

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 2005. 303 gotowe rozwiązania

Autorzy: Jacek Matulewski,
Sławomir Orłowski, Michał Zieliński
ISBN: 83-7361-923-2
Format: B5, stron: 648



Najnowsza wersja Delphi – jednego z flagowych produktów firmy Borland, łączy w sobie trzy znane środowiska programistyczne: tradycyjne Delphi, Delphi dla .NET oraz C# Builder. Zawarcie w jednym systemie tak wielu możliwości pozwala programistom tworzyć różne rodzaje aplikacji – zarówno te, do których pisania przyzwyczaili się, korzystając z poprzednich wersji Delphi, jak i programy wykorzystujące zyskującą coraz większą popularność platformę .NET. Dzięki możliwości przenoszenia kodu do środowiska Kylix pracującego pod kontrolą systemu operacyjnego Linux Delphi jest jedną z najbardziej uniwersalnych platform dla programistów i projektantów rozbudowanych aplikacji przeznaczonych do działania w sieci.

„Delphi 2005. 303 gotowe rozwiązania” to książka dla programistów zainteresowanych tworzeniem rozbudowanych aplikacji i wykorzystywaniem w tym celu wszystkich możliwości oferowanych przez najnowszą wersję środowiska Delphi. Opisuje zasady tworzenia programów wykorzystujących biblioteki systemowe Windows i interfejs WinAPI oraz przedstawia rozwiązania problemów, na jakie można się natknąć pisząc aplikację wykraczającą poza możliwości oferowane przez standardowe komponenty dołączane do Delphi. Książka jest zbiorem funkcji, klas i sztuczek, za pomocą których można rozszerzyć możliwości biblioteki VCL, VCL.NET i Windows Forms, sięgając głębiej do zasobów systemu.

- Elementy środowiska Delphi 2005
- Programowanie obiektowe w Delphi 2005
- Obsługa wyjątków w języku Object Pascal
- Korzystanie z informacji zapisanych w rejestrze
- Obsługa systemu plików
- Tworzenie wygaszaczy ekranu
- Projektowanie własnych komponentów
- Stosowanie interfejsu WinAPI w aplikacjach
- Wykorzystywanie możliwości multimedialnych Windows
- Obsługa mechanizmów OLE oraz kontrolek ActiveX
- Elementy biblioteki Indy
- Tworzenie aplikacji sieciowych
- Grafika i biblioteki DirectX.NET

Jeśli chcesz tworzyć profesjonalne aplikacje, wykorzystując Delphi, przeczytaj tę książkę – znajdziesz tu wiadomości, dzięki którym unikniesz wielu problemów.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl



Spis treści

Wstęp	15
Część I Delphi 2005 — nowoczesne środowisko programistyczne	19
Rozdział 1. Krótki przewodnik po Delphi 2005	21
Typy projektów oferowane przez Delphi 2005	22
Projekt 1. VCL Forms Application — Delphi for Win32	22
Projekt 2. VCL Forms Application — Delphi for .NET	24
Projekt 3. Windows Forms Application — Delphi for .NET	26
Projekt 4. Windows Forms Application — C#	28
Projekt 5. ASP.NET Web Application — Delphi for .NET	28
Rozdział 2. Programowanie obiektowe w języku Object Pascal. Wyjątki	31
Interfejs i implementacja klasy	32
Projekt 6. Deklaracja klasy. Pola i metody	32
Projekt 7. Konstruktor — inicjowanie obiektu	35
Projekt 8. Własności	36
Projekt 9. Metody i funkcje definiujące operacje na obiektach klasy	38
Wyjątki — zgłaszanie błędów	40
Projekt 10. Zgłaszanie i obsługa wyjątków	40
Projekt 11. Własne klasy wyjątków	42
Konwersje i operacje na obiektach klasy	42
Projekt 12. Metody przeciążone dla argumentów typu CTyp	42
Projekt 13. Alternatywne konstruktory określające stałą urojoną i kopiujące obiekt	44
Projekt 14. Funkcje dwuargumentowe implementujące operacje arytmetyczne na liczbach zespolonych	45
Projekt 15. Funkcja Exp i funkcje trygonometryczne dla typu TComplex	46
Projekt 16. Konwersje na typ rzeczywisty	48
Projekt 17. Konwersja na łańcuch. Metoda ToString i funkcja ComplexToStr	49
Krótka uwaga o dziedziczeniu i obsłudze wyjątków	50
Projekt 18. Dziedziczenie klas	50
Projekt 19. Więcej o obsłudze wyjątków. Konstrukcja try-finally	52

Rozdział 3. Edycja, kompilacja i debugowanie kodu w Delphi 2005	55
Instalacja i rejestracja Delphi 2005 Architect Trial	55
Pobieranie klucza rejestracji	56
Instalacja Delphi 2005 Architect Trial	59
Rejestracja Delphi Architect Trial	64
Konfiguracja środowiska	64
Włączenie okna postępu kompilacji	65
Automatyczne zapisywanie plików projektu	66
Jedno czy wiele okien	66
Formatowanie automatycznie tworzonego kodu C#	67
Dostosowanie menu File, New	68
Edytor kodu	69
Opcje edytora	69
Definiowanie bloków	70
Zmiana nazwy klasy za pomocą narzędzia Refactoring	71
Pliki projektu	72
Debugowanie kodu	73
Dystrybucja programów	78
Złote myśli	79
Część II Praktyka projektowania aplikacji	81
Rozdział 4. Delphi z bibliotekami VCL i VCL.NET	83
Sztuczki z oknami	83
Projekt 20. Łagodne znikanie okna przy jego zamknięciu	83
Projekt 21. Dowolny kształt formy przy wykorzystaniu własności TransparentColor	84
Projekt 22. Zamykanie aplikacji naciśnięciem klawisza Esc	86
Projekt 23. Aby okno wyglądało identycznie w systemach z różną wielkością czcionki	86
Projekt 24. Aby ograniczyć rozmiary formy	87
Projekt 25. Przeciąganie formy myszą za dowolny jej punkt	87
Projekt 26. Wizytówka programu (splash screen)	89
Rejestr systemu Windows	90
Projekt 27. Przechowywanie położenia i rozmiaru okna w rejestrze	91
Projekt 28. Aby uruchamiać aplikację po wlogowaniu się użytkownika	94
Projekt 29. Umieszczanie informacji o zainstalowanym programie (aplet Dodaj/Usuń programy)	96
Projekt 30. Gdzie jest katalog z moimi dokumentami?	101
Projekt 31. Dodawanie pozycji do menu kontekstowego związanego z zarejestrowanym typem pliku	103
Pliki INI	105
Projekt 32. Jak umieścić na pulpicie lub w menu Start skrót do strony WWW?	105
Projekt 33. Jak odczytać i zmienić rozmiar formy?	106
Edytor	107
Projekt 34. Wczytywanie pliku ze wskazanego przez użytkownika pliku. Okno dialogowe TOpenDialog	108
Projekt 35. Okno dialogowe wykorzystywane przy zapisywaniu dokumentu do pliku	110
Projekt 36. Przeszukiwanie tekstu. Okno dialogowe TFindDialog	111
Projekt 37. Formatowanie fragmentów tekstu w komponencie TRichEdit. Okno dialogowe TFontDialog	114

