

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Delphi 6.

Nowe narzędzia obliczeniowe

Autor: [Andrzej Daniluk](#)

ISBN: 83-7197-766-2

Format: B5, stron: 189

Zawiera CD-ROM



Delphi 6 jest kolejną wersją najpopularniejszego zintegrowanego środowiska programowania typu RAD dla platformy Windows. Dodatkowo współpracując z Borland Kylixem – pierwszym środowiskiem programistycznym RAD dla Linuksa – powoduje, iż obszary wykorzystania nowego Delphi przez osoby znające język Object Pascal znacznie się rozszerzyły.

Niniejsza książka jest tak pomyślana, aby pokazać Czytelnikowi możliwości nowej wersji Delphi. Poza zilustrowaniem, jak można sprawnie używać zasobów kompilatora zawartych w modułach StdConv, ConvUtils, VarConv, Math, VarCmplx wszędzie – gdzie było to możliwe – autor starał się przedstawić pożyteczne przykłady i algorytmy ilustrujące praktyczne aspekty wykorzystania opisanych elementów środowiska Delphi 6.

Omawiane w tej książce typy danych, stałe, zmienne, funkcje i procedury nie są częścią standardowego języka Object Pascal. Zostały włączone do środowiska programowania w celu uczynienia go jeszcze bardziej przyjaznym użytkownikowi, powodując jednocześnie, iż nowe Delphi wykonało kolejny krok w przybliżeniu swojej funkcjonalności do takich narzędzi obliczeniowych jakimi są Excel, C++ Builder czy Matlab.

Pod względem tematycznym książka została podzielona na trzy główne działy.

- Wielkości fizyczne

Procedury przeliczania wielkości fizycznych. Moduły StdConv, ConvUtils oraz VarConv, opisują możliwości nowego Delphi w zakresie posługiwania się wielkościami fizycznymi oraz manipulowania ich jednostkami. Przedstawiono dostępne z poziomu kompilatora predefiniowane układy jednostek, funkcje przeliczające wybrane wielkości fizyczne oraz sposoby tworzenia zarówno własnych układów jednostek, jak i metody definiowania samodzielnie skonstruowanych funkcji przeliczających. Większość z prezentowanych zasobów Delphi 6 jest częścią standardowej biblioteki VCL, niektóre z nich, oparte na typach wariantowych, należą już do biblioteki CLX – mogą być więc z powodzeniem użyte w aplikacjach międzyplatformowych.

- Moduł Math

Rozdział zawiera opis wyższego poziomu procedur oraz funkcji arytmetycznych, trygonometrycznych, hiperbolicznych, cyklometrycznych, logarytmicznych, statystycznych, funkcji generatora liczb pseudolosowych, funkcji służących do przeprowadzania różnego rodzaju obliczeń finansowych oraz funkcji FPU. Przedstawione funkcje i procedury należy traktować jako uzupełnienie zasobów standardowego języka Object Pascal znajdujących się w module System. W większości stanowią część biblioteki CLX z powodzeniem mogą być używane podczas projektowania aplikacji międzyplatformowych.

- Moduł VarCmplx

Opis zastosowań coraz popularniejszych typów wariantowych na potrzeby działań.



Spis treści

Wstęp	5
Rozdział 1. Wielkości fizyczne. Procedury przeliczania wielkości fizycznych.	
Moduły StdConvs, ConvUtils oraz VarConv	7
Predefiniowane układy jednostek oraz funkcje przeliczające	11
Samodzielne definiowanie układów jednostek	54
Moduł VarConv.....	62
Podsumowanie	67
Rozdział 2. Moduł Math	69
Funkcje miary kąta	69
Funkcje trygonometryczne zmiennej rzeczywistej	73
Odwrotne funkcje trygonometryczne	78
Funkcje hiperboliczne	85
Odwrotne funkcje hiperboliczne	87
Funkcje wykładnicze.....	92
Funkcje potęgowe	93
Funkcje logarytmiczne	96
Funkcje, procedury oraz stałe arytmetyczne i konwersji typów	98
Funkcje oraz procedury FPU.....	120
Funkcje generatora liczb pseudolosowych.....	124
Funkcje służące do wykonywania obliczeń statystycznych.....	133
Funkcje finansowe.....	145
Podsumowanie	154
Rozdział 3. Moduł VarCmplx	155
Liczby zespolone, płaszczyzna liczbowa, moduł i argument liczby	155
Podstawowe funkcje zmiennej zespolonej.....	157
Funkcje trygonometryczne zmiennej zespolonej	172
Odwrotne funkcje trygonometryczne zmiennej zespolonej	175
Funkcje hiperboliczne zmiennej zespolonej	179
Odwrotne funkcje hiperboliczne zmiennej zespolonej	183
Podsumowanie	186
Literatura uzupełniająca	187

Rozdział 1.

Wielkości fizyczne. Procedury przeliczania wielkości fizycznych. Moduły StdConvs, ConvUtils oraz VarConv

Wielkością fizyczną (wielkością mierzalną) nazywamy każdą mierzalną cechę zjawiska lub ciała, którą można porównać ilościowo z takimi samymi cechami innych zjawisk lub ciał. Aby móc w sposób właściwy posługiwać się wybraną wielkością fizyczną, (i nie tylko) należy wielkość taką w odpowiedni sposób zdefiniować, tzn. wyrazić wielkość nieznaną za pomocą wielkości wcześniej określonych, tzn. wielkości podstawowych. *Wielkościami podstawowymi* nazywamy także wielkości, którymi łatwo posługujemy się w życiu codziennym i przez to są one dla nas intuicyjnie jasne. Wszystkie inne zdefiniowane na podstawie wielkości podstawowej będą *wielkościami pochodnymi*. Aby uzyskać ilościową informację o jakiejś wielkości należy porównać ją z wielkością tego samego rodzaju przyjętą za jednostkę.

Wartość liczbowa otrzymana w wyniku takiego porównania zawsze będzie od wyboru jednostki podstawowej. Wszystkie wyniki pomiarów różnych wielkości fizycznych zawsze podawane są w ogólnie akceptowanych jednostkach. Wszystkie jednostki, w których wyrażamy daną wielkość, muszą wywodzić się (być określoną wielokrotnością lub podwielokrotnością) z pewnej podstawowej jednostki danego układu. Jeżeli np. zechcemy zdefiniować wielkość zwaną prędkością, wówczas jako wielkości podstawowe przyjmujemy długość oraz czas, ogólnie przyjętymi jednostkami tych wielkości będą metr oraz sekunda, zatem

$$\text{jednostka prędkości} = \frac{\text{jednostka długości}}{\text{jednostka czasu}},$$

zaś jednostką pochodną prędkości będzie

$$\text{jednostka przyspieszenia} = \frac{\text{jednostka długości}}{(\text{jednostka czasu})^2},$$

Powiemy ogólnie: jeżeli jakąś wybraną wielkość w określimy za pomocą innych wielkości w_1, \dots, w_n , wówczas w będzie pewną funkcją w_i :

$$w = f(w_1, \dots, w_n).$$

Funkcja f jednoznacznie określa wymiar wielkości w , zatem zgodnie z powyższymi zapisami wymiarem prędkości będzie m/s , zaś przyspieszenia m/s^2 . Zbiór wszystkich jednostek podstawowych oraz określonych za ich pomocą jednostek pochodnych tworzy pewien zbiór (układ) jednostek. Na potrzeby dalszych rozważań należy rozróżnić dwa pojęcia: *czynnik przeliczeniowy* oraz *mnożnik*.

Mnożnikiem jest konkretna liczba służąca do opisanie wartości pewnej wielkości w obrębie tej samej jednostki. Podstawową jednostką długości jest 1 metr (1 m). Wartość 100 m otrzymamy wykonując prostą operację mnożenia:

$$100 \text{ m} = 100 * 1 \text{ m}.$$

Liczba 100 jest przykładem niemianowanego mnożnika.

W odróżnieniu od mnożników *czynniki przeliczeniowe* mogą być wielkościami mianowanymi lub niemianowanymi. Jeżeli zechcielibyśmy odległość 1 m wyrazić w centymetrach, łatwo możemy dokonać odpowiedniego przypisania:

$$1 \text{ m} = 100 \text{ cm},$$

dzieląc z kolei obie strony powyższej równości przez 100 cm:

$$\frac{1 \text{ m}}{100 \text{ cm}} = 1,$$

po lewej stronie równości otrzymamy postać czynnika przeliczeniowego przeliczającego wielkości wyrażane w metrach na inne wielkości tego samego układu jednostek wyrażane w centymetrach, np.:

$$1000 \text{ cm} * \frac{1 \text{ m}}{100 \text{ cm}} = 10 \text{ m},$$

wynika stąd, iż:

$$1000 \text{ cm} = 10 \text{ m}.$$

Czynniki przeliczeniowe mogą być też wielkościami niemianowanymi. Równie dobrze przykład, w którym przeliczaliśmy metry na centymetry, można zapisać następująco:

$$10 \text{ m} * \frac{1}{100} = 0,1 \text{ m} = 10 \text{ cm} \Rightarrow 10 \text{ m} = 1000 \text{ cm},$$

gdzie wykorzystano fakt, iż 1 metr liczy 100 centymetrów. Zatem w tym konkretnym przypadku wartość czynnika przeliczeniowego równa będzie $1/100$.

W przeszłości w użyciu było wiele mniej lub bardziej równouprawnionych układów jednostek, takich jak: CGS (centymetr-gram-sekunda), elektromagnetyczny CGS, elektrostacyjny CGS, MKS (metr-kilogram-sekunda) czy brytyjski techniczny układ jednostek *fps* (stopa-funt-sekunda)¹.

Reguły definiujące poszczególne wielkości miały różną postać, różne też były stosowane w obrębie danego układu czynniki przeliczeniowe pomocne w przeliczaniu jednostki podstawowej na jednostki pochodne.

W 1960 roku podczas XI Generalnej Konferencji Miar podjęto próbę zunifikowania jednostek miar, w ten sposób powstał jednolity międzynarodowy układ jednostek miar SI (International System of Units). Nie będziemy tutaj szczegółowo omawiać układu SI, gdyż każdy z nas musiał się z nim spotkać na pierwszej lekcji fizyki w szkole. Ważniejszą rzeczą jest zaprezentowanie mnożników stosowanych w jego obrębie.

Sprawa jest o tyle ciekawa, iż występują pewne różnice w nazewnictwie używanym w USA oraz Europie. Cały problem wynika z nieco innego stosowania na obu kontynentach systemu dziesiętnego w odniesieniu do liczb bardzo małych i bardzo dużych. W USA podstawą liczenia jest system, w którym liczby grupuje się po trzy, np. bilion to tysiąc do potęgi trzeciej, zaś trylion to tysiąc do potęgi czwartej, itd. W Europie pozostano przy starszej wersji tego systemu, mianowicie liczby grupuje się następująco: milion to tysiąc tysięcy, miliard to tysiąc tysięcy tysięcy, itd. Innymi słowy, nasz bilion to amerykański trylion, zaś nasz miliard to amerykański bilion (słowo miliard w USA nie jest używane).

Jeżeli kiedyś będziemy chcieli tworzyć aplikacje o „zasięgu międzynarodowym”, warto zdawać sobie sprawę z obowiązujących reguł nazewnictwa. Tabela 1.1 prezentuje obowiązujące obecnie reguły stosowane w nazewnictwie liczb.

Przedstawione reguły nazewnictwa oraz odpowiednie mnożniki mogą być pomocne nie tylko w poprawnym konstruowaniu aplikacji wykorzystywanych w różnych dziedzinach nauki czy techniki. Trzeba pamiętać, iż nowe Delphi 6 udostępnia programistom również szereg „technologii biznesowych”, takich jak BizSnap (wspierającą integrację działań B2B — Business-to-Business — poprzez tworzenie połączeń XML/SOAP Web Services), WebSnap oraz DataSnap, które pomogą użytkownikom tworzyć internetowe aplikacje typu Web Services, zarówno po stronie serwera jak i klienta.

Tworząc tego typu aplikacje należy zawsze pamiętać o poprawnym stosowaniu nazewnictwa oraz właściwym doborze zarówno mnożników, jak i czynników przeliczających dane wielkości.

Ze względu na to, iż przedstawione w tabeli 1.1 mnożniki dla układu SI bazują na kolejnych całkowitych potęgach liczby 10, nie mogą one być używane do dokładnego reprezentowania dwójkowego rozwinięcia liczb, tzn. liczb w postaci binarnej. Ma to szczególne znaczenie w zagadnieniach związanych z transmisją danych oraz opisami różnych protokołów komunikacyjnych.

¹ W układzie *fps* *funt* jest jednostką siły.

Tabela 1.1. Obowiązujące reguły nazewnictwa wybranych wartości będących niemianowanymi mnożnikami (określających podwielokrotności oraz wielokrotności) dla jednostek układu SI

Nazwa w USA	Nazwa w Europie	Wartość mnożnika	Numeryczna postać wykładnicza mnożnika	Przedrostek określający wielokrotności i podwielokrotności metrycznego układu jednostek	Symbol przedrostka
septylionowa	kwadrylionowa	$(10^3)^{-8}$	1.0E-24	yocto	y
sekstylionowa	tryliardowa	$(10^3)^{-7}$	1.0E-21	zepto	z
kwintyljonowa	trylionowa	$(10^3)^{-6}$	1.0E-18	atto	a
kwadrylionowa	biliardowa	$(10^3)^{-5}$	1.0E-15	femto	f
trylionowa	bilionowa	$(10^3)^{-4}$	1.0E-12	piko	p
bilionowa	miliardowa	$(10^3)^{-3}$	1.0E-9	nano	n
	milionowa	$(10^3)^{-2}$	1.0E-6	mikro	μ
tysięczna	tysięczna	$(10^3)^{-1}$	1.0E-3	mili	m
setna	setna	10^{-2}	1.0E-2	centy	c
dziesiąta	dziesiąta	10^{-1}	1.0E-1	decy	d
jeden	jeden	10^0	—	—	—
dziesięć	dziesięć	10^1	1.0E+1	deka	da
sto	sto	10^2	1.0E+2	hekto	h
tysiąc	tysiąc	$(10^3)^1$	1.0E+3	kilo	k
	milion	$(10^3)^2$	1.0E+6	mega	M
bilion	miliard	$(10^3)^3$	1.0E+9	giga	G
trylion	bilion	$(10^3)^4$	1.0E+12	tera	T
kwadrylion	biliard	$(10^3)^5$	1.0E+15	peta	P
kwintylion	trylion	$(10^3)^6$	1.0E+18	exa	E
sekstylion	tryliard	$(10^3)^7$	1.0E+21	zetta	Z
septylion	kwadrylion	$(10^3)^8$	1.0E+24	yotta	Y

W celu uniknięcia tych niedogodności w 1998 roku Międzynarodowa Komisja Elektrotechniczna (International Electrotechnical Commission — IEC) jako zalecany przyjęła odrębny standard nazewnictwa symboli i przedrostków dla mnożników wielkości reprezentujących liczby zapisane w systemie dwójkowym.

Tabele 1.2 oraz 1.3 przedstawiają odpowiednio system nazewnictwa, przedrostki, mnożniki wielkości reprezentujących liczby zapisane w systemie dwójkowym oraz przykłady ich porównania z wybranymi jednostkami wywodzącymi się z układu SI.

