A PDF-formatter for DSSSL-FlowObjectTrees

A backend to the DSSSL-processors Jade / OpenJade

Christof Drescher

Pro Image GbR, Marburg

29.09.2000, Release 1.02

Table of Contents:

1 Introduction

This is the documentation of the fot2pdf-formatting backend to DSSSL-flow object trees provided by the DSSSL-processor Jade. This documentation is not at all a complete and thoroughly planned document, but rather a short introduction.

1.1 Features

Unlike other backends of JADE, which transform the SGML document into formats which have to be processed by other, external formatters (RTF, TEX etc.), the PDF backend does the final formatting (including line- and page-breaking as well as import of external graphics) itself; therefore, the documents created are final and do not change when being distributed electronically whatsoever.

The formatter has been developped with a focus on high-quality output ready for printing even on high-resolution machines; therefore, it uses PostScript Type1 fonts natively, allows for proper kerning and the use of ligatures and can import other PDF files directely. The line-breaking is done by an optimizing algorithm (as in TEX) and generates pleasant breaks in the source stream.

Its main features are listed here:

- Supports all major flow objects for standard text documents.
- Uses PostScript Type1 fonts, with the possibility to embed the fonts into the document for complete portability.
- Line breaking is performed by an optimizing algorithm to have a professional looking output even if little or no hyphenation information is available.
- Can import other PDF files as an external-graphic to be included into the document; this allows for importing almost any bitmap or vector graphics without any quality loss whatsoever.
- Typographic features include proper kerning for all fonts, support for hanging punctuation, fi/fl-ligatured characters, no-break-space support etc.
- Linking in the document is fully supported, making online reading extremely simple.
- Nesting of box and table objects is fully supported.
- A documented extension allows for links to access any URL from the PDF file.

1.2 Requirements

The following is a (non-exclusive) list of what is required to run the fot2pdf formatter.

- A running JAVA 1.2 runtime environment to execute the program.
- For the (Type1) fonts to be used, an AFM file is needed for proper formatting, and a PFB file is necessary if font embedding is turned on.

- Two auxiliary files need to be accessible. These are for hyphenation and character replacing (hyphen.txt) and for proper font mapping (fontmap.txt).

1.3 Getting started

tbd.

1.4 License

The use of this backend is free for non-commercial use. The version is not crippled in any way, besides the note on non-commercial use in each created PDF document.

For commercial use, which includes the distribution of documents generated with this backend together with other items which are not free of charge, a license by the author is required.

This software is provided “as-is” and any expressed or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author of the software, Christof Drescher / “Pro Image GbR”, be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

There is no guarantee of technical support, though I am usually able to answer enquiries within a few days. Please contact the author by e-mail or visit the web-site http://www.pdftech.com, where information on this formatter will be published in the future.

2 Usage of fot2pdf

The fot2pdf formatting backend is included as a JAVA bytecode program, with all the necessary modules put together in a handy ZIP file. This section shows how to use the program for producing PDF documents from SGML data.

Installation

There is no special installation procedure required to run the fot2pdf formatting system. The JAVA class files (i.e. the JAR file **`fot2pdf.jar`**) should be available to the user; their location is passed as the **`classpath`** parameter to the JAVA runtime environment. Additionally, the *fontmap.txt* and *hyphen.txt* files should be available (their location is also passed as a runtime parameter).

Creating the fot-file using JADE

The input format for fot2pdf is a DSSSL flow object tree which is generated by the DSSSL processor JADE. For a complete description on the capabilities and the usage of JADE, the reader should have a look at the JADE documentation itself. The basic command to produce the fot-file is

`jade -D` *`jadedir`* **`-t fot -d simple.dsl simple.sgml`**

where “jadedir” is the directory the JADE executable is installed. The output file is named *simple.fot* in this case and is the only one read by fot2pdf; the original SGML, DTD or DSSSL files are not needed.

Starting fot2pdf

The fot2pdf program is invoked by a typical JAVA command:

`java -classpath` *`fot2pdf.jar`* **`com.pdftech.fot2pdf.app`** *`infile.fot switches`*

The fot2pdf system needs several files to be accessible; they are first looked for in the current directory, then in the directory denoted by the “-D” command line switch. This allows to have special settings (e.g. for the hyphenation dictionary or font data) together with the document they were created for.

File “hyphen.txt”

The first auxiliary file is the hyphenation dictionary (see Chapter 3.4). The entries used are described in the following table.

