Package ACCENTBX: some problems with accents in TEX and how to solve them

A.S. Berdnikov

Institute of Analytical Instrumentation
Rizskii pr. 26
198103 St.Petersburg
Russia
berd@ianin.spb.su or
berdn@ptslab.ioffe.rssi.ru

Abstract

Weak points of TEX internal command `\accent` are investigated. It appears that some features of `\accent` prevent to construct specific letters which require several accents simultaneously. Special macro which emulates the work of `\accent`, additional macro for fine tuning of accents, if-then-else structures which enable to put correct accents depending on uppercase/lowercase state and font family conditions, etc., are described. Closely related to it are the accents for I/i and J/j which are different for uppercase and lowercase letters, and similar problem occurs for so-called *title letters*. The whole package is coded in such a way that in works for *Plain*, $\mathcal{AMS}$TEX, etc., as well, not only for LATEX 2_ε.

1. Introduction

The nearly inevitable feature of cyrillic encodings is that some cyrillic letters (for example, the majority of accented letters) should be assembled from pieces — accents, modifiers, composite elements, etc. There are the cases (not too rare) where *several* elements should be attached to the same base character. The typical examples are the stresses over accented letters, some specific characters from Nivh language, multi-level accents specific for Saam language, stroke-type modifiers added to some letters, and so on — some of these constructions are shown here:

Ё́, И́, Ҫ̋, ӿ̇, Ӧ̄, $\overline{\ddot{\text{Э}}\text{А}}$, ...

In addition there are the cases where the accent is positioned *between* two characters — for example, in German [1] there is the abbreviation -*bǧ* which substitutes -*burg* when necessary[1]. Such effect could be simulated by a special invisible character with zero width and corresponding height as it is described in the documentation for DC/EC fonts [1], but invisible rule with zero width or `\phantom` command are better for this purpose — unfortunately `\accent` cannot place accents over them.

Another problem arises when we need the accent above the letters I/i or J/j — these letters contain dots for the lowercase form but do not contain it for the uppercase form. It is not a problem when there is no automatic uppercase and lowercase conversion of the user defined text since *dottless* lowercase forms are available in Computer Modern typefaces — but, for example, the Khakassian uppercase 'İ' could be unexpectedly transformed into double-dotted 'i' after `\lowercase` used somethere:

$$\.I = \dot{\mathrm{I}} \quad \uppercase{\.I} = \dot{\mathrm{I}} \quad \lowercase{\.I} = \mathrm{i}$$

Summary: the primitive `\accent` command cannot be used for such constructions (it cannot analyze uppercase/lowercase characters, it puts accents only over the letters and atomic symbols ignoring the composite characters like `\hbox` or `\vrule` and it cannot put accent over already accented characters). To overcome these restrictions and to support inside one encoding as much cyrillic writing systems as possible, it is necessary to simulate the work of the command `\accent` by some other TEX tools.

Finally, there is some problem which does not relate directly to accents — This is the problem of so-called *title letters* which are not supported by TEX when the commands `\lowercase` and `\uppercase` work. Although it is not the problem with `\accent` (and hence corresponding macro are not included into ACCENTBX), it closely relates to the problem with accented I/i and J/j — so, this problem and its solution is discussed in section 10.1.

---

1 In first version of this paper published in Proceedings of EuroTEX-97 there was an error: it states that sometimes in Serbian language the accents *acute*, *grave*, *double-grave* and *frown* (reverted breve) which are the specific Serbian stresses, may be positioned in a similar manner: бр̋вно, мр̀твац, мр̆ква, зр̆тва, ... More detailed investigation shows that in such cases corresponding stresses are placed strictly over the letter 'р'. But in Church Slavonic and Old Slavonic the accents *paerok* and *titlo-in-letters* placed between two letters are not a rare case.

2. How \accent works

Before we proceed to simulate the work of `\accent` it may be a good idea to see what particular operations are performed by this internal TEX command. Its syntaxis is specified as [2, 3]:

`\accent`⟨*8-bit number*⟩ ⟨*optional assignments*⟩ ⟨*character*⟩

Character is a letter over which the accent should be placed. The 8-bit number is the code from the current font which corresponds to the character used as the accent. The optional assignments between the *8-bit number* and the *character* could contain, in particular, the font switching commands which enable to take the accent defined by *8-bit number* and the accented object defined by *character* from different fonts (although it cannot contain `\setbox` assignments).

If the assignments are not followed by a *character* (where *character* is the token with `\catcode=11` or `\catcode=12`, command `\char`, TEX command created through `\chardef`, `\noboundary` command), TEX considers `\accent` as the equivalent to `\char`⟨*8-bit number*⟩ command. Otherwise the character which follows the assignment is accented by the character that corresponds to *8-bit number*. The accenting algorithm is the following[2]:

1. If the *character* has a height equal to `\fontdimen5` for the font where the accent is selected (which is `1ex` for this font), the accent is centered horizontally over the box corresponding to the *character* and no additional vertical shift is inserted.

2. If the *character* has a height unequal to `1ex` and the font is not slanted, i.e., the slant `\fontdimen1` for the font where the accent is selected is equal to `0pt`, the accent `\char`*8-bit number* is placed in a `\hbox` which is vertically shifted by the difference of the height of the *character* and `1ex`. The accent box is shifted up, if the *character* is heigher than `1ex`, otherwise it is shifted down. It is also centered horizontally to the *character* box.

3. If the *character* has a height unequal to `1ex` and the font is slanted, i.e., the value `\fontdimen1` for the font where the accent is selected is not equal to `0pt`, the accent `\char`*8-bit number* is placed in a `\hbox` and vertically shifted as described under 2.

   Horizontally the center of the accent box is displaced from the center of the *character* by the absolute amount of the horizontal shift amount

[2] This description with minor corrections belongs to Bernd Raichle and was extracted by him through investigation of the original Web code for TEX.

multiplied by the font slant `\fontdimen1` (which is given as *slant-per-pt*). It is displaced to the right, either if the box is shifted up and the font slant is positive (i.e., the glyphs of the font are slanted to the right), or if the box is shifted down and the font slant is negative, otherwise the accent box is moved to the left.