Tabela 1.2. System nazewnictwa liczb (binarnych) reprezentowanych w systemie dwójkowym a nazewnictwo stosowane w układzie SI

Przedrostek	Wartość mnożnika	Symbol	Przedrostek stosowany w układzie SI	Wartość mnożnika w układzie SI
kibi	$(2^{10})^1$	Ki	kilo	$(10^3)^1$
mebi	$(2^{10})^2$	Mi	mega	$(10^3)^2$
gibi	$(2^{10})^3$	Gi	giga	$(10^3)^3$
tebi	$(2^{10})^4$	Ti	tera	$(10^3)^4$
pebi	$(2^{10})^5$	Pi	peta	$(10^3)^5$
exbi	$(2^{10})^6$	Ei	exa	$(10^3)^6$

Tabela 1.3. Porównanie nazewnictwa wielkości reprezentowanych w systemie dwójkowym i układzie dziesiętnym oraz wartości ich mnożników. W nawiasach podano nazwy angielskie

Pełna nazwa (nazwa angielska)	Skrót nazwy oraz mnożnik
1 kibibit (kibibit)	1 kibiB = 2^{10} bitów = 1 024 bitów
1 kilobit (kilobit)	1 kbit = 10^3 bitów = 1 000 bitów
1 mebibajt (mebibyte)	1 MiB = 2^{20} bajtów = 1 048 576 B
1 megabajt (megabyte)	1 MB = 10^6 bajtów = 1 000 000 B
1 gibibajt (gibibyte)	1 GiB = 2^{30} bajtów = 1 073 741 824 B
1 gigabajt (gigabyte)	1 GB = 10^9 bajtów = 1 000 000 000 B
1 tebibajt (tebibyte)	1 TiB = 2^{40} bajtów = 1 099 511 627 776 B
1 terabajt (terabyte)	1 TB = 10^{12} bajtów = 1 000 000 000 000 B

Z powyższych zestawień możemy odczytać, iż dość powszechne utożsamianie np. wielkości reprezentującej 1 kilobit (1 kbit) z wartością 1 024 bitów jest sporym błędem, niestety tego typu nieścisłości nader często występują nie tylko w rodzimej literaturze. Niekonsekwentne stosowanie określonych mnożników dla wielkości reprezentowanych w zapisie binarnym jest częstym powodem nie w pełni poprawnego działania algorytmów obsługujących protokoły transmisji danych. Błędy takie są nieraz bardzo trudne do przechwycenia i zdiagnozowania.

Predefiniowane układy jednostek oraz funkcje przeliczające

Niniejszy podrozdział zawiera kompletny opis predefiniowanych jednostek oraz układów jednostek, typów, klas, funkcji oraz procedur przeliczających wielkości fizyczne. Omówione elementy nie są wbudowane w kompilator Delphi 6, są natomiast zdefiniowane w modułach *StdConvs* oraz *ConvUtils* będących częścią standardowej biblioteki VCL.

Należy zwrócić uwagę na to, iż nie wszystkie przedstawione poniżej jednostki należą do układu SI, część z nich właściwa jest układowi fps, jeszcze inne są już tylko sporadycznie używane w krajach anglosaskich. Dla wygody wszystkie prezentowane jednostki zostały pogrupowane w zbiory, za pomocą których można wyrazić i przeliczyć bardzo wiele interesujących nas wielkości.

cbArea — predefiniowana zmienna

Składnia

```
var cbArea: TConvFamily = 2;
```

Opis

cbArea tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek pola powierzchni. Zbiór aktualnie dostępnych jednostek przedstawia tabela 1.4.

Tabela 1.4. Predefiniowane jednostki należące do zbioru cbArea reprezentującego układ jednostek pola powierzchni (przedrostek au jest skrótem od area units). Jednostką podstawową w tym zbiorze jest jeden metr kwadratowy ($1 m^2$)

Literał	Liczbowy identyfikator	Opis
auSquareMillimeters	32	Milimetr kwadratowy
auSquareCentimeters	33	Centymetr kwadratowy
auSquareDecimeters	34	Decymetr kwadratowy, czyli dziesięć centymetrów kwadratowych
auSquareMeters	35	Metr kwadratowy
auSquareDecameters	36	Dekametr kwadratowy, czyli dziesięć metrów kwadratowych
auSquareHectometers	37	Hektometr kwadratowy, czyli sto metrów kwadratowych
auSquareKilometers	38	Kilometr kwadratowy
auSquareInches	39	Cal kwadratowy (ang. <i>square inch</i>)
auSquareFeet	40	Stopa kwadratowa (ang. <i>square foot</i>)
auSquareYards	41	Jard kwadratowy (ang. <i>square yard</i>)
auSquareMiles	42	Łądowa mila kwadratowa
auAcres	43	Akr. Jednostka powierzchni gruntów używana w krajach anglosaskich
auCentares	44	Centar
auAres	45	Ar. Jednostka powierzchni gruntów odpowiadająca 100 metrom kwadratowym
auHectares	46	Hektar. Jednostka powierzchni stosowana do pomiarów gruntów
auSquareRods	47	Pręt kwadratowy (ang. <i>square rod</i>)

Patrz również

Convert, ConvTypeToDescription, DescriptionToConvFamily, DescriptionToConvType, GetConvTypes, TconvFamily

cbDistance — predefiniowana zmienna**Składnia**

```
var cbDistance: TconvFamily = 1;
```

Opis

cbDistance tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek odległości oraz długości. Zbiór aktualnie dostępnych jednostek przedstawia tabela 1.5.

Tabela 1.5. Predefiniowane jednostki należące do zbioru cbDistance reprezentującego układ jednostek długości oraz odległości (przedrostek du jest skrótem od distance units). Jednostką podstawową w tym zbiorze jest jeden metr (1 m)

Literał	Liczbowy identyfikator	Opis
duMicromicrons	1	Mikromikrometr. W układzie SI jednostka ta nazywana jest pikometrem. Reprezentuje skalę pośrednią pomiędzy wielkością promienia atomu a promieniem jądra atomowego
duAngstroms	2	Angström. Jednostka stosowana głównie w krytalografii do wyrażania odległości międzyatomowych
duMillimicrons	3	Milimikrometr. Jednostka ta w układzie SI nazywana jest nanometrem. Reprezentuje skalę wyrażającą średnie rozmiary cząsteczek
duMicrons	4	Mikrometr. Skala odpowiadająca rozmiarom typowej bakterii
duMillimeters	5	Milimetr
duCentimeters	6	Centymetr
duDecimeters	7	Decymetr
duMeters	8	Metr. Podstawowa jednostka długości w układzie SI
duDecameters	9	Dziesięć metrów
duHectometers	10	Sto metrów
duKilometers	11	Kilometr
duMegameters	12	Odpowiada milionowi metrów
duGigameters	13	Odpowiada miliardowi metrów
duInches	14	Cal

Tabela 1.5. Predefiniowane jednostki należące do zbioru `cbDistance` reprezentującego układ jednostek długości oraz odległości (przedrostek `du` jest skrótem od `distance units`). Jednostką podstawową w tym zbiorze jest jeden metr (1 m) — ciąg dalszy

Literał	Liczbowy identyfikator	Opis
<code>duFeet</code>	15	Stopa. Stopa jest całkowitą wielokrotnością cali. Trzy stopy tworzą jeden jard
<code>duYards</code>	16	Jard. Jard może być wielokrotnością stóp lub cali
<code>duMiles</code>	17	Mila lądowa. Anglosaska jednostka miary odległości na lądzie. Tzw. mila statutowa może dzielić się na jardy, stopy lub furlongi (staje)
<code>duNauticalMiles</code>	18	Mila morska. Jednostka miary odległości na morzu. Dzieli się na 10 kabli
<code>duAstronomicalUnits</code>	19	Jednostka astronomiczna stosowana w obrębie Układu Słonecznego. Odpowiada średniej odległości Ziemi od Słońca
<code>duLightYears</code>	20	Rok świetlny. Odległość, jaką przebywa światło w próżni w ciągu jednego roku zwrotnikowego
<code>duParsecs</code>	21	Parsek. Z definicji 1 ps to odległość, z jakiej połowa wielkiej osi orbity ziemskiej (tj. jednostka astronomiczna) jest widoczna jako łuk o długości 1 sekundy
<code>duCubits</code>	22	Odpowiednik 0,5 jarda
<code>duFathoms</code>	23	Sążeń. Jednostka długości oparta na systemie calowym równa rozpiętości rozstawionych ramion dorosłego mężczyzny. Sążeń może być wielokrotnością jardów, stóp lub cali
<code>duFurlongs</code>	24	Staje. Dawna miara odległości będąca wielokrotnością jardów lub łokci
<code>duHands</code>	25	Pięść. Odpowiednik 4 cali
<code>duPaces</code>	26	Krok. Odpowiednik 30 cali lub 2,5 stopy
<code>duRods</code>	27	Pręt. Jednostka długości odpowiadająca ok. 5,029 m
<code>duChains</code>	28	Łańcuch mierniczy. Jednostka długości odpowiadająca ok. 20,117 m
<code>duLinks</code>	29	Link. Jednostka długości odpowiadająca 1/100 łańcucha mierniczego
<code>duPicas</code>	30	Jednostka miary drukarskiej opartej na calu
<code>duPoints</code>	31	Punkt typograficzny. Jest jednostką miary w poligrafii. Stopień pisma drukarskiego posiada z reguły wielkość 12 punktów typograficznych

Patrz również

`Convert`, `ConvTypeToDescription`, `DescriptionToConvFamily`, `DescriptionToConvType`, `GetConvTypes`, `TConvFamily`

cbMass — predefiniowana zmienna

Składnia

```
var cbMass: TConvFamily = 4;
```

Opis

cbMass tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek masy. Zbiór aktualnie dostępnych jednostek przedstawia tabela 1.6.

Tabela 1.6. Predefiniowane jednostki należące do zbioru cbMass reprezentującego układ jednostek masy (przedrostek mu jest skrótem od mass units). Jednostką podstawową w tym zbiorze jest jeden gram (1 g)

Literał	Liczbowy identyfikator	Opis
muNanograms	96	Nanogram
muMicrograms	97	Mikrogram
muMilligrams	98	Miligram
muCentigrams	99	Odpowiednik 0,01 grama
muDecigrams	100	Odpowiednik 0,1 grama
muGrams	101	Gram
muDecagrams	102	Dekagram
muHectograms	103	Odpowiada 100 gramom
muKilograms	104	Kilogram. Podstawowa jednostka masy w układzie SI
muMetricTons	105	Tona metryczna
muDrams	106	Drachma. Jednostka masy handlowej równa ok. 1,772 g lub aptekarska jednostka masy stosowana w krajach anglosaskich równa ok. 3,888 g
muGrains	107	Grain. Aptekarska jednostka masy stosowana w krajach anglosaskich, równa ok. 0,0648 g
muLongTons	109	Anglosaska jednostka masy, tzw. tona angielska
muTons	108	Tona. Jednostka masy stosowana w technice
muOunces	110	Uncja. Anglosaska jednostka masy
muPounds	111	Funt. Brytyjska jednostka masy
muStones	0	Kamień. Anglosaska jednostka masy równa ok. 6,35 kg

Patrz również

Convert, ConvTypeToDescription, DescriptionToConvFamily, DescriptionToConvType, GetConvTypes, TConvFamily, TConvType

cbTemperature — predefiniowana zmienna

Składnia

```
var cbTemperature: TConvFamily = 5;
```

Opis

cbTemperature tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek temperatury. Zbiór aktualnie dostępnych jednostek przedstawia tabela 1.7.

Tabela 1.7. Predefiniowane jednostki należące do zbioru cbTemperature reprezentującego układ jednostek temperatury (przedrostek tu jest skrótem od temperature units). Jednostką podstawową w tym zbiorze jest jeden stopień Celsjusza (1 °C)

Literał	Liczbowy identyfikator	Opis
tuCelsius	112	Skala temperatur Celsjusza °C. Temperaturę wrzenia wody pod normalnym ciśnieniem określa się jako 100 °C, zaś 0 °C jest temperaturą zamarzania wody pod tym samym ciśnieniem
tuKelvin	113	Bezwzględna termodynamiczna skala temperatur Kelvina K. Jednostka temperatury termodynamicznej stanowi 1/273,16 części temperatury punktu potrójnego wody. Temperatura zera bezwzględnego T = 0 K odpowiada stanowi materii o najniższej możliwej energii. Przyjmuje się, iż średnia energia kinetyczna ruchu cieplnego molekuł jest wprost proporcjonalna do tzw. czynnika Boltzmana $k_B T$, gdzie T jest bezwzględną temperaturą. Jednostka temperatury termodynamicznej jest podstawową jednostką temperatury w układzie SI
tuFahrenheit	114	Skala temperatur Fahrenheita °F używana w krajach anglosaskich. Punktem zerowym jest temperatura zamarzania mieszaniny salamiaku (chlorku amonu) z lodem. Przeliczenia skali temperatur Celsjusza na Fahrenheita odbywa się zgodnie z zależnością $T_c = 5/9(T_F - 32)$
tuRankine	115	Skala temperatur Rankine'a °Rank. Modyfikacja skali Fahrenheita poprzez przesunięcie punktu zerowego do temperatury zera bezwzględnego
tuReaumur	116	Obecnie już nie używana skala temperatur Reaumura. W skali Reaumura temperatura topnienia lodu (zamarzania wody) odpowiada 0 °C, zaś wrzenia wody 80 °R (100 °C), dlatego 1 °C odpowiada 0,8 °R w skali Reaumura

Patrz również

Convert, ConvTypeToDescription, DescriptionToConvFamily, DescriptionToConvType, GetConvTypes, TConvFamily, TConvType

cbTime — predefiniowana zmienna

Składnia

```
var cbTime: TconvFamily = 6;
```

Opis

cbTime tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek daty oraz czasu. Zbiór aktualnie dostępnych jednostek przedstawia tabela 1.8.

Tabela 1.8. *Predefiniowane jednostki należące do zbioru cbTime reprezentującego układ jednostek daty i czasu (przedrostek tu jest skrótem od time units). Jednostką podstawową w tym zbiorze jest jeden dzień (1 d)*

Literał	Liczbowy identyfikator	Opis
tuMilliseconds	117	Milisekunda
tuSeconds	118	Sekunda. Sekunda jest podstawową jednostką odstępu czasu w układzie SI
tuMinutes	119	Minuta
tuHours	120	Godzina
tuDays	121	Dzień
tuWeeks	122	Tydzień. Kalendarzowa jednostka rachuby czasu składająca się z 7 dni. W USA jako pierwszy dzień tygodnia traktowana jest niedziela. Zgodnie ze standardem ISO 8601 pierwszym dniem tygodnia jest poniedziałek
tuFortnights	123	Dwa tygodnie
tuMonths	124	Miesiąc. Kalendarzowa jednostka rachuby czasu związana z cyklem faz Księżyca. Miesiąc może liczyć od 28 do 31 dni
tuYears	125	Rok. Jednostka rachuby czasu związana z okresem obiegu Ziemi wokół Słońca
tuDecades	126	Dekada. Dekada liczy 10 lat
tuCenturies	127	Wiek. Okres trwający 100 lat (stulecie)
tuMillennia	128	Milenium. Okres trwający 1 000 lat
tuDateTime	129	Jeden dzień
tuJulianDate	130	Data związana z kalendarzem juliańskim obowiązującym od 46 roku p.n.e. do roku 1582
tuModifiedJulianDate	131	Modyfikacja daty związanej z kalendarzem juliańskim

Patrz również

Convert, ConvTypeToDescription, DescriptionToConvFamily, DescriptionToConvType, GetConvTypes, TConvFamily, TConvType

cbVolume — predefiniowana zmienna

Składnia

```
var cbVolume: TConvFamily = 3;
```

Opis

cbVolume tworzy zbiór predefiniowanych jednostek należących do szerokiego układu jednostek objętości. Zbiór aktualnie dostępnych jednostek przeliczeniowych przedstawia tabela 1.9.

