

Correcting page make-up problems in SGML-based documentation

Pascal Lo Re*

Contents

1	Line-breaking problems	2
1.1	Normal text	2
1.2	URLs and filenames	3
1.3	“Foreign” words	4
1.4	Menus and sub-menus	4
1.5	Final adjustments	5
1.6	Conclusion	5
2	Titles: orphaned or too long	5
3	Orphan glossary entries	6
4	Problems with the table of contents and list of figures	8
4.1	Titles or figures at the start of a page	8
4.2	Long titles	9
4.3	Modified functions	9
5	The <code>inlinegraphic</code> tag	11
6	Font selection	13
7	The file <code>manual-print.dsl</code>	14
8	The file <code>jadetex.cfg</code>	24

The documentation of the Linux-Mandrake distribution is in the first instance written in SGML. It is then converted, using `jade`, into `TEX` format. Finally, `JadeTEX` is used to compile one or other of the final output formats, PostScript or PDF.

Various problems appear in the final document, including:

- incorrect line-breaking;
- section titles are sometimes orphans, and when titles exceed one line they may start on one page and finish on the next;

*Translated by Allin Cottrell from the French report available at the linux-mandrake website. Where I'm confident that I understand the author's intent I have translated fairly freely into idiomatic English (and have made some editorial changes in the interest of brevity). Where I'm less clear on the meaning, I have stuck to a rather literal translation. I have not made any substantive editorial changes, although some sections seem to me to stand in need of this. You can find the original at <http://www.linux-mandrake.com/en/doc/project/>

- glossary entries may be orphaned;
- in the table of contents, page numbers sometimes occur by themselves at the start of a new line rather than right-aligned; and if the section is at the beginning of a page the wrong page number is given. The same goes for the list of figures.
- the `<inlinegraphic>` tag does not allow for adjusting the width of an image and does not flow the text around the image;
- there are difficulties with using alternative fonts.

We can ameliorate these problems via changes to the local style sheet, `manual-print.dsl`. This permits a more precise definition of the style on output. However, `jade` does not support the DSSSL transformation language, which allows for page make-up. But it is possible to add an extension to the conversion to \TeX . We added a new class of Flow Object: this permits the direct writing of non-formatted text, including \TeX commands, into the output file. This is the “formatting-instruction” class. In order to use it, it must be declared at the start of `manual-print.dsl`, after the `<style-specification-body>` tag. These lines are added:

```
(declare-flow-object-class formatting-instruction
"UNREGISTERED::James Clark//Flow Object Class::formatting-instruction")
```

The syntax for using the class is as follows:

```
(make formatting-instruction
 data: "the_text_to_write")
```

In the output file, the content of the data section will be written just as it appears in the input, except that to produce a backslash, “\”, in the output it must be doubled. Thus to write the \TeX function `\penalty`, one must type:

```
(make formatting-instruction
 data: "\\penalty")
```

In addition, some changes to the file `jadetex.cfg` are required, to define certain new commands, load some \LaTeX packages, and modify certain \TeX constants. This file must be treated as part of the document compilation system. (Generally, it doesn’t exist and will have to be created.)

1 Line-breaking problems

There are several sorts of problems with line breaks, depending on the style of the text.

1.1 Normal text

Certain words in the current language are not hyphenated in the body of the text, producing overly long lines. This happens, for example, with the French words “savoir” and “évolution”. Analysing the \TeX error log one sees this sort of thing:

```
Overfull \hbox (3.64668pt too wide) in paragraph at lines 249--256
\T1/ptm/m/n/11 plusieurs ressources li'vees a linux-mandrake.
Si vous souhaitez en savoir
[]
```

This shows that Jade \TeX doesn't know how to hyphenate the word "savoir". To get Jade \TeX to hyphenate correctly it's necessary to tell it to do so with the command `\def\Hyphenate{1}` (and of course the text must be justified, via the command `\def\Quadding{justify}`). Further, if you want to hyphenate French text you have to specify the right language (`\def\Language{FR}`).

If you analyse the generated \TeX file you can see that these definitions are made locally for each paragraph. Globally, there is nothing. Among other things, there is no language specification. Adding, e.g.,

```
\global\def\Language{FR}
```

in `jadetex.cfg` allows the global specification of a language. If the chosen language is English, one must add `\global\def\Language{US}`. One elegant method of getting this right is to add the language line to `jadetex.cfg` automatically via the configuration script. In effect, this defines the language used. The correspondence between languages and their abbreviations is defined in `jadetex.dtx`, the basic Jade \TeX file (see Table 1).

Further, if one uses a recent version of Jade \TeX , one must load the `babel` package. This was done automatically in Jade \TeX version 2.2, but this is no longer the case with versions 3.3 and higher. It's necessary to add the line

```
\usepackage[american,UKenglish,français,german,italian,spanish]{babel}
```

at the beginning of `jadetex.cfg`. This tells Jade \TeX to load the hyphenation patterns for the given languages (others may be added as the need arises). Then there will be no more problems hyphenating words in the current language.

Name of language for babel	Jade \TeX abbreviation
american	US
UKenglish	GB
français	FR
german	DE
italian	IT
spanish	ES

Table 1: Language abbreviations

1.2 URLs and filenames

Jade \TeX does not know how to break long internet addresses or filename paths; this too can lead to overly long lines. The basic hyphenation patterns don't help with this. To overcome this problem we use the `url` package, written by Donald Arseneau (latest version, 1.4). To make use of this it's not enough to tell Jade \TeX to load the package (via `jadetex.cfg`); we also have to transform the `<ulink>` and `<filename>` tags into Jade \TeX commands, which requires modification of `manual-print.dsl`. For the `<ulink>` tag, we want jade to first write the text between the tags then, in parentheses, the address, given by the `id` attribute.

