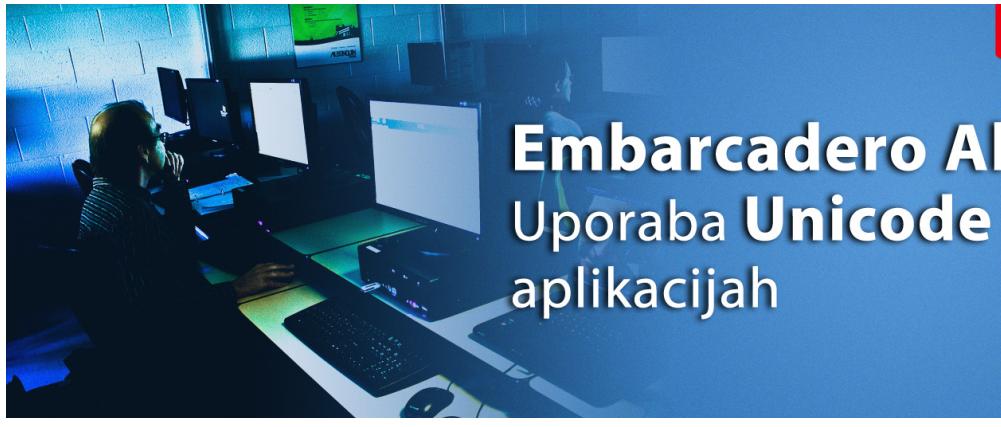


EMBARCADERO AKADEMIJA



## Embarcadero Akademija 2012: Uporaba Unicode v sodobnih Delphi aplikacijah

Primož Gabrijelčič  
<http://thedelphigeek.com>



## Kazalo

Uvod .....	2
Ansi in Unicode.....	3
Ansi .....	3
Unicode .....	4
Kompozicija .....	5
Kodiranja .....	6
BOM.....	7
Novosti v Delphi 2009 .....	8
Nizi – takšni in drugačni.....	9
String[] .....	9
AnsiString.....	9
RawByteString .....	9
UTF8String .....	9
UnicodeString.....	10
WideString.....	10
UCS4String.....	10
Kodiranja .....	10
Character .....	11
CharInSet .....	12
Baze .....	12
Druge spremembe.....	12
Prehod .....	14
Opozorila .....	14
Problemi .....	15
Koda, ki predpostavlja, da je <code>char</code> velik en bajt.....	15
Nizi kot nosilec podatkov .....	16
Pretvorba nizov .....	16
Packanje s kazalci .....	16
Iskanje potencialno problematičnih mest.....	17
Nasveti.....	18
Popolna podpora Unicoda.....	18
Združljivost .....	19
Koristne povezave .....	21

Vsi programi, omenjeni v tem dokumentu, so na voljo na naslovu

<http://17slon.com/EA/EA-Unicode.zip>.

## Uvod

Delphi je z nami že sedemnajst let in od vsega začetka se je – z izjemo nekaj neuspehov stranpoti – razvijal in prilagajal. Večinoma so razvijalci poskrbeli za to, da njihove stranke – torej mi, programerji – nimamo težav s prehodom na novejšo različico, a je včasih vseeno prišlo tudi do manjših ali večjih nezdružljivosti. Prva takšna kleč, s katero pa večina ni imela težav (že zato, ker je bežala proč od Windowsa 3.1), je bil prehod s 16-bitnega Delphija 1 na 32-bitni Delphi 2. Potem smo živeli mirno življenje, nekateri so si privoščili še izlet v Linux, dandanes pa spet razmišljamo o raznih dilemah. Razvijati za 32-bitni ali 64-bitni Windows? Uporabiti VCL ali FireMonkey in spotoma razvijati še aplikacije za OS X? Se ukvarjati s tablicami ali še malo počakati?

Zanimivo je, da se je pred nekaj leti v razvoju Delphija zgodila velika prelomnica, ki jo je marsikdo kar ignoriral – enostavno tako, da ni nadgradil na novejšo različico. Do vključno Delphija 2007 smo razvijali programe, ki so uporabljali uporabniški in programski vmesnik Ansi, od Delphija 2009 dalje pa smo prisiljeni programe predelati, tako da uporablajo uporabniški in programski vmesnik Unicode. [Če se s temi pojmi srečujete prvič, ne skrbite – vse bo podrobno razloženo.]

Ta sprememba je bila predmet obširnih debat in internetnih prepirov. Marsikdo se ni strinjal s pristopom, ki so ga ubrali pri CodeGearu/Embarcaderu (ravno med različicama 2007 in 2009 se je izvršila prodaja). V teoriji je bila stvar preprosta – program prevedete z novim Delphijem in predelava je končana. Če ste pisali »lepo« kodo in niste uporabljali grdih trikov, programa ni bilo treba prav nič spremeniti. A kaj, ko tako kodo najdemo le v primerih, ki se pojavi na predstavitvah, v resničnem življenju pa ne. Zato se marsikdo še vedno ubada s prehodom, nekateri pa z njim niso niti začeli. Največkrat le zaradi tega, ker se prehoda enostavno – bojijo.

V predavanju si bomo ogledali spremembe v Delphiju med različicama 2007 in 2009, posebej pa se bomo osredotočili na težave, na katere utegnete naleteti pri prehodu, in nasvete, kako prehod izpeljati kar najbolj gladko. Ne pozabite – Delphi 7, ki je menda še vedno najbolj uporabljana različica Delphija, je luč sveta zagledal že pred desetimi leti in nikakor ne sodi več med moderna razvojna okolja. Mar ni že čas, da si olajšate programersko življenje in presedlate na novejšo različico?

## Ansi in Unicode

Da bomo lažje razumeli prednosti in pasti prehoda iz sveta Ansi v svet Unicode, najprej obnovimo nekaj osnov.

### Ansi

V kodiranju Ansi je vsak znak (črka, številka, posebni znaki) predstavljen z enim bajtom. Hkrati lahko torej uporabimo največ 256 znakov, v praksi pa še kar nekaj manj, saj se navadno del števil od 0 do 255 ne preslika v izpisljive znake (torej znake, ki jih lahko vidimo na zaslonu).

Preprost zgled – v kodiranju Ansi se moje ime, »Primož«, zapiše kot zaporedje bajtov **\$50 \$72 \$69 \$6D \$6F \$9E**. [Pravzaprav v tem trenutku malo lažem; to za trenutek spreglejte.]

Kako pa v Ansiju zapišemo najrazličnejše pisave? Je v teh 256 znakah prostor za latinico z najrazličnejšimi nacionalnimi znaki, cirilico v vseh različicah in morda še kaj? [Pozabimo za sedaj na arabščino, vzhodnoazijske pisave in podobne komplikacije.]

Odgovor je enostaven – 256 znakov je premalo. Zato je treba pri kodiranju Ansi poleg vrednosti podatkov vedno poznati tudi *kodno stran* (*kodno tabelo*). Kodna stran je predpis, ki določa preslikavo vrednosti od 0 do 255 v znake. Spodnja polovica – znaki od 0 do 127 – je standardna in ustreza standardu ASCII (American Standard Code for Information Interchange), znaki od 128 do 255 pa se razlikujejo.

Zgoraj omenjena pretvorba mojega imena v bajte velja le za kodno tabelo Windows 1250, ki pokriva srednjeevropske pisave. Kot lahko vidite na spodnji sliki, je v kodni tabeli Windows 1251 na mestu \$9E (vrednost znaka »ž« v tabeli Windows 1250) cirilični znak »Ђ«. Če torej zaporedje bajtov **\$50 \$72 \$69 \$6D \$6F \$9E** prikažemo z uporabo kodne tabele Windows 1251, dobimo »PrimoЂ«.

