https://doi.org/10.20378/irb-49314

# Unicode 5.1, Old Church Slavonic, Remaining Problems – and Solutions, including OpenType Features

Sebastian Kempgen (University of Bamberg)

## 1. From Unicode 4.1 to Unicode 5.1

In two previous articles (Kempgen 2006a, 2006b)[1] we took a look at version 4.1 of the Unicode character encoding standard from the perspective of Slavic philology. These papers detailed important achievements (like the addition of Glagolica), noted some minor mistakes and reviewed the standard with respect to which characters would still have to be added to serve the needs of researchers in the area of Slavic philology, especially medievalists.[2] Because these articles also contained a brief introduction to Unicode, its aims and purpose, and the different versions of this world-wide encoding standard, we need not repeat this information here.[3]

Both papers originated from presentations given at two conferences, especially the *Azbuki.net* conference in Sofia in 2005. At this conference, members of the Commission for "Computer Processing of Old Slavic Manuscripts" of the International Congress of Slavists met to discuss various topics. One of the most important ones was the plan to further advance the support for Slavic philology in the Unicode standard by submitting a new proposal. Ralph Cleminson, who had already authored the proposal to add Glagolitic to Unicode, agreed to start work on a new proposal with all others contributing whatever material they had access to or could supply. Thus, during the course of following weeks and months, work on this project started and a preliminary proposal was circulated by Ralph Cleminson as the sole author (Cleminson 2006a).[4] This preliminary proposal was then submitted to Unicode (as mentioned in Kempgen 2006b, 244), but thereafter authorship of the proposal was expanded to include members of the "Script Encoding Initiative" (SEI) at the University of Berkeley. The final outcome of these activities was proposal N3194 (Everson et al. 2007).[5] The final proposal contained much more additions to Unicode than anticipated at first, and not only Cyrillic additions for Slavic languages, but for non-Slavic languages as well. The proposal was reviewed and accepted by Unicode, but could not be added to Unicode 5.0 in time. Instead, it was one of the additions that went into version 5.1 which became official on April 4, 2008 and now should serve as the reference for all future discussions among Slavists.

The present paper is based on the author's contribution for the workshop of the "Slovo" conference in Sofia in February of 2008.[6] The original presentation is available online[7].

---

[1] Besides their printed version, both articles are also available from the author's server at http://kodeks.uni-bamberg.de/Unicode/SlavicUnicode_Fonts.htm. Also on this page are links to the online versions of the original presentations.

[2] These papers thus serve to offer a post-Unicode 4.1 update and, at the same time, a broader look at a topic covered in earlier papers by David Birnbaum (1996; 2002).

[3] For more information, see also the website http://www.unicode.org.

[4] To this preliminary proposal, the author of the present paper has contributed figs. 1, 7, and 12 resp. figs. 15, 24, and 40 to the final proposal. At this point, it might be noted that the final proposal incorrectly attributes figs. 43 to 47 (on page 46) to Karskij. Actually, they are scans from Diels (1934).

[5] Many proposals are collected at http://www.evertype.com/formal.html. It may be worth noting that some of the characters proposed in the N3194 were already contained in a draft put together by D. Birnbaum, R. Cleminson and M. Everson in 2002. This draft, however, never became an official proposal and was withdrawn by the authors; it can still be found on the web, though.

[6] Slovo: Towards a Digital Library of Slavic Manuscripts, Sofia, Bulgaria, Feb. 21-26, 2008. For more info, see also: http://slovo-aso.cl.bas.bg/index.html.

## 2. Advances in Unicode 5.1

With Unicode version 5.1 now being the official standard, let's take a look back and review which additions, missing characters, problems etc. were discussed and noted in our first two papers, on UC 4.1 Here we will concentrate on Cyrillic letters and especially Old Church Slavonic writing.[8]

① Soft Consonants

In our paper, we mentioned the Cyrillic soft consonants as an area not yet covered by Unicode 4.1. Specifically, the soft consonants ⟨ Ҥ ҥ Ԁ ԁ Ԗ ԗ Ҿ ҿ ⟩ were shown using illustrations from palaeographic works (Kempgen 2006a, section 3.6, Fig. 14).[9] Three of these consonant pairs, ⟨ Ԁ ԁ Ԗ ԗ Ҿ ҿ ⟩, are exactly the soft consonants that have been included in the proposal and now are part of Unicode 5.1. The fourth pair ⟨ Ҥ ҥ ⟩ has been identified with a character pair which was already present in Unicode 4.1 for non-Slavic Cyrillic at [U+04A4, U+04A5]. The name of this character is "Cyrillic ligature en ghe", and this indicates a certain problem. All of the aforementioned historical Slavic characters are not ligatures whose second part is a ⟨ Г ⟩. Rather, they are ligatures of the base character with a palatalization hook ⟨ ◌̑ ⟩[10]. Typographically, a ligature of a base character with the Cyrillic character ghe ⟨ Г ⟩ has a much wider right part than a character with an integrated palatalization hook ⟨ Ԁ Ԗ Ҿ ⟩. The arm of these characters is more comparable to the shorter one in ⟨ Ъ Ь ⟩. However, we have to accept the decision which means that ⟨ Ҥ ҥ ⟩ functions both as a ligature ⟨ н-г ⟩ and as a palatalized consonant similar to ⟨ н̑ ⟩. We suggest then, that a note should be added to the Unicode docs pointing out the two-fold function of this character. The most important consequence for Slavists, however, is that effectively all four character are now available in Unicode.

② Cyrillic IÔ

In our paper, we also mentioned Russian Cyrillic ⟨ iô ⟩ which was used in the 18th century before Karamzin introduced ⟨ ë ⟩ instead (Kempgen 2006a, section 3.6). One of the possible shapes this digraph has is a circumflex sitting *between* the two characters. Neither this digraph nor the circumflex between letters are characters on their own in Unicode terms and thus were not included in the proposal. This digraph, however, be can realised – in all of its incarnations – using other techniques and technologies. Let us briefly indicate here how this can be achieved. In Unicode, we have a combining circumflex ⟨ ◌̂ ⟩ at [U+0302]. This circumflex can be

[7] http://kodeks.uni-bamberg.de/AKSL/Schrift/UnicodeOCSProblemsSolutions.htm.

[8] For an overview of additions in Unicode 5.1 in comparison to Unicode 5.0, see http://babelstone.blogspot.com/2007/06/whats-new-in-unicode-51.html

[9] It might be worth noting which Unicode symbols are used here to display these characters. The brackets ⟨ ⟩ are the so-called "left-pointing angle bracket" [U+2329] and the "right-pointing angle bracket" [U+232A]. These brackets should be used for letters and graphemes, not the "less than" and "greater than" symbols, i.e. not < >. They are, admittedly, easier to type on standard keyboards and every font has them, but they have a different function nevertheless – they aren't brackets! Spacing between letters inside these brackets is fine-tuned here using one of many (!) space characters available in Unicode. Here, we are using the so-called "thin space" [U+2009]. Another very handy space character is the "hair space" [U+200A] which is even narrower. All the symbols mentioned here are available in the author's "RomanCyrillic Std" font (v. 3 and later). Using these appropriate spaces is more correct in Unicode terms than reducing the point-size of the standard space (as typed using the space bar) which is still common practice to achieve a similar result.

[10] The character used here to display combining characters is the "dotted circle" [U+25CC]. This is the character that has been added to Unicode exactly for this purpose. Curiously, however, the authors of the Unicode docs use a custom 8-bit-font to display this character – not the correct Unicode character.

combined with the so-called "zero-width joiner" at [U+200D] [11]. This character will not break up a string of characters to the left and to the right (in contrast to the "zero-width space" or the "zero-width non-joiner" which also exist), and justification of text will not have any adverse effect on such a string of characters. Now if one combines the circumflex with the zero-width joiner and puts this combination between the two letters ⟨ i o ⟩, then you will get exactly the desired effect. The only problem on the 'presentation level' (as opposed to the normally invisible 'encoding level') is that the combining circumflex will probably sit too much to the left over a zero-width character. Here, some OpenType programming can help. Because OpenType features of a font will be treated in more detail below, we will only show one similar solution here: и͡о – it makes use of the "combining double macron" [U+035E].

