HiTEX

User Manual

Für Beatriz

Version 1.1 (Draft)

MARTIN RUCKERT *Munich University of Applied Sciences*

The author has taken care in the preparation of this document, but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Internet page `http://hint.userweb.mwn.de/hint/hitex.html` may contain current information, downloadable software, and news.

`ruckert@cs.hm.edu`

Date: Mon Dec 4 15:00:09 2023

Contents

1 Introduction

When I started the HINT project in 2017, I tried to keep the project as small as possible to increase the chances that I would be able to complete it. So one design decision was to keep things simple—or to quote Albert Einstein: "Make things as simple as possible, but not simpler". The other imperative was: keep things out of the viewer if possible because I do not know how much processing power or battery power is available.

As a consequence, I focused on Donald Knuth' original TEX, disregarding all later extensions like ε-TEX or LATEX, and I decided that the TEX interpreter would not need to run in the viewer. Of course TEX's line breaking routine will run in the viewer and modifications of TEX's page breaking routine. But the decision to keep the TeX interpreter out of the HINT viewer implies that HINT files do not contain token lists and that there are neither output routines nor marks. To replace them, the HINT file format includes page templates. I have described the technical means to specify page templates below and try to explain the rationale behind it, but HINT's page templates are at the time of this writing a largely untested area.

By now, the state of the HINT project is far beyond of what I had expected then, and the processing power of even low-cost mobile devices is far better than expected especially when programming the graphics card directly using OpenEGL.

The following sections will describe all the primitive control sequences that are special for HiTEX. I tried to be as close to similar primitives that have proven to be useful in other engines, notably pdftex, to make it easy for package writers to support the HiTEX engine.

While currently HiTEX is the only TEX engine that supports output in the HINT file format, this might not be so forever. To avoid unnecessary complications for package writers, it is strongly suggested that all such TEX engines implement the same primitives according to the same specification. The following is the first draft of this specification. All the primitives use `HINT` as a prefix to avoid name conflicts. The prefix `HINT`, as opposed to e.g. `HiTeX`, was chosen to stress the idea that these primitives are specific for the output format—not for the TEX engine.

It is common practice in other TEX engines to support the `\special` primitive to insert raw code snippets in the output. Using this primitive, it is possible to insert PostScript code into a PS file, or PDF code in a PDF output file. It is currently not planed to support this mechanism for HINT output files for two reasons: First, the development of HiTEX is closely related to the development of the HINT file format and therefore features that are part of the HINT file format will enjoy support in HiTEX by corresponding primitives. Everything that is not available

through primitives in HiTEX should be considered "internal" and might change in the future. Second, HiTEX is not considered a replacement but a supplement to other engines. If your aim is the production of a printed book, your will probably target one of the engines that produce PDF output. But if, on occassion, you want to read what you wrote on a computer screen, you might just use HiTEX to process your source file. At this point you do not want to write `\special` commands for the new target; you want HiTEX as a plug-in replacement for your main target engine, even if it is not completely faithful to your final printed book.

2 HiTEX primitives

Because this is the first specification that will reach a wider user base, it is reasonable to expect changes to occur in the future. Therefore it is recommended that these primitives should not be used directly in a TEX document; instead they should be encapsulated in suitable macros so that the consequences of any change to the primitives will be local to these macros.

2.1 Syntax Description

In the following, we describe the syntax of primitive control sequences which were added to TEX.

- We use a `typewriter font` for text that occurs verbatim in the TEX document.
- We use *<italics>* enclosed in pointed brackets to denote symbols.
- We use rules to define the meaning of symbols. A rule starts with the symbol to be explained, followed by a colon "**:**", and then the text that this symbol stands for. A rule ends with a period "**.**".
- Optional parts of the rule's text are enclosed in [square brackets].
- Alternatives are separated by a vertical bar " | ".
- Some symbols refer to text that is defined as part of standard TEX. These are explained here by an example:

  <integer>`:` an integer as in `\penalty` *<integer>*`.`
  <normal dimension>`:` a dimension as in `\hrule width` *<normal dimension>*`.`
  <dimension>`:` a dimension as in `\vskip 0pt plus` *<dimension>*`.`
  <name>`:` a name as in `\input` *<name>*`.`
  <vertical list>`:` a token list with matching braces as in `\vbox{` *<vertical list>* `}.`
  <horizontal list>`:` a token list with matching braces as in `\hbox{` *<horizontal list>* `}.`
  <general text>`:` a token list with matching braces as in `\write{` *<general text>* `}.`