Type	Description
W-entry	Example: *W "transformation" "trans-for-mation"* This is the main entry: It replaces words by words with soft hyphens, which are feasible breakpoints for line-breaking. A word is actually any sequence of characters between “delimiters”, such as !?,.- and, of course, the space character.
A-entry	Example: *A "fi" "ﬁ"* This rule replaces any occurrence of a character sequence with another; in this case, any “f” followed by an “i” is replaced by the ligature ”fi”. To achieve special results, it is important to notice that the order of A-entries is preserved (for instance one can replace a tilde (~) with a no-break-space, but to get a real tilde glyph, replace two no-break-spaces by a tilde again.
P-entry	Example: *P "for example" "e.g."* If one wishes to replace a certain phrase which contains delimiting characters itself, the word replacement rule would not suffice; the A-rule yet would be too inefficient. Thus, this entry can be used for just that. Actually, if a W-entry does indeed include a delimiting character, it is automatically treated as a P-entry.

The user should notice that the order of replacing characters is defined as follows: First, all A-entrys are replaced in a given text, with the order being preserved in between the entries themselves (see above). Then, all P-entries are worked through. Finally, a given line is walked through and each word is checked against the W-entries (using a hash-table). This is important, since any global replacing by an A-entry precedes the standard word substitution. Thus, the following code does not work:

```
A "fl" "&#64258;"
```

```
W "inflate" "in-flate"
```

Here, the A-entry takes care that only fl-ligatures are used; the W-entry should provide a hyphenation information. Yet, when the word “inflate” has been found in a text, it already includes a ligature (the Unicode char #64258), while the W-entry does not. Thus, no replacing will ever occur. The correct entry would therefore be (though not very simple to type):

```
W "in&#64258;ate" "in-&#64258;ate"
```

For further studying the concepts of the hyphenation dictionary, the reader may have a look at the file used for this diploma thesis.

File “fontmap.txt”

The second auxiliary file being needed is the font-mapping list, which replaces the font-names in the DSSSL script by font filenames of PostScript Type1 data. As described in , this is necessary to have a proper mapping between the fixed values for the **`font-weight/font-posture`** characteristics and the (sometimes quite awkward) naming of Type1 fonts. Again, two types of entries are possible in *fontmap.txt*:

Type	Description
substitution entry	Example: *iso-serif=times* Example 2: *courier-italic=courier-oblique* This replaces a complete font family or one of its faces by another. The first example shows a quite common method to have the default ISO-font replaced by a Base14 font in PDF. The second example is a typical replacement to adjust the different naming conventions; while the slanted version is usually called “italic”, for some fonts (e.g. Courier) it is named “oblique”. By this means, any combination can be mapped correctly. If fot2pdf fails to find a certain font, it issues a warning message with the name requested in the flow object tree; the user can add the appropriate entry to *fontmap.txt* then. A quite interesting feature for practical purposes is to replace a font for a document without changing the DSSSL script itself or to produce different versions from an already created flow object tree.
filename mapping entry	Example: *Times-Bold \diplom\fonts\Base14\tib_____* This is the real mapping of a single font face to the corresponding file. The system needs the AFM file and, in case font embedding is enabled, the PFB file at the named place. Any font used must be properly listed in *fontmap.txt*.

Note that all processing by *fontmap.txt* is done in the order of lines in the file. A typical error would be to have a substitution after a fontname mapping; thus making the substitution invisible to the formatter. As a rule, all substitution entries should be at the beginning of the file.

Command line switches

Several special features can be enabled or disabled by command line parameters. The following is a list of the currently available switches and their impact on the formatting process:

Switch	Description
-o *filename*	Example: *java com.pdftech.fot2pdf.app simple.fot -o testfile.pdf* This sets the output filename explicitly. Normally, the outfile is named the same as the input file with the “fot” extension replaced by “pdf”.
-s *class name of sax parser*	Example: *java com.pdftech.fot2pdf.app -s com.jclark.xml.sax.Driver simple.fot* This enables the use of a SAX compliant parser to feed the flow object tree (which is acutally a well-formed xml document) directly into the formatter. This is usually a bit slower than reading the file directely, but allows for other applications to interact directely with the formatter; in the future, support for XSLT engines might be supported as well
-D *fot2pdfpath*	Example: *java com.pdftech.fot2pdf.app simple.fot -D \programs\bin\fot2pdf* This parameter sets the path where fot2pdf looks for its auxiliary files (see above). This allows to run the fot2pdf program from any location but with the necessary files available nonetheless. This is especially important when including images or other PDF files with relative filenames. Default is the current directory.
-l*[0/1]*	Example: *java com.pdftech.fot2pdf.app simple.fot -l1* This is the “link highlighting” flag; if it is switched on, all the links in a document are automatically colored blue to make them distinguishable from normal text. In print, this shade of blue produces a slightly grayed text tough; it should be used preferably for online display. Default is 0=off.
-c*[0/1/2]*	Example: *java com.pdftech.fot2pdf.app simple.fot -c0* This switch controls the compression level of the resulting document. The value “0” disables compression completely; it can be used to produce Acrobat 2-compatible documents (which do not accept the flate compression used by fot2pdf). The setting “1” compresses all data but the contents, i.e. images and font data, which is interesting to see the PDF markup produced by the formatter. The setting “2” compresses all streams using the flate technology, which should normally be enabled to produce the smallest file size possible. Default is 2=full compression.
-e*[0/1]*	Example: *java com.pdftech.fot2pdf.app simple.fot -e0* This switch controls font-embedding; only PDF documents with embedded fonts are guaranteed to look identical on all systems. Normally, embedding should therefore be enabled. Yet, if there are legal constraints for the distribution of a font's data, it might be necessary to switch embedding off by setting this switch to “0”. Default is 1=fonts are embedded.

-m*[0/1]*	Example: *java com.pdftech.fot2pdf.app simple.fot -m1* This switch enables multithreaded processing of file-reading and formatting; normally, the fot file is read first and processed afterwards. While basically being a good idea to have these tasks multithreaded, several JAVA runtime engines actually slow down the processing when doing so. For certain files, especially when formatted on low-memory systems, it might be interesting to test this switch though. Default is 0=no multithreading.
-b*[0/1]*	Example: *java com.pdftech.fot2pdf.app simple.fot -b1* This switch enables page balancing, i.e. having the page's content end on the same bottom line if possible. The DSSSL script needs to have “flexible” spacing values (using the **`display-space`** constructs) to have an effect when the PDF file is written. The formatter will then try to balance all pages which are not forced by page-break characteristics or are the end of a simple-page-sequence. If balancing could not be done due to missing “stretchability”, a warning message is issued. Default is 0=no balancing.
-B*[0/1]*	Example: *java com.pdftech.fot2pdf.app simple.fot -B1* This switch enables booklet mode; in this mode, the PDF engine sorts the pages for 2up duplex printing. Default is 0=no booklet reordering.
-w *warninglevel*	Example: *java com.pdftech.fot2pdf.app simple.fot -w 1* This parameter is a threshold value for warning messages concerning the optimizing line-breaking algorithm. The value supplied is the number by which the needed stretching of spaces exceeds the normal stretchability of a space (which is fixed to ½ of a standard space character). For example, if the warning level is set to 1, it warns when no “perfect” solution (no stretchability exceeded at all) could be found; a warning level of 16 warns only at extremely bad cases. A value of “0” disables warning altogether. For the setting of tolerance levels for the formatting process of paragraphs, see below. Default is a level of 4, i.e. warning if spaces exceed twice the standard size.

A simplified list of these parameters can be obtained by invoking fot2pdf without any parameters.

Line-breaking/line-composition parameters

As described in , the optimizing line-breaking algorithm as well as the line-composition task can be controlled by certain parameters supplied in the DSSSL script. For this purpose, the DSSSL standard includes the two characteristics **`line-breaking-method`** and **`line-composition-method`** which take a public identifier; these identifiers have been extended for the use by fot2pdf to take the required information.

The public identifiers start with the header *UNREGISTERED::Christof Drescher//fot2pdf//* followed by the type (line-breaking/line-composition), the value and possibly parameters appended, delimited by “::”. The detailed list is as follows:

UNREGISTERED::Christof Drescher//fot2pdf//line-breaking-method::*value*

Value	Description
simple	This selects the simple first-fit algorithm for line-breaking, which breaks after the first possibility which fills a line correctly. This is the default setting for left-aligned, right-aligned and centered paragraphs. It does not take any parameters.
optimizing	This selects the optimizing algorithm, which normally finds the best possible solution for a given paragraph. This is the default setting for justified paragraphs. The following parameters can be appended: **tolerance=***value*: Sets the tolerance, which is the given value multiplied by ½ of a normal character space. The algorithm does only look for breakpoints which do not exceed this limit. Sometimes, a solution cannot be found with the standard value, and an increased value might be necessary. The default value is 1 and is increased as necessary up to a value of 16. It might also be advisable for some paragraphs to have the algorithm start with a higher value from the start, if it is clear that no “simple” solution will be found. If there is no way to find a solution with a tolerance of 16, the simple algorithm is used (unless this parameter is explicitly stated higher). **hyphenpenalty=***value*: Sets the penalty for a line ending with a hyphen to a certain value. The default value is 50. To get even less hyphenated lines than normally, this value might be increased; the algorithm might then actually prefer a solution with lines being more “loose” but with less hyphenated lines. The value must be chosen as desired (e.g. to be 500 or more). **changelength=***value*: Instructs the algorithm to produce a solution which is forced to be a line shorter or one or more lines longer than it would optimally be. Thus, the values normally supplied are -1, +1 or (in rare cases) +2 to have certain paragraphs change in their size. This is for fine-tuning documents after a preliminary run with standard values, e.g. to have some pages look even better.

UNREGISTERED::Christof Drescher//fot2pdf//line-composition-method::*value*

Value	Description
wordspacing	If a line is composed with wordspacing, only the width of its spaces is adjusted to fill the line. If there is only a single word on the line, the line will be left-aligned.
charspacing	When using charspacing, the spacing between the line's glyphs is evenly adjusted to justify the line.
flexible	The flexible method uses a three-fold approach: First, it uses wordspacing up to a given threshold, then charspacing up to the limit given by the justify-glyph-space-max-add characteristic. If there is still need for more space, wordspacing is used again. The appropriate parameter is: **threshold=***value*: The value after which charspacing is enabled.

The different line-composition methods have been described in . Note that for a working charspacing it is necessary to have the **`justify-glyph-space-max-add`** characteristic of a paragraph set to a non-zero value; otherwise, no justification will be possible at all.

3 Some special topics of interest

3.1 Technical usage of fonts

In DSSSL, there are two ways to reference a font: One way is to give a fully qualified name via the **`font-name`** characteristic, which then includes all information (e.g. “Frutiger-BlackItalic” is a certain part of the Frutiger font family). The other way is by giving the several sub-attributes of a font (like weight, posture etc.) separately, e.g. stating that the font to be used is the bold, upright version of the Times font family. Unfortunately, the way PostScript fonts are named is not a clear and uniform system. For instance, a slanted version of a font might be called either “oblique” or “italic”. It would be inconvenient to remember and change all the special cases when changing fonts in a DSSSL-script; thus, a special mapping system had to be established: In fot2pdf, the available fonts have to be listed in a file called *fontmap.txt* along with their mapping to the filename on the host system; furthermore, a replacement-system is integrated to transform certain **`font-weight/font-posture`** combinations to others; if a DSSSL-script names a “Courier-Italic” font for example, it could be automatically changed to “Courier-Oblique”, since this is the correct name for the slanted version. By this means, an appropriate way of dealing with fonts could be implemented.

For compliance with the PostScript- and PDF-standards, there is a certain subset of fonts called the “base 14 fonts”, which are required to be included in all correct implementations of PostScript or PDF software. These include members of the Courier, Helvetica and Times families, along with Symbol and ITC Zapf Dingbats. These fonts need not be included in PDF files; fot2pdf also maps the DSSSL standard font families **`iso-serif`**, **`iso-sanserif`** and **`iso-monospace`** to these base 14 fonts. The information for these fonts has been included in the fot2pdf distribution, also showing the usage of naming conventions described here.

As mentioned before, one advantage of PDF is the possibility to embed fonts into the file, which simplifies the exchange of documents a lot. Yet, there is a legal issue involved in doing so.