③ Iotified Yat

Another character which was missing in Unicode 4.1. and noted as such in our paper (Kempgen 2006a, section 3.6) is the Old Russian iotified yat, ⟨ Ѣ ѣ ⟩. It was part of the proposal and has been accepted for inclusion in Unicode 5.1. Both forms, the more common lower-case form and uppercase form are patterned after the glyphs found in the *Izbornik Svjatoslava 1073g*. It should be noted, however, that not all glyphs which, at first glance, look like a iotified yat, are to be identified with this character – more often they are simply a variant of the non-iotified yat. The illustration below (from Čerepnin 1956, 37) shows samples:

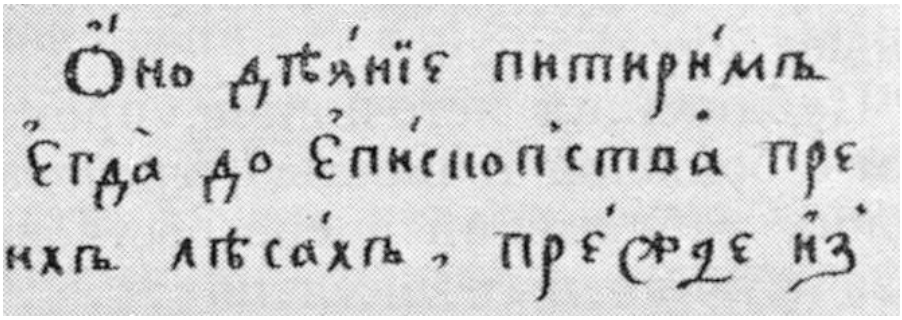

*Fig. 1: Variants of Yat – not a Iotified Yat*

This glyph by the way is missing – along with others – from the glyph repertoire discussed at the Belgrade conference in Oct. 2007.[12] This illustration shows many other interesting glyph variants: note the epsilon-like ⟨ е ⟩, the calligraphic version of the ⟨ ж ⟩ and the ⟨ д ⟩, the square ⟨ в ⟩, etc.

[11] There is also a "zero-width space" [U+24] and a "zero-width non-joiner" [U+] in direct vicinity of the zero-width joiner. The "appearance" of all three is identical, so one should be very careful in selecting the character which fulfills a given purpose best.

[12] For more information on the conference and on the papers resulting from the discussions there, see http://www.sanu.ac.yu/Cirilica/Cirilica.aspx. A paper by Birnbaum et al. (2008) contains a response to the Belgrade documents. – Also missing from the Belgrade inventory are glyphs which are attested in the Novgorod Birch Bark letters (see also below, section 4.2), although it is not clear at the moment which are functional variants and which are mistakes.


*Fig. 2: Shapes (Glyphs) of Ja*

④ Ja

This character, ⟨ Ꙗ ꙗ ⟩, was singled out in our previous paper (Kempgen 2006a, section 3.6) as probably being the most glaring deficiency in Unicode 4.1 of a character needed for Slavic philology. The Ja was included in the proposal and added to Unicode 5.1 so any interim solution which added this character to the private use area of Unicode is now obsolete. A question that is still open to discussion is what the neutral glyph for this character should look like, especially for the lower-case letter, see Fig. 2 (upper half).

The proposal uses variant (a), left, to display the character. (b), middle, clearly is a historical variant very useful for Slavic philology but not as the neutral form. In our view, variant (c), right, the most appropriate neutral form. The "open a" in variant (a) is not a glyph which connects very well with anything to the left. Variant (c) shows that this character in a design which is parallel to ⟨ Ю ⟩. Historically, it goes back to the beginning of the Civil Type (see Fig. 1, lower half), and uses the glyph that an italic ⟨ *а* ⟩ would have anyway. Interestingly, this illustration considers ⟨ ꙗ ⟩ to be exactly that – the italic version of ⟨ я ⟩! Our *RomanCyrillic Std* font offers all three variants (a – c), but variant (c) will be the default one.

⑤ Paerok

From the many letter-like symbols used in early Slavic writing, we mentioned the *paerok* (*payerok, erok, ertica*) in our paper (Kempgen 2006a, section 3.7). It was included in the proposal and added to Unicode, both as a combining and as a spacing character. Other diacritics not mentioned explicitly in our paper (like *vzmet*, and *kavyka*) have been added as well. Support for symbols like these is more complete in Unicode than it seems at first – we will return to this observation later.

⑥ Superscripts

Superscripted letters are also an area mentioned in our previous paper (Kempgen 2006b, section 3.8) as an area where the Latin alphabet had better support in Unicode (in contrast to Cyrillic which had none at the time). Superscripts have been included in the proposal and added to Unicode – but, at present, not yet the complete alphabet. The rest should be added in the future. It is interesting to see, that Unicode accepted exactly the view expressed in our paper: that the *paerok* and all superscript letters should be treated equally, which means either both should be added to Unicode or none. The same is true for our remark concerning the "vertical combining tilde" [U+033E] as being different from the *paerok* – they have indeed been treated separately in Unicode 5.1. For Slavists, the vertical tilde is now identified with the *yerik*, which is similar, but not identical to the *paerok*. And because there was no spacing equivalent to the non-spacing (i.e. combining) vertical tilde, it has now been added as well.

 Orig. publ. in: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

⑦ Numbers

We mentioned the missing remaining number signs used in Russia – see Fig. 17 and section 3.10 of the second paper (Kempgen 2006b). These number signs have all been included in the proposal and added to Unicode so support for all numbers symbols is now complete. Here are some samples: ◌ ◌ ◌ . These numbers are to be composed from their parts: the dotted circle indicates the number character that is to be used (а, в, в, г,…), and the symbols that form a ring or bracket around it are combining diacritics. Typographically, it is therefore clear, that only the lower, historical forms of certain consonants which do not exceed the x-height will fit into the ring – cf., for example, ⟨ в ⟩ vs. ⟨ б ⟩.

One thing which needs to be addressed for a standard practice of using Unicode characters are the numbers from '11' to '19'. These numbers, all consisting of *two* letters, have *one* common *titlo* above them:

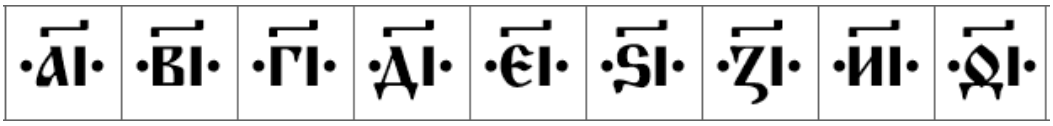

*Fig. 3: Two digit numbers 11–19*

Actually, there are several solutions available for such a case. The first solution is the "combining double tilde" [U+0360]. For the purposes of Slavic philology, think of the tilde as a titlo, so what we have here is a long titlo centered between two characters: ⟨◌͠◌⟩. Another solution are Unicode characters [U+FE22], "combining double tilde left half", and [U+FE23], "combining double tilde right half": ⟨◌ ◌⟩. Set next to each other (without a space in between), they produce the same result as the combining double tilde (as long as the justification of text puts additional white space only between words, not between letters!). For an even longer titlo, a 'middle part' is also available as [U+FE24]: ⟨◌⟩. Thus, a number like '12' could be written either as ⟨в + ◌͠◌ + і⟩ or as ⟨в + ◌ + і + ◌⟩ to produce ⟨в̄і⟩. The glyph ⟨і⟩ used here is an historical variant to standard Cyrillic ⟨i⟩ and is the default glyph in our *Kliment Std* font. Again, even better typographical solutions can be achieved using OpenType features (see below).

