Introduction

ibis.ps implements a typesetting engine and markup language. Text is broken into words and fitted onto lines, adjusting spacing for fully-justified blocks (with custom kerning).

There are various ways of executing a command to change a section of text. All commands are introduced by the @ character, known internally as the "sigil".

The simple command **i** for italics, can use the short form with various sets of delimiters.

@i[italics] produces *italics*

@i(italics) *italics*

@i<italics> *italics*

@i{italics} *italics*

@i 'italics' *italics*

@i :italics; *italics*

These last two require an extra space after the command name since the backquote and colon are not postscript delimiters.

If a short-form **i** command has its arguments all on one source-line, then the *ini/fin* pair of functions are orchestrated as part of parsing the line. Otherwise if the closing delimiter is not on the same source line, it is placed on a stack which governs the searching an parsing of subsequent lines.

The long form uses the same command names, but it is now the *argument* to the @Begin{} or @End{} command.

@Begin{i}*Start italics. @End{i}*End italics.

Incidentally, since the command name is scanned with 'token' and executed with 'exec', it can even be a postscript procedure.

@{/Courier 11 selectfont}text in Courier

```
text in Courier
```

Only "short-form" commands take a delimited argument. The @{arbitrary ps code} commands are not "short-form", and do not take an argument but apply directly to the current state. But they do receive the remaining portion of the line as a string, so a custom command may consume data from the string and yield the remainder to be printed (it should leave a string on the stack).

Now with **bold**.

```
typewriter-text oblique
```

`T` **`B`** *`I`* Roman should override all of ***italics,*** **bold,** and `typewriter` flags.

I finally remembered what "deferal" was all about. So let's see if it works. It should allow bracketed commands to span multiple lines. Like so: *This sentence should be all italics despite spanning lines, in a line-oriented scanning routine.* And back to normal.

Haha! I just read in the scribe paper that @Begin() and @End() sections should always be properly nested. So I just wasted some effort getting this to work:

@Begin{i} italic @Begin{b} bold-italic @End{i} bold @End{b} normal *italic **bold-italic** **bold*** normal

But it's probably best to nest things properly anyway. This should be considered "backup" behavior.

An interest has developed in changing the font size. Currently, this can be hacked with explicit postscript. Big text. back to normal. Double-size text. back to normal. Has the lead actually

been reset, or am I fooling myself? I think I may have written a bug where the lead can only increase. This extra text explains the purpose for this extra rambling text. Whew. Fixed.

I just read in the scribe user manual that the same brackets should be able to nest. This I have to fix. I had assumed that the variety of bracket choices () [] {} <> was for the convenience of the implementation, but I was wrong. It is for the convenience of the user, and the implementation has a little more work to do. Currently ibis does not correctly handle nesting of the same delimiters, and you should use different ones when nesting so it doesn't get confused.

Styles

I think I've built-up the requisite functionality to implement styles in a sensible manner. It's unfortunate that I can't locate the Scribe Expert Manual where specifying styles is supposed to be explained.

So far, a style is a short-hand for any number of "alterations" which can be installed and uninstalled in a controlled manner. An alteration is a dictionary containing two procedures: *ini* and *fin*, where *ini* returns an object which is later passed to *fin*.

The font operations have been implemented in terms of alteration dictionaries. The @i{} command tweaks a variable called *italic* and sets a variable called *fontchange* to **true**. The *ini* function for *i* returns the previous value of *italic* so that *fin*, upon receiving this value may restore it.

So a **style** is a composition of these alteration dictionaries. And pretty-much any behavior desired can be realized by constructing custom alteration dictionaries to be composed. The values returned by the *ini* functions are collected in an array which constitutes the style's *ini* function's return value. The style's *fin* function receives this array, calls `aload pop` and the composed *fin* functions are executed in the reverse order to consume their arguments naturally from the operand stack.