2.2 Version and Revision

The control sequences `\HINTversion` and `\HINTminorversion` are used to determine the major and minor version numbers of the HINT output format that is generated by HiTEX. It can be used as part of the output as in `\the\HINTversion`. The most important use, however, is testing whether the current TEX engine will in fact produce HINT output. As an example the file `ifhint.tex` contains the following code:

```
\newif\ifhint
\expandafter
  \ifx\csname HINTversion\endcsname\relax
  \hintfalse\else\hinttrue\fi
```

2.3 Images

The primitive `\HINTimage` includes an image in a document. The syntax is as follows:

`\HINTimage` [`=`] <*name*> [<*width*>] [<*height*>]

The optional equal sign can be added to make the code look nicer. The <*name*> specifies the image file. The width specification determines the width of the image. If omitted, HiTEX tries to determine the image's width from the image file. The same holds for the height specification.

<*width*>`:` `width` <*normal dimension*>`.`
<*height*>`:` `height` <*normal dimension*>`.`

Note that a <*normal dimension*> that is computed from `\hsize` or `\vsize` retains this dependency when processed by HiTEX. This allows an image to adapt to the size of the viewing area. Scaling in the HINT viewer will, however, never change the aspect ratio of an image. So it may become smaller or larger, but it will never be distorted. For this reason, HiTEX will inspect the image file to determine the aspect ratio of the stored image. The width and height values as given in the TEX file serve as the maximum values for the actual width and height. When rendering, the image will become as large as possible within the given bounds. If TEX does not specify neither width nor height, the image file must specify the absolute width and height of the image. It is considered an error if valid settings for the image's width and height can not be obtained.

2.4 Links, Labels, and Outlines

A link in a HINT document refers to another location in the same document. It can be used to navigate to that location. A link is defined using the primitives `\HINTstartlink` and `\HINTendlink`. Neither of them can be used in vertical mode. The text between the start and the end of the link constitutes the visible part of the link. Depending on the user interface, clicking or tapping or otherwise activating the link (e.g. pronouncing) will navigate to the destination of the link. The user interface might provide a visual clue to make the user aware of the available links

but it also may choose to leave the visual clues to the author of the document (e.g. using a special image or a special font).

The syntax is `\HINTstartlink` <*destination*> and `\HINTendlink` with

<*destination*>`:` `goto` <*label*>`.`
<*label*>`:` `name {` <*general text*>`}` | `num` <*integer*>`.`

As you can see, the link refers to its destination using a label which is either a name or a number. The destination can be defined by using the `\HINTdest` primitive. Forward and backward links are allowed; the definition of a label can either precede or follow the use of the label. If at the end of the document a label is undefined, a warning is given, and the label will reference the beginning of the document.

The syntax is `\HINTdest` <*label*> [<*placement*>] with

<*placement*>`:` `top` | `bot.`

The optional placement argument specifies how to build the page containing the destination location. `top` demands a page starting with the destination location. This is useful if the destination is for example the start of a section or chapter heading. Similarly `bot` asks for a page that ends with the destination location. The most common case is to omit the placement argument. In this case, the viewer will build a "good" page that includes the given destination. In case of a section heading, for example, it will most probably start the page with the section heading because section headings are usually preceded by a negative penalty that will convince the page builder that this is a good place to break the page. But if the section heading is immediately preceded by a chapter heading, the negative penalty found there will probably persuade the page builder to start with the chapter heading instead.

There is a special label that has the form `name {HINT.home}`. It is used to mark the "home page" of the document. User interfaces are encouraged to offer a button or keyboard shortcut to navigate to the document location labeled this way. The page should be a convenient starting point to explore the document. The typical place for this label would be the documents table of content.

The labels that identify destinations in a document can also be used to define document outlines. A document outline is a document summary given as a hierarchical list of headings where each of them refers to a specific location in the document.

The syntax is `\HINToutline` <*destination*> [<*depth*>] `{` <*horizontal list*>`}`.

<*depth*>`:` `depth` <*integer*>`.`

The user interface can format the <*horizontal list*> much like a `\hbox` would do and displays it to the user. When the user selects this text, the document will be repositioned to show the destination location in the same way as with a link. In order to support also simpler user interfaces, the current HINT backend also extracts the characters (and spaces) from the horizontal list (in top-left to bottom-right order) and makes this character string available to the user interface.

