

## 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 7. Kompendium programisty

Autor: Adam Boduch  
ISBN: 83-7361-087-1  
Format: B5, stron: 686



Spśród wielu dostępnych środowisk programistycznych Delphi wyróżnia się łatwością i szybkością tworzenia aplikacji. Zastosowany język programowania ObjectPascal utrwała dobre nawyki programowania strukturalnego, wzbogacając je o nowe możliwości, jakie niesie za sobą programowanie zorientowane obiektowo. Wygodne środowisko RAD, możliwość łatwego uzyskiwania dostępu do relacyjnych baz danych oraz możliwość tworzenia aplikacji wieloplatformowych, to kolejne zalety Delphi, które przekonały doń rzesze programistów.

Książka „Delphi 7. Kompendium programisty” to – jak wskazuje jej tytuł – kompletny przewodnik po Delphi, obejmujący zarówno opis zintegrowanego środowiska programistycznego, języka ObjectPascal, jak i najważniejszych funkcjonalności zawartych w dołączonych do Delphi bibliotekach. Jest to pozycja dla początkujących adeptów sztuki programistycznej, a także dla tych wszystkich, którzy chcą poszerzyć i wzbogacić swoją wiedzę o tym wygodnym narzędziu.

Książka przedstawia:

- Podstawowe informacje o Delphi
- Język ObjectPascal – jego strukturę i składnię
- Programowanie zorientowane obiektowo
- Interfejs Delphi
- Obsługę komunikatów w Delphi
- Korzystanie z rejestru Windows i plików .ini
- Dostęp do plików
- Tworzenie aplikacji wielowątkowych
- Multimedialne funkcje Delphi
- Tworzenie aplikacji sieciowych
- Pisanie własnych bibliotek DLL
- Wykorzystanie oraz tworzenie własnych bibliotek VCL i CLX
- Dostęp do relacyjnych baz danych
- IntraWeb – Delphi i WWW

Do książki dołączony jest CD-ROM, na którym znajdziesz kody źródłowe wykorzystane w książce. Cennym uzupełnieniem jest także dodatek, przedstawiający przetestowane w praktyce zalecenia dotyczące pisania czytelnego kodu w Delphi.



---

# Spis treści

Wstęp.....	17
<b>Część I.....</b>	<b>19</b>
<b>Rozdział 1. Podstawy Delphi.....</b>	<b>21</b>
Czym jest programowanie? .....	21
Przegląd języków programowania .....	22
Czym jest Delphi? .....	23
Co należy umieć? .....	23
Historia Delphi .....	24
Proces instalacji.....	24
Korzystanie z polskich liter .....	24
Wygląd Delphi .....	25
Okno główne .....	26
Projektant formularzy .....	27
Inspektor obiektów.....	28
Drzewo obiektów .....	29
Edytor kodu.....	30
Pierwszy projekt.....	32
Tworzenie projektu .....	32
Zapisywanie projektu .....	34
Otwieranie projektu .....	34
Kompilacja projektu.....	35
Pliki projektu.....	35
Podsumowanie .....	36
<b>Rozdział 2. Język Object Pascal.....</b>	<b>37</b>
Plik źródłowy projektu .....	37
Najprostszy program .....	38
Podstawowa składnia .....	38
Wielkość liter.....	39
Pamiętaj o średniku! .....	39

Bloki begin i end.....	39
Dyrektywa program .....	40
Komentarze .....	40
Zmienne.....	41
Deklaracja zmiennych.....	42
Typy zmiennych.....	42
Deklaracja kilku zmiennych .....	44
Przydział danych.....	44
Stałe.....	46
Domyślne typy stałych.....	46
Używanie stałych i zmiennych w programie .....	47
Tablice danych .....	48
Tablice jako stałe .....	49
Tablice wielowymiarowe.....	50
Tablice dynamiczne .....	51
Operatory .....	53
Aplikacje konsolowe .....	54
Instrukcje warunkowe .....	55
Instrukcja if..then .....	55
Instrukcja case..of .....	58
Instrukcja else .....	60
Procedury i funkcje .....	63
Procedury .....	63
Funkcje.....	64
Zmienne lokalne.....	65
Parametry procedur i funkcji .....	66
Parametry domyślne.....	67
Przeciążanie funkcji i procedur.....	68
Typy parametrów przekazywanych do procedur i funkcji .....	69
Procedury zagnieżdżone .....	71
Własne typy danych .....	71
Tablice jako nowy typ.....	72
Aliasy typów .....	73
Rekordy .....	73
Przekazywanie rekordów jako parametr procedury.....	73
Deklaracja rekordu jako zmienna .....	75
Instrukcja packed .....	75
Instrukcja wiążąca with.....	75
Moduły .....	76
Tworzenie nowego modułu.....	76
Budowa modułu .....	77
Włączanie modułu .....	78
Sekcja Initialization oraz Finalization .....	78
Konwersja typów.....	79
Rzutowanie.....	81
Pętle.....	82
Pętla for..do.....	82
Pętla while..do.....	83
Pętla repeat..until .....	85
Procedura Continue.....	85
Procedura Break.....	86

Zbiory .....	87
Przypisywanie elementów zbioru .....	87
Odczytywanie elementów ze zbioru .....	88
Dodawanie i odejmowanie elementów zbioru .....	89
Wskaźniki .....	90
Tworzenie wskaźnika .....	90
Przydział danych do wskaźników .....	90
Do czego to służy? .....	91
Przydział i zwalnianie pamięci .....	93
Wartość pusta .....	94
Pliki dołączane .....	94
Etykiety .....	95
Podsumowanie .....	95
<b>Rozdział 3. Programowanie obiektowe .....</b>	<b>97</b>
VCL .....	97
Podstawowy kod modułu .....	98
Plik DFM formularza .....	99
Umieszczanie komponentów na formularzu .....	100
Umieszczanie komponentów .....	100
Zmiana właściwości komponentu .....	100
Zdarzenia komponentów .....	101
Kod generowany automatycznie .....	103
Klasy .....	104
Czym właściwie są klasy? .....	105
Tworzenie klas .....	105
Definicja metod .....	105
Tworzenie klasy .....	106
Poziomy dostęp do klasy .....	107
Dziedziczenie .....	108
Typy metod .....	109
Przedefiniowanie metod .....	109
Konstruktory i destruktory .....	112
Przykład użycia klas .....	112
Ogólne założenia .....	112
Tworzenie modułu Engine .....	113
Interfejs programu .....	117
Kod źródłowy formularza głównego .....	118
Parametr Sender procedury zdarzeniowej .....	120
Przechwytywanie informacji o naciśniętym klawiszu .....	120
Obsługa parametru Sender .....	122
Operatory is i as .....	123
Parametr Self .....	123
Łańcuchy tekstowe .....	124
ShortString .....	125
AnsiString .....	125
WideString .....	126
Łańcuchy z zerowym ogranicznikiem .....	126
Operacje na łańcuchach .....	126
Łączenie łańcuchów .....	126
Wycinanie łańcucha .....	127
Uzyskiwanie fragmentów łańcucha .....	127
Wstawianie danych do łańcucha .....	128