Windows 1250

	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	0	@	P	‘	p	’	I	‘	Ń	Đ	í	đ		
1	!	1	A	Q	a	q	‘	‘	Á	Ń	á	ń		
2	”	2	B	R	b	r	‘	‘	À	Ñ	â	ñ		
3	#	3	C	S	c	s	‘	‘	Ł	À	Ó	ä	ó	
4	\$	4	D	T	d	t	“	“	Ä	Ó	ä	ô		
5	%	5	E	U	e	u	…	…	À	Ó	í	ő		
6	&	6	F	V	f	v	†	-	I	Ó	ć	ö		
7	’	7	G	W	g	w	‡	-	Ş	Ç	×	ç	÷	
8	(	8	H	X	h	x	‘	‘	Č	Ř	č	ř		
9	)	9	I	Y	i	y	%	TM	ë	Ü	é	ü		
A	*	:	J	Z	j	z	Š	Š	š	E	Ú	é	ú	
B	+	:	K	[	k	{	<	>	»	É	Ú	ë	ú	
C	,	<	L	\	l		Ś	ś	-	Ł	Ę	ë	ü	
D	-	=	M	]	m	}	‘	‘	‘	Í	Ý	í	ý	
E	.	>	N	^	n	~	Ž	Ž	®	Í	Í	î	ť	
F	/	?	O	-	o	Ž	ž	ž	ž	Đ	Þ	d'	‘	

Windows 1251

	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	0	@	P	‘	p	’	I	Ђ	Ń	Đ	í	đ		
1	!	1	A	Q	a	q	‘	‘	Á	Ń	á	ń		
2	”	2	B	R	b	r	‘	‘	À	Ñ	â	ñ		
3	#	3	C	S	c	s	‘	‘	Ł	À	Ó	ä	ó	
4	\$	4	D	T	d	t	“	“	Ä	Ó	ä	ô		
5	%	5	E	U	e	u	…	…	À	Ó	í	ő		
6	&	6	F	V	f	v	†	-	I	Ó	ć	ö		
7	’	7	G	W	g	w	‡	-	Ş	Ç	×	ç	÷	
8	(	8	H	X	h	x	‘	‘	Č	Ř	č	ř		
9	)	9	I	Y	i	y	%	TM	ë	Ü	é	ü		
A	*	:	J	Z	j	z	Š	Š	š	E	Ú	é	ú	
B	+	:	K	[	k	{	<	>	»	É	Ú	ë	ú	
C	,	<	L	\	l		Ś	ś	-	Ł	Ę	ë	ü	
D	-	=	M	]	m	}	‘	‘	‘	Í	Ý	í	ý	
E	.	>	N	^	n	~	Ž	Ž	®	Í	Í	î	ť	
F	/	?	O	-	o	Ž	ž	ž	ž	Đ	Þ	d'	‘	

Da pa življenje ne bi bilo preveč enostavno, obstajajo različni standardi, ki se uporablja hkrati in so – kaj pa bi pričakovali drugega v računalniškem svetu – nezdružljivi. Srednjeevropske znake lahko prikažete v raznih kodnih tabelah – v svetu Windows tako srečamo DOS-ovsko kodno stran 852, Windows 1250 (pričakana zgoraj) in ISO standard 8859-2. Seveda ni treba posebej omenjati, da so naši šumniki v vsaki od njih na drugem mestu. Cela zmeda!

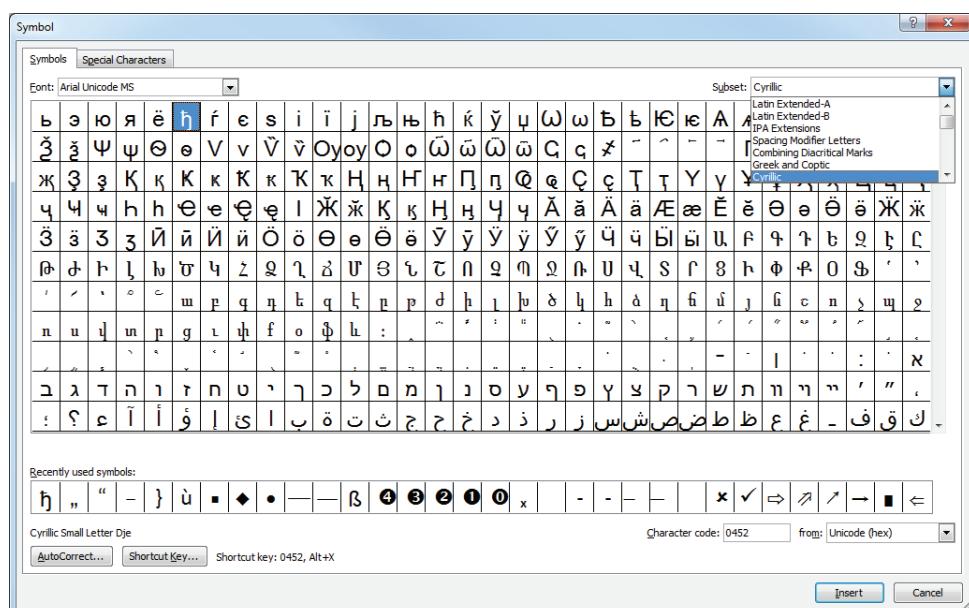
»Standard« Ansi prinaša s sabo kar nekaj problemov. Zapleta internacionalizacijo in lokalizacijo aplikacij, saj moramo poleg prevoda vedno poskrbeti za pravo kodno stran. Še bolj se zapleteta vnos in shranjevanje geografsko različnih podatkov. Kar pomislite – kaj če bi morali v isti bazi podatkov hraniči slovenska in ruska imena? Popolnoma pa Ansi odpove pri jezikih, ki imajo več kot 256 znakov. In to so jeziki, ki jih govorijo (in pišejo) ne milijoni, temveč milijarde!

Delno rešitev zapisa kitajskih, japonskih, korejskih in drugih pismen je prinesel standard DBCS (Double-Byte Character Set), v katerem se je en znak zapisal z enim ali dvema bajtoma. Zaradi tega se je zapletlo vsakršno delo s takimi nizi podatkov (for zanke pač niso uporabne, če ne veš, za koliko bajtov se moraš premakniti v enem koraku) in na srečo v sodobnem programiranju kodiranje DBCS ni zelo pomembno. V našem geografskem okolju pa sploh ne.

## Unicode

Pravo rešitev je prinesel šele standard Unicode, v katerem lahko, vsaj teoretično, definiramo znake s številskimi kodami od 0 do \$10FFFF, kar znese 1,114.112 znakov. To je pa že toliko, da lahko enolično predstavimo vse znake vseh svetovnih pisav, naravnih in umetnih, matematične znake, notni zapis in še in še. Pa bo še vedno ostalo ogromno prostora. Standard Unicode se stalno razvija; trenutna različica 6.1 definira več kot 110.000 znakov iz 100 svetovnih pisav ter pokriva skoraj vse »žive« pisave.

Za delno združljivost z zapisom Ansi so poskrbeli tako, da je prvih 256 znakov Unicoda enakih zahodnoevropski kodni tabeli ISO 8859-1. Naprej pa so znaki razdeljeni v skupine po tematiki (na primer »cyrilica«, »arabščina«, »matematični znaki« itd.)



Celotni nabor znakov Unicode (pravzaprav se ekvivalentu neke številčne vrednosti pravilno reče *kodna točka* (code point) in ne *znak*) je razdeljen na *ravni* (plane). 17 jih je, vsaka pa vsebuje 65536 znakov. Zapis je lepši v šestnajstiškem sistemu – vsaka raven vsebuje znake od \$0000 do \$FFFF, ravni pa so oštevilčene od \$00 do \$10. Največja kodna točka ima torej vrednost \$10FFFF, kar je številka, ki smo jo že omenjali.