To this end we create a `sosof` of type `formatting-instruction`. The text which serves as link is accessible using `data-of (current-node)`. However, this text may have special formatting (e.g. italics), so we have to call the function `(process-children)` in adding it. The address is found using `attribute-string (normalize "url")`. To construct the string to be written we concatenate these elements with `string-append`, and assign the result to the `data` of the `sosof`. The code is then:

```
(element ulink
  (sosof-append
    (process-children
      (make formatting-instruction
        data: (string-append
          " (\url{ (attribute-string (normalize "url")) }) "))))
```

Now there will be hardly any trouble with URLs. However, in the PostScript or PDF output the URL appears in magenta. This is because JadeT_EX uses the `hyperref` package, for which magenta is the default color for URLs. To make the URLs appear in black one must add the following line to `jadetex.cfg`:

```
\hypersetup{colorlinks=false}
```

For filenames, there's not much to be done: we just have to deal with the text between the markers. The `url` package gives us the `path` command for this purpose:

```
(element filename
  (make formatting-instruction
    data: (string-append "\\path{" (data-of (current-node)) "}"))))
```

One further change to `manual-print.dsl` is needed. The redefinition of the `<ulink>` tag (which overrides the original definition in `dbindex.dsl`) produces an unwanted side-effect, since this tag is used in the generation of the index. In that context a `<ulink>` is the child of one of the following: `<primaryie>`, `<secondaryie>` or `<tertiaryie>`. To get around this problem, we copy a section of the original code into `manual-print.dsl`, making `<ulink>` once more a child of the above-mentioned elements.

```
(element (primaryie ulink)
  (indexentry-link (current-node)))
(element (secondaryie ulink)
  (indexentry-link (current-node)))
(element (tertiaryie ulink)
  (indexentry-link (current-node)))
```

1.3 “Foreign” words

These are in general English or French acronyms, found mainly in the glossary. In the SGML file they appear within the tag `<foreignphrase>`. This time it seems the problem is related to fonts, specifically the default italic monospaced font. If we use ordinary italics there's no problem. Thus in `manual-print.dsl` we replace

```
(element foreignphrase ($italic-mono-seq$))
```

with

```
(element foreignphrase ($italic-seq$))
```

Then we do not get overfull lines.

1.4 Menus and sub-menus

When representing paths through several levels of menus, using the `<menuchoice>` tag, DSSSL places arrows between the entries. But JadeT_EX won't break a line after an arrow, which again makes for over-long lines. We have to tell JadeT_EX that it can break such lines. The basic DSSSL directives for the `<menuchoice>` tag are in `dbinline.dsl`. We copy the code from there and modify it. The original code is as follows:

```
(if (or (equal?
  (gi (node-list-first nl))
  (normalize "guimenuitem"))) (equal? (gi
  (node-list-first nl)) (normalize "guisubmenu")))
  (literal "\rightwards-arrow;")
  (literal "+"))
```

Menus are generated via the `<guimenuitem>` and `<guisubmenu>` tags; the code

```
(literal "\rightwards-arrow;")
```

inserts the arrow. We have to replace this line and the next with

```
(make formatting-instruction data:
 "$\rightarrow$\penalty-100")
 (literal "+")
```

The command `\penalty-100` tells \TeX that it can start a new line at that point. The rest of the code is unchanged. The arrow is now produced by the \LaTeX command `\rightarrow`; since this is a math symbol it must be placed between $\$$ s.

1.5 Final adjustments

As indicated above, some problems remain, notably with URLs containing long words that cannot be broken. To help with this we reset \TeX 's `\tolerance` parameter to give the program more leeway in word spacing. We have to find a compromise between spacing and over-long lines, so as to have a document that is as aesthetically pleasing as possible. We modify `jadetex.cfg`, setting

```
\tolerance=2000
```

in place of \TeX 's default value of 200.

1.6 Conclusion

After these changes, hardly any line-breaking problems remain. Some URLs are still too long, given the chosen tolerance, but not many.

Another problem remains: in the glossary, for the term CHAP, the French word "Authentification" is not hyphenated. We have to specify manually that this word may be broken. I don't understand why, since $\text{Jade}\TeX$ knows how to break it when it's in English.

2 Titles: orphaned or too long

Certain titles appear alone at the foot of the page (orphans) or, if they are too long, they begin at the foot of one page and finish at the top of the next. To correct this problem it's necessary to modify a $\text{Jade}\TeX$ function, namely `\QueryPageBreak`. Between versions 2.2 and 3.3, Sebastian Rahtz improved matters, in particular by adding `\penalty-600` to advise $\text{Jade}\TeX$ to break pages before titles. But since there are still cases of orphan titles, a stronger negative penalty is needed. In `jadetex.cfg` we write

```
\def\QueryPageBreak{%
\ifBreakMe
\ifvmode
\penalty \@M
\else
\@bsphack
\vadjust{\penalty \@M}%
\@esphack
\fi
\else
\ifnum\KeepWithNext=1
\penalty-5000% Old : -600
\else
\penalty \@z
```

```

\fi
\fi
}

```

In replacing `-600` with `-5000`, we're telling Jade \TeX that this is a very good place to insert a page break, which helps with the problem of orphan and over-long titles.

However, there are still titles that begin at the foot of one page and finish at the top of the next. To fix this we modify two more \TeX constants. The first, `clubpenalty`, is the penalty that applies when a line that begins a paragraph occurs at the foot of a page. By increasing this penalty we reduce the chance of getting such lines. The second, `widowpenalty`, concerns the last line of paragraphs: a higher value reduces the chance of the last line of a paragraph occurring at the top of a page. `jadetex.cfg` defines by default

```

\clubpenalty=4000
\widowpenalty=2000

```

If we replace these values with `5000`, there are no more problems. In `jadetex.cfg` we write:

```

\clubpenalty=5000
\widowpenalty=5000

```

3 Orphan glossary entries

In the glossary, certain entries are placed at the foot of the page, even though the corresponding text appears at the top of the next page. This problem persists despite the changes made for titles. To fix it, we create a glossary entry in \LaTeX . For each `<glossterm>` we create a \LaTeX environment called `Glossentry`.

