

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

C++ Builder 5.

Vademecum profesjonalisty. Tom II

Autorzy: Jarrod Hollingworth, Dan Butterfield, Bob Swart, Jamie Allsop, et al.

Tłumaczenie: zbiorowe

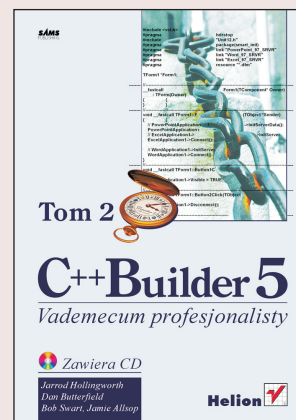
ISBN: 83-7197-586-4

Tytuł oryginału: [C++ Builder 5 Developer](#)

Format: B5, stron: 772

oprawa twarda

Zawiera CD-ROM



Ta długo oczekiwana książka, omawiająca piątą już wersję C++Buildera, nie jest – jak w przypadku wielu innych pozycji poświęconych narzędziom programistycznym – typową prezentacją możliwości środowiska projektowego i języka C++. Poszczególne jej rozdziały były bowiem tworzone nie przez twórców, ale przez użytkowników C++Buildera – dzięki czemu całość nabiera niecodzienny charakter kompendium wiedzy praktycznej, bazującej na niejednokrotnie wieloletnich doświadczeniach programistycznych. Zawartość tomu drugiego stanowi naturalne uzupełnienie treści tomu pierwszego, lecz dzięki specyficznemu charakterowi książki tom ten jest samodzielną, pełnowartościową pozycją, od której można rozpocząć lekturę całości.

Książka ta obejmuje szeroki zakres zagadnień związanych z zaawansowanymi możliwościami piątej wersji C++Buildera. W kolejnych rozdziałach znajdują się omówienia i przykłady prezentujące wykorzystanie komponentów Interbase Express, ADOExpress, InternetExpress i wielu innych. Dzięki tej książce wzbogacisz swoją wiedzę o tworzeniu aplikacji z wykorzystaniem architektur CORBA, COM, COM+. Zaawansowani programiści odświeżą przed Tobą tajniki programowania rozproszonych aplikacji internetowych, biurowych i bazodanowych.

- Wyeliminuj błędy powodujące niekontrolowane wycieki pamięci oraz przechwytyj szczegółowe informacje o błędach i wyjątkach występujących w czasie działania programu
- Rozszerz swoje umiejętności projektowania i implementowania wielowarstwowych aplikacji bazodanowych opartych na technologii MIDAS
- Wykorzystaj w swoich programach bogate możliwości aplikacji pakietu Microsoft Office
- Poznaj techniki programowania rozproszonego z wykorzystaniem architektur COM, COM+ i CORBA
- Wzbogać tworzone strony WWW o obiekty i formularze oparte na technologii ActiveX
- Wykorzystaj w pełni możliwości prezentacji danych i drukowania dokumentów, oferowane przez środowisko graficzne systemu Windows
- Profesjonalnie projektuj i dystrybuuj własne zestawy komponentów.



Spis treści

O Autorach	19
Wprowadzenie	29
Przedmowa do wydania polskiego	33
Rozdział 1. Projekty i środowisko zintegrowane programu C++Builder	37
Wprowadzenie do projektów C++Buildera	37
Pliki używane w projektach C++Buildera	38
Menedżer projektów	41
Korzystanie z repozytorium obiektów	42
Wstawianie elementów do repozytorium obiektów	42
Wykorzystanie elementów z repozytorium obiektów	46
Współużytkowanie elementów w obrębie projektu	46
Dostosowywanie repozytorium obiektów	47
Tworzenie i dodawanie kreatora do repozytorium obiektów	47
Pakiety — podstawy i zastosowanie	50
Uwagi na temat korzystania z pakietów	53
Pakiety wykonywalne programu C++Builder	54
Program narzędziowy tdump	56
Nowe cechy środowiska zintegrowanego C++Builder 5	56
Kategorie właściwości w inspektorze obiektów	56
Obrazki w listach rozwijanych inspektora obiektów	57
Format XML pliku projektu	61
Formularze — zapis w formacie tekstowym	62
Opcje na poziomie węzłów	64
Nowa lista zadań	67
Kreator aplikacji konsolowych	70
Podsumowanie	73
Rozdział 2. Zaawansowane programowanie w C++Builderze	75
Wprowadzenie do SCL (Standard C++ Library) i wzorców	75
Wzorce języka C++	76
Właściwości biblioteki Standard C++ Library	79
Potyczki z kontenerami i iteratorami	80
Zastosowanie standardowych algorytmów	87
Biblioteka SCL — wnioski	88

Wskaźniki typu smart i kontenery typu strong	89
Serta kontra stos	89
Wskaźniki	89
Kontenery typu strong (Strong Containers)	93
Pułapki	95
Wskaźniki smart i kontenery strong — podsumowanie	96
Implementacja zaawansowanej obsługi wyjątków (Advanced Exception Handler)	96
Przegląd strategii	97
Analiza korzyści	97
Wymiana domyślnej procedury obsługi wyjątków	98
Dodawanie do klasy informacji specyficznych dla projektu	99
Kod źródłowy procedury obsługi wyjątków	99
Zaawansowana obsługa wyjątków — podsumowanie	115
Tworzenie aplikacji wielowątkowych	115
Zrozumieć wielozadaniowość	115
Zrozumieć wielowątkowość	116
Tworzenie wątku za pomocą wywołań API	116
Obiekt TThread	119
Główny wątek VCL	123
Ustalanie priorytetów	125
Pomiar czasu wykonania wątków	128
Synchronizacja wątków	129
Wprowadzenie do wzorców projektowych	133
Powtarzalna natura wzorców	133
Wzorce w projektowaniu oprogramowania	134
Wzorce projektowe jako źródło terminologii	134
Format wzorca projektowego	135
Klasyfikacja wzorców projektowych	136
Wzorce projektowe — wnioski	137
Podsumowanie	138
Rozdział 3. Tworzenie własnych komponentów — ciąg dalszy	139
Zagadnienia dotyczące tworzenia własnych komponentów	139
Wyświetlanie złożonych właściwości publikowanych w oknie Object Inspector	139
Przestrzeń nazw w liście parametrów zdarzenia	140
Określanie listy parametrów zdarzenia	141
Przykrywanie funkcji dynamicznych	146
Własne komponenty — obsługa komunikatów	147
Funkcje zwrotne Windows w komponentach	161
Wybór typu bazowego właściwości	169
Udostępnianie komponentu w czasie projektowania i w czasie działania aplikacji	174
Ramki	176
Ramka — co to jest?	177
Klasa TCustomFrame	178
Praca z ramkami w czasie projektowania	178

Praca z ramkami w czasie działania aplikacji	179
Tworzenie klasy wyprowadzonej z TFrame	180
Klasy pochodne klasy wyprowadzonej z TFrame	183
Wielokrotne wykorzystanie ramek	184
Ramki — wnioski	185
Dystrybucja komponentów	186
Umieszczanie komponentów w pakiecie	186
Lokalizacja plików pakietu	191
Nazwy stosowane w pakietach.....	192
Nazwy komponentów.....	193
Dystrybucja pakietu projektowego	194
Komponenty dla różnych wersji C++Buildera	196
Tworzenie ikonki dla palety komponentów	199
Tworzenie komponentów przeznaczonych do dystrybucji — wskazówki.....	199
Inne zagadnienia związane z dystrybucją komponentów	200
Podsumowanie	201
Rozdział 4. Programowanie baz danych	203
Modele architektur aplikacji baz danych.....	203
Mechanizm Borland Database Engine.....	204
Aplikacja macierzysta BDE (jednowarstwowa)	204
Architektura klient-serwer (BDE i SQL Links).....	205
Aplikacje rozproszone (wielowarstwowe).....	206
Metody dostępu do danych	207
Komponenty macierzyste.....	207
Duet ODBC i BDE	208
ODBC i komponenty macierzyste	209
ADO (ActiveX Data Objects).....	210
Wbudowany SQL.....	210
Macierzysty interfejs API	211
Architektury aplikacji baz danych — wnioski.....	212
Inne źródła informacji o architekturach aplikacji baz danych	212
Język SQL.....	213
Tabele i indeksy	213
Parametry	214
Zapytania insert, update, delete i select	215
Funkcje agregujące.....	217
Dodatkowe informacje o języku SQL.....	217
Komponenty ADO Express	217
ADO a BDE	218
Przegląd komponentów ADO	220
Połączenie z bazą danych.....	221
Dostęp do zbiorów danych.....	222
TADOTable.....	223
Dostęp do zbiorów danych.....	226

Zarządzanie transakcjami	227
Zdarzenia komponentów ADO	227
Tworzenie uniwersalnej aplikacji bazy danych	228
Optymalizacja wydajności	230
Obsługa błędów	231
Aplikacje wielowarstwowe a ADO	232
Komponenty ADO Express — podsumowanie	232
Architektury gromadzenia danych	232
Decyzje wstępne	233
Pobieranie danych z wielu źródeł	234
Narzędzie Data Module Designer	236
Moduł danych	236
Zalety stosowania modułu danych	237
Moduły danych w aplikacjach, bibliotekach DLL i obiektach rozproszonych	238
Zawartość modułu danych	239
Dodawanie właściwości do modułu danych	240
Obsługa modułu danych	240
Dla zaawansowanych	243
Dziedziczenie klasy formularza w modułach danych	243
Obsługa nierównego dziedziczenia klas formularzy i modułów danych	244
Moduł danych niezależny od interfejsu użytkownika	245
Moduł danych a komponenty szkieletowe i komponenty specyficzne dla aplikacji	245
Moduły danych w pakietach	248
Moduły danych — podsumowanie	248
InterBase Express	248
Schemat bazy danych Bug Trackera	249
Reguły bazy danych	251
Generatory, wyzwalacze i procedury składowane	251
Implementacja programu Bug Tracker	253
Program Bug Tracker w akcji	261
Podsumowanie	262
Rozdział 5. Krok naprzód — COM+	263
COM+ — wprowadzenie	264
Aplikacje COM+	264
Katalog COM+	265
Usługi COM+	265
Zdarzenia słabo skojarzone	265
Transakcje	267
Synchronizacja	270
Zabezpieczenia	270
Komponenty w kolejce	271
Równoważenie obciążenia	271

Programowanie i zastosowania zdarzeń COM+	271
Tworzenie obiektu zdarzenia COM+	271
Instalacja zdarzenia w aplikacji COM+	273
Tworzenie obiektu wydawcy	277
Tworzenie obiektów subskrybentów.....	278
Konfiguracja subskrybentów	282
Tworzenie subskrypcji trwałej	283
Tworzenie subskrypcji tymczasowej	286
Programowanie i zastosowania obiektów transakcyjnych COM+	295
Tworzenie obiektów transakcyjnych w warstwie biznesowej aplikacji	296
Programowanie menedżera wyrównywania zasobów (CRM).....	305
Aplikacja klienta	316
Podsumowanie	317
Rozdział 6. Rozproszone aplikacje wielowarstwowe — MIDAS 3	319
Wprowadzenie do technologii MIDAS	319
Aplikacje klientów i serwerów MIDAS	321
Tworzenie serwera MIDAS	322
Rejestracja serwera MIDAS.....	326
Tworzenie klienta MIDAS	327
Model aktówki.....	330
Metoda ApplyUpdates().....	333
Implementacja obsługi błędów	334
Demonstracja obsługi błędów aktualizacji rekordów	337
Zdalny dostęp do serwera.....	338
Tworzenie serwera MIDAS w modelu nadrzędny-szczegółowy.....	339
Eksport relacji nadrzędny-szczegółowy.....	342
Tworzenie klienta MIDAS w modelu nadrzędny-szczegółowy	343
Tabele zagnieżdżone	343
MIDAS — „wąskie gardła” systemu	346
Rozszerzenia MIDAS 3	348
Komponent TDataSetProvider	349
Porównanie interfejsów IProvider i IAppServer.....	349
Bezstanowy obiekt pośrednika danych	349
Zastosowania komponentów InternetExpress.....	354
Komponent TWebConnection	357
Pula obiektów	358
Połączenia oparte na gniazdach (TCP/IP).....	360
Pośrednik obiektów	363
Instalacja.....	364
Podsumowanie	365
Rozdział 7. Aplikacje rozproszone w CORBA	367
Wprowadzenie do CORBA.....	367
Jak działa CORBA?	368
Wywołania statyczne i dynamiczne	369
Na każde żądanie.....	369

Płasko lub hierarchicznie	369
Klient, serwer — kto jest kim?	370
Object Request Broker	370
Podstawowy adapter obiektu.....	370
Przenośny adapter obiektu	370
CORBA kontra COM.....	370
Komponenty VisiBrokera	371
Smart agent.....	371
Demon aktywacji obiektu OAD.....	371
Konsola.....	372
Język definicji interfejsu IDL	372
Słowo kluczowe interface	373
Słowo kluczowe attribute.....	373
Metody	373
Definicje typów	374
Wyjątki.....	374
Dziedziczenie	375
Moduły	375
Co nowego w C++Builderze 5?.....	375
Obsługa CORBA w C++Builderze.....	376
Opcje środowiskowe	377
Opcje programu uruchomieniowego.....	377
Opcje projektu.....	378
Kreator serwera CORBA	379
Kreator klienta CORBA	380
Kreator pliku IDL.....	380
Kreator implementacji obiektu CORBA.....	381
Okno aktualizacji projektu	381
Kreator użycia obiektu CORBA	382
Różnice pomiędzy wersją czwartą a piątą C++Buildera	383
Modele implementacyjne.....	384
Dziedziczenie proste	385
Dziedziczenie z implementacją wirtualną.....	385
Model z delegacją (związek).....	386
CORBA dla „zamożnych inaczej”.....	386
Podsumowanie	386
Rozdział 8. Integracja aplikacji z pakietem Microsoft Office.....	387
Integracja aplikacji z Microsoft Office — wprowadzenie	387
Jak integrować?.....	388
Obiekt ToleContainer.....	388
Automatyzacja OLE	390
Obiekty automatyzacji i obiekty typu Variant	392
Automatyzacja w ochronie przed wirusami makr.....	394
Język Word Basic.....	394