Such additional horizontal shift proportional to slant value is an essential component of TEX accenting algorithm since it enables to position the accent properly over the slanted character proportionally to its height.

The proper positioning of the accent box over the accented character is performed by the explicit kerns inserted by TEX. Kerns are inserted in such a manner that the horizontal size of the accented character is not affected. On the contrary, the vertical size of the accented character is corrected following the height of the resulting construction.

The fact that the command `\accent` inserts explicit kerns prevents TEX from using its automatic hyphenation algorithm, as it usually happens for the words with explicitly inserted kerns. It is essential that the commands for the correct alignment of the accent with respect to the accented character are inserted by TEX in the output stream at first — i.e, *before* the main character. If the italic correction command `\/` is placed after the accented character and if it is separated from the main character by some kerns, `\hbox` command, etc., it fails to define properly the *italic correction* shift.

3. How new `\accent` may be implemented

By definition the command `\accent` can put accents only over the atomic letters and symbols. In particular, it cannot put an accent over already accented character (at least in text mode). The other disadvantage is that accent and accented character should be taken usually from the same font. Although the syntaxis of `\accent` enables to use the font switching commands, this possibility is realized not in a comfortable mode. Usually we would like to select the accent from a special font, and the main character — from the current font. Inspite of it the `\accent` syntaxis forces to store and to reconstruct explicitly the current font status when the accent is selected not from the current font.

Some simple macro emulating the work of `\accent` and enabling to overcome its restrictions are suggested in [4][3]:

```
\def\ifnnull#1{\def\inner{#1}\ifx\inner\empty\else}
```

[3] The original commands are modified a little to fit the purposes of this paper.

```
\def\genaccent#1#2#3{%
   \leavevmode\setbox0=\hbox{#3}%
   \vbox{\offinterlineskip
         \ifnnull{#1}\hbox to \wd0{\hss#1\hss}\kern 0.2ex\fi
         \vbox to \ht0{\copy0\vss}%
         \vtop{\null\vbox to \dp0{\vss}%
           \ifnnull{#2}\kern 0.2ex \hbox to \wd0{\hss#2\hss}\fi
         }%
   }%
}
```

It enables to put above and below an arbitrary TEX object as many accents as you wish (defined as `\hbox` if it is necessary). It is also not a problem to select the accents used for this purpose from a separate font without taking care about the current font status. Unfortunately these commands do not take into account the fine accent tuning for slanted fonts, and they also break down the work of italic correction `\/` (the main character is deeply hidden inside `\vbox`).

This scheme could be modified. Suppose the accent is defined as the first parameter of some macro, and the main symbol to be accented – as its second parameter. The following commands

```
\setbox0=\hbox{#2}
\setbox2=\hbox to \wd0{\hss#1\hss}
\raise\ht0\box2\kern-\wd0 \unhbox0
```

do the necessary work:[4]

— put the accent above the main character,

— align it properly in horizontal direction (at least for straight fonts),

— does not change the width of the main character while the height of the composite construction is corrected properly,

— leaves the main character as the last object in the output stream so that the italic correction command `\/` can have a free access to it,

— enables to use arbitrary TEX constructions for accents and accented characters,

[4] Here the command `\unhbox` is used instead of `#2` following the recommendations by Bernd Raichle to avoid the side effects if there are assignments inside the argument `#2`.

Case 1:	Case 2:

`\dimenA=...`
`\dimenB=...`
`\dimenC=\dimenA`
`\countX=\dimenB`
`\multiply\dimenC by \countX`
`\divide\dimenC by 65536`
	`\dimenA=...`
	`\dimenB=...`
	`\dimenC=\dimenA`
	`\countX=\dimenB`
	`\divide\dimenC by 65536`
	`\multiply\dimenC by \countX`

Figure 1: How to multiply $2pt \times 2pt \to 4pt$ in TEX

— conserves the kerning with the characters followed after the accented character (unfortunately the kerning with the *previous* character is destroyed irreversibly).

The only thing to do is to add the fine horizontal tuning which is not a problem as soon as we can calculate it. But the calculation of the horizontal shift necessary for slanted fonts *is* a problem and requires additional tricks! It is necessary to multiply slant value `\fontdimen1` by the vertical shift of the accent, but the problem is that TEX contains *no* primitive tools to multiply `2pt` by `2pt` and to get `4pt` as a result (at least if the multiplied values are the `\dimen` registers).

One way to do such operation is to convert one value to an integer (assuming $1pt$=65536) and perform multiplication as it is shown on Fig. 1. Unfortunately the accuracy of such algorithm is very low: in the first case the overflow could occur after `\multiply\dimenC by \countX` if `\dimenA` is too big; in the second case the result after `\divide\dimenC` by 65536 could be zero id `\dimenA` is too small. Some special macro exist [5, 6, 7, 8] which perform floating point calculations in TEX with high accuracy. Hopefully we do not need such intensive calculations. The macro `\slant@value` which converts the `\fontdimen1` specified for the current font into numerical string (like `\the` command, but without the suffix `pt` at its end) would be enough for our purpose:

```
\setbox0=\hbox{#2}
\setbox2=\hbox to \wd0{\hss#1\hss}
\dimen2=\slant@value\ht0
\kern\dimen2
   \raise\ht0\box2\kern-\wd0
\kern-\dimen2 #2
```

The macro which converts the `\fontdimen1` of the current font into text string without the suffix `pt` at its end can be coded, for example, as

```
\begingroup\catcode`P=12 \catcode`T=12
           \lowercase{\endgroup\def\strip@pt#1PT}{#1}}
\def\slant@value{\expandafter\strip@pt\the\fontdimen1\font}
```

(This trick is suggested by Bernd Raichle and is based on `\kslant` shown on p.375 of TEXbook.)

Other operations require a standard TEXnical work with boxws and does not cause serious problems from the designer. Unfortunately, such operations destroy automatic kerning and prevents automatic hyphenations (like ordinary command `\accent` does).

4. General remarks about accent commands

To create the "universal" set of accenting commants it is necessary to take into account the following factors:

1. Symbol used as accent could be (and should be) an ordinary `\hbox` — i.e., the User does not align it in advance with respect to the height of `1ex` as it is done for accent characters in CM fonts.