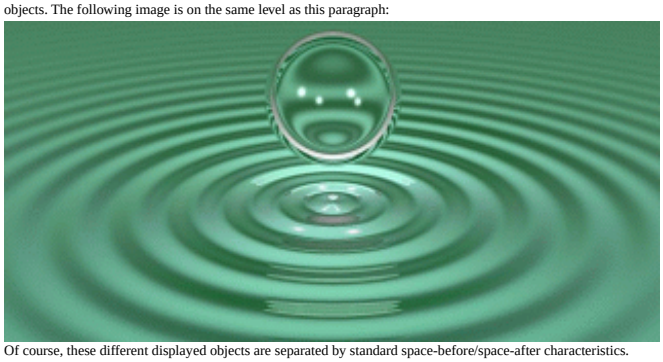
Most font data today, especially using the “professional” Type1 format, is copyrighted material and must be purchased. The license for fonts includes the right to print documents using these fonts, but prohibits copying the font files themselves. Some typeface companies include an explicit note that the font data must not be used in a file format which allows for re-engineering the fonts in its entire form. A solution has been introduced for PDF files by embedding only subsets of fonts, which is largely accepted by typeface companies.

The fot2pdf formatter supports the inclusion of fonts as well; yet, for simplicity reasons it can only include the complete PFB font file. Therefore, the legal issue had to be addressed by allowing the user to disable this font embedding by a command line switch.

3.2 The table and box objects

Two types of flow objects play a special role in the formatting engine: Both the **`table`** and the **`box`** object are not atomic or made of inlined data, but contain several displayed objects as their content. Therefore a recursive approach had to be implemented: When a box object is found, the processing of its content is done using the very same method as of the main DisplayStream, yet the applicable margins are decreased by the indents imposed from the box object. The resulting data is a list of displayed objects which is immediately converted into content boxes, applying all glue objects with their standard values. Together with the graphical marks of the box itself, these content boxes are put into a single DisplayBox ready for pagination.

Since this approach can be iterated, even complex structures like nested tables can be achieved easily. For an example, see Figure 1.

Small column headline	**Large column headline**
First example.	This is the first row. To show the flexibility of the table's implementation, we actually have a sequence of displayed objects. The following image is on the same level as this paragraph:  Of course, these different displayed objects are separated by standard space-before/space-after characteristics.
Second example.	On this second row, we show another fine possibility: A *nested table*:

Product	**Spreading**	**Size**	**read/write**
3.5" floppy	popular	1.4mb	r/w
cd-rom	popular	650mb	read only
dvd	medium	up to 6gb	read only
dat-tape	unpoular	up to 40gb	r/w

Fig. 1: An example for nested tables and the possibility to have several displayed objects in a table-cell. Even more, the whole figure is encapsulated by a box object.

3.3 Import of external graphics

The import of external files is always a rather tedious part for any formatting system. Not only that there are many different file formats available today which could be implemented, but many of them require formatting (such as **`RTF`** files) or even interpreting (such as **`PostScript`** files) to be able to reproduce their content. This often leads to changes when viewed by different applications or to the problem that several different formats need to be used in order to include raster graphics, vector graphics or combinations of these in their native formats.

For professional use, these problems complicate the work flow unnecessarily; thus, a rather simple but straightforward approach has been chosen: The basic format accepted is PDF. This means that any page of an external file in correct PDF can be imported, reproducing the content in the generated document exactly (imported graphics can be properly resized to fit the available space). This decision eliminates the need for different graphics formats, since all other formats can be converted easily into pdf, and there is no loss in quality whatsoever. Furthermore, no problems arise for missing fonts, incomplete import or unsupported features – whatever the original file contains is put into the final PDF file without any change (i.e. other than its scale). The technical procedure is described on page .

For mere image data, the formatter also accepts **`GIF`** and **`JPEG`** files to be imported; these formats cover the standard usage to include photos and bitmap graphics (e.g. for producing both PDF and HTML pages from a single SGML source).

3.4 Hyphenation

For proper line-breaking, hyphenation is a key necessity; in DSSSL, there is only a very limited way of providing the needed information. Yet, implementing a complete hyphenation-algorithm along with a complete dictionary for the languages used would have been a rather complex task. Therefore, a simple word-replacing system has been implemented using a **replacement table**. It enables any word, i.e. a sequence of chars delimited by white space, to be replaced by some other sequence of chars; this sequence possibly includes soft hyphens. By this means, the user can include hyphenation information for a relevant subset of the words used in the document, possibly after a preliminary run of the formatter which might show where hyphenation is necessary after all. Furthermore, a system to replace any sequence of characters apart from words has been implemented as well; this can be used to globally replace ligatures or typographical quotation marks. For a detailed description of the file format, see Chapter 2, *Usage of fot2pdf*, page 5.