Projekt 38. Formatowanie poszczególnych atrybutów czcionki	114
Projekt 39. Powiadamianie o niezapisanych dokumentach	115
Projekt 40. Wczytywanie dokumentu z pliku wskazanego jako parametr linii komend	117
Projekt 41. Jak dodać aplikację do listy edytorów dostępnych z menu kontekstowego plików o danym rozszerzeniu?	118
Mechanizm drag & drop	119
Projekt 42. Mechanizm przenoszenia i upuszczania w obrębie aplikacji	119
Projekt 43. Uelastycznianie kodu. Wykorzystanie referencji Sender	122
Projekt 44. Przenoszenie wielu elementów	124
Projekt 45. Obsługa pliku przeniesionego na formę z zewnętrznej aplikacji	125
Konwersje oraz operacje na łańcuchach i dacie	125
Projekt 46. Konwersja między liczbą i łańcuchem. Liczby w TEdit. Konwersja z formatem ustalonym przez programistę	125
Projekt 47. Prezentowanie daty i czasu w przyjaznej formie	128
Projekt 48. Rozkładanie daty i czasu na elementy	130
Projekt 49. Jak przekonwertować datę utworzenia pliku na datę typu TDateTime i potem na łańcuch?	131
Projekt 50. Jak przekształcić łańcuch na pisany wielkimi lub małymi literami?	131
Pliki i system plików	132
Projekt 51. Jak za pomocą komponentów TDriveComboBox, TDirectoryListBox, TFilterComboBox i TFileListBox stworzyć prostą przeglądarkę plików?	132
Projekt 52. Przeglądanie katalogów w FileListBox	133
Projekt 53. Tworzenie pliku tekstowego	134
Projekt 54. Odczytywanie plików tekstowych	134
Projekt 55. Dopisywanie do plików tekstowych. Pliki rejestrwania zdarzeń	135
Projekt 56. Operacje na plikach i katalogach	137
Projekt 57. Odnajdywanie pliku i odczytywanie jego własności	139
Projekt 58. Komponent TFileDetailsStringGrid — lista plików z informacjami szczegółowymi	141
Projekt 59. Jak z łańcucha wyodrębnić nazwę pliku, jego rozszerzenie lub katalog, w którym się znajduje?	147
Projekt 60. Jak sprawdzić ilość wolnego miejsca na dysku?	148
Projekt 61. Wczytywanie drzewa katalogów i plików	149
Projekt 62. Wczytywanie drzewa katalogów i plików w osobnym wątku	151
Projektowanie wygaszaczy ekranu	153
Projekt 63. Wygaszacz ekranu	153
Projekt 64. Podgląd wygaszacza na zakładce Wygaszacz ekranu apletu Właściwości: Ekran	159
Projekt 65. Konfiguracja wygaszacza ekranu	161
Drukowanie	166
Projekt 66. Drukowanie tekstu znajdującego się w komponencie TRichEdit. Okno dialogowe TPrintDialog	167
Projekt 67. Wybór domyślnej drukarki z poziomu kodu aplikacji	167
Projekt 68. Drukowanie tekstu przechowywanego w TStrings w trybie graficznym	169
Projekt 69. Jak wydrukować obraz z pliku?	171
Projekt 70. Drukowanie tekstu przechowywanego w TStrings w trybie tekstowym	173

Rozdział 5. Współpraca Delphi z biblioteką Windows Forms z platformy Microsoft .NET	175
Sztuczki z oknami	175
Projekt 71. Łagodne znikanie okna aplikacji przy zamknięciu	175
Projekt 72. Wykorzystanie własności TransparencyKey	176
Projekt 73. Dowolny kształt formy	177
Projekt 74. Zamykanie aplikacji po naciśnięciu klawisza Esc	179
Projekt 75. Ograniczenie rozmiarów formy	179
Projekt 76. Przeciąganie formy myszą za dowolny punkt	179
Projekt 77. Tworzenie okna powitalnego (Splash Screen)	181
Rejestr Systemu Windows	182
Projekt 78. Przechowywanie położenia i rozmiaru okna w rejestrze	183
Projekt 79. Uruchomienie aplikacji po zalogowaniu się użytkownika	185
Projekt 80. Umieszczenie informacji o zainstalowanym programie (Dodaj/Usuń programy)	187
Projekt 81. Gdzie jest katalog z moimi dokumentami? Klasa Environment	191
Projekt 82. Dodawanie nowych wpisów do menu kontekstowego	192
Edytor	193
Projekt 83. Wczytywanie pliku przez użytkownika	193
Projekt 84. Zapisywanie dokumentu do pliku za pomocą okna dialogowego	195
Projekt 85. Przeszukiwanie tekstu	196
Projekt 86. Formatowanie zaznaczonych fragmentów tekstu w komponencie RichTextBox. Okno dialogowe FontDialog	197
Projekt 87. Formatowanie poszczególnych własności czcionki	198
Projekt 88. Powiadamianie o niezapisanych dokumentach	199
Projekt 89. Wczytywanie dokumentu z pliku wskazanego jako parametr linii komend	200
Mechanizm drag & drop	201
Projekt 90. Mechanizm przenoszenia i upuszczania w obrębie aplikacji	201
Projekt 91. Uelastycznienie kodu. Wykorzystanie referencji Sender	203
Projekt 92. Przenoszenie wielu elementów	205
Konwersje oraz operacje na łańcuchach i dacie	206
Projekt 93. Konwersje między łańcuchem a liczbą	206
Projekt 94. Prezentowanie daty i czasu w przyjaznej formie	208
Projekt 95. Przekształcanie łańcucha na łańcuch pisany dużymi bądź małymi literami	211
Pliki i systemy plików	211
Projekt 96. Przeglądarka plików	211
Projekt 97. Tworzenie pliku tekstowego	214
Projekt 98. Odczytywanie plików tekstowych	215
Projekt 99. Dopisywanie do plików tekstowych. Pliki rejestrowania zdarzeń	215
Projekt 100. Operacje na plikach	217
Projekt 101. Odnajdywanie pliku	218
Projekt 102. Jak wyodrębnić nazwę pliku bądź katalogu?	219
Projekt 103. Jak sprawdzić ilość wolnego miejsca na dysku?	219
Projekt 104. Wczytywanie drzewa plików i katalogów	221
Projekt 105. Automatyczne śledzenie zmian na dysku	223
Drukowanie	223
Projekt 106. Procedura drukowania tekstu z komponentu RichTextBox	223
Inne	225
Projekt 107. Ikona w zasobniku systemowym	225
Projekt 108. Dźwięk w bibliotece Windows Forms	229
Projekt 109. Informacje o systemie i aplikacji	229