Wyszukiwanie danych w łańcuchu .....	129
Pozostałe funkcje .....	129
Typ wariantowe.....	131
Właściwości .....	132
Zdarzenia .....	137
Wyjątki .....	141
Słowa kluczowe try..except .....	141
Słowa kluczowe try..finally .....	142
Słowo kluczowe raise .....	143
Klasa Exception .....	144
Selektywna obsługa wyjątków.....	145
Zdarzenie OnException .....	145
Klasa TApplication .....	148
Właściwości klasy TApplication .....	148
Metody klasy TApplication .....	149
Zdarzenia klasy TApplication.....	151
Podsumowanie .....	152
<b>Rozdział 4. IDE Delphi .....</b>	<b>153</b>
Paski narzędziowe .....	154
Pasek Standard.....	154
Pasek View.....	154
Pasek Debug.....	155
Pasek Desktop.....	156
Pasek Custom.....	156
Pasek Internet.....	156
Repozytorium .....	157
Dodawanie projektu do Repozytorium .....	157
Ustawienia Repozytorium.....	158
Praca z paletą komponentów.....	158
Umieszczanie kilku obiektów naraz .....	159
Menu palety komponentów.....	159
Projektant formularzy.....	161
Menu projektanta formularzy .....	162
Drzewo obiektów .....	164
Inspektor obiektów .....	165
Menu podręczne Inspektora obiektów .....	165
Właściwości Inspektora obiektów .....	166
Eksplorator kodu .....	168
Przeglądarka projektu.....	169
Lista To-Do .....	169
Znaczniki to-do w kodzie.....	170
Diagramy .....	171
Code Insight .....	172
Code Completion .....	173
Ustawienia Code Insight .....	174
Projekty .....	174
Opcje projektu.....	175
Pliki projektu.....	182
Menedżer projektu.....	183
Pasek narzędziowy.....	183
Praca z menedżerem projektu .....	184
Menu podręczne menedżera projektów .....	184

Kilka wersji językowych projektu.....	188
Tworzymy angielską wersję językową .....	188
Tłumaczenie projektu.....	190
Kompilacja projektu.....	191
Kilka formularzy w jednym projekcie .....	191
Wyświetlenie drugiego formularza.....	192
Dynamiczne tworzenie formularza .....	192
Aplikacje MDI.....	194
Projektowanie interfejsu .....	194
Kod źródłowy przykładowej aplikacji.....	197
Czy warto?.....	198
Delphi a C++Builder .....	198
Rozmiar aplikacji wykonywalnej i czas kompilacji .....	199
Pliki .....	199
Składnia.....	199
Co nowego w Delphi 7?.....	201
Zmiany w IDE.....	202
Nowe komponenty .....	203
Bazy danych.....	205
.NET .....	205
Modyfikacje kompilatora.....	206
Podsumowanie .....	206
<b>Podsumowanie części I .....</b>	<b>207</b>

## **Część II..... 209**

<b>Rozdział 5. Obsługa komunikatów .....</b>	<b>211</b>
Czym są komunikaty? .....	211
Rodzaje komunikatów .....	212
Komunikaty okienkowe .....	212
Komunikaty powiadamiające.....	213
Komunikaty definiowane przez użytkownika .....	213
Jak działają komunikaty? .....	215
Obsługa komunikatów.....	215
Przechwytywanie komunikatów .....	215
Struktura TMsg .....	216
Struktura TMessage .....	217
Specyficzne struktury typów .....	219
Zdarzenie OnMessage .....	220
Wysyłanie komunikatów .....	222
Perform .....	222
Funkcje SendMessage i PostMessage.....	224
Komunikaty rozgłaszające .....	226
Deklarowanie własnych komunikatów .....	226
Funkcje operujące na uchwytach .....	228
„Zahaczanie” okien .....	233
Zakładanie globalnego hooka .....	234
Funkcja obsługująca hooka.....	235
Podsumowanie .....	238

<b>Rozdział 6.</b>	<b>Rejestry i pliki INI</b>	<b>239</b>
	Czym jest Rejestr Windows?	239
	Podstawowe klucze	240
	Podstawowe pojęcia	241
	Pliki INI	241
	Budowa	241
	Rejestr kontra plik INI	242
	Klasa TRegistry	242
	Tworzenie nowych kluczy	242
	Otwieranie istniejących kluczy	243
	Usuwanie kluczy	244
	Dodawanie wartości	244
	Odczyt danych	247
	Inne funkcje operujące na Rejestrze	249
	Praktyczny przykład	253
	Klasa TINIFile	255
	Tworzenie nowego pliku INI	255
	Zapisywanie danych	256
	Odczyt danych	256
	Funkcje związane z operacjami na sekcjach	257
	Przykładowy program	258
	Podsumowanie	260
<b>Rozdział 7.</b>	<b>Obsługa plików</b>	<b>261</b>
	Pliki tekstowe	261
	Inicjalizacja	262
	Tworzenie nowego pliku	262
	Otwieranie istniejącego pliku	263
	Odczyt plików tekstowych	263
	Zapis nowych danych w pliku	265
	Zapis danych na końcu pliku	265
	Pliki amorficzne	266
	Otwieranie i zamykanie plików	266
	Tryb otwarcia pliku	267
	Zapis i odczyt danych	268
	Przykład działania — kopiowanie plików	269
	Inne funkcje operujące na plikach	271
	Funkcje operujące na katalogach	273
	Pliki typowane	274
	Deklaracja	275
	Tworzenie pliku i dodawanie danych	275
	Odczyt rekordu z pliku	276
	Przykład działania — książka adresowa	276
	Kopiowanie i przenoszenie plików	283
	Kopiowanie	283
	Przenoszenie pliku	283
	Struktura TSHFileOpStruct	283
	Strumienie	285
	Podział strumieni	286
	Prosty przykład na początek	286
	Konstruktor klasy TFileStream	288
	Pozostałe metody i właściwości klasy TStream	288
	Właściwości	289

Metody .....	289
Praktyczny przykład.....	290
Wyszukiwanie .....	297
Rekord TSearchRec .....	297
Jak zrealizować wyszukiwanie? .....	297
Rekurencja .....	299
Praktyczny przykład.....	299
Informacja o dyskach .....	303
Pobieranie listy dysków .....	303
Pobieranie rozmiaru dysku .....	304
Pobieranie dodatkowych informacji .....	304
Podsumowanie .....	307
<b>Rozdział 8. Aplikacje wielowątkowe .....</b>	<b>309</b>
Czym tak naprawdę są wątki? .....	309
Klasa TThread .....	310
Deklaracja klasy TThread .....	310
Tworzenie nowej klasy wątku.....	311
Kilka instancji wątku.....	313
Tworzenie klasy .....	313
Kod klasy .....	314
Wznawianie i wstrzymywanie wątków .....	316
Priorytet wątku .....	317
Synchronizacja .....	317
Treść komentarza .....	318
Zdarzenia klasy TThread.....	320
Przykład: wyszukiwanie wielowątkowe .....	320
Jak to działa?.....	320
Wyszukiwanie.....	321
Obliczanie czasu przeszukiwania .....	322
Kod źródłowy modułu .....	322
Podsumowanie .....	325
<b>Rozdział 9. Multimedia.....</b>	<b>327</b>
Z czego będziemy korzystać? .....	327
OpenGL.....	328
DirectX.....	330
Tworzenie bitmap.....	331
Korzystanie z komponentu TImage .....	331
Ładowanie obrazków w trakcie działania programu .....	332
Klasa TBitmap .....	332
Odczytywanie obrazka ze schowka .....	333
Pliki JPEG .....	337
Klasa TJPEGImage.....	337
Wyświetlanie obrazków w komponentcie TImage.....	338
Przykład działania — kompresja plików .....	338
Pliki GIF .....	339
Zasoby .....	339
Tworzenie zasobów .....	339
Wykorzystanie zasobów .....	341
Ręczne tworzenie zasobów .....	344
Doklejanie plików EXE .....	347
Klasa TCanvas.....	347