3.5 Restrictions and omitted features

A couple of compromises had to be made to complete the formatter in acceptable time yet having usable functionality. Some flow objects and characteristics have been left out completely or have been implemented only partially. Apart from a few objects being simply ignored for complexity reasons, some characteristics are not relevant to the present state of Jade (like attributes special to **`column-set`** formatting) or are not likely to be used at all. For a complete list of all characteristics in DSSSL and their implementation state, see Appendix A.

The main features not implemented are:

- The DSSSL flow objects for mathematical formulae have not been implemented at all; a workaround would be to create them using a special program and include them as external graphics via PDF.
- The **`table`** flow object is implemented only for basic formatting needs; especially, neither an auto-width-feature nor cells spanning multiple columns are implemented. Yet, this basic implementation is enough for most standard needs, as can be seen at the tables in this document.

- The **box** flow object class can only enclose a complete displayed object; a complete implementation would require the handling of many special cases like boxes around inlined objects covering several lines, going over page bounds etc. Since the box object is not used very often in standard DSSSL scripts today, these special cases have not been implemented.
- There are no provisions for special alignment modes found in the paragraph flow object.
- Scoring by a character (e.g. canceling a word by putting a sequence of 'X' glyphs over it) is not implemented.
- Displayed objects may have characteristics called **may-violate-keep-before** and **may-violate-keep-after** which allow page-breaking in between a sequence normally being kept together (like an “emergency trapdoor”). fot2pdf ignores these characteristics; in the rare cases of an unbreakable page, it will issue a warning and put the first object on it nevertheless.

All in all, the formatter implements a surely usable subset of the style part of DSSSL.

Appendix A: Overview of supported flow object characteristics

This table lists all supported flow objects with their respective attributes; if a value is listed as “not applicable”, it is either not supported by Jade or is a value for a mode not implemented in this formatter (like right-to-left writing mode). Also, many characteristics are intended for use with **`column-set`** or **`page-sequence`** layouts which are currently not supported by Jade.

object	**characteristics**
any displayed object	**Supported:** space-before, space-after, keep-with-previous?, keep-with-next?, break-before, break-after, keep *(only page applicable)* **Not supported:** may-violate-keep-before?, may-violate-keep-after?
sequence	**Supported:** any
display-group	**Supported:** any **Not applicable:** coalesce-id, position-preference
simple-page-sequence	**Supported:** page-width, page-height, left-margin, right-margin, top-margin, bottom-margin, header-margin, footer-margin, left-header, center-header, right-header, left-footer, center-footer, right-footer **Not applicable:** writing-mode
page-sequence	*(Not implemented in JADE)*
column-set-sequence	*(Not implemented in JADE)*
paragraph	**Supported:** line-spacing, min-pre-line-spacing, min-post-line-spacing, min-leading, first-line-start-indent, last-line-end-indent, hyphenation-ladder-count, hyphenation-keep *(only page applicable)*, font-family-name, font-weight, font-posture, font-structure, font-proportionate-width, font-name, font-size, quadding, last-line-quadding, last-line-justify-limit, justify-glyph-space-max-add, justify-glyph-space-max-remove, line-breaking-method, line-composition-method, widow-count, orphan-count, start-indent, end-indent, hanging-punct? **Not applicable:** line-spacing-priority, asis-truncate-char, asis-wrap-char, first-line-align, alignment-point-offset, hyphenation-char *(fixed to '-' in Jade)*, hyphenation-exceptions *(not supported by Jade, we use internal hyphenation anyway)*, implicit-bidi-method, numbered-lines?, line-number, line-number-side, line-number-sep *(No line-numbering possible since line-number is fixed to #f in Jade)*, language, country, position-preference, writing-mode, span, span-weak?, glyph-alignment-mode **Not supported:** lines, asis-wrap-indent, ignore-record-end?, hyphenation-remain-char-count, hyphenation-push-char-count, expand-tabs?
paragraph-break	*(see paragraph)*
line-field	**Supported:** field-width, field-align **Not applicable:** writing-mode **Not supported:** inhibit-line-breaks?, break-before-priority, break-after-priority
sideline	**Supported:** sideline-side, sideline-sep, color, line-cap, line-thickness, line-repeat, line-sep **Not applicable:** line-dash **Not supported:** layer
anchor	*(Not implemented in JADE; anchors are given implicitly in the flow object tree)*
character	*(Not implemented in fot2pdf at all)*

