

T-DOSE 2011

UNI vs CODE

The Way Unicode Does Not Solve Every Problem
It Was Designed To Solve

as computers grow more powerful
we find more ways
to preserve every tiny bit of legacy mistakes
using the power
given us in intent to get rid of these mistakes.

Intro

Besides the fact that Unicode is a tool for building language barriers, it is a broken tool.

Unicode does not solve every problem it was designed to solve and introduced several new ones:

- Unicode significantly contribute to The Encoding Mess
- Some encodings are not closed under catenation
- Unicode still contains “page-switching” symbols
- Unicode is terribly messy and inconsistent
- Unicode does not imply a language of an encoded text
- Unicoded text demands meta-information to be interpreted
- Unicode does not provide a unified multilingual sort order
- **Unicode prevents text recognition!**

In the first place I must admit these bullets are mutually dependent. They are merely faces of a bigger trouble. **The decline of Alphabet idea.**

“Unicode in intent is a representation of graphemes” - says the Standard.

Unicode is not intended to represent alphabet symbols at all. Period.

The difference is not that easy to grasp.

The idea of alphabet is a very abstraction, the alphabet is a finite set of distinct elements, and nothing more! Alphabet symbols are just symbols without any tangible presentation.

Historically we used to use the uncontested graphical presentation of Alphabet. Because of this, some of us had confused the entity with its presentation. And now, when the reality at last provides another (digital) presentation, they are trying to stick with legacy graphemes.

This mistake causes various consequences that I am going to explain beginning with the least one of them.

Unicode Significantly Contributes To The Encoding Mess

They claimed they have a solution for The Encoding Mess (or at least many of us had hoped for it): the single unified representation instead of many ones.

But what do we have instead? Several extra encodings. How many? Let's count:

- UCS-2
- UTF-1
- UTF-5
- UTF-6
- UTF-7
- UTF-8
- UTF-16
- UTF-32
- UTF-EBCDIC
- UTF-GB18030

Total: 10

Well, you may say, some of them are extinct.

Fortunately! You are right. They are already extinct... hardly survived for few years... Don't you think that so rapid extinction is a symptom of something wrong?.. I hope nobody is using them right now.

Well, let's count the rest:

- UCS-2
- UTF-8
- UTF-16
- UTF-32
- UTF-EBCDIC
- UTF-GB18030

Total: 6

Note: UCS-2 and UTF-16 are not the same, but SIMILAR. UTF-16 is intended to replace UCS-2, but it is still in use.

Look, I had 5 “classical” Russian encodings:

- ISO-8859-5
- CP-1251
- CP-866
- KOI8-R
- Macintosh

And now I have 6 more. Isn't it significant?

You may object: the “classical” encodings must die.

For the sake of an argument, OK, you are right. Then I have 6 instead of 5. What a wonderful solution for the encoding mess!

Really? Only 6? Of course not! I forgot about ENDIANNESS (applicable to UTF-16 and UTF-32). So we have 8 encodings.

And to make this farce complete, the creators have introduced Byte Order Mark, which is of course OPTIONAL - there are encodings with BOM and without BOM (applicable to UTF-8, UTF-16, UTF-32 (yes! It is inconsistently applicable to UTF-8 and even more inconsistently inapplicable to the rest encodings (why?!))).

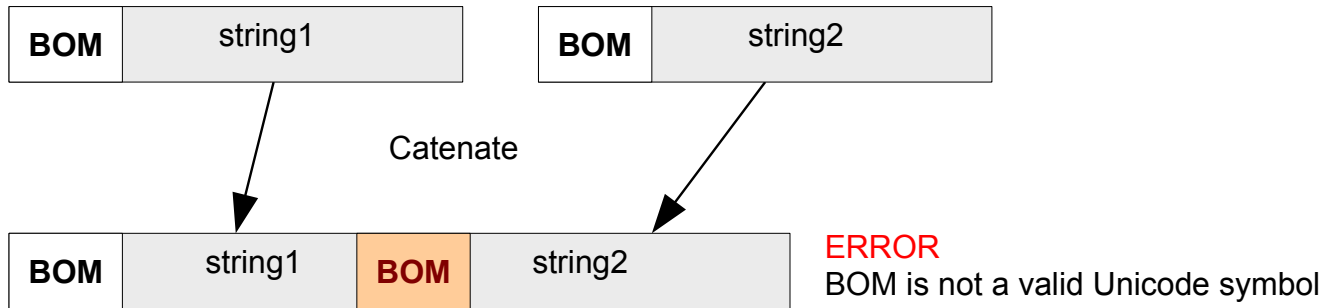
<ul style="list-style-type: none">• UCS-2• UTF-8• UTF-16• UTF-32• UTF-EBCDIC• UTF-GB18030	<ul style="list-style-type: none">• Little Endian• Big Endian	<ul style="list-style-type: none">• With BOM• Without BOM
<p>+ 2 more</p> <ul style="list-style-type: none">• UTF-16LE• UTF-32LE		
Feel free to compose all possible combinations!		