Pióra i pędzle .....	348
Klasa TPen .....	348
Klasa TBrush .....	350
Czcionki .....	351
Właściwości klasy TFont .....	352
Metody klasy TCanvas .....	352
Draw .....	353
FillRect .....	353
StretchDraw .....	354
TextOut .....	354
TextRect .....	355
TextWidth, TextHeight .....	355
TextExtent .....	356
MoveTo .....	356
LineTo .....	356
Inne funkcje służące do rysowania kształtów .....	357
Proste animacje tekstowe .....	360
Tekst trójwymiarowy (3D) .....	360
Efekt maszyny do pisania .....	361
Animacja na belce tytułowej .....	362
Inne płynne animacje .....	365
Odtwarzanie dźwięków .....	366
Funkcja PlaySound .....	366
Użycie komponentu TMediaPlayer .....	367
Odtwarzanie filmów .....	368
Kontrolka Flash .....	375
Instalacja kontrolki .....	375
Wykorzystanie komponentu .....	377
Podsumowanie .....	377
<b>Rozdział 10. Biblioteki DLL .....</b>	<b>379</b>
Czym jest biblioteka DLL? .....	379
Do czego mogą się przydać biblioteki DLL? .....	380
Zalety .....	380
Wady .....	380
Tworzenie bibliotek DLL .....	381
Budowa biblioteki .....	382
Rozmiar biblioteki .....	382
Eksportowanie procedur i funkcji .....	382
Eksportowanie przez nazwę .....	382
Eksport przez indeks .....	383
Ładowanie bibliotek DLL .....	383
Ładowanie statyczne .....	384
Ładowanie dynamiczne .....	384
Konwersje wywołania .....	386
Formularze w bibliotekach DLL .....	387
Tworzenie formularza .....	387
Eksportowanie formularza .....	387
Przekazywanie rekordów do bibliotek .....	389
Budowa pliku mp3 .....	389
Odczyt tagu z pliku mp3 .....	390
Demo .....	392
Łańcuchy w bibliotekach DLL .....	394

Zasoby w bibliotece DLL.....	394
Przygotowanie zasobów.....	394
Ładowanie zasobów z biblioteki DLL.....	395
Procedura inicjująco-kończąca.....	396
Blok begin biblioteki DLL.....	396
DLLProc.....	396
Kod biblioteki.....	397
Program wykorzystujący bibliotekę.....	398
Podsumowanie.....	398
<b>Rozdział 11. Aplikacje sieciowe.....</b>	<b>399</b>
Z czego będziemy korzystać?.....	399
Odrobinę teorii.....	400
IP.....	400
TCP.....	401
Porty.....	402
HTTP.....	402
HTTPS.....	402
FTP.....	403
SMTP.....	403
Biblioteka WinInet.dll.....	403
Ustawianie połączenia.....	404
Otwieranie konkretnego adresu URL.....	405
Odczyt pliku.....	405
Pobieranie rozmiaru pliku.....	406
Sprawdzanie połączenia.....	409
Sprawdzanie IP.....	410
Zainicjowanie biblioteki.....	410
Pobieranie adresu IP.....	411
Łączenie przy użyciu gniazd.....	412
Czego użyć?.....	412
Łączenie pomiędzy komputerami.....	412
Wymiana danych.....	413
Jak działają „konie trojańskie”?.....	416
Pingi.....	418
Wysyłanie sygnału ping.....	418
Odpowiedzi z serwera.....	418
Kontrolka TWebBrowser.....	420
Ładowanie strony.....	422
Odświeżanie.....	422
Następna i poprzednia strona.....	423
Pozostałe kody.....	423
Protokół SMTP.....	424
Interfejs programu.....	424
Działanie programu.....	424
Protokół HTTP.....	428
Łączenie z serwerem.....	429
Wymiana danych.....	429
Pobieranie kodu strony WWW.....	430
Wysyłanie danych przez skrypt PHP.....	431
Praktyczne przykłady wykorzystania HTTP.....	433
Sprawdzenie nowej wersji programu.....	433
Korzystanie z zewnętrznej wyszukiwarki.....	436
Protokół FTP.....	442
Podsumowanie.....	442

<b>Rozdział 12.</b>	<b>WinAPI</b> .....	<b>443</b>
	Czym tak naprawdę jest WinAPI? .....	443
	Zasady tworzenia programów za pomocą WinAPI .....	444
	Brak zdarzeń .....	444
	Brak komponentów .....	444
	Zalety wykorzystania WinAPI .....	444
	Pierwszy program.....	445
	Funkcja okienkowa .....	445
	Rejestracja klasy.....	446
	Tworzenie formularza .....	448
	Komunikaty i uchwytów .....	450
	Łańcuchy .....	451
	Konwersja łańcuchów .....	451
	Funkcje operujące na łańcuchach .....	452
	Tworzenie kontrolerek .....	454
	Umieszczanie kontrolerek przy starcie programu.....	454
	Flagi kontrolerek .....	457
	Obsługa zdarzeń .....	459
	Uchwytów do kontrolerek.....	460
	Tworzenie bardziej zaawansowanych kontrolerek.....	462
	Pozostałe kontrolki.....	464
	Wyświetlanie grafiki .....	464
	Rysowanie w WinAPI.....	465
	Kontekst urządzenia graficznego .....	465
	Obsługa WM_PAINT .....	466
	Ładowanie i wyświetlanie bitmapy .....	467
	Ładowanie zasobów .....	468
	Skompilowane zasoby .....	468
	Wykorzystanie zasobów .....	469
	Podsumowanie .....	473
<b>Rozdział 13.</b>	<b>COM i ActiveX</b> .....	<b>475</b>
	Czym jest COM? .....	475
	Kontrolka w rozumieniu COM .....	475
	Odrobinę historii .....	476
	Tworzenie obiektów COM.....	476
	Metody i właściwości .....	478
	Dodawanie metod .....	478
	Dodawanie właściwości .....	480
	Kod źródłowy kontrolki .....	481
	Przykład działania: kodowanie tekstu .....	484
	Budowa i rejestracja kontrolki .....	486
	Wykorzystanie obiektu COM.....	486
	Interfejsy.....	488
	GUID.....	489
	ActiveX .....	489
	Import kontrolerek ActiveX.....	489
	Wykorzystanie kontrolki TVText .....	491
	Tworzenie kontrolerek ActiveX.....	492
	Przykład: wyświetlanie napisów do filmu .....	493
	Tworzenie interfejsu COM .....	493
	Tworzenie kontrolki ActiveX .....	494
	Budowa i rejestracja.....	506

ActiveX w Internecie .....	506
Względy bezpieczeństwa .....	506
Przykładowa kontrolka .....	506
Podsumowanie .....	509
<b>Podsumowanie części II.....</b>	<b>511</b>
<b>Część III.....</b>	<b>513</b>
<b>Rozdział 14. Komponenty VCL i CLX.....</b>	<b>515</b>
Czym jest komponent? .....	515
VCL .....	516
CLX .....	516
Tworzenie aplikacji opartych na CLX .....	516
CLX a VCL .....	516
Architektura CLX .....	517
Windows a Linux .....	517
Kompilacja warunkowa .....	518
Rodzaje komponentów .....	518
Komponenty wizualne .....	518
Komponenty niewizualne .....	518
Komponenty graficzne .....	519
Hierarchia komponentów .....	519
TObject .....	519
TPersistent.....	521
TComponent .....	522
TControl.....	525
TWinControl i TWidgetControl .....	526
Klasy TCustom .....	527
TGraphicControl.....	527
Budowa komponentu.....	527
Właściwości .....	527
Zdarzenia.....	529
Metody .....	530
RTTI .....	530
Właściwości obiektu .....	530
Dokładniejsze informacje o obiekcie.....	532
Podsumowanie .....	535
<b>Rozdział 15. Tworzenie komponentów .....</b>	<b>537</b>
Tworzenie nowego komponentu .....	537
Edycja kodu .....	538
Konstruktory i destruktory .....	539
Właściwości .....	540
Klauzula default .....	540
Klauzula stored .....	541
Klauzula nodefault .....	541
Właściwość wyliczeniowa .....	541
Właściwość zbiorowa .....	542
Zdarzenia .....	543
Definiowanie własnych zdarzeń .....	544
Ikona dla komponentów .....	544