Tabela 1.9. Predefiniowane jednostki należące do zbioru cbVolume reprezentującego układ jednostek objętości (przedrostek vu jest skrótem od volume units). Jednostką podstawową w tym zbiorze jest jeden metr sześcienny (1 m³)

Literał	Liczbowy identyfikator	Opis
vuCubicMillimeters	48	Milimetry sześciennie
vuCubicCentimeters	49	Centymetry sześciennie
vuCubicDecimeters	50	Decymetry sześciennie
vuCubicMeters	51	Metry sześciennie
vuCubicDecameters	52	Tysiąc metrów sześciennych
vuCubicHectometers	53	Milion metrów sześciennych
vuCubicKilometers	54	Kilometr sześcienny
vuCubicInches	55	Cał sześcienny
vuCubicFeet	56	Stopa sześcienna
vuCubicYards	57	Jard sześcienny
vuCubicMiles	58	Mila sześcienna
vuMilliLiters	59	Mililitr
vuCentiLiters	60	1/100 część litra
vuDeciLiters	61	1/10 część litra
vuLiters	62	Litr. Jednostka objętości stosowana jako miara ilości płynu (cieczy) lub ciał sypkich
vuDecaLiters	63	Dziesięć litrów
vuHectoLiters	64	Sto litrów
vuKiloLiters	65	Tysiąc litrów — kilolitr
vuAcreFeet	66	Acre foot. Jednostka objętości stosowana w melioracji. Określa objętość cieczy w warstwie o grubości jednej stopy i powierzchni jednego akra

Tabela 1.9. Predefiniowane jednostki należące do zbioru *cbVolume* reprezentującego układ jednostek objętości (przedrostek *vu* jest skrótem od *volume units*). Jednostką podstawową w tym zbiorze jest jeden metr sześcienny (1 m^3) — ciąg dalszy

Literał	Liczbowy identyfikator	Opis
vuAcreInches	67	Acre inch. Jednostka objętości stosowana w melioracji. Określa objętość cieczy w warstwie o grubości jednego cala i powierzchni jednego akra
vuCords	68	Cord. Anglosaska jednostka objętości odpowiadająca 128 stopom sześciennym
vuCordFeet	69	Jednostka objętości odpowiadająca 453,06 litrom, tj. ok. 6,25 tony rejestrowej
vuDecisteres	70	Decystere. Jednostka objętości ciał płynnych odpowiadająca stu litrom
vuSteres	71	Stere. Jednostka objętości ciał płynnych odpowiadająca metrowi sześciennemu
vuDecasteres	72	Decastere. Jednostka objętości ciał płynnych odpowiadająca dziesięciu tysiącom litrów
vuFluidGallons	73	Galon (U.S. gal). Jednostka objętości cieczy używana w USA
vuFluidQuarts	74	Kwarta (U.S. fl qt), (U.S. liq qt). Jednostka objętości cieczy używana w USA
vuFluidPints	75	Pinta (U.S. fl pt), (U.S. liq pt). Jednostka objętości cieczy używana w USA. Odpowiada objętości ok. 0,473 litra
vuFluidCups	76	Miska lub miseczka jako kulinarna jednostka objętości cieczy
vuFluidGills	77	Gill (U.S. fl gi), (U.S. liq gi). Jednostka objętości cieczy równa 0,125 amerykańskiej kwarty, czyli połowie ćwiartki kwarty
vuFluidOunces	78	Uncja (U.S. fl oz). Amerykańska jednostka objętości cieczy
vuFluidTablespoons	79	Łyżka stołowa jako jednostka objętości cieczy. Miara wprowadzona w celu zachowania norm kulinarnych
vuFluidTeaspoons	80	Mała łyżka stołowa jako jednostka objętości cieczy. Miara wprowadzona w celu zachowania norm kulinarnych
vuDryGallons	81	Galon (U.S. dry gal). Jednostka objętości ciał sypkich używana w krajach anglosaskich
vuDryQuarts	82	Kwarta (U.S. dry qt). Jednostka objętości ciał sypkich używana w krajach anglosaskich
vuDryPints	83	Pinta (U.S. dry pt). Jednostka objętości (1/8 galona) ciał sypkich używana w krajach anglosaskich
vuDryPecks	84	Amerykańska jednostka objętości ciał sypkich używana w krajach anglosaskich. Odpowiada ok. 8,81 litra

Tabela 1.9. Predefiniowane jednostki należące do zbioru *cbVolume* reprezentującego układ jednostek objętości (przedrostek *vu* jest skrótem od *volume units*). Jednostką podstawową w tym zbiorze jest jeden metr sześcienny (1 m^3) — ciąg dalszy

Literał	Liczbowy identyfikator	Opis
<code>vuDryBuckets</code>	85	Bucket. Czerpak lub wiadro (urządzenia załadownicze stosowane w koparkach ziemnych). Jednostka objętości ciał sypkich używana w krajach anglosaskich. Odpowiada objętości znormalizowanego wiadra, ok. 17,6 litra
<code>vuDryBushels</code>	86	Buszel (U.S. bu) jako jednostka objętości ciał sypkich
<code>vuUKGallons</code>	87	Imperialny brytyjski galon (U.K. gal)
<code>vuUKPottles</code>	88	Imperialny brytyjski garniec
<code>vuUKQuarts</code>	89	Imperialna brytyjska kwarta (U.K. qt)
<code>vuUKPints</code>	90	Imperialna brytyjska pinta (U.K. pt). Odpowiada objętości ok. 0,568 litra
<code>vuUKGills</code>	91	Imperialna brytyjska ćwierć kwarty (U.K. gi)
<code>vuUKOunces</code>	92	Imperialna brytyjska uncja (U.K. oz)
<code>vuUKPecks</code>	93	Imperialna brytyjska jednostka objętości odpowiadająca ok. 9,092 litra
<code>vuUKBuckets</code>	94	Imperialna brytyjska jednostka objętości. Odpowiada objętości znormalizowanego wiadra, ok. 18,1 litra
<code>vuUKBushels</code>	95	Imperialny brytyjski buszel (U.K. bu)

Patrz również

`Convert`, `ConvTypeToDescription`, `DescriptionToConvFamily`, `DescriptionToConvType`, `GetConvTypes`, `TConvFamily`, `TConvType`

CelsiusToFahrenheit() — funkcja

Składnia

```
function CelsiusToFahrenheit(const AValue: Double): Double;
```

Opis

Funkcja umożliwia przeliczenie temperatury wyrażonej w skali Celsjusza i reprezentowanej przez parametr `AValue` na temperaturę wyrażoną w skali Fahrenheita.

Patrz również

`Convert()`, `FahrenheitToCelsius()`

CompatibleConversionType() — funkcja

Składnia

```
function CompatibleConversionType(const AType: TConvType; const
                                AFamily: TConvFamily): Boolean;
```

Opis

Funkcja sprawdza, czy wyspecyfikowany typ jednostki zarejestrowany jest w układzie określonych jednostek. `CompatibleConversionType()` w wyniku działania zwraca wartość prawdziwą (`True`), jeżeli stała `AType` reprezentuje element zbioru określonego przez `AFamily`. W przeciwnym wypadku funkcja zwraca fałsz (`False`).

Przykład

Poniższy przykład przedstawia główny moduł projektu `Kody\Rozdzial1\CompatibleConversionType\p_CompatibleConversionType.dpr`. Określenie prawidłowości wyboru danego typu literału dokonane zostało w oparciu o ich liczbowe identyfikatory oraz odpowiednie rzutowanie na typy `TConvType` oraz `TConvFamily`.

```
unit Unit_CompatibleConversionType;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, StdCtrls, ConvUtils, StdConvs;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.ItemIndex := 0;
  ListBox1.Items[0] := 'tuCelsius'; //112
```

```

ListBox1.Items[1] := 'tuKelvin'; //113
ListBox1.Items[2] := 'tuFahrenheit';//114
ListBox1.Items[3] := 'tuRankine'; //115
ListBox1.Items[4] := 'tuReaumur'; //116

ListBox2.ItemIndex := 0;
ListBox2.Items[0] := 'cbDistance'; //1
ListBox2.Items[1] := 'cbArea'; //2
ListBox2.Items[2] := 'cbVolume'; //3
ListBox2.Items[3] := 'cbMass'; //4
ListBox2.Items[4] := 'cbTemperature';//5
ListBox2.Items[5] := 'cbTime'; //6
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
begin
  if(CompatibleConversionType(
    TConvType(ListBox1.ItemIndex +112),
    TConvFamily(ListBox2.ItemIndex +1))) then
    ShowMessage(Format('%s %s %s',
      [ListBox1.Items[ListBox1.ItemIndex],
        ' należy do układu jednostek',
        ListBox2.Items[ListBox2.ItemIndex]]))
    else
      ShowMessage(Format('%s %s %s',
        [ListBox1.Items[ListBox1.ItemIndex],
          ' nie należy do układu jednostek',
          ListBox2.Items[ListBox2.ItemIndex]]))
end;
//-----
end.

```

Patrz również

Convert(), ConvTypeToFamily(), RegisterConversionType(), TryConvTypeToFamily(), TConvFamily, TConvType

CompatibleConversionTypes() — funkcja

Składnia

```
function CompatibleConversionTypes(const AFrom, ATo: TConvType):
  Boolean;
```

Opis

Użycie CompatibleConversionTypes() określa, czy funkcja Convert() może dokonać przeliczenia dwóch wielkości wyrażonych w wybranych jednostkach. CompatibleConversionTypes() zwraca prawdę jeżeli AFrom oraz ATo należą do tego samego układu jednostek. Funkcji używamy wówczas, jeżeli chcemy mieć pewność, że w trakcie działania aplikacja nie podejmie próby przeliczenia np. długości ciała na jego masę czy temperatury ciała na jego objętość.

Patrz również

Convert(), CompatibleConversionTypes(), ConvTypeToFamily(), RegisterConversionType(), TryConvTypeToFamily(), TConvFamily, TConvType

Convert() — funkcja**Składnia**

```
function Convert(const AValue: Double; const AFrom, ATo: TConvType):
    Double; overload;
function Convert(const AValue: Double; const AFrom1, AFrom2, ATo1,
    ATo2: TConvType): Double; overload;
```

Opis

Funkcja Convert() dokonuje przeliczenia wielkości wyrażanych poprzez dwie różne jednostki. Parametr AValue jest wartością, którą chcemy przeliczyć. W pierwszej postaci funkcji parametr AFrom określa bieżący układ jednostek. Funkcja Convert() poprzez parametr ATo zwraca przeliczoną wartość AValue. Używając drugiej postaci funkcji można dokonać bardziej skomplikowanej konwersji danych. Parametry AFrom1 oraz AFrom2 określają aktualne jednostki, w których wyrażone są wartości AValue, gdzie AValue posiada wymiar [AFrom1/AFrom2], np. [mi/gal]. Funkcja zwraca wartość AValue przeliczoną na wymiar [ATo1/ATo2], np. [km/l].

Jeżeli pojazd zużywa ilość paliwa AValue podaną w galonach na mile, my zaś chcemy przeliczyć to na wielkość zużycia wyrażoną w litrach na kilometr, wówczas wystarczy napisać.

```
Convert(AValue, duMiles, vuGallons, duKilometers, vuLiters);
```

Błędy

AFrom i ATo, AFrom1 i ATo1 oraz AFrom2 i ATo2 muszą należeć do tych samych układów jednostek. Określenia zgodności wybranych jednostek można dokonać poprzez wywołanie funkcji CompatibleConversionTypes(). Jeżeli jednostki nie należą do tego samego układu jednostek, funkcja Convert() generuje wyjątek EConversionError.

Przykład

Poniższy przykład przedstawia aplikację reprezentowaną przez projekt znajdujący się na dołączonej do książki płycie CD-ROM w katalogu *Kody\Rozdzial1\Convert\p_Convert.dpr*, dzięki której można dokonać wzajemnej konwersji wielkości opisujących długość lub odległość oraz pole powierzchni. W przykładzie tym nie zastosowano funkcji CompatibleConversionTypes() oraz CompatibleConversionType() do określenia zgodności typów podanych jednostek. Zamiast wykorzystania tych funkcji zastosowano nowoczesne komponenty biznesowe będące reprezentantami klasy TFrame. Posługiwanie się tego typu komponentami w znacznym stopniu upraszcza budowę aplikacji oraz minimalizuje prawdopodobieństwo popełnienia błędu przez programistę. Wzajemne przeliczanie jednostek długości (odległości) dokonywane jest w module *cbDistance.pas* (komponent

Frame1), zaś jednostek, w których wyrażane jest pole powierzchni w module *cbArea.pas* (komponent Frame2). Rozróżnianie odpowiednich jednostek następuje dzięki przypisaniu odpowiadających im indeksów (liczbowych identyfikatorów).

```

unit cbDistance;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs,
  StdCtrls, ConvUtils, StdConvs, ExtCtrls;

type
  TFrame1 = class(TFrame)
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    function ConvertToString: string;
  public
    { Public declarations }
  end;

var
  ConvertFrom, ConvertTo: Int64;

implementation

{$R *.dfm}
function TFrame1.ConvertToString: string;
begin
  Result := Format('%g %s',
    [Convert(Abs(StrToFloat(Edit1.Text)),
      TConvType(ConvertFrom),
      TConvType(ConvertTo)), '']);
end;
//-----
procedure TFrame1.Button1Click(Sender: TObject);
begin
  ConvertFrom := RadioGroup1.ItemIndex + 1;
  ConvertTo := RadioGroup2.ItemIndex + 1;
  Edit2.Text:=ConvertToString;
end;
//-----
end.

unit cbArea;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls,
  ConvUtils, StdConvs;

type

```

```

TFrame2 = class(TFrame)
  RadioGroup1: TRadioGroup;
  RadioGroup2: TRadioGroup;
  Button1: TButton;
  Edit1: TEdit;
  Edit2: TEdit;
  procedure Button1Click(Sender: TObject);
private
  { Private declarations }
  function ConvertToString: string;
public
  { Public declarations }
end;

var
  ConvertFrom, ConvertTo: Int64;

implementation

{$R *.dfm}
function TFrame2.ConvertToString: string;
begin
  Result := Format('%g %s',
    [Convert(Abs(StrToFloat(Edit1.Text)),
      TConvType(ConvertFrom),
      TConvType(ConvertTo)), '']);
end;
//-----
procedure TFrame2.Button1Click(Sender: TObject);
begin
  ConvertFrom := RadioGroup1.ItemIndex + 32;
  ConvertTo := RadioGroup2.ItemIndex + 32;
  Edit2.Text:=ConvertToString;
end;
//-----
end.

// główny moduł projektu p_Convert.dpr
unit Unit_Convert;

interface

uses
  Windows, Messages, SysUtils, Variants,
  Classes, Graphics, Controls, Forms,
  Dialogs, cbArea, Menus, cbDistance;

type
  TForm1 = class(TForm)
    Frame21: TFrame2;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Exit1: TMenuItem;
    Frame11: TFrame1;
    Ukryj1: TMenuItem;
    cbAreaHide: TMenuItem;
    cbDistanceHide: TMenuItem;
    N1: TMenuItem;
    cbAreaShow: TMenuItem;
    cbDistanceShow: TMenuItem;
    procedure Exit1Click(Sender: TObject);
    procedure cbAreaShowClick(Sender: TObject);
    procedure cbDistanceShowClick(Sender: TObject);
  end;

```

```

        procedure cbAreaHideClick(Sender: TObject);
        procedure cbDistanceHideClick(Sender: TObject);
        procedure FormCreate(Sender: TObject);
        procedure Frame21Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    Frame11.RadioGroup1.ItemIndex := 0;
    Frame11.RadioGroup2.ItemIndex := 0;
    Frame21.RadioGroup1.ItemIndex := 0;
    Frame21.RadioGroup2.ItemIndex := 0;
    Frame11.DragMode := dmAutomatic;
    Frame11.DragKind := dkDock;
    Frame21.DragMode := dmAutomatic;
    Frame21.DragKind := dkDock;
end;
//-----
procedure TForm1.Exit1Click(Sender: TObject);
begin
    Application.Terminate();
end;
//-----
procedure TForm1.cbAreaShowClick(Sender: TObject);
begin
    Frame21.Show();
end;
//-----
procedure TForm1.cbDistanceShowClick(Sender: TObject);
begin
    Frame11.Show();
end;
//-----
procedure TForm1.cbAreaHideClick(Sender: TObject);
begin
    Frame21.Hide();
end;
//-----
procedure TForm1.cbDistanceHideClick(Sender: TObject);
begin
    Frame11.Hide();
end;
//-----
procedure TForm1.Frame21Button1Click(Sender: TObject);
begin
    Frame21.Button1Click(Sender);
end;

end.

