LaTeX document class for J3 work

Van Snyder

February 11, 2002

Contents

1 Introduction

This paper describes a LaTeX document class designed to be used for constructing J3 documents. It is intended to be used both for setting the standard, and for writing meeting papers.

2 Large-scale document structure

LaTeX documents begin with a `\documentclass` command. The J3 document class is derived from the `book` document class. All of the options of that class continue to work. An additional option `memo` has been added that makes `\section` the top level structure. If `memo` does not appear, `\chapter` is the top level structure. The `\documentclass` command at the beginning of this document is:

```
\documentclass[twoside,11pt,memo]{j3}
```

In addition to the `memo` option and options of the `book` class, one can put the following options in the `\documentclass` command:

`nocolor` turns off background color for notes. The reason for this is that most versions of the X-windows previewer, `xdvi`, are not able to cope with the commands that generate background colors.

`pdf` generates PDF special commands to make cross-references "live" in PDF documents. Unfortunately, references from the table of contents and the index aren't "live." There's also a `nopdf` option, mainly for symmetry. There are also `\pdf` and `\depdf` (probably the latter should be `\nopdf`) commands.

`html` generates html special commands to make cross-references "live" in the "device independent" (`dvi`) output file. The X-windows previewer, `xdvi`, understands these specials. Unfortunately, the most commonly used program to convert `dvi` to PostScript doesn't understand html specials. Also unfortunately, references from the table of contents and

the index aren't "live." There's also a `nohtml` option, mainly for symmetry. There are also `\html` and `\dehtml` (probably the latter should be `\nohtml`) commands.

The primary differences between the `book` document class and the `j3` document class are:

(1) The default page style is to have headers and footers on every page. The headers and footers have a flush-left part, a flush-right part and a centered part.
If `memo` does not appear, the page heading is as for the draft standard. If `memo` appears, nothing is put into the center of the headers and footers, and the page numbering becomes "Page <this-page> of <last-page>." In the latter case, one is expected to put `\label{lastpage}` immediately before the `\end{document}` command. The `memo` option is intended for producing meeting papers.
Two commands, that you are expected to renew, are invoked during production of the page headers and footers. The first is `\hdate`, and the second is `\vers`. Neither one has an argument. Here are examples of the commands to renew them. You can put them immediately after the `\documentclass` command.
```
\renewcommand{\hdate}{\today\ \printtime}   % Date for headers and footers
\renewcommand{\vers}{<paper number>}        % Version for headers
```
The `\hftitle` command is used to fill the center part of the header and footer. Its default if `memo` is absent is `WORKING DRAFT`. If `memo` appears, its default is empty. In this document, it's
```
\renewcommand{\hftitle}{\LaTeX\ class for J3}
```
The `\hff` command, default `\sffamily\bfseries\large`, is used to set the header and footer font.

(2) There is a new sectioning command `\annex`. It generates the correct form of page heading for annexes of 007. It is a synonym for `\appendix`.

(3) The sectioning commands invoke a command `\secfont` to set the font for sections. The default is `\sffamily`. You can, of course, renew this command. The `\chapter` command puts "Section" before the chapter number, and a colon after. Unlike in the `book` class, our `\chapter` command puts the title all on one line.

(4) The page layout is adjusted to be the same as the draft standard.

(5) Numerous environments and commands have been added. These are described below.

3 Cross-reference labels

The document class provides a command `\divn` that takes two arguments. The first is expected to be a sectioning command, and the second is the title of the section. It invokes its first argument and gives it its second argument. Then it creates a label consisting of "D" followed by the chapter number in arabic numerals and a colon, and then the second argument. Blanks and everything else except TeX special characters, e.g. " and `}`, are significant in labels, and the case of letters is significant. The chapter number is inserted in an attempt to make labels unique. If `memo` is specified, the chapter number is zero. Remember that in LaTeX one can refer to the text of an entity's number with the `\ref` command, and to the text of its page number with the `\pageref` command. This section was begun with

```
\divn\section{Cross-reference labels}
```

This reference, i.e. (3), was produced with `\ref{D0:Cross-reference labels}`.

You can't use `\divn` if the section title has a command in it (because of the \).

In any case, you can create your own labels, on section commands or elsewhere, with the LaTeX `\label` command. If a label is in a table, an equation, a figure, an item in a list, the left-hand side of a BNF term, a constraint (6), and perhaps a few other places, a `\ref` to that label will produce the object's number, not the section number. If the label is in a note (8) the `\ref` produces "Note" and the note number, including the section number.