⑧ Cyrillic Jota

An area we drew the reader's attention to in our previous paper was the necessity to review the Cyrillic block again after the Glagolitic alphabet had been added to Unicode. As we demonstrated in Kempgen (2006a, section 3.7), a Cyrillic Jota was missing – a character not used as such in genuine Cyrillic texts, as it only occurs in Glagolitic texts transliterated into Cyrillic. This character, ⟨Ꙗ⟩, has also been added to the proposal and accepted by Unicode. The only thing that can be called unfortunate is the selection of the reference glyph in the proposal and in the Unicode docs. The proposal erroneously uses a sans-serif design of the uppercase Cyrillic iota and a serifed design for the lower-case design. In our paper, we had clearly shown what a serifed reference design of this character should look like (see Fig. 20 in Kempgen 2006b). However, changing the appearance of reference glyphs is not a problem at all, as the definition of the character does not include a specific representation of the character on the *glyph* level.

⑨ Neutral Yer and Blended Yus

Both characters were mentioned in our second paper (Kempgen 2006b, section 3.3). They have been added to the proposal and accepted for inclusion to Unicode: ⟨Ъ ъ Ѫ ѫ⟩. Fig. 9 of our paper showed Old Church Slavonic black letter designs of the characters in question. For

a serifed modern typeface, we have developed the character shapes shown here. In contrast, the proposal uses glyphs whose design cannot be considered very fortunate from a typographical perspective. This is a question we won't tackle in more detail here but will do so on a separate occasion. The same figure also showed an iotified variant of the blended yus. This glyph was not part of the proposal. If this character is to be added to Unicode in the future, concluding evidence for its existence and use must be offered first.

⑩ Transliteration of Glagolitic Nasals

In our paper, we also checked whether the transliteration of all Glagolitic nasal vowels was complete in the Cyrillic section of Unicode (see Kempgen 2006b, section 3.4). We noted two vowels where this wasn't the case: ⟨ ⰡⰑ ⰔⰔ ⟩ are both present in the Glagolitic block of Unicode. The transliteration for the first of these, the so-called "closed little yus", i.e. ⟨ Ѧ ѧ ⟩, has now been included in the proposal and added to Unicode 5.1. Support for the transliteration of the second Glagolitic character pair is thus still missing. Once, however, it is a separate character in the Glagolitic block, it probably needs to have its one-to-one correspondence in Cyrillic, just like the first pair, because it is common practice to transliterate Glagolitic texts into Cyrillic for publishing. Probably, the best way to represent ⟨ ⰔⰔ ⟩ in Cyrillic would be to use Iotifiers, i.e. ⟨ Ꙗ ꙗ ⟩. Such iotifiers have been mentioned by others to be desirable additions to the Cyrillic part of Unicode.[13]

⑪ Cyrillic Yn

This Romanian addition to the Old Church Slavonic writing was also mentioned in our paper (Kempgen 2006b, section 3.11). It has been included in the proposal and accepted by Unicode for inclusion in the standard. Interestingly, it is the only additional character in the OCS script that evolved naturally, and the only one to stem from its singular non-Slavic use (i.e. for Romanian).

⑫ Yery with Back Yer

The basic variants of *yery,* i.e. ⟨ Ъі Ы ⟩, plus connected variants plus lowercase variants with overdot ⟨ i ⟩ or without, were also discussed by the author in his paper (Kempgen 2006b, section 4.2). The *yery* with the back vowel, i.e. ⟨ Ъіъ ы ⟩ has been included in the proposal as a separate character and accepted for inclusion in Unicode. The connected versions are to be treated as glyph variants of the two basic pairs that are now available.

Thus, if we compare the contents of the – successful and large – proposal N3194 with the 'missing' OCS characters mentioned in the author's two papers, we find that nearly everything is now included in Unicode, version 5.1. Even better: the proposal contained additional characters not mentioned explicitly in our papers. Thus, for slavists, the jump from Unicode 4.1 to version 5.1 is a great step forward and an important progress indeed. However, even after this success, there is still work to be done because of the ongoing debate about the state of certain additional glyphs that some think should be included in Unicode. More on this topic will be published in vol. 6 of the journal *Scripta & e-Scripta* which contains papers for the XIV International Congress of Slavists in Ohrid (September 2008) plus additional material.

[13] Unless, of course, those Iotifiers are identified with Cyrillic ⟨ J j ⟩ as historical variants.

## 3. Problems in using Unicode

With such a wealth of characters, symbols, diacritics Unicode now has to offer it can sometimes be difficult to make the right use of all these characters, to make the right choices, to properly distinguish between the presentation level (appearance) and the encoding level of a text and its entities (code points) etc.. Furthermore, not every font designer will be familiar with all fine details of Cyrillic letters, their distinctive features (in comparison to other Cyrillic letters), or with the expected appearance of diacritics etc. This may lead to confusion on the side of the end-user who is presented with incomplete or, worse, incorrect choices. Last but not least, to certain extent this problem also depends on the user-interface an operation system presents: users will only make educated choices if they have access to the various options. In other words, what we really need today would be an "Introduction to Unicode for Slavists". Because we do not have such an introduction, we will use the opportunity to present some points here.

### 3.1 The 'Unicode Font' Buzzword

Today, editors of scientific journals or volumes often require that "only Unicode fonts" be used by their prospective authors. The reason for this is obvious to anyone who has ever dealt with publishing: a submitted paper should in principle be re-usable without re-typing anything, and applying a different set of fonts should change only the appearance but should otherwise leave all characters intact. In practice, this need not be the case, however: When users speak of Unicode fonts they assume them to be fonts where each glyph has been put into the character slot it is supposed to go into. However, nothing prevents a font designer to put a 'wrong' character into some slot, either by mistake or deliberately. In this respect, Unicode fonts are not in principle different to old 8-bit-fonts – they maybe offer less reason but many more options to deviate from the standard. Mistakes in choosing the right slot for a character are not seldom to be seen, and deliberately using a 'wrong' slot is a common strategy for giving users a character they need which does not (yet) have its own proper slot assigned. It may be true that fonts that deviate somehow from assigning all characters to their proper slots should not be called "Unicode" fonts any longer – however the common practice is not that strict.

### 3.2. What You Know Is What You Use

Using Unicode fonts (even if they fully conform to Unicode standards in the strict sense) does not prevent a user from making wrong choices in selecting characters. One example is the transliteration of the soft and the hard sign in Cyrillic: both have their own counterparts in Unicode because just like the original Cyrillic character, the transliteration characters need to be exactly that: *characters*. Out of tradition (and because these new characters are not always present in standard fonts) or maybe because they simply do not know about the correct Unicode transliteration characters, most users will use the customary 'curly quotes' ⟨' & "⟩ instead which certainly "look better" but are not the correct choice from an encoding perspective. Let us give an example: The famous Rila Monastery (Bulgarian: *Рилски манастир*) is called *Рыльский монастырь* in Russian.[14] Transliterated, this usually becomes *Ryl'skij monastyr'*. That, however, is the presentation level. What about the encoding level? On this level, this string could be represented this way (CH = character; PUNCT = punctuation):

[14] An alternate spelling is *Рильский монастирь*.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Р | ы | л | ь | с | к | и | й | | м | - | р | ь |
| 2. | R | y | l | ’ | s | k | i | j | | m | - | r | ’ |
| 3. | CH | CH | CH | PUNCT | CH | CH | CH | CH | PUNCT | CH | CH | CH | PUNCT |
| 4. | R | y | l | ’ | s | k | i | j | | m | CH | r | ’ |
| 5. | CH | CH | CH | CH | CH | CH | CH | CH | PUNCT | CH | CH | CH | CH |