```

Alternatywny sposób rozwiązania przedstawionego zagadnienia można znaleźć w plikach pomocy Delphi 6.

Patrz również

CompatibleConversionTypes(), CompatibleConversionTypes(), ConvertFrom(), ConvertTo(), ConvTypeToFamily(), RegisterConversionType(), TryConvTypeToFamily(), TConvFamily, TConvType

ConvertFrom() — funkcja

Składnia

```
function ConvertFrom(const AFrom: TConvType; const AValue: Double):
    Double;
```

Opis

Funkcja `ConvertFrom()` przelicza wartość reprezentowaną przez `AValue` do wartości reprezentowanej w jednostkach układu `AFrom`. Parametr `AValue` przyjmowany jest jako jednostka podstawowa zbioru wielkości (np. dla `cbDistance` jest to 1 m). Jeżeli parametr `AFrom` określa jedynie nazwę układu jednostek, przeliczenie zostanie wykonane w taki sposób, iż parametrowi `AValue` zostanie przypisana pierwsza jednostka występująca w tym zbiorze. Jeżeli `AFrom` jawnie określa typ jednostki należącej do danego układu jednostek, przeliczenie zostanie wykonane zgodnie z podanym typem jednostki.

W przypadku, gdy `AFrom` nie jest zarejestrowanym typem jednostki, `ConvertFrom()` generuje wyjątek `EConversionError`.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit2.Text := FloatToStr(ConvertFrom(duDecimeters,
        StrToFloat(Edit1.Text)));
    Edit3.Text := FloatToStr(ConvertFrom(cbDistance,
        StrToFloat(Edit1.Text)));
end;
```

Patrz również

Convert(), ConvertTo(), RegisterConversionType(), TConvFamily

ConvertTo() — funkcja

Składnia

```
function ConvertTo(const AValue: Double; const ATo: TConvType):
    Double;
```

Opis

Funkcja `ConvertTo()` przelicza wartość reprezentowaną przez `AValue` do wartości reprezentowanej przez wielkość wyrażoną w jednostkach `ATo`. Parametr `AValue` przyjmowany jest jako jednostka podstawowa układu jednostek (np. dla `cbDistance` jest to 1 m). Jeżeli parametr `ATo` określa jedynie nazwę zbioru jednostek, parametr `AValue` zostanie potraktowany jako najmniejsza jednostka danego zbioru, zaś przeliczenie wartości nastąpi do jednostki podstawowej układu jednostek. Jeżeli `ATo` jawnie określa konkretną jednostkę należącą do danego zbioru, przeliczenie zostanie wykonane zgodnie z podaną wartością.

W przypadku, gdy `ATo` nie jest zarejestrowanym typem jednostki, `ConvertTo()` generuje wyjątek `EConversionError`.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit2.Text := FloatToStr(ConvertTo(StrToFloat(Edit1.Text),
    duDecimeters));
  Edit3.Text := FloatToStr(ConvertTo(StrToFloat(Edit1.Text),
    cbDistance));
end;
```

Patrz również

`Convert()`, `ConvertFrom()`, `RegisterConversionType()`, `TConvFamily`

ConvFamilyToDescription() — funkcja

Składnia

```
function ConvFamilyToDescription(const AFamily: TConvFamily): string;
```

Opis

Funkcja zwraca łańcuch znaków reprezentujący pełną nazwę (bez przedrostka) układu predefiniowanych jednostek, którego elementy podlegają przeliczaniu. Jeżeli `AFamily` reprezentuje nazwę pojedynczej jednostki należącej do danego zbioru (np. `tuFahrenheit`), funkcja zwraca odpowiadający jej numer w postaci heksadecymalnej.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := ConvFamilyToDescription(cbMass);
end;
```

Patrz również

`ConvTypeToDescription()`, `DescriptionToConvFamily()`, `RegisterConversionFamily()`

ConvTypeToDescription() — funkcja

Składnia

```
function ConvTypeToDescription(const AType: TConvType): string;
```

Opis

Funkcja zwraca łańcuch znaków reprezentujący pełną nazwę (bez przedrostka) określonej jednostki. Wyszczególniona jednostka musi być elementem zarejestrowanego układu jednostek.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Text := ConvTypeToDescription(tuFahrenheit);
end;
```

Patrz również

ConvFamilyToDescription(), ConvTypeToFamily(), ConvUnitToStr(), DescriptionToConvType(), RegisterConversionType()

ConvTypeToFamily() — funkcja

Składnia

```
function ConvTypeToFamily(const AType: TConvType): TConvFamily;
    overload;
function ConvTypeToFamily(const AFrom, ATo: TConvType): TConvFamily;
    overload;
```

Opis

Funkcja ConvTypeToFamily() zwraca identyfikator układu jednostek, którego elementy podlegają przeliczaniu. Parametr AType jest zarejestrowanym typem jednostki. Jeżeli istnieje potrzeba sprawdzenia, czy dwie jednostki należą do tego samego układu, należy skorzystać z drugiej postaci funkcji. Jeżeli parametr AType nie jest zarejestrowany lub AFrom oraz ATo nie są zarejestrowane w tym samym układzie jednostek, funkcja ConvTypeToFamily() generuje wyjątek EConversionError.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit1.Text := VarToStr(ConvTypeToFamily(vuCubicFeet));
    Edit2.Text := VarToStr(ConvTypeToFamily(vuCubicYards, tuRankine));
end;
```

Patrz również

CompatibleConversionType(), ConvTypeToDescription(), DescriptionToConvFamily(), RegisterConversionType(), TryConvTypeToFamily()

ConvUnitAdd() — funkcja

Składnia

```
function ConvUnitAdd(const AValue1: Double; const AType1: TConvType;
                    const AValue2: Double; const AType2,
                    AResultType: TConvType): Double;
```

Opis

Funkcja ConvUnitAdd() dodaje dwie wartości AValue1 oraz AValue2. Parametry AType1 oraz AType2 reprezentują czynniki przeliczeniowe należące do układu jednostek, w których wyrażone są odpowiednio wartości AValue1 oraz AValue2. AResultType jest jednostką, w której chcemy otrzymać wynik. Trzeba zwrócić uwagę na to, iż korzystając z przedstawionej funkcji należy rozsądnie wybierać czynniki przeliczeniowe wielkości, które mają być dodane (np. nie można otrzymać sensownego wyniku z dodania 1 litra do 1 kilograma), oraz jednostkę, w której chcemy otrzymać wynik. Funkcja ta nie działa poprawnie przy próbie dodania do siebie dwóch wielkości posiadających identyczne czynniki przeliczeniowe (jednostki). Wyjątkiem są podstawowe jednostki danego zbioru wielkości, np. metr, stopień Kelvina, itp. Dokładniej z tym zagadnieniem Czytelnik może zapoznać się testując projekt *Kody\Rozdział1\ConvUnitAdd\p_ConvUnitAdd.dpr*, którego główny moduł został przedstawiony w poniższym przykładzie.

Przykład

```
// Dodanie dwóch różnych wielkości reprezentujących
// różne skale temperatur
unit Unit_ConvUnitAdd;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, StdCtrls, ConvUtils, StdConvs;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    ListBox3: TListBox;
    Label1: TLabel;
    Label2: TLabel;
```

```

    Label3: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    ListBox1.ItemIndex := 0;
    ListBox1.Items[0] := 'Celsius'; //112
    ListBox1.Items[1] := 'Kelvin'; //113
    ListBox1.Items[2] := 'Fahrenheit'; //114
    ListBox1.Items[3] := 'Rankine'; //115
    ListBox1.Items[4] := 'Reamur'; //116

    ListBox2.ItemIndex := 0;
    ListBox2.Items[0] := 'Celsius'; //112
    ListBox2.Items[1] := 'Kelvin'; //113
    ListBox2.Items[2] := 'Fahrenheit'; //114
    ListBox2.Items[3] := 'Rankine'; //115
    ListBox2.Items[4] := 'Reamur'; //116

    ListBox3.ItemIndex := 0;
    ListBox3.Items[0] := 'Celsius'; //112
    ListBox3.Items[1] := 'Kelvin'; //113
    ListBox3.Items[2] := 'Fahrenheit'; //114
    ListBox3.Items[3] := 'Rankine'; //115
    ListBox3.Items[4] := 'Reamur'; //116
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    Edit3.Text := FloatToStr(ConvUnitAdd(StrToFloat(Edit1.Text),
        TConvType(ListBox1.ItemIndex +112), StrToFloat(Edit2.Text),
        TConvType(ListBox2.ItemIndex +112),
        TConvType(ListBox3.ItemIndex +112)));

end;
//-----
end.

```

Patrz również

ConvUnitDec(), ConvUnitDiff(), ConvUnitInc()

ConvUnitCompareValue() — funkcja

Składnia

```
function ConvUnitCompareValue(const AValue1: Double; const AType1:
                               TConvType; const AValue2: Double; const
                               AType2: TConvType): TValueRelationship;
```

Opis

Funkcja ConvUnitCompareValue() porównuje dwie wartości ze względu na ich aktualne jednostki. Testowane wartości muszą reprezentować wyniki pomiarów tych samych wielkości należących do tego samego układu jednostek. AValue1 oraz AValue2 są wartościami, które poddawane są operacji porównania. AType1 oraz AType2 są jednostkami odpowiednio dla AValue1 oraz AValue2.

Funkcja ConvUnitCompareValue() zwraca wartość:

- ◆ LessThanValue – (-1) jeżeli AValue1 jest mniejsze niż AValue2;
- ◆ EqualsValue – (0) jeżeli AValue1 jest równe AValue2;
- ◆ GreaterThanValue (1) — jeżeli AValue1 jest większe niż AValue2.

Jeżeli zechcemy jawnie korzystać z przedstawionych predefiniowanych stałych, w deklaracji uses głównego modułu aplikacji należy włączyć moduł Types.

Przykład

Aplikacja projektu *Kody\Rozdział1\ConvUnitComparevalue\p_ConvUnitComparevalue.dpr* przedstawia jeden ze sposobów porównania długości mierzonego czasu.

```
unit Unit_ConvUnitComparevalue;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Types,
  Dialogs, StdCtrls, ConvUtils, StdConv;

type
  TForm1 = class(TForm)
    ListBox1: TListBox;
    ListBox2: TListBox;
    Button1: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```

var
  Form1: TForm1;
implementation

{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.ItemIndex := 0;
  ListBox1.Items[0] := 'Milliseconds'; //117
  ListBox1.Items[1] := 'Seconds'; //118
  ListBox1.Items[2] := 'Minutes'; //119
  ListBox1.Items[3] := 'Hours'; //120
  ListBox1.Items[4] := 'Days'; //121
  ListBox1.Items[5] := 'Weeks'; //122
  ListBox1.Items[6] := 'Fortnights'; //123
  ListBox1.Items[7] := 'Months'; //124

  ListBox2.ItemIndex := 0;
  ListBox2.Items[0] := 'Milliseconds'; //117
  ListBox2.Items[1] := 'Seconds'; //118
  ListBox2.Items[2] := 'Minutes'; //119
  ListBox2.Items[3] := 'Hours'; //120
  ListBox2.Items[4] := 'Days'; //121
  ListBox2.Items[5] := 'Weeks'; //122
  ListBox2.Items[6] := 'Fortnights'; //123
  ListBox2.Items[7] := 'Months'; //124

end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
begin
  case(ConvUnitCompareValue(Abs(StrToFloat(Edit1.Text)),
    TConvType(ListBox1.ItemIndex +117),
    Abs(StrToFloat(Edit2.Text)),
    TConvFamily(ListBox2.ItemIndex +117))) of

    LessThanValue:
      ShowMessage(Format('%g %s %s %g %s',
        [Abs(StrToFloat(Edit1.Text)),
        ListBox1.Items[ListBox1.ItemIndex],
        ' ma wartość mniejszą niż ',
        Abs(StrToFloat(Edit2.Text)),
        ListBox2.Items[ListBox2.ItemIndex]]));

    EqualsValue:
      ShowMessage(Format('%g %s %s %g %s',
        [Abs(StrToFloat(Edit1.Text)),
        ListBox1.Items[ListBox1.ItemIndex],
        ' jest równe ',
        Abs(StrToFloat(Edit2.Text)),
        ListBox2.Items[ListBox2.ItemIndex]]));

    GreaterThanValue:
      ShowMessage(Format('%g %s %s %g %s',
        [Abs(StrToFloat(Edit1.Text)),
        ListBox1.Items[ListBox1.ItemIndex],
        ' ma wartość większą niż ',

```

```

        Abs(StrToFloat(Edit2.Text)),
        ListBox2.Items[ListBox2.ItemIndex]));
    end;

end;
//-----
end.

```

Patrz również

CompareValue(), ConvUnitSameValue(), TValueRelationship

ConvUnitDec() — funkcja

Składnia

```

function ConvUnitDec(const AValue: Double; const AType: TConvType;
                    const AAmount: Double; const AAmountType:
                    TConvType): Double; overload;

function ConvUnitDec(const AValue: Double; const AType, AAmountType:
                    TConvType): Double; overload;

```

Opis

Funkcja ConvUnitDec() zmniejsza wartość wyspecyfikowanej wielkości o zadaną liczbę. Parametry AValue oraz AType są odpowiednio oryginalną wyjściową wartością wielkości oraz jej jednostką. AAmount jest wartością, którą odejmujemy od wartości wyjściowej. W przypadku, gdy nie została ona ustalona domyślnie przyjmuje się 1.

AAmountType jest typem jednostki, w której wyrażany jest parametr AAmount.

Przykład

Poniższy fragment kodu obrazuje prosty sposób manipulacji czasem.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    ListBox1.ItemIndex := 0;
    ListBox1.Items[0] := 'Minutes'; //119
    ListBox1.Items[1] := 'Hours'; //120
    ListBox1.Items[2] := 'Days'; //121
    ListBox1.Items[3] := 'Weeks'; //122
    ListBox1.Items[4] := 'Fortnights'; //123
    ListBox1.Items[5] := 'Months'; //124
    ListBox1.Items[6] := 'Years'; //125
    ListBox1.Items[7] := 'Decades'; //126
    ListBox1.Items[8] := 'Centuries'; //127
    ListBox1.Items[9] := 'Millennia'; //128
    ListBox1.Items[10] := 'DateTime'; //129
    ListBox2.ItemIndex := 0;
    ListBox2.Items[0] := 'Minutes'; //119
    ListBox2.Items[1] := 'Hours'; //120

```

```

ListBox2.Items[2] := 'Days';           //121
ListBox2.Items[3] := 'Weeks';         //122
ListBox2.Items[4] := 'Fortnights';    //123
ListBox2.Items[5] := 'Months';        //124
ListBox2.Items[6] := 'Years';         //125
ListBox2.Items[7] := 'Decades';       //126
ListBox2.Items[8] := 'Centuries';     //127
ListBox2.Items[9] := 'Millennia';     //128
ListBox2.Items[10] := 'DateTime';     //129

end;
//-----
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  Edit2.Text := FloatToStr(ConvUnitDec(Abs(StrToFloat(Edit1.Text)),
    TConvType(ListBox1.ItemIndex +119),
    SpinEdit1.Value,
    TConvFamily(ListBox2.ItemIndex +119)));
end;

```

Patrz również

ConvUnitAdd(), ConvUnitDiff(), ConvUnitInc()

ConvUnitDiff() — funkcja

Składnia

```

function ConvUnitDiff(const AValue1: Double; const AType1: TConvType;
  const AValue2: Double; const AType2,
  AResultType: TConvType): Double;

```

Opis

Funkcja ConvUnitDiff() wylicza różnicę pomiędzy dwiema wartościami. AValue1 oraz AType1 są odpowiednio wartością wyjściową oraz jej jednostką, zaś AValue2 jest wartością, którą odejmujemy od parametru wyjściowego. AType2 jest aktualną jednostką, w której wyrażamy AValue2. AResultType jest jednostką, w której chcemy otrzymać wynik operacji odejmowania dwóch wartości.