4 Font specifiers

There are several font specifiers:

`st` The `\st` command sets its argument in "syntax term" type face. The default definition is `\emph`, which in turn defaults to italic.

`obs` The `\obs` command sets its argument in "obsolete font". The command `\obs{obsolete}` produces obsolete.

`cf` The `\cf` command sets its argument in "code font" font. The command `\cf{code font in notes}` produces `code font`.

`obscf` is a combination of `\obs` and `\cf`

5 Support for BNF

Numerous commands are provided to support BNF.

5.1 Commands to create BNF

5.1.1 The `bnf` command

The `\bnf` command is the basic command to set BNF rules. It takes three arguments. The first is the syntax number and syntax term. The second is either **is** or **or** (of course, you can stick anything you want in there). The third is the right-hand side of the BNF rule. The first argument is set in a box 2.25 inches wide. The second is set in `\bf` font in a box equal to the width of **or** plus 1em. The third one is set in a LaTeX `\mbox`, so if it is long, it will extend into the margin instead of being folded. It isn't folded automatically, because we want the continuation mark (see 5.1.7).

If an internal flag `@bnfindex` is `true` it puts the entire syntax rule in the index of syntax rules. This flag is set by `\bnfi` (5.1.4) and cleared by `\bnfn` (5.1.9) and `\bnfx` (5.1.8) commands. There is a `\bmf` command that doesn't put things in the index.

For example, the command `\bnf{\st{abc}}{is}{DEF \st{ghi} JK}` produces

abc **is** DEF *ghi* JK

The `\bnf` command doesn't automatically start or finish a paragraph, so if you don't put blank lines or `\\` around it, you will find a BNF rule in the middle of a line.

Other commands described below are usually easier to use, so you probably won't use either `\bnf` or `\bmf` directly.

5.1.2 The `xsn` command

The `\xsn` ("explicit syntax number") command takes two arguments. The first is an optional syntax rule number (optional arguments are enclosed in square brackets). The second argument is a syntax term. It puts "R" in front of the first argument and sets it in a box 0.5in wide, and then sets the second in the `\st` type face. This is one of the ways to create the first argument for the `\bnf` command. Using `\xsn` in the previous example, e.g.

`\bnf{\xsn[604]{abc}}{is}{DEF \st{ghi} JK}` produces
R604 *abc* **is** DEF *ghi* JK
You probably won't use `\xsn` directly.

5.1.3 The `sn` command

The `\sn` ("syntax name") command takes one argument, a syntax term. It sets its argument in `\st` type face. Then it creates a new syntax number by incrementing the `sr` ("syntax rule") counter, and concatenating it (with at least two digits) onto the chapter or section number (the latter if `memo` is specified). Finally, it creates a label consisting of `sr:` (for "syntax rule") followed by the argument.

Using `\sn` in the previous example, e.g. `\bnf{\sn{abc}}{is}{DEF \st{ghi} JK}` produces
R501 *abc* **is** DEF *ghi* JK
Notice that we're in section 5, and that is the leading digit of the syntax rule number. Also notice that "R" has been put ahead of the syntax number. This is because `\sn` uses `\xsn` (5.1.2) to combine the generated syntax number and the syntax term. You probably won't use `\sn` directly.

5.1.4 The `bnfi` command

The `\bnfi` ("BNF **is**") command takes two arguments. The first is the syntax rule name, and the second is the (first line of) its right-hand side. It generates a syntax rule number and sets the name, using the `\sn` command. Then it puts this as the first argument of the `\bnf` command, puts `is` as the second argument, and puts its second argument as the third argument of `\bnf`. If `memo` is not present in the `\documentclass` command, it enters the syntax term into the index of syntax terms, to be displayed with the syntax rule number and a bold face page number, enters the entire syntax rule in the index of syntax rules, and sets a switch that causes subsequent syntax rules also to be entered in that index. Our above example could have been written `\bnfi{abc}{DEF \st{ghi} JK}`, producing
R502 *abc* **is** DEF *ghi* JK
This would put "*abc* (R502), **4**" into the index of syntax terms. By the way, the "index term" is *abc* alone, so if you put a reference to *abc* in the syntax term index (using the `\tindex` command – see section 7), it will come at the same place in the index.

Notice that the example syntax rule above is not exactly the same as in section 5.1.3, because a new syntax rule number has been invented. In this document, it also generates a duplicate label `sr:abc`, because the term *abc* was also defined in section 5.1.3. (If you have duplicate labels, a `\ref` command refers to the last one of them, so references to `sr:abc` will be to R502.)