Integracja aplikacji z programem Word	394
Kolekcje	395
Obiekt aplikacji (Application Object).....	395
Praca z dokumentami	396
Pobieranie tekstu z dokumentu Worda	399
Umieszczanie obiektów w dokumentach Worda	402
Integracja z programem Excel	405
Obiekt aplikacji (Application Object).....	405
Praca ze skoroszytami	405
Modyfikacja komórek arkuszy Excela.....	408
Odczytywanie komórek arkuszy Excela	410
Komponenty palety Servers C++Buildera 5.....	410
Komponenty WordApplication i WordDocument.....	411
Skorowidz — wersja druga	411
Biblioteka ATL i komponenty OleServer — wnioski	416
Co dalej?	416
Word.....	416
Excel.....	417
Pozostałe aplikacje pakietu Office	417
Podsumowanie	419
Rozdział 9. Zastosowania technologii ActiveX	421
Podstawy aktywnych obiektów serwera	421
Tworzenie obiektów Request i Response za pomocą kreatora	
Active Server Object Wizard	422
Kolejne wbudowane obiekty ASP: Session, Server i Application	429
Obiekty ASP i obsługa WebBrokera.....	430
Aktualizacje aktywnych obiektów serwera.....	431
Diagnostyka aktywnych obiektów serwera.....	432
Aktywne formularze — wprowadzenie	433
Tworzenie aktywnego formularza	433
Instalacja aktywnego formularza w celu wyświetlenia w oknie przeglądarki	
Internet Explorer	435
Opcje instalacji aktywnego formularza.....	436
Nawiązywanie połączenia z aktywnym formularzem	438
Aktywne formularze reprezentujące dane	440
Pliki CAB i pakiety	443
Aktualizacje aktywnego formularza	443
Katalogi OCCACHE i Downloaded Program Files.....	444
Aktywny formularz jako klient architektury MIDAS	445
Aktywne formularze w Delphi	447
Tworzenie szablonów komponentów za pomocą aktywnych formularzy.....	448

Programowanie powłoki systemowej (shell).....	450
Powłoka — podstawy	450
Odczytywanie zawartości folderu	453
Przekazywanie obiektów powłoki.....	455
Podsumowanie	460
Rozdział 10. Prezentacja danych w C++Builderze.....	461
Prezentacja raportów.....	461
Na czym polega wartość raportu?.....	461
Tworzenie raportów z użyciem komponentów QuickReport	462
Jak działa przeglądarka raportów?.....	463
Podsumowanie	472
Drukowanie tekstu i grafiki	472
Drukowanie tekstu.....	472
Drukowanie grafiki	480
Zaawansowane techniki drukowania	485
Określenie rozdzielczości drukarki	485
Określenie rozmiarów pola wydruku.....	485
Określenie fizycznych rozmiarów strony	486
Określenie możliwości graficznych drukarki	486
Obracanie czcionki wydruku.....	487
Manipulowanie ustawieniami drukarki.....	488
Odczytanie nazwy drukarki domyślnej.....	489
Wybranie drukarki domyślnej.....	490
Inicjalizacja obiektu Printer().....	493
Dostęp do struktury DEVMODE za pomocą klasy TPrinter.....	493
Wykorzystanie struktury PRINTER_INFO_2	494
Pozostałe funkcje związane z podawaniem papieru	499
Obsługa zadań wydruku	506
Przechwycenie klawisza PrintScreen.....	508
Drukowanie formularza	510
Utworzenie podglądu wydruku	510
Konwersje współrzędnych związane z drukowaniem	510
Kilka dodatkowych wskazówek.....	512
Komponent TChart i tworzenie wykresów	513
Pierwsze kroki	514
Modyfikowanie postaci wykresu w trakcie działania programu	515
Wymiana danych z wykresem	516
Dynamiczne tworzenie wykresów	518
Drukowanie wykresów.....	518
Komponent TeeChart Pro i jego możliwości.....	520
Podsumowanie	520

Rozdział 11. Dystrybucja oprogramowania.....	523
Wersje międzynarodowe aplikacji i lokalizacja	523
Omówienie problemu tworzenia wersji międzynarodowych	523
Aplikacja Localize.....	524
Warto zapamiętać	529
Resource DLL Wizard	530
Zasada działania	530
Tworzenie biblioteki zasobów DLL.....	531
Testowanie	533
Inne dołączane pliki i programy	534
Pliki aplikacji.....	534
Etapy dystrybucji.....	537
Ochrona praw autorskich i licencjonowanie oprogramowania	537
Ochrona praw autorskich	538
Umowa licencyjna.....	539
Zabezpieczanie oprogramowania	540
Zabezpieczanie aplikacji	540
Zabezpieczanie aplikacji z użyciem komponentów niezależnych firm	541
Zabezpieczanie aplikacji za pomocą komponentów innego typu.....	542
Zabezpieczanie oprogramowania — uwagi końcowe	543
Shareware.....	543
Zabezpieczanie oprogramowania shareware	544
Realizacja zabezpieczeń typu shareware	546
Metody zabezpieczeń shareware w skrócie	546
Dystrybucja i marketing za pośrednictwem Internetu	547
Witryny WWW	547
Pomoc techniczna.....	547
Reklama.....	548
Darmowe banery reklamowe	549
Akceptowanie kart kredytowych i dostarczanie kodów odblokowujących.....	550
Wskazówki i porady dotyczące marketingu w Internecie	551
Podsumowanie	552
Rozdział 12. Sztuczki, uwagi i porady	553
Symulowanie klawisza tabulacji klawiszem Enter.....	553
Rozwiązanie problemu.....	554
Objaśnienie kodu.....	554
Kilka pułapek	557
Symulowanie klawisza tabulacji — podsumowanie.....	558
Ustalanie wersji systemu operacyjnego	558
Rozwiązanie problemu.....	558
Objaśnienie kodu.....	558
Ustalanie wersji systemu operacyjnego — podsumowanie.....	560

Programowanie z wykorzystaniem liczb zmiennopozycyjnych.....	561
Wiadomości ogólne.....	561
Praca na liczbach.....	562
Dodawanie i odejmowanie.....	563
Rozwinięte ciągi przekształceń.....	566
Porównywanie danych.....	567
Liczby zmiennopozycyjne — uwaga końcowa.....	567
Tworzenie ekranu tytułowego.....	567
Funkcja WinMain().....	568
Tworzenie ekranu tytułowego.....	569
Zapobieganie uruchamianiu więcej niż jednego egzemplarza aplikacji.....	571
Rozwiązanie problemu.....	571
Objaśnienie kodu.....	572
Podsumowanie.....	576
Praca w trybie „przeciągnij i upuść”.....	576
Rozwiązanie problemu.....	576
Objaśnienie kodu.....	576
Jak to działa?.....	579
„Przeciągnij i upuść” — podsumowanie.....	580
Wykonanie zrzutu ekranowego.....	581
W jaki sposób Windows radzi sobie z oknami?.....	581
Rozwiązanie problemu.....	581
Wykonywanie zrzutów ekranowych — podsumowanie.....	585
Wykorzystanie komponentu TJoyStick.....	586
Tworzenie aplikacji podobnej do monitora systemu Windows.....	596
Zasoby systemu Windows.....	596
Rozwiązanie problemu.....	598
Monitor systemu — podsumowanie.....	604
Aplikacja Soundex.....	604
Implementacja.....	605
Komponenty klasy TTreeView.....	611
Podstawy.....	612
Dodawanie węzłów.....	612
Piktogramy węzłów.....	615
Nawigacja w widoku hierarchicznym.....	616
Dostęp do węzłów.....	617
Wyszukiwanie węzłów.....	618
Wyświetlanie liczby węzłów podrzędnych.....	620
Przemieszczanie węzłów w hierarchii.....	621
Przeciąganie i opuszczanie węzłów.....	622
Modyfikacje węzłów.....	624
Usuwanie węzła.....	626
Wycofywanie i zatwierdzanie operacji usuwania.....	627
Zapisanie zawartości drzewa w pliku.....	630
Komponenty zestawu TTree — podsumowanie.....	631

Narzędzie do pozyskiwania ikon	631
Aplikacja naśladowująca działanie Eksploratora Windows	639
Funkcje i interfejsy powłoki Windows	639
Implementacja	640
WinExplorer — podsumowanie	645
Usługi Windows NT	646
Program SendMsg	647
Usługa Stickums	649
Program Stickem	652
Usługi NT — podsumowanie	653
Kryptografia	653
Implementacja	654
Szyfrowanie wiadomości	657
Deszyfrowanie wiadomości	661
Zegar słoneczny	664
Specjalny dodatek dla niedowiarków	669
Podsumowanie	670
Rozdział 13. Zastosowanie praktyczne — aplikacja World Wave Statistics	671
Program World Wave Statistics	671
Analiza kodu źródłowego	672
Plik nagłówkowy math.h	673
Plik nagłówkowy mapunit.h	673
Plik nagłówkowy wavedata.h	673
Plik źródłowy about.cpp	674
Analiza kodu modułu TMainUnit	675
Ulepszenia	679
Podsumowanie	681
Dodatek A Spis treści tomu I	683
Dodatek B Skorowidz tomu I	695
Skorowidz	743

Rozdział 8.

Integracja aplikacji z pakietem Microsoft Office

Integracja C++Buildera z pakietem Microsoft Office daje możliwość konstruowania różnorodnych przydatnych aplikacji — od prostych programów użytkowych pomocnych w wypełnianiu codziennych obowiązków biurowych po rozbudowane zestawy oprogramowania wykorzystujące pakiet Microsoft Office jako kolekcję przydatnych komponentów edycyjnych i kalkulacyjnych. W rozdziale zostaną przedstawione przesłanki integrowania tworzonych aplikacji z pakietem Office i sposoby realizacji tego zadania.

Integracja aplikacji z Microsoft Office — wprowadzenie

Jedną z korzyści stosowania pakietu biurowego jest możliwość zaoszczędzenia czasu i wysiłku programisty. Wielu z nas uszczęśliwiłby nawet znikomy ułamek czasu zaoszczędzonego dzięki wykorzystaniu operacji wycinania i wklejania.

Microsoft Office umożliwia dalsze uproszczenie wielu czynności dzięki wykorzystaniu wbudowanego języka programowania — Visual Basic for Applications — tworzenie skrótów, automatyzację codziennych czynności. Dzisiejsze *wytnij i wklej* może być jutro automatycznym wymianianiem całych akapitów.

Poza możliwością dostosowania pakietu Microsoft Office do własnych potrzeb, bardzo interesująco zapowiada się fakt, że Office jest także serwerem automatyzacji OLE. Uprzejmi panowie z Microsoftu pozostawili nam do dyspozycji wszystkie „haki” i „furtki”, pozwalając nam kontrolować pakiet Office i wykorzystywać go we własnych programach.

Oznacza to, że możemy w prosty sposób zapewnić naszemu programowi wydajność i funkcjonalność procesora tekstu. Używamy po prostu programów już zainstalowanych przez użytkownika. Możemy uruchamiać jawnie pakiet Office lub przejąć nad nim kontrolę i wykorzystywać go w tle. Jeśli kiedykolwiek marzyłeś o otrzymywaniu ze swoich własnych programów raportów w standardowym formacie, teraz możesz to osiągnąć. Chociaż Word nie przewyższa komponentu TQuickRep w dziedzinie łatwości programowania, możesz np. przesłać dokument w formacie Worda pocztą elektroniczną, mając pewność, że odbiorca za pomocą Worda go odczyta.

Integracja własnych aplikacji z pakietem Microsoft Office daje następujące korzyści:

1. Dostęp do uznanych programów i interfejsów rozszerzających możliwości naszych własnych aplikacji.

Jeśli chcesz udostępnić użytkownikom arkusz kalkulacyjny lub edytor tekstów jako część swojego programu, nie musisz już sam tworzyć lub zdobywać odpowiednich komponentów. Użytkownicy Twojego programu docenią wygodę tego rozwiązania, pracując z dobrze znanymi sobie aplikacjami.

2. Dostęp do popularnych formatów dokumentów.

Nie zajmuj się już formatami plików dokumentów. Zamiast szukać niezbędnych informacji dotyczących sposobu czytania i zapisywania powszechnie stosowanych plików, zrób użytek z już istniejącego kodu, który to potrafi. Zastosuj go w swojej aplikacji.

3. Automatyzacja zadań.

Jeśli kiedykolwiek musiałeś wykonać tę samą operację na wielu dokumentach, wiesz, jak wiele wysiłku trzeba włożyć w wykonanie zadania, otwarcie następnego pliku, wykonanie zadania, otwarcie następnego pliku... Dzięki automatyzacji możemy przygotować żadaną operację, wybrać pliki do przetwarzania i uruchomić całość.

Jak integrować?

Kiedy już wiemy, że możemy integrować własne aplikacje z Microsoft Office, musimy odpowiedzieć sobie na pytanie: jak to zrobić? Na szczęście Microsoft Office udostępni kilka interfejsów, zarówno w postaci serwerów automatyzacji OLE, jak i obiektów COM. C++Builder oferuje dwie możliwości przejęcia tych usług — wykorzystanie komponentu `TOLEContainer` lub obiektów automatyzacji.

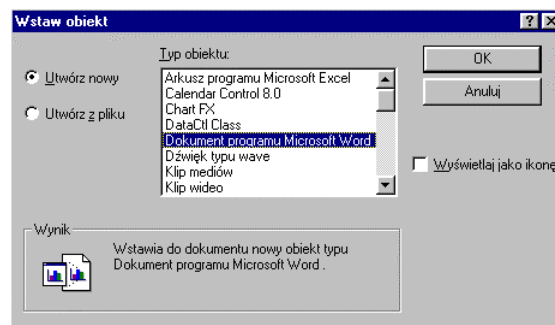
Wybór jednej z tych możliwości uzależniony jest od planowanego zakresu wykorzystania pakietu Office. Komponent `TOLEContainer` jest znakomitym wyborem, jeśli zamierzasz osadzić jedną z aplikacji Office w swoim programie w taki sposób, aby pojawiła się jako jego część. Z drugiej strony, jeżeli potrzebujesz większej kontroli nad działaniem takiej hybrydowej aplikacji, użyj `OleVariant`, aby pozyskać obiekty automatyzacji zaimplementowane w ramach pakietu Office.