Prvo, najpomembnejšo raven imenujemo *osnovna večjezična raven* (basic multilingual plane), kodne točke v njej pa zapišemo s štirimestno vrednostjo (uvodni \$00 izpustimo). Črka »A«, ki ima ASCII kodo 65 (ozioroma šestnajstico \$41), je tako predstavljena z vrednostjo U+0041. Če imamo znak iz katere od višjih ravn, pa uporabimo vseh šest števk. Znak \$01D11E bi torej zapisali kot U+01D11E.

Osnovna raven je hkrati tudi najpomembnejša in vsebuje najpogosteje uporabljane znake. Raven 1 vsebuje dodatne znake za različne jezike, raven 2 dodatne znake za ideografske pisave (na primer kitajščino), ravni 3 do 13 še niso uporabljeni, raven 14 hrani znake za posebne namene, ravni 15 in 16 pa sta namenjeni privatni rabi. V njiju lahko definirate lastne znake – če imate seveda na voljo pisavo, ki te znake vsebuje. Prikaz zapisov Unicode je poseben problem, saj zaradi ogromnosti standarda ni na voljo veliko pisav, ki bi vsebovale vse definirane znake. V svetu Windows se popolnosti še najbolj približa Arial Unicode MS.

## Kompozicija

Ena od posebnosti standarda Unicode je možnost, da nekatere znake zapišemo na dva različna načina. Pravimo, da imajo znaki sestavljeni (precomposed) in komponentno (component) obliko. Takšni so vsi znaki z akcenti (ozioroma z diakritičnimi oznakami), pa Hangul (zapis korejskega jezika) in še kaj bi se našlo.

Oglejmo si oba zapisa na enostavnem zgledu. Znak »ö« najdemo v sestavljeni obliki na mestu U+00F6. Lahko pa namesto njega uporabimo znak »o« in mu dodamo diakritično oznako »''. V tem primeru dobimo dve kodni točki – U+006F, U+0308 – njun rezultat pa je popolnoma enak – »ö«.

Diakritične oznake lahko tudi kombiniramo in tako en znak opremimo z več kot enim »dodatkom«. Na tak način lahko pridemo do kombinacij, ki se ne pojavljam v nobenem svetovnem jeziku, pripeljejo pa do »zanimivih« rezultatov, kar dokazuje tudi spodnji odlomek slavnega prispevka »*RegEx match open tags except XHTML self-contained tags*« s spletišča StackOverflow ([stackoverflow.com/q/1732348/4997](http://stackoverflow.com/q/1732348/4997)).

ex parsers for HTML will instantly transport a programmer's consciousness into a world of ceaseless screaming, he comes, the pestilent sithy regex-infection will devour your HTML parser, application and existence for all time like Visual Basic only worse he comes he comes do not fight he comes, his unholy radianice' destroying all enlightenment, HTML tags leaking from your eyes like liquid pain, the song of regular expression parsing will extinguish the voices of mortal man from the sphere I can see it can you see if it is beautiful the final snuffing of the lies of Man ALL IS LOST ALL IS LOST the pony he comes he comes he comes the ichor permeates all MY FACE MY FACE oh god no NO NOOO NO stop the angels are not real ZAI GO IS TONY THE PONY, HE COMES

## Kodiranja

Ena od težav standarda Unicode je velika poraba prostora. Za vsak znak potrebujemo tri bajte, tri pa je v računalništvu zelo nerodno število. V praksi bi torej morali porabiti štiri bajte na znak (in prav res obstaja takšna oblika zapisa, UTF-32), a si lahko pomagamo z raznimi triki in porabo pomnilnika bistveno zmanjšamo. To dosežemo z uporabo enega od spodaj opisanih *kodiranj*.

### UTF-32

Že omenjeni UTF-32 (imenovan tudi UCS-4) za vsak znak porabi štiri bajte. Ima eno samo, a veliko slabost – ogromno porabo prostora. Prednost pred drugimi kodiranji je enostavna obdelava nizov, zapisanih v tem zapisu. Kot bomo videli, vsa druga kodiranja uporabljajo spremenljivo dolžino kodiranja enega znaka (ozioroma ene kodne točke), kar zaplete premikanje po nizu, kopiranje in podobno. V kodiranju UTF-32 pa so vse takšne operacije popolnoma enostavne – prav tako enostavne kot v svetu Ansi.

UCS = Universal Character Set

UTF = Unicode (ali UCS)

Transformation Format

Obstajata dve različici kodiranja UTF-32 – *big endian* (UTF-32BE; najpomembnejši bajt je zapisan prvi) in *little endian* (UTF-32LE; najpomembnejši bajt je zapisan zadnji). V okolju Windows se običajno uporablja slednja.

### UCS-2

Kodiranje UCS-2 za vsako kodno točko porabi dva bajta. Zaradi tega lahko v njem zapišemo le znake iz osnovne ravni BMP. UCS-2 se dandanes ne uporablja več, v okolju Windows pa so na njem temeljile različice od NT do 4.0.

### UTF-16

»Domače« kodiranje operacijskih sistemov Windows od različice 2000 dalje za zapis ene kodne točke porabi dva ali štiri bajte. Če se omejimo na znake z osnovne ravni, je UTF-16 enak UCS-2, torej za vsako kodno točko porabi le dva bajta. S tem doseže bistveno boljšo izkoriščenost zapisa kakor UTF-32, še vedno pa porabi dvakrat več prostora od zapisa Ansi. Če se omejimo na raven BMP, je delo z zapisom UTF-16 prav tako enostavno kakor s kodiranjema Ansi in UTF-32, ker je zapis vsake kodne točke velik dva bajta. Znaki nad \$FFFF se zapišejo z *nadomestnimi pari* (surrogate pair). Iz kodne točke U+01D11E tako dobimo dve *kodni enoti* (code unit) U+D834, U+DD1E.

Računalničarji pa ne bi bili srečni, če ne bi komplikirali. Kodiranje UTF-16 obstaja v dveh različicah – *big endian* (UTF-16BE) in *little endian* (UTF-16LE). Razlikujeta se po tem, kako zapišemo vsako posamezno besedo (v programerskem smislu, torej vrednost od 0 do \$FFFF). UTF-16BE najprej zapiše višji (pomembnejši) bajt in nato nižji bajt, UTF-16LE pa najprej nižji in nato višji bajt. Slednji je privzeto uporabljen v Windows in zato zelo razširjen v računalništvu nasploh, čeprav v praksi srečamo tudi UTF-16BE.

Moje ime, »Primož«, ki ga sestavljaša kodne točke U+0050, U+0072, U+0069, U+006D, U+006F in U+017E, se v kodiranju UTF-16LE zapiše kot \$50 \$00 \$72 \$00 \$69 \$00 \$6D \$00 \$6F \$00 \$7E \$01, v kodiranju UTF-16BE pa kot \$00 \$50 \$00 \$72 \$00 \$69 \$00 \$6D \$00 \$6F \$01 \$7E.

## UTF-8

Najboljšo izkoriščenost prostora dosega kodiranje UTF-8, ki za vsako kodno točko porabi od 1 do 6 bajtov. Pri tem se znaki ASCII (torej s kodami od 0 do 127) kodirajo v en bajt, veliko posebnih evropskih znakov (tudi naši šumniki) v dva bajta, preostanek v tri bajte, ostanka (4 do 6 bajtov) pa pri kodiranju kodnih točk Unicode ne moremo izkoristiti, ker je uporabljen le za vrednosti nad \$1FFFFF.

Za primerjavo z UTF-16 – moje ime se v UTF-8 zapiše kot \$50 \$72 \$69 \$6D \$6F \$C5 \$BE.

Slabost zapisa UTF-8 je, da je popolnoma neprimeren za obdelavo nizov, ker so znaki spremenljive dolžine. Vseeno pa je strašno popularen v operacijskem sistemu Unix ter zaradi dobre izkoriščenosti prostora izjemno uporaben za prenos (komunikacije) ter shranjevanje podatkov (baze).