5.1.5 The `bnfo` command

The `\bnfo` ("BNF **or**") command takes one argument – the (first line of the) right-hand side of the **or** part of a syntax rule. It's the same as `\bnf{}{or}{`<right-hand-side>`}`. We might continue our above example with `\bnfo{PQR \st{xyz}}`, which produces
or PQR *xyz*

5.1.6 The `bnfr` command

The `\bnfr` ("BNF right-hand-side") command takes one argument – (one line of) the right-hand side of a syntax rule. It puts the `\bnfc` syntax rule continuation symbol (see 5.1.7) before its argument, and then uses the result as the third argument for `\bnf`, i.e. it's the same as `\bnf{}{}{\bnfc` <right-hand side>`}`.

5.1.7 The bnfc command

The \bnfc ("BNF continuation") command has no arguments. It produces the BNF continuation symbol, *viz.* ■ . You need to put this at the end of the right-hand side of continued BNF rules, but \bnfr (see 5.1.6) will put it at the beginning of continuing lines for you.

5.1.8 The bnfx command

The \bnfx ("BNF **is** with eXplicit rule number") command takes three arguments. The first is an explicitly specified rule number. The second is the syntax term. The third is the (first line of the) right-hand side of the rule. The first two arguments are put together by \xsn. This result is then used as the first argument of \bnf, with is for the second argument, and the third argument of \bnfx is used as the third argument for \bnf. This one is intended to be useful for producing meeting papers, wherein you want to refer to a syntax rule number in the standard, not have LaTeX invent one for you. It does not enter its name into the index of syntax terms, or the rule into the index of syntax rules, and it turns off the switch that causes subsequent BNF-generation commands to put their rules into the index of syntax rules.

5.1.9 The bnfn command

The \bnfn ("BNF **is** with rule number gotten by reference to a Name") takes two arguments: the syntax term for the left-hand side, and the (first line of the) right-hand side. The syntax number is gotten by reference to the name (the first argument). This command is used when quoting a syntax rule in a place other than its home. The syntax rules that are defined in section 7 of the standard but referenced in section 3 are set using the \bnfn command. It does not enter its name into the index of syntax terms, or the rule into the index of syntax rules, and it turns off the switch that causes subsequent BNF-generation commands to put their rules into the index of syntax rules.

5.1.10 The bnfz command

The \bnfz ("BNF *zilch* – BNF **is** with no rule number") command takes two arguments: the syntax term for the left-hand side, and the (first line of the) right-hand side. No syntax rule number is produced, but the left-hand side is indented the same amount it would be if a syntax number were provided.

5.1.11 The bnfb command

The \bnfb ("BNF block") command takes one argument: a part of the right-hand side of a syntax rule. It doesn't put \bnfc before the right-hand side. It is intended to be used for constructs.

5.2 Commands to reference syntax terms

There are several commands to display and reference syntax terms and rule numbers. The ones that claim to enter syntax terms into the index of syntax terms only do so if the memo option is not present in the documentclass command.

5.2.1 The si command

The \si ("syntax index") command takes one argument – a syntax term. It sets it in \st type face, and enters the reference into the index of syntax terms.

5.2.2 The sir command

The \sir ("syntax index with reference number") command takes one argument – a syntax term. It sets it in \st type face, then sets its rule number between parentheses, and finally

enters the reference into the index of syntax terms. For example, `\sir{abc}` produces *abc* (R502) (*abc* was defined in section 5.1.4).

5.2.3 The `sid` command

The `\sid` ("syntax index with definition page number") command takes one argument – a syntax term. It sets it in `\st` type face, then enters the reference into the index of syntax terms, with its rule number, as a definition – i.e., with a bold-face page number.

5.2.4 The `sidn` command

The `\sidn` ("syntax index with no syntax number but a definition page number") command takes one argument – a syntax term. It sets its argument in `\st` type face, and enters it in the index with a bold face page number. This is intended to be used for the definition of terms that are defined by explanation rather than BNF rules – e.g. *letter*.

5.2.5 The `sinr` command

The `\sinr` ("syntax index with no syntax number") command takes one argument – a syntax term. It sets its argument in `\st` type face, and enters it in the index. This is intended to be used for terms that are defined by explanation rather than BNF rules – e.g. *letter*.

