FOPWritingModesAndBidiDesign

Authors: OlegTkachenko (ot), VictorMote (wvm), <add yourself>.

This page is an editable working draft of the

Writing modes and Bidi support design

Goals

The main goal is to achieve a robust support of all three main writing modes defined in the XSL Recommendation [6] : lr-tb (e.g. Western writing systems), rl-tb (e.g. Hebrew, Arabic) and tb-rl (e.g. Japanese, Chinese). (I'm not sure about tb-rl actually[ot]). Additional writing modes defined in Appendix A.1 of the Recommendation[9] are out of scope of this design.

Definitions

```
writing-mode
```

property[6] represents global cultural traditions of placement an information on a media and it's a main way in XSL to control the directions of placement of glyphs, blocks, lines, words etc. This is done by deriving

```
block-progression-direction
```

(BPDir) and

```
inline-progression-direction
```

(IPDir) traits from

```
writing-mode
```

property. In addition, IPDir for a sequence of characters may be implicitly determined using the Unicode Character Database (UCD)[10] and the Unicode Bidirectional Algorithm (Bidi) [8].

In most cases that information would be enough to properly process bidirectional text, but sometimes current writing mode and implicit direction need to be overriden and such kind of fine-tuning of bidirectional processing can be controlled by

```
direction
```

and

```
unicode-bidi
```

properties of

```
fo:bidi-override
```

formatting object.

```
writing-mode
```

```
writing-mode
```

property [6] applies only to the following formatting objects:
#

```
fo:simple-page-master
```

#

```
fo:region-*
```

#

```
fo:table
```

#

```
fo:block-container
```

#

```
fo:inline-container
```

Values: lr-tb (default) | rl-tb | tb-rl | lr (shorthand for lr-tb) | rl (shorthand for rl-tb) | tb (shorthand for tb-rl) | inherit. Each value of

```
writing-mode
```

sets IPDir, BPDir and

```
shift-direction
```

traits on the reference area.

Mapping of corresponding properies from absolute to relative one is as follows:
lr-tb

${renderedContent}

rl-tb

${renderedContent}

tb-rl

${renderedContent}

How

```
writing-mode
```

is used:
#IPDir and BPDir are used for stacking respectively inline and block areas
#

```
fo:simple-page-master
```

- placement of the regions on the master
#

```
fo:region-body
```

- stacking of columns and default flow of text from column to column
#

```
fo:table
```

- layout of rows and columns (BPDir determines row-stacking direction and IPDir determines colums-stacking direction (and cell order within a row)).

```
direction
```

Values: ltr (default) | rtl | inherit.

Usage of

```
direction
```

property [4] in XSL is almost deprecated, except for the case of controlling/overriding of IPDir determined by the current writing mode and the implicit direction determined by UCD and Bidi. Applies only to

```
fo:bidi-override
```

formatting object.

The property only has an effect on text in which orientation of the glyphs is perpendicular to the IPDir, therefore for lr-tb and rl-tb writing modes

```
direction
```

property affects only non rotated glyphs and for tb-rl - only rotated glyphs.

```
unicode-bidi
```

```
unicode-bidi
```

property [5] only applies to

```
fo:bidi-override
```

formatting object. Values: normal (default) | embed | bidi-override | inherit. Along with

```
direction
```

property it opens new embedding level or creates override with respect to Bidi, see [8].

How it affects Bidi processing:

*

```
<bidi-override direction="ltr" unicode-bidi="embed"> foo </bidi-override>
```

*

${renderedContent}

*

```
<bidi-override direction="rtl" unicode-bidi="embed"> foo </bidi-override>
```

*

    ${renderedContent}

*

```
<bidi-override direction="ltr" unicode-bidi="bidi-override"> foo </bidi-override>
```

*

    ${renderedContent}

*

```
<bidi-override direction="rtl" unicode-bidi="bidi-override"> foo </bidi-override>
```

*

    ${renderedContent}

Where LRE is U+202A, RLE is U+202B, PDF is U+202C, LRO is U+202D and RLO is U+202E.

"normal" value was designed in CSS to change directional type of non-textual emtities such as images, but as in XSL the property appplies only to

```
fo:bidi-override
```

formatting object, therefore "normal" value is not used and all non-textual entities are treated as neutral characters (more specifically as OBJECT REPLACEMENT CHARACTER (U+FFFC) according to Bidi algorithm).