To do this, we need to add several lines to `manual-print.dsl`. First, we need to tell jade when to create the environment: we open the environment when we encounter a `<glossterm>` tag. The text enclosed by the tags is the glossary entry; it becomes the parameter to a \LaTeX `\item`; we obtain it using `data-of`. The definition comes after the `<glossdef>` tag. Since we need to take formatting into account we use `process-node-list`. Once the children of `<glossterm>` are taken care of, we only have to close the environment and insert a skip to separate the items. The function we use is recursive, so as to manage all the descendants of `<glossterm>`. We have the following code:

```

(element glossentry
  (sosofo-append
    (process-gloss (children (current-node)))
    (make formatting-instruction
      data: (string-append "\\end{Glossentry}\\bigskip")
    )
  )
)

(define (process-gloss ndl)
  (if (node-list-empty? ndl)
    (empty-sosofo)
    (sosofo-append
      (cond
        ((string=? (gi (node-list-first ndl)) (normalize "glossterm"))
          (make formatting-instruction
            data: (string-append "\\begin{Glossentry}\\item[" (data-of
              (node-list-first ndl)) "]" )))
        ((string=? (gi (node-list-first ndl)) (normalize "glossdef"))

```

```

        (process-node-list (children (node-list-first ndl)))
    )(process-gloss (node-list-rest ndl))
)
)
)

```

However, the effect is that jade doesn't create a paragraph. In fact, the manner in which they are defined produces an error in the environment. Thus when we encounter the tag <para>, we just treat its descendants, without making the paragraph. The page make-up is done by the environment. Therefore the following code is needed in manual-print.dsl:

```

(element (glossdef para)
  (sosofo-append
    (process-children)
    (make formatting-instruction
      data: "\\par")
  )
)

```

The same problem arises with the tags <glossseealso>, requiring this addition:

```

(element glossseealso
  (if (first-sibling?)
    (sosofo-append ;; Old : make paragraph
      ($italic-seq$ (literal (gentext-element-name (current-node))
        (gentext-label-title-sep (gi))))
    (with-mode glossseealso
      (process-node-list
        (select-elements (children (parent)) '(glossseealso)))
      (literal ".")
      (empty-sosofo))
  )
)

```

It only remains to define the environment in jadetex.cfg. To keep the original pagination we have to use the packages ifthen and calc. We load them with these lines:

```

\usepackage{ifthen}
\usepackage{calc}

```

The environment code itself is:

```

\newlength{\tailleEntree}
\newcommand{\styleEntree}[1]{%
\settowidth{\tailleEntree}{\textbf{\textit{#1}}}%
\ifthenelse{\lengthtest{\tailleEntree > \labelwidth}}%
  {\parbox[b]{\labelwidth}%
  {\makebox[0pt][l]{\textbf{\textit{#1}}}\}}%
  {\parbox[b]{\labelwidth}%
  {\textbf{\textit{#1}}}\}}%
}

```

```

\newenvironment{Glossentry}
{\begin{list}{}
{\renewcommand{\makelabel}{\styleEntree}
\setlength{\labelwidth}{0.8cm}
\setlength{\labelsep}{0cm}
\setlength{\leftmargin}{\labelwidth + \labelsep}
\setlength{\listparindent}{0cm}
}}{\end{list}}

```

4 Problems with the table of contents and list of figures

The DSSSL instructions for the table of contents and list of figures are found in `dbautoc.dsl`. The addition of an entry to the table of contents is done by the function

```
($toc-entry$ tocentry level)
```

For the list of figures, it's the function

```
($loc-entry$ tocentry)
```

4.1 Titles or figures at the start of a page

For titles or figures occurring at the top of a page, the number appearing in the table of contents is that of the previous page; the hyperlink in the table of contents also points to the previous page. This is due to the fact that the target of the link is `<sect*>` or `<figure>`. `jade` translates this into \TeX by creating a `\Node`. It then creates another `\Node` for the title, corresponding to the `<title>` tag. When the document is compiled, a page break may be inserted between these two `\Nodes`. Because `<sect*>` is used to make the link and to define the page number, the table of contents does not refer to the page where the title is located. The correction involves telling `jade` to make the link point to `<title>`. This tag is a descendant of `<sect*>` or `<figure>`. According to the DocBook DTD `<sect*>` and `<figure>` accept only one descendant, `<title>`.

The links are made by the code

```
(make link
  destination: (node-list-address tocentry))
```

and the page number is retrieved with

```
(with-mode toc-page-number-mode
  (process-node-list tocentry))
```

To get access to the `<title>` tag, we have to replace `tocentry` with

```
(node-list-first (select-elements (children tocentry)(normalize
  "title")))
```

which corresponds to the first `<title>` tag which is a descendant of the `<sect*>` or `<figure>` tag under consideration. This works because there is only one `<title>` in each such element. We therefore get:

```
(make link
  destination: (node-list-address (node-list-first (select-elements
    (children tocentry)(normalize "title") ))))
```

and

```
(with-mode toc-page-number-mode
  (process-node-list (node-list-first (select-elements (children
    tocentry)(normalize "title") ))))
```

We then only have to tell JadeTeX to write the page number of these elements into the .aux file, in order to be able to add them. To do this we put the following code into jadetex.cfg:

```
\makeatletter
\def\endNode#1{%
  \FlowObjectSetup{1}%
  \let\Label\@empty\let\Element\@empty%
}\makeatother
```

The call to the function `\FlowObjectSetup{1}` when a Node is completed writes out the page on which the Node is located.

4.2 Long titles

For some long titles, the page number in the table of contents is placed on a new line. Moreover, it appears at the start of the line rather than being right-aligned along with the other numbers. Normally we get a dot leader following the title. This is generated by the JadeTeX command `\Leader`, which calls the TeX command `\hfill`. This command fills the current line between the cursor position (*position de curseur*) and the right margin. In the pathological case we are examining this element is created alright, but has zero length. JadeTeX therefore puts it on a new line, with the effect that the page number appears at the start of the new line.

To fix this, we prohibit a line break before the leader. In the functions (`$toc-entry$ tocentry level`) and (`$loc-entry$ tocentry`) you can see the line

```
(make leader (literal "."))
```

which generates the leader separating the end of the title and the page number. We replace this line with

```
(make formatting-instruction
  data: "\\penalty10000\\dotfill")
```

where the penalty of 10000 stops JadeTeX from breaking the line. Since it can no longer put a zero-length `\hfill` on a new line, it puts the last word of the title on a new line and inserts a dot leader to the page number.