## BOM

Kodiranj je torej precej (še več kakor smo jih omenili zgoraj, a se druga redko uporabljajo) in dokler z nizi Unicode delamo samo znotraj programa ali skupine programov, ki so nastali znotraj enega podjetja, ne pride do težav. Kaj pa, ko od zunanjega sodelavca dobimo podatke v obliki Unicode, a ne vemo, v kakšnem kodiranju jih je shranil? Kako ugotovimo uporabljeni kodiranje?

Standard Unicode določa standardna zaporedja bajtov, imenovana BOM (byte order mark), ki jih program lahko (ni pa to obvezno!) zapiše na začetek podatkov (denimo na začetek datoteke). Pri branju nato le pogledamo začetek datoteke in uporabimo pravo kodiranje (ozioroma to namesto nas naredi kar Delphi).

V spodnji tabeli so prikazani standardni začetki najpogostejših kodiranj Unicode.

Kodiranje	BOM
UTF-32LE	\$FF \$FE \$00 \$00
UTF-32BE	\$00 \$00 \$FE \$FF
UTF-16LE	\$FE \$FF
UTF-16BE	\$FF \$FE
UTF-8	\$EF \$BB \$BF

Omenili smo, da programi lahko zapišejo BOM na začetek podatkov. Žal to ni obvezno in predvsem v svetu Unix programi neradi uporabljajo BOM. Še slabše – če je na začetku podatkov BOM, se zna zgoditi, da podatkov ne bodo znali prebrati. Pri sodelovanju z drugimi programi je zato treba biti pazljiv.

Če BOM na začetku manjka, nam ne ostane drugega, kot da analiziramo prvih nekaj kilabajtov podatkov. Namesto da bi tak analizator pisali sami, uporabite Charset Detector ([chsdet.sourceforge.net](http://chsdet.sourceforge.net)), ki je zapakiran v DLL z licenco LGPL in ga torej lahko uporabljate v komercialnih programih. Poleg kodiranj Unicode zna (ozioroma vsaj poskuša) določiti tudi kodno stran pri kodiranjih Ansi.

## Novosti v Delphi 2009

Precej prostora smo že posvetili standardu Unicode, nič pa še v uvodu omenjenim spremembam v Delphiju 2009. Omenili smo le, da programi, prevedeni z Delphijem 2009, uporabljajo vmesnik Unicode. Pravzaprav gre za tri tesno povezane spremembe.

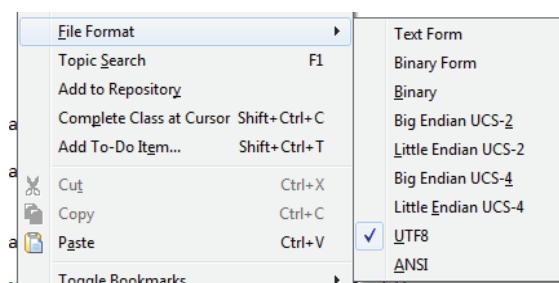
Prvič, vsa Windows okna programa, napisanega v Delphiju, so po novem narejena s klicem funkcije `CreateWindowsW` (prej `CreateWindowsA`). Zaradi tega Windows ve, da okna razumejo Unicode in temu ustrezno dela z njimi. Predvsem pričakuje, da so vsi tekstovni nizi, ki jih program pošlje oknu (torej recimo takrat, ko nastavi `TEdit.Text`) nizi Unicode v kodiranju UTF-16. Prav tako Windows враča vsebino oken (denimo takrat, ko program prebere `TEdit.Text`) v kodiranju UTF-16. [Spotoma lahko omenimo, da se je z dodatnimi komponentami (TNT Unicode Controls, avtor Troy Wolbrink) to dalo narediti že v Delphiju 2007 in starejših.]

Druga velika sprememba je, da VCL in RTL po novem kličeta »široke« različice sistemskih funkcij. Namesto (na primer) `GetModuleFileNameA` novi RTL pokliče `GetModuleFileNameW` in podobno. Tako lahko program dostopa do datotečnega sistema z Unicode imeni datotek, bere Unicode podatke iz registra in podobno. [Tudi to se je dalo narediti že v starejših Delphijih, vendar je bilo treba »na roke« naložiti ustrezne funkcije (`LoadLibrary`) in jih klicati v programu.]

Tretja velika sprememba je prizeti zapis nizov. V Delphiju 2007 in starejših je bil tip `string` definiran kot sopomenka za `AnsiString`, po potrebi pa smo lahko uporabljali tudi `WideString`, ki je shranjeval podatke Unicode v kodiranju UTF-16. Po novem je `string` sopomenka za novi tip `UnicodeString`, ki prav tako kot `WideString` vsebuje podatke Unicode v kodiranju UTF-16 (o razlikah med obema bomo več povedali v nadaljevanju), uporabljam pa lahko tudi `AnsiString`. Temu ustrezno se je spremenila definicija tipa `char`, ki je po novem sopomenka tipa `WideChar`, torej 16-bitnega znaka, ki je osnovni element nizov `WideString` in `UnicodeString`. Nova funkcija `StringElementSize` vrne velikost (v bajtih) enega znaka v nizu.

Vidimo torej, da Delphi 2009 pravzaprav ni prinesel ničesar, česar se z manjšim ali večim trudom ni dalo narediti že v prejšnjih Delphijih. Velika sprememba je bila v tem, da so spremembe obvezne in da v Delphijih od 2009 dalje preprosto ni več mogoče delati »po starem«. Sprememba pomeni tudi to, da ne moremo več izdelovati programov za Windows 95, 98 in ME. Če imate takšne zahteve, boste morali ostati na starejših Delphijih, ali pa programe predelati, tako da bodo dobro delovali v obeh svetovih – Ansi in Unicode (tudi o tem bomo še pisali).

Pravzaprav je Delphi 2009 prinesel še četrto veliko spremembo, a ta ne vpliva na delovanje programov, temveč le na obliko izvirne kode. Po novem so namreč izvirne datoteke (.pas) lahko zapisane v enem od kodiranj Unicode. Poleg tega lahko po novem uporabljamo nacionalne znake v imenih podprogramov in spremenljivk. Uporabimo lahko torej metodo, ki se imenuje `IzpišiPodatke`, spremenljivko `štavec` in podobno.



Spremembe se poznajo tudi v »notranjosti«, torej v programski kodi, ki sestavlja sam Delphi. Funkcije v RTL in VCL po novem uporabljajo Unicode. Večina pogosto uporabljenih funkcij (na primer `Pos`) je predelana, tako da sprejme različne vrste argumentov (`AnsiString` in `UnicodeString`), nekatere pa po novem delujejo le z nizi Unicode. Vrsta združljivostnih funkcij s parametri `AnsiString` je pospravljenih v enoto `AnsiStrings`, ki jo lahko uporabimo po potrebi.

## Nizi – takšni in drugačni

Oglejmo si bolj natančno, kakšne vrste nizov podpirajo Delphiji od 2009 dalje.

### `String[]`

Še vedno, a le iz zgodovinskih razlogov, so v Delphiju podprtji kratki nizi s sintakso `string[dolžina]` (na primer `var ime: string[40]`). Če jih že uporabljate, jih poznate, sicer pa se ne ubadajte z njimi. Kratki nizi so ostanek davne preteklosti.

### `AnsiString`

Tipi `AnsiString` je podoben tistemu iz Delphija 2007 in predhodnikov. Še vedno vsebuje do 2 GB Ansi znakov, a se je njegova interna struktura spremenila. (Interni struktura je natančneje opisana v dokumentu *Delphi and Unicode*. Povezavo do zadnjega najdete čisto na koncu tega dokumenta.)