Obiekt `TOLEContainer`

Wykorzystanie obiektu `TOLEContainer` to najłatwiejszy sposób integracji pakietu Office z utworzoną aplikacją. Utwórz nową aplikację i umieść w niej (na środku formularza głównego) egzemplarz komponentu `TOLEContainer` (zakładka *System*). Rozszerz go tak, aby pozostał około 50-punktowy margines z wszystkich czterech stron elementu.

Kliknij teraz dwukrotnie obiekt `OleContainer1`. Zostanie otwarte okno dialogowe *Wstaw obiekt*, które powinno wyglądać tak, jak na rysunku 8.1.

Rysunek 8.1.
Okno dialogowe
Wstaw obiekt
C++Buildera



Wybierz pozycję *Dokument programu Microsoft Word* i kliknij przycisk *OK*.

Przygotowania są zakończone — skompiluj i uruchom program. Powinieneś zobaczyć formularz z pustym panelem na środku. Kliknij dwukrotnie panel — po chwili okno ulegnie zmianie. Menu i pozostałe elementy Worda pojawią się w górnej części formularza, a panel zostanie zastąpiony dokumentem programu Word. Kiedy zakończysz eksperymenty, zamknij program i wróć do C++Buildera.

Prawym przyciskiem myszy kliknij obiekt *OleContainer1* i wybierz z menu polecenie *Delete Object* (usuń obiekt). Spowoduje to odłączenie Twojej aplikacji od programu Word. Poniżej obiektu *OleContainer1* umieść trzy przyciski nazwane *Button_Word*, *Button_Excel* oraz *Button_PPoint* i etykietowane, odpowiednio: „Word”, „Excel” i „PowerPoint”. Dodaj kod poniższego wydruku do metod *OnClick* przycisków. Przykładowy projekt zawierający poniższy kod znajdziesz na płycie CD-ROM dołączonej do książki, w katalogu *TOleContainer*.

Wydruk 8.1.
Program *Word-Excel-PowerPoint*

```
// Kod dla przycisku "Word"
void __fastcall TForm_OleContainer::Button_WordClick(TObject *Sender)
{
    OleContainer1->CreateObject("Word.Document", false);
    OleContainer1->DoVerb(ovShow);
}

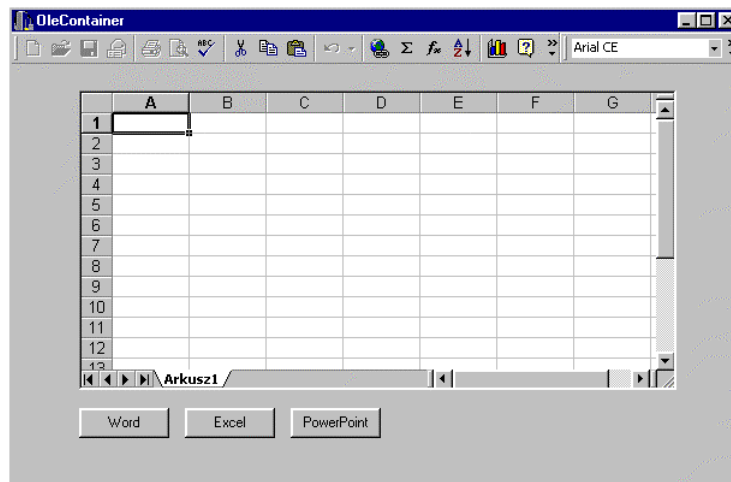
// Kod dla przycisku "Excel"
void __fastcall TForm_OleContainer::Button_ExcelClick(TObject *Sender)
{
    OleContainer1->CreateObject("Excel.sheet", false);
    OleContainer1->DoVerb(ovShow);
}

// Kod dla przycisku "PowerPoint"
void __fastcall TForm_OleContainer::Button_PPointClick(TObject *Sender)
{
    OleContainer1->CreateObject("Powerpoint.show", false);
    OleContainer1->DoVerb(ovShow);
}
```

Kiedy już wprowadzisz powyższy kod, skompiluj i uruchom program. Tym razem podwójne kliknięcie panelu wygeneruje wyjątek *Operation not allowed on an empty OLE container* (nieodzwolona operacja na pustym panelu). Kliknij przycisk *OK*, aby wznowić pracę.

Zamiast klikać w obszarze panelu, spróbuj teraz nacisnąć jeden z trzech przycisków umieszczonych pod nim. Po kliknięciu przycisku *Excel* lub *Word* panel zmieni się w odpowiedni dokument, a formularz otrzyma pasek narzędziowy odpowiedniej aplikacji (patrz rysunek 8.2).

Rysunek 8.2.
Arkusz Excela
w programie
OleContainer



Kliknięcie przycisku *PowerPoint* zmienia panel w pusty slajd, ale na formularzu nie pojawiają się paski narzędzi ani menu. Dlaczego? Otóż PowerPoint został zaimplementowany inaczej niż Word i Excel.

Automatyzacja OLE

Zastosowanie obiektów automatyzacji jest metodą bardziej wszechstronną niż integracja z wykorzystaniem egzemplarza klasy *TOleContainer*. Metoda ta umożliwia dużo lepszą kontrolę nad każdym obiektem i jego właściwościami. Jediną jej wadą jest to, że wraz ze wzrostem możliwości sterowania obiektami zwiększa się złożoność ich użytkowania i programowania.

Każdy obiekt automatyzacji posiada pewną liczbę właściwości i metod. Kiedy tworzysz obiekt automatyzacji, uzyskujesz dostęp do jego właściwości i metod, podobnie jak uzyskujesz dostęp do właściwości i metod obiektu z chwilą otrzymania wskaźnika do niego.

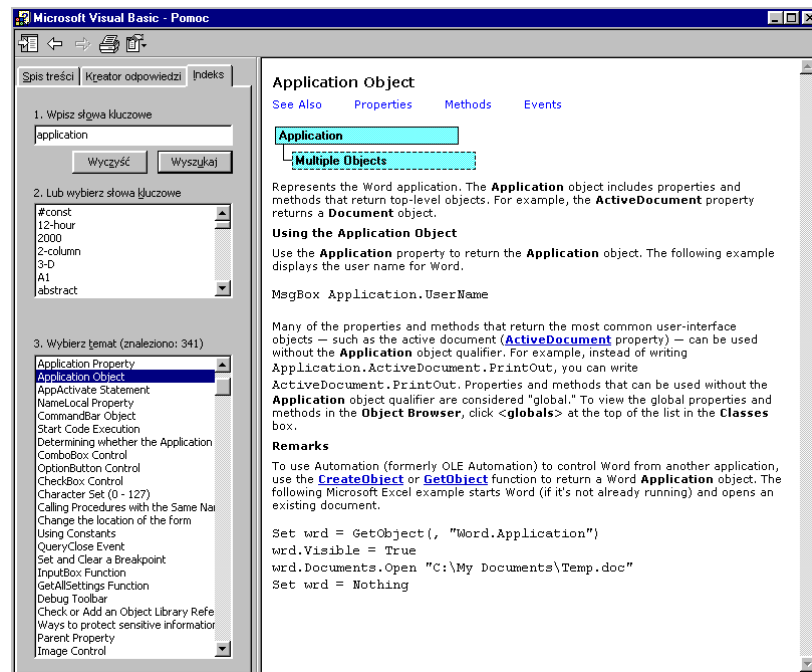
C++Builder oferuje dwa podstawowe sposoby stosowania automatyzacji: wykorzystanie bibliotek typów (*Type Library*) oraz obiektów automatyzacji OLE. Biblioteki typów to pliki bibliotek zawierające opis interfejsów udostępnianych przez aplikację i wszelkie typy, struktury i klasy umożliwiające ich eksploatację. Zaletą wykorzystania bibliotek typów jest możliwość kontroli typów, wadą — konieczność ich utworzenia za pomocą polecenia menu *Import Type Library*, lub też zdobycia w inny sposób. Ponadto ich zastosowanie oznacza wydłużenie czasu kompilacji projektu.

Wykorzystanie obiektów automatyzacji OLE wymaga znajomości nazw przeróżnych interfejsów oraz udostępnianych przez nie właściwości i metod. Stosowanie obiektów automatyzacji OLE uniemożliwia kontrolę typów, poza tym wymusza na programiście konieczność posiadania jakiegoś rodzaju dokumentacji na temat automatyzowanej aplikacji. Jednakże ich stosowanie bardzo przyspiesza kompilację — obiekty te nie używają dodatkowego kodu ani plików bibliotecznych wymaganych w przypadku bibliotek typów.

Dla uproszczenia, w rozdziale będziemy używać głównie obiektów automatyzacji OLE. Zwrócimy też uwagę na udogodnienia oferowane przez C++Buildera w przypadku wykorzystania bibliotek typów. Będziesz więc potrzebował dobrej dokumentacji. Na szczęście pakiet Office zawiera własną, wbudowaną dokumentację.

Uruchom program Word i wywołaj edytor Visual Basica (naciśnij *Alt+F11* lub wybierz z menu pozycję *Tools/Macro/Visual Basic Editor* (odpowiednio: *Narzędzia/Makro/Edytor VisualBasic* w wersji polskiej pakietu Office). Zostanie wyświetlony edytor Visual Basica, zawierający pliki pomocy interfejsu programu Word dla Visual Basica — o ile je zainstalowałeś. Jeśli nie — zrób to teraz. Wpisz w polu *index* (indeks) słowo *Application* i zlokalizuj opis obiektu *Application* programu Word. Widok ekranu powinien być zbliżony do tego z rysunku 8.3.

Rysunek 8.3.
Okno pomocy edytora
Visual Basic for
Applications



Możesz używać tej metody w celu uzyskania niezbędnych informacji o właściwościach, metodach i wyjątkach interesujących Cię obiektów. Technika ta działa ze wszystkimi aplikacjami pakietu Microsoft Office, więc jeśli szukasz informacji o obiektach pakietu Office, znajdziesz je właśnie tu.

Obiekty automatyzacji i obiekty typu Variant

Wykorzystanie obiektów klasy Variant C++Buildera bardzo ułatwia stosowanie obiektów automatyzacji. Obiekty Variant nie tylko mogą zawierać obiekty automatyzacji, ale również zawierają pewną liczbę procedur ułatwiających ich wykorzystanie.

Najbardziej użytecznymi w pozyskiwaniu i manipulowaniu obiektami automatyzacji metodami są:

- CreateObject()
- GetActiveObject()
- OleFunction()
- OleProcedure()
- OlePropertyGet()
- OlePropertySet()

CreateObject() wraz z GetActiveObject() służą, odpowiednio: do stworzenia i pobrania aktywnego obiektu automatyzacji. Działanie pozostałych funkcji jest zgodne z ich nazwami. Warto podkreślić, że funkcje OleFunction(), OleProcedure(), OlePropertySet() oraz OlePropertyGet() są w zasadzie otoczką (ang. *wrapper*) dla metody Exec().

Metoda Exec() używana jest przez C++Builder do uruchomienia procedury lub funkcji lub do pozyskania lub ustawienia wartości właściwości. W celu uzyskania dostępu do żądanej funkcji obiektu automatyzacji odwołuje się ona do czterech klas:

- Function()
- Procedure()
- PropertyGet()
- PropertySet()

Klasa Function() używana jest do określenia funkcji, PropertyGet() służy do pozyskania wartości właściwości itd. Obiekty tych klas powinny być powołane przed ich użyciem w Exec() i muszą zostać odświeżone za pomocą ich metod ClearArgs() przed ponownym użyciem. Przyjrzyjmy się temu mechanizmowi, zanim przejdziemy do bardziej szczegółowej analizy kilku aplikacji.

Stwórz nową aplikację, umieść na formularzu przycisk, nazwij go Button_LaunchWord i opatrz etykietą „Uruchom Worda”. Uzupełnij kodem z wydruku 8.2 metodę OnClick przycisku. Wydruk ten jest dostępny na dołączonej do książki płycie CD-ROM — projekt przykładu *automation_objects.bpr* znajduje się w folderze *AutomationObj*.

Wydruk 8.2.

Uruchomienie programu Word

```
void __fastcall TForm_Automation::Button_LaunchWordClick(TObject *Sender)
{
    Variant word_app;
    Variant word_docs;
```

```

Variant word_this_document;
Variant word_range;

word_app = Variant::CreateObject("word.application");
word_docs = word_app.OlePropertyGet("documents");

word_docs.OleProcedure("Add");

word_app.OlePropertySet("Visible", (Variant) true);
}

```

Ten sam fragment kodu, ale z wykorzystaniem metody `Exec()`, zawarty jest na wydruku 8.3. Stwórz nowy przycisk o nazwie `Button_LaunchWordExec`, opatrz go etykietą „Uruchom Worda (Exec)” i uzupełnij jego metodę `OnClick` kodem zamieszczonym na wydruku 8.3.

Wydruk 8.3.

Wywołanie programu Word z użyciem metody `Exec()`

```

void __fastcall TForm_Automation::Button_LaunchWordExecClick(TObject *Sender)
{
    Variant word_app;
    Variant word_docs;
    Variant word_this_document;
    Variant word_range;

    Function Documents("Documents"); //-- Definicja funkcji Documents
    Function AddDocument("Add"); //-- Definicja funkcji Add dokumentu
    PropertySet Visibility("Visible"); //-- Definicja właściwości Visibility

    word_app = Variant::CreateObject("word.application");
    word_docs = word_app.Exec(Documents);

    word_docs.Exec(AddDocument);

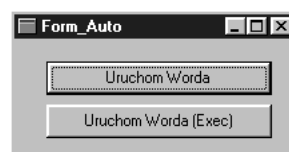
    Visibility << True;
    word_app.Exec(Visibility);
}

```

Rysunek 8.4 przedstawia program *LaunchWord* w czasie działania.

Rysunek 8.4.

Program
„LaunchWord”
w czasie działania



Pozostało nam już tylko sprawdzić, czy plik nagłówkowy *utilcls.h* jest włączony do programu. Używanie obiektów zmiennego typu w automatyzacji powoduje powołanie egzemplarza klasy `TAutoArgs`, zdefiniowanej właśnie w pliku *utilcls.h*. Dopisując więc w jednym z plików nagłówkowych linię

```
#include <utilcls.h>
```

unikniesz ewentualnych problemów. Skompiluj i uruchom program. Kiedy pojawi się formularz, kliknij przycisk „Uruchom Worda (Exec)”. Spowoduje to uruchomienie nowej kopii programu Word (niezależnie od tego, czy jest już uruchomiony jakiś egzemplarz programu Word) z pustym dokumentem, czy też nie.

