

Microsoft®

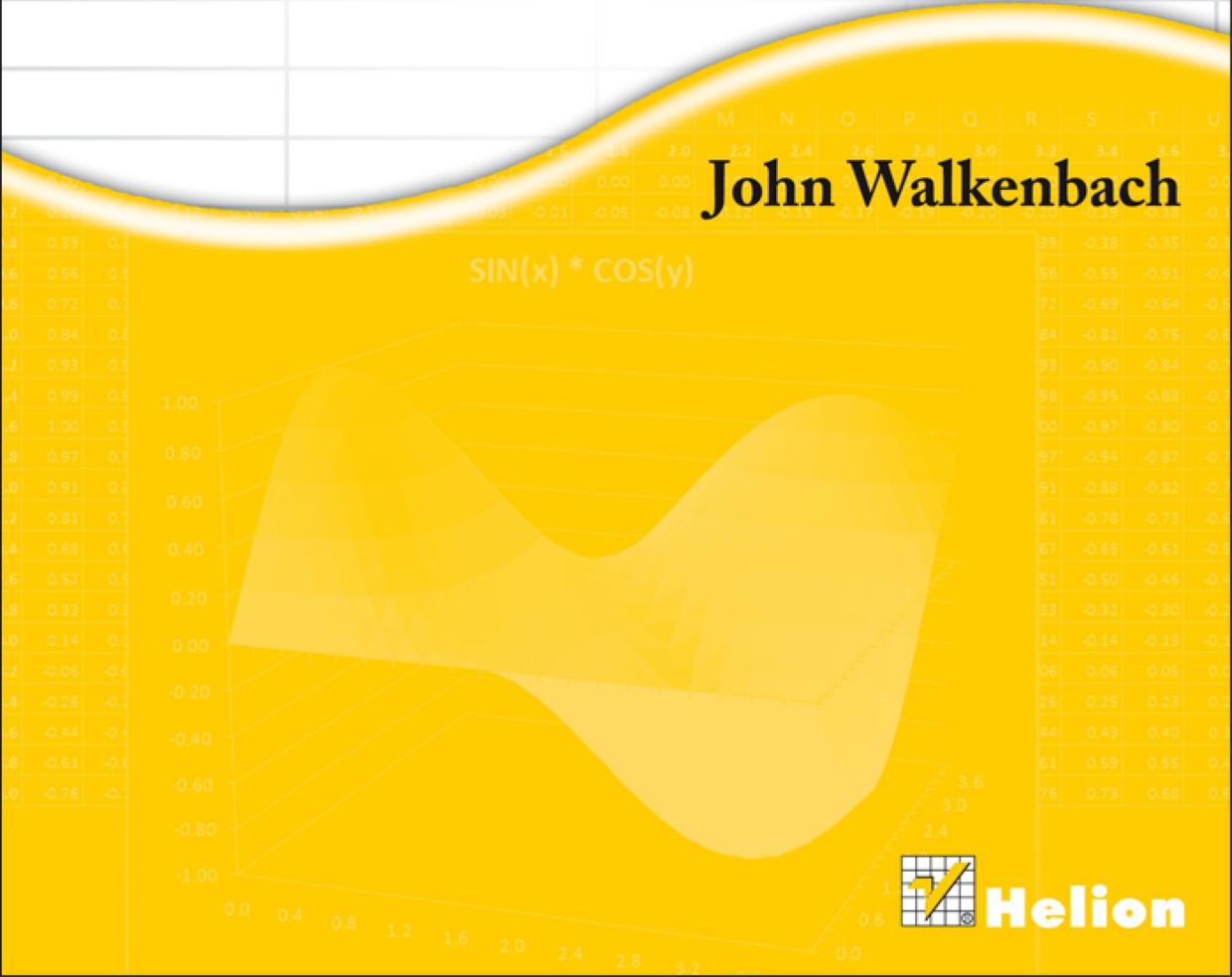
Excel® 2013 PL

Programowanie w VBA.

Vademecum Walkenbacha

John Walkenbach

SIN(x) * COS(y)



Helion

Tytuł oryginału: Excel 2013 Power Programming with VBA

Thumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-246-7895-2

Copyright © 2013 by John Wiley & Sons, Inc., Hoboken, New Jersey.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.

All rights reserved. This translation published under license with the original publisher John Wiley & Sons, Inc.

Translation copyright © 2013 by Helion S.A.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise without either the prior written permission of the Publisher.

Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. Excel is a registered trademark of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiekolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

http://helion.pl/user/opinie/e23pvw_ebook

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• [Poleć książkę na Facebook.com](#)

• [Kup w wersji papierowej](#)

• [Oceń książkę](#)

• [Księgarnia internetowa](#)

• [Lubię to! » Nasza społeczność](#)

Spis treści

O autorze	19
Przedmowa	21
Część I Podstawowe informacje	29
Rozdział 1. Program Excel w zarysie	31
O Excelu	31
Myślenie w kategoriach obiektów	31
Skoroszyty	32
Arkusze	33
Arkusze wykresów	34
Arkusze makr XLM	34
Arkusze dialogowe programów Excel 5 i 95	35
Interfejs użytkownika programu Excel	35
Wprowadzenie do Wstążki	37
Menu podrzędne i minipasek narzędzi	44
Okna dialogowe	45
Panel zadań	46
Skróty klawiszowe	47
Wprowadzanie danych	48
Formuły, funkcje i nazwy	48
Zaznaczanie obiektów	50
Formatowanie	51
Opcje ochrony	52
Ochrona formuł przed nadpisaniem	52
Ochrona struktury skoroszytu	53
Ochrona skoroszytu przy użyciu hasła	53
Ochrona kodu VBA przy użyciu hasła	54
Wykresy	55
Kształty i obiekty typu SmartArt	56
Dostęp do baz danych	56
Arkuszowe bazy danych	57
Zewnętrzne bazy danych	58
Funkcje internetowe	58
Narzędzia analizy danych	59
Dodatki	60
Makra i programowanie	60
Zgodność formatu plików	60
System pomocy Excela	61

Rozdział 2. Wybrane zasady stosowania formuł	63
Formuły	63
Obliczanie formuł	64
Odwołania do komórki lub zakresu	65
Dlaczego warto używać odwołań, które nie są względne?	66
Notacja W1K1	66
Odwołania do innych arkuszy lub skoroszytów	67
Zastosowanie nazw	69
Nadawanie nazw komórkom i zakresom	69
Nadawanie nazw istniejącym odwołaniom	71
Stosowanie nazw z operatorem przecięcia	71
Nadawanie nazw kolumnom i wierszom	71
Zasięg nazw zakresów	72
Nadawanie nazw stałym	73
Nadawanie nazw formułom	73
Nadawanie nazw obiektom	75
Błędy występujące w formułach	75
Narzędzia inspekcji	76
Identyfikowanie komórek określonego typu	77
Przeglądanie formuł	78
Śledzenie zależności między komórkami arkusza	79
Śledzenie błędów	82
Naprawianie błędów spowodowanych odwołaniami cyklicznymi	82
Zastosowanie mechanizmu sprawdzania błędów w tle	82
Zastosowanie mechanizmu szacowania formuł	84
Formuły tablicowe	85
Przykładowa formuła tablicowa	85
Kalendarz oparty na formule tablicowej	86
Zalety i wady formuł tablicowych	87
Metody liczenia i sumowania	88
Przykłady formuł liczących	88
Przykłady formuł sumujących	89
Inne narzędzia liczące	90
Formuły wyszukiwanie i adresu	91
Przetwarzanie daty i czasu	93
Wprowadzanie daty i czasu	94
Przetwarzanie dat sprzed roku 1900	94
Tworzenie megaformuł	95
Rozdział 3. Pliki programu Excel	99
Uruchamianie Excela	99
Formaty plików	100
Formaty plików obsługiwane w programie Excel	101
Formaty plików tekstowych	102
Formaty plików baz danych	103
Inne formaty plików	103
Kompatybilność plików Excela	105
Widok chroniony	106
Zastosowanie mechanizmu Autoodzyskiwania	107
Odzyskiwanie poprzednich wersji bieżącego skoroszytu	107
Odzyskiwanie niezapisanych skoroszytów	108
Konfigurowanie mechanizmu Autoodzyskiwania	108