Przykład

Kolejny przykład manipulacji przedziałami czasu.

```

procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  Edit2.Text := FloatToStr(ConvUnitDiff(Abs(StrToFloat(Edit1.Text)),
    TConvType(ListBox1.ItemIndex +119),
    SpinEdit1.Value,
    TConvFamily(ListBox2.ItemIndex +119),
    TConvFamily(ListBox2.ItemIndex +119)));
end;

```

Patrz również

ConvUnitAdd(), ConvUnitDec(), ConvUnitInc(), ConvUnitWithinNext(),
ConvUnitWithinPrevious()

ConvUnitInc() — funkcja

Składnia

```
function ConvUnitInc(const AValue: Double; const AType: TConvType;  
                    const AAmount: Double, const AAmountType:  
                    TConvType): Double; overload;  
  
function ConvUnitInc(const AValue: Double; const AType, AAmountType:  
                    TConvType): Double; overload;
```

Opis

Funkcja ConvUnitInc() zwiększa wartość AValue wyrażoną w jednostkach AType o zadaną liczbę. AAmount jest wartością, którą odejmujemy od wartości wyjściowej. W przypadku, gdy nie została ona ustalona domyślnie, przyjmuje się 1. AAmountType jest jednostką, w której wyrażany jest parametr AAmount. Funkcja zwraca rezultat wyrażony w jednostkach wartości wyjściowej.

Przykład

Patrz przykład do opisu funkcji ConvUnitDec().

Patrz również

ConvUnitAdd(), ConvUnitDec(), ConvUnitDiff()

ConvUnitSameValue() — funkcja

Składnia

```
function ConvUnitSameValue(const AValue1: Double; const AType1:  
                           TConvType; const AValue2: Double; const  
                           AType2: TConvType): Boolean;
```

Opis

Funkcja służy do sprawdzania zamienności używania dwóch różnych wartości.

Dwie porównywane wartości AValue1 oraz AValue2 mogą być wyrażone poprzez różne jednostki, niemniej jednak muszą należeć do tego samego układu jednostek. AType1 oraz AType2 są odpowiednio jednostkami, poprzez które wyrażone są wartości AValue1 oraz AValue2. Funkcja ConvUnitSameValue() zwraca prawdę, jeżeli dwie wartości mogą być używane zamiennie, w przeciwnym wypadku zwraca fałsz.

Przykład

Poniższy fragment kodu głównego modułu projektu *Kody\Rozdzial1\ConvUnitSameValue\p_ConvUnitSameValue.dpr* sprawdza, czy wybrane wartości określające objętości ciał są sobie równoważne. Testując algorytm łatwo przekonamy się, iż np. w odniesieniu do ciał płynnych 4 kwarty amerykańskie są równoważne jednemu amerykańskiemu galonowi (1 galon U.S. = 4 kwarty U.S.), oraz że jedna połowa ćwiartki kwarty (fl gi) równa jest 0,125 części kwarty (fl gi). Nie będzie to oczywiście prawdą, jeżeli zechcemy zamiennie użyć np. 1 galonu U.S. (fl gal) oraz 4 kwart U.S. (dry qt) w odniesieniu do ciał sypkich.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.ItemIndex := 0;
  ListBox1.Items[0] := 'FluidGallons'; //73
  ListBox1.Items[1] := 'FluidQuarts'; //74
  ListBox1.Items[2] := 'FluidPints'; //75
  ListBox1.Items[3] := 'FluidCups'; //76
  ListBox1.Items[4] := 'FluidGills'; //77
  ListBox1.Items[5] := 'FluidOunces'; //78
  ListBox1.Items[6] := 'FluidTablespoons'; //79
  ListBox1.Items[7] := 'FluidTeaspoons'; //80
  ListBox1.Items[8] := 'DryGallons'; //81
  ListBox1.Items[9] := 'DryQuarts'; //82
  ListBox1.Items[10] := 'DryPints'; //83
  ListBox1.Items[11] := 'DryPecks'; //84
  ListBox1.Items[12] := 'DryBuckets'; //85
  ListBox1.Items[13] := 'DryBushels'; //86

  ListBox2.ItemIndex := 0;
  ListBox2.Items[0] := 'FluidGallons'; //73
  ListBox2.Items[1] := 'FluidQuarts'; //74
  ListBox2.Items[2] := 'FluidPints'; //75
  ListBox2.Items[3] := 'FluidCups'; //76
  ListBox2.Items[4] := 'FluidGills'; //77
  ListBox2.Items[5] := 'FluidOunces'; //78
  ListBox2.Items[6] := 'FluidTablespoons'; //79
  ListBox2.Items[7] := 'FluidTeaspoons'; //80
  ListBox2.Items[8] := 'DryGallons'; //81
  ListBox2.Items[9] := 'DryQuarts'; //82
  ListBox2.Items[10] := 'DryPints'; //83
  ListBox2.Items[11] := 'DryPecks'; //84
  ListBox2.Items[12] := 'DryBuckets'; //85
  ListBox2.Items[13] := 'DryBushels'; //86

end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
begin
  if(ConvUnitSameValue(Abs(StrToFloat(Edit1.Text)),
    TConvType(ListBox1.ItemIndex +73),
    Abs(StrToFloat(Edit2.Text)),
    TConvFamily(ListBox2.ItemIndex +73))) then
    ShowMessage(Format('%s %s %s',
      [ListBox1.Items[ListBox1.ItemIndex],
        ' można używać zamiennie z: ',
        ListBox2.Items[ListBox2.ItemIndex]]))

```

```

else
  ShowMessage(Format('%s %s %s',
    [ListBox1.Items[ListBox1.ItemIndex],
      ' nie można używać zamiennie z: ',
      ListBox2.Items[ListBox2.ItemIndex]]))
end;
//-----

```

Patrz również

ConvUnitCompareValue()

ConvUnitToStr() — funkcja

Składnia

```

function ConvUnitToStr(const AValue: Double; const AType:
  TConvType ): string;

```

Opis

Funkcja ConvUnitToStr() formatuje wartość AValue wraz z jej jednostką AType na odpowiedni łańcuch znaków.

Przykład

```

procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.Text := ConvUnitToStr(9.9251, duAstronomicalUnits);
end;

```

Patrz również

RegisterConversionType(), StrToConvUnit()

ConvUnitWithinNext() — funkcja

Składnia

```

function ConvUnitWithinNext(const AValue, ATest: Double; const AType:
  TConvType; const AAmount: Double; const
  AAmountType: TConvType): Boolean;

```

Opis

Funkcja ConvUnitWithinNext() określa, czy wartość ATest przekracza wartość AValue o więcej niż to określono parametrem AAmount. AType jest jednostką dla obu wartości AValue oraz ATest. AAmountType jest jednostką, w której wyrażony jest parametr AAmount. Jednostki, w których wyrażono AAmountType oraz ATest, nie muszą być zgodne, jednak muszą należeć do tego samego układu jednostek. ConvUnitWithinNext() zwraca prawdę (True), jeżeli wartość ATest równa jest wartości AValue lub przekracza ją o wartość AAmount.

Przykład

Przykład będący fragmentem głównego modułu projektu *Kody\Rozdzial1\ConvUnitWithinNext\p_ConvUnitWithinNext.dpr* obrazuje określanie wzajemnych relacji wielkości stosowanych przy pomiarach odległości astronomicznych.

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  ListBox1.ItemIndex := 0;
  ListBox1.Items[0] := 'AstronomicalUnits'; //19
  ListBox1.Items[1] := 'LightYears'; //20
  ListBox1.Items[2] := 'Parsecs'; //21

  ListBox2.ItemIndex := 0;
  ListBox2.Items[0] := 'AstronomicalUnits'; //19
  ListBox2.Items[1] := 'LightYears'; //20
  ListBox2.Items[2] := 'Parsecs'; //21

end;
//-----
procedure TForm1.SpinEdit1Change(Sender: TObject);
begin
  if(ConvUnitWithinNext(Abs(StrToFloat(Edit1.Text)), //AValue
    Abs(StrToFloat(Edit2.Text)), //ATest
    TConvType(ListBox1.ItemIndex +19),
    SpinEdit1.Value, //AAmount
    TConvFamily(ListBox2.ItemIndex +19))) then
    ShowMessage(Format('%s %s %s %s %s',
      [ConvUnitToStr(StrToFloat(Edit2.Text),
        TConvType(ListBox1.ItemIndex +19)),
        ' jest większe niż ',
        ConvUnitToStr(StrToFloat(Edit1.Text),
          TConvType(ListBox1.ItemIndex +19)),
        ' o około ', ConvUnitToStr(SpinEdit1.Value,
          TConvType(ListBox2.ItemIndex +19))]);
end;
//-----

```

Patrz również

ConvUnitDiff(), ConvUnitWithinPrevious()

ConvUnitWithinPrevious() — funkcja**Składnia**

```

function ConvUnitWithinPrevious(const AValue, ATest: Double; const
  AType: TConvType; const AAmount:
  Double; const AAmountType:
  TConvType): Boolean;

```

Opis

Funkcja `ConvUnitWithinPrevious()` określa, czy wartość `ATest` jest mniejsza od wartości `AValue` o więcej niż to określono parametrem `AAmount`. `AType` jest jednostką, w której wyrażono obie wartości `AValue` oraz `ATest`. `AAmountType` jest jednostką parametru `AAmount`.

Opis

Funkcja `DescriptionToConvType()` zwraca identyfikator określający rodzaj jednostki, która musi należeć do zbioru `AFamily` identyfikującego odpowiedni układ jednostek. `ADescription` jest nazwą pojedynczej jednostki należącej do zbioru `AFamily`. Na przykład jednostka identyfikowana przez literał `auAcres` reprezentowana jest przez łańcuch `'Acres'`. `AType` zwraca identyfikator typu jednostek mogących podlegać operacji przeliczania przy wykorzystaniu odpowiednich czynników przeliczeniowych. Funkcja `DescriptionToConvType()` zwraca prawdę, jeżeli wybrana jednostka należy do właściwego układu jednostek.

Przykład

Poniższy przykład w postaci kompletnego modułu projektu `Kody\Rozdzial1\DescriptionToConvType\p_DescriptionToConvType.dpr` obrazuje kolejny sposób przeliczania wielkości będących odległościami wyrażanymi w skali astronomicznej. Dzięki zastosowaniu w poniższym algorytmie funkcji `DescriptionToConvType()` oraz `DescriptionToConvFamily()` zrezygnowano z indeksowania kolejnych jednostek.

```
unit Unit_DescriptionToConvType;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, StdCtrls, ConvUtils, StdConvs;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    ListBox1: TListBox;
    ComboBox1: TComboBox;
    Button1: TButton;
    ListBox2: TListBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation

{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Text := 'Wielkości';
  ComboBox1.Items[0] := 'Distance';
  //ComboBox1.Items[1] := 'Volume';
  //ComboBox1.Items[2] := 'Temperature';

  ListBox1.Items[0] := 'AstronomicalUnits';
  ListBox1.Items[1] := 'LightYears';
  ListBox1.Items[2] := 'Parsecs';
```

```

    ListBox2.Items[0] := 'AstronomicalUnits';
    ListBox2.Items[1] := 'LightYears';
    ListBox2.Items[2] := 'Parsecs';
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
var
    newValue: Double;
    AFamily : TConvFamily;
    FromType, ToType: TConvType;
    ADescription: string;
begin
    ADescription := ComboBox1.Items[ComboBox1.ItemIndex];
    if DescriptionToConvFamily(ADescription, AFamily) then

        DescriptionToConvType(AFamily,
                               ListBox1.Items[ListBox1.ItemIndex],
                               FromType);
        DescriptionToConvType(AFamily,
                               ListBox2.Items[ListBox2.ItemIndex],
                               ToType);
        newValue := Convert(Abs(StrToFloat(Edit1.Text)),
                            FromType, ToType);
        ShowMessage(Format('%g %s %s %g %s',
                            [Abs(StrToFloat(Edit1.Text)),
                             ConvTypeToDescription(FromType), 'równoważne jest ',
                             newValue, ConvTypeToDescription(ToType)]))

end;
//-----
end.

```

Patrz również

ConvTypeToDescription(), DescriptionToConvFamily(), RegisterConversionType(), StrToConvUnit()

EConversionError — wyjątek

Opis

EConversionError jest klasą wyjątków przechwytyjących błędy powstałe podczas przeliczania jednostek pomocą funkcji Convert() lub jej pochodnych. Klasa EConversionError formalnie zdefiniowana w module SysUtils dziedziczy po klasie Exception. Błędy przeliczania powstają najczęściej z czterech powodów:

- ◆ Układ jednostek nie jest zarejestrowany.
- ◆ Jeden z czynników przeliczeniowych wchodzących w skład układu jednostek nie jest zdefiniowany.
- ◆ Wartość czynnika przeliczeniowego nie jest liczbą (w sensie numerycznym).
- ◆ Przeliczanie wartości następuje niezgodnie z typami odpowiednich jednostek (np. przeliczanie długości na objętość czy prędkości na temperaturę).

FahrenheitToCelsius() — funkcja

Składnia

```
function FahrenheitToCelsius(const AValue: Double): Double;
```

Opis

Funkcja przelicza temperaturę wyrażoną w stopniach Fahrenheita (°F) określoną parametrem AValue na temperaturę wyrażoną w skali Celsjusza (°C).

Patrz również

CelsiusToFahrenheit(), Convert()

GetConvFamilies() — procedura

Składnia

```
procedure GetConvFamilies(out AFamilies: TConvFamilyArray);
```

Opis

Procedura GetConvFamilies() zwraca dynamiczną tablicę AFamilies identyfikatorów aktualnie zarejestrowanych układów jednostek.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AFamilies: TConvFamilyArray;
  i: Integer;
begin
  GetConvFamilies(AFamilies);
  for i := 0 to Length(AFamilies) - 1 do
    ListBox1.Items.Add(ConvFamilyToDescription(AFamilies[i]));
end;
```

Patrz również

GetConvTypes(), RegisterConversionFamily(), TConvFamilyArray

GetConvTypes() — procedura

Składnia

```
procedure GetConvTypes(const AFamily: TConvFamily; out ATypes:
  TConvTypeArray);
```

Opis

Procedura `GetConvTypes()` wypełnia tablicę dynamiczną `ATypes` wartościami typu `TConvType` zarejestrowanymi w odpowiednim układzie jednostek. Wszystkie wielkości pomiarowe bazują na jednym z zarejestrowanych typów jednostek. Parametr `AFamily` identyfikuje zbiór określonych jednostek właściwych danej wielkości fizycznej. `ATypes` jest tablicą wypełnianą identyfikatorami typów jednostek podlegających przeliczaniu.

Przykład

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ATypes: TConvTypeArray;
  AFamily: TConvFamily;
  ADescription: string;
  i: Integer;
begin
  ADescription := ListBox1.Items[ListBox1.ItemIndex];
  if DescriptionToConvFamily(ADescription, AFamily) then
    begin
      GetConvTypes(AFamily, ATypes);
      for i := 0 to Length(ATypes) - 1 do
        ListBox1.Items.Add(ConvTypeToDescription(ATypes[i]));
      ListBox2.Items := ListBox1.Items;
    end;
end;
```

Patrz również

`GetConvFamilies()`, `RegisterConversionType()`

RaiseConversionError() — procedura**Składnia**

```
procedure RaiseConversionError(const AText: string; const AArgs:
    array of const); overload;
procedure RaiseConversionError(const AText: string); overload;
```

Opis

Wywołanie procedury `RaiseConversionError()` skutkuje przechwyceniem wyjątku `EConversionError`. W większości wypadków aplikacje nie potrzebują jawnego wywołania procedury przechwytyjącej ewentualne wyjątki, gdyż problemem tym zajmuje się funkcja `Convert()`. Procedura `RaiseConversionError()` jest pomocna w przechwytywaniu wyjątków podczas przeliczania nowo zarejestrowanych za pomocą funkcji `RegisterConversionType()` jednostek w ramach określonego ich zbioru. Parametr `AText` jest komunikatem błędu. `AArgs` jest listą argumentów komunikatu błędu.