2. Since typical accents in CM fonts are shifted in advance verically by `1ex` and horizontally by corresponding slant correction, before we use these characters as accents we should compensate these shifts backward. Hence, the User should have at his/her disposal necessary macro.

3. Although typically accents are placed above the characters, there are cases where the modifiers are placed below the character, at the baseline of the character and over the character to make special composite objects. Hence, a variety of accenting commands is necessary.

4. Accents which are placed above the character or below the character may have additional vertical space between the accent and the character for fine tuning.

5. Accents which are the modifiers joined without space with the character may require to shift it closer to the character to make intersection well expressed.

6. Symbol like comma may require to raise it above the baseline before it can be used as the upper accent.

7. Accents can be placed before the letter at the beginning of the word, after the letter at the end of the word, or between two letters in the middle of the word, as in it explained in section 1.

8. Accents "*overline*" (*macron*) and "*underline*" may be longer than the width of one letter[5].

9. Accents may be justified with respect to the character in non-standard way — say, it can be aligned to the left corner or to the right corner of the character.

10. Pseudo accents like strokes, ets., artificially added to already existing characters, may require fine tuning of their position with respect to the body of the character. In particular, such tuning can depend on font style (italic, slanted, etc.).

11. It is necessary to take into account that the composite characters may be influenced by the commands `\uppercase` and `\lowercase` (or by equivalent LaTeX 2ε commands `\MakeUppercase` and `\MakeLowercase`). Accents should be stable (since these operations change the height and the width of the character) and robust (since accented characters may be placed inside section headers, etc., where — at least for LaTeX 2ε — fragile commands are not welcome).

12. Closely related to it is the problem with the accents for I/i and J/j: when converted by `\uppercase` and `\lowercase` these letters can get or loose the upper dot accent.

5. Some examples

Before we proceed further let me demonstrate some simple examples. (Although the commands included into ACCENTBX are rather lengthy, it is done intensionally: these commands are mainly designed for creating your own accent commands and specific characters available through macros, not for "everyday usage" inside your text.)

Suppose we would like to put a tiny letter 'a' as the accent over some character. Using the package ACCENTBX we create special command

```
\def\tA#1{\upaccent{\tiny a}{#1}}
```

[5] The Saam (Lappish) writing system is the only known up to now which has such modificators.

(`\upaccent` places one box as the accent over the second box). Now we can see how it works:

$\overset{\text{a}}{\text{X}}$ = `\tA{X}`	$\overset{\text{a}}{\text{X}}$ = `\tA X`
$\overset{\text{a}}{\text{x}}$ = `\tA{x}`	$\overset{\text{a}}{\text{x}}$ = `\lowercase{\tA X}`
$\overset{\text{a}}{\mathbf{X}}$ = `\bf\tA{X}`	$\overset{a}{X}$ = `\it\tA{X}`
$\overset{\text{a}}{\overset{\text{a}}{\text{X}}}$ = `\tA{\tA X}`	$\overset{\text{a}}{\overset{\text{a}}{\text{X}}}$ = `\tA{\tA{\tA X}}`
$\overset{a}{\overset{a}{X}}$ = `\it\tA{\tA X}`	$\overset{a}{\overset{a}{X}}$ = `\it\tA{\tA{\tA X}}`

If you would like to reproduce exactly what is typed above, you'll discover that the result is a little bit different: `\it\tA{\tA{X}}` gives $\overset{a}{\overset{a}{X}}$, not $\overset{a}{\overset{a}{X}}$. Yes, actually the previous example was defined as

```
\def\tA#1{\upaccent{\aboxsplit{\tiny a}}{#1}}
```

— since `\upaccent` just places a box over box, it is necessary to add some space around tiny 'a' to separate these boxes, and this is just what macro `\aboxsplit` does.

If you try to make ordinary TEX accents with `\upaccent`, you'll discover another essential feature:

```
\upaccent{\char"13}{X}=
```

$\overset{´}{\text{X}}$

while you would like to have Ẋ — the space is too big because the character `\char"13` in *Computer Modern* typeface is shifted in advance to fit the letter 'x' without further adjustments. To correct this artificial shift typical for TEX accents special macro `\aboxshift` should be used:

```
\upaccent{\aboxshift{\char"13}}{X}=
```

Ẋ.

This macro subtracts the height `1ex` in vertical direction and corresponding slant correction in horizontal direction.

Similar effect is when you try to put *comma* as the accent without preliminary adjustment:

```
\def\tcomma#1{\upaccent{,}{#1}}
\tcomma{X}=
```

$\overset{,}{\text{X}}$

Since `\upaccent` alignes the boxes with respect to the baseline and comma has non-zero depth, its descender overlaps with the character to be accented. To correct it we should use this macro:

```
\def\tcomma#1{\upaccent{\aboxsplit{\aboxbase{,}}}{#1}}
\tcomma{X}=X̦
```

Macro `\aboxbase` raises the character by its depth so that the baseline of the result corresponds to its geometrical bottom, and macro `\aboxsplit` adds some extra space above and below it.

Finally, let us show how the interletter accents shown in section 1 can be created:

```
\def\burg{b\upaccent{\aboxshift{\char"15}}{\markchar}g}
St.~Peters\burg=St. Petersb̆g
```

Here macro `\markchar` creates invisible rule with zero width and the height corresponding to that of letter 'x'. Since `\upaccent` adjusts the height and the depth of the accented character but not its width, the result is the object with zero width (from TEX' point of view) placed between characters *b* and *g*.

If we would like to make our macro case-sensitive, it can be defined as

```
\def\burg{b\upaccent{\Aboxshift{\char"15}}{\markchar}g}
\def\BURG{B\upaccent{\Aboxshift{\char"15}}{\MARKCHAR}G}
```

This results to:

```
St.~Peters\burg=St. Petersb̆g
ST.~PETERS\BURG=ST. PETERSB̆G
\MakeUppercase{st.~peters\burg}=ST. PETERSB̆G
\MakeLowercase{ST.~PETERS\BURG}=st. petersb̆g
```

(Here macro `\MARKCHAR` creates the invisible rule with the height of capital letter 'X', macro `\markchar` and `\MARKCHAR` operate in such a way that they are transformed into each other after uppercasing and lowercasing, macro `\Aboxshift` is the robust version of `\aboxshift` which is not destroyed by the LATEX 2_ε macro `\MakeUppercase` and `\MakeLowercase`.)

