`wumpus-core` Guide

Stephen Tetley

January 9, 2011

1 About `wumpus-core`

This guide was last updated for `wumpus-core` version 0.41.0.

`wumpus-core` is a Haskell library for generating 2D vector pictures. It was written with portability as a priority, so it has no dependencies on foreign C libraries. Output to PostScript and SVG (Scalable Vector Graphics) is supported.

`wumpus-core` is rather primitive, the basic drawing objects are paths and text labels. A two additional libraries `wumpus-basic` and `wumpus-drawing` contain code for higher level drawing but they are experimental and the APIs they present are a long way from stable (they should probably be considered a *technology preview*).

Although `wumpus-core` is heavily inspired by PostScript it avoids PostScript's notion of an (implicit) current point and the movements `lineto`, `moveto` etc., instead `wumpus-core` aims for a more *coordinate free* style.

2 Exposed modules

`wumpus-core` exports the following modules:

`Wumpus.Core.` Top-level *shim* module to import all the exposed modules. Some internal data types are also exported as opaque signatures - the implementation is hidden, but the type name is exposed so it can be used in the type signatures of *userland* functions. Typically, where these data types need to be *instantiated* smart constructors are provided.

`Wumpus.Core.AffineTrans.` The standard affine transformations (scaling, rotation, translation) implemented as type classes, with a set of derived operations - reflections about the X or Y axes, rotations through common angles.

`Wumpus.Core.BoundingBox.` A data type representing a bounding box and operations on it. Bounding boxes are necessary for EPS output, they also support the definition of *picture composition* operators, e.g. aligning composite pictures horizontally, vertically, etc.

Wumpus.Core.Colour. A single colour type `RGBi` is supported. This type defines colour as a triple of integers (Word8) - black is (0, 0, 0); white is (255, 255, 255). Some named colours are defined, although they are hidden by the top level shim module to avoid name clashes with libraries providing more extensive lists of colours. `Wumpus.Core.Colour` can be imported directly if a simple list of named colours is required.

Wumpus.Core.FontSize. Various calculations for font size metrics. `wumpus-core` has only approximate handling of font / character size as it does not interpret the metrics within font files (doing so is a substantially task attempted by `Wumpus.Basic` but currently only for the simple and out-dated `AFM` font format). Instead, `wumpus-core` makes do with operations based on measurements derived from the Courier mono-spaced font. Generally using metrics from a mono-spaced font over-estimates for proportional fonts, though in practice this is tolerable.

Wumpus.Core.Geometry. The usual types an operations from affine geometry - points, vectors and 3x3 matrices, also the `DUnit` type family. Essentially this type family is a trick used heavily within `wumpus-core` to avoid annotating class declarations with constraints on the unit of measurement (usually `Double` representing the Point unit of publishing). With the `DUnit` trick, type constraints like `Fractional u` can be shifted to instance declarations rather than burden class declarations.

Wumpus.Core.GraphicProps. Data types modelling the attributes of PostScript's graphics state (stroke style, dash pattern, etc.). Note that `wumpus-core` labels all primitives - paths, text labels - with their rendering style, unlike PostScript there is no *inheritance* of a Graphics State in `wumpus-core` .

Wumpus.Core.OutputPostScript. Functions to write PostScript or encapsulated PostScript files.

Wumpus.Core.OutputSVG. Functions to write SVG files.

Wumpus.Core.Picture. Operations to build *pictures* - paths and labels within an affine frame. Generally the functions here are convenience constructors for data types from the hidden module `Wumpus.Core.PictureInternal`. The data types from `PictureInternal` are exported with opaque signatures by `Wumpus.Core.WumpusTypes`.

Wumpus.Core.PtSize. Text size calculations in `Core.FontSize` use *printer's points* (i.e. 1/72 of an inch). The `PtSize` module is a numeric type to represent them.

Wumpus.Core.Text.Base. Types for handling escaped *special* charcters within input text. Wumpus mostly follows SVG conventions for escaping strings, although glyph names should *always* correspond to PostScript names and never XML / SVG ones, e.g. for `&` use `#ampersand;` not `#amp;`.

Also note, unless only SVG output is being generated, glyph names should be used rather than char codes. For PostScript the resolution of char codes is dependent on the encoding of the font used to render it. As the core PostScript fonts use their own encoding rather than the common Latin1 encoding, using using numeric char codes (intending to be Latin1) can produce unexpected results.