4.3 Modified functions

Based on the discussion above, we modify the following two functions in `manual-print.dsl`:

```
(define ($toc-entry$ tocentry level)
  (make paragraph
    start-indent: (+ %body-start-indent%
      (* %toc-indent% level))
    first-line-start-indent: (* -1 %toc-indent%)
    font-weight: (if (= level 1) 'bold 'medium)
    space-before: (if (= level 1) (* %toc-spacing-factor% 6pt) 0pt)
    space-after: (if (= level 1) (* %toc-spacing-factor% 6pt) 0pt)
    quadding: 'start
    (make link
      destination: (node-list-address (node-list-first (select-elements
        (children tocentry)(normalize "title") ))))
    (make sequence
```

```

        (if (equal? (element-label tocentry) "")
            (empty-sosofo)
            (make sequence
                (element-label-sosofo tocentry)
                (literal (gentext-label-title-sep (gi tocentry))))))
        (element-title-sosofo tocentry)))
    (make formatting-instruction data: "\\penalty10000\\dotfill")
    (make link
        destination: (node-list-address (node-list-first (select-elements
            (children tocentry)(normalize "title") )))
        (with-mode toc-page-number-mode
            (process-node-list (node-list-first (select-elements
                (children tocentry)(normalize "title") )))))
    )))

(define ($lot-entry$ tocentry)
    (make paragraph
        start-indent: (+ %body-start-indent% %toc-indent%)
        first-line-start-indent: (* -1 %toc-indent%)
        font-weight: 'medium
        space-before: 0pt
        space-after: 0pt
        quadding: 'start
        (make link
            destination: (node-list-address (node-list-first (select-elements
                (children tocentry)(normalize "title") )))
            (make sequence
                (if (equal? (element-label tocentry) "")
                    (empty-sosofo)
                    (make sequence
                        (element-label-sosofo tocentry #t)
                        (literal (gentext-label-title-sep (gi tocentry))))))
                (element-title-sosofo tocentry)))
            (make formatting-instruction
                data: "\\penalty10000\\dotfill")
            (make link
                destination: (node-list-address (node-list-first (select-elements
                    (children tocentry)(normalize "title") )))
                (make sequence
                    (if %page-number-restart%
                        (make sequence
                            (literal (substring (element-label tocentry #t)
                                0 (string-index (element-label tocentry #t) "-")))
                            (literal (gentext-intra-label-sep "_pagenumber")))
                        (empty-sosofo))
                    (with-mode toc-page-number-mode
                        (process-node-list (node-list-first (select-elements
                            (children tocentry) (normalize "title") )))))
            )))
    )))

```

5 The inlinegraphic tag

The relevant code is found in `dbgraph.dsl`. We have to modify two functions. The first is `img`. This serves to pass the various attributes as parameters to the second function, `$graphic$`, which inserts the image. To take the width attribute into account we add a function `$InlineImg$` in `manual-print.dsl`. This code of the latter is copied from `img`, to which we add handling of the width attribute. First, at the level of the assignment of the variables (`let*`), we create a variable `width`:

```
(width (attribute-string (normalize "width") nd))
```

Then, when calling the function that inserts the image, we must pass this parameter. The insertion function we use is called `$InlineGraphic$`. The two lines:

```
($graphic$ fileref display format scale align)
($graphic$ (entity-generated-system-id entityref)
            display format scale align)
```

then become:

```
($InlineGraphic$ fileref display format scale align width)
($InlineGraphic$ (entity-generated-system-id entityref)
                 display format scale align width)
```

We must now write the function `$InlineGraphic$`, and ensure that it takes into account the width of the image. We can't use the Flow-Object `external-graphic`, because it has no provision for specifying a width. We will encapsulate it in a `\scalebox`, which uses a scale factor. The whole thing is put in a `wrapfigure` environment, to allow the text to flow around the image, which must be placed on the left or right of the page (it can't go in the middle). The alignment is passed via the `align` parameter (if this has the value `center` we put the image on the left). We also have to take into account the eventual scale factor for the *taille* of the box created by the `wrapfigure` environment. For this, we use a variable whose value is the width of the image multiplied by the scale factor. A DSSSL variable is added. It is one unit by default (the `width` parameter can be a simple number). We have chosen the centimeter. [C'est l'unité par défaut (le paramètre `width` peut être un simple chiffre). Nous avons choisi le centimètre.]

The code for this function is given below. The first lines define the variables. There are many tests because some of these are optional. There follows a series of calculations concerning widths. The function

```
(length-string-unit-part str)
```

retrieves the unit given in the string `str`. The function `(number->string int)` converts a number into a string, and the function `(string->number str)` performs the inverse operation.

Finally, we have to arrange that when an `<inlinegraphic>` tag is encountered the redefined functions are used. To that end we add to `manual-print.dsl` the following line:

```
(element inlinegraphic ($InlineImg$))
```

The whole function is:

```
(define %defaultUnit% "cm")

(define ($InlineGraphic$ fileref
        #!optional (display #f) (format #f) (scale #f) (align #f)
        (width #f))
  (let ((unit (if width
                  (if (string=? (length-string-unit-part width) "")
                      %defaultUnit% (length-string-unit-part width))
```

```

                                %defaultUnit%
                                )))
    (let ((graphic-format (if format format ""))
          (graphic-scale (if scale (number->string (/ (string->number scale) 100))
                              "1")))
      (graphic-align (cond ((equal? align (normalize "center"))
                            "l")
                           ((equal? align (normalize "right"))
                            "r")
                           (else
                            "l"))))
        (box-width          (if width
                                (if scale (string-append (number->string
                                                           (* (string->number (length-string-number-part width)) (/ (string->number
                                                                 scale) 100))) unit) (string-append (length-string-number-part width) unit))
                                (if scale (string-append (number->string
                                                           (/ (string->number scale) 100)) unit) (string-append "1" unit))
                                ))
          (graphic-width (if width (string-append (length-string-number-part
                                                    width) unit) (string-append "1" unit))))
        (make formatting-instruction
          data: (string-append "\\begin{wrapfigure}{ " graphic-align "}{ "
                                box-width " }\\scalebox{ " graphic-scale " }{\\includegraphics[width="
                                graphic-width ",keepaspectratio=true]{ " (graphic-file fileref) " } }
                                \\end{wrapfigure}")
          )
        )
    )
)

(define ($InlineImg$ #!optional (nd (current-node)) (display #f))
  (let* ((fileref (attribute-string (normalize "fileref") nd))
         (entityref (attribute-string (normalize "entityref") nd))
         (format (if (attribute-string (normalize "format") nd)
                     (attribute-string (normalize "format") nd)
                     (if entityref
                         (entity-notation entityref)
                         #f)))
         (align (attribute-string (normalize "align") nd))
         (scale (attribute-string (normalize "scale") nd))
         (width (attribute-string (normalize "width") nd)))
    (if (or fileref entityref)
        (if (equal? format (normalize "linespecific"))
            (if fileref
                (include-file fileref)
                (include-file (entity-generated-system-id entityref)))
            ($InlineGraphic$ fileref display format scale align width)
            ($InlineGraphic$ (entity-generated-system-id entityref)
                              display format scale align width)))
        (empty-sosofo)))