6. Commands for accents

Important note. In most cases the symbol used as the accent or modifier should be adjusted in advance — see examples in the previous section and section 7 for more details.

The style file ACCENTBX contains the following commands used to make accents:

`\upaccent{`*accent*`}{`*main-symbol*`}` — put *accent* above *main-symbol*

`\dnaccent{`*accent*`}{`*main-symbol*`}` — put *accent* below *main-symbol*

`\baseaccent{`*accent*`}{`*main-symbol*`}` — put *accent* at the baseline below symbol *main-symbol*

`\nullaccent{`*accent*`}{`*main-symbol*`}` — the *accent* (i.e., some character modifier like stroke, etc., attached to the main character) is centered with respect to the height of the *main-symbol*

`\upaccentC{`*accent*`}{`*main-symbol*`}` — synonim for `\upaccent`.

`\upaccentL{`*accent*`}{`*main-symbol*`}` — put *accent* above *main-symbol* like the command `\upaccent` does, but aligns it over the left corner of the character.

`\upaccentR{`*accent*`}{`*main-symbol*`}` — put *accent* above *main-symbol* like the command `\upaccent` does, but aligns it over the right corner of the character.

`\dnaccentC{`*accent*`}{`*main-symbol*`}`, `\dnaccentL{`*accent*`}{`*main-symbol*`}`, `\dnaccentR{`*accent*`}{`*main-symbol*`}` — corresponding modifications of the command `\dnaccent`.

`\baseaccentC{`*accent*`}{`*main-symbol*`}`, `\baseaccentL{`*accent*`}{`*main-symbol*`}`, `\baseaccentR{`*accent*`}{`*main-symbol*`}` — corresponding modifications of the command `\baseaccent`.

`\nullaccentC{`*accent*`}{`*main-symbol*`}`, `\nullaccentL{`*accent*`}{`*main-symbol*`}`, `\nullaccentR{`*accent*`}{`*main-symbol*`}` — corresponding modifications of the command `\nullaccent`.

`\upaccentbar[`*height*`]{`*main-symbol*`}` — place the horizontal bar above the *character* taking into account its width (i.e., it can place the bar over more than one character). Optional parameter *height* specifies additional space which is placed around the bar. Default value of additional space is 0.2ex.

`\dnaccentbar[`*height*`]{`*main-symbol*`}` — put the horizontal bar below the *character* taking into account its width (i.e., it can place the bar under more than one character). Optional parameter *height* specifies additional space which is placed around the bar. Default value of additional space is 0.2ex.

You can use a variety of TEX constructions as the *accent* and *main-symbol* — mainly those that can be used inside `\hbox`. In particular, the accented letter can be used as the main symbol (i.e., the multiple commands `\upaccent`, `\dnaccent`, `\baseaccent`, etc., could be used). It is essential to note that if the final symbol in a chain of accenting commands is the symbol with `\catcode` 11 or 12, the italic correction command `\/` will work correctly if it follows the whole construction.

7. Commands which create and correct accent characters

The special objects useful if they are used as the *accents* can be created using the following commands:

`\aboxshift[`*height*`]{`*character*`}` — lower by *height* the character (which is usually the accent from Computer Modern font family). Default value for *height* is 1ex. Except the height the additional horizontal shift typical to slanted fonts is subtracted as well.

`\aboxbase[`*height*`]{`*character*`}` — raise the character by the specified *height* and, if there is still some depth of the result, by the residue depth. Default value for *height* is 0pt which means that the character is just raised if some its elements are below the baseline.

`\aboxbaseline{`*character*`}` — raise or lower the character so that its bottom becomes the baseline (i.e., if the depth of the character is positive, the character is raised, if the depth is negative — the character is lowered).

`\aboxsplit[`*height*`]{`*character*`}` — insert above and below the *character* the white space with the specified *height*. Default value for the *height* is 0.2ex.

`\aboxjoin[`*height*`]{`*character*`}` — subtract above the *character* and below it the specified *height*. Default value for the *height* is 0.1ex.

`\aboxsplitup[`*height*`]{`*character*`}` — insert below the *character* the white space with the specified *height* (it adds white space *below* the accent

because the accent created by `\upaccent` is supposed to be *above* the character). Default value for the *height* is 0.2ex.

`\aboxsplitdn[`*height*`]{`*character*`}` — insert above the *character* the white space with the specified *height* (it adds white space *above* the accent because the accent created by `\dnaccent` is supposed to be *below* the character). Default value for the *height* is 0.2ex.

`\aboxsplitup[`*height*`]{`*character*`}` — subtract below the *character* the white space with the specified *height* (it subtracts white space *below* the accent because the accent created by `\upaccent` is supposed to be *above* the character). Default value for the *height* is 0.1ex.

`\aboxsplitdn[`*height*`]{`*character*`}` — subtract above the *character* the white space with the specified *height* (it subtracts white space *above* the accent because the accent created by `\dnaccent` is supposed to be *below* the character). Default value for the *height* is 0.1ex.

`\markchar` — make the invisible rule with the height equal to `1ex`. After `\uppercase` and `\MakeUppercase` it is converted into `\MARKCHAR`.

`\MARKCHAR` — make the invisible rule with the height equal to the height of the capital letter *X*. After `\lowercase` and `\MakeLowercase` it is converted into `\markchar`.

`\marktwochar{`$char_1$`}{`$char_2$`}` — make the invisible rule with zero width, with height equal to the maximal height of two characters and with corresponding depth. Can be substituted by `\aboxmarker` but is conserved for compatibility reasons.

`\aboxmarker{`*character*`}` — make the invisible rule with zero width, with height equal to the height of the box and with corresponding depth.

`\aboxnull{`*character*`}` — makes an artificial box with zero height and zero depth but with the same the contents as the source. The width and the baseline are conserved as well.