`AnsiString` po novem s seboj nosi tudi številko kodne strani. Če uporabite le zapis `AnsiString`, bo vsebina označena s kodno stranjo Windows uporabniškega vmesnika (kodna stran, ki je nastavljena kot »jezik za programe, ki ne podpirajo Unicode«). Lahko pa definirate nov tip z eksplisitno navedeno kodno stranjo. Naredite si na primer poseben tip za kodno stran 1251: `type CyrillicString = type AnsiString(1251)`. Žal ste pri uporabi zelo omejeni, saj ne morete deklarirati spremenljivke na način `var s: AnsiString(1251)`; definirate lahko le nov tip in ga pozneje uporabite.

### `RawByteString`

Tip `RawByteString` je sopomenka za `AnsiString($FFFF)`. Pri tem \$FFFF ne pomeni kodne strani, temveč predstavlja posebno oznako, ki pomeni, da naj se Delphi ne vtika v podatke, shranjene v spremenljivkah tega tipa. Kadar uporabljamo `AnsiString`, Delphi namreč samodejno pretvarja podatke med različnimi kodnimi stranmi. Spodnji programček denimo v spremenljivko `b` spravi 'csz', ker se šumniki med preslikavo iz kodne strani 1250 v 1252 izgubijo.

```
type
  Ansi1250 = type AnsiString(1250);
  Ansi1252 = type AnsiString(1252);
var
  a: Ansi1250;
  b: Ansi1252;
  a := 'čšž';
  b := a;
```

### `UTF8String`

Kodiranje UTF-8 se v Delphiju 2009 in novejših obravnava kot posebna kodna stran tipa `AnsiString`. Tip je deklariran kot `UTF8String = type AnsiString(65001);`. Zaradi sistemski podpore in samodejnega pretvarjanja med kodnimi stranmi je delo s kodiranjem UTF-8 strašno enostavno – le

deklarirate spremenljivko ali parameter ustreznega tipa, ji priredite poljubne podatke in Delphi bo samodejno pretvarjal podatke iz Ansi/Unicode v UTF-8 in nazaj.

### UnicodeString

Novi tip `UnicodeString` (ozioroma njegova sopomenka `string`) vsebuje podatke Unicode v kodiraju UTF-16LE; v obliki, ki jo pričakuje tudi Windows API. Za razliko od starega tipa `WideString` je `UnicodeString` dodeljen z običajnim Delphijevim upravljalnikom pomnilnika (memory manager) in uporablja števec referenc (reference count). Števec referenc omogoči, da si več referenc (spremenljivk, parametrov), ki kažejo na isti niz, deli isti kos pomnilnika.

Če torej deklarirate dve spremenljivki tipa `string` in ju nastavite, tako da obe kažeta na isto vrednost `[var a, b: string; a := 'test'; b := a;]`, si bosta obe delili isti kos pomnilnika, kjer je zapisan niz. S tem se bistveno zmanjša število kopiranj podatkov v programu in pospeši delovanje. Če pozneje spremenite eno od teh spremenljivk [primer: `b := b + '1';`], se bo povezava med spremenljivkama prekinila. Med izvajanjem prireditvenega stavka, ki spremeni `b`, bo program ustvaril kopijo podatkov; `a` bo kazal na stare podatke, `b` pa na nove. Na enak način deluje tudi `AnsiString`.

Ker je kodiranje UTF-16, funkcija `Length` ne vsebuje nujno dolžine niza v znakih. Če imamo na primer niz `#$D834#$DD1E`, je v njem zapisan le en znak, U+01D11E, funkcija `Length` pa bo vrnila vrednost 2. Enota `SysUtils` vsebuje funkcijo `ElementToCharLen`, ki izračuna dolžino v znakih, če boste hoteli v takem primeru dostopati do posameznih znakov, pa boste uporabili funkcijo `NextCharIndex`.

### WideString

Tip `WideString` je poznal Delphi že davno pred različico 2009, vpeljan pa je bil zaradi podpore COM, ki tekstovne podatke prenaša v obliku Unicode. Ravno zaradi združljivosti s COM uporablja nizi `WideString` sistemski dodeljevalnik pomnilnika, ki je počasnejši od Delphijevega. Prav tako ne uporablja štetja referenc – če torej priredite eno spremenljivko tega tipa drugi `[var a, b: WideString; a := 'test'; b := a;]`, bo program vedno naredil kopijo podatkov. Zaradi tega je delo z nizi `WideString` počasnejše od dela z nizi `UnicodeString`. Še vedno pa `WideString` potrebujemo za združljivost s komponentami COM.

### UCS4String

Delphi pozna tudi tip za shranjevanje podatkov v obliku UTF-32LE, a z njim ne moremo kaj dosti početi. `UCS4String` je namreč le polje (array) štiribajtnih znakov, od sistemskih podpor pa najdemo le funkcije za pretvorbo (`UnicodeStringToUCS4String`, `UCS4StringToUnicodeString`, `ConvertFromUTF32`, `ConvertToUTF32`).

### Kodiranja

Na kratko sem že omenil, da zna Delphi samodejno razpozнатi kodiranje podatkov, če je le na začetku napisan BOM. Za to poskrbi razred `TEncoding`, ki je uporaben tudi za zapisovanje podatkov v različnih kodiranjih, beremo pa lahko tudi na »ročni« način, v katerem sami nastavimo želeno kodiranje.

Razredi, ki berejo in pišejo nize v tokove, so posodobljeni, tako da znajo uporabljati razred `TEncoding` za branje in pisanje. S podporo za različna kodiranja so opremljeni `TStrings`, `TStringStream`, `TStreamReader`, `TStreamWriter` in še nekaj manj zanimivih razredov.

To pomeni, da se za enostavnim `Memo.Lines.LoadFromFile` po novem skriva zapleten postopek. `LoadFromFile` najprej odpre datotečni tok, `TFileStream`, in kliče `LoadFromStream`. Ta se obrne na `TEncoding.GetBufferEncoding`, ki iz toka prebere prvih nekaj bajtov. Če se tok začne z `$EF $BB` `BF`, koda ustvari primerek razreda `TUTF8Encoding`. Če se tok začne z `$FE $FF`, ustvari `TUnicodeEncoding`, če se začne z `$FF $FE`, pa `TBigEndianUnicodeEncoding`. V skrajnem primeru, če na začetku toka ni nič od naštetege, koda ustvari primerek razreda `TMBCSEncoding`, ki skrbi za branje in pisanje podatkov `Ansi`. V slednjem primeru je `Ansi` vezan na trenutno kodno stran uporabniškega vmesnika, tako kot tip `AnsiString`, če mu ne določite kodne strani.

Vrsto kodiranja lahko določite tudi sami. Postopek je preprost – le pokličete ustrezno metodo razreda `TEncoding` in rezultat posredujete kot drugi parameter v klicu `LoadFromFile` (ali `LoadFromStream`, `SaveToFile` ali `SaveToStream`). Nekaj primerov:

```
inpMemo.Lines.LoadFromFile('tekst.txt', TEncoding.Ansi);
inpMemo.Lines.SaveToFile('tekst.txt', TEncoding.UTF8);
```

Vsa kodiranja, ki jih privzeto pozna `TEncoding`:

```
class property ANSI: TEncoding read GetANSI;
class property ASCII: TEncoding read GetASCII;
class property BigEndianUnicode: TEncoding read GetBigEndianUnicode;
class property Default: TEncoding read GetDefault;
class property Unicode: TEncoding read GetUnicode;
class property UTF7: TEncoding read GetUTF7;
class property UTF8: TEncoding read GetUTF8;
```

Zanimivo je kodiranje `Default`, ki na sistemih Windows vrne `TEncoding.Ansi`, sicer (na Mac OS X) pa `TEncoding.UTF8`.