And moreover they have burdened this mess with VERSIONING. The current version of Unicode is 6.1. Tell me what version does your software support? Are You sure? Honestly, I can not tell. The standard is a thing that was intended to FIX THINGS and it is constantly changing. Who can chase it? But this is the subject of another discussion.

Some Encodings Are Not Closed Under Catenation

If we have two valid strings (A and B) with BOM then the concatenation of them (cat A B) is not a valid string. No more and no less. Simple and painful.

This pain is caused by BOM, because the BOM itself is not a valid unicode string.



This is the reason for many developers to hate BOM. And this feature caused tons of bugreports since it was invented.

And, moreover, BOM is meaningless without mentioning of encoding. So we have to specify encoding anyway. Then why should we specify endianness separately? We always have a possibility to specify endianness in one piece with encoding specification. So the BOM is a useless piece of junk.

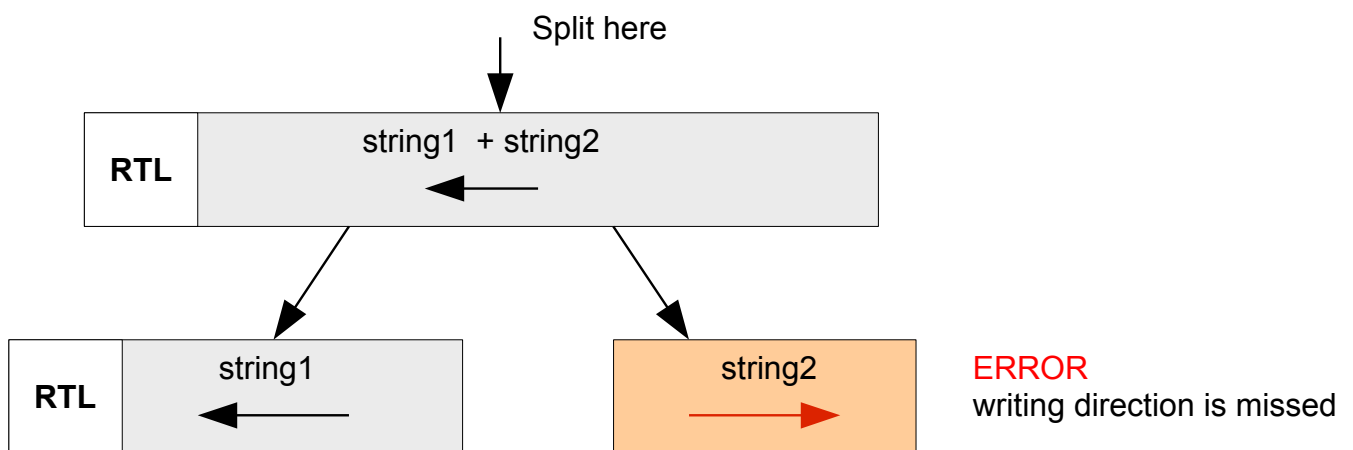
Unicode Contains Page-Switching Symbols

Historically there is an alternative way to codify multilingual strings - codepage-switching symbols. This way has its own natural disadvantages. A page-switching symbol creates a context that affects all subsequent symbols. And Unicode supporters proudly claimed that Unicode design is particularly good for not having codepage-switching symbols. Then Unicode strings could be split between ANY adjacent symbols without any damage.

It is wrong.

Unicode contains at least “writing direction” control symbols (LTR is U+200E and RTL is U+200F) which role is IDENTICAL to the role of codepage-switching symbols with the associated disadvantages. The “Writing-Direction” symbols do create a context that affects subsequent symbols.