Automatyzacja w ochronie przed wirusami makr

Automatyzacja ma wielką zaletę — udostępnia programiście uniwersalne mechanizmy, a użytkownikowi daje możliwość nieinteraktywnego i masowego wykonywania pospolitych zadań. Ujemną cechą automatyzacji jest potencjalne zagrożenie wirusami — otwierając dokument programu Office zawierający wirusa makrodefinicji, aplikacja zakłada, że wiesz o jego istnieniu i nie broni się przed otwarciem dokumentu.

Jeżeli zakładasz wykorzystanie automatyzacji w dokumentach powszechnie dostępnych, infekcja dokumentu wirusem makrodefinicji stanowi realne zagrożenie. Zadbaj o aktualizację programu antywirusowego. Aplikacje pakietu Office dopuszczają otwarcie zainfekowanego dokumentu, ale dobry program antywirusowy powinien uniemożliwić użycie takiego pliku.

Język Word Basic

Wydany przez Microsoft program Word 97 udostępniał pewne udogodnienie umożliwiające tworzenie małych programów rozszerzających funkcjonalność aplikacji z wykorzystaniem nowego języka — Word Basica. Mimo że później, w edycji '98 programu Word, zastąpiono go językiem Visual Basic for Applications, Microsoft pozostawił w kolejnych wersjach programu Word działający interfejs języka Word Basic.

Interfejs ten, dostępny za pośrednictwem obiektu automatyzacji `word.basic`, nie oferuje takiego zakresu funkcji jak VBA, jednak w pewnych sytuacjach doskonale się sprawdza. Powinieneś rozważyć jego wykorzystanie, jeśli nie masz pewności, której wersji pakietu Office będą używali odbiorcy tworzonych przez Ciebie oprogramowania.

Jeśli przypuszczasz, że może to być Office 97, użyj (ze względu na kompatybilność) języka Word Basic zamiast VBA. Dzięki temu zyskasz pewność, że Twojego oprogramowania będą mogli używać zarówno użytkownicy wcześniejszych, jak i nowszych wersji pakietu Office.

Integracja aplikacji z programem Word

Microsoft Office to duży i złożony pakiet oprogramowania, dający programistom dostęp do niektórych swoich wewnętrznych interfejsów. Czytając ten podrozdział, dowiesz się, w jaki sposób można zatrudnić te interfejsy do wykonywania prostych zadań, takich jak wczytywanie, zapisywanie i tworzenie nowych dokumentów programu Word.

Kolekcje

Zgodnie z nazwą, kolekcja (ang. *collection*) używana jest w programie Word do przechowywania grup obiektów. Każda kolekcja posiada dwa kluczowe elementy: właściwość `Count` oraz metodę `Item`. Właściwość `Count` informuje, ile obiektów przechowywanych jest w kolekcji, a metoda `Item` pozwala pobrać z kolekcji referencję dożądanego elementu, przez określenie jego numeru lub nazwy (jeśli obiekt takową posiada). Niektóre kolekcje posiadają też inne metody i właściwości zwiększające ich możliwości. Znakomitym przykładem jest kolekcja `Documents`, która udostępnia metodę `Add` tworzącą nowy dokument.

Obiekt aplikacji (Application Object)

Obiekt aplikacji jest rdzeniem programu Microsoft Word i podstawowym obiektem programów chcących korzystać z funkcji Microsoft Word. Po stworzeniu lub pozyskaniu obiektu aplikacji możemy uzyskać referencje do poszczególnych kolekcji lub właściwości wymaganych podczas wykonywania planowanych zadań.

Są dwa sposoby uzyskania obiektu aplikacji programu Word: stworzenie nowego obiektu lub pozyskanie referencji do już istniejącego egzemplarza. Obiekty typu `Variant` udostępniają dwie metody zwracające obiekt aplikacji: `CreateObject()` i `GetActiveObject()`. `CreateObject()` tworzy nowy egzemplarz obiektu automatyzacji, natomiast metoda `GetActiveObject()` pozwala uzyskać referencję do jednego z już istniejących obiektów automatyzacji. Obie metody w przypadku wystąpienia błędu generują wyjątek `EOLESysError`.

Aby otrzymać obiekt automatyzacji programu Word, zastosuj identyfikator `word.application`. Przy pierwszym odwołaniu do obiektu aplikacji często wskazane jest wykorzystanie (jeśli to możliwe) istniejącego egzemplarza obiektu. Pozwala to uniknąć uruchamiania kilku kopii tej samej aplikacji i może przyspieszyć działanie Twojego program, ponieważ nie będzie on musiał czekać na załadowanie części pakietu Office.

Stwórz nową aplikację i dodaj do niej przycisk `Button_LaunchWord`, z etykietą „Uruchom Worda”. Uzupełnij metodę `OnClick` przycisku kodem zamieszczonym na wydruku 8.4.

Wydruk 8.4.

Program Launch Word

```
void __fastcall TForm1::Button_LaunchWordClick(TObject *Sender)
{
    try
    {
        my_word = Variant::GetActiveObject("word.application");
    }
    catch (...)
    {
        // Próba pozyskania istniejącego egzemplarza obiektu automatyzacji programu Word
        // nie powiodła się. Należy stworzyć nowy obiekt.
        try
        {
            my_word = Variant::CreateObject("word.application");
        }
        catch (...)
    }
}
```

```
    {  
        // Nie można utworzyć obiektu automatyzacji programu Word  
        Application->MessageBox("Obiekt automatyzacji nieosiągalny", "Błąd:", MB_OK |  
            MB_ICONERROR);  
    }  
}  
my_word.OlePropertySet("Visible", (Variant)True);  
}
```

Kod źródłowy tego programu zawarty jest w projekcie *example3.bpr*, umieszczonym w folderze *LaunchWord* dołączonej do książki płyty CD_ROM.

Dodaj też do klasy definiującej formularz zmienną:

```
Variant my_word;
```

Skompiluj i uruchom program, a kiedy pojawi się jego formularz główny, kliknij przycisk „Uruchom Worda”. Jeśli w tle uruchomiony jest program Word, nic się nie wydarzy (nie działamy jeszcze w żaden sposób na obiekcie aplikacji). Jeśli jednak Word nie jest uruchomiony, w momencie wygenerowania wyjątku C++Builder zatrzyma program i poinformuje nas o tym. Zdarzenie to nie wystąpi w przypadku uruchamiania programu spoza środowiska uruchomieniowego C++Buildera.

Praca z dokumentami

Po uruchomieniu kopii programu Word warto zrobić z jego pomocą coś pożytecznego. Zobaczmy więc, w jaki sposób przebiega praca z dokumentami programu Word.

Po uzyskaniu interfejsu do aplikacji Worda mamy dostęp do jego kolekcji dokumentów. Jest ona obiektem rządzącym się własnymi prawami i zawierającym właściwości i metody pozwalające tworzyć, otwierać i zapisywać dokumenty, przeglądać zawartość kolekcji itp.

Obiekt aplikacji udostępnia kolekcję dokumentów w następujący sposób:

```
my_documents = my_word.OlePropertyGet("Documents");
```

Za chwilę dodamy powyższą linię do naszej aplikacji.

Tworzenie nowego dokumentu

W celu dodania nowego dokumentu do kolekcji stosujemy metodę `Add()`. Metoda ta przyjmuje kilka opcjonalnych parametrów pozwalających lepiej kontrolować jej działanie.

Do utworzenia zwykłego pustego dokumentu wystarczy sama metoda `Add()`. Tworzenie nowego dokumentu na podstawie szablonu wymaga przekazania nazwy tego szablonu. Dodaj nowy przycisk do aplikacji i nazwij go `Button_CreateDocument`. Opatrz go etykietą „Nowy Dokument” i uzupełnij jego metodę `OnClick` kodem zamieszczonym na wydruku 8.5.

Wydruk 8.5.*Tworzenie nowego dokumentu*

```
void __fastcall TForm1::Button_CreateDocumentClick(TObject *Sender)
{
    Variant this_doc;
    long doc_count;

    my_docs = my_word.OlePropertyGet("Documents");
    this_doc = my_docs.OleFunction("Add");

    doc_count = my_docs.OlePropertyGet("Count");
    ShowMessage((AnsiString)"Otwarto " + doc_count + " dokument (ów)");
}
```

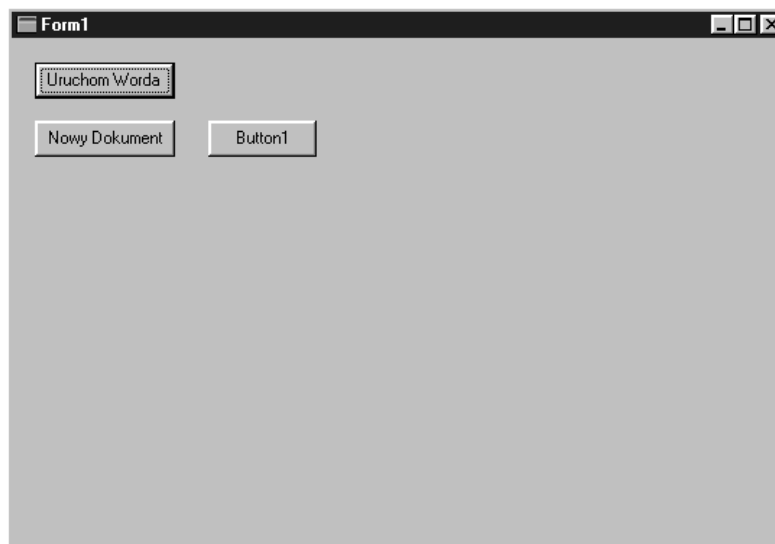
Ponieważ wprowadziliśmy nową zmienną, dodaj do definicji formularza następującą linię:

```
Variant my_docs;
```

Skompiluj i uruchom aplikację. Na ekranie powinno pojawić się okno przedstawione na rysunku 8.5. Kliknij przycisk *Nowy Dokument* — zostanie wygenerowany wyjątek z komunikatem Variant does not reference an automation object (zmienna nie wskazuje obiektu automatyzacji). Przed przystąpieniem do jakichkolwiek działań na obiekcie aplikacji Word musimy sprawdzić, czy uzyskaliśmy do niego referencję. Pozwól aplikacji kontynuować, a następnie kliknij przycisk *Uruchom Worda*. Dopiero wtedy kliknij po raz drugi przycisk *Nowy dokument*. Nowy dokument został dodany do aplikacji Word, a Ty zostałeś powiadomiony, ile dokumentów aktualnie jest otwartych — to często bardzo przydatna informacja.

Rysunek 8.5.

Aktualny wygląd programu „Launch Word”

**Zapisywanie dokumentów**

Zapisywanie dokumentów jest kolejną często wykonywaną operacją. Aby zapisać dokument, można wykorzystać jedną z metod zapisu lub metodę `SaveAs` dokumentu.

Dokument można zapisać na trzy sposoby: zapisać pojedynczy dokument, zapisać wszystkie otwarte dokumenty lub zapisać wersję dokumentu. To, który sposób zostanie zastosowany, zależy od obiektu będącego właścicielem metody. Obiekt `Document` lub `Version` zapisuje kopię samego siebie, natomiast obiekt `Documents` zapisuje całą kolekcję otwartych dokumentów.

Zademonstrujemy teraz działanie funkcji zapisu. Dodajmy do naszego formularza nowy przycisk `Button_QuickSave` etykietowany napisem „Zapisz” i wypełnijmy poniższym kodem (wydruk 8.6) jego metodę `OnClick`.

Wydruk 8.6.

Zapisywanie dokumentu

```
void __fastcall TForm1::Button_QuickSaveClick(TObject *Sender)
{
    Variant this_doc;
    this_doc = my_word.OlePropertyGet("ActiveDocument");
    this_doc.OleProcedure("Save");
}
```

Skompiluj i uruchom aplikację, po czym wykonaj cykl uruchamiania programu Word i tworzenia nowego dokumentu. Kliknij przycisk *Zapisz*. Word powinien wyświetlić okno dialogowe *Save As*¹, ponieważ nie została dotychczas określona nazwa dokumentu. Zamknij aplikację i powróć do C++Buildera.

Dodaj kolejny przycisk, `Button_QuickSaveAs`, z etykietą „Zapisz jako” i umieść poniższy kod wewnątrz metody `OnClick` przycisku.

Wydruk 8.7. Zapisywanie dokumentu pod określoną nazwą

```
void __fastcall TForm1::Button_QuickSaveAsClick(TObject *Sender)
{
    Variant this_doc;

    this_doc = my_word.OlePropertyGet("ActiveDocument");
    this_doc.OleProcedure("SaveAs", "c:\\zapisany_jako.doc");
}
```

Skompiluj i uruchom aplikację. Tym razem po uruchomieniu programu Word i utworzeniu nowego dokumentu kliknięcie przycisku *Zapisz jako* spowoduje zapisanie dokumentu pod nazwą zdefiniowaną w wywołaniu metody `SaveAs`.

Metoda `SaveAs` przyjmuje więcej parametrów, więc jeśli chcesz jej używać w pełnej postaci, rozważ bezpośrednie zastosowanie metody `Exec()` (`OleProcedure()` jest otoczką dla metody `Exec()`).

Otwieranie istniejącego dokumentu

Ponieważ umiemy już tworzyć i zapisywać dokumenty programu Word, może przydać się nam możliwość późniejszego ich otwierania. Służy do tego metoda `Open` kolekcji `documents`.

¹ *Zapisz jako...* w polskiej wersji MS Office — *przyp. thum*.

Jeśli znamy nazwę dokumentu, możemy ją przekazać do metody `Open`. Dodaj do formularza przycisk `Button_QuickOpen` i opatrz go etykietą „Otwórz dokument”. Spróbujemy z jego pomocą otworzyć wcześniej zapisany dokument.

Wypełnij metodę `OnClick` nowego przycisku kodem prezentowanym na wydruku 8.8. Następnie skompiluj i uruchom aplikację. Po uruchomieniu programu Word kliknij przycisk *Otwórz dokument*. Word załaduje uprzednio zapisany dokument.