Część III Projektowanie komponentów	233
Rozdział 6. Przykład komponentu graficznego VCL/VCL.NET	235
Grafika. Rysowanie linii	235
Projekt 110. Rysowanie linii	236
Projekt 111. O wyborze koloru za pomocą komponentu TColorDialog oraz wskaźnikach	238
Kolorowy pasek postępu	240
Projekt 112. Przygotowanie komponentu	240
Projekt 113. Testowanie kolorowego paska postępu. Dynamiczne tworzenie komponentu	246
Projekt 114. Upublicznianie wybranych własności i zdarzeń chronionych w klasie bazowej	247
Projekt 115. Definiowanie zdarzeń na przykładzie OnPositionChanged	249
Projekt 116. Ikona komponentu	250
Projekt 117. Aby stworzyć projekt pakietu dla Win32	252
Projekt 118. Instalowanie komponentu VCL dla Win32	253
Projekt 119. Aby stworzyć projekt pakietu przeznaczonego dla platformy .NET ..	254
Projekt 120. Instalacja komponentu .NET	256
Projekt 121. Automatyczna zmiana koloru. Testowanie zdarzenia OnPositionChanged	257
Rozdział 7. Kontrolka Windows Forms	259
Część statyczna kontrolki	259
Projekt 122. Tworzenie projektu aplikacji testującej oraz interfejsu nowego komponentu	260
Projekt 123. Zdefiniuj prywatne pola wykorzystywane przez TFileListBox	261
Projekt 124. Stworzyć metodę pobierającą nazwy plików i podkatalogów znajdujących się we wskazanym katalogu	263
Projekt 125. Z pełnej ścieżki dostępu do plików wyłaniamy samą nazwę plików i katalogów	265
Projekt 126. Uwzględnić filtrowanie plików z maską określoną przez pole filtr ...	267
Część dynamiczna kontrolki	269
Projekt 127. Aby napisać metodę zmieniającą prezentowany w komponencie katalog	269
Własności — komponent w środowisku RAD	271
Projekt 128. Definiowanie własności komponentu	272
Zdarzenia — interaktywność komponentu	274
Projekt 129. Wiązanie zdarzeń komponentu i jego elementu	275
Projekt 130. Definiowanie zdarzenia SelectedFileChanged	276
Projekt 131. Definiowanie zdarzenia DirectoryPathChanged	278
Projekt 132. Definiowanie zdarzenia FileDoubleClicked	280
Nadpisywanie metody Refresh	282
Projekt 133. Zdefiniować publiczną metodę Refresh odświeżającą zawartość komponentu	283
Automatyczne śledzenie zmian na dysku	283
Projekt 134. Przygotować mechanizm pozwalający na śledzenie zmian dokonywanych w katalogu prezentowanym w komponencie TFileListBox	283
Kompilacja komponentu do postaci biblioteki DLL	286
Projekt 135. Aby stworzyć projekt biblioteki komponentu	287
Projekt 136. Aby zainstalować nowy komponent Windows Forms w środowisku Delphi	288

Część IV Programowanie Windows z wykorzystaniem WinAPI289

Rozdział 8. Kontrola stanu systemu	291
Zamykanie i wstrzymywanie systemu Windows	291
Projekt 137. Funkcja ExitWindowsEx	291
Projekt 138. Program służący do zamykania lub ponownego uruchamiania wszystkich wersji systemu Windows	296
Projekt 139. Funkcja InitiateSystemShutdown	297
Projekt 140. Program zamykający wybrany komputer w sieci	301
Projekt 141. Hibernowanie i wstrzymywanie systemu za pomocą funkcji SetSystemPowerState	303
Projekt 142. Program umożliwiający hibernację komputera lub jego „usypianie” ...	305
Projekt 143. Blokowanie dostępu do komputera	306
Projekt 144. Uruchamianie wygaszacza ekranu	306
Projekt 145. Odczytywanie informacji o baterii notebooka	307
Kontrola trybu wyświetlania karty graficznej	309
Projekt 146. Pobieranie dostępnych trybów pracy karty graficznej	310
Projekt 147. Identyfikowanie bieżącego trybu działania karty graficznej	312
Projekt 148. Zmiana trybu wyświetlania	313
Rozdział 9. Uruchamianie i kontrola aplikacji oraz ich okien	315
Uruchamianie, zamykanie i zmiana priorytetu aplikacji	315
Projekt 149. Uruchamianie aplikacji za pomocą WinExec	316
Projekt 150. Uruchamianie aplikacji za pomocą ShellExecute	318
Projekt 151. Przygotowanie e-maila za pomocą ShellExecute	319
Projekt 152. Zmiana priorytetu bieżącej aplikacji	319
Projekt 153. Sprawdzenie priorytetu bieżącej aplikacji	321
Projekt 154. Zmiana priorytetu innej aplikacji	322
Projekt 155. Zamykanie innej aplikacji	323
Projekt 156. Uruchamianie aplikacji za pomocą funkcji CreateProcess	324
Projekt 157. Wykrywanie zakończenia działania uruchomionej aplikacji	330
Projekt 158. Kontrola ilości instancji aplikacji na podstawie unikalnej nazwy klasy. Refactoring	331
Kontrola własności okna	334
Projekt 159. Lista otwartych okien	334
Projekt 160. Modyfikacja stanu okna bieżącej aplikacji	339
Projekt 161. Obecność aplikacji w pasku zadań	340
Projekt 162. Sygnał dźwiękowy	341
Numery identyfikacyjne procesu vs. uchwyt okna	341
Projekt 163. Jak zdobyć identyfikator procesu, znając uchwyt okna?	341
Projekt 164. Jak zdobyć uchwyt okna głównego, znając identyfikator procesu (wersja dla Win32)?	342
Projekt 165. Jak zdobyć uchwyt okna głównego, znając identyfikator procesu (wersja dla .NET)?	346
Projekt 166. Kontrola okna innej aplikacji	349
Projekt 167. Kontrola innej aplikacji — komponent TControlProcess	354
Projekt 168. Pakiet dla komponentu TControlProcess i instalacja komponentu	362
Okna o dowolnym kształcie	363
Projekt 169. Okno o kształcie koła	363
Projekt 170. Łączenie obszarów. Dodanie ikon z paska tytułu	364
Projekt 171. Okno z wizjerem	366
Projekt 172. Aby przenosić formę myszką pomimo usuniętego paska tytułu	366