The order in which outline items are defined is important because this is the order in which they will be presented to the reader of the document. The optional depth argument allows to structure the list of outline items as a hierarchy. Outline items with a higher depth value are considered to be sub-items of items earlier in the list with lower depth values. If no depth value is given, the depth value is set to zero. It is not necessary to define depth values strictly consecutive.

2.5 Page Templates and Streams

To produce the final page, TEX uses a special piece of program called the output routine. Because a HINT file is pure data, it can not contain output routines. Instead it uses page templates to assemble pages from the main text, footnotes, floating illustrations, and other material. I start here by describing how HINT's page templates work and the special syntax used to specify them in a TEX file that is about to be processed with HiTEX. For those interested in how the design decision was made and how page templates relate to TEX's page building mechanism, a separate section follows at the end.

The syntax of a page template specification is: `\HINTsetpage` *<integer>* [`=`] *<name>* [*<priority>*] [*<width>*] [*<height>*] `{` *<vertical list>* *<stream definition list>* `}`

The *<integer>* specifies the page templates number in the range 1 to 255. The number 0 is reserved for the build in page template of the HINT file format, which is used if no other page template has been defined. The page template 0 can not be redefined. The *<name>* associates a name with the page template. The name can be displayed by the HINT viewer to help the user selecting a suitable page template.

After the name follows an optional priority; it is used to select the "best page template" if multiple templates are available. The default value is 1. The build-in template has priority 0.

<priority>`:` `priority` *<integer>*`.`

After that follows an optional width and height of the full page including the margins. After subtracting `\hsize` from the width and `\vsize` from the height, the remainder is used for the margins around the displayed text. For example giving the width as 1.2`\hsize` will leave 0.1`\hsize` for the margins on both sides. In this case the margins will grow together with the available width of the display. If the width is computed by adding 20pt to `\hsize`, the margin will be 10pt on both sides. In this case the margin will not grow with the size of the display, but it will grow with the magnification factor. Of course both methods can be used together. The default is `\hsize` for the width and `\vsize` for the height so there are no margins.

The following *<vertical list>* defines the page itself. It should assign suitable values to `\topskip` and `\maxdepth` because the values valid at the end of the vertical list are stored in the page template and are used in the page building process. The vertical list usually also specifies the insertion of content streams using a *<stream insert point>*.

<stream insert point>`:` `\HINTstream` *<integer>*`.`

Here the ⟨*integer*⟩ must be in the range 0 to 254. The value 255 is invalid; the value 0 indicates the main body of text (what TEX's page builder would normally put into box 255 before calling the output routine). Otherwise, the ⟨*integer*⟩ is TeX's insertion number, that is the number of TEX's box containing the insertions. As usual, this box is filled using TEX's `\insert` primitive. So after plain TEX has defined `\footins`, the footnotes for the current page can be inserted after the main body of text in the ⟨*vertical list*⟩ by saying `\HINTstream0` followed by `\HINTstream\footins`. Of course you might want to have a footnote rule and a small skip to separate the footnotes —if there are any—from the main text. This can be achieved by a suitable ⟨*stream definition*⟩ in the ⟨*stream definition list*⟩ .

⟨*stream definition list*⟩`:` | ⟨*stream definition list*⟩ ⟨*stream definition*⟩`.`
⟨*stream definition*⟩`:` `\HINTsetstream` ⟨*integer*⟩ [`=`] [`preferred` ⟨*integer*⟩] [`next` ⟨*integer*⟩] [`ratio` ⟨*integer*⟩] `{` ⟨*vertical list*⟩ `}.`

The first ⟨*integer*⟩ is the streams insertion number i, and it must match the ⟨*integer*⟩ previously used in the ⟨*stream insert point*⟩ . Then follows the optional specification of a preferred stream with insertion number p, a next stream with insertion number n, and a split ratio r. If $r > 0$, the contributions to stream i are split between stream p and n in the ratio $r/1000$ for p and $1 - r/1000$ for n before contributing streams p and r to the page. Else if $p \geq 0$ any insertion to stream i is moved to stream p as long as possible, and if $n \geq 0$ we move an insert to stream n if there is "no room left" in p nor in i. How much "room" is available for the insertions is specified inside the vertical list that follows. Here `\dimen`i should be set to the maximum total height of the insertions in class i per page. `\count`i should be set to the magnification factor f, such that inserting a box of height h will contribute $h * f/1000$ to the main page; and `\skip`i should be set to the extra space needed if an insertion in class i is present.