In the first line, the Cyrillic representation is given, and in the second line, the transliteration, using a curly quote for the soft signs. The third line represents the string of characters according to their types as used in line 2 – either as characters (letters) or as punctuation marks. As you can see here, although *Рыльский* certainly is one single word, its encoding actually consists of two strings of characters broken up by a punctuation symbol, if the curly quote is being used to transliterate the soft sign. This has negative consequences for the electronic processing of such a text: searching, sorting, counting etc. simply cannot deliver correct results in this case because there is no way to distinguish between a ‘true’ quote and a quote used for transliteration. In the fourth row, the correct Unicode character has been used, and the encoding-level representation in line 5 changes accordingly. Here, we have two words separated by a space which is a punctuation mark – just as in the original text in line 1. Text-processing tools will have no problem in parsing such a string correctly – and that includes, for example, getting correct matches when using the “search” box in a pdf file. As one can see from this simple example, it is important to choose the correct Unicode characters if one is concerned with the encoding level of a text and with reliable text processing. If one is concerned with the “nicest-looking” representation only, then the curly quotes are what looks good in print.[15]

### 3.3 Similar Characters and Confusing Choices

Another example we would like to mention here is the confusing similarity of the Cyrillic semisoft sign ⟨ Ҍ ҍ ⟩ and the Cyrillic yat ⟨ Ѣ ѣ ⟩. Both do not simply look similar, they are also very close to each other in the Cyrillic Unicode block: *yat* is among the last characters in the basic (Slavic) Cyrillic block, and the *semisoft sign* is the second character in the Cyrillic Supplement block (see fig. 3). Slavists will usually not even be aware of the existence of this semisoft sign which is not required for Slavic Cyrillic at all. This is also an example where the operating system user interface plays a certain role. Fig. 4 shows the so-called “Character Palette” from OS X in one of its several display settings. The Character Palette is a handy tool to enter seldom-used characters or characters not supported by any keyboard layouts present on the user’s machine – the user simply clicks on the desired character to input it into the text file.

[15] Another mistake that we have come across several times is made by Czech-speaking users: sometimes they are using the Czech letters to transliterate Russian, i.e. they use *d’* and *t’*, which, of course, are *single* letters in Czech but represent *two* letters in the transliteration of Russian. Besides being wrong on the encoding level, this is also not the best representation typographically: the apostrophe sits much to tight next to the base letters to look correct in reproducing Russian Cyrillic.

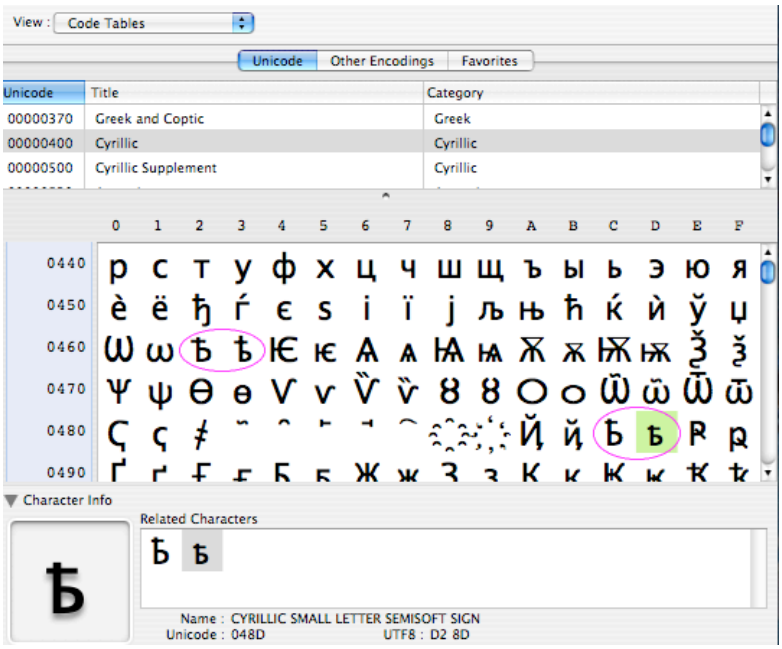


*Fig. 4: Yat and Semisoft Sign in OS X's 'Character Palette'*

However, choose another setting for the Character Palette, and it changes its view to display all characters in their alphabetical order – see Fig. 5. This order, however does not occur in any single language – it is script-centered, not language-centered. It displays the ordering of all Cyrillic characters as defined by Unicode, for whatever purposes that may be useful to an end-user. Notice that in this view, the semisoft sign comes right behind the soft sign, and the yat is two rows down. Here, it is especially easy for a Slavist to mistake the semisoft sign for the yat because the former is located approximately where one would rather expect to find the yat.

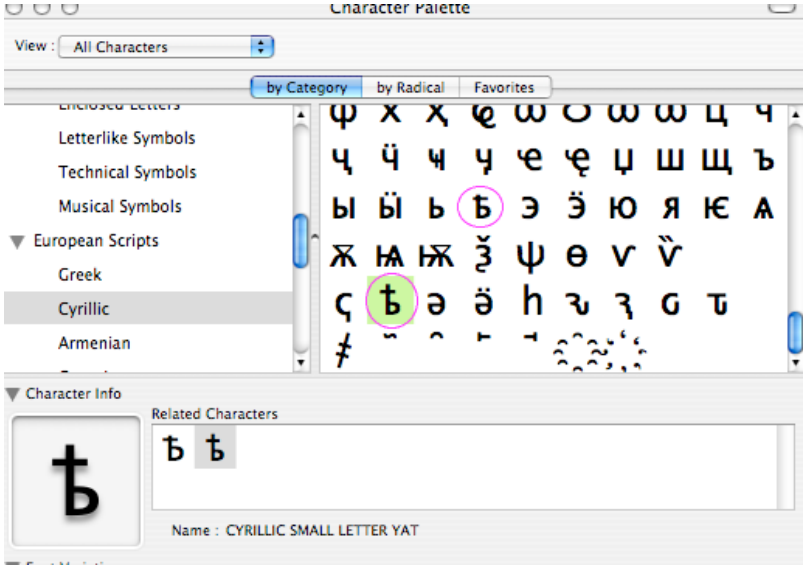


*Fig. 5: Yat and Semisoft Sign – another view*

### 3.4. Watch Your Language – Don't Mix Scripts!

The next example we will show here involves the reproduction of the orthography in Petăr Beron's "Fish Primer" (*Riben bukvar*). Interestingly, the same work has also been referred in another context in the proposal. The printing type used in this primer is the Old Church

 Orig. publ. in: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

Slavonic script, although the language, of course, is 19th century Bulgarian. Let us cite some sample sentence (p. 36): **Мóли́тва прéдъ да нача́ӗтъ да четӑтъ дѣца́-та.** There is one element in here that does not occur in Modern Bulgarian orthography: the writing of the schwa sound: **ӑ**. The word **четӑтъ** surely is Bulgarian, it's also without a doubt Cyrillic, but the ⟨ă⟩ is not available in the Slavic Cyrillic Unicode block. It is, however, a well-known Romanian character. So why not use the Romanian character, if it is not available in the Slavic Cyrillic block? The same glyph is, however, also available in the non-Slavic Cyrillic Extended Unicode block. Which one should be used – the appearance of both is, after all, the same? Why not simply use the one which is easier to type? The table below shows the answer.

| | | | | | | |
|---|---|---|---|---|---|---|
| 1. | ч | е | т | ă | т | ъ |
| 2. | CY | CY | CY | CY | CY | CY |
| 3. | ч | е | т | ă | т | ъ |
| 4. | CY | CY | CY | LT | CY | CY |
| 5. | ч | е | т | a˘ | т | ъ |
| 6. | CY | CY | CY | CY-D | CY | CY |