```

(element inlinegraphic ($\$InlineImg\$\$$))

We now just have to load the package using `\usepackage{wrapfig}` in `jadetex.cfg`.

In addition we can regulate the margin around the image, with the line `\setlength{\columnsep}{0.5cm}` in `jadetex.cfg`. In this case the margin is 0.5cm, which is acceptable. But this value can be altered. Now the text flows around the image. But we have to take care not to place this tag in the middle of a phrase, otherwise the results are not pleasant: a line break is made and very wide spaces appear between words.

6 Font selection

There are six DSSSL variables to define the fonts to be used depending on the type of text:

- `%title-font-family%`: the font used for text treated as titles. Examples: titles, glossary entries (by default).
- `%admon-font-family%`: font for formatting notes and warnings. Example: the `<note>` tag.
- `%guilabel-font-family%`: font for representation of a GUI. Example: paths within menus (`<guimenuitem>`).
- `%mono-font-family%`: monospaced font. Examples: file names, commands, URLs.
- `%refentry-font-family%`: font used for references.
- `%body-font-family%`: font used for the body of the text.

These variables can, by default, take one of the following values

Helvetica
Palatino
Bookman
Courier
Wingdings
Avant-Garde
New-Century-Schoolbook
Times-Roman
Zapf-Dingbats
Computer-Modern-Typewriter
Computer-Modern-Sans
Computer-Modern
Computer-Modern-Caps-And-Small-Caps

Other names can be used, but in fact they correspond to one or other of the fonts given above. We have for instance:

Arial = iso-sanserif = Helvetica
Courier-New = Courier
Times-New-Roman = iso-serif = Times-Roman
WingDings = Wingdings

To use other fonts, they must be in T1 encoding (the coding used by $\text{T}_{\text{E}}\text{X}$). One must find a file on the pattern `t1***.fd` where `***` corresponds to the codename of the font (the first letter representing the foundry, the next two the name of the font). Moreover, all other necessary font files must be present, namely the `.tfm`, `.afm`, `.vf`,

.pfa and .pfb files. Eventually .pk files may be needed too. The names of all these files begin with the three characters found in t1***.fd.

Then one must ensure that JadeTeX can associate the symbolic name of the font (e.g. Utopia) with its code-name (e.g. put). This is done by adding to jadetex.cfg a line of the form:

```
\def\Family@nom_symbolique{***}
```

For instance

```
\def\Family@Utopia{put}
```

On the compilation system, several other fonts are available. For some of them, we have not found the symbolic name — these we have defined as Font***. Here is the list to be placed in jadetex.cfg to use the extra fonts.

```
\makeatletter
\def\Family@Utopia{put}
\def\Family@ZapfChancery{pzc}
\def\Family@Fibonacci{cmfib}
\def\Family@Funny{cmfr}
\def\Family@Dunhill{cmdh}
\def\Family@Concrete{ccr}
\def\Family@Charter{bch}

\def\Family@Fontpxr{pxr}
\def\Family@Fontaer{aer}
\def\Family@Fontaess{aess}
\def\Family@Fontaett{aett}
\def\Family@Fontlcmss{lcms}
\def\Family@Fontlcmstt{lcmstt}
\def\Family@Fontcmvtt{cmvtt}
\def\Family@Fontcmbr{cmbr}
\def\Family@Fontcmtl{cmtl}
\def\Family@Fontpxss{pxss}
\def\Family@Fontpxtt{pxtt}
\def\Family@Fonttxss{txss}
\def\Family@Fonttxtt{txtt}
\def\Family@Fonttxr{txr}
\makeatother
```

The foregoing declarations must be preceded by \makeatletter and followed by \makeatother since the functions use the character as a normal letter. Then, if one wants for example to put titles into the font “Fonttxr” and body text into “Concrete”, it’s enough to put the following two lines into manual-print.dsl:

```
(define %title-font-family% "Concrete")
(define %body-font-family% "Fonttxr")
```

7 The file manual-print.dsl

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN" [
<!ENTITY % words
PUBLIC "-//Norman Walsh//ENTITIES DocBook Stylesheet Localization//XX"
```

```

"local_words.ent">

%words;

<!ENTITY docbook.dsl SYSTEM "dsssl/print/docbook.dsl" CDATA DSSSL >

<!ENTITY mdk-de SYSTEM "mdk-de.dsl">
<!ENTITY mdk-en SYSTEM "mdk-en.dsl">
<!ENTITY mdk-es SYSTEM "mdk-es.dsl">
<!ENTITY mdk-fr SYSTEM "mdk-fr.dsl">
<!ENTITY mdk-it SYSTEM "mdk-it.dsl">

]>

<style-sheet>
<style-specification id="print" use="docbook">
<style-specification-body>

;; Include the flow object class "formatting-instruction" : ONLY for Jade
(declare-flow-object-class formatting-instruction
 "UNREGISTERED::James Clark//Flow Object Class::formatting-instruction")

(define default-backend 'tex)

(define %generate-partintro-on-titlepage% #f)

(define %bf-size% 11pt)

(define %verbatim-size-factor% 1)

(define %paper-type%
 ;; Name of paper type
 ;; "A4"
 "A5")

(define %page-width% 148mm)
(define %page-height% 210mm)