Rozdział 10. Systemy plików, multimedia i inne funkcje WinAPI	367
Pliki i system plików — funkcje powłoki	367
Projekt 173. Jak za pomocą funkcji WinAPI powłoki systemu odczytać ścieżkę do katalogu specjalnego użytkownika?	368
Projekt 174. Tworzenie pliku skrótu .lnk (wersja Win32)	369
Projekt 175. Tworzenie pliku skrótu (wersja .NET)	372
Projekt 176. Odczyt i edycja skrótu .lnk (wersja Win32)	374
Projekt 177. Odczyt i edycja skrótu .lnk (wersja .NET)	375
Projekt 178. Umieszczenie skrótu na pulpicie	377
Projekt 179. Operacje na plikach i katalogach (kopiowanie, przenoszenie, usuwanie i zmiana nazwy)	377
Projekt 180. Jak usunąć plik, umieszczając go w koszu?	379
Projekt 181. Operacje na całym katalogu	380
Projekt 182. Odczytywanie wersji pliku .exe i .dll	381
Projekt 183. Monitorowanie zmian na dysku	385
Projekt 184. Jak dodać nazwę dokumentu do listy ostatnio otwartych dokumentów w menu Start?	386
Odczytywanie informacji o dysku	386
Projekt 185. Funkcja (wersja Win32)	386
Projekt 186. Funkcja (wersja .NET)	391
Projekt 187. Test funkcji	393
Projekt 188. Klasa	395
Projekt 189. Komponent	397
Ikona w obszarze powiadamiania (zasobniku)	402
Projekt 190. Dodawanie i usuwanie ikony do obszaru powiadamiania (zasobnika)	402
Projekt 191. Menu kontekstowe ikony	407
Projekt 192. Obsługa lewego przycisku myszy	410
Projekt 193. Uwagi na temat budowania komponentu	413
Internet	414
Projekt 194. Aby sprawdzić, czy komputer jest połączony z siecią	415
Projekt 195. Aby pobrać plik z Internetu	416
Projekt 196. Aby uruchomić domyślną przeglądarkę ze stroną	416
Projekt 197. Aby sprawdzić adres IP lub nazwę DNS wskazanego komputera (wersja Win32)	416
Projekt 198. Aby sprawdzić adres IP lub nazwę DNS wskazanego komputera (wersja .NET)	421
Projekt 199. Mapowanie dysków sieciowych	421
Multimedia (MCI)	422
Projekt 200. Aby wysunąć lub wsunąć tackę w napędzie CD/DVD	423
Projekt 201. Wykrywanie wysunięcia lub umieszczenia płyty w napędzie CD/DVD	425
Projekt 202. Sprawdzanie stanu wybranego napędu CD/DVD	425
Projekt 203. Aby zbadać, czy w napędzie jest płyta audio	426
Projekt 204. Kontrola napędu CDAudio	427
Projekt 205. Odtworzyć asynchronicznie plik WAV	428
Projekt 206. Jak wykryć obecność karty dźwiękowej?	429
Projekt 207. Kontrola poziomu głośności odtwarzania plików dźwiękowych	430
Projekt 208. Kontrola poziomu głośności CDAudio	431

Inne	431
Projekt 209. Podpinanie okna (wyświetlić zaprojektowaną przez nas formę w innym oknie)	432
Projekt 210. Malowanie na pulpicie	432
Projekt 211. Czy językiem Windows jest polski?	433
Projekt 212. Jak zablokować automatycznie uruchamiany wygaszacz ekranu?	433
Projekt 213. Zmiana tła pulpitu (wersja Win32)	434
Projekt 214. Zmiana tła pulpitu (wersja .NET)	435

Część V Wybrane technologie Windows437

Rozdział 11. Komunikaty Windows 439

Projekt 215. Lista komunikatów odbieranych przez kolejkę komunikatów aplikacji (TApplicationEvents.OnMessage)	440
Projekt 216. Filtrowanie zdarzeń	441
Projekt 217. Odczytywanie informacji dostarczanych przez komunikat	443
Projekt 218. Lista wszystkich komunikatów odbieranych przez okno (metoda WndProc)	444
Projekt 219. Metody obsługujące komunikaty nie umieszczane w kolejce komunikatów aplikacji. Wykrywanie zmiany położenia formy	446
Projekt 220. Wykrycie zmiany trybu pracy karty graficznej	447
Projekt 221. Wysyłanie komunikatów. Symulowanie zdarzeń	449
Projekt 222. Wysłanie komunikatu uruchamiającego wygaszacz ekranu	450
Projekt 223. Blokowanie zamknięcia sesji Windows	450
Projekt 224. Wykrycie włożenia lub wysunięcia płyty CD/DVD z napędu lub pamięci Flash z gniazda USB	451
Projekt 225. Wykorzystanie komunikatów do kontroli innej aplikacji na przykładzie WinAmp	452
Projekt 226. Przenoszenie plików pomiędzy aplikacjami	453
Projekt 227. Zmiana aktywnego komponentu za pomocą klawisza Enter	456
Projekt 228. XKill, czyli zamknij się, proszę!	457
Projekt 229. Modyfikowanie menu systemowego formy	459
Projekt 230. Modyfikowanie menu systemowego aplikacji w pasku zadań	460

Rozdział 12. Niezarządzane biblioteki DLL oraz mechanizm PInvoke 463

Procedury i funkcje w bibliotece DLL	464
Projekt 231. Tworzenie biblioteki DLL — eksport procedur i funkcji	464
Projekt 232. Statyczne łączenie bibliotek DLL — import funkcji	467
Projekt 233. Dynamiczne ładowanie bibliotek DLL	469
Projekt 234. Powiadamianie biblioteki o jej załadowaniu lub usunięciu z pamięci ..	471
Projekt 235. Import funkcji WinAPI	473
Formy w bibliotece DLL	474
Projekt 236. Jak umieścić formę w bibliotece DLL?	474
Projekt 237. Wykorzystanie biblioteki DLL z funkcją tworzącą formę	477
Aplet panelu sterowania	479
Projekt 238. Przygotowanie biblioteki DLL z funkcją zwrotną CPlApplet	479
Projekt 239. Przygotowanie instalatora apletu dla Windows XP i Windows 2003 ...	484
PInvoke	486
Projekt 240. Importowanie funkcji z niezarządzanych bibliotek DLL	487
Projekt 241. Przekazywanie łańcuchów	488
Projekt 242. Import funkcji WinAPI. Dźwięk w Windows Forms	488
Projekt 243. Bezpieczny wskaźnik IntPtr i szeregowanie	489