In the first row, the character from the non-Slavic Cyrillic block has been used. Characters are not defined for languages, but for scripts, so this is a Cyrillic character like all others. As can be seen, the string of characters consists of Cyrillic characters only, as indicated by the abbreviation CY in the second row. In the third row, we have used the Romanian character for the character in question. The effect is that we now have three distinctive sub-strings on the encoding level – see row four: CY-CY-CY, then LT, followed again by a string CY-CY. In other words: selecting the Latin character will encode this word so that all text processing, like searching and sorting, spell checking etc. will deliver wrong results. In the fifth row, we have used yet another method to produce the desired result: a Cyrillic ⟨ă⟩ can also be composed by using the normal Cyrillic vowel plus a combining breve above it – diacritics are not script-specific (although some may be introduced to Unicode along with a certain script – this is why you will find some diacritics needed by Slavists right in basic Slavic block, others in the Combining Diacritics block). So as to demonstrate the two separate steps in entering the characters, we have set the breve beneath the vowel, not above it. The only difference here is that on an encoding level, we don't have six, but seven characters.[16]

▶ Even when a Unicode font is being used, this does not automatically ensure that a user encodes a text correct in every respect, especially on the encoding level. What all these examples show us is that users need to be educated about correctly using Unicode characters and applying all of its possibilities.

## 4. Remaining Problems – and Some Solutions

In this section, we will review certain selected problems and several solutions to these problems. The discussion is not intended to be exhaustive either in covering all problems that still remain after UC 5.1 or in outlining all solutions, rather it will present some typical problems and solutions that are of practical relevance. All problems (and solutions outlined here) of course concentrate on missing characters and glyphs.

[16] On a side note, we should mention that the handling of strings such as the one in line 6 presents some kind of "lackmus" test for word processing tools. Which value would they return for the length of this string, i.e. how long is this word – six or seven characters long? Usually, the answer should be "6" on a linguistic level.

### 4.1. Missing Characters – Writing Proposals

Suppose you think that Unicode is missing a character, i.e. an entity which is not a stylistic or historical variant of some existing character, but something which you think is a character in its own right. Then the obvious and best solution would be to write a formal proposal and to submit it to Unicode Inc. However, for such a proposal to be successful, one should build a consensus in the scientific community what is a character and what isn't, and any proposal should be based upon such a consensus. Further, one should educate oneself in several different ways. First, on the Unicode web-site there is information about characters "in the pipeline", i.e. characters which have already been accepted or are in one of the various stages of acceptance but are not yet included in the current version of the Unicode standard. It is, of course, a waist of time to propose such characters a second time. Conversely, there is also information about characters which were proposed once but rejected on various grounds. Again, in most cases it will be a waist of time proposing such characters again except, of course, compelling evidence could be offered. Additionally, Unicode publishes "Errata" on its web-site. Due to the strict quality control, this list is small but sometimes errors or misunderstandings happen.

It is also important to get acquainted with current Unicode policies – which can change, and have done so in the past. To offer one example here, it would be possible to bring up the case of the "r grave accent" again (see Kempgen 2006a, section 3.1). Out of 24 Štokavian accents, this is the only one which is not available as a precomposed letter. When Unicode Inc. was approached by the author to find out whether this character had any chance of being added, the answer was a simple "no". Why? Because the current Unicode policy is that no more precomposed letters will be added if they can as well be composed from existing parts. This is the case here, and because the "r grave accent" is not part of an alphabet of any language, it must be composed. From a systematic linguistic approach to the Štokavian accent system such a decision must surely be regretted, but such is the Unicode policy.

So as not to provoke rejection by the standardization bodies, it is therefore important to be realistic about what can be added and what doesn't have a chance. Clearly, extremes must be avoided ("for true OCS support, thousands of combinations would have to be added"). Examples of how to write a successful proposal are available on the web – they can serve as a blueprint. Sometimes they propose only a single letter, sometimes a handful of them, sometimes a completely new script etc. The formal requirements of proposals have changed a bit over time so it is a good idea to look at recent samples.

▶ From a successful proposal, everybody in the user community benefits; it's an official solution, and it's the best solution for problems that fit this category. However, writing a proposal and getting it accepted by Unicode does not only require philological and technical work, but also some patience – one cannot expect an immediate acceptance (only a rejection could possibly be immediate). Usually, the formal process will take months at the very least, but can take one or two years as well. So this is not an immediate solution for urgent encoding needs one might have.

### 4.2. The "Private Use Area" (PUA) of Unicode

The Private Use Area of Unicode is large block of empty cells waiting to be filled with any characters which users need but which are not part of the Unicode standard. The reasons for this can be manifold. The convenience associated with the PUA certainly is that this is an immediate solution: you can add the required glyphs to a font and use it immediately. Also, the effect is that you can have these extra glyphs, maybe a corporate logo, or a non-standard

glyph, along with all basics characters in one font – you don't need custom corporate symbol fonts etc. But the use of this PUA has some disadvantages as well. The same slot that one font creator uses for some specific symbol may be used by another user for another symbol. The consequence, of course, is, that the compatibility between fonts is somewhat limited – one cannot simply use any other font for the final layout during prepress work when the original font and text makes use of the PUA. Furthermore, the use of the PUA is not even coordinated between all Western philologies. For those glyphs the PUA seems appropriate for, Slavic philology would need some sort of central "clearing house" and coordinated effort similar to the function the "Medieval Unicode Font Initiative" (MUFI: http://www.mufi.info/) fulfils for German philology, Nordic philology, English Studies etc. Also, specific uses of the PUA are an intermediate solution only. Suppose after the introduction of Unicode 4.1 someone put certain missing characters in the PUA and these characters have now been added officially to UC 5.1. Then it would no longer be justified to put them into the PUA, and to stay up-to-date users of the font in question would have to update their text files in accordance with the new standard and font version. The danger that this is a never-ending circle will, of course, becomes less and less as more and more characters are added to Unicode. For Slavic philology the step from Unicode 4.1 to 5.1 has indeed been an important change which obsoletes many intermediate solutions, and version 5.1 can now be considered to fulfil most 'normal' needs even for medievalists.

▶ The minimum requirement would seem to be that everybody who uses the PUA declares what he is doing by putting appropriate information online so others at least know about it.

Let us look at a (bad) example of using the PUA. For Slavists interested in medieval Novgorod and its birchbark letters, the website http://gramoty.ru is an exciting new resource a lot of funding has gone into. This site uses a special font to display the contents of the birchbark letters (except for the pictures, of course), and users of the site are required to download and install the font if they want to work with the actual texts. See the following screen-shot that demonstrates what the text of one such birchbark letter looks like when the required font is *not* installed: the user will see only empty boxes because there can be no 'fall-back' or 'default' font for displaying the PUA.

