cernman, a CERN style for preparing computer documentation[1]

Michel Goossens
CN Division
CH-1211 Geneva 23
Switzerland
Email: <goossens@cernvm.cern.ch>

Printed July 29, 2015

Short summary

This report describes a LaTeX (major) style file, used at CERN for writing computer documentation. It is based on Lamport's[1] "standard" report style. Some of the basic LaTeX commands and environments are redefined and several other commands are added. The New Font Selection Scheme is used throughout, as are several extensions to LaTeX available on the major archives. The main emphasis has been put on providing high level commands, somewhat in the spirit of SGML, so that typographic and page layout conventions can be easily implemented in a coherent and global way. The quasi-automatic translation from the commands defined in this style into SGML and HTML (www), and from there into a hierarchical information retrieval systems, e.g. hypertext based, is also an important consideration. The basic idea is to tag all information to a level as detailed as possible and then to ignore these tags if they are not needed in a given application. It is important to realize that throwing away non-needed information is trivial, while looking for needed information, which is absent, is impossible.

[1] This file has version number v3.07, last revised 93/07/08, documentation dated 93/07/08.

Table of Contents

Chapter 1: The user commands

In this first part we describe sectioning and simple text generating commands, with or without argument, which you can use with the `cernman` style.

For the layout `cernman` uses LaTeX's standard `report` style. This has the advantage that the present style file is more robust with respect to future internal changes to the LaTeX styles.

1.1 Sectioning commands

To provide somewhat more control over the page breaking behaviour at the boundaries of sections in the text, the higher level sectioning commands `\section`, `\subsection` and `\subsubsection` have been given the dual equivalents `\Section`, `\Subsection` and `\Subsubsection`. These latter commands have two mandatory parameters, namely firstly the space which must still be available on the current page to contain the title and the beginning of the section and secondly the title of the section (i.e. the standard parameter of the original LaTeX commands)

As an example the following command first verifies whether at least 4cm remain available on the current page. If the answer is positive, the command acts as a normal `\section` command, while if the answer is negative, the current page is ended by issueing a `\newpage` command, then on the next page the title is written in the conventional way and the section is started.

```
\Section{4cm}{Title of the section}
Start of the text of the section...
```

1.2 Array, equations and other goodies

In order to use the extended functionality of F. Mittelbach's `array` style, the latter is included by default.

To increase the functionality for handling equations, arrays and figures various style files are included (like `subeqnarray`, `subeqn` and `subfigure`; (see "TeX at CERN"[**?**] for details).

LaTeX is known for its poor handling of floats. All too often you end up with all tables and figures at the end of your document. Sometimes you can can get around the problem by using enough `\clearpage` commands, but a simpler solution is to take over the placement control of the material and put it were it should go yourself (of course you must then be sure that there is enough space available, or else you can end with large patches of white space ...). Two enviroments `Tabhere` and `Fighere` provide the needed functionality of handling the caption material correctly in the case of a table and a figure respectively, in both cases forcing at the same time the material to be placed **here**, i.e. no float is generated.

1.3 Simple text generating commands

In order to retain consistency across documents a series of text generating commands which start a new paragraph and may or may not be followed by another start of paragraph are defined below.

`\Action`

Typesets the string "**Action:**" to initiate the description of a routine or command.

`\Example`

Typesets the string "**Example:**" to indicate the start of an example.

`\Examples`

Typesets the string "**Examples:**" to indicate the start of several examples.

`\Idesc`

Typesets the string "**Input parameter description:**" to indicate the start of the description of input parameters.

`\Odesc`

Typesets the string "**Output parameter description:**" to indicate the start of the description of output parameters.

`\Pdesc`

Typesets the string "**Parameter description:**" to indicate the start of the description of routine parameters.

`\Remark`

Typesets the string "**Remark:**" to indicate the start of an remark.

`\Remarks`

Typesets the string "**Remarks:**" to indicate the start of several remarks.

1.3.1 Text generating commands with arguments

This section contains a list of simple commands, which allow you to "tag" text in a document in a generic way, i.e. by specifying what its function is, rather than its precise formatting. In some cases the tagged string (or the one specified as an optional argument) will also be entered in the index.

`\Carg{`*com-arg*`}`

Flags the **use** of the **command argument** *com-arg*, e.g., `\Carg{MYCARG}` typesets MYCARG.

`\Cargdef{`*com-arg*`}`

Flags the **definition** of the **command argument** *com-arg*, e.g., `\Cargdef{MYCARG}` typesets `MYCARG` and enters `MYCARG` also into the index as an argument defining entry. There should only be one such defining entry per argument in a document.

`\Cdef{`*com-name*`}`

Flags the **definition** of the **command name** *com-name*, e.g., `\Cdef{MYCOM}` typesets `MYCOM` and enters `MYCOM` also into the index as a defining entry. There should be only one defining entry in a document.