5.2.6 The `snref` command

The `\snref` ("syntax number reference") command takes one argument – a syntax term. It sets its syntax rule number (not between parentheses). For example `\snref{abc}` produces R502 (*abc* was defined in section 5.1.4).

5.2.7 The `sref` command

The `\sref` ("syntax reference") command takes one argument – a syntax term. It does everything that the `\sir` command does, except for putting a reference in the index.

6 Constraints

There is a list environment for setting several consecutive constraints, and a command for setting one constraint. They both invent new constraint numbers in the same way that syntax rule numbers are invented (but with a "C" instead of an "R"). See section 5.1.3.

6.1 The `cons` environment

The `cons` environment is a list environment for setting several consecutive constraints. Each constraint is introduced by an `\item` command. For example,

```
\begin{cons}
\item First constraint.
\item Second constraint.
\end{cons}
```

produces

C601 First constraint.
C602 Second constraint.

As with any list environment, you can put your own labels in optional arguments of the `\item` command.

The width of the label is the same as the space allowed for the syntax rule number in a BNF definition (actually 0.5in + 1em).

6.2 The dcons command

The \dcons command produces one constraint. It takes two arguments. The first one is optional (remember that optional arguments are enclosed in square brackets). It is an explicit constraint number (with "C" if you want it) to override the generated one. The generated constraint number includes the section number. The constraint counter is incremented even if an explicit one is provided.

The second argument is the text of the constraint. Here is an example of a constraint on R502, produced by `\dcons{(\snref{abc}) The \st{ghi} shall be a ghi.}`:

C603 (R502) The *ghi* shall be a ghi.

This command does *not* comprise a paragraph. Moreover, its body is set using "hanging indentation," which has a scope of the entire paragraph in which it appears. This paragraph is an example of the surprise you'll get if you try to separate \dcons from adjacent text with \\.

7 Commands for indexing

There are three low-level commands to generate index terms. The reason for three is to have separate indices for general terms, syntax terms, and the syntax rules themselves.

The commands are \mindex to enter a term in the "main" index, \rindex to enter a complete syntax rule in the "syntax rule" index, and \tindex to enter a syntax term in the "syntax term" index. There is also a \mindex* command that sets its text *and* puts it in the index. There are also \mindexd and \mindexd* commands that are for definitions – they put a bold-face page number in the index.

The BNF commands use \tindex and \rindex. You will probably not use \tindex directly – it is preferable to use it by way of \si (5.2.1) or \sir (5.2.2). The \tindex command is not effective if memo appears in the \documentclass command. The \rindex command doesn't do anything if the class-internal flag @bnfindx is false, so there's no reason to try to use \rindex directly. It is also not effective if memo appears in the \documentclass command.

The \kw command puts a keyword into the index. The \kw* command puts the keyword into the text and the index. There are also \kwd and \kwd* commands that are for definitions – they put a bold-face page number in the index.

8 Environment for notes

The note environment increments a note counter, sets **NOTE** followed by the section and note numbers separated by a period, and then puts the body of the note in a box. The box is created using the longtable environment, which allows to split tables across page boundaries. If a note is split, the bottom of the box is not drawn on the continued page, the note heading is duplicated on the continuing page with "**(cont.)**", and the top of the box on the continuing page is not drawn. The only place notes can split is after an "end of item" (\\) signal. The text in a note box can be colored by defining notefore, e.g., `\definecolor{notefore}{rgb}{1,0,0}`. You can also turn off text coloring by redefining \beforenote to do something else, or nothing (e.g. `\renewcommand{\beforenote}{\relax}`). In case you want to do something entirely different, there's an \afternote command that the document class defines to do nothing.

Note backgrounds are also colored. The color can be specified by defining noteback, e.g., `\definecolor{noteback}{gray}{0.95}`. Note coloring, both foreground and background, can be turned off by putting the nocolor option in the \documentclass command. The

`\beforenote` and `\afternote` commands are defined to do nothing, but they're still invoked, so you can turn foreground coloring back on by defining `\beforenote` to be `\color{notefore}`. One reason to turn off note background coloring is that it is done by PostScript specials. Neither `xdvi` nor `dvilj` know what to do with these; they just throw up their hands in despair "Oh Dear! PostScript color specials! I better just do black (no matter what the color)!" So you get a black background with black text on it.

Here's a note created by

```
\begin{note}
 This is a note.  Its background color is noteback and its
 foreground color is notefore.
\end{note}
```

NOTE 8.1

This is a note. Its background color is noteback and its foreground color is notefore.