(define %left-margin% 20mm)
(define %right-margin% 20mm)
(define %top-margin% 10mm)
(define %bottom-margin% 15mm)

(define %header-margin% 10mm)
(define %footer-margin% 10mm)

(define %body-start-indent% 0pi)
(define %visual-acuity% "normal")
(define %hsize-bump-factor% 1.1)
(define %default-quadding% 'justify)

```

```

(define %two-side% #t)

(define %line-spacing-factor%
  ;; Factor used to calculate leading
  1.2)

;; number chapters and sections

(define %chapter-autolabel% #t)
(define %section-autolabel% #t)

;; control TOC depth
(define (toc-depth nd)
  (if (string=? (gi nd) (normalize "book"))
      3      ;;change this to what you want and add to your custom driver
      (if (string=? (gi nd) (normalize "appendix"))
          01)))
(declare-characteristic preserve-sdata?
  "UNREGISTERED::James Clark//Characteristic::preserve-sdata?"
  #f)

(define %mono-font-family% "Computer-Modern-Typewriter")

(element glossterm ($bold-italic-seq$))

;; New glossary to avoid orphan entries
;; It use a special LaTeX environment
;; Original : dbgloss.dsl
(element glossentry
  (sosofo-append
    (process-gloss (children (current-node)))
    (make formatting-instruction ;;Close the nevironment and add a little jump
      data: (string-append "\\end{Glossentry}\\bigskip")
    )
  )
)
);; This function writes the LaTeX tags to create the environment
(define (process-gloss ndl)
  (if (node-list-empty? ndl)
      (empty-sosofo)
      (sosofo-append
        (cond
          ((string=? (gi (node-list-first ndl)) (normalize "glossterm"))
            (make formatting-instruction ;;Open the environment and write the item
              data: (string-append "\\begin{Glossentry}\\item[" (data-of
(node-list-first ndl)) "]" )
            )
          )
        ))
      (string=? (gi (node-list-first ndl)) (normalize "glossdef"))
    )
  );;Writes the content of the entry
  (process-node-list (children (node-list-first ndl)))
)
)

```

```

)

;;Recursiv : process the rest of the FOT
      (process-gloss (node-list-rest ndl))
    )
  )
)

;; For paragraphs of the definition, Jade mustn't put \endPar
;; because it generate problems with the environment used.
;; When metting the "para" tag, we just process the children.
;; The layout is defined in the environment
(element (glossdef para)
  (sosofo-append
    (process-children)
    (make formatting-instruction      ;;End of the paragraph
      data: "\\par")
  )
)

;; Redefinition to avoid problems
;; Originaly, it create a paragraph, which leads to problems (cf above)
(element glosseealso
  (if (first-sibling?)
    (sosofo-append;make paragraph
      ($italic-seq$ (literal (gentext-element-name (current-node))
        (gentext-label-title-sep (gi))))
    (with-mode glosseealso
      (process-node-list
        (select-elements (children (parent)) '(glosseealso))))
    (literal ".")
    (empty-sosofo)))

;; Use of an italic font for 'foreignphrase'
;; => no more hyphenation problems with long sentences
;; Old definition : (element foreignphrase ($italic-mono-seq$))
(element foreignphrase ($italic-seq$))
(element emphasis ($bold-seq$))
(element command ($mono-seq$))
(element option ($mono-seq$))
(element literal ($mono-seq$))
(element systemitem ($italic-mono-seq$))
(element hardware ($italic-mono-seq$))
(element application ($italic-mono-seq$))

;; footnotes at the bottom of each page (and not chapter)
(define bop-footnotes #t)

;; create a general TOC?
(define %generate-book-toc% #t)

```

```

;; *** URLs ***
;; Use of the "url" package for LaTeX
;; Original : dblink.dsl
(element ulink
  (sosofo-append
    (process-children)          ;; Wrote the text with its format (anchor in HTML)
    (make formatting-instruction  ;; Wrote : " (\url{" + theUrl + "}")"
      data: (string-append " (\\url{" (attribute-string (normalize "url")) "}")")
    )
  )
)

;; *** Filenames ***
;; Use of the "url" package for LaTeX : 'path' command
;; Original : dbinline.dsl
(element filename
  (make formatting-instruction  ;; Wrote : " \path{" + theFilename + "}"
    data: (string-append "\\path{" (data-of (current-node)) "}")
  )
)
);; Must use "(data-of (current-node))" instead of "(process-children)"

;; These three elements are from "dbindex.dsl"
;; Must be placed here because of the redefinition of "ulink"
(element (primaryie ulink)
  (indexentry-link (current-node)))

(element (secondaryie ulink)
  (indexentry-link (current-node)))

(element (tertiaryie ulink)
  (indexentry-link (current-node)))

;; redefine screen and programlisting definition so that they can
;; use a different size factor from other verbatim environments

(define %screen-size-factor% 0.7)

(define ($screen-verbatim-display$ indent line-numbers?)
  (let* ((width-in-chars (if (attribute-string (normalize "width"))
    (string->number (attribute-string (normalize
"width"))))
    %verbatim-default-width%))
    (fsize (lambda () (if (or (attribute-string (normalize "width"))
    (not %screen-size-factor%))
    (/ (/ (- %text-width% (inherited-start-indent))
    width-in-chars)
    0.7)
    (* (inherited-font-size)
    %screen-size-factor%))))))

```

```

        (vspace (if (INBLOCK?)
                    Opt
                    (if (INLIST?)
                        %para-sep%
                        %block-sep%))))
(make paragraph
 use: verbatim-style
 keep: #t
 space-before: (if (and (string=? (gi (parent)) (normalize "entry"))
                     (absolute-first-sibling?))
                 Opt
                 vspace)
 space-after: (if (and (string=? (gi (parent)) (normalize "entry"))
                      (absolute-last-sibling?))
                 Opt
                 vspace)
 font-size: (fsize)
 line-spacing: (* (fsize) %line-spacing-factor%)
 start-indent: (if (INBLOCK?)
                  (inherited-start-indent)
                  (+ %block-start-indent% (inherited-start-indent)))
 (if line-numbers?
     ($linespecific-line-by-line$ indent line-numbers?)
     (process-children))))