Wydruk 8.8.*Otwieranie dokumentu*

```
void __fastcall TForm1::Button_QuickOpenClick(TObject *Sender)
{
    Variant this_doc;
    my_docs = my_word.OlePropertyGet("Documents");

    this_doc = my_docs.OleFunction("Open", "c:\\zapisany_jako.doc");
}
```

Pobieranie tekstu z dokumentu Worda

Umiemy już wykonywać podstawowe operacje na dokumentach programu Word — tworzyć je, zapisywać i odczytywać. Teraz zajmiemy się czymś bardziej użytecznym — nauczymy się wydobywać informacje z dokumentu programu Word. Stworzymy aplikację, która otworzy dokument, wydobędzie z niego listę słów i umieści ją (posortowaną) w nowym dokumencie.

Do utworzenia listy słów potrzebne są dwa elementy: obiekt `Range` (zakres) i kolekcja `Words`. Obiekt `Range` określa, która część dokumentu jest aktualnie używana (w naszym przypadku — cały dokument). Ustawienie go pozwoli nam uzyskać kolekcję słów zawartych w kolekcji `Words`.

Obiekt `Range` ustawia pozycję znaku początkowego i końcowego pokrywanego przez siebie tekstu. Jeżeli nie określisz jawnie początku i końca zakresu, obejmie on cały dokument. Aby oznaczyć wybraną część dokumentu, zdefiniuj samodzielnie punkty początkowy i końcowy (np. 20 znaków od początku dokumentu) lub wykorzystaj fakt definiowania obiektu `Range` w innych obiektach składowych dokumentu. Wydruk 8.9 prezentuje kilka fragmentów kodu — są to przykłady różnych sposobów wybierania zakresu.

Wydruk 8.9.*Zaznaczanie zakresu, zaznaczanie akapitu*

```
// Pobierz obiekt Range obejmujący cały dokument
my_range = my_docs.OleFunction("Range");

// Weź pod uwagę pierwsze 33 znaki
my_range = my_docs.OleFunction("Range", (Variant)0, (Variant)33);

// Zaznacz drugi akapit
my_paras = my_docs.OlePropertyGet("Paragraphs");
my_para = my_paras.OleFunction("Item", (Variant)2);
p_range = my_para.OlePropertyGet("Range");
my_range = my_docs.OleFunction("Range", p_range.OlePropertyGet("Start"),
    ↳p_range.OlePropertyGet("End"));
```

Po określeniu zakresu we wnętrzu dokumentu należy uzyskać referencję do kolekcji Words. Dokonamy tego, odczytując wartość właściwości Words z zakresu objętego aktualnie aktywnym obiektem Range.

Bezpośrednio po zdobyciu referencji do kolekcji Words możemy uzyskać listę słów zawartych w dokumencie. W tym celu utworzymy program pobierający aktualnie otwarty dokument programu Word. Stwórz nową aplikację zawierającą komponent `ListBox1` (pozostaw niezmienną nazwę komponentu) i przycisk o nazwie `Button_LoadWords`, opatrzony etykietą „Wczytaj słowa”. Wypełnij metodę `OnClick` przycisku kodem prezentowanym na wydruku 8.10.

Przykład ten znajduje się w projekcie *example4.bpr*, umieszczonym w folderze *LoadWords* na dołączonej do książki płycie CD-ROM.

Wydruk 8.10.

Program „Wczytaj słowa”

```
void __fastcall TForm1::Button_LoadWordsClick(TObject *Sender)
{
    Variant my_word_app;
    Variant my_doc;
    Variant my_words;
    long word_count;
    long index;

    try
    {
        my_word_app = Variant::GetActiveObject("word.application");
    }
    catch (...)
    {
        //-- Word nie jest uruchomiony – zakończ, wyświetlając monit
        ShowMessage("Uruchom program Word i spróbuj ponownie po załadowaniu dokumentu");
        return;
    }

    //-- Uzyskanie aktualnego dokumentu programu Word
    my_doc = my_word_app.OlePropertyGet("ActiveDocument");

    //-- Pobranie wszystkich słów dokumentu (zakres obejmuje domyślnie cały dokument)
    my_words = my_doc.OlePropertyGet("Words");

    //-- Ile słów w dokumencie?
    word_count = my_words.OlePropertyGet("Count");

    //-- Przenieś wszystkie do listy ListBox1
    for (index = 0; index < word_count; index++)
    {
        ListBox1->Items->Append(my_words.OleFunction("Item", (Variant)index));
    }
}
```

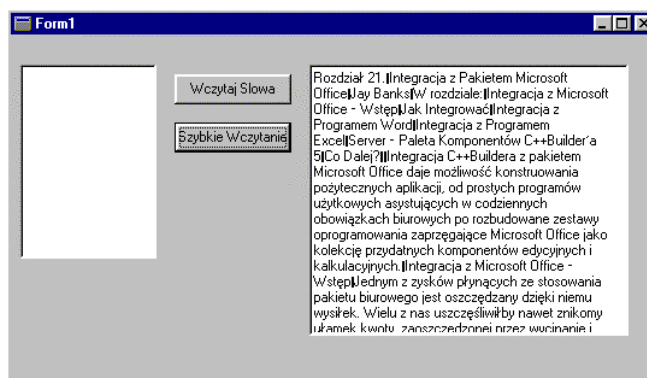
Skompiluj i uruchom aplikację, następnie uruchom program Word. Dopisz kilka przypadkowych słów do nowego dokumentu i kliknij przycisk *Wczytaj słowa* w swojej aplikacji. Lista powinna wypełnić się wprowadzonymi przez Ciebie słowami.

Zauważ, że oprócz słów pobrałeś również znaki interpunkcyjne (jeżeli je wprowadziłeś) oraz pewne znaki sterujące. Word zazwyczaj traktuje je tak jak zwykłe słowa, co może ułatwić lub utrudnić wykonanie zadania — w zależności od jego rodzaju.

Jeśli spróbujesz wprowadzać większe dokumenty, zauważysz, że pobranie wszystkich słów jest procesem długotrwałym. To niekorzystna cecha tej metody. Jeżeli bowiem pobieramy z dokumentu kolejno pojedyncze słowa, skolekcjonowanie wszystkich potrwa dłuższą chwilę.

Na szczęście Word udostępnia pewne uprawnienia eliminujące konieczność przeglądania całego dokumentu słowo po słowie w poszukiwaniu akapitu czy innego znacznika. Za pomocą obiektu Selection możemy uzyskać dostęp do całego tekstu dokumentu jednocześnie jako pojedynczego elementu i skopiowanie tego tekstu do standardowych komponentów C++Buildera, takich jak np. RichEdit (patrz rysunek 8.6).

Rysunek 8.6.
Kontrolka RichEdit1
wyświetla zawartość
tego rozdziału



Wróć do aplikacji, dodaj do formularza nowy przycisk o nazwie Button_FastLoad z etykietą „Szybkie Wczytanie”. Dodaj też obiekt RichEdit. Umieść kod z wydruku 8.11 w metodzie OnClick przycisku. Po uruchomieniu programu kliknij przycisk *Szybkie Wczytanie* — po chwili zobaczysz zawartość swojego dokumentu w oknie kontrolki RichEdit. Po skopiowaniu tekstu do standardowych kontrolki C++Buildera wydzielanie z niego poszczególnych słów będzie znacznie szybsze i łatwiejsze niż pobieranie z dokumentu każdego słowa z osobna.

Wydruk 8.11.
Program „Szybkie Wczytanie”

```
void __fastcall TForm1::Button_FastLoadClick(TObject *Sender)
{
    Variant my_doc;
    Variant my_word_app;
    Variant my_words;
    Variant my_selection;

    try
    {
        my_word_app = Variant::GetActiveObject("word.application");
    }
    catch (...)
    {
        //-- Word nie jest uruchomiony - wyświetl monit i zakończ pracę
    }
}
```

```

        ShowMessage("Uruchom program Word i spróbuj ponownie po załadowaniu dokumentu");
        return;
    }

    //-- Pobierz aktualnie aktywny dokument programu Word
    my_doc = my_word_app.OlePropertyGet("ActiveDocument");

    //-- Zaznacz cały tekst dokumentu
    my_doc.OleProcedure("Select");
    my_selection = my_word_app.OlePropertyGet("Selection");

    RichEdit1->Clear();
    RichEdit1->Text = my_selection.OlePropertyGet("Text");
}

```

Umieszczanie obiektów w dokumentach Worda

Potrąfimy już wykonywać podstawowe operacje pobierania tekstu dokumentu. Naszym kolejnym zadaniem będzie umieszczanie tekstu w dokumencie programu Word. Dysponując listą słów, nasz kolejny program utworzy nowy dokument i umieści w nim słowa z listy, w porządku alfabetycznym. Dokonamy tego z pomocą naszego dobrego znajomego — obiektu `Range`.

Wprowadzanie tekstu do dokumentu Worda

Istnieją dwie podstawowe metody wstawiania tekstu do dokumentu: `InsertBefore` oraz `InsertAfter`. Wstawiają one tekst, odpowiednio: na początek i na koniec zakresu `Range`. Rozszerzają też zakres tak, aby pokrywał również nowo wstawiony tekst. Musisz więc podjąć decyzję o miejscu wstawiania nowych wierszy i wywołać odpowiednią metodę.

Wprowadzanie obiektów do dokumentu Worda

Można również wstawiać do dokumentu elementy takie jak nowe akapity, podziały stron i inne informacje formatujące. Osobne metody umożliwiają dodawanie tabel, rysunków itp. Prześledźmy proces dodawania tabelki do dokumentu.

Jak wspominałem, w procesie wprowadzania tekstu wykorzystywane są obiekty zakresu — dodawanie pozostałych obiektów przebiega w podobny sposób. Różnica polega na tym, że zamiast obiektu `Range` użyjemy kolekcji, do której nowy obiekt miałby należeć, a wywołanie metody `Add` zastąpi nowym elementem aktywny zakres, o ile ten nie zostanie odpowiednio zamknięty. Zamknięcie zakresu polega na zrównaniu punktu startowego z końcowym punktem zakresu, co chroni przed możliwością utraty treści dokumentu podczas wstawiania nowych obiektów.

Kod tworzący tabelkę na końcu dokumentu zamieszczony jest na wydruku 8.12.

Wydruk 8.12.

Tworzenie tabelki

```

    long num_rows = 5;
    long num_columns = 3;

    //-- Pobierz zakres

```

```

my_doc = my_word.OlePropertyGet("ActiveDocument");
my_range = my_doc.OleFunction("Range");

/-- Zamknij zakres
my_range.OleProcedure("Collapse", (Variant)0);

/-- Wstaw tabelkę
my_tables = my_range.OlePropertyGet("Tables");
my_table = my_tables.OleFunction("Add", my_range, (Variant)num_rows,
    (Variant)num_columns);

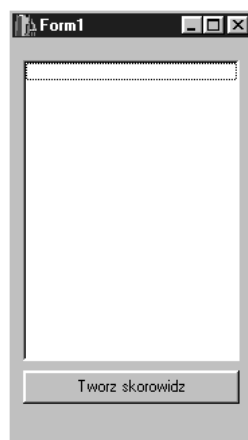
```

Powyższy kod pobiera aktualny zakres aktywnego dokumentu i zamyka go tak, że punkt wstawienia nowych obiektów znajduje się na jego końcu. Pomięcie zera lub zastąpienie go jedynką zmieni punkt wstawienia na punkt początkowy zakresu. Nowy obiekt `table` dodawany jest do kolekcji `tables`. Zakres jest używany jako parametr funkcji `Add`. Pokazujemy w ten sposób metodzie `Add` żądane miejsce wstawienia tabelki — nie dodajemy jej bezpośrednio do zakresu.

Tworzymy skorowidz

Wcześniej wspomniałem, że napiszemy program wstawiający do nowego dokumentu słowa zawarte w innym dokumencie. Programem tym będzie program *Skorowidz (Vocabulary)* (patrz rysunek 8.7). Możesz go znaleźć na płycie CD-ROM dołączonej do tej książki, w folderze *VocabA*. Projekt zawierający przykład jest opatrzony nazwą *vocab_proj.bpr*.

Rysunek 8.7.
Program „Skorowidz”



Program ten ładuje dokument Worda, pobiera całą jego zawartość do obiektu klasy `AnsiString` i wydziela z niej słowa, szukając spacji i innych znaków podziału. Po odnalezieniu słowa umieszcza go w posortowanej liście słów. Z listy tej automatycznie eliminowane są duplikaty. Następnie budowana jest tabelka — ta część programu najbardziej nas interesuje (patrz wydruk 8.13).

Wydruk 8.13.
Wstawianie słów do tabelki

```

void __fastcall TVocabForm::ReportWordsFromList(Variant doc, TListBox *lb)
{
    Variant my_tables;
    Variant this_range;

```

```
Variant this_table;
Variant start_cell;
Variant end_cell;
Variant cell_range;

long num_columns = 4;
long num_rows = (lb->Items->Count + num_columns - 1)/num_columns;
long words;
long this_column;
long this_row;

Procedure InsertAfter("InsertAfter"); //-- Procedura dla Exec

//-- Ustaw zakres obiektu
this_range = doc.OleFunction("Range");

//-- Pobierz kolekcję tabel z zakresu
my_tables = this_range.OlePropertyGet("Tables");

//-- Utwórz nową tabelkę zawierającą nagłówek i słowa
this_table = my_tables.OleFunction("Add", this_range, (Variant)(num_rows + 1),
    ⚡(Variant)num_columns);

start_cell = this_table.OleFunction("Cell", (Variant)1, (Variant)1);
end_cell = this_table.OleFunction("Cell", (Variant)1, (Variant)num_columns);

start_cell.OleProcedure("Merge", end_cell);

//-- Tytuł tabelki
cell_range = start_cell.OlePropertyGet("Range");
cell_range.Exec(InsertAfter << "Słowa z '" + OpenFileDialog->FileName + "'");

this_column = 1;
this_row = 2;

//-- Wstaw słowa z listy...
for (words = 0; words < lb->Items->Count; words++)
{
    start_cell = this_table.OleFunction("Cell", (Variant)this_row,
        ⚡(Variant)this_column);

    cell_range = start_cell.OlePropertyGet("Range");

    InsertAfter.ClearArgs();
    cell_range.Exec(InsertAfter << lb->Items->Strings[words]);

    this_column++;
    if (this_column > num_columns)
    {
        this_row++;
        this_column = 1;
    }
}
}
```

Ponadto program formatuje tabelkę w taki sposób, że nagłówek kolumny zawiera nazwę przetwarzanego pliku dokumentu. Dokonywane jest to za pośrednictwem kolekcji Cells (komórki), będącej składową tabelki. Do danej komórki tabeli można się odwołać, określając numer wiersza i kolumny komórki (wiersze i kolumny tabeli są numerowane od 1).