Rozdział 13. Automatyizacja i inne technologie bazujace na COM	491
COM	491
Projekt 244. Wykorzystanie obiektu COM do tworzenia plików skrótu .lnk	492
Osadzanie obiektów OLE2	493
Projekt 245. Statyczne osadzanie obiektu	493
Projekt 246. Aby zakonczyc edycje dokumentu. Laczzenie menu aplikacji klienckiej i serwera OLE	494
Projekt 247. Wykrywanie niezakonczonej edycji przy zamknieciu programu	495
Projekt 248. Ręczne inicjowanie edycji osadzonego obiektu	495
Projekt 249. Dynamiczne osadzanie obiektu	496
Automatyizacja	497
Projekt 250. Laczzenie z serwerem automatyzacji Excel z uzyciem komponentu TExcelApplication	498
Projekt 251. Laczzenie z serwerem automatyzacji Excel z uzyciem funkcji GetActiveOleObject	501
Projekt 252. Uruchamianie aplikacji Excel za posrednictwem mechanizmu automatyzacji (funkcja CreateOleObject)	502
Projekt 253. Eksplorowanie danych w arkuszu kalkulacyjnym	504
Projekt 254. Korzystanie z okien dialogowych serwera automatyzacji. Zapisywanie danych do pliku	505
Projekt 255. Zapisywanie danych z wykorzystaniem okna dialogowego aplikacji klienckiej	506
Projekt 256. Edycja danych w komorkach Excela	507
Projekt 257. Reagowanie na zdarzenia komponentu TExcelApplication	508
Projekt 258. Korzystanie z funkcji matematycznych i statystycznych Excela	509
Projekt 259. Uruchamianie aplikacji Microsoft Word i tworzenie nowego dokumentu lub otwieranie istniejacego	510
Projekt 260. Wywoływanie funkcji Worda na przykladzie sprawdzania pisowni i drukowania	511
Projekt 261. Wstawianie tekstu do biezacego dokumentu Worda	511
Projekt 262. Zapisywanie biezacego dokumentu Worda	512
Projekt 263. Zaznaczanie i kopiowanie calego tekstu dokumentu Worda do schowka	513
Projekt 264. Kopiowanie zawartosci dokumentu Worda do komponentu TRichEdit bez uzycia schowka (z pominieniem formatowania tekstu)	513
Projekt 265. Formatowanie zaznaczonego fragmentu tekstu w dokumencie Worda	513
Projekt 266. Serwer automatyzacji OLE przegladarki Internet Explorer	515
Projekt 267. Projektowanie serwera automatyzacji	515
Projekt 268. Testowanie serwera automatyzacji	519
ActiveX	521
Projekt 269. Korzystanie z kontrolek ActiveX w projektach dla platformy Win32 ..	521
Rozdział 14. Internet Direct (Indy)	525
Podstawy	526
Projekt 270. Polaczenie TCP klient-serwer	526
Projekt 271. Ping	528
Protokol FTP	530
Projekt 272. Pobieranie plikow	530
Projekt 273. Wyslanie plikow	534
Projekt 274. Podtrzymywanie polaczenia	534
Projekt 275. Zdalne tworzenie i usuwanie katalogow	535

Komunikator	536
Projekt 276. Przygotowanie interfejsu i nawiązywanie połączenia	536
Projekt 277. Wysłanie wiadomości	538
Projekt 278. Automatyczne nawiązanie połączenia	539
Projekt 279. Lista kontaktów	541
Wysyłanie listów e-mail	542
Projekt 280. Wysłanie niesformatowanej wiadomości e-mail	542
Projekt 281. Wysłanie wiadomości e-mail z załącznikami	543
Przesyłanie plików z użyciem protokołu HTTP	544
Projekt 282. Klient HTTP. Sprawdzenie, czy istnieje nowa wersja programu	544
Projekt 283. Pobieranie plików za pomocą protokołu HTTP	546
Projekt 284. Wskaźnik postępu	546
Projekt 285. Przeciwdziałanie blokowaniu interfejsu użytkownika	547
Rozdział 15. Wstęp do DirectX.NET	549
Projekt 286. Inicjacja grafiki DirectX.NET	550
Projekt 287. Wyświetlanie trójkątów (bufor wierzchołków)	556
Projekt 288. Teksturowany trójkąt	561
Projekt 289. Teksturowany czworokąt	566
Projekt 290. Dodajemy trzeci wymiar	567
Projekt 291. Źródła światła	573
Projekt 292. Tworzenie brył. Tetraedr	578
Projekt 293. Tworzenie brył. Kilka tetraedrów	582
Projekt 294. Ukrywanie niewidocznych powierzchni (bufor Z)	586
Projekt 295. Paski trójkątów	589
Projekt 296. Materiały oraz odbicia zwierciadlane	593
Projekt 297. Bardziej skomplikowane oświetlenie	596
Projekt 298. Siatki	599
Projekt 299. Tryb pełnoekranowy	604
Projekt 300. Wychodzenie z kłopotów	607
Projekt 301. Dźwięk. Sposób pierwszy	610
Projekt 302. Video	612
Projekt 303. Dźwięk. Sposób drugi	614
Dodatki	617
Skorowidz	619

Rozdział 7.

Kontrolka Windows Forms

Przykładem kontrolerek, których rzeczywiście brakuje w bibliotece .NET, są komponenty wizualne związane z plikami, a więc lista zawierająca zawartość wskazanego katalogu, drzewo katalogów czy lista dostępnych dysków. Poniżej zajmiemy się uzupełnieniem tego braku, tworząc komponent `TFileListBox` zawierający listę dostępnych plików i katalogów we wskazanym miejscu na dysku (z możliwością wędrowania po nich, łącznie ze zmianą dysku). To będzie dość uniwersalny komponent, zaprojektowany tak, żeby mógł stanowić dobry wzorzec dla pisania kolejnych.



Omówiony poniżej komponent został wcześniej opisany w wersji dla języka C# w książce napisanej przez Jacka Matulewskiego pt. *Język C#. Programowanie dla platformy .NET w środowisku Borland C# Builder*. Wydawnictwo HELP, 2004.

Komponenty najłatwiej tworzyć z innych komponentów. W przypadku komponentu `TFileListBox` jest oczywiste, że najwygodniej oprzeć się na komponencie `ListBox` dostępnym w bibliotece Windows Forms z platformy Microsoft .NET. Nie miałoby przecież sensu „rysowanie” nowego interfejsu komponentu od podstaw. Dzięki temu pozostaje nam jedynie problem pobrania listy plików i katalogów oraz odpowiedniego zareagowania na zdarzenia wywoływane przez użytkownika komponentu.

Część statyczna kontrolki

W praktyce najłatwiej projektować komponenty tworząc najpierw projekt aplikacji, a dopiero potem dodać do niego moduł komponentu. Dzięki temu uzyskujemy wygodną platformę testowania komponentu bez konieczności jego instalowania w środowisku. Tak postąpimy w poniższych przykładach — stworzymy projekt aplikacji, która będzie pełniła funkcję środowiska testowania, i dopiero do tego projektu dodamy komponent. Kompilacją komponentu do postaci DLL zajmiemy się po zakończeniu jego projektowania i testowania.

Projekt 122. Tworzenie projektu aplikacji testującej oraz interfejsu nowego komponentu

1. Tworzymy projekt *FileListBox_Demo*, w którym będziemy testować nowy komponent: z menu *File*, podmenu *New* wybieramy *Windows Forms Application — Delphi for .NET*.
2. Do projektu dodajemy moduł przyszłego komponentu:
 - a) z menu *File*, podmenu *New*, wybieramy pozycję *Other...*;
 - b) w oknie *New Items* na zakładce *Delphi for .NET Projects, New Files* zaznaczamy pozycję *User Control for Windows Forms* i klikamy *OK*.



Na tej samej zakładce dostępna jest również pozycja *Component for Windows Forms*, ale w jego rozwijaniu niedostępny jest widok projektowania, który mamy w przypadku kontrolki.