`\Cind[`*idxname*`]{`*com-name*`}`

Flags the **use** of the **command name** *com-name*. The mandatory argument *com-name* will by typeset in the text and also in the index as a reference to the command name in question. If the optional argument *idxname* is specified, then the latter will be used as the index entry. Either of the two strings can be empty, with the expected result. For instance if you want to flag an entry **without** putting it into the index, you can use `\Cind[]{cmdname}`, while `\Cind[idxname]{}` will enter `idxname` into the index, but typeset nothing in the text. The latter is useful if you want to flag the use of e.g., some routines or commands in an example, and you do not want to enter these LaTeX commands inside the text of the example itself. So you can put a series of the latter commands just following the example, each instance containing the name of a command or routine used in the example in question. Finally, `\Cind{MYCOM}` typesets `MYCOM` and enters `MYCOM` also into the index as a reference entry.

`\Copt{`*com-opt*`}`

Flags the **use** of the **command (argument) option** *com-opt*, e.g., `\Copt{MYCOPT}` typesets MYCOPT.

`\Coptdef{`*com-arg*`}`

Flags the **definition** of the **command (argument) option** *com-opt*, e.g., `\Coptdef{MYCOPT}` typesets `MYCOPT` and enters `'MYCOPT'` also into the index as an option defining entry. There should only be one such defining entry per option in a document.

`\Rarg{`*rout-arg*`}`

Flags the **use** of the **routine argument** *rout-arg*, e.g., `\Rarg{MYRARG}` typesets MYRARG. Note that the word "routine" should be understood in its widest possible meaning, i.e., Fortran as meaning Fortran funtion, routine, or C, Pascal, etc., procedure.

`\Rargdef{`*rout-arg*`}`

Flags the **definition** of the **routine argument** *rout-arg*, e.g., `\Rargdef{MYRARG}` typesets `ARG` and enters `MYRARG` also into the index as an argument defining entry. There should only be one such defining entry per argument in a document.

`\Rdef{`*rout-name*`}`

Flags the **definition** of the **routine name** *rout-name*, e.g., `\Rdef{MYROUT}` typesets `MYROUT` and enters `MYROUT` also into the index as a defining entry. There should be only one defining entry in a document.

`\Rind[`*idxname*`]{`*rout-name*`}`

Flags the **use** of the **routine name** *rout-name*. The mandatory argument *rout-name* will by typeset in the text and also in the index as a reference to the routine name in question. grep Cind If the optional argument *idxname* is specified, then the latter will be used as the index entry. Either of the two strings can be empty, with the expected result. For instance, `\Rind{MYROUT}` typesets `MYROUT` and enters `MYROUT` also into the index as a reference entry.

`\Ropt{`*rout-opt*`}`

Flags the **use** of the **routine (argument) option** *rout-opt*, e.g., `\Ropt{MYROPT}` typesets 'MYROPT'.

`\Roptdef{`*rout-arg*`}`

Flags the **definition** of the **routine (argument) option** *rout-opt*, e.g., `\Roptdef{MYROPT}` typesets `'MYROPT'` and enters `'MYROPT'` also into the index as an option defining entry. There should only be one such defining entry per option in a document.

`\Ucom{`*user-com*`}`

The argument *user-com* corresponds to a command to be types by the user and it is typeset in a special way to make it stand out clearly, eg `\Ucom{ftp asis01}` is typeset as `ftp asis01`.

Chapter 2: More complex commands

2.1 Commands to highlight calling sequences and commands

This section describes a series of commands for describing the structure of Fortran routines or function, C procedures, KUIP commands, etc. For most of the commands an optional argument allows you to enter information into the left margin, so that commands can be easily located. The other parameters specify the name of the label with which the entity being defined will be cross-referenced in the index, the next argument the part of the command or sequence to be highlighted and finally the parameters themselves. Apart from the optional (first) parameter between square brackets, all other parameters, as described below **must** be specified. In particular, if there are no parameters, the last parameter to the command should be empty `{}`.

The general structure of these commands is:

`\Sxxx[`*margintext*`]{`*labeltext*`}{`*maintext*`}{`*parameters*`}`

margintext	This optional parameter specifies the text should be written into the left margin.
labeltext	Character string to be used to refer to the entity in the text.
maintext	Character string corresponding to the name of routine/command being defined. In some cases this string acts also as `labeltext`.
parameters	List of the parameters which the routine/command has.

`\Shubr[`*margintext*`]{`*Rname*`}{`*arguments*`}`

The first **obligatory** parameter *Rname*, which is the routine's name, acts in this case also as label to be used in a `\ref` command. This means that one can reference the page where the given command is defined as `\pageref{Rname}`. The routine will also be entered under that name in the index. In hypertext application, this commands serves as an anchor defining entry for the routine *Rname*, and will be referenced by the anchor referencing commands `\Rind`. An example and its typeset result are:

```
\Shubr[HBOOK]{HBNT}{(ID,CHTITL,CHOPT)}
```

HBOOK `CALL HBNT (ID,CHTITL,CHOPT)`

You can reference this entry by `\Rind{HBNT}` (typeset like HBNT) and start talking about the arguments of the routine as `\Rarg{CHOPT}` (typeset as CHOPT) and about the option for this CHOPT parameter as `\Ropt{M}` (e.g.,, 'M' for memory resident Ntuple).