Obiekt `Cell` posiada metodę `Merge`, służącą do scalania podanego zakresu komórek. W programie scalimy pierwsze cztery komórki, aby uzyskać nagłówek tabeli.

Każda komórka posiada własny zakres. Może on być używany tak jak każdy inny obiekt zakresu. Możesz na przykład bez problemu stworzyć rekursywnie tabelkę zawierającą w lewej górnej komórce tabelkę itd. Kłopoty z taką konfiguracją zaczynają się dopiero w momencie wstawiania tekstu do komórki — potrzebujemy wtedy referencji do obiektu `Cells` i jego zakresu. Jest to bardziej uciążliwe niż problematyczne, ale można na tym utknąć.

Integracja z programem Excel

Teraz zajmiemy się inną aplikacją pakietu Microsoft Office — programem Excel. Podobnie jak Word, Excel udostępnia większość swoich mechanizmów przetwarzających jako obiekty automatyzacji. I, podobnie jak Word, używa kolekcji do przechowywania elementów dokumentu.

Z doświadczenia w pracy z Wordem i Excelem wiesz, że to bardzo różne w użytkowaniu programy — są także zupełnie inne w programowaniu. Na szczęście wiele mechanizmów programowania w Wordzie działa także z Excelem. Nawet jeśli pewien mechanizm nie ma swojego odpowiednika w Excelu, opanowanie jego obsługi nie jest trudne.

Obiekt aplikacji (Application Object)

Pozyskanie obiektu aplikacji Excela przebiega w taki sam sposób jak w przypadku obiektu aplikacji Worda. Jediną różnicą jest nazwa żądanego obiektu automatyzacji.

Aby programować z wykorzystaniem Excela, musimy uzyskać dostęp do obiektu `excel.application`:

```
Variant my_excel = Variant::CreateObject("excel.application");
```

Praca ze skoroszytami

W Wordzie bazową kolekcją obiektów jest dokument, zawierający cały tekst, tabelki, rysunki itp. W Excelu dokumenty traktowane są inaczej — używa się pojęcia skoroszytu (ang. *Workbook*), który może zawierać arkusze (ang. *Worksheets*). Arkusze przechowują tekst, formuły i inne elementy Excela.

Istnieją dwa sposoby dostępu do skoroszytów: pobranie z aplikacji kolekcji `Workbooks` lub odczytanie wartości właściwości `ActiveWorkbook`. Kolekcja `Workbooks` daje dostęp do wszystkich funkcji skoroszytów (dodawanie, otwieranie, zamykanie itd.), natomiast `ActiveWorkbook` umożliwia kontrolę jedynie aktywnego skoroszytu.

Tworzenie nowych skoroszytów

Nowy skoroszyt tworzymy metodą `Add` kolekcji `Workbooks` (skoroszyty). Możesz skorzystać z domyślnego szablonu skoroszytu lub też, planując nietypowe zadanie, przekazać do metody `Add` nazwę szablonu, na podstawie którego będziesz pracował. Excel

przypisuje każdemu nowemu skoroszytowi nazwę (*Book1*, *Book2* — *Zeszyt1*, *Zeszyt2* w polskiej wersji MS Office), wykorzystywaną np. przy późniejszym zapisie skoroszytu do pliku.

Nowy skoroszyt będzie utworzony z kilkoma czystymi arkuszami, w zależności od wartości właściwości `SheetsInNewWorkbook` obiektu aplikacji (patrz wydruk 8.14). Kod źródłowy przykładu znajdziesz w folderze *GetExcel* na dołączonej do książki płycie CD-ROM, w projekcie o nazwie *GetExcel_Project.bpr*.

Wydruk 8.14.

Tworzenie nowego skoroszytu

```
void __fastcall TForm_GetExcel::Button_NewWorkbookClick(TObject *Sender)
{
    Variant all_workbooks;
    Variant my_workbook;

    //-- Pobranie kolekcji skoroszytów
    all_workbooks = my_excel.OlePropertyGet("Workbooks");

    //-- Ustawienie liczby arkuszy na 1
    my_excel.OlePropertySet("SheetsInNewWorkbook", (Variant)1);

    //-- Utworzenie skoroszytu
    my_workbook = all_workbooks.OleFunction("Add");
}
```

Zapisywanie skoroszytów

Skoroszyt można zapisać na kilka sposobów; najczęstszym sposobem jest wykorzystanie metod `Save` i `SaveAs`. Metody te są bardzo podobne do swoich odpowiedników w Wordzie, jednak istnieje między nimi subtelna różnica. Ponieważ skoroszyty Excela tworzone są wraz z nazwami, mogą być zapisane bez pytania użytkownika o nazwę pliku. Metoda `Save` nie pyta więc o nazwę pliku, lecz wykorzystuje już przypisaną nazwę. Działanie funkcji `Save` demonstruje kod z wydruku 8.15.

Wydruk 8.15.

Zapisywanie aktywnego skoroszytu

```
void __fastcall TForm_GetExcel::Button_SaveClick(TObject *Sender)
{
    Variant my_workbook = my_excel.OlePropertyGet("ActiveWorkbook");
    my_workbook.OleProcedure("Save");
}
```

Funkcja `SaveAs` działa podobnie do `Save`. W rzeczywistości funkcja `SaveAs` wywołana bez określania dodatkowych parametrów działa dokładnie tak samo jak `Save`. Jednakże możesz podać parametry określające, w jakim pliku zapisany będzie dokument. W swej najprostszej postaci metoda `SaveAs` umożliwia zapisanie skoroszytu pod określoną nazwą, zamiast pod nazwą domyślną dokumentu. Spójrz na wydruk 8.16.

Wydruk 8.16.*Metoda SaveAs skoroszytu*

```
void __fastcall TForm_GetExcel::Button_SaveAsClick(TObject *Sender)
{
    Procedure SaveAs("SaveAs");
    Variant my_workbook = my_excel.OlePropertyGet("ActiveWorkbook");

    my_workbook.Exec(SaveAs << "C:\\moj_plik.xls");
}
```

Otwieranie istniejącego skoroszytu

Umiejętność tworzenia i zapisywania dokumentów nie wystarczy — trzeba również umieć otworzyć skoroszyt i odzyskać wyniki wcześniejszej pracy. Ta operacja również podobna jest do znanej nam z Worda — korzystamy z metody `Open` kolekcji `Workbooks`. Jeżeli znamy nazwę dokumentu, jego otwarcie sprowadza się do przekazania tej nazwy do metody (patrz wydruk 8.17).

Wydruk 8.17.*Otwieranie skoroszytu*

```
void __fastcall TForm_GetExcel::Button_OpenClick(TObject *Sender)
{
    Variant all_workbooks = my_excel.OlePropertyGet("workbooks");
    Procedure Open("Open");

    if (OpenDialog1->Execute())
    {
        all_workbooks.Exec(Open << OpenDialog1->FileName);
    }
}
```

Podobnie jak w Wordzie, metoda `Open` może służyć jako narzędzie importu, ponieważ odczyta każdy plik obsługiwany przez program Excel. Jeśli planujesz pracę z dużą liczbą plików tekstowych, rozważ wykorzystanie metody `OpenText`. Metoda ta ładuje zawartość pliku tekstowego do arkusza i próbuje podzielić tekst na gotowe do użycia komórki. Bardzo podobna do zwykłej metody `Open`, metoda `OpenText` służy do importu plików tekstowych i w związku z tym przyjmuje kilka dodatkowych parametrów. Po dodatkowe informacje odsyłam do plików pomocy.

Korzystanie z aktywnego skoroszytu

Obiekt aplikacji sprawdza, który ze skoroszytów jest aktualnie używany przez program Excel i udostępnia go jako właściwość `ActiveWorkbook`. Po uzyskaniu obiektu skoroszytu powinieneś wybrać jeden z przechowywanych w nim arkuszy (kolekcja `Worksheets`) lub pobrać arkusz aktywny (właściwość `ActiveSheet`). Jeśli arkusz, który zamierzasz używać, nie jest w danej chwili aktywny, musisz wywołać jego metodę `Activate`, aby uaktywnić go przed rozpoczęciem pracy.

Teraz możemy przystąpić do wydobywania informacji z Excela. W tym celu użyjemy obiektu Range. Jest on bardzo uniwersalny i może zostać wykorzystany do reprezentacji zbiorów komórek, obejmujących zarówno cały skoroszyt, jak i pojedynczą komórkę. Wydruk 8.18 zawiera przykład wykorzystania obiektu Range.



Należy zwrócić uwagę na konwencję, zgodnie z którą Microsoft nazwał różne właściwości i obiekty. Wiele referencji do obiektów uzyskuje się, pobierając wartość właściwości o tej samej nazwie. Przykładowo, referencja do obiektu Range uzyskiwana jest w wyniku odczytania właściwości Range. Konstruując aplikacje zintegrowane z pakietem Office, powinniśmy mieć świadomość, jakiego rodzaju obiektem aktualnie operujemy.

Wydruk 8.18.

Właściwość Range

```
//-- Pobierz referencję do komórek arkusza
my_range = my_worksheet.OlePropertyGet("Range");

//-- Pobierz referencję do komórek od A1 do E7
my_range = my_worksheet.OlePropertyGet("Range", (Variant)"A1:E7");

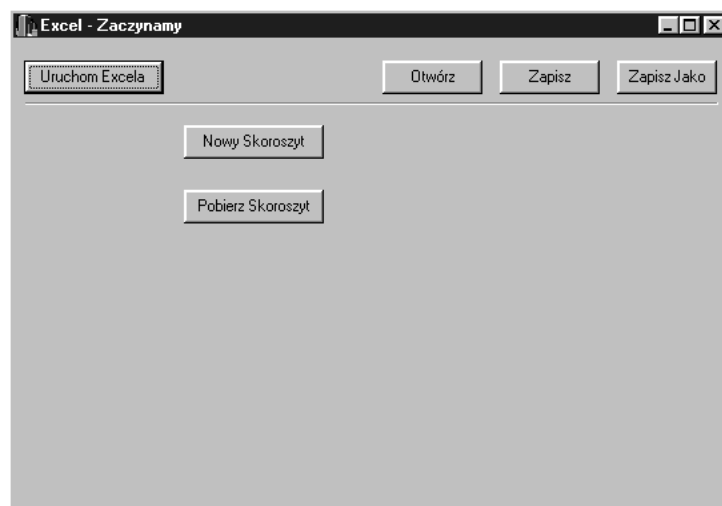
//-- Pobierz referencję do pojedynczej komórki
my_range = my_worksheet.OlePropertyGet("Range", (Variant)"C4");
```

Modyfikacja komórek arkuszy Excela

Każdy obiekt zakresu ma właściwości umożliwiające wprowadzanie danych i formuł do Excela. Są to właściwości Value i Formula. Właściwości te mogą modyfikować stan wielu komórek równocześnie, a korzystanie z nich przebiega w podobny sposób. Przeanalizuj obsługę zdarzenia OnClick na wydruku 8.19 pochodzącym z programu *Excel — Zaczynamy* (patrz rysunek 8.8).

Rysunek 8.8.

Program „Excel
— Zaczynamy”



Wydruk 8.19.*Ustawianie zawartości komórek*

```
void __fastcall TForm_GetExcel::Button_WorkbookClick(TObject *Sender)
{
    Variant my_workbook;
    Variant my_worksheet;
    Variant my_range;

    PropertyGet Range("Range");
    PropertySet SetValue("Value");
    PropertySet SetFormula("Formula");
    PropertyGet GetValue("Value");
    PropertyGet GetFormula("Formula");

    my_workbook = my_excel.OlePropertyGet("ActiveWorkbook");
    my_worksheet = my_workbook.OlePropertyGet("ActiveSheet");

    Range.ClearArgs();
    SetValue.ClearArgs();
    my_range = my_worksheet.Exec(Range << "A1");
    my_range.Exec(SetValue << "Mój arkusz");

    Range.ClearArgs();
    my_range = my_worksheet.Exec(Range << "B1:B5");
    my_range.Exec(SetFormula << "=rand()");

    Range.ClearArgs();
    SetFormula.ClearArgs();
    my_range = my_worksheet.Exec(Range << "A2");
    my_range.Exec(SetFormula << "=sum(b1:b5)");

    ShowMessage(my_range.OlePropertyGet("Value"));
    ShowMessage(my_range.OlePropertyGet("Formula"));
}
```

Główną różnicą pomiędzy obiema omawianymi właściwościami jest format wartości przypisywanych komórkom. Właściwość `Value` przyjmuje bezpośrednio przypisaną wartość, natomiast wartość przekazywana do właściwości `Formula` musi być sformatowana zgodnie z zasadami formatowania formuł w Excelu.

Są też inne sposoby wprowadzania danych do arkuszy, takie jak wykorzystanie właściwości `Text`, bardzo zbliżonej do właściwości `Value`, z tym że przypisywana lub odczytywana wartość zawsze traktowana jest jak ciąg znaków (nie jak liczba). W większości przypadków wystarczające jest wykorzystanie właściwości `Value`, dobrze jest jednak wyrobić sobie nawyk stosowania właściwości `Text` w przypadku ustawiania wartości, która ma być traktowana wyłącznie jako tekst. Wpisanie do komórki ciągu 12345 za pośrednictwem `Value` spowoduje przypisanie jej liczby 12345, niezależnie od tego, czy chciałeś wprowadzić liczbę, czy ciąg znaków.

Każdy obiekt `Range` posiada właściwości i metody służące do zmiany wyglądu pokrywanych przez niego komórek. Może to być proste — jak w przypadku zmiany kroju czcionki komórek, lub skomplikowane — np. gdy rysujemy ramki wokół bloków komórek.

Na przykład, właściwość `Font` obiektu `Range` zwraca obiekt `Font`, który z kolei posiada właściwości takie jak kolor (`Color`), styl (`Bold`, `Italic`, `Shadow` itd.) oraz rozmiar.