`\akern{`$char_1$`}{`$char_2$`}` — make the kern corresponding to the pair of characters. It is useful if you would like to reconstruct kerning destroyed by accents. Example:

```
\upaccent{.}{A}\upaccent{.}{W}          =  ȦẆ
\upaccent{.}{A}\akern AW\upaccent{.}{W} =  ȦẆ
```

`\aboxbar{`*character*`}` — create the horizontal bar which is as wide as necessary to overline or to underline the *character*. The commands `\upaccentbar` and `\dnaccentbar` described in the previous section are defined with its help as:

```
\def\upaccentbar#1{\upaccent{\aboxsplit{\aboxbar{#1}}}{#1}}
\def\dnaccentbar#1{\dnaccent{\aboxsplit{\aboxbar{#1}}}{#1}}
```

The width of the bar is scaled with respect to the nominal width of the character by factor `\accentwidthfactor` (default value 0.9). The thickness of the bar is the height of the dot selected from currect font and scaled twice by factor `\accentthickness` (default value is 0.1) — so by default the bar thickness is in 5 times less than the thickness of the dot (i.e., the character '.' of the current font).

`\aboxrule{`*width*`}` — just the same as `\aboxbar` but the width of the object to be covered by the bar is specified explicitly and the width is not scaled by `\accentwidthfactor`.

8. General accent `\makeaccent` and general frame `\aboxframed`,`\aboxtuning`

Macro listed above are enough for 99% of applications. But for extremely exotic accent constructions and for cases which require fine tuning of the accent and the main character ACCENTBX contains three additional commands:

— `\makeaccent` — for fine tuning of the accent and the main character (both could be some `\hbox` constructions!) with respect to each other,

— `\aboxframed` — for fine tuning of the frame and the baseline of the character used as the accent,

— `aboxtuning` — for explicit specification of frame and baseline of the box.

8.1. Macro `\aboxframed` and `\aboxtuning`

Macro `\aboxframed` enables to control white spaces around the character used as the accent and the position of its baseline. It has the syntaxis:

`\aboxframed[`*options*`]{`*object*`}`

where *object* is the `\hbox`-object to be adjusted, and *options* is the list of letters describing the operations:

`r` — increase right boundary by one step,

`u` — increase upper boundary by one step,

`l` — increase left boundary by one step,

`d` — increase lower boundary by one step,

`b` — increase baseline position by one step,

`R` — decrease right boundary by one step,

`U` — decrease upper boundary by one step,

`L` — decrease left boundary by one step,

`D` — decrease lower boundary by one step,

`B` — decrease baseline position by one step.

You can use as much lettwers as you wish, and each letter performs an independent increment or decrememt of corresponding value. The step values are defined by macro:

`\aboxframestepV` — step in vertical direction (i.e., for upper and lowerr boundaries and for baseline position), default value is `0.2ex`,

`\aboxframestepH` — step in horizontal direction (i.e., for right and left boundaries), default value is `0.08em`.

The algorithm behaves like following:

1. All five increments (right boundary, upper boundary, left boundary, lower boundary and baseline position) are collected and stored in corresponding variables.
2. The height and the depth of the object are corrected by corresponding values without any changes inside the object.
3. Left and right spaces (may be, negative) are added to the object.
4. The object is shifted (raised) by the value corresponding to the shift of the specified shift of the baseline.

To make life easier, there is the macro `\aboxtuning` which performs just the same operation in one step. Its syntaxis is:

`\aboxtuning{`Δr`,`Δh`,`Δl`,`Δd`,`Δb`}{`*object*`}`

where Δr shows the shift of the right boundary, Δh — the shift of the upper boundary, Δl — the shift of the left boundary, Δd — the shift of the lower boundary, Δb — the shift of the baseline. Some values can be omitted (in this case they are assumed to be zero), trailing commas can be omitted also if there are no more values in a list.

8.2. Macro `\makeaccent`

Macro `\makeaccent` enables to control mutial alignment of the accent and the main character in more details. It has the syntaxis:

`\makeaccent[`*options*`]<`*shifts*`>{`*accent-character*`}{`*main-character*`}`

where

— *accent-character* is the hbox-object used as the accent,

— *main-character* is the hbox-object to be accented,

— *options* is the list of letters describing roughly what base points of the accent and the main character are joined when these two object are overlapped,

— *shifts* is the list of four numeric values which control fine tuning of the base points of the accent and the main character.

Macro `\makeaccent` overlaps the *main character* and the *accent character* so that their base points coinside with each other. When the accent character is lowered or raised, the corresponding slant correction is added to is horizontal shift. By default the accent and the main character are aligned so that the accent is centered above the main character and its baseline is placed on top of the main character (see below the defasult value for the parameter *options*). Parameters *options* and *shifts* are optional — they may be omitted together with surrounding angular and square braces.

Following letters can be used in *options* to specify the base points of the *accent-character* and the *main-character* which should be overlapped by TEX when composing the accented character:

`r` — right side of the main character,

`l` — left side of the main character,

`o` — center between left and right boundaries of the main character,

`t` — top of the main character,

`d` — bottom of the main character (including its depth),

`b` — baseline of the main character,

`c` — vertical center of the main character (the middle between its top and bottom boundaties including its depth),

`s` — 1/2 of the height of the main character,

`z` — 1/2 of the depth of the main character.

`R` — right side of the accent,

`L` — left side of the accent,

`O` — center between left and right boundaries of the accent,

`T` — top of the accent,

`D` — bottom of the accent (including its depth),

`B` — baseline of the accent,

`C` — vertical center of the accent,

`S` — 1/2 of the height of the accent,

`Z` — 1/2 of the depth of the accent.

For example, the combination `rt` means that the right top point of the main character is selected as the base point, and the combination `OD` means that the center of the lower boundary of the accent is selected as its base point. By default *options*=`[tBoO]` which means that the accent and the main character are aligned horizontally so that their vertical middle lines coinside, and that the top of the main character and the baseline of the accent counside as well.

In addition to rough specification of the base points of the main character and the accent character by *options*, the User may perform fine tuning of their positions using *shifts*. Namely, this parameter is the list of four values (lengths) separated by commas which are added to the coordinates of the base points specified by *options*. First value corresponds to x-shift of the base point of

the main character, second value — to y-shift of the base point of the main character, third value — to x-shift of the base point of the accent character, forth value — to y-shift of the base point of the accent character. If some values are equal to zero, they can be omitted. If there are no more non-zero values in a list, trailing commas can be omitted as well. By default parameter *shift* consistes of four zero values.