9 Support for the intrinsic procedures sections

9.1 Subsections in the intrinsic procedures sections

The `\insubsection` ("intrinsic subsection") command sets its argument with the same spacing, size and font as a `\subsection` command, but it doesn't create a table-of-contents entry.

9.2 Environment for the table of specific and generic names

The `threecol` environment is used to set the three-column table of specific and generic names in section 13.6. Actually, there are four columns – one for the bullet that indicates the specific name is not allowed to be an actual argument, but the equivalent tag in Frame was named `threecol`.

9.3 Environment to display intrinsic procedure summaries

The `\insum` environment is a list environment intended for the intrinsic procedure summaries in section 13.

9.4 Environment for arguments for intrinsic procedures

The `args` environment is a list environment. Each item sets its optional argument (the one in square brackets) in bold face type in a 1.5in box. Also see 9.5.

9.5 A command to display intrinsic procedure arguments

The `\intrinarg` command takes two arguments. The first is an intrinsic procedure argument name, and the second is its description. It does the same thing as the `args` environment (9.4), but only for one argument.

9.6 An enumeration environment for intrinsic function argument cases

The `incase` environment is a list environment. The label of each item is set in `\emph` type face. It consists of the word "Case" followed by the optional argument of the `\item` command in parentheses, followed by a colon. If no item label argument is given, one is generated in lower-case roman numerals. The item label is set in a box 0.8125 inches wide.

9.7 Paragraphs in intrinsic procedure descriptions

In the intrinsic procedures sections, paragraphs are introduced by a word in bold-face type – or not – but in either case, the paragraph is indented using the `\inp` command (11.2) and the length `\II` ("intrinsic indent"), which has a value of 0.5in.

The paragraphs that are introduced by a word in bold-face type are generated by the following commands. The commands that end in B are intended to be "bigger" – then have more line spacing.

command	introductory word	command	introductory word
`argument`	**Argument**	`arguments`	**Arguments**
`class`	**Class**	`desc`	**Description**
`example`	**Example**	`exampleB`	**Example**
`examples`	**Examples**	`examplesB`	**Examples**
`reschar`	**Result Characteristics**	`restriction`	**Restriction**
`result`	**Result**	`resvalue`	**Result Value**
`resvalueB`	**Result Value**		

These are generated by an `inpara` ("intrinsic paragraph") command, which takes two arguments – the bold-faced word, and the rest of the paragraph. It puts a period after the bold-faced word, and sets the whole thing as an "indented paragraph" using the `\inp` command.

10 Miscellaneous list environments

There are several list environments, with their label widths and styles chosen to match the draft standard.

10.1 A general enumeration environment

The `enum` environment is similar to the LaTeX `enumerate` environment. The differences are

(1) the outermost label width is 3/4 inch
(2) the remaining label widths are 3/8 inch
(3) the numbering for the outermost level is arabic in parentheses
(4) the numbering for the second level is lower-case alphabetic in parentheses
    (a) that is, like this one

The remainder of its behavior is the same as for the LaTeX `enumerate` environment. I looked superficially for three-level lists in the draft standard, but didn't find any. If there are any, it will be easy to change `enum` to have the same style.

10.2 A "non-bold label" description environment

The `nbdesc` environment is a list environment that works like the LaTeX `description` environment, except that it doesn't set the labels in bold face type.

11 Miscellaneous commands

11.1 A command to define a term

The `\tdef` command sets its argument in bold face type, and creates an index entry for it that will have a bold face page number.

11.2 A command to generate an indented paragraph

The `\inp` command generates an "indented paragraph." It takes one argument: The amount to indent the paragraph. The entire paragraph is indented this amount. It doesn't matter where it appears in the paragraph.

11.3 A command to generate a hanging indented paragraph

The `\hin` command generates a "hanging indented paragraph." It takes one argument: The amount to indent the paragraph. The first line of the paragraph is not indented, but the rest of the paragraph is indented the amount given by the first argument. It doesn't matter where it appears in the paragraph.

11.4 Captions in tables

The `\jcaption` ("J3 caption") command generates a caption for a table that consists of the word "Table" followed by the table number and a colon, and then its argument in bold-face type. It also makes a label for the table that consists of "T" followed by a colon, followed by the text of the caption.

The `\ccaption` ("continued caption") command generates a caption for the part of a table continued onto a subsequent page as for `\jcaption` but followed by "(cont.)". It doesn't generate a label.