`\Shubrii[`*margintext*`]{`*Rname1*`}{`*parameters1*`}{`*Rname2*`}{`*parameters2*`}`

`\Shubrii` describes two similar Fortran subroutines at the same time, with the word **and** between the two definitions. Of course only one marginal comment can be specified as an optional parameter *margintext* if needed. An example and its typeset result are:

```
\Shubrii{HBPROX}{(ID,VMX)}{HBPROY}{(ID,VMX)}
```

```
 CALL HBPROX (ID,VMX)  and   CALL HBPROY (ID,VMX)
```

`\Sfunc[`*margintext*`]{`*Fname*`}{`*Value=Fname(parameters)*`}`

`\Sfunc` describes a Fortran function. In this case, as there should always be at least one function parameter, the last parameter is parsed and the part between the equal sign and the opening bracket will be highlighted. As in the case of `\Shubr` the first obligatory parameter is used as cross-reference tag and entered into the index. An example is given below.

```
\Sfunc{HEXIST}{LOGVAR = HEXIST (ID)}
```

```
 LOGVAR = HEXIST (ID)
```

`\Sfuncii[`*margintext*`]{`*Fnam1*`}{`*val1=Fnam1(par1)*`}{`*Fnam2*`}{`*val2=Fnam2(par2)*`}`

`\Sfuncii`, as `\Shubrii`, describes two Fortran functions which are quite similar and whose description is short enough to fit on one line. An example is given below.

```
\Sfuncii[Hbook]{HI}{VARIAB = HI (ID,I)}{HIJ}{VARIAB = HIJ (ID,I,J)}
```

Hbook
```
 VARIAB = HI (ID,I)  and   VARIAB = HIJ (ID,I,J)
```

`\SCubr[`*margintext*`]{`*Pname*`}{`*parameters*`}`

`\SCubr` describes a C routine. Its calling sequence is completely identical to the one of `\Shubr`. The only difference is that the word `CALL`, characteristic of a Fortran language's calling sequence is not prepended to the output string.

`\SCubrii[`*margintext*`]{`*Pnam1*`}{`*pars1*`}{`*Pnam2*`}{`*pars2*`}`

`\SCubrii` describes two similar C routines. Similar remarks as for `\Shubrii` and `\SCubr` apply.

`\SKUIP[`*short˙name*`]{`*Cname*`}{`*arguments*`}`

`\SKUIP` describes a KUIP command. In this case the optional parameter *short˙name* is the short name to be used a cross-reference label. This is usefull if the command is complex, e.g.,

```
\SKUIP[ADD]{HISTOGRAM/OPERATIONS/ADD}{id1 id2 id3 [c1 c2]}
```

```
 HISTOGRAM/OPERATIONS/ADD  id1 id2 id3 [c1 c2]
```

References to this command can be made with the `\Cind` command. Therefore it is perhaps better to chose a somewhat more precise *short˙name*, since there might be other commands which "add" something together.

2.2 Extended list environments

LaTeX's standard list environments have been augmented by introducing description (definition) lists where the width of the term can be specified. Lists have also been given a "dense" version, i.e., there is less spacing between the items. The table below gives an overview of these new list environments.

Environment	Description
`\begin{DL}{`*width*`}`	Description list with width of largest item Text of term is in bold
`\begin{DLtt}{`*width*`}`	Description list with width of largest item Text of term is in teletype font
`\begin{DLc}{`*width*`}`	Denser version of `DL` list
`\begin{DLctt}{`*width*`}`	Denser version of `DLtt` list
`\begin{OL}`	Numbered (ordered) list
`\begin{OLc}`	Dense numbered (ordered) list
`\begin{UL}`	Unnumbered (itemize) list
`\begin{ULc}`	Dense unnumbered (itemize) list

Examples of the use of these lists are

`CHOPT` Option chosen

`ID` Identifier

`IFLAG` Flag

- `0` Start
- `1` Continue
- `2` Stop

```
\begin{DLtt}{12345}
  \item[CHOPT] Option chosen
  \item[ID]    Identifier
  \item[IFLAG] Flag
    \begin{DLtt}{1}
      \item[0] Start
      \item[1] Continue
      \item[2] Stop
    \end{DLtt}
\end{DLtt}
```

2.3 (Almost) verbatim environments

Often it is desirable to show text (computer programs, listings) "literally", i.e., "as typed", but at the same time it is still sometimes necessary to be able to use some LaTeX commands inside such an environment, e.g., to highlight some text inside the example. Short literal text can be tagged with the `\Lit` command, e.g.,

`Literal` **`bold`** `#$%^&*`

```
\Lit{Literal \textbf{bold} #$%^&*}
```

For multiple lines two "example" environments have been defined, namely

```
\begin{XMP}
\begin{XMPt}{Example title}
```

They are inspired by LaTeX's standard `verbatim` environment. As with the `\Lit` command, all characters, but `\`, `{` and `}` are typeset "as typed", i.e., without having to take special precautions. Inside the

`XMP` environments all blanks and empty lines are output as such.