To illustrate the power of macro `\makeaccent` let us see how the accent macro described in section 6 can be specified:

```
\def\upaccentC#1#2{\makeaccent[BtOo]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\upaccentR#1#2{\makeaccent[BtRr]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\upaccentL#1#2{\makeaccent[BtLl]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\dnaccentC#1#2{\makeaccent[TdOo]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\dnaccentR#1#2{\makeaccent[TdRr]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\dnaccentL#1#2{\makeaccent[TdLl]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\baseaccentC#1#2{\makeaccent[TbOo]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\baseaccentR#1#2{\makeaccent[TbRr]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\baseaccentL#1#2{\makeaccent[TbLl]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\nullaccentC#1#2{\makeaccent[SsOo]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\nullaccentR#1#2{\makeaccent[SsRr]<0pt,0pt,0pt,0pt>{#1}{#2}}
\def\nullaccentL#1#2{\makeaccent[SsLl]<0pt,0pt,0pt,0pt>{#1}{#2}}
```

9. Robust versions of some commands

Since LaTeX 2$_\varepsilon$ macro `\@ifnextchar` is fragile and is destroyed by some LaTeX 2$_\varepsilon$ commands (for example, by `\MakeUppercase` and `\MakeLowercase`), commands with optional arguments have similar robust versions which do not have options but which are not destroyed inside LaTeX 2$_\varepsilon$ stomach. The list of fragile commands is the following:

— `\aboxshift[`*height*`]{`*character*`}`:

  – `\Aboxshift{`*character*`}` assumes that *height*=1ex,

  – `\AboxShift{`*height*`}{`*character*`}` is robust.

— `\aboxbase[`*height*`]{`*character*`}`:

  – `\Aboxbase{`*character*`}` assumes that *height*=0pt,

  – `\AboxBase{`*height*`}{`*character*`}` is robust.

— `\aboxsplit[`*height*`]{`*character*`}`:

  – `\Aboxsplit{`*character*`}` assumes that *height*=0.2ex,

  - `\AboxSplit{`*height*`}{`*character*`}` is robust.

— `\aboxsplitup[`*height*`]{`*character*`}`:

  - `\Aboxsplitup{`*character*`}` assumes that *height*=0.2ex,
  - `\AboxSplitup{`*height*`}{`*character*`}` is robust.

— `\aboxsplitdn[`*height*`]{`*character*`}`:

  - `\Aboxsplitdn{`*character*`}` assumes that *height*=0.2ex,
  - `\AboxSplitdn{`*height*`}{`*character*`}` is robust.

— `\aboxjoin[`*height*`]{`*character*`}`:

  - `\Aboxjoin{`*character*`}` assumes that *height*=0.1ex,
  - `\AboxJoin{`*height*`}{`*character*`}` is robust.

— `\aboxjoinup[`*height*`]{`*character*`}`:

  - `\Aboxjoinup{`*character*`}` assumes that *height*=0.1ex,
  - `\AboxJoinup{`*height*`}{`*character*`}` is robust.

— `\aboxjoindn[`*height*`]{`*character*`}`:

  - `\Aboxjoindn{`*character*`}` assumes that *height*=0.1ex,
  - `\AboxJoindn{`*height*`}{`*character*`}` is robust.

— `\upaccentbar[`*height*`]{`*character*`}`:

  - `\Upaccentbar{`*character*`}` assumes that *height*=0.2ex,
  - `\UpaccentBar{`*height*`}{`*character*`}` is robust.

— `\dnaccentbar[`*height*`]{`*character*`}`:

  - `\Dnaccentbar{`*character*`}` assumes that *height*=0.2ex,
  - `\DnaccentBar{`*height*`}{`*character*`}` is robust.

— `\aboxframed[`*options*`]{`*character*`}`:

  - `\Aboxframed{`*character*`}` assumes that *options*=ud,
  - `\AboxFramed{`*options*`}{`*character*`}` is robust.

— `\makeaccent[`*options*`]<`*shifts*`>{`*accent*`}{`*character*`}` :

  - `\Makeaccent{`*accent*`}{`*character*`}` assumes that the parameters *options* and *shifts* are empty,
  - `\MakeAccent{`*options*`}{`*shifts*`}{`*accent*`}{`*character*`}` is robust.

10. Case-sensitive accents for *I* and *J*

The letters *I* and *J* should be separated in a specific class among all the letters used with accents. The essential difference is that the lowercase form contains the *dot* accent while the uppercase form does not have such modifiers. To make correct accented versions of lowercase *i* and *j* TEX contains special characters *dotless-i* and *dottless-j*, usually defined by the commands `\i` and `\j`. As a result the correctly working command `\accent` should analyze the uppercase and lowercase forms of these letters and select proper glyph to produce correct forms of letters even after `\uppercase` and `\lowercase`.

As soon as we consider the cyrillic writing systems, the capital *Ukranian I* (Ї) should be produced by `\"I`, while the lowercase *ukranian i* (ї) uses the command `\"\i`. Contrary to that example, the Khakassian letter *dotted-I*, which has a dot in uppercase form as well as in lowercase one, requires `\.I` to produce the uppercase variant İ, and an ordinary `i` to produce the lowercase variant *i*. Since it is assumed that *Mr. User* or *Mrs. User* can distinguish what letter does he/she want and to separate uppercase and lowercaee variants explicitly, it seems that the existance of two various types for specification of lowercase and uppercase forms does not produce any troubles. Unfortunately it is so before the commands `\uppercase` and `\lowercase` starts their work:

```
İ = \.I   \uppercase{\.I} = İ   \lowercase{\.I} = i
i =  i     \uppercase{i}  = I    \lowercase{i}  = i
Ї = \"I   \uppercase{\"I} = Ї   \lowercase{\"I} = ї
ї = \"\i  \uppercase{\"\i} = ї  \lowercase{\"\i} = ї
```

Although in principle `\uppercase` and `\lowercase` can be encountered at any place, in practice only the `\uppercase` and only in LATEX headers is used outside explicit User's control (although, for example, to create the headers using SMALL CAPITALS font one could insert in some style the macro containing the construction like `\lowercase{\sc ....}`). Roughly the LATEX mechanism for automatically created headers works as follows:

1. The header text *some-header-text* containing the text and TEX commands is created elsewhere.

2. The object *some-header-text* is expanded into non-expandable tokens.

3. The uppercase transformation over the list of non-expandable tokens is performed. The latter means that the characters are substituted by their `\uccode` values while the TEX primitive commands remain unchangable.