It means that you can not randomly split a Unicode string (see the figure below). You always **MUST** search for a context until the beginning of a string. This is an unpleasant property that the Unicode creators were afraid of, nevertheless they had implemented.



But if Unicode controls writing direction anyway, why does it have only two options RTL and LTR? How about Hangul? How about diacritical marks which are from some point of view just a special case of writing direction.

Unicode Is Terribly Messy And Inconsistent

Unicode contains the letter “A” U+0041 and the diacritic mark “BREVE” U+0306, and the standard prescribes a method to combine them to represent a letter “Ă” U+0041 U+0306. And at the same time Unicode contains “Ă” U+0102.

And how do you think this diversity affects a textual search?!

A	U+0041	
˘	U+0306	U+02D8
Ă	U+0041 U+0306	U+0102

Duplicates ^^

And there are tons of diacritical letters (such as “À Á Â Ã Ä” and others) - whole codepane of a complete trash! Why did they plant hundreds of diacritical codepoints alongside alternative presentation?

...taking in account that this presentation (a letter itself + a diacritical mark) is far more rational and flexible, and surprisingly more useful for a textual search, since it is very easy to ignore diacritical marks if needed.

Moreover, Unicode contains EVERY diacritical mark TWICE: one to be combined with a letter and one to stand alone, with absolutely identical appearance.

˘	U+0306	U+02D8
	Combinational	Standalone

Is there any reason to maintain the both codepoints?

Regardless of uselessness of standalone diacritical marks, I see at least two ways to avoid duplication.

- (1) Throw away standalone marks, and combine marks with a space character. Pretty natural and simple way.
- (2) Throw away combinational marks, and introduce a set of combination control symbols, taking in account the variety of combinations.

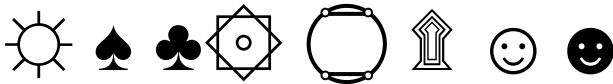
˘	U+0306	U+0020 U+0306
˘	Combine Symbol U+02D8	U+02D8
	Combinational	Standalone

The second way evidently allows us to combine more than one mark and to combine them in more than one manner. And finally it allows to codify Hangul properly and save the whole codeplane.

But prudence is not a virtue of the Unicode creators.

They had grabbed 32 bits -- more than FOUR BILLIONS codepoints! And now they are trying with great urge to fill the space with something... mostly with trash symbols.

I do not complain about useless codepoints: “domino tiles”, “stars”, “cars”, “teddy bears” you name it



there are thousands of them greater than U+1F200 - they simply do not have a reason to exist.

...I am interested in the symbols distorting the system of Unicode, such as vulgar fractions.

There are $\frac{1}{3}$ $\frac{2}{3}$ $\frac{1}{8}$ $\frac{3}{8}$ $\frac{5}{8}$ $\frac{7}{8}$... So what about 17/37 ???

17/37 symbol does not exist.

It is inconsistency. And not the only one.

Well, tell me would you be surprised to find some RED graphemes in Unicode?

Yes, I mean red colour... I know, it sounds absurd.

As absurd as Unicode is!

Since Unicode contains superscripts and subscripts (U+2070..U+2094).

And Unicode contains ITALIC and BOLD (U+1D400..U+1D49B)!

The next logical step is to introduce colours in Unicode. So answering the question I would **not** be surprised to find a RED grapheme one day.

By the way, there is another inconsistency. To codify superscripts Unicode must contain ENTIRE Unicode. This is endless recursion. This is why they codify only few superscripts, and this is why they should not do it at all- superscripts should be codified with control symbols. ...and bold and italic too. (hmm... I wonder, if I ask Open Office to display a bold grapheme in a bold font would it be bolder?)

Another issue is duplicate graphemes. The noticeable examples are “μ”, “omega”, “K”, “v” “i” (and many many more) - each of them is associated with the two codepoints. You are certainly curious, how it have happened? Simple enough... The alternatively gifted creators of Unicode decided to create a codepoint for “micro”-prefix, a codepoint for “Ohm” sign, a codepoint for “Kelvin” sign, a codepoint for “roman numeric 1” sign, thus, duplicated graphemes were planted.