Przykładowy komponent.....	545
Ogólny zarys klasy.....	545
Komunikaty.....	545
Kod źródłowy komponentu.....	546
Instalacja komponentów.....	548
Demonstracja możliwości komponentu TURLLabel.....	550
Zachowanie komponentu.....	551
Komponenty graficzne.....	551
Ogólny zarys klasy komponentu.....	552
Kod źródłowy komponentu.....	553
Pakiety komponentów.....	558
Podsumowanie.....	560
Podsumowanie części III.....	561
<b>Część IV.....</b>	<b>563</b>
<b>Rozdział 16. Bazy danych BDE.....</b>	<b>565</b>
Czym jest baza danych?.....	565
Typy baz danych.....	566
Lokalne bazy danych.....	566
Bazy danych typu klient-serwer.....	566
Wielowarstwowość baz danych.....	566
Bazy danych w Delphi.....	567
Borland Database Engine.....	567
Sterowniki baz danych.....	567
Przykładowa baza danych.....	567
Komponenty bazodanowe.....	569
Komponent TTable.....	570
TDataSource.....	571
TDataSet.....	571
Komponent TQuery.....	572
BDE Administrator.....	573
Tworzenie nowej bazy danych.....	573
Tworzenie bazy w kodzie programu.....	574
Tworzenie tabel.....	575
Tworzenie rekordów.....	577
Odczytywanie wartości z tabeli.....	578
Przykładowy program korzystający z naszej bazy danych.....	578
Podsumowanie.....	580
<b>Rozdział 17. Bazy danych dbExpress.....</b>	<b>581</b>
Aplikacje klient-serwer.....	581
Narzędzia.....	582
Komponenty.....	582
Łączenie z serwerem.....	582
Kontrola procesu logowania.....	584
Zdarzenia AfterConnect i AfterDisconnect.....	585
Jak działa MySQL?.....	585
Tabele.....	585
Zapytania.....	586
Tworzenie tabel.....	586

Dodawanie rekordów .....	587
Kasowanie rekordów .....	589
Procedura kasująca .....	590
Odczytywanie rekordów .....	590
Przykładowy program .....	591
Zmiana wartości rekordów .....	593
Przykładowy program: spis sprzedanych towarów .....	594
Inne komponenty dbExpress .....	597
Programy oparte o dbExpress .....	598
Podsumowanie .....	598
<b>Podsumowanie części IV .....</b>	<b>599</b>
<b>Część V .....</b>	<b>601</b>
<b>Rozdział 18. Delphi a Internet .....</b>	<b>603</b>
Z czego będziemy korzystać? .....	603
Serwer Personal Web Server .....	604
CGI, ISAPI, NSAPI .....	605
Tworzenie rozszerzeń serwera .....	606
Akcje serwera .....	607
Uruchamianie biblioteki .....	609
Kod źródłowy biblioteki ISAPI .....	609
TWebRequest i TWebResponse .....	610
Wykorzystanie szablonów .....	612
Tworzenie nowego szablonu .....	613
Szablony dynamiczne .....	613
Przykładowy program .....	615
Dodatkowe parametry .....	617
Wysyłanie i odbieranie cookies .....	618
Ustawianie pliku cookies .....	618
Odczyt cookies .....	619
Wysyłanie strumieni .....	620
Korzystanie z baz danych .....	621
WebSnap .....	623
Podsumowanie .....	623
<b>Rozdział 19. IntraWeb .....</b>	<b>625</b>
Czym właściwie jest IntraWeb? .....	625
Tworzenie projektu IntraWeb .....	626
Umieszczanie komponentów .....	627
Podgląd wyglądu formularza .....	627
Uruchamianie projektu .....	628
Obsługa aplikacji serwera .....	628
Generowanie zdarzeń .....	629
Zdarzenia zastępcze .....	630
Kilka formularzy w jednym projekcie .....	630
Funkcja ShowMessage w IntraWeb .....	633
Elementy HTML i JavaScript .....	633
Wysyłanie plików .....	634

IntraWeb jako DLL .....	635	
Konwertowanie aplikacji do ISAPI .....	636	
IntraWeb kontra tradycyjne metody .....	636	
Podsumowanie .....	637	
<b>Podsumowanie części V .....</b>	<b>639</b>	
<b>Dodatki .....</b>	<b>641</b>	
<b>Dodatek A</b>	<b>Zasady pisania kodu .....</b>	<b>643</b>
	Stosowanie wcięć .....	644
	Instrukcje begin i end .....	644
	„Styl wielbłądzi” w nazwach procedur .....	645
	Stosuj wielkie litery .....	645
	Parametry procedur .....	645
	Instrukcja if .....	646
	Instrukcja case .....	646
	Obsługa wyjątków .....	647
	Klasy .....	647
	Komentarze .....	647
	Pliki i nazwy formularzy .....	648
	Notacja węgierska .....	648
	Czy warto? .....	649
	<b>Zakończenie .....</b>	<b>651</b>
	<b>Skorowidz .....</b>	<b>653</b>

# 8.

---

## Aplikacje wielowątkowe

Słowo *wątek* może mieć różne znaczenie. W świecie programistów może oznaczać możliwość wykonywania wielu czynności naraz. Przykładowo w systemie Windows możemy uruchamiać kilka programów działających jednocześnie — każdy program jest osobnym wątkiem. W tym rozdziale zajmiemy się tworzeniem kilku wątków w ramach jednego procesu.



### Wskazówka

---

Procesem można nazwać każdą aplikację, uruchomioną w danym momencie. Taką też terminologię będę stosował w dalszej części tego rozdziału. Zatem przyjmijmy, że proces to egzemplarz aplikacji uruchomiony w systemie.

## Czym tak naprawdę są wątki?

Zacznijmy od wyjaśnienia, czym tak naprawdę są wątki. Każda aplikacja (proces) działająca w systemie Windows posiada tzw. *wątek główny* (ang. *primary thread*), który może uruchamiać inne wątki poboczne (ang. *secondary threads*). W tym samym czasie może działać kilka wątków pobocznych, które wykonują różne lub te same operacje. Spójrz na rysunek 8.1. Program przedstawiony na tym rysunku dokonuje wyszukiwania wielowątkowego, analizując jednocześnie wszystkie dyski znajdujące się w systemie.

W tym wypadku zadaniem każdego wątku jest wyszukanie plików na osobnym dysku. W rezultacie jeden wątek przypada na każdy dysk, dzięki czemu wyszukiwanie trwa naprawdę szybko.

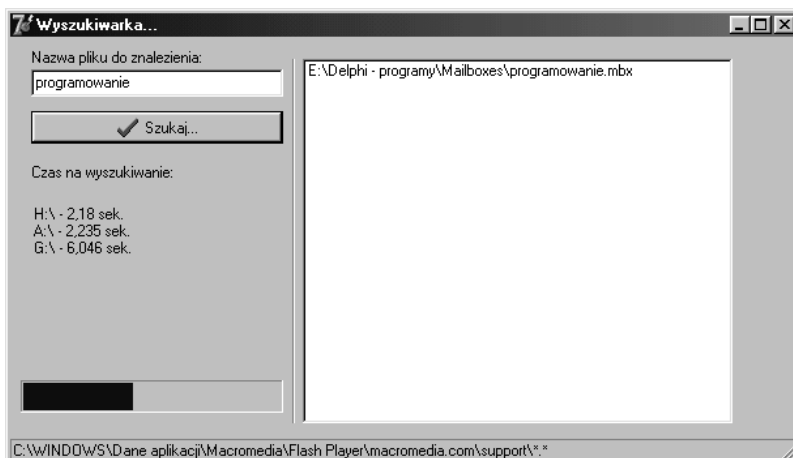


### Wskazówka

---

Pełny kod źródłowy programu *Wyszukiwanie wielowątkowe* możesz znaleźć na płycie CD-ROM w katalogu `../listingi/8/Wyszukiwarka`.