`TEncoding` zna poljuben niz spremeniti v zaporedje bajtov (tip `TBytes`) in nazaj, kar zelo poenostavi pretvorbe med kodiranji, ki jih Delphi ne zna narediti samodejno. Če hočemo spremeniti niz iz »navadnega« UTF-16 v različico *big endian*, naredimo le:

```
var
  ime16be: string;
  ime16be := TEncoding.BigEndianUnicode.GetString(
    TEncoding.Unicode.GetBytes(FIMe));
```

Funkcija `GetBytes` spremeni niz v zaporedje bajtov, funkcija `GetString` pa naredi obratno.

## Character

Enota `Character` vsebuje številne funkcije, ki vrnejo lastnosti določenega znaka Unicode (`IsLower`, `IsSeparator`, `IsWhiteSpace` ...), funkcije za preverjanje nizov (`IsLower`, `IsNumber`, `IsLetter` ...), ter funkcije za spremicanje nizov (`ToLower`, `ToUpper`). Vsebuje tudi funkcije za spremicanje UTF-16 nizov v kodiranje UTF-32 in nazaj (`ConvertToUtf32`, `ConvertFromUtf32`).

## CharInSet

Funkcija `CharInSet` preveri, ali znak sodi v neko množico znakov. Se sliši zapleteno? Funkcija naredi isto kot vsem znani vzorec:

```
var  
  ch: char;  
  
  if ch in ['A'..'Z'] then
```

Po novem prevajalnik ob takšni kodi izpiše opozorilo.

`W1050 WideChar reduced to byte char in set expressions. Consider using 'CharInSet' function in 'SysUtils' unit.`

`Ch` je namreč 16-bitni znak, množice (notacija `['A'..'Z']`) pa imajo lahko največ 256 elementov. Zato prevajalnik spremeni `ch` v navaden znak Ansi, kar nas v zgornjem primeru sicer ne bo motilo, vseeno pa se ob tem izpiše opozorilo. Če se ga hočete znebiti, morate kodo spremeniti v:

```
if CharInSet(ch, ['A'..'Z']) then
```

Spotoma pa se lahko vprašate, ali je v aplikaciji Unicode ta koda še vedno pravilna. Če hočete pravilno podpirati Unicode, morate pravzaprav preveriti, ali je vrednost znaka `ch` katera koli Unicode črka.

```
if IsLetter(ch) then
```

Funkcija `IsLetter` je definirana v zgoraj omenjeni enoti `Character`.

## Baze

Pri podpori podatkovnim bazam se pravzaprav ni dosti spremenilo, saj so že Delphiji pred 2009 znali delati s podatki Unicode (`TField.AsWideString`). Je pa po novem Unicode podprt v vseh »data-aware« komponentah.

Bistveno je, da poskrbite, da povezava s strežnikom podpira znake Unicode (morda bo treba dodati »`ServerCharSet=UTF8`« v »connection string«) ter da uporabljate `TStringField` za dostop do podatkov Ansi in `TWideString` za dostop do podatkov Unicode. Ker ste to morali početi že prej, vam kode najverjetneje ne bo treba spremenjati.

Podrobnosti so odvisne od konkretnne podatkovne baze, se pa dostikrat zgodi, da se podatki Unicode v bazi shranijo v obliki UTF8. Pri tem se lahko dolžina niza spremeni (zaradi kodiranja, ki za en znak porabi od en do tri bajte), kar lahko pripelje do tega, da je kodirani niz daljši od omejitve stolpca, v katerega se shrani vrednost.

Druga sprememba v novejših Delphijih, ki ni vezana na Unicode, lahko pa vam zagreni prehod na novo različico, je tip `TBookmark`. Ta po novem ni več deklariran kot `pointer`, temveč kot polje bajtov, `TBytes`.

## Druge spremembe

V povezavi z novim, razširjenim tipom `AnsiString` sta uporabni funkciji `StringCodePage` (vrne kodno stran niza) in `SetCodePage` (nastavi kodno stran in pri tem po želji pretvori niz, tako da bo čim pravilneje predstavljen v novi kodni strani).

Razni deli RTL, ki niso povezani z novim razredom TEncoding, a vseeno delajo z nizi (na primer tekstovne datoteke »stare šole« – `textfile`) za pretvorbo v Unicode uporabljajo kodno stran uporabniškega vmesnika (pri opisu tipa `AnsiString` omenjeni »jezik za programe, ki ne podpirajo Unicode«). To obnašanje pa lahko spremenite, tako da nastavite sistemsko spremenljivko `DefaultSystemCodePage` na številko poljubne kodne strani.

## Prehod

Zdaj veste »vse« (no, vsaj vse pomembno) o Unicodu na splošno in o Unicodu v Delphiju 2009 in novejših, ničesar pa še nismo povedali o težavah, ki jih utegne imeti s prehodom s starega (pa naj bo to Delphi 7 ali Delphi 2007 ali kateri koli drugi) na novi Delphi.

Težavnost prehoda je nekje med *trivialno* in *ogromno dela, še misliti si ne morete, koliko*. Z nekaj sreče se bo program »kar prevedel« in ne boste imeli nobenega dela. Z nekaj manj sreče lahko za prehod porabite mesece. V praksi bo težavnost ležala nekje med temo ekstremoma, le od vaše kode pa je odvisno, koliko dela boste imeli.

Za začetek bi vas opozorili na knjižnice drugih avtorjev. Tudi če imate izvorno kodo zanje, se ne ukvarjajte s predelavo, temveč raje kupite novo različico (ozioroma jo prenesite z interneta, če gre za odprtokodne komponente). Če različice za sodoben Delphi ni (žal se to še prepogosto dogaja), vam priporočamo, da najdete zamenjavo. Le v skrajnem primeru se lotite predelave, saj so avtorji knjižnic in komponent vse prepogosto uporabljali trike, ki krepko zapletejo prehod na nov Delphi.

Uporabniki Turbo Power komponent in knjižnic se odpravite na SourceForge – njihove Delphi knjižnice so prešle v odprtakodne projekte in so vsaj deloma prilagojene novejšim Delphijem.

<http://sourceforge.net/directory?q=turbo%20power>

## Opozorila

Ko boste program prvič prevedli (ali poskusili prevesti, prav možno je, da bo prevajalnik javil kakšno napako), bodite pozorni na spodnja opozorila.

W1050 WideChar reduced to byte char in set expressions. Consider using 'CharInSet' function in 'SysUtils' unit.

W1058 Implicit string cast with potential data loss from 'string' to 'AnsiString'

W1057 Implicit string cast from 'AnsiString' to 'string'

W1059 Explicit string cast

W1060 Explicit string cast with potential data loss

W1062 Narrowing given wide/Unicode string constant lost information

W1063 Narrowing given WideChar constant to AnsiChar lost information

W1063 Widening given AnsiChar constant to WideChar lost information

W1064 Widening given AnsiString constant lost information

Opis teh napak in razlago njihovega nastanka najdete na [http://docwiki.embarcadero.com/RADStudio/en/Error\\_and\\_Warning\\_Messages\\_\(Delphi\)\\_Index](http://docwiki.embarcadero.com/RADStudio/en/Error_and_Warning_Messages_(Delphi)_Index), mi pa vas bomo opozorili na nekaj najpomembnejših.

Spodnji primer izpiše nekaj najpogostejših opozoril. Njihove oznake so podane v komentarjih pri posameznih vrsticah.

```

const
  c: AnsiString = 'čšžäëöü€ääáđ'; //W1058, W1062
var
  s: string;
  a: AnsiString;
  ch: char;
begin
  s := 'čšžäëöü€ääáđ';
  a := s; //W1058
  s := a; //W1057
  ch := s[1];
  if ch in ['č', 'š', 'ž'] then //W1050
    ShowMessage('!');
end;

```

Na W1062 boste redko naleteli. Če ga najdete, ga le odpravite, saj gre skoraj zagotovo za napako. To opozorilo vas namreč obvešča, da poskušate v niz stlačiti zname, ki ne gredo skupaj (jih ne najdete v isti kodni tabeli) in bo prišlo do izgube informacij.