4. The result is substituted into the header of the page and the list of tokens is processed by TEX as usually.

(The detailed investigation of LATEX source code shows that it is not actually true, but the result of 'true' macro is close to this description.)

The behaviour of `\uppercase` and `\lowerrcase` is different in such formats as *Plain*, $\mathcal{AMS}$TEX and LATEX 2.09, and LATEX 2_ε. The transformation of the characters in old formats uses only the values `\uccode` and `\lccode`, while LATEX 2_ε uses additional list `\@uclclist` of characters which use their own rule different from `\lccode`–`\uccode`. In particular, for old formats and LATEX2_ε in OT1 with encoding the symbols `\i` and `\j` are not affected by the commands `\uppercase` and `\lowercase`, which means that the page header ctreated automatically by `\uppercase` will be wrong.

In this respect the behaviour of LATEX 2_ε in T1 encoding is quite different. The letter *dotless-i* remains constant after `\lowercase`, and is transformed into capital *I* after `\uppercase`. The letter *I* after `\uppercase` remais the same, but `\lowercase` transforms it into ordinary *i*. Finally, the letter *i* is transformed into *I* after `\uppercase`, and conserves its form after `\lowercase`. Similar transformations happen with $\jmath \longleftrightarrow J \longleftrightarrow j$. This property enables to create headers and footers properly as soon as only one `\uppercase` is used for it, but creates funny errors when the User tries to apply `\uppercase`–`\lowercase` by himself[6]. The optional variant where the stress is put above some of these letters makes the life even more unhappy.

The following macro enables to analyze if the text is transformed by some `\uppercase` or `\lowercase` commands and put correct form of the letters. The commands `\i` and `\I` create the letters $\imath/I$, which are correctly processed by `\uppercase`–`\lowercase`, commands `\ii` and `\II` create the letters $i/\dot{I}$ with corresponding properties:

```
\edef\temp{\the\i} \chardef\idotless=\temp\relax
\def\i{\ifnum‘i=‘\i \idotless \else I\fi}
\def\I{\ifnum‘I=‘\I I\else \idotless \fi}
\def\ii{\ifnum‘i=‘\i i\else \.I\fi}
\def\II{\ifnum‘I=‘\I \.I\else i\fi}
```

Finally, the ordinary letters `i` and `I` correspond to the case where the lowercase form has a dot while the uppercase form does not. These macro work due to

6 The problem becomes even more complex taking into account that there is also the letter $\dot{I}$ in encoding T1. It is transferred to ordinary *i* by `\lowercase`, and into *I* after subsequent `\uppercase`.

the fact that the name of the macro `\I` and `\i` is conserved after `\uppercase`–`\lowercase` transformation, while in expressions `‘i` and `‘I` it is coverted appropriately. It is worth to note that although *then* and *else* parts of the macro are also affected by `\uppercase` and `\lowercase`, the results are correct:

```
\uppercase\expandafter{\i}=I   \uppercase\expandafter{\ii}=İ
\lowercase\expandafter{\i}=ı   \lowercase\expandafter{\ii}=i
\uppercase\expandafter{\I}=I   \uppercase\expandafter{\II}=İ
\lowercase\expandafter{\I}=ı   \lowercase\expandafter{\II}=i
```

It is more or less evident that the combination “lowercase-i-without-dot” + “uppercase-I-with-dot” is of no interest, although corresponding macro could be created as well.

Similar commands are intoduced for *ȷ/j* and *J/J̇*.

```
\edef\temp{\the\j} \chardef\jdotless=\temp\relax
\def\j{\ifnum‘j=‘\j \jdotless \else J\fi}
\def\J{\ifnum‘J=‘\J J\else \jdotless \fi}
\def\jj{\ifnum‘j=‘\j j\else \.J\fi}
\def\JJ{\ifnum‘J=‘\J \.J\else j\fi}
```

It is essential that the commands `\i` and `\I`, `\ii` and `\II`, `\j` and `\J`, `\jj` and `\JJ` creates the correct forms when the commands `\uppercase` and `\lowercase` or `\MakeUppercase` and `\MakeLowercase` affect them. They also work correctly when there are additional upper and lower accents created by the commands from the section 6. It is worth to note that the invisible characters created by the commands `\markchar` and `\MARKCHAR` described in section 7 are also transformed into each other after transformation by `\uppercase` and `\lowercase`.

10.1. Title letters ‘L+soft-sign’ and ‘N+soft-sign’

Similar problem of case-sensitivity exists in the writing systems for Serbian, Makedonian, Itelmen and Nganasan languages which use the ligatures ‘*L+soft-sign*’ and ‘*N+soft-sign*’: Лъ, лъ, Нь, нь. Except the uppercase and the lowercase variants shown above there is also the ‘title’ variant[7] which is composed from the uppercase Л and Н and the bowl from the *lowercase* soft-sign ь. Such ligature is useful for titles where the first letter is the uppercase variant while the rest of the test is composed from the lowercase letters.

7 The Dutch ligature *IJ* also have the title form.

The command `\uppercase` applied to the title letter should transfer it into the normal uppercase form, and `\lowercase` — into normal lowercase form. Changing `\lccode`–`\uccode` values for the title letters (which could help in this case) is not allowed in LaTeX 2$_\varepsilon$, and the list `\@uclclist` used in LaTeX 2$_\varepsilon$ to make lower/uppercase conversion is of no usage since different assignments are necessary for `\uppercase` and `\lowercase`.