3. Od razu zapiszmy projekt wraz modulem kontrolki na dysku, klikając przycisk *Save All* (lub klawisze *Shift+Ctrl+S*) na pasku narzędzi. Zapiszmy kod pustego na razie komponentu do pliku *FileListBox.pas*. W menadżerze projektu nowy plik wyróżniony jest specyficzną ikoną wyróżniającą komponenty użytkownika.
4. W widoku projektowania pliku *FileListBox.pas* (zakładka na górze edytora powinna wskazywać ten plik) w inspektorze obiektów widoczne są własności klasy *TUserControl*. Przede wszystkim dzięki pozycji (Name) (w grupie *Design*) zmieniamy jej nazwę na *TFileListBox* (nie możemy w Delphi użyć nazwy *FileListBox*, ponieważ została już użyta do nazwania modułu).
5. W widoku projektowania zwiększamy rozmiar kontrolki.
6. Na palecie komponentów zaznaczamy *Listbox* i umieszczamy go na interfejsie komponentu.
7. Zaznaczamy go i za pomocą inspektora ustawiamy jego własność *Dock* na *Fill*.
8. Jego własność *HorizontalScrollbar* zmieniamy na *True*.
9. Kompilujemy projekt naciskając *Ctrl+F9*.
10. Przechodzimy na zakładkę modułu *WinForm* — naszego środowiska testowania komponentu.
11. W palecie komponentów znajdujemy grupę *My User Controls* (na końcu listy), a w niej nasz komponent *TFileListBox*. Umieścimy go na powierzchni formy.

Punkt 11. będzie możliwy do wykonania jedynie dzięki skompilowaniu pliku komponentu (punkt 9.). Ogólną zasadą w tworzeniu komponentów w ten sposób powinno być kompilowanie projektu w momencie zakończenia, nawet częściowych, modyfikacji wprowadzanych w kodzie i zmianie zakładki na plik, w którym jest on testowany. Podgląd komponentu oraz widoczne w inspektorze jego własności i zdarzenia są bowiem widoczne dzięki skompilowanym plikom binarnym, a nie analizie kodu.

Wykonując czynności z projektu 122. stworzyliśmy wygodne środowisko testowania komponentu. Każde naciśnięcie klawisza *F9* spowoduje rekompilację zarówno kodu komponentu, jak i aplikacji, na której oknie został umieszczony.

Należy zwrócić uwagę, że nasz komponent nie rozszerza klasy `ListBox`. Wykorzystujemy komponent tego typu jako prywatne pole nowego komponentu (relacja *ma* zamiast *jest*). Klasą bazową jest natomiast `System.Windows.Forms.UserControl`. Jest to standardowa metoda, która pozwala na ukrycie niektórych własności i metod komponentu `ListBox` (w bibliotece Windows Forms nie ma bowiem znanych z VCL i VCL.NET komponentów typu `TCustomListBox`).

Projekt 123. Zdefiniuj prywatne pola wykorzystywane przez `TFileListBox`

Zanim przystąpimy do tworzenia komponentu, warto przemyśleć, jak ma być on zbudowany i jaką powinien pełnić funkcję. Warto spisać sobie nawet jego strukturę na kartce, choć szczerze mówiąc sam robię to dopiero wtedy, gdy pogubię się w kodzie. Zadaniem naszego komponentu ma być zaprezentowanie listy plików i katalogów znajdujących się we wskazanym miejscu na dysku. Potrzebna więc będzie zmienna typu `String`, która przechowuje pełną ścieżkę do katalogu, oraz dwie tablice tego samego typu zawierające listy plików i katalogów. Ponadto w komponencie pokazemy katalog nadrzędny (dwie kropki), oczywiście poza sytuacją, gdy prezentowany katalog to główny katalog na dysku. Za listą plików i katalogów umieścimy także listę dysków, aby w każdej chwili użytkownik mógł przenieść się na inny dysk. Wszystko te elementy powinny podlegać konfiguracji, a więc potrzebne nam będą również zmienne logiczne, które zapewnią możliwość decyzji użytkownikowi, czy chce widzieć w komponencie pliki, katalogi i (lub) dyski. Warto uwzględnić również możliwość filtrowania widocznych w liście plików.

1. Przechodzimy do pliku `FileListBox.pas` (zakładka na górze edytora).
2. W edytorze kodu (zakładka `Code` na dole edytora), do sekcji `interface` wstawiamy deklaracje nowych pól klasy `TFileListBox`, definiując przy okazji dla nich oddzielny blok edytora o nazwie „Prywatne pola i metody” (listing 7.1):

Listing 7.1. Moduł komponentu

```
unit FileListBox;

interface

uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data;

type
  TFileListBox = class(System.Windows.Forms.UserControl)
  {$REGION 'Designer Managed Code'}
  strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
```

```

Components: System.ComponentModel.Container;
ListBox1: System.Windows.Forms.ListBox;
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure InitializeComponent;
{$ENDREGION}
{$REGION 'Prywatne pola i metody'}
strict private
    //pola wewnetrzne
    listaKatalogow :Array of String;
    listaPlikow :Array of String;
    listaDyskow :Array of String;
    pokazujDwieKropki :Boolean;
    //pola przechowujace ustawienia komponentu
    sciezkaKatalogu :String;
    uwzględnijKatalogi :Boolean;
    uwzględnijPliki :Boolean;
    uwzględnijDyski :Boolean;
    uwzględnijKatalogNadrzedny :Boolean;
    filtr :String;
{$ENDREGION}
strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
private
    { Private Declarations }
public
    constructor Create;
end;

[assembly: RuntimeRequiredAttribute(typeof(TFileListBox))]

//dalsza modułu

```

3. W konstruktorze umieszczamy polecenia inicjujące nowe pola klasy (listing 7.2):

Listing 7.2. Konstruktor klasy inicjuje własności komponentu wartościami domyślnymi

```

constructor TFileListBox.Create;
begin
    inherited Create;
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent;

    //inicjacja prywatnych pol klasy
    pokazujDwieKropki:=True;
    sciezkaKatalogu:=nil;
    uwzględnijKatalogi:=True;
    uwzględnijPliki:=True;

```

```
uwzględnijDyski:=True;  
uwzględnijKatalogNadrzedny:=True;  
filtr:=nil;  
end;
```

Pola w grupie oznaczonej komentarzem `//wewnetrzne` będą potrzebne do funkcjonowania „silnika” komponentu, natomiast własności z drugiej grupy określają jego funkcjonowanie i po zbudowaniu komponentu dostęp do nich powinien być możliwy w inspektorze obiektów. Zrealizujemy to definiując własności i dlatego wszystkie te pola klasy zadeklarowane zostały jako prywatne. A dokładniej — ściśle prywatne (ang. *strictly private*), a więc niedostępne nawet dla funkcji i procedur zdefiniowanych w module, ale nie będących metodami klasy.

Potrzebujemy teraz prywatnej metody, która będzie pobierała listę plików i podkatalogów znajdujących się w katalogu określonym przez pole `sciezkaKatalogu` i która umieści obie listy w komponencie `listBox1`. Metoda powinna oczywiście uwzględnić wartości zdefiniowanych w projekcie 123 pól komponentu. Będzie to serce komponentu.