Pozorni bodite na W1058, saj vas opozarja na možno izgubo informacij pri pretvarjanju niza Unicode v niz Ansi. Če ugotovite, da je koda pravilna, lahko opozorilo odpravite, tako da uvedete eksplisitno pretvorbo: `a := AnsiString(s);`.

W1057 je pretežno nenevaren. Pretvarjanje iz niza Ansi v niz Unicode sicer pomeni porabo časa (če to počnete v zanki, se utegne poznati tudi pri hitrosti izvajanja programa), a navadno ne povzroči izgube informacij. Če se hočete opozorila znebiti, ga lahko odpravite z eksplisitno pretvorbo: `s := string(a);`. Seveda pa lahko to opozorilo (tako kot tudi prejšnje) odpravite, tako da spremenite spremenljivko Ansi v Unicode ali nasprotno. Prava rešitev je odvisna od konkretnega primera.

O W1050 smo več napisali v razdelku »CharInSet«.

## Problemi

Naštejmo nekaj tipičnih programskeh pristopov, ki vam bodo zagrenili življenje pri prehodu na Unicode Delphi.

### Koda, ki predpostavlja, da je `char` velik en bajt.

Pri pretvarjanju obstoječe kode pogosto naletimo na kaj takega:

```

var
  buffer: array [0..255] of char;
  FillChar(Buffer, Length(Buffer), 0);

```

Ta košček kode predpostavlja, da funkcija `Length` (ta vrne število elementov v polju – 256 v našem primeru) poda velikost polja v bajtih. Če je `char` velik en bajt, je to sicer res, a kaj, ko je po novem `char` velik dva bajta.

Kodo lahko popravimo na nekaj različnih načinov. Najbolj pravilen je morda ta:

```

var
  buffer: array [0..255] of char;

```

```
FillChar(Buffer, Length(Buffer) * SizeOf(buffer[0]), 0);
```

Zdaj bo program deloval pravilno, tudi če kdaj v prihodnosti spremenimo tip podatkov, shranjenih v polju `buffer`.

### Nizi kot nosilec podatkov

Vse prepogosto smo programirali tako, da smo spremenljivke tipa `string` uporabljali za izmenjavo podatkov z zunanjim svetom, bodisi z uporabo datotek, tokov, protokola TCP/IP ali kako drugače. Tipičen primer je spodnja koda:

```
function ReadFromStream(s: TStream): string;
var
  len: integer;
begin
  s.Read(len, 4);
  SetLength(Result, len);
  s.Read(Result[1], len);
end;
```

Tukaj je problem podoben kakor v prejšnjem primeru. `SetLength` bo nastavil dolžino niza na `len` znakov, kar je po novem `len * 2` bajtov, nato pa bo prebral le polovico tega niza. Poleg tega bo (najverjetneje) prebral podatke Ansi in jih zapakiral v niz Unicode. Rezultat bo približno tak: » 封 封 ». Nauk: kadar se v programu besedilo nenačoma spremeni v kitajščino, najverjetneje niz Ansi interpretirate, kot da vsebuje podatke Unicode.

Očitna rešitev – spremeniti tip rezultata funkcije v `AnsiString` – ni najboljša, saj pri nadalnjem pritejanju še vedno lahko pride do predelav zaradi spremembe kodne strani in s tem do spremembe podatkov. Bolje le, da tip deklarirate kot `RawByteString`, potem se Delphi v njegovo vsebino ne bo vtikal.

```
function ReadFromStream(s: TStream): RawByteString;
var
  len: integer;
begin
  s.Read(len, 4);
  SetLength(Result, len);
  s.Read(Result[1], len);
end;
```

### Pretvorba nizov

Pri kombiniranju nizov Ansi in Unicode pride do pretvorbe vsebine, zaradi česar lahko pride do izgube podatkov. Bodite pozorni na opozorila, ki smo jih omenjali zgoraj v razdelku »Opozorila«.

### Packanje s kazalci

Tudi programske kode, ki uporablja `PChar` za neposredno packanje po vsebini nizov, je vse preveč. Spodnji, izmišljeni in rahlo prisiljeni zgled na ta način prešteje število presledkov v nizu.

```
function CountSpaces(const s: string): integer;
var
  i : integer;
  pc: PChar;
```

```

begin
  Result := 0;
  if s <> '' then begin
    pc := @(s[1]);
    for i := 1 to Length(s) do begin
      if pc^ = ' ' then Inc(Result);
      pc := pointer(integer(pc)+1);
    end;
  end;
end;

```

Programer je privzel, da so znaki v nizu veliki en bajt in to v svetu Unicode seveda ne drži. Preprost popravek bi bil spremeniti zadnje pritejanje v `pc := pointer(integer(pc)+StringElementSize(s))`; a da se tudi bistveno enostavneje: `Inc(pc)`. Spremenljivke tipa PChar so po novem takšne, da lahko na njih uporabljamo operaciji `Inc` in `Dec`, ki delujeta tako, kot bi pričakovali – povečata ali zmanjšata kazalec za 2. [Podobno velja za PAnsiChar.]

Možen problem zgornje kode je tudi, da ne upošteva dejstva, da je v Unicodu več različnih vrednosti, ki vse pomenijo *prazen prostor*. Morda bo treba spremeniti tudi to. Morda pa ne, odvisno od ostanka programa.

```

function CountSpaces(const s: string): integer;
var
  i : integer;
  pc: PChar;
begin
  Result := 0;
  if s <> '' then begin
    pc := @(s[1]);
    for i := 1 to Length(s) do begin
      if IsWhiteSpace(pc^) then Inc(Result);
      Inc(pc);
    end;
  end;
end;

```

## Iskanje potencialno problematičnih mest

Naslednje funkcije so pogosto povezane z napačno rabo nizov ali kazalcev nanje in se zato splača preveriti vsa mesta, kjer se pojavljajo: `Move`, `FillChar`, `NewStr/DisposeStr`, `AllocMem`, `GetMem`, `StrAlloc`, `Read`, `ReadBuffer`, `Write`, `WriteBuffer`. Preverite tudi mesta, kjer se pojavljata `PChar` ali `PString`.

Eksplisitni klici funkcij za pretvorbo nizov Ansi v Unicode in nazaj morda ne bodo več potrebni. Poiščite mesta, kjer se uporablja `MultiByteToWideChar` in `WideCharToMultiByte` in preverite njuno rabo.

Pozorni bodite na `LoadFromFile`, `LoadFromStream`, `SaveToFile` in `SaveToStream`, ker se je njihovo delovanje spremeno (več o tem v razdelku »Kodiranja« v poglavju »Novosti v Delphi 2009«).

Pozorni bodite na rabo funkcij `Uppercase`, `Lowercase`, `CompareStr`, `CompareText`, `SameStr` in podobnih. Skoraj vedno boste namesto njih želeli uporabiti različico, ki se začne z »Ansi« – `AnsiUppercase`, `AnsiLowercase`, `AnsiCompareStr`, `AnsiCompareText` itd, čeprav delate z nizi

Unicode. Različice brez »Ansi« pri znakih s kodami nad 127 dajejo nepričakovane rezultate. `Uppercase('aâäá')` na primer vrne 'Aâäá', `AnsiUppercase('aâäá')` pa 'AAÄÄ'.