A oto inne właściwości przydatne w procesie formatowania komórek: kolekcja *Borders* (brzeży), właściwości *Height* (wysokość), *Width* (szerokość) i kolekcja *Interior* (wnętrze) Właściwości te określają styl ramek otaczających komórki, wielkość komórek, ich kolor i wzór wypełnienia.

Odczytywanie komórek arkusza Excela

Rzućmy okiem na ostatnią linię kodu z wydruku 8.19. Ta linia nie wprowadza żadnych informacji do komórki Excela, używając do odczytania zawartości komórki następującego zapisu:

```
ShowMessage(my_range.OlePropertyGet("Value"));
```

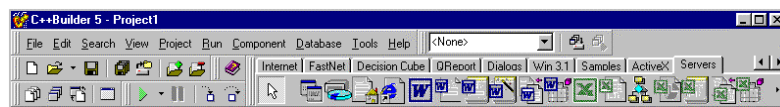
zamiast `OlePropertyGet`.

Metoda ta działa nie tylko w odniesieniu do wartości komórki, ale także jej formuły, więc możesz z jej pomocą pozyskiwać informacje o sposobie obliczania komórki.

Komponenty palety Servers C++Buildera 5

Poza możliwościami zastosowania technologii *OleAutomation*, z której korzystaliśmy dotychczas, C++Builder umożliwia również wykorzystanie w pracy nad projektem biblioteki *ATL* (*ActiveX Template Library*) i udostępnia jej mechanizmy za pośrednictwem komponentu *OleServer* i jego pochodnych. Na zakładce *Servers* palety komponentów C++Buildera umieszczonych jest wiele komponentów umożliwiających połączenie z *Wordem* i *Exclem* (patrz rysunek 8.9).

Rysunek 8.9.
Zakładka *Servers*
palety komponentów
C++Buildera



Komponenty te umożliwiają zintegrowanie tworzonych programów z pakietem *Office* i tym samym np. kontrolowanie *Worda* przez proste dołączenie ich do aplikacji i wypełnienie szkieletu ich kodu. Poznamy te możliwości, modyfikując nasz program *Sko-rowidz* w taki sposób, aby korzystał z komponentów *OleServer*.

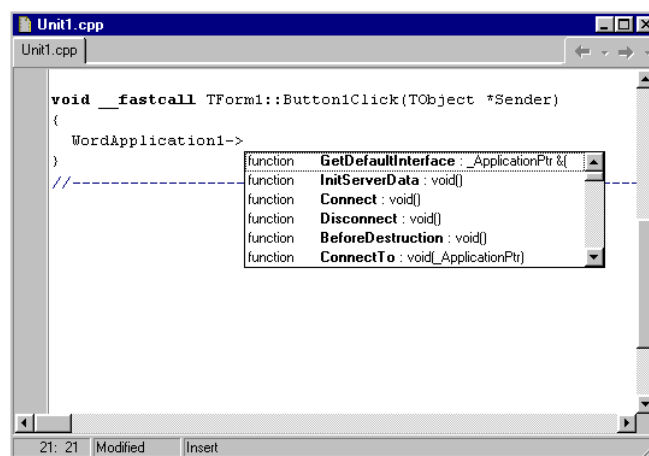
Zanim zaczniemy pracę nad nowym programem, przetestujmy kilka komponentów grupy *Server*. Włącz opcję uzupełniania kodu w opcjach edytora (menu *Tools/Editor Options*, opcja *Code Completion*) i stwórz nową aplikację. Następnie przeciągnij komponenty *WordApplication* i *WordDocument* na formularz główny. Utwórz też przycisk, a potem dwukrotnie kliknij go myszką, aby edytować kod przycisku.

Po wyświetleniu okienka edytora kodu wprowadź do niego następujący tekst i zaczekaj na uaktywnienie funkcji uzupełniania kodu:

```
WordApplication1->
```

Po chwili zostanie wyświetlona lista dostępnych metod i właściwości (patrz rysunek 8.10).

Rysunek 8.10.
Właściwości i metody
komponentu
WordApplication



Przewijając zawartość listy wyświetlonej przez C++Buildera, zauważysz, że wiele funkcji i właściwości używanych przez nas wcześniej w ramach *OleAutomation* wciąż jest dostępnych. Będzie to działało oczywiście na naszą korzyść, ale pamiętajmy — działanie niektórych funkcji i właściwości nie jest identyczne. Mają one zmienione nazwy, niektóre z nich otrzymały też dodatkowe parametry (inne zaś zostały pozbawione niektórych parametrów).

Komponenty WordApplication i WordDocument

Komponent *WordApplication* zawiera rdzeń aplikacji Worda, podobnie jak obiekt aplikacji używany przez nas wcześniej w rozdziale. Komponent *WordApplication* oferuje ściślejszą integrację z Wordem, jednak kosztem zwiększonej złożoności aplikacji. Twoje programy zyskają łatwiejszy dostęp do wielu nowych możliwości Worda i będą dokładniej sprawdzane podczas kompilacji (mechanizmy *OleAutomation* sprawdzały wywołania i parametry tylko w czasie działania programu).

Niestety, Twoje programy będą również większe i będą się dłużej kompilować niż aplikacje tworzone z wykorzystaniem *OleAutomation*. Ich działanie będzie ściśle zależne od jakości napisanego kodu (oraz od kodu komponentów C++Buildera), w mniejszym zaś stopniu od aplikacji, z którą będziesz integrował swój program. Komponent *WordApplication* jest odpowiednikiem obiektu *application* Worda. Analogicznie, komponent *WordDocument* reprezentuje wersję ATL obiektu *document* Worda. Przedstawione powyżej zastrzeżenia odnoszą się więc również do niego.

Mając to na uwadze, przejdźmy do głównego tematu podrozdziału — programowania nowej wersji *Skorowidza*.

Skorowidz — wersja druga

Bardziej zasadna od przepisywania całego kodu programu *Skorowidz* byłaby aktualizacja tych jego części, które dotyczą współpracy z programem Microsoft Word. Zaczniemy od kodu uruchamiającego Worda, a następnie przyjrzymy się metodom wydobywania tekstu z dokumentów, tworzenia nowych dokumentów oraz wprowadzania do nich tekstu.

Projekt zawierający omawiany przykład (*vocab_proj.bpr*) znajduje się w folderze *VocabB* na płycie CD-ROM dołączonej do książki.

Zaczynamy

Pierwszym zadaniem będzie nawiązanie połączenia z Wordem i przejęcie nad nim kontroli. Wcześniej należało sprawdzić, czy istnieje uruchomiony egzemplarz programu Word, a jeśli nie — uruchomić go. Tym razem, dzięki wykorzystaniu metody `Connect()`, możemy zaoszczędzić kilka linii kodu. Na wydruku 8.20 zamieszczony jest kod uruchamiający i wyświetlający interfejs programu Word.

Wydruk 8.20.

Uruchamianie programu Word

```
/*
//
// Metoda PrepareWord (Przygotuj program Word)
//
*/
void __fastcall TVocabForm::PrepareWord(void)
{
    try
    {
        WordApplication1->Connect();
    }
    catch(...)
    {
        ShowMessage("Nie można uruchomić Word'a");
    }
    WordApplication1->Visible = True;
}
```

Jak widać, kod jest podobny do tego z pierwotnego programu *Skorowidz*, z tym że jest nieco prostszy i przejrzystszy. Gdyby ominąć fragment kodu związany z obsługą wyjątku (nie zalecam), cały kod zmieściłby się w dwóch liniijkach.

Wspominałem już, że kontrola nad programem Word uzyskiwana jest za pośrednictwem metody `Connect()`. To jedna z podstawowych metod dostępnych w klasach pochodnych klasy `OleServer`. Wyszukuje ona kopię odpowiedniego serwera automatyzacji OLE i uzyskuje dostęp do istniejącego serwera lub — w razie niepowodzenia — tworzy jego nową instancję.

Pojawienie się okna programu było spowodowane przypisaniem odpowiedniej wartości do właściwości `Visible`. Jak widać w tym przykładzie, w przeciwieństwie do mechanizmów `OleAutomation`, właściwości obiektów klasy `OleServer` są dostępne bez pośrednictwa metod w rodzaju `OlePropertySet()`.

Pobranie tekstu z dokumentu Word'a

Po udanym uruchomieniu naszego interfejsu do Word'a musimy nauczyć się otwierać dokumenty i wydobywać z nich tekst. Będziemy więc musieli użyć komponentu `WordApplication` do otwarcia dokumentu i podłączyć do niego komponent `WordDocument` tak, aby uzyskać dostęp do zawartości załadowanego dokumentu.

Otwieranie dokumentu w Wordzie sprowadza się do użycia metody `Open()` kolekcji `Documents` (nie ulega to zmianie po przejściu z `OleAutomation` na `OleServer`).

Wydruk 8.21.

Nowa wersja kodu niezbędnego do otwarcia dokumentu i wydobywania jego zawartości

```
/*
//
// Wyodrębnij słowa dokumentu
//
*/
void __fastcall TVocabForm::Button_VocabClick(TObject *Sender)
{
    OleVariant FileName;
    wchar_t *doc_contents;
    RangePtr my_range;

    PrepareWord();

    if (OpenDialog1->Execute())
    {
        FileName = OpenDialog1->FileName;

        //-- Mamy nazwę pliku - otwórzmy dokument
        WordDocument1->ConnectTo(WordApplication1->Documents->Open(FileName));
        WordDocument1->Select();

        //-- Pobierzmy słowa z dokumentu
        my_range = WordDocument1->Range(EmptyParam, EmptyParam);
        doc_contents = my_range->get_Text();

        //-- Przekształćmy tekst do postaci listy słów
        //-- Przyspieszmy działanie listy, wyłączając sortowanie listy    //-- i ukrywając ją
        ListBox_Words->Sorted = false;
        ListBox_Words->Visible = false;
        ListBox_Words->Clear();

        TransferWordsToList(doc_contents, ListBox_Words);

        //-- Usuńmy duplikaty..
        RemoveDuplicateWordsFromList(ListBox_Words);

        //-- Wyświetlmy listę
        ListBox_Words->Visible = true;

        //-- Tworzymy nowy dokument
        WordDocument1->ConnectTo(WordApplication1->Documents->Add());

        //-- Przepiszmy skorowidz do nowego dokumentu
        ReportWordsFromList(ListBox_Words);
    }
}
```

Na pierwszy rzut oka stwierdzimy podobieństwo powyższego kodu do tego w oryginalnej wersji programu *Skorowidz*. Główną zmianą jest to, że nie wykonujemy już tak wielu czynności pośrednich. Zamiast kilku obiektów `OleVariant` przechowujących różne zmienne tymczasowe, używamy tylko trzech zmiennych.

Bardziej przejrzysty jest również kod służący do pobierania tekstu z dokumentu. Większość powyższego wydruku zajmuje kod służący do manipulowania listą słów, natomiast poprzednio większość kodu obsługiwała sterowanie Wordem.

Operacja ładowania dokumentu jest teraz zawarta w pojedynczej linii kodu:

```
WordDocument1->ConnectTo(WordApplication1->Documents->Open(FileName));
```

Mimo że wywołanie to obejmuje w rzeczywistości kilka linii kodu, jest ono dość przejrzyste. Kolekcję `Documents` otrzymujemy bezpośrednio z komponentu `WordApplication`, a następnie używamy metody `Open()` kolekcji do otwarcia dokumentu. Po załadowaniu pliku przekazujemy rezultat wywołania do metody `ConnectTo()`, uzyskując dostęp do zawartości dokumentu.

Moglibyśmy rozbić tę linię na kilka etapów. Mechanizm uzupełniania kodu odpowiedziałby nam nazwy różnych struktur i typów przekazywanych pomiędzy tymi wywołaniami. Nie jest to tak proste, jak zastosowanie pozbawionego typów wywołania `Ole Automation`, ale umożliwia uzyskanie szybszego, bezpośredniego dostępu do metod i właściwości.

Po otwarciu dokumentu zaznaczamy jego zawartość i kopiujemy ją do bufora tekstowego. Tak jak poprzednio, musimy wykorzystać obiekt `Range` do zaznaczenia interesującego nas obszaru.

```
my_range = WordDocument1->Range(EmptyParam, EmptyParam);
```

Przekazanie w wywołaniu parametru `EmptyParam` jest ekwiwalentem przekazania wartości `Null`. Powyższe wywołanie zaznacza cały dokument.

Po wyznaczeniu zakresu użyjemy metody `get_Text()` do pobrania tekstu. Zauważ, że używamy tu nazwy różniącej się nieznacznie od nazw w `Ole Automation`, ponieważ mamy bezpośredni dostęp do właściwości.

```
doc_contents = my_range->get_Text();
```

Powinniśmy przyjrzeć się bliżej metodzie `get_Text()`. Typ zwracanej przez nią wartości nie jest zwykłym typem znakowym, lecz ciągiem znaków rozszerzonych (ang. *wide char*). Nie powinno to stanowić problemu, jednak jeśli zauważysz dziwną zawartość ciągów zwracanych przez `get_Text()`, sprawdź zgodność typów, którymi operujesz, z typem `wchar_t`.

W naszym przykładzie nie zajmujemy się problemem konwersji tekstu rozszerzonego. Po prostu przeliczamy go prosto do funkcji `TransferWordsToList()`, której jeden z parametrów powinien być typu `AnsiString`. `C++Builder` zajmuje się resztą.

Umieszczanie tekstu w dokumencie Worda

Zaprezentowaliśmy już prosty sposób otwierania dokumentu i wydobywania z niego zawartości. Tworzenie i wypełnianie treścią nowych dokumentów jest równie proste.

Wydruk 8.21 zawiera fragment kodu realizujący powyższe zadanie. Przed wywołaniem funkcji `ReportWordsFromList()`, wstawiającej otrzymane słowa do dokumentu, tworzony jest nowy dokument.

Kod tworzący nowy dokument wygląda następująco:

```
WordDocument1->ConnectTo(WordApplication1->Documents->Add());
```

Ponownie używamy komponentu `WordApplication` w celu uzyskania kolekcji `Documents`, a następnie wywołujemy jedną z metod (w tym przypadku `Add()`) kolekcji. Ten fragment kodu w prosty sposób tworzy nowy dokument według szablonu domyślnego i zwraca uchwyt dokumentu.