Projekt 124. Stworzyć metodę pobierającą nazwy plików i podkatalogów znajdujących się we wskazanym katalogu

1. Do sekcji `uses` modułu *FileListBox* dodajemy deklarację użycia przestrzeni nazw *System.IO*, dorzucając ją do listy.

```
uses  
  System.Drawing, System.Collections, System.ComponentModel,  
  System.Windows.Forms, System.Data, System.IO;
```

2. Do zdefiniowanego przez nas bloku *Prywatne pola i metody*, za deklaracjami pól, umieszczamy deklarację metody (listing 7.3):

Listing 7.3. Deklaracja metody *PobierzZawartoscKatalogu* dodana do bloku *Prywatne pola i metody*

```
{$REGION 'Prywatne pola i metody'}  
strict private  
  //pola wewnetrzne  
  listaKatalogow :array of String;  
  listaPlikow :array of String;  
  listaDyskow :array of String;  
  pokazujDwieKropki :Boolean;  
  //pola przechowujace ustawienia komponentu  
  sciezkaKatalogu :String;  
  uwzględnijKatalogi :Boolean;  
  uwzględnijPliki :Boolean;  
  uwzględnijDyski :Boolean;  
  uwzględnijKatalogNadrzedny :Boolean;  
  filtr :String;  
  
  //metody  
  procedure PobierzZawartoscKatalogu;  
{$ENDREGION}
```


3. Do sekcji implementation dodajemy natomiast definicję tej metody (listing 7.4):

Listing 7.4. *Serce komponentu*

```
procedure TFileListBox.PobierzZawartoscKatalogu;
type TablicaObiektow = array of TObject;
begin
  if sciezkaKatalogu=nil then sciezkaKatalogu:=Directory.GetCurrentDirectory;

  pokazujDwieKropki:=(sciezkaKatalogu<>Path.GetPathRoot(sciezkaKatalogu))
    and uwzględnijKatalogNadrzedny;

  if (not Directory.Exists(sciezkaKatalogu)) then
    raise Exception.Create('Katalog '+sciezkaKatalogu+' nie istnieje!');

  ListBox1.Items.Clear;

  if uwzględnijKatalogi then
    begin
      if pokazujDwieKropki then ListBox1.Items.Add(['..']);
      listaKatalogow:=Directory.GetDirectories(sciezkaKatalogu);
      System.Array.Sort(listaKatalogow);
      listBox1.Items.AddRange(listaKatalogow as TablicaObiektow);
    end;
  if uwzględnijPliki then
    begin
      listaPlikow:=Directory.GetFiles(sciezkaKatalogu);
      System.Array.Sort(listaPlikow);
      listBox1.Items.AddRange(listaPlikow as TablicaObiektow);
    end;
  if uwzględnijDyski then
    begin
      listaDyskow:=Directory.GetLogicalDrives();
      listBox1.Items.AddRange(listaDyskow as TablicaObiektow);
    end;
end;
```

4. Do konstruktora klasy dodajemy wywołanie zdefiniowanej metody (listing 7.5):

Listing 7.5. *Wywołanie funkcji PobierzZawartoscKatalogu finalizuje inicjację komponentu*

```
constructor TFileListBox.Create;
begin
  inherited Create;
  //
  // Required for Windows Form Designer support
  //
  InitializeComponent;

  //inicjacja prywatnych pol klasy
  pokazujDwieKropki:=True;
  sciezkaKatalogu:=nil;
  uwzględnijKatalogi:=True;
  uwzględnijPliki:=True;
  uwzględnijDyski:=True;
  uwzględnijKatalogNadrzedny:=True;
```

```
filtr=nil;  
  
//zapełnianie listy  
PobierzZawartoscKatalogu;  
end;
```

5. Kompilujemy i uruchamiamy projekt naciskając klawisz F9.

Jak działa metoda `PobierzZawartoscKatalogu`? Na początku metody sprawdzamy, czy własność określająca ścieżkę do katalogu nie jest przypadkiem pusta. Jeżeli jest, to umieszczamy w niej ścieżkę do bieżącego katalogu roboczego odczytanego za pomocą polecenia `Directory.GetCurrentDirectory`. Kolejny warunek sprawdza, czy katalog wskazywany przez własność `sciezkaKatalogu` istnieje na dysku. Jeżeli nie — zgłaszany jest wyjątek z odpowiednim komunikatem.

Następnie sprawdzamy, czy na początku listy powinny znajdować się dwie kropki reprezentujące katalog nadrzędny (to okaże się bardzo wygodne, gdy będzie już możliwe wędrowanie po katalogach za pomocą tego komponentu). Abyśmy mogli dodać owe dwie kropki, muszą być spełnione dwa warunki. Po pierwsze katalog, którego zawartość zamierzamy zaprezentować, nie może być katalogiem głównym dysku oraz równocześnie własność `uwzględnijKatalogNadrzedny` musi być ustawiona na `True`.

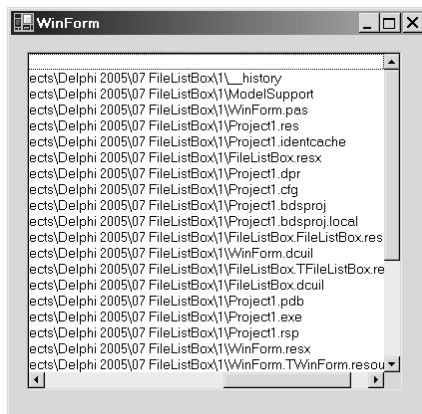
Za pomocą polecenia `ListBox1.Items.Clear` czyścimy zawartość `ListBox1` i przystępujemy do odczytania listy plików, katalogów i dysków przy użyciu odpowiednich metod statycznych klasy `Directory`. Wreszcie, po posortowaniu, umieszczamy je w `ListBox1` za pomocą jego metody `ListBox1.Items.AddRange`, uwzględniając wartość odpowiednich pól z grupy `uwzględnij...`. Aby skorzystać z metody `AddRange`, rzutuujemy dynamiczną tablicę łańcuchów na tablicę obiektów (operator `as`). Korzystamy w tym celu ze zdefiniowanego w nagłówku metody typu `TablicaObiektow`. Zasadnicze znaczenie ma w powyższym kodzie klasa `Directory`. Poza wykorzystaniem przez nas odczytywaniem zawartości wskazanego katalogu udostępnia ona także statyczne metody, pozwalające na manipulowanie katalogami (tworzenie, kasowanie).

Dodając wywołanie metody do konstruktora musimy zwrócić uwagę na to, aby znalazło się ono za wywołaniem metody `InitializeComponent` inicjującej umieszczone w trakcie projektowania komponenty. Nie musimy inicjować pola `sciezkaKatalogu` łańcuchem odpowiadającym konkretnemu katalogowi, bo gdy metoda `PobierzZawartoscKatalogu` wykryje, że jest ono niezainicjowane, sama zapisze do niej ścieżkę bieżącego katalogu.

Projekt 125. Z pełnej ścieżki dostępu do plików wyłaniamy samą nazwę plików i katalogów

Jak zwykle przy pierwszej kompilacji kodu nie wszystko działa jak należy (o ile w ogóle się kompiluje i uruchamia). Aplikacja Czytelnika powinna wyglądać jak na rysunku 7.1, co oznacza, że również u Czytelnika komponent prezentuje pliki z pełną ścieżką dostępu. Musimy zatem zmodyfikować tę metodę w taki sposób, aby widoczna była tylko sama nazwa pliku lub katalogu.