This extra space is usually taken up by material that is inserted before and after the insertions, such as for example the footnote rule. This material can be defined by a ⟨*before list*⟩ and an ⟨*after list*⟩ .

⟨*before list*⟩`:` `\HINTbefore` [`=`] `{` ⟨*vertical list*⟩ `}.`
⟨*after list*⟩`:` `\HINTafter` [`=`] `{` ⟨*vertical list*⟩ `}.`

If you are interested in the design decision that motivate the definitions that have be given in this section, you should read section 4.

3 Other Primitives

Since I consider the support for LaTeX to be crucial for the success of the HINT project, quite a few primitives have been added to HiTeX that go beyond TeX's original specification.

3.1 ε-TeX

First, the primitives of ε-TeX have been added with the exception of those primitives that deal with line breaking, with right to left reading, and with marks. Here is a list of ε-TeX primitives that are missing in HiTeX:

- `\TeXXeTstate` (current reading direction)
- `\beginL`, `\endL` (switching reading direction)
- `\beginR`, `\endR` (switching reading direction)
- `\predisplaydirection` (reading direction)
- `\lastlinefit` (line breaking)
- `\marks` (multiple marks)
- `\botmarks`, `\splitbotmarks` (multiple marks)
- `\firstmarks`, `\splitfirstmarks` (multiple marks)
- `\topmarks` (multiple marks)

3.2 LaTeX and PRóTE

Second, the primitives required to support LaTeX were added using Thierry Laronde's implementation of PRóTE.

- `\Proteversion`, `\Proterevision` (version information)
- `\resettimer`, `\elapsedtime` (timing information)
- `\creationdate`, `\filemoddate`, `\filesize`, `\filedump`, `\mdfivesum` (file information)
- `\shellescape` (Currently only a dummy implementation.)
- `\setrandomseed`, `\randomseed`, `\normaldeviate`, `\uniformdeviate` (random numbers)
- `\expanddepth`, `\expanded` (programming)
- `\ifincsname`, `\ifprimitive` `\primitive` (programming)
- `\savepos`, `\lastxpos`, `\lastypos`, `\pageheight`, `\pagewidth` (Only dummy implementations since this information is not available to HiTeX at runtime.)
- `\strcmp` (comparing strings)

3.3 `kpathsearch` and `\input`

In Don Knuths implementation of TEX, the `\input` primitive will add the extension `.tex` to any filename that does not have an extension. This implies that a file without extension can not be opened as an input file. The usual engines do not add such an extension but pass the filename as given to `kpse_find_file` function. HiTEX does the same. The `kpathsearch` library will find files in a variety of directories and yes, it will also find files without extension. Using this library is just mandatory for any engine that wants to process LATEX input.

4 Replacing TEX's Page Builder

TEX uses an output routine to finalize the page. The output outline takes the material which the page builder had accumulated in `box255` and attaches headers, footers, and floating material like figures, tables, and footnotes. The latter material is specified by insert nodes while headers and footers are often constructed using mark nodes. Running an output routine requires the full power of the TEX engine and is not part of the HINT viewer. Therefore, HINT replaces output routines by page templates. TEX can use different output routines for different parts of a book—for example the index might use a different output routine than the main body of text.

TEX uses insertions to describe floating content that is not necessarily displayed where it is specified. Three examples may illustrate this:

- Footnotes* are specified in the middle of the text but are displayed at the bottom of the page. Several footnotes on the same page are collected and displayed together. The page layout may specify a short rule to separate footnotes from the main text, and if there are many short footnotes, it may use two columns to display them. In extreme cases, the page layout may demand a long footnote to be split and continued on the next page.
- Illustrations may be displayed exactly where specified if there is enough room on the page, but may move to the top of the page, the bottom of the page, the top of next page, or a separate page at the end of the chapter.
- Margin notes are displayed in the margin on the same page starting at the top of the margin.

HINT uses page templates and content streams to achieve similar effects. But before I describe the page building mechanisms of HINT, let me summarize TEX's page builder.