μ	U+03BC	U+00B5	μ	Micro sign
Ω	U+03A9	U+2126	Ω	Ohm sign
K	U+004B	U+212A	K	Kelvin sign
v	U+0076	U+2174	v	Roman 5

Different semantic - same praphem. One codepoint per semantic unit.

What do we codify then? Do you remember that Unicode codifies GRAPHEMES???

This is not my idea, this is the idea of the creators!

“Unicode takes the role of providing unique codepoint for a grapheme” - says the standard.

They did not manage to implement their own idea!!!

You may argue every point but this. This is beyond arguments: **Unicode is not unicode compliant**. Alternatively gifted creators do not follow the unicode standard - they broke their own rules they had imposed.

And besides this epic fail there is another inconsistancy!

μ	U+03BC	U+00B5	micro
Ω	U+03A9	U+2126	Ohm
K	U+004B	U+212A	Kelvin
k	U+006B	???	kilo
M	U+004D	???	mega
n	U+006E	???	nano
N	U+004E	???	Newton
J	U+004A	???	Joul
m	U+006D	???	meter

Why only “micro” is codified personally? What about “kilo”, “mega”, “nano”? Why do they all absent?... What about “Newton”, “Joul”, “meter”?.. Whether Kelvin is more illustrious than Newton?!

All in all Unicode looks like an insane compromise between horribly controversial arguments. It contains many more than these obvious duplicate graphemes.

The creators rightfully noticed many languages (with different alphabets) share the same graphemes, and they intended to codify each grapheme once ...and failed once again. Because **latin and cyrillic share many graphemes!**

In the first place they blended all cyrillic based graphics together, reflecting the fact of the common genetic roots. And then they put latin separately ignoring the same fact.

Please do not confuse modern cyrillic with original cyrillic. This two alphabets have very little in common. The modern cyrillic is just a slightly distorted copy of latin. Yes! Literally. It was copied, and it happened not so long ago, in the early of 18th century.

So we may insist that latin “A” and cyrillic “A” are exactly the same grapheme (associated with two codepoints alongside many other latin graphemes).

Find 10 differences:

A	A
B	B
C	C
E	E
H	H
I	I
K	K
M	M
O	O
P	P
T	T
X	X
Y	Y
Latin	Cyrillic

You may object: we should not confuse Cyrillic and Latin graphemes!

Why should we not? Why should we codify Cyrillic and Latin separately?

May be... for the sake of a textual search... We should not confuse English word “A” with Russian word “A”.

I reply: forget the textual search. You have ALREADY confused english word “A” with czech word “A” and russian word “A” with serbian word “A”. These are 4 different words with 3 different meanings. So the separation Latin from Cyrillic could change nothing. As you are codifying graphemes (by their appearance) there is no reason to separate them by nationality.

Forget the textual search - since you have started to codify appearance of a text - you can not search for a text pattern, **you may only search for something that LOOKS LIKE a desired text pattern**. Simply because of the codified attribute.

What you have codified is what you have to search!

Unicode Does Not Imply A Language Of A Text

While “classical” encodings do.

If you send me a message: “A”

I ask you: “what encoding?”

if you reply: “KOI8-R”

then I know that this message is in Russian

If you reply: “UTF-8”

then I ask you another question: “what language?”

Thus, our negotiation become a bit longer (with Unicode). ...even if we eliminate all “classical” encodings from existence the both questions will remain.

Imagine that we finally elaborated single unified Unicode encoding, then the second question still remains. In other words, any Unicode message to be properly interpreted still needs a meta information explicitly specified outside a message. We can not get rid of this negotiation above.

Do you see any differences to the current situation?! The problem is not solved.

Why do I consider it as a problem?..

Take a look at a database. A database is always about search and order. To perform these tasks it needs metainformation. And it has to store this information somewhere.

Historically Postgresql stored it in a database cluster. This method implies “all the string values in a single database are encoded in the same encoding and are written in the same language” (according to the specified collation).

It was bad for a multilingual content. The metainformation should be specified more preciecly.

Per Table?

It is not enough. We may have (and it is very likely we have) several columns to represent the same string in different natural languages.

So Postgresql now supports PER-COLUMN collation specification.

For example we have 4-column table:

Item_id	description_EN	description_RU	description_NL
---------	----------------	----------------	----------------

Collations of columns are: UTF-8 English

UTF-8 Russian

UTF-8 Dutch

But it is not nearly enough!