Rysunek 7.1.
Jak widać pierwsza wersja komponentu nie działa w pełni zgodnie z naszymi oczekiwaniami



Inny sposób wyodrębnienia nazwy pliku za pomocą klasy `FileInfo` opisany został w projekcie 102.

1. Przystępujemy do edycji modułu komponentu (zakładka `FileListBox`).
2. Na każdą pozycję z list `listaPlikow` i `listaKatalogow` działamy metodą `Path.GetFileName` przed dodaniem jej do `ListBox1`. Nazwy katalogów uzupełniamy nawiasami kwadratowymi, a symbole dysków — nawiasami utworzonymi ze znaków `< i >` (listing 7.6):

Listing 7.6. *Poprawione serce komponentu*

```

procedure TFileListBox.PobierzZawartoscKatalogu;
type TablicaObiektow = array of TObject;
var
  i :Integer;
  sciezka :String;
begin
  if sciezkaKatalogu=nil then sciezkaKatalogu:=Directory.GetCurrentDirectory;

  pokazujDwieKropki:=(sciezkaKatalogu<>Path.GetPathRoot(sciezkaKatalogu))
    and uwzględnijKatalogNadrzedny;

  if (not Directory.Exists(sciezkaKatalogu)) then
    raise Exception.Create('Katalog '+sciezkaKatalogu+' nie istnieje!');

  ListBox1.Items.Clear;

  if uwzględnijKatalogi then
    begin
      if pokazujDwieKropki then ListBox1.Items.Add(['..']);
      listaKatalogow:=Directory.GetDirectories(sciezkaKatalogu);
      System.Array.Sort(listaKatalogow);
      for sciezka in listaKatalogow do
        listBox1.Items.Add([''+Path.GetFileName(sciezka)+'']);
    end;

  if uwzględnijPliki then

```

```
begin
  if (filtr<>nil) then
    listaPlikow:=Directory.GetFiles(sciezkaKatalogu,filtr)
  else
    listaPlikow:=Directory.GetFiles(sciezkaKatalogu);
  System.Array.Sort(listaPlikow);
  for sciezka in listaPlikow do
    listBox1.Items.Add(Path.GetFileName(sciezka));
  end;

  if uwzględnijDyski then
    begin
      listaDyskow:=Directory.GetLogicalDrives();
      for sciezka in listaDyskow do
        listBox1.Items.Add('<' + sciezka.Substring(0,2) + '>');
      end;
    end;
end;
```

Metodę `Items.AddRange`, dodającą całą tablicę do `ListBox1`, zastąpiliśmy wywołującą w pętli metodą `Items.Add`, dodającą tylko jedną, już zmodyfikowaną pozycję. Poza tym podkreśliliśmy różnice między plikami a podkatalogami znajdującymi się w prezentowanym katalogu przez otoczenie nazw tych ostatnich nawiasami kwadratowymi. Natomiast w przypadku dysków usunęliśmy znak ukośnika (*slash*) i pozostawiliśmy jedynie symbol dysku typu „C:”, który otoczyliśmy znakami `< i >`.

Sama pętla użyta w powyższym kodzie jest również interesująca. Zastosowaliśmy konstrukcję

```
for sciezka in listaPlikow do
  listBox1.Items.Add(Path.GetFileName(sciezka));
```

Konieczne jest oczywiście zadeklarowanie zmiennej `sciezka` typu `String`, która będzie reprezentowała bieżący element tablicy lub kolekcji wskazanej za słowem kluczowym `in`. Jest to konstrukcja analogiczna do znanej z `C#` `foreach` o identycznym działaniu jak zwykła pętla `for` następującej postaci:

```
for i:=0 to Length(listaPlikow)-1 do
  listBox1.Items.Add(Path.GetFileName(listaPlikow[i]));
```

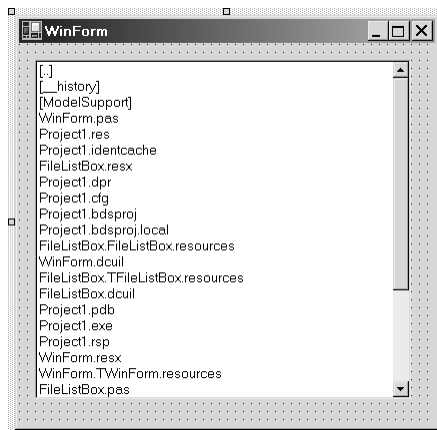
Przejdźmy na zakładkę pliku `WinForm.pas`. W widoku projektowania formy, na której umieszczony został komponent, powinna być widoczna zawartość katalogu projektu. Możemy skontrolować, czy zmiany w kodzie odniosły zamierzony skutek (wcześniej należy koniecznie kod skompilować). Po tych zmianach komponent powinien prezentować się znacznie lepiej (rysunek 7.2).

Projekt 126. Uwzględnić filtrowanie plików z maską określoną przez pole filtr

Wzorem analogicznego komponentu VCL dodamy możliwość filtrowania prezentowanej listy plików zgodnie z ustaloną przez użytkownika maską. Wystarczy do kodu metody `PobierzZawartoscKatalogu` dodać jedną linię, która w przypadku gdy referencja

Rysunek 7.2.

Podgląd formy
w widoku
projektowania



filtr nie jest równa nil (nie jest pusta), wykorzystuje ją jako drugi argument przeciążonej metody `Directory.GetFiles` (listing 7.7)

Listing 7.7. Uwzględniamy możliwość określenia maski plików w metodzie `PobierzZawartoscKatalogu`

```

if uwzględnijPliki then
begin
  if (filtr<>nil) then
    listaPlikow:=Directory.GetFiles(sciezkaKatalogu,filtr)
  else
    listaPlikow:=Directory.GetFiles(sciezkaKatalogu);
  System.Array.Sort(listaPlikow);
  for i:=0 to Length(listaPlikow)-1 do
    listBox1.Items.Add(Path.GetFileName(listaPlikow[i]));
end;

```

Aby przetestować działanie filtra:

1. zmieniamy na chwilę polecenie inicjacji pola `filtr` w konstruktorze na:
`filtr:='*.pas';`,
2. kompilujemy kod projektu i komponentu naciskając klawisze `Ctrl+F9`,
3. przechodzimy do widoku projektowania formy `WinForm`.

Wykorzystaliśmy wersję przeciążonej metody `Directory.GetFiles`, w której drugi argument jest maską plików zwracanych w tablicy łańcuchów przez tę funkcję. Efekt widoczny jest na rysunku 7.3. Podobnie przeciążona jest metoda `Directory.GetDirectories`. Po sprawdzeniu działania filtra przywracamy pierwotną postać inicjującego go polecenia, a mianowicie `filtr:=nil;`

Wykonajmy jeszcze jeden test. Zmieńmy podaną w listingu 7.2 ścieżkę tak, żeby wskazywała na katalog główny jakiegoś dysku w celu upewnienia się, że w takiej sytuacji nie są pokazywane dwie kropki symbolizujące katalog nadrzędny. Warto również przetestować pola `uwzględnijKatalogi`, `uwzględnijPliki` i `uwzględnijDyski`.

Rysunek 7.3.
*Efekt działania
filtra *.pas*