XSL Bidi processing conceptual model

The final phase of refinement uses Bidi algorithm and UCD "...to convert the implicit directionality of the text into explicit markup in terms of formatting objects". E.g. LLLRRR text with ltr IPDir would be translated into LLL<fo:bidi-override direction="rtl" unicode-bidi="bidi-override">RRR</fo:bidi-override>. Bidi algorithm as defined in [8] requires some adaptations to fit into XSL processing model. Here is adopted conceptual model:

Step 1. Breaking into DTRs

In XSL Bidi algorithm is applied to delimited text ranges (DTR) instead of paragraphs. A DTR is a maximal flattened sequence of characters (FSC) that doesn't contain any delimiters. A FSC is created by pre-order traversing of a FO tree fragment down to

```
fo:character
```

level. During the traversal, every

```
fo:character
```

formatting object adds a character to the sequence and

```
fo:bidi-override
```

formatting object with

```
unicode-bidi
```

property with a value of "embed" or "bidi-override" adds appropriate directional formatting code (such as LRE, RLE, LRO or RLO) before traversing its content and PDF code after. Delimiters are: any formatting object that generates block-areas,

```
fo:multi-case
```

and any text with glyph orientation that is not perpendicular to the dominant-baseline.

For each DTR, default bidirectional orientation (paragraph embedding level) is determined according to IPDir of the nearest ancestor-or-self formatting object that generates a block-area.

Step 2. Resolution of the embedding levels

Each character in the DTR is labeled with a resolved embedding level. rtl text will always end up with an odd level and ltr and numeric text will always end up with an even level. Then new

```
fo:bidi-override
```

formatting objects with appropriate values of the

```
direction
```

and

```
unicode-bidi
```

properties are inserted into the FO tree fragment that was flattened into the DTR such that the following constraints are satisfied:

1. For any character in the DTR, IPDir matches resolved embedding level.
2. Newly insterted

   ```
   fo:bidi-override
   ```

   formatting objects don't break nesting relationship and retain computed property values.
3. Minimum number of

   ```
   fo:bidi-override
   ```

   formatting objects was inserted.

Step 3. Reordering the text

The final text reordering step is not done during refinement. Instead, XSL equivalent of reordering is done during formatting. IPDir of each glyph, which is explicitly determined during the previous step is used to control the stacking of glyphs.

(wvm+ot)Note that according to Bidi algorithm the reordering acts on a per-line basis and process of breaking a paragraph into lines is outside the scope of Bidi algorithm. (This effectively means that line breaking algorithm acts on a text in logical order). For example, suppose that lower-case letters represent Latin text, and upper-case letters represent Hebrew text, and that the layout context is lr-tb. The text "abc defg ABCD EFG hijk lmno", requiring a line break between the two Hebrew words, should be rendered as follows:

```
abc defg DCBA
GFE hijk lmno
```

NOT as:

```
abc defg GFE
DCBA hijk lmno
```

Bidi mirroring

When characters are shaped into glyphs, the mirroring process defined in Bidi algorithm must take place. This takes resolved embedding levels into acount: if glyph-orientation="90" and the embedding level is odd or if glyph-orientation="-90" and the embedding level is even - the character needs to be mirrored.

Implementation

Bidi implementation

Requirements

Bidi implementation must provide API for the following tasks:

- Checking out whether a text requires bidi processing.
- Resolution of embedding levels within a text.
- Reordering objects in a visual order according to their levels.
- Mirroring of characters.

Available implementations

First of all the mirroring is actually rather simple task, there is a predefined character-to-character mapping set including about 150 characters, which defines bidi mirroring. We can do it ourself of alternatively we can make use of static <code>mirrorChar(int c)</code> method of <code>org.apache.batik.gvt.text.BidiAttributedCharacterIterator</code> class.

...

Reference materials

- [1] The spec 1.2.5 Internationalization and Writing-Modes
- [2] The spec 5.5.3 Writing-mode and Direction Properties (refinement)
- [3] The spec 7.27 Writing-mode-related Properties
- [4] The spec 7.27.1 "direction"
- [5] The spec 7.27.6 "unicode-bidi"
- [6] The spec 7.27.7 "writing-mode"
- [7] The spec 5.8 Unicode BIDI Processing
- [8] The Bidirectional Algorithm – http://www.unicode.org/unicode/reports/tr9/
- [9] The spec A.1 Additional "writing-mode" values
- 10 Unicode Character Database – http://www.unicode.org/Public/UNIDATA/