Zamiast zachować uchwyt dokumentu, od razu podłączamy się do niego za pośrednictwem serwera dokumentu `WordDocument`. Po utworzeniu dokumentu wywoływana jest funkcja `ReportWordsFromList()`, wstawiająca listę słów do dokumentu (patrz wydruk 8.22).

Wydruk 8.22.

Wstawianie tekstu do dokumentu Worda

```
/*  
//  
// Wypisanie wszystkich słów z listy w raporcie  
//  
*/  
void __fastcall TVocabForm::ReportWordsFromList(TListBox *lb)  
{  
    long words;  
  
    WordDocument1->Range(EmptyParam, EmptyParam)->InsertAfter(StringToOleStr("Słowa z '"  
    &+ OpenFileDialog1->FileName + "'\n\n"));  
  
    for (words = 0; words < lb->Items->Count; words++)  
    {  
        WordDocument1->Range(EmptyParam, EmptyParam)->InsertAfter(StringToOleStr(lb-&  
        &Items->Strings[words]+"\n"));  
    }  
}
```

Wstawianie tekstu do dokumentu poprzedzane jest wyznaczeniem zakresu (nie chcemy niespodzianek ze znikającym tekstem!). Tekst wstawiany jest na początek lub koniec zakresu.

```
WordDocument1->Range(EmptyParam, EmptyParam)->InsertAfter(StringToOleStr(lb->Items-&  
&Strings[words] + "\n"));
```

Jak widzisz, nie jest to aż tak skomplikowane. Warto zaznaczyć, że istnieje potrzeba jawnego wprowadzania znaku nowej linii (`\n`). Podczas pierwszego uruchomienia programu nie wprowadzaliśmy znaków nowej linii i program wyświetlał mieszaninę słów.

Z pewnością zauważysz, że w tej wersji programu nie ma kodu wprowadzającego słowa do tabelki. Jest kilka przyczyn pominięcia tego fragmentu, jedną z nich była chęć pokazania, jak proste jest operowanie tekstem w ramach dokumentu po zaniechaniu używania `OleAutomation`. Drugą przyczyną jest fakt, że wstawianie tabelki do dokumentów z wykorzystaniem komponentu `WordDocument` nie jest już tak przejrzyste i proste jak w `OleAutomation`.

Zestaw funkcji udostępniany przez komponent `WordDocument` nie jest całkowicie zgodny z wywołaniami `OleAutomation` (pomogłoby to w konwersji kodu programów). To jeden z minusów stosowania biblioteki ATL — jeżeli jej twórcy z premedytacją blokują pewne możliwości, sam nic na to nie poradzisz.

Biblioteka ATL i komponenty OleServer — wnioski

Każde narzędzie programistyczne daje określone korzyści, ale i powoduje niedogodności związane z jego wykorzystaniem — nie inaczej jest w przypadku ATL i komponentów OleServer. Z ich pomocą możesz bardziej ściśle integrować własne programy z innymi aplikacjami, dzięki czemu otrzymujesz niedostępne wcześniej możliwości, a Twoje programy działają szybciej, omijając kilka warstw dostępu do aplikacji.

Kosztom tych udogodnień jest większe obciążenie C++Buildera w sensie dłuższego czasu kompilacji i większych rozmiarów Twoich programów; czasami bardziej ścisła integracja oznacza mniejszą elastyczność programu. Podczas instalacji C++Buildera 5 zostaniesz poproszony o określenie wersji pakietu Office, z którą będą pracować komponenty OleServer (Office 97 lub 2000). Możliwe, że używany przez nas komponent WordDocument jest zgodny z obiema wersjami MS Office, pewne jest natomiast, że mniej ściśle związki architektury OleAutomation działają niezależnie od wybranej wersji.

Poznaj swoje narzędzia tak, żeby wiedzieć, które z nich najlepiej odpowiada Twoim potrzebom. W niektórych sytuacjach zalecane jest wykorzystanie OleAutomation, ale istnieją zastosowania, w których nic nie zastąpi możliwości biblioteki ATL.

Co dalej?

Dotychczas poznaliśmy jedynie kilka sposobów korzystania z zaledwie dwóch aplikacji należących do pakietu Microsoft Office. Zostało jeszcze sporo do zrobienia.

Word

Poniżej przedstawione są przykładowe projekty wykorzystujące różne możliwości programu Word:

■ Menedżer dokumentów

Program wyszukuje dokumenty programu Word i konstruuje dla każdego z nich listę słów. Usuwa z listy najpopularniejsze słowa (który, na, i, w, oraz itp.), a pozostałe wprowadza do bazy danych, aby później można było odnaleźć dokument na podstawie zawartych w nim słów. Możesz nawet rozbudować ten program tak, aby przeszukiwał również foldery sieciowe i automatycznie aktualizował bazę dokumentów po ich zmianie.

■ Biblioteka raportów

Stwórz bibliotekę procedur tworzących raporty, wykorzystując możliwości aplikacji Word. Zamiast QReport, możesz użyć komponentu Word i tworzyć swoje raporty w rozpowszechnionym formacie. Wykorzystując możliwości Worda, nie tylko zyskasz na łatwości wymiany plików, ale będziesz mógł przejść kontrolę nad formatowaniem wyglądu dokumentu. Możesz nawet wprowadzić opcję zapisu w pliku HTML — można dzięki temu np. aktualizować strony WWW.

■ Menedżer korespondencji

Integracja z systemem poczty jest częstą cechą edytorów tekstu, ale edytory te nie zawsze współpracują z wykorzystywanymi przez Ciebie źródłami danych i nie zawsze oferują wszystkie pożądane możliwości. Przejmując kontrolę nad Wordem, możesz przeszukiwać każdy dokument w poszukiwaniu pól do wypełnienia. Możesz przechowywać kopie korespondencji każdego adresata, a nawet prowadzić audyt umożliwiający sprawdzenie, komu, gdzie i jakie dokumenty wysłałeś.

Excel

Możesz rozszerzyć współpracę własnych aplikacji z Excelem, wprowadzając do komórek wykresy, dane z bazy danych. Możesz używać ich do prezentowania i pozyskiwania informacji.

Przykładowe projekty:

■ Analizator dziennika serwera WWW

Pobierz ze swojej strony pliki dziennika, a następnie wypełnij nimi kolumny arkusza, umożliwiając analizę ruchu na Twojej stronie. Skoroszyt projektu może zawierać kilka arkuszy (jeden będzie zawierał nieprzetworzone dane, drugi — informacje zbiorcze, a w pozostałych będą umieszczone wykresy i raporty).

■ Kwestionariusz (ankieta)

Jeżeli musisz pozyskać od respondentów dużą ilość informacji, możesz rozważyć stworzenie arkusza zawierającego kwestionariusz (można też wykorzystać do tego celu zwykły dokument Worda). Napisz program importujący i przetwarzający dane pochodzące z kwestionariuszy otrzymanych od respondentów.

■ Lista zadań

Jeśli odczujesz potrzebę zorganizowania swojej pracy w formie listy zadań, możesz ją prowadzić z wykorzystaniem Excela. Każdy wiersz arkusza odpowiada jednemu zadaniu i zawiera jego status (zakończony, w toku itp.) oraz wymagany termin zakończenia. Napisz program prowadzący taką listę i przypominający okresowo o czekających zadaniach. Program ten mógłby też drukować listę zadań w przejrzystym formacie.

Pozostałe aplikacje pakietu Office

Oprócz najbardziej popularnych — Worda i Excela — w pakiecie Office zawarte są również inne programy użytkowe, a wiele spośród nich można kontrolować z wykorzystaniem automatyzacji znanej z Worda i Excela.

Outlook

Oprócz standardowego wykorzystania Outlooka jako programu pocztowego, może on także posłużyć jako harmonogram, lista zadań, system przypominający — jednym słowem asystent osobisty. Jego obiekt aplikacji dostępny jest jako `outlook.application`.

W przeciwieństwie do Worda czy Excela, w Outlooku w celu pozyskania interfejsu obiektu nie stosuje się zazwyczaj kolekcji obiektów, lecz wywołuje metody generujące interfejsy. Wydruk 8.23 zawiera kod tworzący nową wiadomość e-mail w programie Outlook:

Wydruk 8.23.

Tworzenie nowej wiadomości w programie Outlook

```
Variant my_outlook = Variant::CreateObject("outlook.application");
Variant my_message = my_outlook.OleFunction("CreateItem", (Variant)0);
    //-- 0 - Nowa wiadomość e-mail
my_message.OleProcedure("Display");
```

Access

Access może się wydawać niezbyt szczęśliwym kandydatem do integracji. C++Builder udostępnia lepszą obsługę baz danych, a Access — choć idealny jako narzędzie szybkiego tworzenia małych aplikacji bazodanowych — nie jest szczególnie niezawodny czy skalowalny. Nie jest też najczęściej używaną częścią pakietu Office (poza wersjami Professional i Small Business).

Pomimo tego C++Builder jest idealnym narzędziem do przenoszenia aplikacji z Accessa do innej bazy danych. Obiekt aplikacji programu Access udostępniany jest w odpowiedzi na żądanie referencji do obiektu `access.application`. Oto kilka propozycji aplikacji zintegrowanych z programem Access:

- Aplikacja odwzorowująca bazę danych

Ponieważ Access może być użytecznym narzędziem w fazie tworzenia prototypów aplikacji, przydatną jest możliwość uzyskania szczegółowego opisu wszystkich tabel i kwerend. Access udostępnia tę informację, lecz aby ją uzyskać, trzeba zaznaczać i sprawdzać każdą tabelkę z osobna. Proponowane narzędzie byłoby wygodniejsze, umożliwiając załadowanie bazy danych programu Access i wygenerowanie raportu na podstawie zawartych w niej tabel i kwerend.

- Aplikacja — marionetka

Jeżeli użytkownicy są przyzwyczajeni do pracy z Accessem i nie chcesz tego zmieniać, alternatywnym rozwiązaniem może być stworzenie aplikacji wykorzystującej program Access jako interfejs użytkownika, a faktycznie operującej na danych w tle za pomocą C++Buildera.

- Narzędzie pozyskiwania danych

Access jest dobrym kandydatem na narzędzie służące do kolekcjonowania danych — lepszym nawet niż Excel, jeśli dane, na których operujesz, są złożone. Z kolei C++Builder jest idealnym narzędziem do odpytywania bazy danych programu Access i zestawiania danych z kilku kopii bazy.

Project

Microsoft Project nie jest właściwie składową pakietu Office, ale udostępnia takie same możliwości integracji. Tworząc egzemplarz obiektu aplikacji `msproject.application`, uzyskujesz kolekcje projektów, zasobów, zadań itd. Wszystkie one dają się połączyć z innymi aplikacjami za pośrednictwem C++Buildera.

Oto kilka sugestii dla twórców aplikacji:

- Menedżer zadań zespołu

Każdy członek zespołu projektowego może posiadać program podpowiadający, którym z zadań powinien się aktualnie zajmować. Program może prowadzić audyt — zapamiętywać w pliku dziennika datę i czas pracy poszczególnych członków nad określonymi zadaniami. Rozwiązanie to posiada tę zaletę, że umożliwia uzyskanie odpowiedzi na pytania, jakie zadanie jest aktualnie realizowane, kto je realizuje oraz jakie zadania zagrożone są opóźnieniami.

- Prezentacja postępu prac nad projektem w postaci animacji

Można użyć C++Buildera do pozyskiwania danych o projekcie i zapamiętywania ich w bazie danych wersji projektów. Później można odtworzyć proces realizacji projektu, pobierając dane z bazy.

- System billingowy

Jedną z funkcji projektów zarządzanych za pomocą aplikacji Microsoft Project jest śledzenie kosztów zużycia zasobów związanych z danym projektem. Dane te można połączyć z systemem finansowo-księgowym, generującym faktury i inne dokumenty.

Inne aplikacje

Aplikacje obsługujące automatyzację OLE są coraz bardziej popularne. Metody używane w odniesieniu do pakietu Office są również przydatne w integracji z innymi programami.

Oto niektóre aplikacje umożliwiające integrację z wykorzystaniem OLE:

- FrontPage (frontpage.application),
- Visio (Visio.Application),
- Internet Explorer (InternetExplorer.Application),
- Adobe Photoshop (Photoshop.Application),
- Fusion (fusion.application).

Inne metody integracji

Poznaliśmy jedynie dwa sposoby sprawowania kontroli nad programami Microsoft Office z poziomu programu C++Builder. Rozwój pakietu C++Builder dał nam między innymi możliwość wykorzystania bibliotek typów. Wersja 5. programu udostępnia wiele serwerów OLE w postaci komponentów, daje też możliwość tworzenia kolejnych przez konstruowanie podklas klasy TOLEServer.

Podsumowanie

W tym rozdziale przeanalizowaliśmy sposoby integrowania aplikacji tworzonych w C++Builderze z programami pakietu Microsoft Office. Omówiliśmy główne metody sterowania pakietem Office z poziomu C++Buildera, a także wskazaliśmy kilka możliwych zastosowań takiej integracji.

Mimo że skupiliśmy się na Wordzie i Excelu, techniki, jakie poznaliśmy, mogą być używane w odniesieniu do większości aplikacji pakietu Office, a także kilku innych programów umożliwiających wykorzystanie dobrodziejstw automatyzacji. Poza metodami opisanymi w tym rozdziale, w celu uzyskania podobnych efektów można również wykorzystać alternatywne mechanizmy automatyzacji.

Tworzenie i manipulowanie dokumentami jest użyteczne, ale przede wszystkim daje możliwość wprowadzania i pozyskiwania ich zawartości, co czyni integrację potężnym narzędziem. Integracja aplikacji z programami pakietu Office wprowadza sporo komplikacji, ale nie jest bardzo trudna. Temat jest obszerny, ponieważ Office to bardzo rozbudowana kolekcja złożonych programów. Umiejętność uruchomienia aplikacji i wykonania za jej pomocą prostych zadań można nabyć szybko i łatwo, ale później trzeba jeszcze długo uczyć się sposobów realizacji konkretnych zadań z wykorzystaniem programów pakietu Office. Nauka ta może być źródłem wielkiej satysfakcji.

Trzymaj więc pod ręką pliki pomocy i śmiało eksperymentuj. Jeżeli nie masz pewności co do działania któregoś z poleceń, zastosuj je. Office szybko da Ci znać, jeśli nie będzie mógł zrealizować wydawanych przez Ciebie poleceń.