If our hypothetical database stores user-defined content, we may have several users who input their data in different languages in the same column.

For example every user may script his name in his native language:

user_id	Name	phone
10	John Doe	555-17-17
11	Вася Пупкин	555-22-22

How to collate the “Name” column???

Should we specify per-value collation? NO! It's absurd, such a specification is meaningless. So the problem can not be solved by means of a database. The necessity of metainformation itself is the issue. Database is innocent.

Let us look at the same example from a different point of view...

Unicode Does Not Provide Unified Sort Order

Which collation to apply to a database column containing multilingual values? How to order it?

Assume we have a telephone book where every name is written in native notation. We want it to be ordered and searchable.

Russian collation table says: “a” < “б”.

Greek collation table says: “α” < “β”.

And who says whether “a” < “α”?

Some may say it is meaningless relation. But he will be wrong.

In fact “a” < “α” and “a” > “α” have exactly the same meaning for an end user, But the possibility of comparison is meaningful. In order to order strings we have to be able to compare ANY two strings with determined result.

Several attempts to elaborate a unified sort order were made... for example European Order Rules partially solved the problem, but partial solution is NOT a solution.

As long as my example involves Greek alphabet, the problem does not look insolvable. But what about sort order of bilingual Russian + Bulgarian content (both cyrillic)? Even if you elaborate the most generalized collation for Unicode, it would not help. This problem has no solution since Unicode has been designed to encode GRAPHEMES.

Unicode allows us to mix up many languages into a single string, but it does not provide a clue to separate them back.

It is the mess.

Unicode Prevents Text Recognition

And what is particularly disgusting that is Unicode makes the whole class of problems completely unresolvable. I mean text recognition problems.

Imagine that a blind man uses a text to audio converter. The converter has to know which language to speak to work properly. And a Unicode string does not contain this information. Our blind man may specify a language manually before reading, but the text is MULTILINGUAL in intent. The converter has no clue when and how to switch speaking language at all. **So with Unicode our blind man is unable to read a multilingual text!**

For example we have a bilingual Russian+Bulgarian text. Russian and Bulgarian use the same graphemes, so Unicode makes no difference between them, at all. But these languages use different phonemes, so a text to audio converter makes a big difference between these languages. And with Unicode it can't tell a Bulgarian word from a Russian word.

Unicode punishes all the blind men for their physical disability! **Unicode withhold the help that a computer had provided.**

A Tale Of A Letter

Would you like to hear a real story about the most troubled letter in Russian alphabet?

Here is the life story of the letter «Ё» («E» with trema). Pronounced always as short, soft and accented “o”, similar to “yo!”. This is the youngest letter in Russian alphabet, introduced in 1783 without a serious reason - to replace the combination “io” - this is definitely not a serious reason.

For a long time it was considered equal to “io” and its usage was optional. Until in 1917 the letter “i” was eliminated and consequently the “ё” usage became mandatory. In fact this decree had the opposite effect. Due to the lack of technical means of typing this new letter, it had been spontaneously replaced with “e”. Though “e” has nothing in common with “ё”, no grammatically nor phonetically (“ё” is sometimes pronounced as “o” and never “e”).

Thus, the struggle about «Ё» had begun. The struggle between the mostly ignorant society and the mostly incompetent government. From time to time our government issues decrees about relation between «Ё» and «E». Sometimes these letters are declared equal, sometimes not-equal.

Once the government had prescribed to not count an absence of dots a missprint, trying to make the life of printing workers easier. Yet in manuscripts every dot was considered mandatory. Later they began to argue whether we need these dots generally or not, and situation became the complete mess.

The dispute still lasts and no one noticed that modern digital machines provide absolutely accurate notation and even DEMAND IT..

Not so many people can notice the real big TROUBLE...

But I have this unfortunate letter in my surname. Consequently I possess several documents signed with DIFFERENT NAMES, such as passports, social security etc... The name depends on a will of a particular clerk scripting a document, sometimes they wish to script “Ё” in my surname, sometimes “E”. AND SOMETIMES THE GOVERNMENT PRESCRIBES TO CONSIDER THESE NAMES DIFFERENT.

You may ask me: How does Unicode affect your personal problems?!

Honestly, a very little. Unicode just add yet another notation of “Ё”. So I had two different names and with Unicode I have three of them.