`ftp asis01`
user: `anonymous`

```
\begin{XMP}
\Ucom{ftp asis01}
user: \Ucom{anonymous}
\end{XMP}
```

Typing funny characters

After the empty line we type #$%^&

Above is an *empty* line.

```
\begin{XMPt}{Typing funny characters}
After the empty line we type #$%^&

Above is an \textem{empty} line.
\end{XMPt}
```

Chapter 3: Short references for special characters

Table 3.1 shows which control sequences for special characters are predefined in the `cernamn` style. They should be used in preference to lower level LaTeX or TeX control sequences when you want to tag the special function of such a character. Table 3.2 shows control sequences to be used for tagging CN-AS Group's packages.

&	\amp	´	\apos	*	\Ast	@	\atsign	\	\bs
\	\bsol	^	\Circ	:	\Colon	@	\commat	$	\dollar
!	\excl	-	\hyphen	<	\lab	{	\lcb	{	\lcub
(	\lpar	[	\lsb	[	\lsqb	`	\lsquo	#	\num
.	\period	%	\percent	%	\percnt	?	\quest	"	\quot
>	\rab	}	\rcb	}	\rcub	)	\rpar	]	\rsb
]	\rsqb	´	\rsquo	␣	\sbl	;	\semi	/	\sol
~	\Tilde	_	\us	|	\verbar				

Table 3.1: Command sequences for special characters

cmz	\CERNLIB	cmz	\CMZ	comis	\COMIS
cspack	\CSPACK	fatmen	\FATMEN	geant	\GEANT
gks	\GKS	hbook	\HBOOK	hepdb	\HEPDB
higz	\HIGZ	hplot	\HPLOT	kuip	\KUIP
minuit	\MINUIT	patchy	\PATCHY	paw	\PAW
zebra	\ZEBRA				

Table 3.2: Command sequences to tag CNAS packages

Chapter 4: Special minor styles

The `cmz` and `geant` manuals have a somewhat special layout, since they contain many (over 200) more or less short descriptions of independent routines. Therefore a special macro-command was defined in each case to format these manual pages, tagging the relevant information for each routine in an adequate way. These specific commands are grouped in two minor styles, `cernlib` and `crngeant`, which are described below.

4.1 The `cernlib` style

The `cernlib` style defines command sequences for flagging detailed information about every package. These commands are described below:

`\Accuracy`

Typesets the string "**Accuracy:**" on a line by itself.

`\Errorh`

Typesets the string "**Error handling:**" on a line by itself.

`\Longwr`

Typesets the string "**Long Write-up:**" on a line by itself.

`\Method`

Typesets the string "**Method:**" on a line by itself.

`\Notes`

Typesets the string "**Notes:**" on a line by itself.

`\Refer`

Typesets the string "**References:**" on a line by itself.

`\Restrict`

Typesets the string "**Restrictions:**" on a line by itself.

`\Source`

Typesets the string "**Source:**" on a line by itself.

`\Structure`

Typesets the string "**Structure:**" on a line by itself.

`\Timing`

Typesets the string "**Timing:**" on a line by itself.

`\Usage`

Typesets the string "**Usage:**" on a line by itself.

`\Cernhead{`*Write-up title*`}`

The `\Cernhead` command starts the description of a new routine, resetting the page-counters, enters information into index, and typesets the title heading for the writeup. This command uses the information defined by the commands below:

`\Author{`*Author name*`}`
`\Authors{`*Author names*`}`

You can specify the author(s) of the routine with the `\Author` and `\Authors` commands, e.g., `\Authors {F.Bruya`

`\Keywords{`*List of keywords*`}`

The `\Keywords` command allows you to specify keywords associated to the routine being described. The keywords are written to an external file and can be used in online or hypertext versions of the documentation. e.g.,

```
\Keywords{integration, rational, square root}
```

`\Language{`*language˙name*`}`

This command specifies in which computer language the package/routine is written, e.g., Fortran, C, Assembler, etc.

`\Library{`*library˙name*`}`

This command specifies the name of the computer library in which the given package/routine can be found, e.g., `KERNLIB`, `MATHLIB`.

`\Origin{`*Source*`}`

With the `\Origin` command (`geant` only) you can specify the source where the routine comes from, e.g., `\Origin{GEANT3}`.

`\Revised{`*revision˙date*`}`

This commands specifies the date of the last revision.

`\Routid{`*Routine˙identifier*`}`

Identifier for routine (`geant` only), e.g., `BASE020`.

`\Submitter{`*Name˙of˙submitter(s)*`}`

The `\Submitter` command allows you to specify the name(s) of the person(s) who submitted the routine(s).