(element screen          ($screen-verbatim-display$
                        %indent-screen-lines%
                        %number-screen-lines%))

(element programlisting ($screen-verbatim-display$
                       %indent-screen-lines%
                       %number-screen-lines%))

(define ($object-titles-after$)
;; REFENTRY object-titles-after
;; PURP List of objects who's titles go after the object
;; DESC
;; Titles of formal objects (Figures, Equations, Tables, etc.)
;; in this list will be placed below the object instead of above it.
;;
;; This is a list of element names, for example:
;; '(list (normalize "figure") (normalize "table"))'.
;; /DESC
;; AUTHOR N/A
;; /REFENTRY
(list (normalize "figure") (normalize "table")))

;; *** Menu ***
;; Redefine menu, so that they can be linebreaks after an arrow
;; Original : dbinline.dsl
(element menuchoice

```

```

        (let* ((shortcut (select-elements (children (current-node))
            (normalize "shortcut"))))
          (items
            (node-list-filter-by-not-gi
              (children (current-node))
              (list (normalize "shortcut")))))
            (make sequence
              (let loop ((nl items) (first? #t))
                (if (node-list-empty? nl)
                    (empty-sosofo)
                    (make sequence
                      (if first?
                          (process-node-list (node-list-first nl))
                          (make sequence
                            (if (or (equal? (gi (node-list-first nl))
                                (normalize "guimenuitem"))
                                (equal? (gi (node-list-first nl))
                                    (normalize "guisubmenu"))))
                              (make formatting-instruction
                                data: "$\rightarrow$\penalty-100")
                                ;; Tell TeX that it can break the line after the arrow
                                (literal "+")
                                )(process-node-list (node-list-first nl))))
                                (loop (node-list-rest nl) #f))))
                      (if (node-list-empty? shortcut)
                          (empty-sosofo)
                          (make sequence
                            (literal " (")
                            (process-node-list shortcut)
                            (literal ")")
                            )
                          )
                      )
                    )
                )
              )
            )
          )
        )

;; *** TOC ***
;; Redefine TOC entry so that their is no longer page number
;; alone on a new line : forbid linebreak before eventual dotfill
;; and so that there is no more error with page number at the
;; bottom of a new page
;; Original : dbautoc.dsl
(define ($toc-entry$ tocentry level)
  (make paragraph
    start-indent: (+ %body-start-indent%
      (* %toc-indent% level))
    first-line-start-indent: (* -1 %toc-indent%)
    font-weight: (if (= level 1) 'bold 'medium)
    space-before: (if (= level 1) (* %toc-spacing-factor% 6pt) 0pt)
    space-after: (if (= level 1) (* %toc-spacing-factor% 6pt) 0pt)
    quadding: 'start
    (make link

```

```

        destination: (node-list-address (node-list-first (select-elements
(children tocentry)(normalize "title") )))
;; The anchor for the link when clicking on the title
;; Original version : (node-list-address tocentry)
;;[it corresponds to the "sect*" tag]
    (make sequence
(if (equal? (element-label tocentry) "")
    (empty-sosofo)
    (make sequence
    (element-label-sosofo tocentry)
    (literal (gentext-label-title-sep (gi tocentry))))))
    (element-title-sosofo tocentry)))
    (make formatting-instruction
    data: "\\penalty10000\\dotfill")
;; Forbid linebreak before dotfill, so before the page number in TOC
;; Original version : (make leader (literal "."))
    (make link
        destination: (node-list-address (node-list-first (select-elements
(children tocentry)(normalize "title") )))
;; The anchor for the link when clicking on the page number
;; Original version : (node-list-address tocentry)
        (with-mode toc-page-number-mode
(process-node-list (node-list-first (select-elements (children tocentry)
(normalize "title") ))))
;; Display the page number of the 'title' tag, and not the "sect*" tag
;; Original version : (process-node-list tocentry))
)))

;; *** LOF ***
;; Redefine LOF entry so that their is no longer page number
;; alone on a new line : forbid linebreak before eventual dotfill
;; and so that there is no more error with page number at the
;; bottom of a new page
;; (unfortunatly it doesn't always work)
;; Original : dbautoc.dsl
(define ($lot-entry$ tocentry)
    (make paragraph
        start-indent: (+ %body-start-indent% %toc-indent%)
        first-line-start-indent: (* -1 %toc-indent%)
        font-weight: 'medium
        space-before: Opt
        space-after: Opt
        quadding: 'start
        (make link
            destination: (node-list-address (node-list-first (select-elements
(children tocentry)(normalize "title") )))
;; The anchor for the link when clicking on the title
;; Original version : (node-list-address tocentry)
;;[it corresponds to the "figure" or "table" tag]
            (make sequence
(if (equal? (element-label tocentry) "")

```

```

    (empty-sosof)
    (make sequence
      (element-label-sosof tocentry #t)
      (literal (gentext-label-title-sep (gi tocentry))))))
(element-title-sosof tocentry)))
  (make formatting-instruction
    data: "\\penalty10000\\dotfill")
;; Forbid linebreak before dotfill, so before the page number in LOT/LOF
;; Original version : (make leader (literal "."))
  (make link
    destination: (node-list-address (node-list-first (select-elements
(children tocentry)(normalize "title") )))
;; The anchor for the link when clicking on the page number
;; Original version : (node-list-address tocentry)
    (make sequence
(if %page-number-restart%
  (make sequence
    (literal (substring (element-label tocentry #t)
0 (string-index (element-label tocentry #t) "-")))
    (literal (gentext-intra-label-sep "_pagenumber"))))
  (empty-sosof))
(with-mode toc-page-number-mode
  (process-node-list (node-list-first (select-elements (children tocentry)
(normalize "title") ))))
;; Display the page number of the 'title' tag, and not the "figure"
;; or "table" tag
;; Original version : (process-node-list tocentry))
))))

;; *** Inline Graphics ***
;; Redefine the function calles by inlinegraphic so that text can be
;; wrapped around
;; It also permit to use the 'width' attribute of the "inlinegraphic" tag
;; Original : dbgraph.dsl
;;
;; Default unit (if no other sspecified)
(define %defaultUnit% "cm")