Formally, from the point of view of a governmental database I am three different persons.

U+0435

U+0401

U+0435 U+00A8

Since we obtained this powerful tool - digital computers - we are constantly trying to preserve the mess created by the ABSENCE of this tool instead of sorting this mess out with the new powerful tool we had never before. And as computers grow more powerful we find more ways to preserve every tiny bit of legacy mistakes using the power given us in intent to get rid of these mistakes. That is the irony.

Instead Of Conclusion

If you want to have a truly multilingual support then any string should imply a language it belongs to. This is absolutely necessary to manipulate multilingual texts. And it was easy to achieve.

Imagine the following hypothetical procedure. We count all known languages, and pick alphabet that each language uses: one alphabet per language. Pretty natural relation, isn't it? Then we simply UNION all alphabets. Then obviously every symbol of this united alphabet would imply the natural language it belongs to. The problem is solved.

And this solution implicitly gives you natural alphabetic order (without collation tables) and plain possibility to map from the formal model to graphical presentation and to audio presentation as well.

On the other hand we still need a graphical presentation. And this task demands exactly opposite approach. An approach similar to the Unicode's one...

Yes. This is the root of Unicode misdesign. They mixed up two mutually exclusive approaches. They blended badly two different abstraction levels: the textual level which corresponds to a language idea and the graphical level which does not care of a language, yet cares of writing direction, subscripts, superscripts and so on.

In other words we need two different Unicodes built on these two opposite principles, instead of the one built on an insane mix of controversial axioms.

And each one of them to occupy his own ecological niche.

But these are merely words.

Unicode has already occupied the both niches.

There is no way back since the whole society has been moved.

Unicode has already murdered the alphabet and replaced it.

Every scripture is no longer a text - it is a composition of glyphs now.

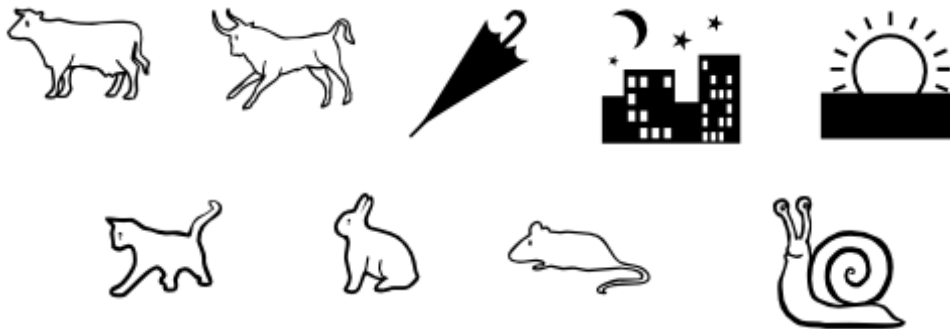
We have refused the Abstraction for the sake of the Tangibility.

The dark ages have begun.

Post Scriptum

If you disagree to what I had said, you just underestimate the role of Alphabet in your everyday life, and I do not force you to read further. Yet for those who are concerned I am sending the following message.

I have paid not enough attention to the junk symbols... Take a look at the U+1F300 and the higher, you must find a multitude of nouns: ice cream, house, holiday... - these are the future hieroglyphs.



All these symbols above are Unicode graphemes! It looks pretty stupid and ridiculous and make you laugh at the alternatively gifted creators of Unicode. I also found it funny at the first look. But it is not that simple. And it is definitely not fun at all. **It is a trap!** Set on for our kids, for your kids! Extremely sophisticated and sadistic trap. It would not be set off tomorrow, and you and me would never set it off. We have too strong affinity with alphabet today. But one day, one child (a mere child without your educational background, without your linguistic habits) discover these cute, nice, funny, nya-nya symbols and immediately find it useful for a tweet, chat, SMS, MMS (underline the appropriate option).

Multiply the effect by the public education overall degradation. And do not forget that the government IS ALREADY TRYING TO DEPRECATE MANUAL WRITING.

And now, my friend (may I call you that, since you had read it?) recall the basic fact that the alphabet is the key for a CREATIVE THINKING - it infects us in our early age with the strong habit of DECOMPOSITION AND ABSTRACTION. I do not know who have decided to turn us back into the dark ages of superstition and magic.