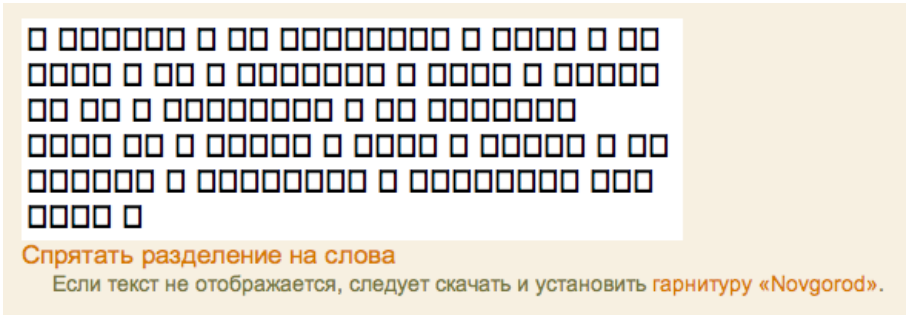


*Fig. 6: Text box of gramoty.ru without font installed*

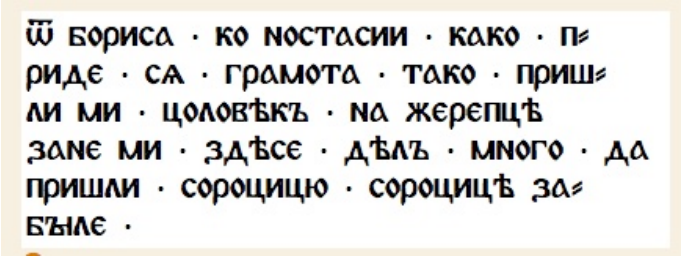


*Fig. 7: Text box of gramoty.ru with font installed*

 Orig. publ. in: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

Let us take a look at the font and its use of Unicode. This font has a basic Latin alphabet and also a Russian alphabet in the correct Unicode blocks; these parts of the font are a Times-like modern typeface. For the 'Novgorodian' glyphs, however, this font relies *completely* on the private use area – not only for those glyphs that were not present in Unicode at the time when the project was realized, but *for the complete alphabet*! In other words: even for those Cyrillic characters that are part of Unicode this project uses the PUA. If this has been done deliberately, it is a gross misunderstanding of Unicode principles. Font changes (i.e. changes in the display – modern vs. historic, for example) are never to be realised by using the private use area – this is what different fonts are for.[17]

Fig. 7 shows the same text (Gramota Nr. 43) with the font properly installed. As this example shows, there is just one letter which has only been introduced in Unicode 5.1, and that is the yery with the back yer (see last line, *byle*). All other character, i.e. 99%, were already present in Unicode 4.1 and would have presented no problem at all to handle in a standard Unicode font without making use of the PUA. (The connected form of the yery with back yer is a variant of the basic glyph which has non-connected parts, so this detail can only be reproduced by a) a photograph, or b) by using a special font. Everything else, however, is available in standard Unicode 5.1.).

What are the consequences of building such an important project on such foundations? First, text copied from the site into a user's documents can only be displayed and printed in the original font, and in no other font. Second, because *all* characters are in the PUA, the site itself has no "search" box for text – a user has to look up the birchbark letters by their numbers. In other words: one cannot search for words, for specific forms, for specific spellings etc. Second, because *all* text is in the PUA, it looses its script-identification, case information etc. This means: the text is no longer identified as being in a Cyrillic script, and no text-processing tool knows anything about which characters form an uppercase/lowercase character pair, how to sort these characters or character strings etc. because such information is only available for characters contained in the standard Unicode blocks. All in all then, this is some kind of "bad practise" for any contemporary web and font project – the uses of the material it offers are severely limited and restricted.

4.3. Using Alternate Fonts for Glyph Variants

One common practice to implement glyph variants without making use of the private use area is to use an alternate font: the basic font would contain the basic glyph of a character, and the alternate font would contain an alternate glyph of the same character in the same Unicode slot. See the following simple example:

| Unicode code point | glyph in font 1 | glyph in font 2 |
|---|---|---|
| character U+0442 | т | т |
| character U+0447 | ч | ч |

Using such alternate fonts will result in a text which conforms to Unicode on the encoding level (both glyphs have the same Unicode number) *and* on the presentation level (the output either on-screen or in print). A slight inconvenience which is connected with this solution is that user must switch fonts if they want to input alternate glyphs – an inconvenience which might be neglected, one might say, because Unicode is not concerned with typing conve-

[17] We won't discuss the quality of the font design as such here. It is, in our view, not perfect either.

nience. A more important aspect of this solution is that for some characters there may be several variants (a, b, c, d, e), while for others there may be only two (a, b) or none. Thus, the largest number of variants that are to be implemented determines the number of fonts required.[18] This possibly results in some redundancy in the fonts (if one does not put only the variants in font no. 2, 3... but also those glyphs which do not change) and possibly some overhead for the system (and any pdf files such fonts could be embedded in). The most important aspect, though, is that this solution solves certain problems only – the handling of glyph variants, but, for example, not the handling of missing characters. The use of alternate fonts can also be a solution, if support for OpenType features (see below) is lacking in the text processor being used.

## 5. OpenType Features

Like Bitmap, PostScript and TrueType, OpenType is a font format. OpenType fonts are the successor to PostScript fonts (they draw characters using the same type of mathematical description); they are files ending in .otf . At the same time they are cross-platform fonts – Macs (running OS X) and PC's can both use these types of fonts and font files which removes any remaining obstacles in cross-platform compatibility of any text which makes use of such a font. But OpenType is more than that. OpenType is also a font technology which can be used for advanced typography. Font files of the .otf type do not automatically or necessarily have such advanced features, but they *can* have them, and older fonts can't. OpenType fonts, however, should not be confused with "free fonts" or public domain fonts. Because the implementation of OpenType features needs a lot of time and experience, the majority of fonts that have OpenType features at present are commercial.

What distinguishes OpenType fonts from older font formats is that they can contain not just the character outlines, but some scripting, too. In other words: OpenType fonts have some characteristics of small programs: a scripting language is being used to write instructions, the instructions are then compiled and stored along with the character outlines in the font file. Because these small programs inside fonts have their own syntax, they need to be tested and, if necessary, debugged before they will compile. This scripting language has been developed by Adobe – for more information, see their website.[19] For OpenType features to work, application programs must know what they do, which is why there is a registry of OpenType features.[20] Some such very basic OpenType features may be used to solve many requirements that Slavists still have even after the release of Unicode 5.1. Below, we will introduce some of them and give a few examples to show what they are good for and what can be expected from them.

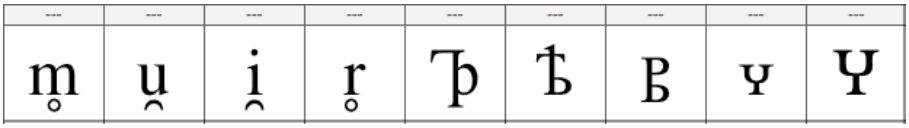

*Fig. 8: Sample OpenType-defined glyphs*

[18] From the author's own experience it can be said that for OCS a three or four font solution will usually cover normal typesetting requirements. For example, the author has created several such solutions where font A has all basic glyphs, font B variants, and font C superscripts. A fourth font might contain additional glyph variants, ligatures etc.

[19] See http://www.adobe.com/type/opentype/.

[20] See http://www.microsoft.com/typography/otspec/featurelist.htm.

In the above figure, you can see several such examples that have been created for the purpose of this paper: some Latin characters with diacritics (combinations that are very important for Slavic phonetics), a ligature, and some character variants. Above the characters in the small cells, you can see some hyphens (i.e. ---). In the font editor this screen-shot was produced from the hyphens indicate that the character in the box below does not have a Unicode number assigned. None of these sample characters are part of the Unicode standard, yet they can all be present in a Unicode font without having been placed in the private use area. This is exactly what OpenType features can do.

5.1. Feature ‘liga’

First, let us take a look at the behaviour of a text processor using some common font features. Normally, all fonts contain ligatures for the character combinations ⟨f‿i = fi, f‿l = fl⟩. In a text processor which supports OpenType features, when a font which contains appopriate OpenType features is being used, the following will happen: as soon as one types the letters ⟨f⟩ + ⟨i⟩, they will automatically be replaced by the much better looking ligature ⟨fi⟩ – see the figure below.