4.1 TEX's page building mechanism

TEX's page builder ignores leading glue, kern, and penalty nodes until the first box or rule node is encountered; whatsit nodes do not really contribute anything* to a page; mark nodes are recorded for later use. Once the first box, rule, or insert arrives, TEX makes copies of all parameters that influence the page building process and uses these copies. These parameters are the `page_goal` and the `page_max_depth`. Further, the variables `page_total`, `page_shrink`, `page_stretch`,

* Like this one.

* This changes when images are implemented as whatsit nodes.

page_depth, and insert_penalties are initialized to zero. The top skip adjustment is made when the first box or rule arrives—possibly after an insert. Now the page builder accumulates material: normal material goes into box255 and will change page_total, page_shrink, page_stretch, and page_depth. The latter is adjusted so that is does not exceed page_max_depth.

The handling of inserts is more complex. TEX creates an insert class using newinsert. This reserves a number i and four registers: box$i$ for the inserted material, count$i$ for the magnification factor f, dimen$i$ for the maximum size per page d, and skip$i$ for the extra space needed on a page if there are any insertions of class i.

For example plain TEX allocates $n = 254$ for footnotes and sets count254 to 1000, dimen254 to 8in, and skip254 to \bigskipamount.

An insertion node will specify the insertion class i, some vertical material, its natural height plus depth x, a split_top_skip, a split_max_depth, and a floating_penalty.

Now assume that an insert node with subtype 254 arrives at the page builder. If this is the first such insert, TEX will decrease the page_goal by the width of skip254 and adds its stretchability and shrinkability to the total stretchability and shrinkability of the page. Later, the output routine will add some space and the footnote rule to fill just that much space and add just that much shrinkability and stretchability to the page. Then TEX will normally add the vertical material in the insert node to box254 and decrease the page_goal by $x \times f/1000$.

Special processing is required if TEX detects that there is not enough space on the current page to accommodate the complete insertion. If already a previous insert did not fit on the page, simply the floating_penalty as given in the insert node is added to the total insert_penalties. Otherwise TEX will test that the total natural height plus depth of box254 including x does not exceed the maximum size d and that the page_total + page_depth + $x \times f/1000$ − page_shrink ≤ page_goal. If one of these tests fails, the current insertion is split in such a way as to make the size of the remaining insertions just pass the tests just stated.

Whenever a glue node, or penalty node, or a kern node that is followed by glue arrives at the page builder, it rates the current position as a possible end of the page based on the shrinkability of the page and the difference between page_total and page_goal. As the page fills, the page breaks tend to become better and better until the page starts to get overfull and the page breaks get worse and worse until they reach the point where they become awful_bad. At that point, the page builder returns to the best page break found so far and fires up the output routine.

4.2 HINT Page Templates

Let's look at the problems that show up when implementing a replacement for TEX's page building mechanism.

1. An insertion node can not always specify its height x because insertions may contain paragraphs that need to be broken in lines and the height of a paragraph depends in some non obvious way on its width.
2. Before the viewer can compute the height x, it needs to know the width of the

insertion. Just imagine displaying footnotes in two columns or setting notes in the margin. Knowing the width, it can pack the vertical material and derive its height and depth.

3. TEX's plain format provides an insert macro that checks whether there is still space on the current page, and if so, it creates a contribution to the main text body, otherwise it creates a `topinsert`. Such a decision needs to be postponed to the HINT viewer.
4. HINT has no output routines that would specify something like the space and the rule preceding the footnote.
5. TEX's output routines have the ability to inspect the content of the boxes, split them, and distribute the content over the page. For example, the output routine for an index set in two column format might expect a box containing index entries up to a height of $2 \times$ `vsize`. It will split this box in the middle and display the top part in the left column and the bottom part in the right column. With this approach, the last page will show two partly filled columns of about equal size.
6. HINT has no mark nodes that could be used to create page headers or footers. Marks, like output routines, contain token lists and need the full TEX interpreter for processing them. Hence, HINT does not support mark nodes.

Instead of output routines, HINT uses page templates. Page templates are basically vertical boxes with *<stream insert points>* marking the positions where the content of the box registers, filled by the page builder, should appear. To output the page, the viewer traverses the page template, replaces the placeholders by the appropriate box content, and sets the glue.

It is only natural to treat the page's main body, inserts, and marks using the same mechanism. We call this mechanism a content stream. Content streams are identified by a stream number in the range 0 to 254; the number 255 is used to indicate an invalid stream number. The stream number 0 is reserved for the main content stream; it is always defined.