Rysunek 8.1. Wyszukiwanie wielowątkowe

Być może to, co napisałem do tej pory przybliżyło Ci trochę zasadę funkcjonowania wątków. Wyobraź sobie możliwość wykonywania innych czynności *w tle* aplikacji — bez jej jednoczesnego blokowania. Dajesz użytkownikowi możliwość dokonywania zmian w programie, a w tle może działać inny wątek, który wykonywać będzie pozostałe operacje.

## Klasa TThread

Podczas tworzenia aplikacji wielowątkowych będziemy korzystali z klasy VCL — TThread. Istnieje oczywiście możliwość tworzenia wątków przy wykorzystaniu mechanizmów WinAPI, lecz klasa TThread w dużym stopniu zwalnia nas z mozolnego kodowania — jest po prostu łatwiejsza w obsłudze.



### Wskazówka

Klasa TThread znajduje się w module *Classes.pas*.

## Deklaracja klasy TThread

Deklaracja klasy TThread znajduje się w pliku *Classes.pas* i przedstawia się w następujący sposób:

```
TThread = class
private
    FHandle: THandle;
    FThreadID: THandle;
    FTerminated: Boolean;
```

```

FSuspended: Boolean;
FFreeOnTerminate: Boolean;
FFinished: Boolean;
FReturnValue: Integer;
FOnTerminate: TNotifyEvent;
FMethod: TThreadMethod;
FSynchronizeException: TObject;
procedure CallOnTerminate;
function GetPriority: TThreadPriority;
procedure SetPriority(Value: TThreadPriority);
procedure SetSuspended(Value: Boolean);
protected
  procedure DoTerminate; virtual;
  procedure Execute; virtual; abstract;
  procedure Synchronize(Method: TThreadMethod);
  property ReturnValue: Integer read FReturnValue write FReturnValue;
  property Terminated: Boolean read FTerminated;
public
  constructor Create(CreateSuspended: Boolean);
  destructor Destroy; override;
  procedure Resume;
  procedure Suspend;
  procedure Terminate;
  function WaitFor: LongWord;
  property FreeOnTerminate: Boolean read FFreeOnTerminate write FFreeOnTerminate;
  property Handle: THandle read FHandle;
  property Priority: TThreadPriority read GetPriority write SetPriority;
  property Suspended: Boolean read FSuspended write SetSuspended;
  property ThreadID: THandle read FThreadID;
  property OnTerminate: TNotifyEvent read FOnTerminate write FOnTerminate;
end;

```

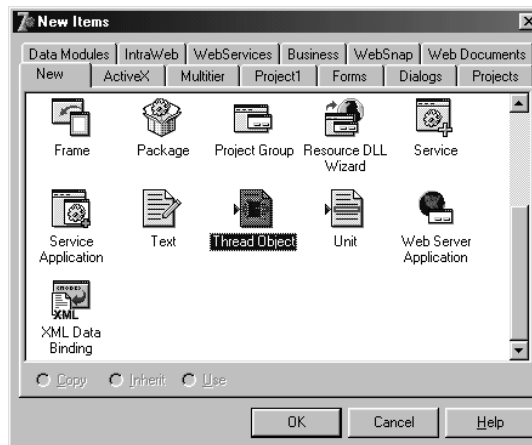
Działanie wątku można wstrzymać lub wznowić dzięki metodom `Suspend` i `Resume`. Rozpoczęcie wątku jest jednak realizowane za pomocą metody `Execute`.

## Tworzenie nowej klasy wątku

Jeżeli chcemy utworzyć nowy wątek, jedynym rozwiązaniem jest zadeklarowanie w kodzie programu nowej klasy, dziedziczącej po `TThread`. Klasę tę możemy samodzielnie wpisać bezpośrednio w kod programu lub skorzystać z kreatora Delphi.

Z menu *File* wybierz *New/Other*, co spowoduje otwarcie *Repozytorium* (o *Repozytorium* pisałem w rozdziale 4.). Wystarczy na zakładce *New* wybrać pozycję *Thread Object* (rysunek 8.2).

Po naciśnięciu przycisku *OK* zostaniesz poproszony o wpisanie nazwy klasy w odpowiednim oknie. Wpisz np. `TMojWatek`. Wówczas stworzony zostanie nowy moduł, a w nim deklaracja nowej klasy (patrz listing 8.1).



Rysunek 8.2. Okno Repozytorium

Listing 8.1. Kod źródłowy nowego modułu wygenerowanego przez Delphi

```

unit Unit2;

interface

uses
  Classes;

type
  TmojWatek = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;

implementation

{ Important: Methods and properties of objects in visual components can only be
  used in a method called using Synchronize, for example,

    Synchronize(UpdateCaption);

  and UpdateCaption could look like,

    procedure TmojWatek.UpdateCaption;
    begin
      Form1.Caption := 'Updated in a thread';
    end; }

  { TmojWatek }

procedure TmojWatek.Execute;
begin
  { Place thread code here }
end;

end.

```

Nowy moduł zawiera klasę `TMojWatek`, w której umieszczona jest jedna metoda (w sekcji `protected`). To właśnie w metodzie `Execute` należy umieścić właściwy kod wątku. Ponadto w module znajduje się ciekawy komentarz, który zostanie przeze mnie omówiony w dalszej części rozdziału.

W każdym bądź razie nie jest konieczne tworzenie nowego modułu dla klasy wątku. Nie jest także konieczne tworzenie samej klasy w taki sposób, w jaki to przedstawiłem. Równie dobrze można zadeklarować klasę samodzielnie.



### Wskazówka

Podczas samodzielnego deklarowania klasy dziedziczącej po `TThread` nie wolno zapominać o deklaracji metody `Execute`. Metoda `Execute` musi być umieszczona w sekcji `protected` i opatrzona dyrektywą `override`.

## Kilka instancji wątku

W każdej klasie wątku mogą być oczywiście deklarowane metody i właściwości — zupełnie tak samo, jakby to była zwykła klasa. Istnieje także możliwość uruchamiania kilku klas wątku jednocześnie! Powoduje to stworzenie dla każdej klasy osobnej instancji zmiennej i zarezerwowanie osobnego bloku pamięci.

Tworzenie wątku przedstawia się następująco:

```
TMojWatek.Create(False);
```

Po wywołaniu konstruktora klasy uruchamiany jest cały proces (metoda `Execute`), a to za sprawą parametru typu `Boolean` zawartego w konstruktorze. Jeżeli wartość tego parametru to `True`, uruchomienie wątku nastąpi dopiero po wywołaniu metody `Resume`.



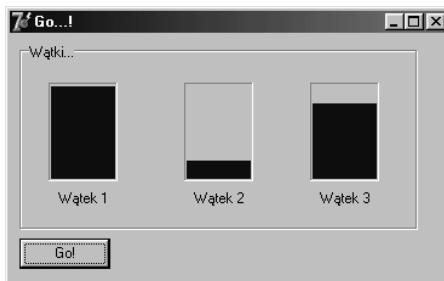
### Wskazówka

Nie zaleca się uruchamiania w tym samym czasie dużej ilości wątków w ramach tego samego procesu. Zalecana ilość to 16 wątków w ramach jednego procesu.