The following macro uses the same idea to distinguish `\uppercase` and `\lowercase` modes and to produce correct letter as the output:

```
\def\makeCYRlje{%
   \ifnum‘x=‘\X        \CYRLJE  % there was \uppercase
   \else \ifnum ‘X=‘\x \cyrlje  % there was \lowercase
   \else               \CYRlje  % natural title form
   \fi\fi
}
```

Here `\CYRLJE` corresponds to the uppercase form ЉЬ, `\cyrlje` — to the lowercase form љь, and `\CYRlje` — to the title form. Similar commands can be defined for the letter ‘N+soft-sign’.

11. Conditional operators if-then-else

Fine tuning of main characters and accents (modifiers) may require the analysis of the font style used for the composite construction. To make it easier, ACCENTBX contains special if-then-else macro:

`\ifupper{`*character*`}{`*if-upper*`}{`*if-lower*`}` — checks that you deal with the uppercase character: first parameter is the character to be checked, second — commands which are executed for *true*, third — commands to be executed for *false*.

`\iflower{`*character*`}{`*if-upper*`}{`*if-lower*`}` — checks that you deal with the lowercase character: first parameter is the character to be checked, second — commands which are executed for *true*, third — commands to be executed for *false*.

`\ifcaseupper{`*uppercase-or-none*`}{`*lowercase*`}` — checks that your construction was influenced by `\lowercase`: first group of commands is executed if there was `\uppercase` conversion or no conversion, and second group of commands is executed if there was `\lowercase` conversion.

`\ifcaselower{`*lowercase-or-none*`}{`*uppercase*`}` — checks that your construction was influenced by `\uppercase`: first group of commands is executed if there was `\lowercase` conversion or no conversion, and second group of commands is executed if there was `\uppercase` conversion.

`\ifcasetitle{`*no-case*`}{`*uppercase*`}{`*lowercase*`}` — checks allthree variants: first group of commands is executed if there is no conversion, second — for `\uppercase` conversion, third — for `\lowercase` conversion.

`\ifwide{`*character*`}{`*if-wide*`}{`*if-narrow*`}` — checks that you deal with wide character (character is considetred as wide if its width is more than the value `\accentwidefactor` (default value for it is 0.65em)).

`\ifwidth{`*character*`}{`*width*`}{`*if-wide*`}{`*if-narrow*`}` — just the same as `\ifwide`, but the width for comparison is specified explicitly as the second parameter.

if-then-else operators for font families: command `\ifrm{#1}{#2}` checks that you are in normal (upright roman) mode, command `\ifit{#1}{#2}` — that you are in *italic* mode, command `\ifsl{#1}{#2}` — that you are in *slanted* mode, command `\ifbf{#1}{#2}` — that you are in **boldface** mode, command `\ifsf{#1}{#2}` — that you are in sans serif mode, command `\ifsc{#1}{#2}` — that you are in SMALL CAPS mode, command `\iftt{#1}{#2}` — that you are in `typewriter` mode. There are also the commands `\ifmit{#1}{#2}` and `\ifsym{#1}{#2}` which checks that by some mistake you are in mathematical mode and are working with mathematical letters or mathematical symbols — but these should never used in practice 😀.

12. Changes in version 1.4

Version 1.4 is actually the first version which is uploaded to CTAN for wide distribution among TEX-users. Previous version 1.3 was distributed only between restricted set of people as a result of private communications. Nevertheless, the changes are the following:

— `\aboxadjust` is renamed into `\aboxshift`,

— `\makeaccent`, `\aboxframed` and `\aboxtuning` are added,

— commands `\akern` and `aboxmarker` appear,

— thickness and width od `\aboxbar` is controlled now by the User; macro `\aboxrule` appeared,

— optional arguments are introduced for some commands (by default, i.e., without such arguments, these commands behave as it was in previous version),

— robust versions of the commands with optional arguments are included,

— family of `\nullaccent...` commands is reviewed and modified (it was safe because in the previous version these commands were not documented and hence were not available for the User),

— behaviour of `\aboxbase` is changed — now it shifts the character only if it has positive depthy value; for compatibility with older application macro `\aboxbaseline` which shifts the character to its baseline always, is added,

— compatibility with *Plain*, $\mathcal{AMS}$TEX, etc., is checked and some errors are corrected,

— internal macro `\slant@value` is renamed and macro `\strip@pt` is deleted (since LaTeX 2_ε also has such macro),

— if-then-else structures now have two or three parameters — groups of commands to be executed when corresponding condition is satisfied are specified explicitly,

— if-then-else structures for uppercase/lowercase conversion and for title letters are added,

— Now you have more or less detailed documentation 🙂.

Acknowledgements

I am grateful to Olga Lapko, Mikhail Kolodin and Andrew Janishevskii for numerous useful discussions which result to the appearance of the style file ACCENTBX. The examples of TEX programming demonstrated by Kees van der Laan helped to make the package ACCENTBX more effective and compact. The discussions with Jörg Knappen helped to understand better the base ideology of accents in T1 and EC/DC fonts. The extensive comments and recommendations of the first referee of this paper, Bernd Raichle, enabled to improve greatly the description of the `\accent` algorithm in section 2 and to correct numerous errors in suggested macro. Robin Fairbairns spent a lot of his time polishing the text of the first variant of this paper. I express my warmest thanks to all these TEXnicians and TEXperts.

And I would like to thank separately Aaldert Compagner from Delft University of Technology for his long-term friendship and patient attention as a teacher — not necessarily relating to some TEX or LATEX joint works.

Bibliography

[1] Jörg Knappen, documentation supplementary to European Computer Modern font family.

[2] Donald E. Knuth. The TEXbook. Addison-Wesley, 1984 (reprinted with corrections in 1989).

[3] Victor Eijkhout. TEX by Topic, a TEXnician's Reference. Addison-Wesley, 1991.

[4] D.Salomon. NTG's Advanced TEX course: Insights & Hindsights. MAPS special issue, 1994.

[5] Eitan M. Gurari. TEX & LATEX: drawing & literate programming. McGraw-Hill, Inc., 1994.

[6] Kresten Krab Thorup and Frank Jensen, The style file CALC.STY (infix notation arithmetic in LATEX, version 4.0c).

[7] Michael Mehlich, The package *FP.sty* (*Fixed Point Package*), version 0.8 (April 1995).

[8] Frank Buchholz, The style file REALCALC.STY, dated January 1993.