 environment, a number of tools have been developed for creating and editing drawings. These tools are freely available, and their source code, mostly in C, has a long history. Our goal is to reimplement some of the functionality of these tools in Java. The tools use various graphics description languages as file formats for persistent storage of graphics, so we describe those first.

PostScript is a page description language, used for final rendering of graphics, usually to raster devices such as printers and display screens. It consists of a graphics model, accessed through a set of graphics commands, and a full-fledged programming language, with procedural abstraction. Arbitrarily complex graphics can be computed within a PostScript program itself. This power comes at a price: it is not possible, in general, to edit PostScript code. Some drawing editors use limited, stylized PostScript as their file format, but xfig uses its own language, fig.

Our goal in the first project is to read in fig descriptions and translate them into PostScript. This task is accomplished by the program transfig, a companion to xfig. (Additionally, transfig can produce output in a vast array of other formats.)

Our goal in the second project will be to produce fig files corresponding to plots of mathematical functions as well as empirical data files, similar to the gnuplot utility.

Our goal in the final project will be to edit fig files, similar to xfig.

1.2 Stages

Project handed out on 17 September. High-level design due on 21 September (analyze the flow of data; plan components; if you like UML, draw diagrams). Java classes and interfaces specification due on 23 September (make a point of using interfaces extensively). Low-level design due on 28 September (describe in detail how each kind of fig object will be translated to PostScript, including hand-translated examples for each). Java implementation due on 7 October.

1.3 The task

Obtain the xfig and transfig source distribution from `http://www.xfig.org/`. Learn how to use xfig. Study the format of fig files by referring to the official description in file `xfig.3.2.4/Doc/FORMAT3.2` of the xfig source distributions, and by looking at sample fig files you will create using xfig. Understand how the various kinds of objects are encoded in the fig format. Write a Java application to translate fig to Encapsulated PostScript.

The behavior of transfig can be your guide. You cannot go wrong if your output is exactly the same

as transfig's. However, there is an infinite variety of correct translations into PostScript, and you may choose one that is different from transfig's. Ultimately, the correctness is judged by examining if the rendered image from your PostScript output is the same as the rendered image from transfig's PostScript output.

The implementation of transfig in C can also be a useful reference. Note that transfig delegates all the work to fig2dev.

1.3.1 Structure

Keep in mind that in a subsequent project you will be implementing xfig. Therefore, think about organizing your Java classes so that you can reuse them. (You will want your xfig to be able to invoke your transfig.) Design your data structures for internal representation of drawing objects carefully. These should be shared between your xfig and your transfig, so that when xfig invokes transfig it can do so directly, with a Java call, rather than through a file.

In another subsequent project you will be producing fig; it makes sense to export an interface that allows another package (in your case it will be your gnuplot package) to construct a fig internal representation using Java calls and not just through a file .

1.3.2 Optional features

These features are highly desirable, but are not necessary for full credit on the assignment.

- support for curves [(3.6) SPLINE]; I expect that students who have studied splines in their math courses will implement this feature. (Take a look at `trans_spline.c` in the fig2dev implementation.)
- pattern fill

1.3.3 Packaging

Your code must be placed in a package named
`edu.unm.cs.cs351.fall2004.`*yourusername*`.fig.transfig`.
The main class of your program must be called `Transfig`.

The program accepts one command-line argument, the name of the fig file to be translated. Thus, it is invoked as:
yourfavoritejvm `edu.unm.cs.cs351.fall2004.`*yourusername*`.fig.transfig.Transfig` *figfilename*.

The output is placed in *epsfilename*, which is obtained by stripping the trailing `.fig` from *figfilename* and replacing it with `.eps`.

How to turn in

For the intermediate stages, bring your designs in hard copy to class on the designated date.

The following is the same as for Homework 1 except for the additional LOG file. That file must contain dated entries for all major design considerations and decisions you make during the project.

For the final submission, turn in your code by running

~jbrown/handin your-file

on a regular UNM CS machine.

You should use whatever filename is appropriate in place of your-file. You can put multiple files on the command line, or even directories. Directories will have their entire contents handed in, so please be sure to clean out any cruft.

Normally, you will turn in a top-level directory containing at least the following:

- a `README` file (a brief description of the structure and purpose of the code, with basic instructions for its use);
- a `COPYING` file (a copyright notice or something functionally equivalent);
- a `LOG` file containing your observations and design and coding decisions;
- a `TODO` file (a list of known bugs and deficiencies with respect to the assignment, or identified but unimplemented extensions);
- a `src` directory containing the Java source hierarchy for your code;
- a `classes` directory containing the compiled Java class files hierarchy, exactly corresponding to the `src` files;
- a `doc` directory containing various detailed project documentation as needed (including a detailed description of code structure (classes, interfaces, and how they are related) in a file `structure.txt` or, preferably, `structure.ps`, a description of the design process that led to this code structure in a file `design.txt` or, preferably, `design.ps`, and a subdirectory `generated/javadoc` with Javadoc-generated HTML files, exactly corresponding to the `src` files);
- a `tests` directory with input files used for testing the correctness of the program, including both valid and invalid inputs;
- a `test-results` directory with output files generated by the program on test inputs; and
- a `RESULTS` file (a summary of the testing process, documenting the program's compliance with the specification).