Unfortunately, even core fonts are often missing glyphs that familiarity with Unicode and Web publishing might expect them support. Generally, a PostScript renderer cannot do anything about missing glyphs - it might print a space or an open, tall rectangle. As `wumpus-core` is oblivious to the contents of fonts, it cannot issue a warning if a glyph is not present when it generates a document, so PostScript output must be proof-read if extended glyphs are used.

`Wumpus.Core.Text.GlyphIndices.` An map of PostScript glyph names to Unicode code points.

`Wumpus.Core.Text.GlyphNames.` An map of Unicode code points to PostScript glyph names. Unfortunately this table is *lossy* - some code points have more than one name, and as this file is auto-generated the resolution of which glyph name matches a code point is arbtirary. `wumpus-core` uses this table only as a fallback if PostScript glyph name resolution cannot be solved through an encoding vector.

`Wumpus.Core.Text.Latin1Encoding.` An encoding vector for the Latin 1 character set.

`Wumpus.Core.Text.StandardEncoding.` An encoding vector for the PostScript StandardEncoding set. This encoder is associated with the core text fonts - Helvetica, Courier and Times-Roman. Typically core fonts will include further glyphs not indexed by the Standard Encoding, for PostScript these glyphs are addressable only by name and not by index.

`Wumpus.Core.Text.Symbol.` An encoding vector for the Symbol font which uses distinct glyph names. Unfortunately whilst the Symbol font is useful for PostScript, its use in SVG is actively discouraged by the W3C and some browsers.

`Wumpus.Core.VersionNumber.` Current version number of `wumpus-core` .

`Wumpus.Core.WumpusTypes.` This module collects internal types for Pictures, Paths etc. and presents them as opaque types - i.e. their constructors are hidden.

3 Drawing model

`wumpus-core` has two main drawable primitives *paths* and text *labels*, ellipses are also a primitive although this is a concession to efficiency when drawing

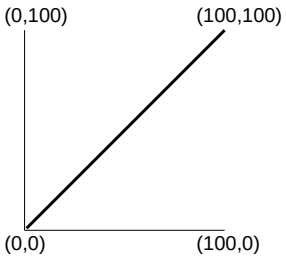

Figure 1: The world frame, with origin at the bottom left.

dots (which would otherwise require 4 to 8 Bezier arcs to describe). Paths are made from straight sections or Bezier curves, they can be open and *stroked* to produce a line; or closed and *stroked*, *filled* or *clipped*. Labels represent a single horizontal line of text - multiple lines must be composed from multiple labels.

Primitives are attributed with drawing styles - font name and size for labels; line width, colour, etc. for paths. Primitives can be grouped to support support hyperlinks in SVG output (so Primitives are not strictly *primitive*). The function `frame` assembles a list of primitives into a `Picture` with the standard affine frame where the origin is at (0,0) and the X and Y axes have the unit bases (i.e. they have a *scaling* value of 1).

`wumpus-core` uses the same picture frame as PostScript where the origin at the bottom left, see Figure 1. This contrasts to SVG where the origin is at the top-left. When `wumpus-core` generates SVG, the whole picture is generated within a matrix transformation [1.0, 0.0, 0.0, -1.0, 0.0, 0.0] that changes the picture to use PostScript coordinates. This has the side-effect that text is otherwise drawn upside down, so `wumpus-core` adds a rectifying transform to each text element.

Once labels and paths are assembled as a *Picture* they are transformable with the usual affine transformations (scaling, rotation, translation).

Graphics properties (e.g colour) are opaque once primitives are assembled into pictures - it is not possible to write a transformation function that turns elements in a picture blue. In some ways this is a limitation - for instance, the `Diagrams` library appears to support some notion of attribute overriding; however avoiding mutable attributes does keep this part of `wumpus-core` conceptually simple. To make a blue or red arrow with `wumpus-core` , one would make drawing colour a parameter of the arrow constructor function.

4 Affine transformations

For affine transformations Wumpus uses the `Matrix3'3` data type to represent 3x3 matrices in row-major form. The constructor `(M3'3 a b c d e f g h i)` builds this matrix:

$$\begin{matrix} a & b & c \\ d & e & f \\ g & h & i \end{matrix}$$

Note, in practice the elements g and h are superflous. They are included in the data type to make it match the typical representation from geometry texts. Also, typically matrices will implicitly created with functions from the `Core.Geometry` and `Core.AffineTrans` modules.