leader	**Supported:** length, align-leader, min-leader-repeat **Not supported:** truncate-leader?, inhibit-line-breaks?, break-before-priority, break-after-priority
embedded-text	*(Not implemented in fot2pdf since there is just left-to-right writing mode)*
rule	**Supported:** orientation, length, color, line-cap, line-thickness, line-repeat, line-sep, position-point-shift, display-alignment, start-indent, end-indent **Not applicable:** line-dash, position-preference, writing-mode, span, span-weak? **Not supported:** layer, inhibit-line-breaks?, break-before-priority, break-after-priority
external-graphic	**Supported:** display?, scale, max-width, max-height, entity-system-id, notation-system-id, display-alignment, start-indent, end-indent **Not applicable:** color *(color is preserved from the imported graphic)*, writing-mode, span, span-weak? **Not supported:** layer, position-point-x, position-point-y, escapement-direction, inhibit-line-breaks?, break-before-priority, break-after-priority
included-container	*(Not implemented in JADE)*
score	**Supported:** type *(not by character)*, score-spaces?, color, line-cap, line-thickness, line-repeat, line-sep **Not applicable:** line-dash **Not supported:** layer, inhibit-line-breaks?, font-family-name, font-weight, font-posture, font-structure, font-proportionate-width, font-name, font-size
box	**Supported:** box-type, background-color, color, line-cap, line-thickness, line-repeat, line-sep, start-indent, end-indent **Not applicable:** display?*(only displayed boxes implemented)*, box-open-end?*(box-content is always kept together)*, box-size-before, box-size-after, line-dash, line-miter-limit, writing-mode, position-preference, inhibit-line-breaks?, break-before-priority, break-after-priority, span, span-weak? **Not supported:** background-layer, box-corner-rounded, box-corner-radius, layer, line-join
side-by-side	*(Not implemented in JADE)*
side-by-side-item	*(Not implemented in JADE)*
glyph-annotation	*(Not implemented in JADE)*
alignment-point	*(Not implemented in fot2pdf)*
aligned-column	*(Not implemented in JADE)*
multi-line-inline-note	*(Not implemented in JADE)*
emphasizing-mark	*(Not implemented in JADE)*
math flow objects	*(Not implemented in fot2pdf at all)*
table	**Supported:** table-width, table-border, before-row-border, after-row-border, before-column-border, after-column-border, display-alignment, start-indent, end-indent **Not applicable:** table-auto-width-method, table-corner-rounded, table-corner-radius, position-preference, writing-mode, span, span-weak?
table-part	*(Not implemented in fot2pdf)*
table-column	**Supported:** column-number, width, display-alignment **Not supported:** n-columns-spanned, start-indent, end-indent
table-row	*(No characteristics)*

table-cell	**Supported:** column-number, cell-before-row-margin, cell-after-row-margin, cell-before-column-margin, cell-after-column-margin, cell-before-row-border, cell-after-row-border, cell-before-column-border, cell-after-column-border **Not applicable:** line-dash **Not supported:** n-columns-spanned, n-rows-spanned, cell-row-alignment, cell-background?, background-color, background-layer, starts-row?, ends-row?, cell-crossed, line-cap, line-thickness, line-repeat, line-sep, float-out-sidelines?, float-out-marginalia?, float-out-line-numbers?
table-border	**Supported:** border-priority, border-alignment, border-present?, color, line-cap, line-thickness, line-repeat, line-sep **Not applicable:** line-dash, line-miter-limit **Not supported:** border-omit-at-break?, layer, line-join
scroll	*(for online display only (without pages); thus not applicable)*
multi-mode	*(for online display only)*
marginalia	*(for online display only)*

Appendix B: Enhancements over standard DSSSL attributes

Some features of fot2pdf are not described by the DSSSL standard, but are special to this formatter. They are either special types of usage (like the linking enhancments) or additional characteristics to the standard flow objects (like the “initial-page-count” characteristic). Note that all the additional characteristics cannot be used with standard Jade/OpenJade fot files, since any attributes declared by the dsssl script are **not** reported to the fot output generated; this is a deficiency of OpenJade which might be changed in the future. As long as that is, one needs to edit the fot file manually or feed the input into the formatter via SAX.

simple-page-sequence/initial-page-count

The characteristic called “initial-page-count” takes the values “odd”, “even” or “any”. It forces the page-sequence to start on a page with the given type (e.g. “odd” will force it to start on page 1,3,5,…, while “even” only allows for 2,4,6,8,…). The page inserted (if necessary) is a simple blank page.

[to be done: description of these…]