Patrz również

`Convert()`, `EConversionError`, `RegisterConversionType()`

RegisterConversionFamily() — funkcja

Składnia

```
function RegisterConversionFamily(const ADescription: string):
    TConvFamily;
```

Opis

Wywołanie funkcji `RegisterConversionFamily()` powoduje zarejestrowanie nowego układu jednostek oraz zwrócenie jego identyfikatora. Zarejestrowany układ musi tworzyć pewien zbiór czynników przeliczeniowych określających typ używanych w jego obrębie jednostek. Oryginalną nazwę nowego układu jednostek reprezentuje łańcuch `ADescription`.

Odwołanie do funkcji `RegisterConversionFamily()` powinno następować w sekcji inicjalizacji modułu.

Przykład

Patrz opis modułu *MyConvs.pas* w podrozdziale *Samodzielne definiowane układów jednostek*.

Patrz również

`Convert()`, `RegisterConversionType()`, `UnregisterConversionFamily()`

RegisterConversionType() — funkcja

Składnia

```
function RegisterConversionType(const AFamily: TConvFamily; const
    ADescription: string, const AFactor:
    Double): TConvType; overload;
```

```
function RegisterConversionType(const AFamily: TConvFamily; const
    ADescription: string, const
    AToCommonProc, AFromCommonProc:
    TConversionProc): TConvType; overload;
```

```
function RegisterConversionType(AConvTypeInfo: TConvTypeInfo; out
    AType: TConvType): Boolean; overload;
```

Opis

Funkcja `RegisterConversionType()` dodaje nową jednostkę do zarejestrowanego za pomocą funkcji `RegisterConversionFamily()` układu jednostek. Funkcja `Convert` może samodzielnie dokonać wzajemnej konwersji wartości dwóch wybranych nowo zarejestrowanych jednostek. Parametr `AFamily` jest identyfikatorem wybranego układu jednostek. Może identyfikować jeden z predefiniowanych układów udostępnianych poprzez moduł `StdConvs` (patrz tabele 1.4 – 1.9) lub identyfikuje nowo zarejestrowany zbiór jednostek.

ADescription jest łańcuchem opisującym nazwę jednostki. AFactor jest wartością liczbową czynnika przeliczeniowego będącego podstawą dalszego przeliczania jednostki podstawowej wybranego układu jednostek. Przykładowo, liczbową wartością czynnika przeliczeniowego duKilometers jest 1 000, ponieważ jednostką podstawową układu jednostek długości i odległości cbDistance jest 1 metr.

Parametry AToCommonProc oraz AFromCommonProc typu TConversionProc są funkcjami przeliczającymi wybrane wielkości pomiędzy jednostką podstawową układu a jednostkami pochodnymi. Funkcji tych należy używać wówczas, gdy zachodzi potrzeba przeliczania odpowiedniej wielkości poprzez szereg różnych czynników przeliczeniowych.

Definicje funkcji AToCommonProc oraz AFromCommonProc umieszcza się w sekcji implementacji modułu. Wywołanie funkcji RegisterConversionType() następuje w sekcji inicjalizacji modułu.

Parametr AConvTypeInfo jest egzemplarzem obiektu TConvTypeInfo dziedziczącym po TObject.

Funkcja RegisterConversionType() zwraca wartość True, jeżeli nowe typy jednostek wraz z ich czynnikami przeliczeniowymi zostały prawidłowo zarejestrowane. Wartość False zwracana jest w przypadku ich błędnej rejestracji.

Przykład

Patrz opis modułu *MyConvs.pas* w podrozdziale *Samodzielne definiowanie układów jednostek*.

Patrz również

Convert(), RegisterConversionFamily(), TConversionProc(), TConvTypeInfo(), UnregisterConversionType()

StrToConvUnit() — funkcja

Składnia

```
function StrToConvUnit(AText: string; out AType: TConvType): Double;
```

Opis

StrToConvUnit() działa w sposób odwrotny do ConvUnitToStr(). AText jest łańcuchem znaków zawierającym liczbową wartość czynnika przeliczeniowego oraz nazwę (nie literał) jednostki. Poprzez parametr AType funkcja zwraca identyfikator jednostki należącej do odpowiedniego układu jednostek. Jeżeli AText nie reprezentuje poprawnie skonstruowanego łańcucha, funkcja StrToConvUnit() generuje wyjątek Econversion Error.

Przykład

W odpowiedzi na wywołanie funkcji obsługi zdarzenia Button1Click() wyświetlony zostanie odpowiedni komunikat pokazujący łańcuch 3,35 CubicKilometers oraz liczbowy identyfikator (54) literału vuCubicKilometers (por. tabela 1.9).

```

procedure TForm1.Button1Click(Sender: TObject);
var AType: TConvType;
begin
  ShowMessage(Format('%g %s %s',
    [StrToConvUnit('3,35 CubicKilometers', AType),
    ConvTypeToDescription(AType), IntToStr(AType)]));
end;

```

Patrz również

ConvUnitToStr(), DescriptionToConvType(), RegisterConversionType(), TryStrToConvUnit()

TryConvTypeToFamily() — funkcja

Składnia

```

function TryConvTypeToFamily(const AType: TConvType); out AFamily:
    TConvFamily) : Boolean; overload;
function TryConvTypeToFamily(const AFrom, ATo: TConvType;
    out AFamily: TConvFamily) : Boolean;
    overload;

```

Opis

Funkcja TryConvTypeToFamily() sprawdza zgodność identyfikatora AFamily układu jednostek z identyfikatorem podanej jednostki. Odpowiednia jednostka identyfikowana przez AType i wchodząca w skład układu jednostek musi zostać wcześniej zarejestrowana za pomocą funkcji RegisterConversionType(). W drugiej postaci funkcji parametry AFrom oraz ATo określają dwie różne jednostki przynależne do danego układu jednostek.

TryConvTypeToFamily() zwraca prawdę, jeżeli AType reprezentuje zarejestrowaną jednostkę lub AFrom oraz ATo są jednostkami zarejestrowanymi w tym samym układzie jednostek.

Przykład

W odpowiedzi na wywołanie funkcji obsługi zdarzenia Button1Click() wyświetlona zostanie nazwa wybranej jednostki wraz z jej liczbowym identyfikatorem oraz nazwa odpowiedniego układu jednostek również z liczbowym identyfikatorem.

```

procedure TForm1.Button1Click(Sender: TObject);
var AType: TConvType;
    AFamily: TConvFamily;
begin
  AType := vuUKGallons;
  TryConvTypeToFamily(AType, AFamily);
  ShowMessage(Format('%s %s %s %s %s %s %s %s',
    [ConvTypeToDescription(AType), '(' , IntToStr(AType), ')',
    ConvFamilyToDescription(AFamily),
    '(' , IntToStr(AFamily), ')']));
end;

```

Patrz również

CompatibleConversionType(), ConvFamilyToDescription(), ConvTypeToDescription(), ConvTypeToFamily(), DescriptionToConvFamily(), RegisterConversionType()

TryStrToConvUnit() — funkcja

Składnia

```
function TryStrToConvUnit(AText: string; out AValue: Double; out
                          AType: TConvType): Boolean;
```

Opis

Funkcja TryStrToConvUnit() działa odwrotnie do ConvUnitToStr(). AText jest łańcuchem znaków reprezentującym liczbową wartość AValue danej wielkości oraz nazwę przyporządkowanej jednostki. AType zwraca identyfikator typu jednostki. Wykonanie funkcji zostanie zakończone pomyślnie, jeżeli łańcuch AText zostanie prawidłowo przekonwertowany na odpowiednie wielkości.

Przykład

```
procedure TForm1.Button1Click(Sender: TObject);
var AType: TConvType;
    AValue: Double;
begin
  TryStrToConvUnit('3,33 Parsecs', AValue, AType);
  ShowMessage(Format('%g %s',
                    [AValue, ConvTypeToDescription(AType)]));
end;
```

Patrz również

ConvUnitToStr(), DescriptionToConvType(), RegisterConversionType(), StrToConvUnit()

UnregisterConversionFamily() — procedura

Składnia

```
procedure UnregisterConversionFamily(const AFamily: TConvFamily);
```

Opis

Użycie w programie procedury UnregisterConversionFamily() powoduje wyrejestrowanie układu jednostek zarejestrowanych uprzednio poprzez RegisterConversionFamily(). Nazwa układu jednostek reprezentowana jest poprzez parametr AFamily.

Funkcja RegisterConversionFamily() zwraca do systemu identyfikator układu jednostek będący w istocie niewielką 16-bitową liczbą. W momencie zakończenia pracy z określonym zbiorem danych identyfikator ten powinien zostać zwolniony. Zwolnienie identyfikatora układu jednostek powoduje automatyczne zwolnienie wszystkich identyfikatorów jednostek wchodzących w jego skład.

Użycie procedury `UnregisterConversionFamily()` powinno nastąpić w sekcji finalizacji modułu.

Wywołanie funkcji `Convert()` z argumentem w postaci wyrejestrowanej jednostki skutkuje pojawieniem się wyjątku `EConversionError`.

Przykład

Patrz opis modułu *MyConvs.pas* w podrozdziale *Samodzielne definiowanie układów jednostek*.

Patrz również

`EConversionError`, `GetConvFamilies()`, `RegisterConversionFamily()`, `UnregisterConversionType()`

UnregisterConversionType() — procedura

Składnia

```
procedure UnregisterConversionType(const AType: TConvType);
```

Opis

Użycie w programie procedury `UnregisterConversionType()` powoduje wyrejestrowanie określonej jednostki (wraz z jej czynnikiem przeliczeniowym) zarejestrowanej uprzednio poprzez funkcję `RegisterConversionType()`. Nazwa jednostki reprezentowana jest poprzez parametr `AType`. Funkcja `RegisterConversionType()` zwraca do systemu identyfikator jednostki będący również niewielką 16-bitową liczbą. W momencie zakończenia pracy z określonym zbiorem danych identyfikator ten powinien zostać zwolniony. Zwolnienie identyfikatora określonej jednostki nie powoduje automatycznego zwolnienia identyfikatora odpowiedniego układu jednostek.

Wywołanie procedury `UnregisterConversionType()` powinno nastąpić w sekcji finalizacji modułu.

Wywołanie funkcji `Convert()` z argumentem w postaci wyrejestrowanej jednostki skutkuje pojawieniem się wyjątku `EConversionError`.

Przykład

Patrz opis modułu *MyConvs.pas* w podrozdziale *Samodzielne definiowanie układów jednostek*.

Patrz również

`EConversionError`, `GetConvFamilies()`, `RegisterConversionType()`, `UnregisterConversionFamily()`

TConvFamily — typ

Składnia

```
type
  TConvFamily = type Word;
```

Opis

16-bitowe dane typu TConvFamily identyfikują określony układ jednostek. Układ taki jest zbiorem jednostek reprezentowanych przez wielkości typu TConvType oraz odpowiednich czynników przeliczeniowych. Kiedy dwie wielkości TConvType zostaną zarejestrowane w jednym układzie jednostek funkcja Convert() może dokonać wzajemnego przeliczenia ich wartości. Wartości TConvFamily otrzymywane są jako rezultat wykonania funkcji RegisterConversionFamily().

TConvFamilyArray — typ

Składnia

```
type
  TConvFamilyArray = array of TConvFamily;
```

Opis

TConvFamilyArray jest typem tablicy dynamicznej, której elementy typu TConvFamily reprezentują identyfikatory odpowiednich układów jednostek. Każdy element tablicy zawiera z kolei identyfikator odpowiedniej jednostki typu TConvType wchodzącej w skład określonego układu jednostek.

TConversionProc — typ

Składnia

```
type
  TConversionProc = function(const AValue: Double): Double;
```

Opis

W momencie zarejestrowania nowej jednostki należącej do określonego układu jednostek należy określić wzajemne relacje pomiędzy jednostką podstawową a jednostkami pochodnymi. Funkcje typu TConversionProc pozwalają określić wzajemne relacje podczas przeliczania dwóch lub więcej wartości mogących być wyrażonymi poprzez dany układ jednostek. Parametr AValue reprezentuje nową wartość czynnika przeliczeniowego (należącego do bieżącego układu jednostek), otrzymanego na podstawie wartości jednostki podstawowej układu lub którejś z jednostek pochodnych.

TConvType — typ

Składnia

```
type
  TConvType = type Word;
```

Opis

16-bitowe dane typu TConvType reprezentują poszczególne jednostki wchodzące w skład układu jednostek.

TConvTypeInfo — klasa

Składnia

```
TConvTypeInfo = class(TObject)
private
  FDescription: string;
  FConvFamily: TConvFamily;
  FConvType: TConvType;
public
  constructor Create(const AConvFamily: TConvFamily; const
    ADescription: string);
  function ToCommon(const AValue: Double): Double; virtual;
    abstract;
  function FromCommon(const AValue: Double): Double; virtual;
    abstract;
  property ConvFamily: TConvFamily read FConvFamily;
  property ConvType: TConvType read FConvType;
  property Description: string read FDescription;
end;
```

Opis

Klasa TConvTypeInfo dziedzicząc bezpośrednio po TObject udostępnia dodatkowo szereg metod i właściwości.

Właściwości

```
ConvFamily
property ConvFamily: TConvFamily;
```

Właściwość ConvFamily jest identyfikatorem układu jednostek. W momencie wywołania funkcja RegisterConversionType() tworzy automatycznie egzemplarz klasy TConvTypeInfo przypisując określone jednostki do wybranego układu jednostek.

```
ConvType
property ConvType: TConvType;
```

Właściwość ConvType identyfikuje określony typ jednostki. Funkcja RegisterConversionType() przypisuje wybranej wielkości unikalny identyfikator typu TConvType.

```
Description
property Description: string;
```

Właściwość `Description` określa nazwę jednostki wchodzącej w skład wybranego układu jednostek. Przykładowo jednostka identyfikowana poprzez literał `tuFortnights` reprezentowana jest przez właściwość łańcuchową `'Fortnights'`.

Metody

```
Create
constructor Create(const AConvFamily: TConvFamily; const
                  ADescription: string);
```

Konstruktor `Create()` tworzy nowy egzemplarz klasy `TConvTypeInfo` przydzielając parametr `AConvFamily` do właściwości `ConvFamily` oraz parametr `ADescription` do właściwości `Description`. Jeżeli `AConvFamily` nie identyfikuje określonego układu jednostek (nie jest on zarejestrowany), konstruktor generuje wyjątek `EConversionError`.

```
FromCommon
function FromCommon(const AValue: Double): Double; virtual; abstract;
```

Metoda `FromCommon()` dokonuje przeliczenia (przemnożenia) wartości reprezentowanej przez określony typ jednostki pochodnej na inną jednostkę układu. Funkcja `Convert()` dokonuje niejawnego wywołania metody `ToCommon()` dla wybranego egzemplarza wyjściowej wartości, następnie wykonywane jest wywołanie metody `FromCommon()` dokonującej zasadniczego przeliczania danej wartości.

W klasie `TConvTypeInfo` `FromCommon()` deklarowana jest jako metoda abstrakcyjna. Klasy potomne z reguły używają `FromCommon()` do przeliczania wartości wyrażanych w jednostkach podstawowych układu na wartości wyrażane w jednostkach pochodnych.

```
ToCommon
function ToCommon(const AValue: Double): Double; virtual; abstract;
```

Metoda `ToCommon()` dokonuje przeliczenia (zmiennopozycyjnego dzielenia) wartości reprezentowanej przez określony typ jednostki na wartość wyrażoną w jednostce podstawowej układu. W klasie `TConvTypeInfo` `ToCommon()` deklarowana jest jako metoda abstrakcyjna. Klasy potomne z reguły używają `ToCommon()` do przeliczania wartości wyrażanych w jednostkach pochodnych układu na wartości wyrażane w jednostce podstawowej.

TConvTypeFactor — klasa

Składnia

```
TConvTypeFactor = class(TConvTypeInfo)
private
    FFactor: Double;
protected
    property Factor: Double read FFactor;
public
    constructor Create(const AConvFamily: TConvFamily; const
                    ADescription: string; const AFactor: Double);
    function ToCommon(const AValue: Double): Double; override;
    function FromCommon(const AValue: Double): Double; override;
end;
```

Opis

Klasa `TConvTypeFactor` dziedzicząc po `TConvTypeInfo` opisuje wartość pojedynczego zarejestrowanego typu czynnika przeliczeniowego należącego do układu jednostek.