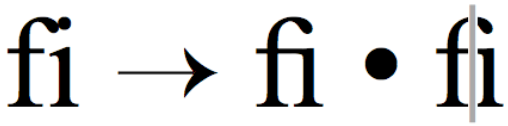


*Fig. 9: OpenType Ligature: Still Two Characters*

However, the application still “knows” that this a string a *two* characters and treats them accordingly. A quick test – and proof – is this: if you move the cursor left one step (using the cursor keys on the keyboard) it will be right in the middle of the ligature – see the grey bar in the above fig., which indicates the position of the cursor. And if the cursor is behind (right) to the ⟨f i⟩ and you delete the last letter, i.e. ⟨i⟩, the application will *not* delete the complete ligature ⟨fi⟩, but only the second letter and return to the first letter to its basic form, i.e. to ⟨f⟩. As we said, this is the behaviour of an OpenType-savvy application. In an application which does not support such advanced features, one would have to *type* the ligature as such and it would ‘count’ as a single letter. Let us stress this once more, because it is very important to realize the possibilities of this feature: *In an OpenType-aware application, a ligature may be used on the presentation level (for the output) where the text encoding has two distinct characters!*

Now imagine that the second ‘character’ used in such a ‘ligature’ is not a character in the usual sense, but a diacritic – this is how the first four samples in fig. 7 can be achieved: as soon as you type an ⟨m⟩ followed by a ‘combining ring below’ this combination will automatically be replaced by pre-accented combination.

The OpenType script that accomplishes this looks like this:

```
feature liga {
 sub u breveinvertedbelowcmb by ubrevebelow;
 sub i breveinvertedbelowcmb by ibrevebelow;
 sub m ringbelowcmb by mringbelow;
 sub r ringbelowcmb by rringbelow;
} liga;
```

The syntax of this instruction is simple: ‘sub[stitute] X Y by Z;’ In this expression, ‘X’, ‘Y’ and ‘Z’ are the names of the characters which are used in the font. Such instructions can be

 Orig. publ. in: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

used, for example, to solve the problem of precisely placing diacritics over/under characters of varying width or with high stems, over lowercase and uppercase letters etc. Of course, this also works for stacking two or more diacritics over a letter.

In the next figure (Fig. 10) some sample applications of this feature to OCS glyphs are shown. These examples demonstrate another important consequence: Unicode may have only one titlo in its repertoire, but using OpenType features it is possible to have titla of different length above narrower or broader characters!

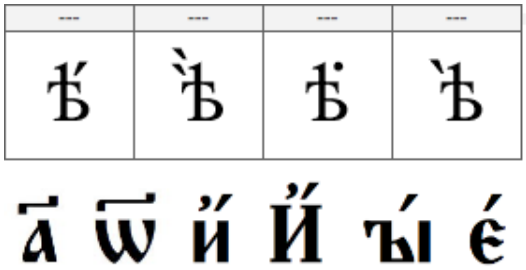

*Fig. 10: Sample OpenType 'Ligatures': placement of diacritics*

5.2. Feature 'dlig'

The OT feature 'liga' is always applied to the source text (although the replacement can be restricted to certain languages). No let us turn to a ligature like the Cyrillic ⟨тр⟩, as shown below. In contrast to the precise placement of diacritics as in our first example (OT feature 'liga'), one would not want to have *all* instances of ⟨тр⟩ *automatically* being replaced by a ligature – in some (rare) cases they should be replaced, but normally they should be left untouched. For such ligatures which apply only at the user's discretion, the OpenType feature 'dlig' has been defined. The next fig. (Fig. 11) shows an example of its implementation in the author's *RomanCyrillic Std* font.

It is obvious that to make such discretionary ligatures work some kind of user interface to this function is needed in the application that it is being used to typeset the text.

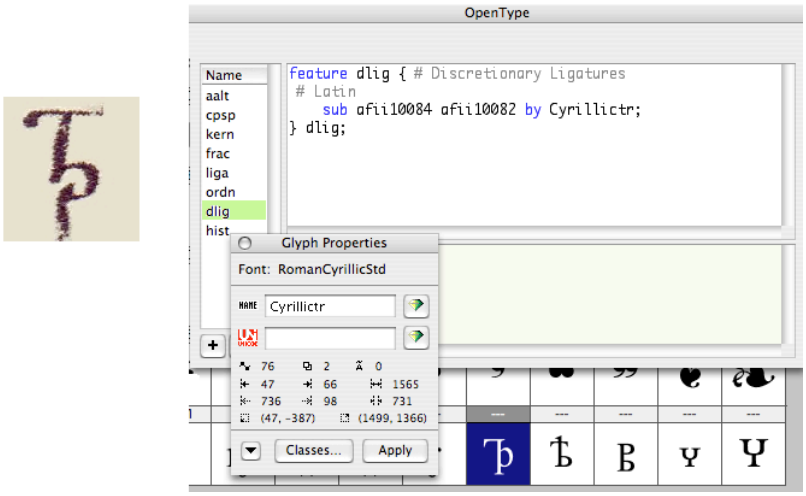


*Fig. 11: Sample OpenType Feature definition and programming interface*

5.3 Feature 'hist'

The OpenType feature 'hist' has been registered to substitute a 'modern' glyph by a historic variant. Originally intended to give a text some 'historic flair', it is very useful for Slavic me-

dievalists. This feature can be ideally used to implement glyph pairs like ⟨ч – ҷ, щ – щ, і – ı, а – ⍺⟩ etc. Again, this is a feature that normally one would not want to apply in all cases. However, if the text file contains a historical text and nothing else, another solution might preferable – an alternate font which has the historic variants as their basic glyphs. In fact, this is exactly what constitutes the difference between our UC 5.1-compliant 'RomanCyrillic Std' and 'Kliment Std' fonts. In the basic font, i.e. in RomanCyrillic Std, the historical glyphs have been implement in the way described here. The syntax of the feature is similar to others already shown:

```
feature hist {
   sub Iotifieda by Iotifieda.hist;
   sub iotifieda by iotifieda.hist;
 } hist;
```

In this example, one uppercase/lowercase pair ⟨Ꙗꙗ⟩ is substituted by its historical variant ⟨Ѣꙗ⟩; it is one of the letter pairs added in Unicode 5.1.

5.4 Feature 'aalt'

The last OpenType feature we would like to mention here, is 'aalt'. Its purpose is a bit different, and consequently the syntax differs, too. The feature offers a selection of alternate glyphs for a given character. In the following example, one alternate glyph is being offered for the uppercase character, and two are offered for the lowercase character. The names of the alternate glyphs consist of two parts: the first part is identical to the name the basic glyph has in the given font, and the second part is a suffix that identifies an alternate by giving it a number. Of course, the '.a1' lowercase and uppercase substitution should match in design, i.e. form a case pair.

```
feature aalt{
   sub Iotifieda from [Iotifieda.a1];
   sub iotifieda from [iotifieda.a1 iotifieda.a2];
} aalt;
```

This feature has been used to offer dozens of alternate Cyrillic glyphs in the updated 'Roman-Cyrillic Std' font; in fact, the font features all major variants one would normally need. As should be clear from the preceding, this again is a feature which relies on user interaction and, consequently, on a user interface that applications must offer to access this feature. It is not sufficient that a font has such an OpenType feature built-in: without application support for OpenType features the feature itself is useless.