Praca z plikami szablonów	108
Przeglądanie dostępnych szablonów	109
Tworzenie szablonów	109
Tworzenie szablonów skoroszytu	111
Budowa plików programu Excel	112
Zaglądamy do wnętrza pliku	112
Dlaczego format pliku jest taki ważny?	115
Plik OfficeUI	116
Plik XLB	117
Pliki dodatków	118
Ustawienia Excela w rejestrze systemu Windows	119
Rejestr systemu Windows	119
Ustawienia Excela	120
Rozdział 4. Podstawy projektowania aplikacji arkusza kalkulacyjnego	123
Czym jest aplikacja arkusza kalkulacyjnego?	123
Podstawowe etapy projektowania	124
Określanie wymagań użytkownika	125
Planowanie aplikacji spełniającej wymagania użytkownika	126
Wybieranie odpowiedniego interfejsu użytkownika	128
Dostosowywanie Wstążki do potrzeb użytkownika	129
Dostosowywanie menu podręcznego do potrzeb użytkownika	129
Tworzenie klawiszy skrótu	129
Tworzenie niestandardowych okien dialogowych	130
Zastosowanie formantów ActiveX w arkuszu	131
Rozpoczęcie prac projektowych	133
Zadania realizowane z myślą o końcowym użytkowniku	134
Testowanie aplikacji	134
Uodpornianie aplikacji na błędy popełniane przez użytkownika	136
Nadawanie aplikacji przyjaznego, intuicyjnego i estetycznego wyglądu	137
Tworzenie systemu pomocy i dokumentacji przeznaczonej dla użytkownika	139
Dokumentowanie prac projektowych	140
Przekazanie aplikacji użytkownikom	140
Aktualizacja aplikacji (kiedy to konieczne)	140
Pozostałe kwestie dotyczące projektowania	141
Wersja Excela zainstalowana przez użytkownika	141
Wersje językowe	141
Wydajność systemu	142
Tryby karty graficznej	142
Część II Język Visual Basic for Applications	145
Rozdział 5. Wprowadzenie do języka VBA	147
Podstawowe informacje o języku BASIC	147
Język VBA	148
Modele obiektowe	148
Porównanie języka VBA z językiem XML	148
Wprowadzenie do języka VBA	149
Edytor VBE	152
Wyświetlanie karty DEVELOPER	152
Uruchamianie edytora VBE	153
Okna edytora VBE	153

Tajemnice okna Project Explorer	155
Dodawanie nowego modułu VBA	156
Usuwanie modułu VBA	156
Eksportowanie i importowanie obiektów	157
Tajemnice okna Code	157
Minimalizacja i maksymalizacja okien	158
Przechowywanie kodu źródłowego języka VBA	158
Wprowadzanie kodu źródłowego języka VBA	159
Dostosowywanie środowiska edytora Visual Basic	166
Karta Editor	166
Karta Editor Format	169
Karta General	170
Zastosowanie karty Docking	171
Rejestrator makr Excela	172
Co właściwie zapisuje rejestrator makr?	173
Odwołania względne czy bezwzględne?	173
Opcje związane z rejestrowaniem makr	177
Modyfikowanie zarejestrowanych makr	178
Obiekty i kolekcje	180
Hierarchia obiektów	180
Kolekcje	181
Odwoływanie się do obiektów	181
Właściwości i metody	182
Właściwości obiektów	182
Metody obiektowe	183
Tajemnice obiektu Comment	185
Pomoc dla obiektu Comment	185
Właściwości obiektu Comment	185
Metody obiektu Comment	185
Kolekcja Comments	187
Właściwość Comment	188
Obiekty zawarte w obiekcie Comment	189
Sprawdzanie, czy komórka posiada komentarz	190
Dodawanie nowego obiektu Comment	190
Kilka przydatnych właściwości obiektu Application	191
Tajemnice obiektów Range	193
Właściwość Range	193
Właściwość Cells	194
Właściwość Offset	198
Co należy wiedzieć o obiektach?	199
Podstawowe zagadnienia, które należy zapamiętać	199
Dodatkowe informacje na temat obiektów i właściwości	200
Rozdział 6. Podstawy programowania w języku VBA	203
Przegląd elementów języka VBA	203
Komentarze	205
Zmienne, typy danych i stałe	206
Definiowanie typów danych	208
Deklarowanie zmiennych	209
Zasięg zmiennych	213
Zastosowanie stałych	216
Praca z łańcuchami tekstu	218
Przetwarzanie dat	218
Instrukcje przypisania	220
Tablice	222