> It is planed to implement a replacement for TEX's mark nodes using different types of streams:
>
> - normal streams correspond to TEX's inserts and accumulate content on the page,
> - first streams correspond to TEX's first marks and will contain only the first insertion of the page,
> - last streams correspond to TEX's bottom marks and will contain only the last insertion of the page, and
> - top streams correspond to TEX's top marks. Top streams are not yet implemented.

Nodes from the content section are considered contributions to stream 0 except for insert nodes which will specify the stream number explicitly. If the stream is not defined or is not used in the current page template, its content is simply ignored.

The page builder needs a mechanism to redirect contributions from one content stream to another content stream based on the availability of space. Hence a HINT content stream can optionally specify a preferred stream number, where content should go if there is still space available, a next stream number, where content should go if the present stream has no more space available, and a split ratio if the content is to be split between these two streams before filling in the template.

Various stream parameters govern the treatment of contributions to the stream and the page building process.

- The magnification factor f: Inserting a box of height h to this stream will contribute $h \times f/1000$ to the height of the page under construction. For example, a stream that uses a two column format will have an f value of 500; a stream that specifies notes that will be displayed in the page margin will have an f value of zero.
- The height h: The extended dimension h gives the maximum height this stream is allowed to occupy on the current page. To continue the previous example, a stream that will be split into two columns will have $h = 2 \cdot$ `vsize` , and a stream that specifies notes that will be displayed in the page margin will have $h = 1 \cdot$ `vsize`. You can restrict the amount of space occupied by footnotes to the bottom quarter by setting the corresponding h value to $h = 0.25 \cdot$ `vsize`.
- The depth d: The dimension d gives the maximum depth this stream is allowed to have after formatting.
- The width w: The extended dimension w gives the width of this stream when formatting its content. For example margin notes should have the width of the margin less some surrounding space.
- The "before" list b: If there are any contributions to this stream on the current page, the material in list b is inserted *before* the material from the stream itself. For example, the short line that separates the footnotes from the main page will go, together with some surrounding space, into the list b.
- The top skip glue g: This glue is inserted between the material from list b and the first box of the stream, reduced by the height of the first box. Hence it specifies the distance between the material in b and the first baseline of the stream content.
- The "after" list a: The list a is treated like list b but its material is placed *after* the material from the stream itself.
- The "preferred" stream number p: If $p \neq 255$, it is the number of the *preferred* stream. If stream p has still enough room to accommodate the current contribution, move the contribution to stream p, otherwise keep it. For example, you can move an illustration to the main content stream, provided there is still enough space for it on the current page, by setting $p = 0$.
- The "next" stream number n: If $n \neq 255$, it is the number of the *next* stream. If a contribution can not be accommodated in stream p nor in the current stream, treat it as an insertion to stream n. For example, you can move contributions to the next column after the first column is full, or move illustrations to a separate page at the end of the chapter.

- The split ratio r: If r is positive, both p and n must be valid stream numbers and contents is not immediately moved to stream p or n as described before. Instead the content is kept in the stream itself until the current page is complete. Then, before inserting the streams into the page template, the content of this stream is formatted as a vertical box, the vertical box is split into a top fraction and a bottom fraction in the ratio $r/1000$ for the top and $(1000 - r)/1000$ for the bottom, and finally the top fraction is moved to stream p and the bottom fraction to stream n. You can use this feature for example to implement footnotes arranged in two columns of about equal size. By collecting all the footnotes in one stream and then splitting the footnotes with $r = 500$ before placing them on the page into a right and left column. Even three or more columns can be implemented by cascades of streams using this mechanism.

HINT allows multiple page templates but HiTEX currently does not implement restricting them to individual page ranges and the viewer selects the page template with the highest priority. To support different output media, the page templates are named and a suitable user interface may offer the user a selection of possible page layouts. In this way, the page layout remains in the hands of the book designer, and the user has still the opportunity to pick a layout that best fits the display device.

The build-in page template with number 0 is always defined and has priority 0. It will display just the main content stream. It puts a small margin of `hsize`$/8-4.5$pt all around it. Given a letter size page, 8.5 inch wide, this formula yields a margin of 1 inch, matching TEX's plain format. The margin will be positive as long as the page is wider than 1/2 inch. For narrower pages, there will be no margin at all. In general, the HINT viewer will never set `hsize` larger than the width of the page and `vsize` larger than its height.

Index

P

R

S

T

V

W