## Tworzenie klasy

Przedstawię Ci teraz przykładowy program tworzący trzy wątki pochodne, które będą działać jednocześnie. Ich działanie nie spowoduje zablokowania programu — użytkownik będzie mógł przeciągać okno programu, minimalizować go itp.

Przykładowy program będzie banalny i raczej niepraktyczny. Wątek wylosuje jakąś liczbę z zakresu od 0 do 999 i wykona pętlę `for` od liczby 1 do tej wylosowanej wartości. Pętla będzie wykonywana tylko przez jakiś czas —dzięki spowalnianiu (funkcja `Sleep`). Przerwa między kolejnymi iteracjami to 100 milisekund. Program przedstawiony został na rysunku 8.3.



Rysunek 8.3. Działanie trzech wątków naraz

Postęp wykonywania pętli przedstawiony jest za pomocą komponentów TProgressBar.

## Kod klasy

Deklaracja klasy jest dość prosta — wykorzystujemy jedną metodę, konstruktor oraz dwie właściwości:

```
TGoThread = class(TThread)
private
    FV : Integer; // wylosowana liczba
    FCounter : Integer; // numer wątku
protected
    procedure Execute; override;
public
    constructor Create(Counter : Integer);
end;
```

Deklarowanie konstruktora przez programistę nie jest konieczne, lecz ja stworzyłem go ze względu na konieczność przekazania do klasy pewnego parametru, jakim jest numer wątku:

```
constructor TGoThread.Create(Counter: Integer);
begin
    inherited Create(False); // wywołanie wątku
    FCounter := Counter; // przypisanie wartości do zmiennej
end;
```

Na początku w konstruktorze wywołujemy konstruktor klasy bazowej. Następnie zmiennej FCounter przypisujemy wartość, która została podana wraz z parametrem konstruktora.

Oto, jak wygląda główna procedura — Execute:

```
procedure TGoThread.Execute;
var
    i : Integer;
begin
    FreeOnTerminate := True; // zwolnij po zakończeniu wątku

    Randomize;
    FV := Random(1000);
    { odnalezienie komponentu na formularzu }
    TProgressBar(MainForm.FindComponent('ProgressBar' + InttoStr(FCounter))).Max := FV;
```

```
for i := 0 to FV do
begin
  SSteep(10);
  TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Position := i;
end;
end;
```

Zwróć uwagę na przypisanie do właściwości `FreeOnTerminate` wartości `True`. Spowoduje to zwolnienie klasy po zakończeniu działania wątku.

Kolejne instrukcje są już ściśle związane z działaniem owego wątku. Ciekawą konstrukcją jest:

```
TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Max := FV;
```

Taki zapis umożliwia znalezienie na formularzu komponentu bez znajomości jego nazwy. Wystarczy jedynie podać nazwę komponentu w parametrze funkcji `FindComponent`. Kompletny kod źródłowy modułu znajduje się w listingu 8.2.

### Listing 8.2. Kod źródłowy modułu

```
unit MainForm;

interface

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

type
  TMainForm = class(TForm)
    gbHome: TGroupBox;
    ProgressBar1: TProgressBar;
    ProgressBar2: TProgressBar;
    ProgressBar3: TProgressBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    btnGo: TButton;
    procedure btnGoClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TGoThread = class(TThread)
  private
    FV : Integer; // wylosowana liczba
    FCounter : Integer; // numer wątku
  protected
    procedure Execute; override;
  public
    constructor Create(Counter : Integer);
  end;

var
  MainForm: TMainForm;
implementation
```

```
{ $R *.dfm }

procedure TMainForm.btnGoClick(Sender: TObject);
begin
{ utworzenie trzech wątków }
  TGoThread.Create(1);
  TGoThread.Create(2);
  TGoThread.Create(3);
end;

{ TGoThread }

constructor TGoThread.Create(Counter: Integer);
begin
  inherited Create(False); // wywołanie wątku
  FCounter := Counter; // przypisanie wartości do zmiennej
end;

procedure TGoThread.Execute;
var
  i : Integer;
begin
  FreeOnTerminate := True; // zwolnij po zakończeniu wątku

  Randomize;
  FV := Random(1000);
  { odnalezienie komponentu na formularzu }
  TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Max := FV;

  for i := 0 to FV do
  begin
    Sleep(10);
    TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Position := i;
  end;
end;

end.
```

---

## Wznawianie i wstrzymywanie wątków

Klasa `TThread` posiada metody, dzięki którym możemy wznowić lub zatrzymać wykonywanie danego wątku. Zadanie wstrzymywania i wznowiania wykonywania danego wątku realizuje metoda `Suspend` i `Resume`.

```
var
  MojWatek : TMojWatek;
begin
  MojWatek := TMojWatek.Create(True);
  MojWatek.Resume; // uruchomienie wątku
  MojWatek.Suspend; // wstrzymanie
end;
```

O tym, czy wątek jest w danym momencie uruchomiony, informuje właściwość `Suspended`. Przyjmuje ona wartość `True`, jeżeli wątek jest wstrzymany, natomiast w przeciwnym wypadku — `False`.

## Priorytet wątku

Wątkom można nadawać różne priorytety, zależnie od „ważności” zadania, jakie dany wątek wykonuje. Nadając operacji wyższy priorytet, uzyskujesz pewność, że procesor przydzieli czas wykonania właśnie naszemu wątkowi.

Priorytet nadaje się wątkom poprzez właściwość `Priority`, wykorzystując takie oto wartości: `tpIdle`, `tpLowest`, `tpLower`, `tpNormal`, `tpHigher`, `tpHighest`, `tpTimeCritical`. Najniższym priorytetem jest `tpIdle` — taki wątek jest wykonywany wtedy, gdy żaden inny proces nie wymaga użycia procesora (np. wygaszacze ekranu). Natomiast priorytet `tpTimeCritical` otrzymują procesy, które wymagają użycia procesora w trybie natychmiastowym.

```
MojWatek.Priority := 1pHigher; // nadanie wyższego priorytetu
```



### Wskazówka

Nie należy zbyt przesadzać z nadawaniem wątkom priorytetów. Zalecane jest zachowanie priorytetu normalnego (`tpNormal`). Nadanie wątkowi zbyt wysokiego priorytetu może spowodować nieprawidłowe działanie pozostałych programów uruchomionych w tym samym czasie.

## Synchronizacja

Należy rozważyć jeszcze jedną sytuację, a mianowicie uruchamianie kilku wątków w tym samym czasie. Jeżeli owe wątki modyfikują właściwości lub dokonują jakichkolwiek innych zmian w bibliotece VCL, może dojść do kolizji. Dotyczy to np. przypadku, gdy owe wątki muszą pobierać jakieś wartości z komponentów i jednocześnie je modyfikować. W tym celu zalecane jest użycie metody `Synchronize` klasy `TThread`.

```
procedure TGoThread.Execute;  
begin  
  Synchronize(SetProprties);  
end;
```

W ten sposób wątek wywołuje metodę `Synchronize`, w której podana została nazwa procedury do wykonania — `SetProprties`. Dzięki temu masz pewność, że spośród kilku uruchomionych w danym momencie funkcji tylko jedna będzie wykonywana w danym czasie i tylko ona będzie mogła dokonywać zmian w bibliotece VCL.