`\Submitted{`*submission˙date*`}`

This commands specifies the date when the package was submitted, e.g., `\Submitted{01.10.84}`.

`\Version{`*package˙code˙name*`}`

The `\Version` command specifies the name with with the package is identified in the library (`cmz`) (e.g., `\Version{BINOM}`) or the version number for `geant`, (e.g., `\Version{Geant 3.11}`).
As an example, figure 4.1 on the facing page shows the input and generated output for `cmz` routine `B100`

4.2 The `crngeant` style

The `crngeant` minor style includes all the commands of `cernman` and `cernlib`. As the layout for the description of the individual routines in the `geant` manual is somewhat different from the one of the `cmz` short writeups, a different command, `\Makehead` (instead of `\Cernhead`) should be used:

`\Makehead{`*Heading title*`}`

An example of the beginning of the description of `geant` routine `BASE020` is shown below.

```
\Authors {F.Bruyant,M.Maire}  \Origin{GEANT3}
\Submitted{01.10.84}          \Revised{23.05.91}
\Version{Geant 3.11}          \Routid{BASE020}
\Makehead{The Data Structures and their Relationship}
\begin{figure}[hbt]
....
```

In order to ease the introduction of certain physical quantities, the following definitions, which can be used in text and math mode, are also provided:

kev	`\UkeV`	Mev	`\UMeV`	Gev	`\UGeV`	Tev	`\UTeV`
e	`\Pe`	m_e	`\Pme`	e^-	`\Pem`	e^+	`\Pep`

```
\Version{BINOM}                      \Routid{B100}
\Keywords{BINOMIAL COEFFICIENT}
\Author{K.S. K\"olbig}               \Library{MATHLIB}
\Submitter{}                         \Submitted{15.02.1989}
\Language{Fortran}                   \Revised{15.03.1993}
\Cernhead{Binomial Coefficient}
Function subprograms \Rind{BINOM} and \Rind{DBINOM} calculate
the binomial coefficient
\[ x \choose k \ = \
  \left\{ \begin{array}{ll}
           x(x-1)\ldots(x-k+1)/k! &  (k>0) \\
           1                      &  (k=0) \\
           0                      &  (k<0) \\
          \end{array}                          \right.  \]
for real $x$ and integer $k$.
Function subprogram {\tt KBINOM} calculates the
binomial coefficient only for integer $x=n$.

On CDC and Cray computers, the double-precision version
\Rind{DBINOM} is not available.
\Structure
{\tt FUNCTION} subprograms \\
User Entry  Names: \Rdef{BINOM}, \Rdef{DBINOM}, \Rdef{KBINOM} \\
Files Referenced: {\tt Unit 6}
\Usage
In any arithmetic expression,
\begin{center}
  \Lit{BINOM(X,K)}, \quad \Lit{DBINOM(X,K)} \quad or \quad
  \Lit{KBINOM(N,K)}
\end{center}
has the value of the binomial coefficient. \Rind{BINOM} is
of type {\tt REAL}, \Rind{DBINOM} is of type
{\tt DOUBLE PRECISION} and \Rarg{X} has the
same type as the function name. \Rarg{KBINOM}, \Rarg{N} and
\Rarg{K} are of type {\tt INTEGER}.
\Restrict
Function subprogram \Rind{KBINOM} can compute only binomial
coefficients which lie in the integer range of the machine.
\Accuracy
Full machine accuracy.
\Errorh
If the result of \Rind{KBINOM} would lie outside the integer
range ofthe machine, \Rind{KBINOM} is set equal to zero and
an error message is printed.  \newline
$\bullet$
```

Figure 4.1: Example of using the `cernman` and `cernlib` styles

Chapter 5: Making online versions – `www`

The major style `cernman` has been originally optimized to generate the PostScript version of the manuals and writeups describing the software of CN/AS Group. Therefore, in order to use the same input LaTeX files in an (almost) unaugmented way also for producing an online terminal browsable version or a text version with SGML/HTML tags for viewing with an hypertext system, many of the LaTeX commands and font definitions have to be redefined. This transformation is performed by overloading the `cernman` major style (and possibly the `cernlib` and `crngeant` minor styles) with the minor styles `cerndoc` and `cernhtml`. The latter style also introduces the necessary HTML tags and hypertext links into the document.

Instead of transforming the `dvi`-file into PostScript with the `dvips` program, in the latter case the program `ptd` (Post TeX Doc), is used to generate an ASCII version of the `dvi`-file, which can be viewed or handled by a post-processor to generate the necessary files and resolve the necessary links for the hypertext (e.g., `www`) systems.

`cernhtml` and `cerndoc` redefine most commands set up in the `cernman` and its associated styles `crngeant` and `cernlib` into a form that can be used for an ASCII text or for an HTML tagged version.