5.5 User Interfaces, Application Support and Compatibility

As we have pointed out in the above context, OpenType features can work globally ('liga') or the user's discretion ('dlig'). An application must be able to turn the latter feature on and off for strings of characters. Other features need a user interface for a selection process to take place if a character is selected ('aalt'). Currently, only advanced typesetting applications (like InDesign from Adobe) support such features; other text editors support certain features only (like TextEdit, the OS X text editor), and still others may support none (Word 2004, for example). More and more OpenType support is being added to Operating Systems (like Vista and OS X), and 'pro' fonts from commercial vendors also usually have some OpenType features built-in. The general expectation, then, is that OpenType support has only just started

on wider scale and that its usefulness and application will absolutely grow in the foreseeable future. Below, the user interface to OpenType features in InDesign is shown.[21]

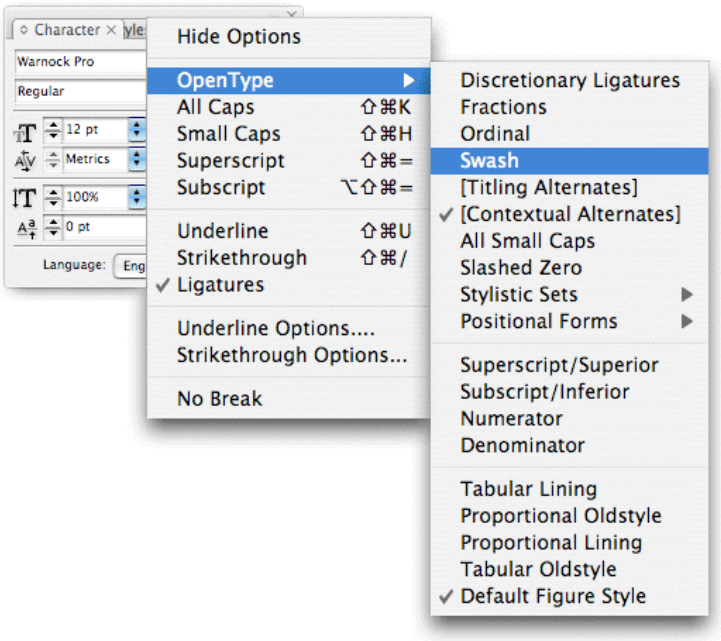


*Fig. 12: User interface to OpenType Features (Adobe InDesign)*

The next illustration shows a simple example of how the glyph palette in InDesign works: a character in the text is selected, and the glyph palette shows the two additional variants that are available for the given character. The desired variant can then be selected using the mouse. The glyph palette can also be set to another view: showing all variants in the font at the same time.

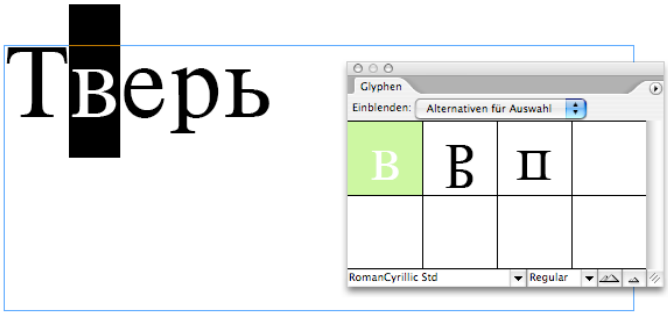


*Fig. 13: Glyph Palette in Adobe InDesign*

An important questions that stems from the use of such open features in one application is the following: what happens if text that uses some of these features is being copied from an OpenType-savvy application to an OpenType-ignorant application, say from InDesign to Word 2004? This has been tested by the author and we can confirm now that what happens is what has to be expected in a Unicode context: the application that does not support the OT feature in questions will display the basic glyph as defined in the font that is being used. In other words: one might lose an alternate character shape, but an 'a' is still an 'a'. Why is this to be expected and why is this a good result of our testing? The result shows that the basic glyph and the alternate variant do indeed have the same Unicode points (their Unicode names

[21] Illustration taken from the Adobe web-site: http://www.adobe.com/type/opentype/.

are simply differentiated by suffixes added at the end). In other words, the result shows that one can have alternate glyphs in a font without putting them into the Private Use Area.

### 5.6 Recommendations

Let us sum up some points that could serve as a first attempt at formulating some recommendations for creating fonts that especially serve the needs of medievalists.

▶ For precise placement of (single or stacked) diacritics use the OT feature 'liga'. Under current Unicode policy demanding more precomposed characters will have no chance of being accepted.

▶ For all ligatures that serve as typographic embellishment of a text on a case-by-case basis, use the OT feature 'dlig'.

▶ For the selection between a 'normal' glyph and a (single) historical variant, use the OT feature 'hist'.

▶ For the selection of a specific glyph from a group of glyphs, use the OT feature 'aalt'. This selection may include historical variants (as defined above) and/or stylistic variants.

None of the aforementioned OpenType features require that the Private Use Area is used.

## 6. Resume

In our paper, we hope to have shown how important a progress especially for Slavists has been made from Unicode 4.1 to Unicode 5.1. All 'normal' characters and diacritics are now in place with only some minor work left to be done. For all issues that are still open, we have discussed in our paper various strategies and solutions – writing a proposal to Unicode Inc., using alternate fonts or using the Private Use Area of Unicode. Finally, and perhaps most important, we have demonstrated how some OpenType programming can be put to good use in such areas as precise placement of diacritics, ligatures, glyph variants etc. Our free reference font, *RomanCyrillic Std*, in its latest version (v. 3) implements all of the Slavic additions to Unicode 5.1 (plus many more) and also contains some extensive OpenType programming so it can serve as technology demonstration and a reference at the same time.

**References**

Birnbaum, David

1996 Standardizing characters, glyphs, and SGML entities for encoding early Cyrillic writing. *Computer Standards and Interfaces* 18, 201-252.

Birnbaum, David

2002 *Unicode for Slavic Medievalists*. Presentation for Sofia conference. < http://clover.slavic.pitt.edu/~repertorium/resources/ unicode_sofia_1_post.pdf >

Birnbaum, D., Cleminson, R., Everson, M.

2002 Additional letters for Early Cyrillic. Draft. Online available at < http://www.evertype.com/standards/ iso10646/pdf/n2xxx-cyrillic.pdf >

Birnbaum, D., Cleminson, R., Kempgen, S., Ribarov, K.

2008 Character Set Standardization for Early Cyrillic Writing after Unicode 5.1. In: *Scripta & e-Scripta* vol. 6, Sofia 2008, 161–193.

Cleminson, Ralph

2006a *Preliminary Proposal*. < http://userweb.port.ac.uk/~cleminsr/Unicode.pdf >

2006b *Proposal for additional cyrillic characters*. < pdf; identical in contents to 2006a >

Čerepnin, L.V.

1956 *Russkaja paleografija*. Moskva.

Diels, Paul

1934 Altkirchenslawische Grammatik. Mit einer Auswahl von Texten und einem Wörterbuch. II. Teil: Ausgewählte Texte und Wörterbuch. Heidelberg.

Everson, Michael et al.

2007 *Proposal to Encode Additional Cyrillic Characters in the BMP of the UCS*. < http://std.dkuug.dk/jtc1/sc2/wg2/docs/n3194.pdf >

Kempgen, Sebastian

2006a Unicode 4.1 and Slavic Philology – Problems and Perspectives (I). In: A. Miltenova, D. Radoslavova, E. Pancheva (eds.), *Computer Applications in Slavic Studies. Proceedings of Azbuky.net. International Conference and Workshop. 24-27 October 2005*, Sofia, Bulgaria. Sofia, 131–159.
Online available at https://fis.uni-bamberg.de/handle/uniba/49294

2006b Unicode 4.1 and Slavic Philology – Problems and Perspectives (II). In: T. Berger, J. Raecke, T. Reuther (eds.), *Slavistische Linguistik 2004/2005*, München, 223–248.
Online available at https://fis.uni-bamberg.de/handle/uniba/49295

 Orig. publ. in: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

**Bibliographische Angaben / Bibliographical Entry:**

Sebastian Kempgen: Unicode 5.1, Old Church Slavonic, Remaining Problems – and Solutions, including OpenType Features. In: *Slovo: Towards a Digital Library of South Slavic Manuscripts. Proceedings of the International Conference, 21–26 February 2008, Sofia, Bulgaria*. Sofia 2008, 200–219.

The original presentation given at the conference is available separately.



© Prof. Dr. Sebastian Kempgen 2021; ORCID: 0000-002-2534-9423
Bamberg University, Germany, Slavic Linguistics
https://www.uni-bamberg.de/slavling/personal/prof-em-dr-sebastian-kempgen/
mailto:sebastian.kempgen@uni-bamberg.de




January 2021, postprint