## Treść komentarza

Na początku rozdziału chcąc stworzyć nowy wątek, użyłeś *Repozytorium*. W module, który został utworzony przez Delphi, widniał taki komentarz:

```
{ Important: Methods and properties of objects in visual components can only be used in a method called using Synchronize, for example,
```

```
    Synchronize(UpdateCaption);
```

and UpdateCaption could look like,

```
    procedure TMyWatek.UpdateCaption;
    begin
        Form1.Caption := 'Updated in a thread';
    end; }
```

Oto jego tłumaczenie:

**Ważne!** Metody i właściwości obiektów VCL mogą być użyte jedynie w metodzie wywołanej za pomocą *Synchronize*.

Pamiętasz jeszcze program, który prezentowałem Ci kilka stron wcześniej (trzy wątki modyfikujące właściwość *Position* komponentu *TProgressBar*)? W listingu 8.3 zaprezentowany jest program wykorzystujący metody *Synchronize*, dzięki której w jednym momencie dostęp do komponentów VCL ma tylko jeden wątek.

### Listing 8.3. Dostęp do VCL ma tylko jeden wątek

---

```
unit MainFrm;

interface

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

type
    TMainForm = class(TForm)
        gbHome: TGroupBox;
        ProgressBar1: TProgressBar;
        ProgressBar2: TProgressBar;
        ProgressBar3: TProgressBar;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        btnGo: TButton;
        procedure btnGoClick(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
```

```
TGoThread = class(TThread)
private
    FV : Integer; // wylosowana liczba
    FCounter : Integer; // numer wątku
    procedure SetProprties;
protected
    procedure Execute; override;
public
    constructor Create(Counter : Integer);
end;

var
    MainForm: TMainForm;

implementation

{$R *.dfm}

procedure TMainForm.btnGoClick(Sender: TObject);
begin
    { utworzenie trzech wątków }
    TGoThread.Create(1);
    TGoThread.Create(2);
    TGoThread.Create(3);
end;

{ TGoThread }

constructor TGoThread.Create(Counter: Integer);
begin
    inherited Create(False); // wywołanie wątku
    FCounter := Counter; // przypisanie wartości do zmiennej
end;

procedure TGoThread.Execute;
begin
    Synchronize(SetProprties);
end;

procedure TGoThread.SetProprties;
var
    i : Integer;
begin
    FreeOnTerminate := True; // zwolnij po zakończeniu wątku

    Randomize;
    FV := Random(1000);
    { odnalezienie komponentu na formularzu }
    TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Max := FV;

    for i := 0 to FV do
    begin
        Sleep(10);
        TProgressBar(MainForm.FindComponent('ProgressBar' + IntToStr(FCounter))).Position := i;
    end;
end;

end.
```

---

## Zdarzenia klasy TThread

Chciałem w tym miejscu wspomnieć jeszcze o zdarzeniach, a właściwie o jednym zdarzeniu, znajdującym się w klasie TThread. To zdarzenie to OnTerminate, które może się przydać, jeżeli chcemy przechwycić zakończenie działania wątku.

Najlepiej przypisać do zdarzenia odpowiednią procedurę w momencie utworzenia klasy, czyli w konstruktorze Create.

```
constructor TMainThread.Create;  
begin  
    inherited Create(False);  
    OnTerminate := MyTerminate; // przypisanie procedury do zdarzenia  
end;  
  
procedure TMainThread.Execute;  
begin  
    { kod wątku }  
end;  
  
procedure TMainThread.MyTerminate(Sender: TObject);  
begin  
    { kod zdarzenia }  
end;
```

### Wskazówka

Użycie metody DoTerminate klasy TThread powoduje wywołanie zdarzenia OnTerminate.

## Przykład: wyszukiwanie wielowątkowe

Na rysunku 8.1 przedstawione zostało działanie wielowątkowej wyszukiwarki. Co prawda cały kod źródłowy umieszczony jest na dołączonej do książki płycie CD-ROM, lecz warto omówić jego działanie. Zaprezentuję więc proces tworzenia takiego programu krok po kroku.

### Jak to działa?

Sam proces wyszukiwania opisany został w poprzednim rozdziale. Nasz program będzie się różnił tym, że zadaniem każdego wątku będzie wyszukanie plików na innej partycji, co w konsekwencji potrwa krócej niż w sytuacji, gdyby miałyby to być realizowane w ramach jednego wątku.

## Wyszukiwanie

Procedura wyszukiwania jest podobna do tej, którą prezentowałem w poprzednim rozdziale. Poniżej przedstawiona procedura jest *rekurencyjna*, czyli — jak zapewne pamiętasz — realizuje przeszukiwanie również w podkatalogach.

```

procedure Search(StartDir : String);
var
  SR, DR : TSearchRec;
  Found, FoundFile : Integer;

  { ta procedura sprawdza, czy na końcu zmiennej znajduje się znak \ - jeżeli
    tak, nic nie jest wykonywane; jeżeli tego znaku brak, zostaje on dodany... }
  function IsDir(Value : String) : String;
  begin
    if Value[Length(Value)] <> '\' then // jeżeli na końcu znajdziesz znak
      Result := Value + '\' else Result := Value; // dodaj go... w przeciwnym wypadku nie
      ↪wykonuj nic
    end;

begin
  Found := FindFirst(IsDir(StartDir) + '*.*', faDirectory, DR); // następuje pobieranie
  ↪katalogów z podanej lokalizacji
  while Found = 0 do // pętelka
  begin
    if ((DR.Attr and faDirectory) = faDirectory) and // sprawdza, czy pozycja jest katalogiem
      ((DR.Name <> '.') and (DR.Name <> '..')) then
    begin
      MainForm.StatusBar.SimpleText := IsDir(StartDir) + DR.Name + '*.*'; //
      ↪na komponencie wyświetl aktualnie przeszukiwany katalog
      if Pos(FFileName, DR.Name) > 0 then // sprawdź, czy w nazwie jest szukany ciąg znaków
        MainForm.lbResults.Items.Add(IsDir(StartDir) + DR.Name);

      { pobierz na razie wszystkie pliki z danego katalogu - potem je przeanalizujemy }
      FoundFile := FindFirst(IsDir(StartDir) + DR.Name + '*.*', faAnyFile, SR);
      while FoundFile = 0 do
      begin
        if ((SR.Name <> '.') and (SR.Name <> '..')) then //
          if Pos(FFileName, SR.Name) > 0 then // następuje sprawdzenie, czy plik nie
            ↪zawiera części szukanego ciągu
              MainForm.lbResults.Items.Add(IsDir(StartDir) + DR.Name + '\' + SR.Name);

        FoundFile := FindNext(SR); // kontynuuj przeszukiwanie
      end;
      FoundClose(SR); // zakończ

      Search(IsDir(StartDir) + DR.Name); // tutaj następuje rekurencja
    end;
    Found := FindNext(DR); // kontynuuj
  end;
  FoundClose(DR);
end;

```

W powyższej procedurze zagnieżdżona jest kolejna — `IsDir`. Sprawdza ona, czy na końcu ścieżki znajduje się znak *backslash* (`\`). Jeżeli go nie ma, dodawany jest ten znak, gdyż wymagany jest on do prawidłowego działania funkcji *rekurencyjnej*.

Znalezienie konkretnego pliku jest kwalifikowane za pomocą funkcji `Pos`. Jeżeli dany plik lub katalog zawiera szukany ciąg znaków (a sprawdza to funkcja `Pos`), następuje wyświetlenie ścieżki w komponencie `TListBox`.

