Technical gabble

126476_PixelRule.tiff ¬

This section contains more technical details. You can ignore everything that follows, and remain a happy Convert PICT user! If you continue, note that I'm assuming you know at least a little bit about PICT and PS file details (Inside Mac V p. 96-105 is a good place to look for more info about PICT).

The quick overview of how this application works is as follows: When given a PICT file, Convert PICT will start reading in opcodes and their parameters one at a time. In general this data undergoes essentially no alteration, but is simply written out in an equivalent form in the PS file. Thus:

```
300010001000FF00FF
```

becomes

```
16 16 255 255 frameRect
```

While it has undergone a bit of improvement in legibility, you'll see that the fundamental information remains unchanged. Of course, this isn't universally true. The four (or eight if you count the PICT II versions as distinct from the PICT I versions) bitmap opcodes all end up mapping to two PS routines, so a certain amount of data alteration is done there. And the data describing a region, if it's more than just a rectangle, is extremely different from it's PICT form. But, these are exceptions to the general rules.

The Convert PICT app, then, mainly serves to do these conversions into PS form. It then writes all these instructions out with a decent sized prolog with definitions for all the calls made (e.g. the frameRect call above). When this PS file is run, it then generates an image that should look nearly identical to the original PICT image.

There are a couple things worth noting about the above process.

The first point is that those collections of PS code mentioned elsewhere (commented and uncommented ones) contain the definitions of the routines as mentioned above.

The more important point, though, is that the Convert PICT application doesn't do all the conversion internally. It converts the binary data to calls to equivalent routines in PS, and deals with the special cases like the bitmaps and regions. Because of this, a large amount of the work is done by the PS routines in thr prolog that is written out. This means it is easy for you to modify the behavior of the converted images. Do you think a routine is running too slowly? (this would not be a surprise. I've put no effort into optimization) If so, you can change the PS code files that this app uses when it writes out files. Convert PICT also tries to write out meaningful data, even for the `ignored' opcodes. So, if you wanted to provide support for the text drawing modes (which the default PS routines ignore), or pict comments, or whatever, you can make those

changes right in the PS code this relies on.

When the converter generates an eps file, it doesn't write the entire contents of one of those PS code collections. Instead, it only writes the parts that are needed (i.e. if lines were used in the picture, but rectangles weren't, then only the file containing line routines is written to the converted eps file). This explains why there are several files in each PS code collection.

The two collections of code are the `CommentedPSCode' and `UncommentedPSCode' directories. The Commented set is, essentially, the master set. That directory contains a script (`makeUncommented') which will copy all these commented files to the UncommentedPSCode directory while stripping comments out.

I hope this provides a decent overview of how the processing is done, and what you can do to modify it. I think the commented set of PS code will provide adequate details about what's being done to

implement the various instructions. If the info is inadequate, please don't hesitate to contact me!