;; This function that produce the LaTeX output
;; It uses the 'wrapfigure' environment.
;; There is a few calculs for making a box which size is scale*image width
(define ($InlineGraphic$ fileref
  #!optional (display #f) (format #f) (scale #f) (align #f) (width #f))
  (let ((unit (if width
    (if (string=? (length-string-unit-part width) "")
%defaultUnit% (length-string-unit-part width))
    %defaultUnit%
  )))
    (let ((graphic-format (if format format ""))
      (graphic-scale (if scale (number->string (/ (string->number scale) 100))
"1"))

```

```

(graphic-align (cond ((equal? align (normalize "center"))
                    "l")
                    ((equal? align (normalize "right"))
                     "r")
                    (else
                     "l"))))
  (box-width          (if width
                        (if scale (string-append (number->string
(* (string->number (length-string-number-part width))
(/ (string->number scale) 100))) unit) (string-append
(length-string-number-part width) unit))
                      (if scale (string-append (number->string
(/ (string->number scale) 100)) unit) (string-append "1" unit))
                      ))
  (graphic-width (if width (string-append
(length-string-number-part width) unit) (string-append "1" unit))))
  (make formatting-instruction
   data: (string-append "\\begin{wrapfigure}{ " graphic-align " }
{" box-width " }\\scalebox{" graphic-scale "}{\\includegraphics[width="
graphic-width ",keepaspectratio=true]{ (graphic-file fileref) "}}
\\end{wrapfigure}")
  )
)
)
)
)

```

```

;; This function is preparing the call of $InlineGraphic$
;; Only one change : attribute 'width' added
(define ($InlineImg$ #!optional (nd (current-node)) (display #f))
  (let* ((fileref (attribute-string (normalize "fileref") nd))
         (entityref (attribute-string (normalize "entityref") nd))
         (format (if (attribute-string (normalize "format") nd)
                     (attribute-string (normalize "format") nd)
                     (if entityref
                         (entity-notation entityref)
                         #f))))
         (align (attribute-string (normalize "align") nd))
         (scale (attribute-string (normalize "scale") nd))
         (width (attribute-string (normalize "width") nd)))
    (if (or fileref entityref)
        (if (equal? format (normalize "linespecific"))
            (if fileref
                (include-file fileref)
                (include-file (entity-generated-system-id entityref)))
            (if fileref
                ($InlineGraphic$ fileref display format scale align width)
                ($InlineGraphic$ (entity-generated-system-id entityref)
                display format scale align width)))
        (empty-sosofo)))
  (element inlinegraphic ($InlineImg$))
)

```

```

&mdk-de;
&mdk-en;
&mdk-es;
&mdk-fr;
&mdk-it;

</style-specification-body>
</style-specification>

<external-specification id="docbook" document="docbook.dsl">

</style-sheet>

```

8 The file jadetex.cfg

```

% This file is used to define JadeTeX modification
%
% 2001/06/25

% Load package Babel => JadeTeX loads hyphenations patterns for french
\usepackage[american,UKenglish,francais,german,italian,spanish]{babel}

% Include LaTeX package 'url'
\usepackage{url}

% These packages are used in the Glossary environment
\usepackage{ifthen}
\usepackage{calc}

% These package is used for wrapping text around graphics
\usepackage{wrapfig}
% Define the space between the text and the image
\setlength{\columnsep}{0.5cm}

% Change the penalty, to avoid orphan title
\makeatletter
\def\QueryPageBreak{%
\ifBreakMe
\ifvmode
\penalty\@M
\else
\@bsphack
\vdjust{\penalty\@M}%
\@esphack
\fi
\else

```

```

\ifnum\KeepWithNext=1
  \penalty-5000% Old : -600
\else
  \penalty\z@
\fi
\fi
}
\makeatother

% Change these penalty to avoid long title to be split between two pages
\clubpenalty=5000 % Old : 4000
\widowpenalty=5000 % Old : 2000

% Change the tolerance, to make hyphens of URLs easier
\tolerance=2000

% URLs color is black
\hypersetup{colorlinks=false}

% Redefine this command so that JadeTeX write TOC info in the .aux file
\makeatletter
\def\endNode#1{%
  \FlowObjectSetup{1}%
  \let\Label\@empty\let\Element\@empty%\noindent%
}
\makeatother

% Define an environment for the glossary
\newlength{\tailleEntree}
\newcommand{\styleEntree}[1]{%
\settowidth{\tailleEntree}{\textbf{\textit{#1}}}%
\ifthenelse{\lengthtest{\tailleEntree > \labelwidth}}%
  {\parbox[b]{\labelwidth}%
   {\makebox[Opt][l]{\textbf{\textit{#1}}}\}}%
  {\parbox[b]{\labelwidth}%
   {\textbf{\textit{#1}}}\}}%
}

\newenvironment{Glossentry}
{
\begin{list}{}
{
\renewcommand{\makelabel}{\styleEntree}
\setlength{\labelwidth}{0.8cm}
\setlength{\labelsep}{0cm}
\setlength{\leftmargin}{\labelwidth + \labelsep}
\setlength{\listparindent}{0cm}
}
}
{\end{list}}

```

```

% Define new aliases for Fonts installed on the system
\makeatletter
\def\Family@Utopia{put}
\def\Family@ZapfChancery{pzc}
\def\Family@Fibonacci{cmfib}
\def\Family@Funny{cmfr}
\def\Family@Dunhill{cmdh}
\def\Family@Concrete{ccr}
\def\Family@Charter{bch}

\def\Family@Fontpxr{pxr}
\def\Family@Fontaer{aer}
\def\Family@Fontaess{aess}
\def\Family@Fontaett{aett}
\def\Family@Fontlcmss{lcmss}
\def\Family@Fontlcmtt{lcmtt}
\def\Family@Fontcmvtt{cmvtt}
\def\Family@Fontcmbr{cmbr}
\def\Family@Fontcmtl{cmtl}
\def\Family@Fontpxss{pxss}
\def\Family@Fonttxss{txss}
\def\Family@Fonttxr{txr}
\makeatother

% Define the language : must be done by the ./configure
% Syntax example.
% For french, add the following line :
% \global\def\Language{FR}

```