12 Generating the standard

The standard is organized as a top-level document that includes low-level documents. LaTeX provides an `\includeonly` command that allows to process only a part of the document, without clobbering the cross references and indexes for the rest of it. If you want to generate just one part, uncomment the `\includeonly` near the top of the main document and put a file name in it (without `.tex`). Unfortunately, you frequently get an extra page or two at the beginning and/or end of the section.

There is a `Makefile` to make the standard.

The command `make 007.dvi` runs `latex` on `007.tex`. Then it runs `makeindex` twice, once for the index of ordinary terms, and once for the index of syntax terms. These use the J3 index style file `j3.ist`. Then it converts the index of syntax rules to something that can be put back into the document. This consists only of copying it because the first thing `latex` does is clobber it, so the `\input` for it in subsequent passes would just get an empty file. Using `makeindex` wouldn't be appropriate, because that would sort the syntax rules incorrectly (e.g. all of the **or** rules would come at the end). Finally, it runs `latex` twice more, to make sure that all of the cross references and line numbers are correctly resolved. The result is the TeX "device independent" file `007.dvi`.

Having `007.dvi`, one can convert it for output on different printers. One can use `make 007.ps` to make a PostScript file `007.ps`. One could also view it using `xdvi` on Unix systems or an equivalent program on other systems, or convert it for different printers. There aren't any `Makefile` sections for other conversions.

PDF is generated by `make 007.pdf`. This section in the `Makefile` doesn't handle any indexing, so even though PDF is made directly from the `tex` file by `pdflatex`, this section causes `007.dvi` to be made first in order to get the indexing done.

Text is made from PDF by `make 007.txt`.

The command `make all` makes `dvi`, PostScript, PDF and text.

The command `make clean` deletes all of LaTeX's output and intermediate files.

The command `make ui-index` makes the "index of unresolved issues" paper. It uses the file `ui-index.tex`, into which you will need to insert the `\hdate` and `\vers` commands (2).

13 Commands useful in generating meeting papers

13.1 The `edits` command

The `\edits` command generates a description of the typographical conventions. It takes two arguments. The first one is optional (remember that optional arguments appear in square brackets). The section title is "Edits" followed by the optional first argument. The second one is the version of the draft standard to which the edits apply, e.g. `01-007r2`.

13.2 The `sep` command

The `\sep` command creates a vertical space of 5pt, and then generates a line that goes all the way across the page. It has no arguments. Here's what it does:

13.3 The `mgpar` command

The `\mgpar` command creates a marginal paragraph. Its primary use is to put page and line numbers in the margin. There's a marginal paragraph adjacent to the first line of this paragraph. The `\mgpar*` command doesn't begin with an empty `mbox`. This changes vertical spacing, which usually makes it wrong, but improves things for marginal paragraphs adjacent to notes. `\mgpar`

13.4 The `mgpare` command

The `\mgpare` command creates a marginal paragraph in `\emph` font (hence the "e" in the name). There's also `\mgpare*` that works as for `\mgpar*`. *`\mgpare`*

13.5 Put boxes around stuff

The `boxit` environment puts a box around its content. The `lbox` environment also puts a box around its content, but it does it by using the `longtable` environment, so it can be split at "new item" (`\\`) signals at page boundaries.

13.6 Note with explicit note number

The `xnote` environment creates a note – in the same way that `note` does (8), but instead of inventing a note number, you specify it. The command `\begin{xnote}{XYZ}` introduces an environment that creates a note box with **NOTE XYZ** above it.

13.7 J3 internal note

The `jnote` environment creates a note – in the same way that `note` does, but instead of inventing a note number, it puts **J3 internal note** above the box.

13.8 References to the standard

The `xr` package can be used to make "external references," by putting the following in a paper:

```
\usepackage{xr}
\externaldocument{007}
```

This requires that the `007.aux` file be accessible. Using the `tetex` TeX distribution on Linux or Unix, this can be done by naming the appropriate directory in the `TEXMFLOCAL` environment variable. The directory named in `TEXMFLOCAL` needs to have a `tex` subdirectory, and that subdirectory needs to have a `latex` subdirectory. All of the subdirectories there are automatically

searched. I put a "soft link" from there to `$HOME/f2000/007`, which is in turn a "soft link" to the directory having the `.tex` files for the current standard. There are probably analogous ways to set things up for Windows-based distributions of TeX.

I run `latex` on the standard, using the `Makefile` the editor prepared, to generate the `.aux` files.

Once things are set up, and the `.aux` files generated, cross references in meeting papers can be identical to cross references in the standard, thereby saving some work for the editor.