Težave utegneče imeti tudi pri obdelavi parametrov tipa `array of const`. Ti po novem lahko vsebujejo tudi tip `vtUnicodeString` in polje `VUnicodeString`, ki nosi podatke. Skrajšan primer iz obstoječe kode predstavlja vse možne načine, na katere je v takem parametru lahko shranjen niz.

```
case VType of
  vtChar:           ovc.Add(string(VChar));
  vtString:         ovc.Add(string(VString^));
  vtPChar:          ovc.Add(string(StrPas(VPChar)));
  vtAnsiString:    ovc.Add(string(VAnsiString));
  vtWideString:    ovc.Add(WideString(VWideString));
  vtUnicodeString: ovc.Add(string(VUnicodeString)); // novo
end;
```

## Nasveti

Včasih je enostavna pot do delajočega programa le ena – vzamete kodo in zamenjate vse `string` v `AnsiString`, `PString` v `PAnsiString`, `char` v `AnsiChar` in `PChar` v `PAnsiChar`. Da se ne boste preveč mučili, je Roger Connel napisal programček, ki bo to naredil namesto vas. Dobite ga na spletni strani <http://www.innovasolutions.com.au/delphistuf/ADUGStringToAnsiStringConv.htm>.

Ansi različice funkcij, ki po novem delajo le z nizi Unicode, so pospravljene v enoto `AnsiStrings`.

Če v kodi uporabljate `TWideStrings` ali `TWideStringList` iz enote `WideStrings` ju morda lahko zamenjate v običajna `TStrings` in `TStringList`.

Če podatki na zaslonu niso prikazani pravilno, morda uporabljate pisavo, ki ne podpira znakov Unicode.

API funkcija `CreateProcess` je problematična, ker različica Unicode dela drugače kot različica Ansi. Različica Unicode lahko spreminja vrednost parametra `lpCommandLine`, kar lahko pokvari podatke v vašem programu. Iz istega razloga vrednost parametra ne sme biti konstanta, temveč mora biti shranjena v spremenljivki. Pred klicem `CreateProcessW` zato vedno prepišite ukazno vrstico programa v lokalno spremenljivko tipa `WideString` (ne `string` ali `UnicodeString`, ker ta uporablja štetje referenc in ne bosta naredila prave kopije) in nato to spremenljivko pošljite v `CreateProcessW`.

Če imate obstoječo funkcijo (denimo v zunanji dinamični knjižnici), ki sprejme parameter tipa `PAnsiChar`, je najbolj varen način pošiljanja takšnega parametra dvojna eksplisitna pretvorba: `PAnsiChar(Ansistring(s))`.

## Popolna podpora Unicoda

Pravilna in popolna podpora Unicoda v programu je zapleteno opravilo. Delno nam življenje grenita kompozicija (glej razdelek »Kompozicija«) in nadomestni pari (glej razdelek »UTF-16«), dodatne težave pa prinesejo jeziki, ki se pišejo z desne proti levi (arabščina), znotraj katerih se lahko pojavijo deli, ki se pišejo z leve proti desni (imena v latinici, indoevropska števila). Obravnavanje takšnih primerov žal presega obseg in cilj tega besedila.

Delna podpora Unicoda, s katero želimo pokriti le zahodnoevropske pisave, centralnoevropske pisave in cirilico, pa je preprosta in ne zahteva posebnih sprememb v programu.

## Združljivost

Hiter prehod na novo različico Delphija se pogosto izkaže za nemogoče opravilo. Dela je enostavno preveč, da bi ga opravili v kratkem času, med prilagajanjem programov pa stranke že zahtevajo nove različice in popravke. Zato je najbolje, če se dela lotimo tako, da bodo programi hkrati delovali v starem in novem Delphiju. Res je, da bomo s tem imeli več dela, a ga lahko opravimo postopoma, med tem pa opravljam običajen razvoj in vzdrževanje programa (ali programov). Tudi če predelujete le en program, je tak pristop priporočljiv, saj bo vzdrževanje različice Ansi možno ves čas predelave v Unicode. Da pa bo takšen »mehak« prehod sploh možen, je dobro upoštevati nekaj nasvetov.

Najprej poiščite interakcijske točke med programi ali med deli programa. To so lahko datoteke, baze, včasih pa tudi pomnilniške strukture – cevovodi, skupni pomnilnik, podatki, ki tečejo po protokolu TCP/IP, podatki na zaporednem vmesniku ... Vse takšne dele enostavno »popravite«, tako da spremenite `string` v `AnsiString` in podobno (več v razdelku »Nasveti«). Te dele programa boste lahko predelali (če bo to sploh potrebno) šele, ko bodo vsi sodelujoči deli prestavljeni v Unicode.

Če gre za komunikacijo (TCP/IP, cevovodi ...) je včasih smiselno predelati protokol, tako da bo omogočil sočasni prenos podatkov Ansi in Unicode. Pošiljatelj bo potem poslal vse različice podatkov, kar jih zna poslati, prejemnik pa bo interpretiral tisto, ki jo zna obdelati in prikazati.

Kodo, ki jo delite med programi (ozioroma jo prevajate za obe tarči – Ansi in Unicode) bo navadno treba popraviti. Vrsta popravka je zelo različna od potreb. Včasih bo treba le parametre tipa `string` spremeniti v `AnsiString`. Spet drugič bomo želeli, da funkcija deluje različno v obeh primerih – takrat bomo bodisi napisali kodo, ki deluje v obeh svetovih (nekaj primerov je v razdelku »Problemi«), bodisi bomo uporabili pogojno prevajanje.

```
{$IFDEF Unicode}
// ta del se prevede le v Delphiju 2009 in novejših
{$ELSE}
// ta del se prevede le v Delphiju 2007 in starejših
{$ENDIF}
```

Veliko primerov eksplizitnega navajanja pretvorb (`string` v `AnsiString` in obratno) lahko odpravimo, tako da navedemo dve različici funkcije. V Ansi Delphijih potrebujemo le eno, v Unicode Delphijih pa obe.

```
function FromHex (x: string): Int64; {$IFDEF Unicode}overload;
function FromHex (x: AnsiString): Int64; overload;{$ENDIF Unicode}
```

Implementacija nove funkcije je lahko napisana na novo, lahko pa nova funkcija kar kliče staro.

```
function FromHex (x: AnsiString): Int64;
begin
  Result := FromHex(string(x));
end;
```

Na tak način lahko deklarirate tudi dvojnice funkcije iz RTL ali VCL. Delphi denimo pozna le različico funkcije `FindFirst`, ki sprejme niz Unicode.

```
function FindFirst(const Path: string; Attr: Integer;
  var F: TSearchRec): Integer;
```

Za lažjo rabo obstoječe kode, si naredite enoto in jo uporabite le v Unicode Delphiju (z uporabo ukaza `{$IFDEF Unicode}`). V njej definirate dve različici te funkcije.

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
overload;

function FindFirst(const Path: AnsiString; Attr: Integer;
                   var F: TSearchRec): Integer; overload;
```

Izvedba je preprosta – le pokličite sistemsko funkcijo.

```
function FindFirst(const Path: string; Attr: Integer; var F: TSearchRec): Integer;
begin
  Result := SysUtils.FindFirst(Path, Attr, F);
end;

function FindFirst(const Path: AnsiString; Attr: Integer;
                   var F: TSearchRec): Integer;
begin
  Result := SysUtils.FindFirst(string(Path), Attr, F);
end;
```

Za prehod bo verjetno potrebnega veliko dela, a nekoč ga bo treba narediti.

## Koristne povezave

Za nadaljnje izobraževanje na temo Unicoda v Delphiju priporočamo tri tehnične dokumente, ki jih je objavil Embarcadero:

- Marco Cantù, *Delphi and Unicode*,  
<http://www.embarcadero.com/images/dm/technical-papers/delphi-and-unicode-marco-cantu-2.pdf>
- Nick Hodges, *Delphi in a Unicode World*,  
<http://www.embarcadero.com/images/dm/technical-papers/delphi-in-a-unicode-world-updated-2.pdf>
- Cary Jensen, *Delphi Unicode Migration for Mere Mortals: Stories and Advice from the Front Lines*.  
<http://www.embarcadero.com/images/dm/technical-papers/delphi-unicode-migration.pdf>

Dokumentacija za Delphi XE2 je dostopna na naslovu <http://docwiki.embarcadero.com/VCL/en>.