## Obliczanie czasu przeszukiwania

Do obliczenia czasu potrzebnego na przeszukanie konkretnej partycji skorzystamy z funkcji `GetTickCount`. Funkcja ta zwraca ilość milisekund, jakie upłynęły od czasu uruchomienia systemu. Wystarczy więc pobrać wartość początkową przed wywołaniem wątku oraz wartość końcową po zakończeniu wykonywania operacji — np. przy zakończeniu wątku:

```
destructor TSearchThread.Destroy;
begin
  Stop := GetTickCount; // pobierz czas zakończenia
  Total := Stop - Start; // odejmij czas startu od czasu zakończenia
  Total := Total / 1000; // podziel przez 1000, aby uzyskać liczbę sekund

  { wyświetl na komponencie czas wyszukiwania na danym dysku }
  MainForm.lbEnd.Items.Add(FDrive + '\ - ' + CurrToStr(Total) + ' sek. ');
  inherited;
end;
```

Zmienna `Start` jest uprzednio pobraną wartością, określającą czas rozpoczęcia wątku. Finalną wartość `Total` należy podzielić przez 1 000, aby uzyskać liczbę sekund.

## Kod źródłowy modułu

Pełny kod źródłowy modułu znajduje się w listingu 8.4.

### Listing 8.4. Kod źródłowy modułu

---

```
{
  Copyright (c) 2002 by Adam Boduch
}

unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Buttons, ComCtrls;

type
  TMainForm = class(TForm)
    lbFileName: TLabelEdit;
```

```

    btnFind: TBitBtn;
    Bevel1: TBevel;
    lbResults: TListBox;
    StatusBar: TStatusBar;
    lbEnd: TListBox;
    Label1: TLabel;
    ProgressBar: TProgressBar;
    procedure btnFindClick(Sender: TObject);
end;

TSearchThread = class(TThread)
private
    Start, Stop : Integer; // wartości te przechowują czas rozpoczęcia i zakończenia
    ─działania wątku
    Total : Currency; // wartość całkowitego czasu przeszukania
    FFileName : String; // nazwa pliku do odnalezienia
    FDrive : Char; // dysk, na którym odbędzie się szukanie
    procedure MyOnTerminate(Sender: TObject); // obsługa zdarzenia OnTerminate
public
    constructor Create(const FileName : String; Drive : Char); // konstruktor dla klasy
    destructor Destroy; override; // destruktor dla klasy
    procedure SearchInDrive; // procedura poszukiwawcza
protected
    procedure Execute; override;
end;

var
    MainForm: TMainForm;
    SearchThread : TSearchThread;

implementation

{$R *.dfm}

constructor TSearchThread.Create(Const FileName : String; Drive : Char);
begin
    inherited Create(False); // wywołanie konstruktora klasy bazowej
    FreeOnTerminate := True; // zwolnij przy zakończeniu
    OnTerminate := MyOnTerminate; // przypisz procedurę zdarzenia
    FFileName := FileName; // nazwa pliku do znalezienia
    FDrive := Drive; // dysk
    Start := GetTickCount; // pobierz czas startu (w milisekundach)
end;

destructor TSearchThread.Destroy;
begin
    Stop := GetTickCount; // pobierz czas zakończenia
    Total := Stop - Start; // odejmij czas startu od czasu zakończenia
    Total := Total / 1000; // podziel przez 1000, aby uzyskać liczbę sekund

    { wyświetl na komponencie czas wyszukiwania na danym dysku }
    MainForm.lbEnd.Items.Add(FDrive + ':\' - ' + CurrToStr(Total) + ' sek. ');
    inherited;
end;

procedure TSearchThread.SearchInDrive;

```

```

procedure Search(StartDir : String);
var
  SR, DR : TSearchRec;
  Found, FoundFile : Integer;

  { ta procedura sprawdza, czy na końcu zmiennej znajduje się znak \ - jeżeli
    tak, nic nie jest wykonywane; jeżeli tego znaku brak, zostaje on dodany... }
  function IsDir(Value : String) : String;
  begin
    if Value[Length(Value)] <> '\' then // jeżeli na końcu znajdziesz znak
      Result := Value + '\' else Result := Value; // dodaj go... w przeciwnym wypadku nie
      ↪wykonuj nic
    end;
end;

begin
  Found := FindFirst(IsDir(StartDir) + '*.*', faDirectory, DR); // następuje pobieranie
  ↪katalogów z podanej lokalizacji
  while Found = 0 do // pętelka
  begin
    if ((DR.Attr and faDirectory) = faDirectory) and // sprawdza, czy pozycja jest katalogiem
      ((DR.Name <> '.') and (DR.Name <> '..')) then
    begin
      MainForm.StatusBar.SimpleText := IsDir(StartDir) + DR.Name + '*.*'; // na
      ↪komponencie wyświetl aktualnie przeszukiwany katalog
      if Pos(FFileName, DR.Name) > 0 then // sprawdź, czy w nazwie jest szukany ciąg znaków
        MainForm.lbResults.Items.Add(IsDir(StartDir) + DR.Name);

      { pobierz na razie wszystkie pliki z danego katalogu - potem je przeanalizujemy }
      FoundFile := FindFirst(IsDir(StartDir) + DR.Name + '*.*', faAnyFile, SR);
      while FoundFile = 0 do
      begin
        if ((SR.Name <> '.') and (SR.Name <> '..')) then //
          if Pos(FFileName, SR.Name) > 0 then // następuje sprawdzenie, czy plik nie
            ↪zawiera części szukanego ciągu
              MainForm.lbResults.Items.Add(IsDir(StartDir) + DR.Name + '\' + SR.Name);

        FoundFile := FindNext(SR); // kontynuuj przeszukiwanie
      end;
      FindClose(SR); // zakończ

      Search(IsDir(StartDir) + DR.Name); // tutaj następuje rekurencja
    end;
    Found := FindNext(DR); // kontynuuj
  end;
  FindClose(DR);
end;

begin
  Search(FDrive + '\'); // rozpocznij wyszukiwanie na danym dysku
end;

procedure TSearchThread.Execute;
begin
  (SearchInDrive); // wywołaj procedurę...
end;

```

```
procedure TSearchThread.MyOnTerminate(Sender: TObject);
begin
{ podczas kończenia wyszukiwania wyświetl na komponencie ilość odnalezionych pozycji }
  MainForm.ProgressBar.Position := MainForm.ProgressBar.Position + 1;
  MainForm.StatusBar.SimpleText := 'Znaleziono: ' +
  IntToStr(MainForm.LbResults.Items.Count) + ' plików...';
end;

procedure TMainForm.btnFindClick(Sender: TObject);
var
  i : Integer;
  DriveType : Integer;
begin
  LbResults.Clear; // wyczyść komponent
  LbEnd.Clear; // wyczyść komponent
  ProgressBar.Max := 0; // ustaw wartość maksymalną na 0
  ProgressBar.Position := 0; // pozycja na 0
  for I := Ord('A') to Ord('Z') do // pętla po wszystkich dyskach
  begin
    DriveType := GetDriveType(PChar(chr(i) + ':\')); // pobierz informacje o dysku

    if not (DriveType = 0) and not (DriveType = 1) then // jeżeli dysk istnieje
    begin
      ProgressBar.Max := ProgressBar.Max + 1; // zwiększ właściwość maks. o jeden
      SearchThread := TSearchThread.Create(LeFileName.Text, Chr(i)); // wywołaj wątek
      ─z literą dysku jako początkowy parametr
    end;
  end;
end;

end.
```

---

## Podsumowanie

W niniejszym rozdziale przedstawiłem Ci zasadę działania wątków. Myślę, że po dokładniejszym zapoznaniu się z tym zagadnieniem nie wygląda ona tak strasznie, tym bardziej, że nie jesteśmy zmuszeni do korzystania z funkcji WinAPI, ale używamy wygodnej klasy VCL. Na pewno nieraz będziesz w swojej aplikacji wykorzystywał wątki...