Figure 5.1 on the next page gives a complete overview of the various files and programs involved in producing the various forms of the CN/AS documentation. At the left you have the various documents, marked up in LaTeX, and containing generic tags for flagging document elements. These files are run through LaTeX and depending on the package or output destination, various style files are loaded together with the generic style file `cernman`. The bibliography database `cnasbibl.bib` can be read by BibTeX and the index generated with . In the next step the `dvi`-file is translated into PostScript with `dvips` or in a text file with `ptd` for further treatment with an awk script for generating the HTML files or the terminal readable text files.

5.1 The HTML interface

The only HTML specific addition, which was introduced into the LaTeX sources of the documents was a command `\Filename`, which indicates the filename to be given to the part of the text starting at that point. In principle the introduction of this information is not needed per se, but it helps to manage the many hundreds of files generated for `www` related to the CN/AS manuals. The filename information is used by LaTeX (in the `cernhtml` style) to build the definition part of the hypertext links (anchors). This information is later extracted in the post-processing stage of the ASCII form of the document in question to resolve, using a shell script `makewww.sh`, the hypertext cross-references. During this latter stage possible unresolved references generate a warning message on the user's screen.

If, in certain points of your document, you want to include HTML specific information (I think you should not, but, for instance, on the title page you might have to), you can use the command

`\HTML{`*HTML specific text and commands*`}`

The text specified as argument will only be treated when you specify the `cernhtml` style option. This command is ignored by the other styles.

If you want to treat some fo your LaTeX source lines only when *not* using HTML, the you can use:

`\notHTML{`*non-HTML text and commands*`}`

Figure 5.1: Producing CN/AS documentation

5.2 A short overview of HTML

This section presents a short overview of HTML, www's text processing language. HTML, the HyperText Markup Language, is defined in terms of the ISO Standard Generalized Markup Language (SGML). The latest version of the HTML language specification can always be found in www, by choosing the "www project" e.g. on the CERN Welcome page, from there go to "Technical aspects of www" and "Hypertext Markup Language".

5.2.1 A HTML document instance

Example of a simple HTML document

```
¡HTML¿
¡TITLE¿This is a simple HTML document¡/TITLE¿
¡H1¿An Example of a level one heading¡/H1¿
This is the first paragraph of level one.
¡H2¿An example of a second level heading¡/H2¿
This is the first paragraph of level two.
¡P¿Another paragraph of level two with a list.
¡UL¿
¡LI¿This item contains an ¡A
   NAME="#my-anchor"¿anchor definition¡/A¿.
¡LI¿This item the second item in the list.
¡/UL¿
And here follows an ordered list.
¡OL¿
¡LI¿This item references the ¡A
   HREF="#my-anchor"¿previous anchor¡/A¿.
¡LI¿Another item in the list.
¡/OL¿
¡P¿Let me show you a definition list:
¡DL¿
¡DT¿Term 1¡DD¿Text of description 1.
¡DT¿Term 2¡DD¿Text of description 2.
¡/DL¿
¡/HTML¿
```

The output corresponding to this document will depend on the viewing program used. Below you will find the result of viewing this document with two HTML viewing programs, namely the terminal oriented program www (Fig. **??** and the NCSA X-windows HTML-browser Xmosaic (Fig. **??**

Below we givew a tabular overview of some of the more important commands, and their LaTeX functional equivalent.

This is a simple HTML document

AN EXAMPLE OF A LEVEL ONE HEADING

This is the first paragraph of level one.

An example of a second level heading

This is the first paragraph of level two.
Another paragraph of level two with a list.

This item contains an anchor definition.
This item the second item in the list.

And here follows an ordered list.
This item references the previous anchor[1].
Another item in the list.

Let me show you a definition list:

Term 1 Description 1.
Term 2 Description 2.

[End]
1, Quit, or Help:

Figure 5.2: The output of the example HTML source shown by the `www` program

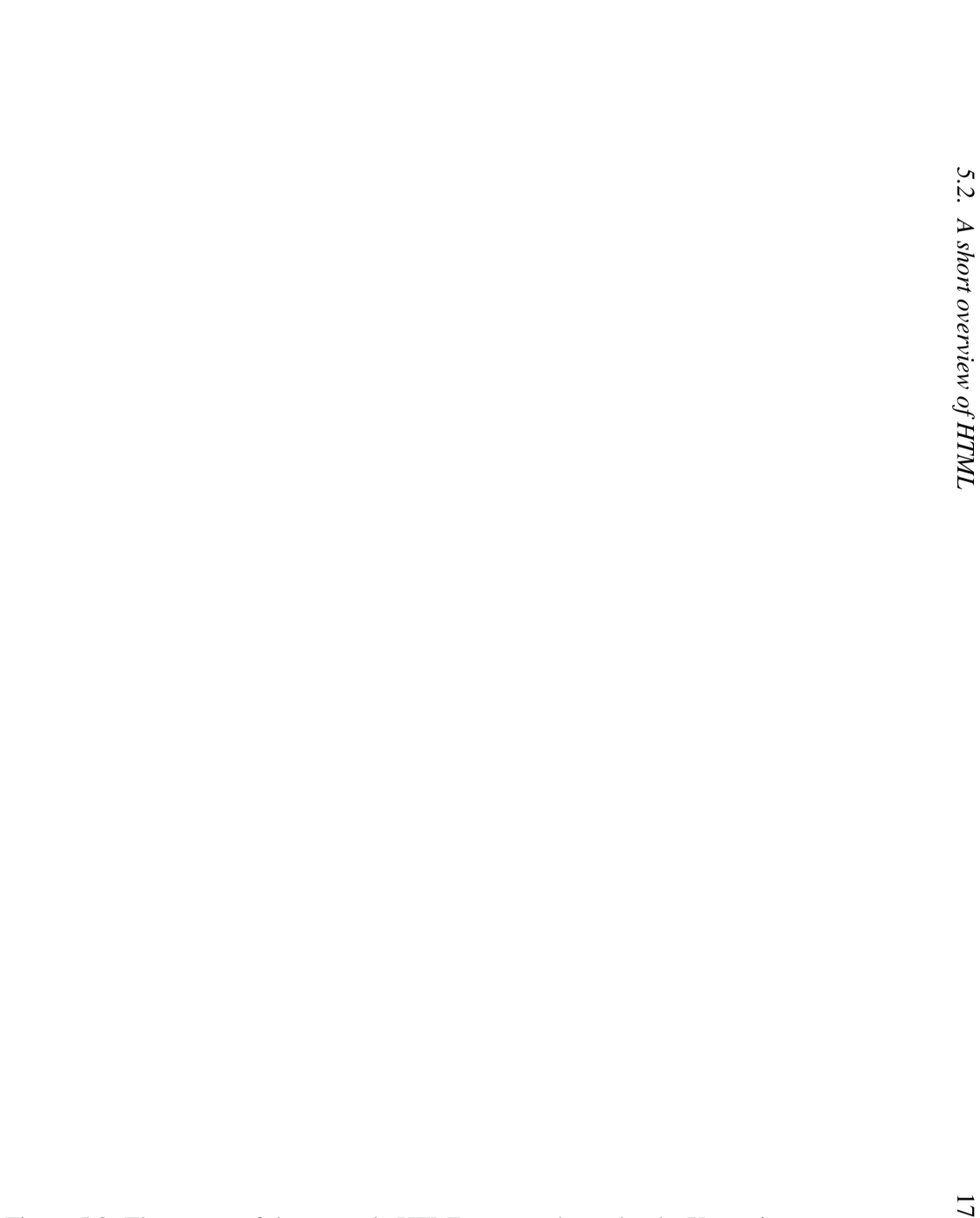

Figure 5.3: The output of the example HTML source shown by the Xmosaic program

Description	**HTML**	**LaTeX**
Headings	`<H1>...</H1>`	`\chapter{...}`
	`<H2>...</H2>`	`\section{...}`
	`<H3>...</H3>`	`\subsection{...}`
	`<H4>...</H4>`	`\subsubsection{...}`
	`<H5>...</H5>`	`\paragraph{...}`
	`<H6>...</H6>`	`\subparagraph{...}`
Lists	`OL`	`\begin{enumerate}`
	`<LI>`	`\item`
	`</OL>`	`\end{enumerate}`
	`UL`	`\begin{itemize}`
	`<LI>`	`\item`
	`</UL>`	`\end{itemize}`
	`DL`	`\begin{description}`
	`<DT>...<DD>`	`\item[...]`
	`</DL>`	`\end{description}`
Specific markup	`<P>`	`\par`
	`<PRE>...</PRE>`	`\begin{XMP}...\end{XMP}`
	`<EM>...</EM>`	`\textem{...}`
	`<SAMP>...</SAMP>`	`\Lit{...}`
	`<KBD>...</KBD>`	`\Ucom{...}`

5.2.2 Using hypertext link: defining and using anchors

An anchor is a piece of text which marks the beginning and/or the end of a hypertext link.

The most two most important attributes of an anchor are:

¡**A** HREF=*link-name* NAME=*link-def* ¿anchor text¡**/A**¿

The text between the opening tag `<A>` and the closing tag `</A>` is either the start or destination (or both) of a link.

The definition of these fields is:

`HREF` If the `HREF` attribute is present, the anchor is sensitive text, i.e., the start of a link. If you select this text, you should be presented with another document whose network address (URL, or "Universal Resource Locator") is defined by the value of the `HREF` attribute . If you specify a form like `HREF="#identifier"` the you are referring to another anchor in the same document. Otherwise the anchor points to another document, and the attribute is a relative name , relative to the documents address. An address like `HREF="docname#anchor"` specified an anchore inside the document "`docname`"

`NAME` If present, the attribute `NAME` allows the anchor to be the destination of a link. The value of the attribute is an unique arbitrary string identifying the anchor. Another document can then reference this anchor (using the `HREF` attribute, by putting the identifier after the document address, separated by a hash sign .