For example a translation matrix moving 10 units in the X-axis and 20 in the Y-axis will be encoded as `(M3'3 1.0 0.0 10.0 0.0 1.0 20.0 0.0 0.0 1.0)`

$$\begin{matrix} 1.0 & 0.0 & 10.0 \\ 0.0 & 1.0 & 20.0 \\ 0.0 & 0.0 & 1.0 \end{matrix}$$

Affine transformations on Pictures are communicated to PostScript as `concat` commands. For Pictures, `wumpus-core` performs no transformations itself, delegating all the work to PostScript or SVG. Internally `wumpus-core` transforms the bounding boxes of Pictures - it needs to do this to maintain their size metrics allowing transformed pictures to be composed with picture composition operators like the `picBeside` combinator.

PostScript uses column-major form and uses a six element matrix rather than a nine element one. The translation matrix above would produce this concat command:

```
[1.0 0.0 0.0 1.0 10.0 20.0] concat
```

Similarly, it would be communicated to SVG via a `group` element:

```
<g transform="matrix(1.0, 0.0, 0.0, 1.0, 10.0, 20.0)"> ... </g>
```

`wumpus-core` also supports the regular affine transformations on Primitives (the arbitrary matrix transformation `transform` is not supported). Transformations are implicitly interpreted in the standard affine frame - origin at (0,0) and unit scaling vectors for the bases.

For paths, all the transformations are precomputed on the control points before the output is generated. For labels and ellipses the *start point* of the primitive (baseline-left for label, center for ellipse) is transformed by `wumpus-core` and matrix operations are transmitted to PostScript and SVG to transform the actual drawing (`wumpus-core` has no access to the paths that describe character glyphs so it cannot precompute transformations on them).

One consequence of transformations operating on the control points of primitives is that scalings do not scale the tip of the *drawing pen*. If a path is stroked, lifted to a Picture and then scaled the whole graphics state is effectively scaled including the pen tip so the path is drawn with a thicker outline. However, if a path is stoked and then scaled as a Primitive, the drawing pen is not scaled so the path will be drawn with the regular line width.

5 Font handling

Font handling is quite primitive in `wumpus-core` . The bounding box of text label is only estimated - based on the length of the label's string rather than the metrics of the individual letters encoded in the font. Accessing the glyph metrics in a font requires a font loader - `wumpus-basic` has a font loader for the simple AFM font format but this is considered a special requirement and adds a lot of code[1]. As `wumpus-core` is considered to be a fairly minimal system for generating pictures it can live without font metrics.

In PostScript, mis-named fonts can cause somewhat inscrutable printing anomalies depending on the implementation. At worst, GhostScript may do no subsequent drawing after a failing to load a font. SVG renderers fallback to some common font if a font cannot be found.

A PostScript interpreter should have built-in support for the *Core 14* fonts. For the *writing* fonts, SVG renderers appear to support literal analogues quite well rather than needing general descriptors (e.g. `sans-serif` or `monospace`). The symbolic fonts Symbol and ZapfDingbats should be avoided for SVG - the W3C does not condone their use in SVG or HTML. Likewise, certain browsers reject them out of course.

The following table lists the Core 14 PostScript fonts and their SVG analogues, `wumpus-basic` includes a module `Wumpus.Basic.SafeFonts` encoding the fonts in this list and matching them to their appropriate encoding vectors.

PostScript name	SVG name
Times-Roman	Times New Roman
Times-Italic	Times New Roman - style="italic"
Times-Bold	Times New Roman - font-weight="bold"
Times-BoldItalic	Times New Roman - style="italic", font-weight="bold"
Helvetica	Helvetica
Helvetica-Oblique	Helvetica - style="italic"
Helvetica-Bold	Helvetica - font-weight="bold"
Helvetica-Bold-Oblique	Helvetica - style="italic", font-weight="bold"
Courier	Courier New
Courier-Oblique	Courier New - style="italic"
Courier-Bold	Courier New - font-weight="bold"
Courier-Bold-Oblique	Courier New - style="italic", font-weight="bold"
Symbol	(see text)
ZapfDingbats	(see text)

6 Acknowledgments

PostScript is a registered trademark of Adobe Systems Inc.

[1] Also, although it is still useful, the AFM format is out-of-date, supporting the more universal TrueType / OpenType formats would be a great deal of work.