Wywołanie globalnej funkcji `RegisterConversionType()` powoduje automatyczne utworzenie egzemplarza klasy `TConvTypeFactor` przechowującego wielokrotność (niekoniecznie całkowitą) jednostki podstawowej układu. `TConvTypeFactor` udostępnia metody: `Create()`, `FromCommon()`, `ToCommon()`, właściwości: `ConvFamily`, `ConvType`, `Description` oraz dodatkowo jeszcze jedną właściwość.

Właściwości

`Factor`
property `Factor`: `double`;

`Factor` jest w istocie wartością liczbową czynnika przeliczeniowego definiującego relacje pomiędzy jednostką podstawową układu a jej jednostkami pochodnymi. Metoda `ToCommon()` dokonuje operacji dzielenia (przez wartość czynnika `Factor`) wartości danej wielkości reprezentowanej przez określony typ jednostki. Metoda `FromCommon()` zwielokrotnia (przemnaża o czynnik `Factor`) wartość danej wielkości reprezentowanej przez określony typ jednostki (najczęściej podstawowej) na inną wartość wyrażoną w jednostce pochodnej układu jednostek.

Przykłady

Przykłady zastosowań opisanych klas `TConvTypeFactor` oraz `TConvTypeInfo` można znaleźć w katalogu instalacyjnym Delphi 6 `Demos\ConvertIt\`.

TConvTypeProcs — klasa

Składnia

```
TConvTypeProcs = class(TConvTypeInfo)
private
  FToCommonProc: TConversionProc;
  FFromCommonProc: TConversionProc;
public
  constructor Create(const AConvFamily: TConvFamily; const
                    ADescription: string; const AToCommonProc,
                    AFromCommonProc: TConversionProc);
  function ToCommon(const AValue: Double): Double; override;
  function FromCommon(const AValue: Double): Double; override;
end;
```

Opis

Klasa `TConvTypeProcs` dziedzicząc z kolei po `TConvTypeInfo` opisuje pojedynczą zarejestrowaną jednostkę. Globalna funkcja `RegisterConversionType()` automatycznie tworzy nowy egzemplarz klasy `TConvTypeProcs` (przez co odwołanie do niej w sposób jawny nie jest konieczne) w momencie, gdy zostanie zarejestrowana nowa jednostka oraz odpowiedni czynnik przeliczeniowy w danym zbiorze jednostek.

TValueRelationship — typ

Składnia

```
type
  TValueRelationship = -1 ... 1;
```

Opis

Typ TValueRelationship należy do modułu Types i jest globalnym typem danych otrzymywanych jako rezultat porównania dwóch dowolnych wielkości.

Tabela 1.10. Wartości zwracane przez dane typu TValueRelationship

Literał	Stała dziesiętna	Opis
LessThanValue	-1	Wartość występująca po lewej stronie wyrażenia jest mniejsza od wartości występującej po jego prawej stronie
EqualsValue	0	Obie porównywane wartości są równe
GreaterThanValue	1	Pierwsza wartość jest większa od wartości drugiej

Samodzielne definiowanie układów jednostek

Przedstawione w poprzednim podrozdziale wiadomości na temat posługiwania się predefiniowanymi układami jednostek oraz różnymi funkcjami przeliczającymi mogą okazać się niewystarczające dla osób pragnących wykorzystywać komputer jako narzędzie pomocne przy wykonywaniu bardziej skomplikowanych obliczeń. Z tego powodu powinniśmy również posiadać pewną wiedzę na temat samodzielnego definiowania i praktycznego wykorzystania różnorodnych układów jednostek.

W tym celu zdefiniujemy od podstaw cztery układy:

- ◆ jednostek przeliczających wielkości kątowe (cbAngle)
- ◆ jednostek mocy (cbPower)
- ◆ jednostek siły (cbForce)
- ◆ jednostek energii (cbEnergy).

W każdym z tych układów zdefiniujemy wybrane jednostki, tak jak pokazują to tabele 1.11 – 1.14. Wybiegając trochę do przodu w poniższych tabelach zamieściłem również identyfikatory liczbowe określonych jednostek (wartości poszczególnych identyfikatorów będą reprezentowane poprzez pokazane liczby, jeżeli oczywiście zostaną zarejestrowane w podanej kolejności). Liczbowe wartości wykorzystanych w niniejszym podrozdziale czynników przeliczeniowych zostały zaczerpnięte z książki Dawida Hollidaya i Roberta Resnicka *Physics Part I*, John Willey&Sons (1964), wydanie polskie PWN (1980).

Tabela 1.11. Samodzielnie zdefiniowane jednostki należące do zbioru *cbAngle* reprezentującego układ jednostek wyrażających wielkości kątowe. Jednostką podstawową w tym zbiorze jest jeden radian (1 rad.)

Literał	Liczbowy identyfikator	Opis
auDegrees	132	Stopień $1^\circ = (\pi/180)$ rad
auMinutes	133	Minuta $1' = (1/60)^\circ = (\pi/10\ 800)$ rad
auSeconds	134	Sekunda $1'' = (1/60)' = (\pi/648\ 000)$ rad
auRadians	135	Radian
auCycles	136	Obrót (cykl). 1 obrót liczbowo równy jest 2π rad = 360°

Tabela 1.12. Samodzielnie zdefiniowane jednostki należące do zbioru *cbPower* reprezentującego układ wybranych jednostek mocy. Jednostką podstawową w tym zbiorze jest jeden wat (1 W)

Literał	Liczbowy identyfikator	Opis
puHoursPower	137	Konie mechaniczne (KM)
puCaloriesPerSecond	138	Kalorie na sekundę (cal/s)
puKilowatts	139	Kilowaty (kW)
puWatts	140	Waty (W)

Tabela 1.13. Samodzielnie zdefiniowane jednostki należące do zbioru *cbForce* reprezentującego układ wybranych jednostek siły. Jednostką podstawową w tym zbiorze jest jeden Niuton (1 N)

Literał	Liczbowy identyfikator	Opis
fuDynes	141	Dyna. 1 dyna odpowiada $1E-5$ N
fuNewtons	142	Niuton (N). 1 N jest siłą, która nadaje masie 1 kg przyspieszenie równe $1m/s^2$
fuGramsForce	143	Gram siły (G)
fuKilogramsForce	144	Kilogram siły (kG). Kilogram siły odpowiada 9,806 N
fuPoundsForce	145	Funt siły (lbf). Funt siły odpowiada 4,448 222 N

Tabela 1.14. Samodzielnie zdefiniowane jednostki należące do zbioru *cbEnergy* reprezentującego układ wybranych jednostek energii. Jednostką podstawową w tym zbiorze jest jeden elektronowolt (1 eV)

Literał	Liczbowy identyfikator	Opis
eueV	146	Elektronowolt (eV). 1 eV jest energią, jaką uzyskuje elektron przyspieszany w polu elektrycznym o różnicy potencjałów 1 V (1 wolta)
euKilograms	147	Kilogram. Jest jednostką energii używaną w fizyce jądrowej. Odpowiada wielkości energii, jaka wydzieli się przy całkowitej zmianie na energię 1 kg masy, zgodnie z relatywistyczną zależnością pomiędzy masą a energią: $E=mc^2$

Zebrane w powyższych tabelach definicje zarejestrujemy w oddzielnym module, nazwijmy go *My_Convs.pas*. Wydruk 1.1 przedstawia jego kompletny kod źródłowy.

Wydruk 1.1. Kod źródłowy modułu *My_Convs.pas* rejestrującego samodzielnie zdefiniowane wybrane układy jednostek wraz z odpowiadającymi im czynnikami przeliczeniowymi

```

unit My_Convs;

interface

uses
  SysUtils, ConvUtils;

var
  //-----
  { Układ jednostek kątowych }
  cbAngles: TConvFamily;
  { Jednostką podstawową jest 1 radian }
  auDegrees: TConvType;
  auMinutes: TConvType;
  auSeconds: TConvType;
  auRadians: TConvType;
  auCycles: TConvType;

  //-----
  { Układ jednostek siły }
  cbForce: TConvFamily;
  { Jednostką podstawową jest 1 niuton }
  fuDynas: TConvType;
  fuNewtons: TConvType;
  fuGramsForce: TConvType;
  fuKilogramsForce: TConvType;
  fuPoundsForce: TConvType;

  //-----
  { Układ jednostek mocy }
  cbPower: TConvFamily;
  { Jednostką podstawową jest 1 wat }
  puHoursPower: TConvType;
  puCaloriesPerSecond: TConvType;
  puKilowatts: TConvType;
  puWatts: TConvType;

  { Układ jednostek energii }
  cbEnergy: TConvFamily;

  {Jednostką podstawową jest 1 eV (elektronowolt) }
  eueV: TConvType;
  euKilograms: TConvType;

  { łańcuchy znaków używane przez moduł }
resourcestring

  //-----
  { Układ jednostek kątowych }
  AAnglesDescription = 'Angles';

  { Wybrane jednostki należące do układu jednostek kątowych }
  ADegreesDescription = 'Degrees'; //stopnie
  AMinutesDescription = 'Minutes'; //minuty
  ASecondsDescription = 'Seconds'; //sekundy
  ARadiansDescription = 'Radians'; //radiany
  ACyclesDescription = 'Cycles'; //obroty (cykle)

```

```

//-----
{ Układ jednostek siły }
AForceDescription = 'Force';

{ Wybrane jednostki należące do układu jednostek siły}
ADynasDescription = 'Dynas'; //dyny
ANewtonsDescription = 'Newtons'; //niutony
AGramsForceDescription = 'GramsForce'; //gramy siły
AKilogramsForceDescription = 'KilogramsForce'; //kilogramy siły
APoundsForceDescription = 'PoundsForce'; //funt y siły

//-----
{ Układ jednostek mocy }
APowerDescription = 'Power';

{ Wybrane jednostki należące do układu jednostek mocy}
AHoursPowerDescription = 'HoursPower'; //konie mechaniczne
ACaloriesPerSecondDescription = 'CaloriesPerSecond'; //kalorie na
//sekundę
AKilowattsDescription = 'Kilowatts'; // kilowaty
AWattsDescription = 'Watts'; //waty

{ Układ jednostek energii }
AEnergyDescription = 'Energy';

{ Wybrane jednostki należące do układu jednostek energii}
AeVDescription = 'eV';
AKilogramsDescription = 'Kilograms';

(*
function eVoltToJoule(const AValue: Double): Double;
function JouleToAtomUnitMass(const AValue: Double): Double;
function AtomUnitMassToKilograms(const AValue: Double): Double;
function KilogramsToeV(const AValue: Double): Double;
*)

implementation

uses
    Math;

//-----
function eVoltToJoule(const AValue: Double): Double;
begin
    Result := AValue * 1.602189E-19; //przelicza eV na džule
end;

function JouleToAtomUnitMass(const AValue: Double): Double;
begin
    Result := eVoltToJoule(AValue*6.705E+9);
    //przelicza džule na a.u.m
end;

function AtomUnitMassToKilograms(const AValue: Double): Double;
begin
    Result := AValue*1.660E-27; //przelicza a.u.m na kg
end;

```

```

function KilogramsToeV(const AValue: Double): Double;
begin
  Result := AtomUnitMassToKilograms(AValue*5.610E+35);
  //przelicza kg na eV
end;

initialization
//-----
{ Układ jednostek kątowych }
cbAngles := RegisterConversionFamily(AAnglesDescription);
{Czynniki przeliczeniowe do układu jednostek kątowych –
angle units}
auDegrees := RegisterConversionType(cbAngles,
                                   ADegreesDescription, PI/180);
auMinutes := RegisterConversionType(cbAngles,
                                   AMinutesDescription, 2.909E-4);
auSeconds := RegisterConversionType(cbAngles,
                                   ASecondsDescription, 4.848E-6);
auRadians := RegisterConversionType(cbAngles,
                                   ARadiansDescription, 1);
auCycles := RegisterConversionType(cbAngles,
                                   ACyclesDescription, 2*PI);

//-----
{ Układ jednostek mocy }
cbPower := RegisterConversionFamily(APowerDescription);

{Czynniki przeliczeniowe do układu jednostek mocy - power units}
puHoursPower := RegisterConversionType(cbPower,
                                       AHoursPowerDescription, 745.7);
puCaloriesPerSecond := RegisterConversionType(cbPower,
                                               ACaloriesPersecondDescription, 4.186);
puKilowatts := RegisterConversionType(cbPower,
                                       AKilowattsDescription, 1000);
puWatts := RegisterConversionType(cbPower,
                                   AWattsDescription, 1);

//-----
{ Układ jednostek siły }
cbForce := RegisterConversionFamily(AForceDescription);

{Czynniki przeliczeniowe do układu jednostek siły - force units}
fuDynas := RegisterConversionType(cbForce,
                                   ADynasDescription, 1E-5);
fuNewtons := RegisterConversionType(cbForce,
                                    ANewtonsDescription, 1);
fuGramsForce := RegisterConversionType(cbForce,
                                        AGramsForceDescription, 9.80665E-3);
fuKilogramsForce := RegisterConversionType(cbForce,
                                           AKilogramsForceDescription, 9.80665);

fuPoundsForce := RegisterConversionType(cbForce,
                                        APoundsForceDescription, 4.448222);

{ Układ jednostek energii }
cbEnergy := RegisterConversionFamily(AEnergyDescription);
{Czynniki przeliczeniowe do układu jednostek energii –

```

```

energy units}
eueV := RegisterConversionType(cbEnergy, AKilogramsDescription,
                               KilogramsToeV, AtomUnitMassToKilograms);
eukilograms := RegisterConversionType(cbEnergy, AeVDescription,
                                      JouleToAtomUnitMass, eVoltToJoule);

finalization

{ Wyrejestrowanie wszystkich wcześniej zarejestrowanych
  w module układów jednostek }
UnregisterConversionFamily(cbAngles);
UnregisterConversionFamily(cbPower);
UnregisterConversionFamily(cbForce);
UnregisterConversionFamily(cbEnergy);
end.

```

Przedstawiony algorytm wykorzystuje dobrze nam znane z poprzedniego podrozdziału funkcje. Powinniśmy jednak zauważyć, iż w jednym jego fragmencie zastosowaliśmy nieco odmienny sposób przeliczania jednostek układu energii. W części modułu, gdzie rejestrowane są czynniki przeliczeniowe do układu jednostek energii, skorzystaliśmy z czterech funkcji: `eVoltToJoul()` — przeliczającej elektronowolty na dżule, `JoulToAtomUnitMass()` — przeliczającej dżule na atomowe jednostki masy, `AtomUnitMassToKilograms()` — przeliczającej atomowe jednostki masy na kilogramy oraz `KilogramsToeV()` — przeliczającej kilogramy na elektronowolty. Ciała tych funkcji zostały umieszczone w sekcji implementacji modułu. Prezentowany sposób przeliczania jednostek układu energii może wydawać się nieco zawiły, jednak dobrze oddaje ideę przeliczania różnych wartości metodą „nie wprost”. Przeliczenia elektronowoltów na kilogramy dokonaliśmy etapami. Najpierw elektronowolty przeliczane są na dżule, następnie otrzymany wynik przeliczany jest na atomowe jednostki masy, które z kolei przeliczane są na kilogramy, by w efekcie wynik otrzymać powtórnie w elektronowoltach. Zastosowanie tego rodzaju techniki pozwala nam nieustannie kontrolować sposób obliczania pewnych wartości. Jeżeli algorytm będzie skonstruowany prawidłowo, otrzymany wynik powinien być identyczny z otrzymanym na podstawie prostego przeliczenia elektronowoltów na kilogramy. Podobne zabiegi mają zastosowanie nie tylko w nauce i technice (gdzie niekiedy należy bardzo dokładnie sprawdzić, czy np. masa 1 kg pewnej substancji wywoła pożądaną efekt), stosuje się je również w algorytmach „obsługujących” różnego rodzaju operacje finansowe, gdzie bardzo często zachodzi potrzeba przeliczenia np. oprocentowania wkładu złotówkowego na inne waluty. Niemniej jednak reguła jest prosta — poprawnie skonstruowany algorytm zawsze da takie same wyniki niezależnie od „kierunku” przeprowadzanych obliczeń. Prosty schemat blokowy kolejności prezentowanych tu obliczeń został zamieszczony w diagramie (*Unit_MyValues.dpp*) modułu *Kody\Rozdzial1\My_Convs\Unit_MyValues.pas*.