Examples of anchors

See ¡A HREF=”http://info.cern.ch/”¿CERN¡/A¿’s information for more details.

A ¡A NAME=WSdef¿workstation¡/A¿ is a personal computer with a bitmap graphics screen, enough memory, computer power and disk space ...

Programmer productivity is increased when using a ¡a href=”#WSdef”¿workstation¡/a¿.

Embedded images

With HTML and viewers, which support the functionality, like Xmosaic, you can embed graphics images in your hypertext documents.

You should use the IMG tag to include a small graphic image or an icon.

¡**IMG** SRC=*graphURL* ALIGN=*aligment*¿

SRC This is the URL of the document to be embedded. It has a syntax similar to that of the HREF attribute of the <A> tag. The attribute is mandatory

ALIGN The optional attribute specifies the vertical alignment of the graphics with respect to the baseline of the surrounding text. Possible values are BOTTOM, MIDDLE and TOP.

You can use <IMG> tags inside anchors.

Usually you will specify gif images for embedded graphics. Figure **??** shows the input and the resulting output of a small HTML file containing a gif image vertically positioned in various ways. In section **??** I shall explain how to generate gif image from PostScript files.

5.3 Using ptd

The program ptd converts a .dvi file into a plain text file. It uses a slightly different approach from the popular dvi2tty program. Instead of trying to position text without overlapping, ptd simply puts all characters into a two-dimensional grid (chars vs lines).

A special run of LaTeX (i.e. using a dedicated style) is needed to arrange the formatted text in a proper way. This style (cerndoc.sty) not only maps all fonts onto the typewriter font, but it also calculates the grid steps and adjusts all dimensions in accordance with these steps. A few macros (e.g. \underline, \verb are redefined too.

The ptd program is primarily designed to generate on-line documentation from LaTeX sources. It has also been successfully used for the conversion of LaTeX mark-up to other tagging schemes, like SGML or HTML (for WWW).

The calling sequence of the program is:

```
ptd [-dfpv]  file.dvi outfile
```

The parameters have the following meaning:

-d debugging option;

-f show font parameters;

-p show preamble/postamble parameters;

-v suppress warnings about replaced characters.

[Viewing the document with Xmosaic]

```
<TITLE>Looking at images</TITLE>
<P>Text around CERN Logo
   <IMG SRC="cernlogo.gif">
   Text around CERN Logo
<P>xxxxx <IMG ALIGN=BOTTOM
        SRC="cernlogo.gif">
   xxxxx <IMG ALIGN=TOP
        SRC="cernlogo.gif">
   xxxxx
```

[Input HTML document]

Figure 5.4: Including gif images in HTML documents

As a design feature, the ptd program will overwrite text. For example, when you reduce the \baselineskip parameter inside a document, you will get a lot of warnings about characters being replaced. The debug option is useful to locate where these replacements occur. With this option active, the ptd program prints a few lines in front and following the point where the replacement occurs. Note however, that the program does not care about the replacement of non-alphabetic characters, such as hyphens used for rendering rules.

5.4 Supported \special commands

\special{txtout: *argument*}

This \special command recognizes the following fields:

H horizontal step for the grid (in sp);

V vertical step for the grid (in sp);

G global offset (in characters);

F form feed flag.

Usually this command is specified once before the first output. The first two parameters are calculated by the style file cerndoc.sty and supplied to ptd via the following commands:

```
\newdimen\hquant   \newdimen\vquant
% the width of each letter in the font
\hquant=\the\fontdimen2\the\font
\ifcase \@ptsize
      \vquant=12pt
  \or \vquant=14pt
  \or \vquant=16pt
\fi
\newcount\along@h  \newcount\along@v
\along@h=\hquant   \along@v=\vquant
\special{textout: H:\the\along@h\space V:\the\along@v\space}
```

The global offset is used as horizontal shift for the output (by default there is no shift), e.g.

```
\special{textout: G:2} % shift everything two positions to the left
```

If the form feed flag is present no form-feed character will be output between pages (this character is output by default).

```
\special{textout: F:} % Don't separate pages by form feed
```

\special{hidetext: *argument*}

This kind of \special command inserts the string *argument* into the output file without affecting the formatted text. It is very useful for embedding "hidden" commands, to be used e.g. by a (hypertext) browser.
Assuming you need to specify an anchor which has "hot text" (visible) and a link (invisible), then you can (e.g. using WWW's HTML syntax) define a command \anchor with two parameters: the hot text and the link.

```
\def\anchor#1#2{\special{hidetext:<A HREF=#2>}#1\special{hidetext:</a>}}
```

You can use this command in WWW to include anchors, and be sure that you do not affect the layout of the output page.

```
\anchor{My hot text}{http://info.cern.ch:80/mytext.html}
```

5.4.1 Creating gif files with gs and xv

Bibliography

[1] L. Lamport. *LaTeX A Document Preparation System*. Addison-Wesley, 1986.

Index