Proste przykłady zastosowań funkcji przeliczających w odniesieniu do operacji walutowych można również znaleźć w katalogu instalacyjnym Delphi 6 *Demos\ConvertIt\EuroConv.pas*.

Należy zwrócić uwagę, iż z: `eVoltToJoul()`, `JoulToAtomUnitMass()`, `AtomUnitMassToKilograms()` oraz `KilogramsToeV()` możemy również korzystać tak jak z normalnych predefiniowanych funkcji i wywoływać je bezpośrednio z parametrami aktualnymi

w głównym module aplikacji, ich prototypy należy wówczas umieścić tuż przed początkiem sekcji implementacji. Liczbowe identyfikatory nowo zarejestrowanych jednostek oraz odpowiadających im układów jednostek z łatwością odczytamy dzięki funkcji `IntToStr()`, np.

```
Edit1.Text := IntToStr(eueV).
```

Na wydruku 1.2 pokazano kod głównego modułu *Unit_MyValues.pas* znajdującego się na płycie CD-ROM w katalogu *Kody\Rozdzial1\My_Convs\p_MyValues.dpr* realizującego operacje przeliczania samodzielnie zdefiniowanych i zarejestrowanych układów jednostek.

Wydruk 1.2. *Kod źródłowy modułu Unit_MyValues.pas implementującego wybrane układy jednostek wraz z odpowiadającymi im czynnikami przeliczeniowymi*

```
unit Unit_MyValues;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms,
  Dialogs, StdCtrls, ConvUtils, StdConvs;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    ListBox1: TListBox;
    ComboBox1: TComboBox;
    Button1: TButton;
    ListBox2: TListBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure ComboBox1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses My_Convs;
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  ComboBox1.Text := 'Wielkości';
  ComboBox1.Items[0] := 'Angles';
  ComboBox1.Items[1] := 'Power';
  ComboBox1.Items[2] := 'Force';
  ComboBox1.Items[3] := 'Energy';

  ListBox1.ItemIndex := 0;
  ListBox2.ItemIndex := 0;
end;
```

```
//-----  
procedure TForm1.ComboBox1Change(Sender: TObject);  
begin  
  
    case ComboBox1.ItemIndex of  
    0:  
        begin  
            ListBox1.Items[0] := 'Degrees';  
            ListBox1.Items[1] := 'Minutes';  
            ListBox1.Items[2] := 'Seconds';  
            ListBox1.Items[3] := 'Radians';  
            ListBox1.Items[4] := 'Cycles';  
  
            ListBox2.Items[0] := 'Degrees';  
            ListBox2.Items[1] := 'Minutes';  
            ListBox2.Items[2] := 'Seconds';  
            ListBox2.Items[3] := 'Radians';  
            ListBox2.Items[4] := 'Cycles';  
        end;  
    1:  
        begin  
            ListBox1.Items[0] := 'HoursPower';  
            ListBox1.Items[1] := 'CaloriesPerSecond';  
            ListBox1.Items[2] := 'Kilowatts';  
            ListBox1.Items[3] := 'Watts';  
            ListBox1.Items[4] := '';  
  
            ListBox2.Items[0] := 'HoursPower';  
            ListBox2.Items[1] := 'CaloriesPerSecond';  
            ListBox2.Items[2] := 'Kilowatts';  
            ListBox2.Items[3] := 'Watts';  
            ListBox2.Items[4] := '';  
        end;  
    2:  
        begin  
            ListBox1.Items[0] := 'Dynas';  
            ListBox1.Items[1] := 'Newtons';  
            ListBox1.Items[2] := 'GramsForce';  
            ListBox1.Items[3] := 'KilogramsForce';  
            ListBox1.Items[4] := 'PoundsForce';  
  
            ListBox2.Items[0] := 'Dynas';  
            ListBox2.Items[1] := 'Newtons';  
            ListBox2.Items[2] := 'GramsForce';  
            ListBox2.Items[3] := 'KilogramsForce';  
            ListBox2.Items[4] := 'PoundsForce';  
        end;  
    3:  
        begin  
            ListBox1.Items[0] := 'eV';  
            ListBox1.Items[1] := '';  
            ListBox1.Items[2] := '';  
            ListBox1.Items[3] := '';  
            ListBox1.Items[4] := '';  
  
            ListBox2.Items[0] := 'Kilograms';  
            ListBox2.Items[1] := '';  
            ListBox2.Items[2] := '';  
            ListBox2.Items[3] := '';  
            ListBox2.Items[4] := '';  
        end;  
    end;  
end;
```

```

        end;
    end;
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
var
    newValue: Double;
    AFamily : TConvFamily;
    FromType, ToType: TConvType;
    ADescription: string;
begin
    ADescription := ComboBox1.Items[ComboBox1.ItemIndex];
    if DescriptionToConvFamily(ADescription, AFamily) then
        begin
            DescriptionToConvType(AFamily,
                ListBox1.Items[ListBox1.ItemIndex],
                FromType);
            DescriptionToConvType(AFamily,
                ListBox2.Items[ListBox2.ItemIndex],
                ToType);
            newValue := Convert(Abs(StrToFloat(Edit1.Text)),
                FromType, ToType);
            ShowMessage(Format('%g %s %s %.4e %s',
                [Abs(StrToFloat(Edit1.Text)),
                ConvTypeToDescription(FromType),
                '=',
                newValue, ConvTypeToDescription(ToType)]))
        end;
end;
//-----
end.

```

Analizując powyższe zapisy należy zwrócić uwagę na to, iż zarówno w głównym module aplikacji *Unit_MyValues.pas*, jak i w module *My_Convs.pas* niezależnie odwołano się do modułów *StdConvs* oraz *ConvUtils*. Wynika to z faktu, iż zmienne, typy, funkcje oraz procedury z tych modułów nie są współdzielone przez wszystkie moduły wchodzące w skład aplikacji, natomiast są przekazywane każdemu jej modułowi jako odrębna kopia.

Moduł VarConv

Moduł *VarConv* implementuje dane typu *Variant* na potrzeby reprezentacji zarówno pojedynczych jednostek, jak i układów jednostek różnorodnych wielkości. Funkcje udostępniane przez ten moduł z reguły nie mogą występować samodzielnie w aplikacji. W wielu wypadkach konieczne jest używanie dodatkowych typów i procedur udostępnianych w module *Variants*. Do najważniejszych należy zaliczyć typ *TVarData* będący implementacją typu *Variant* w Delphi 6. Typ *TVariantManager* używany jest przez procedury *GetVariantManager()* oraz *SetVariantManager()*. Definiuje on wszystkie funkcje oraz procedury mające zastosowanie w konwersji typów wariantowych na inne typy danych. Z kolei *TVarOp* identyfikuje operatory arytmetyczne, logiczne i bitowe mające zastosowanie w działaniach na typach wariantowych. Więcej szczegółów na temat wspomnianych typów oraz procedur obsługujących dane wariantowe można znaleźć w dokumentacji Delphi 6 oraz w książce *Delphi. Almanach* (Helion, 2002).

Prezentowane w tym podrozdziale funkcje są częścią biblioteki CLX, zatem mogą być wykorzystywane w trakcie projektowania aplikacji międzyplatformowych.

VarAsConvert() — funkcja

Składnia

```
function VarAsConvert(const AValue: Variant; const AType: TConvType
    !Alink(tconvtype,1,TopicNotFound,main)): Variant; overload;

function VarAsConvert(const AValue: Variant): Variant; overload;
```

Opis

Funkcja `VarAsConvert()` konwertuje typ wariantowy określony przez parametr `AValue` na postać wariantową `VarConvert`.

W pierwszej postaci funkcji `AValue` reprezentuje wartość numeryczną wybranej wielkości, zaś `AType` identyfikuje jej jednostkę. W drugiej postaci funkcji parametr wariantowy `AValue` powinien reprezentować odpowiedni łańcuch znaków, np. `'1,5 miles'`.

Jeżeli funkcja `VarAsConvert()` nie zostanie wykonana pomyślnie, generowany jest wyjątek `EInvalidCast`.

Przykład

Poniższy fragment kodu w postaci funkcji obsługi zdarzenia `Button1Click()` dodaje wartości dwóch różnych wielkości należących do układu jednostek długości.

```
var
    Form1: TForm1;
    VariantManager: TVariantManager;

implementation
{$R *.xpm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    GetVariantManager(VariantManager);
    SetVariantManager(VariantManager);
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
var
    Left, VarParameter, Right: Variant;
begin
    VarParameter := '1000.00 meters';
    Left:=VarAsConvert(1000.0, duMeters);
    Right:=VarAsConvert(1.0, duMiles);
    VariantManager.VarOp(Left, Right, opAdd);
    ShowMessage(Format('%s %s %s %s %s',
        [Varparameter, ' + ', Right,
        ' = ', Left]))
end;
//-----
```

VarConvert() — funkcja

Składnia

```
function VarConvert: TVarType;
begin
  Result := ConvertVariantType.VarType;
end;
```

Opis

Każdy typ wariantowy przechowywany jest w polu VType rekordu TVarData. Wartość tego pola zwraca właściwość VarType typu TVarType. Bezparametrowa funkcja odzyskuje kod zmiennej typu Variant reprezentującej jednostkę należącą do określonego układu jednostek. Kody typów wariantowych są dynamicznie alokowane w momencie, gdy aplikacja ładuje i implementuje dołączony moduł. Funkcji tej z reguły nie używa się w sposób jawny, natomiast korzystają z niej takie funkcje modułu VarConv, jak: VarConvertCreateInto() czy VarIsConvert().

VarConvertCreate() — funkcja

Składnia

```
function VarConvertCreate(const AValue: Double; const AType:
  TConvType!ALink(tconvtype,1,TopicNotFound,main)): Variant;
  overload;

function VarConvertCreate(const AValue: string): Variant; overload;
```

Opis

Funkcja VarConvertCreate() zwraca wartość wariantową wybranej wielkości wraz z jej jednostką.

W pierwszej postaci funkcji parametry AType oraz AValue określają typ jednostki oraz wartość reprezentowanej przez nią wielkości. AType musi reprezentować zarejestrowaną jednostkę.

W drugiej postaci funkcji łańcuch AValue reprezentuje wartość wielkości wraz z jej jednostką, np. '10,0 liters'. Jeżeli łańcuch AValue nie odpowiada zarejestrowanej jednostce, funkcja generuje wyjątek EInvalidCast.

Wskazówki i porady

- ◆ Wykonując operacje dodawania, odejmowania i dzielenia odpowiednich wartości należy zwrócić uwagę na to, aby były one reprezentowane w tym samym układzie jednostek.
- ◆ Operacje mnożenia czynników przeliczeniowych przez odpowiednią liczbę są dozwolone.

Przykład

Główny moduł *Unit_VarConvertCreate.pas* projektu *Kody\Rozdzial1\VarConvertCreate\p_VarConvertCreate.dpr* realizuje operacje (dodawania, odejmowania, dzielenia całkowitego oraz mnożenia) określonych wielkości reprezentowanych jako typy wariantowe z wykorzystaniem różnych postaci funkcji *VarConvertCreate()*.

```

unit Unit_VarConvertCreate;

interface

uses
  SysUtils, Types, Classes, QGraphics, QControls,
  QForms, QDialogs, QStdCtrls, StdConv, Variants,
  VarConv;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  VariantManager: TVariantManager;

implementation

{$R *.xfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
  GetVariantManager(VariantManager);
  SetVariantManager(VariantManager);
end;
//-----
procedure TForm1.Button1Click(Sender: TObject);
var
  Left, VarParameter, Right: Variant;

begin
  VarParameter := '1,33 AstronomicalUnits';
  Left:=VarConvertCreate(VarParameter);
  Right:=VarConvertCreate('3,33 AstronomicalUnits');
  VariantManager.VarOp(Left, Right, opAdd);

```

```

        ShowMessage(Format(' %s %s %s %s %s',
            [VarParameter, ' + ', Right,
              ' = ', Left]))
    end;
    //-----
    procedure TForm1.Button2Click(Sender: TObject);
    var
        Left, VarParameter, Right: Variant;
    begin
        VarParameter := '33000,33 Meters';
        Left:=VarConvertCreate(VarParameter);
        Right:=VarConvertCreate(3.33, duKilometers);
        VariantManager.VarOp(Left, Right, opSubtract);
        ShowMessage(Format(' %s %s %s %s %s',
            [VarParameter, ' - ',
              Right, ' = ', Left]))
    end;
    //-----
    procedure TForm1.Button3Click(Sender: TObject);
    var
        Left, VarParameter, Right: Variant;
    begin
        VarParameter := '10,0 liters';
        Left:=VarConvertCreate(VarParameter);
        Right:=VarConvertCreate(1.0, vuUKPints);
        VariantManager.VarOp(Left, Right, opIntDivide);
        ShowMessage(Format(' %s %s %s %s %s',
            [VarParameter, ' div ', Right,
              ' = ', Left]))
    end;
    //-----
    procedure TForm1.Button4Click(Sender: TObject);
    var
        Left, VarParameter, Right: Variant;
    begin
        VarParameter := '10,0 Parsecs';
        Left:=VarConvertCreate(VarParameter);
        Right:=1235;
        VariantManager.VarOp(Left, Right, opMultiply);
        ShowMessage(Format(' %s %s %s %s %s',
            [VarParameter, ' * ', Right,
              ' = ', Left]))
    end;
    //-----
    end.

```

VarIsConvert() — funkcja

Składnia

```
function VarIsConvert(const AValue: Variant): Boolean;
```

Opis

Funkcja sprawdza, czy parametr wariantowy określony przez *AValue* jest zarejestrowanym typem jednostki lub reprezentuje zarejestrowany układ jednostek.

Przykład

Często z funkcji *VarIsConvert()* nie korzysta się bezpośrednio. Zamiast jawnego jej wywołania, niekiedy większy pożytek przynosi posługiwanie się następującą konstrukcją:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  V: Variant;

begin
  V := duFur1ongs;
  //V := cbDistance;
  if (TVarData(V).VType and varTypeMask) <> varEmpty
  then
    ShowMessage(Format(' %s %s %s ',
      ['Jednostka (układ jednostek) nr ',
        V, 'jest zarejestrowana(y)']));
end;
//-----
```

Podsumowanie

W rozdziale tym zostały opisane moduły *StdConvs*, *ConvUtils* oraz *VarConv*. Umiejętność posługiwania się udostępnianymi przez nie zmiennymi, typami, procedurami oraz funkcjami pozwoli Czytelnikom zaznajomić się z nowoczesnymi metodami wykorzystywania Delphi 6 jako narzędzia pomocnego w wykonywaniu skomplikowanych nieraz obliczeń na różnego rodzaju wielkościach fizycznych. W przykładzie wyjaśniającym ideę kontrolowania własnych układów jednostek oraz czynników przeliczeniowych wykorzystano układy często występujące w obliczeniach naukowo-technicznych, niemniej jednak może on zostać szybko zaadaptowany również do innych dziedzin wiedzy, takich jak bankowość, finanse (obliczanie oprocentowania lokat w różnych walutach). Korzystając z wiadomości przedstawionych w tym rozdziale każdy będzie mógł dla własnych potrzeb zaprojektować aplikację, za pomocą której bardzo szybko będzie można dokonać nawet wysoce skomplikowanych i „egzotycznych” przeliczeń — bez potrzeby ciągłego odwoływania się do różnego rodzaju obszernych, książkowych poradników.