Deklarowanie tablic	222
Deklarowanie tablic wielowymiarowych	223
Deklarowanie tablic dynamicznych	223
Zmienne obiektowe	224
Typy danych definiowane przez użytkownika	225
Wbudowane funkcje VBA	226
Praca z obiektami i kolekcjami	227
Konstrukcja With ... End With	229
Konstrukcja For Each ... Next	229
Sterowanie sposobem wykonywania procedur	231
Polecenie GoTo	232
Konstrukcja If ... Then	232
Konstrukcja Select Case	236
Wykonywanie bloku instrukcji w ramach pętli	240
Rozdział 7. Tworzenie procedur w języku VBA	249
Kilka słów o procedurach	249
Deklarowanie procedury Sub	250
Zasięg procedury	251
Wykonywanie procedur Sub	252
Uruchamianie procedury przy użyciu polecenia Run Sub/UserForm	253
Uruchamianie procedury z poziomu okna dialogowego Makro	253
Uruchamianie procedury przy użyciu skrótu z klawiszem Ctrl	254
Uruchamianie procedury za pomocą Wstążki	255
Uruchamianie procedur za pośrednictwem niestandardowego menu podręcznego	255
Wywoływanie procedury z poziomu innej procedury	256
Uruchamianie procedury poprzez kliknięcie obiektu	259
Wykonywanie procedury po wystąpieniu określonego zdarzenia	261
Uruchamianie procedury z poziomu okna Immediate	262
Przekazywanie argumentów procedurom	262
Metody obsługi błędów	266
Przechwytywanie błędów	266
Przykłady kodu źródłowego obsługującego błędy	268
Praktyczny przykład wykorzystujący procedury Sub	271
Cel	271
Wymagania projektowe	271
Co już wiesz	272
Podejście do zagadnienia	273
Co musimy wiedzieć?	273
Wstępne rejestrowanie makr	273
Wstępne przygotowania	275
Tworzenie kodu źródłowego	275
Tworzenie procedury sortującej	277
Dodatkowe testy	281
Usuwanie problemów	282
Dostępność narzędzia	285
Ocena projektu	285
Rozdział 8. Tworzenie funkcji w języku VBA	287
Porównanie procedur Sub i Function	287
Dlaczego tworzymy funkcje niestandardowe?	288
Twoja pierwsza funkcja	289
Zastosowanie funkcji w arkuszu	289
Zastosowanie funkcji w procedurze języka VBA	290
Analiza funkcji niestandardowej	291

Procedury Function	293
Zasięg funkcji	294
Wywoływanie procedur Function	294
Argumenty funkcji	298
Przykłady funkcji	299
Funkcja bezargumentowa	299
Funkcje jednoargumentowe	302
Funkcje z dwoma argumentami	305
Funkcja pobierająca tablicę jako argument	305
Funkcje z argumentami opcjonalnymi	306
Funkcje zwracające tablicę VBA	308
Funkcje zwracające wartość błędu	311
Funkcje o nieokreślonej liczbie argumentów	312
Emulacja funkcji arkuszowej SUMA	313
Rozszerzone funkcje daty	317
Wykrywanie i usuwanie błędów w funkcjach	319
Okno dialogowe Wstawianie funkcji	320
Zastosowanie metody MacroOptions	321
Definiowanie kategorii funkcji	323
Dodawanie opisu funkcji	324
Zastosowanie dodatków do przechowywania funkcji niestandardowych	325
Korzystanie z Windows API	326
Przykłady zastosowania funkcji interfejsu API systemu Windows	327
Identyfikacja katalogu domowego systemu Windows	327
Wykrywanie wcisnięcia klawisza Shift	328
Dodatkowe informacje na temat funkcji interfejsu API	329
Rozdział 9. Przykłady i techniki programowania w języku VBA	331
Nauka poprzez praktykę	331
Przetwarzanie zakresów	332
Kopiowanie zakresów	332
Przenoszenie zakresów	334
Kopiowanie zakresu o zmiennej wielkości	334
Zaznaczanie oraz identyfikacja różnego typu zakresów	335
Zmiana rozmiaru zakresu komórek	337
Wprowadzanie wartości do komórki	338
Wprowadzanie wartości do następnej pustej komórki	339
Wstrzymywanie działania makra w celu umożliwienia pobrania zakresu wyznaczonego przez użytkownika	341
Zliczanie zaznaczonych komórek	342
Określanie typu zaznaczonego zakresu	344
Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli	346
Usuwanie wszystkich pustych wierszy	348
Powielanie wierszy	349
Określanie, czy zakres zawiera się w innym zakresie	351
Określanie typu danych zawartych w komórce	351
Odczytywanie i zapisywanie zakresów	352
Lepsza metoda zapisywania danych do zakresu komórek	354
Przenoszenie zawartości tablic jednowymiarowych	355
Przenoszenie zawartości zakresu do tablicy typu Variant	356
Zaznaczanie komórek na podstawie wartości	357
Kopiowanie nieciągłego zakresu komórek	358
Przetwarzanie skoroszytów i arkuszy	360
Zapisywanie wszystkich skoroszytów	360
Zapisywanie i zamknięcie wszystkich skoroszytów	360

Ukrywanie wszystkich komórek arkusza poza zaznaczonym zakresem	361
Tworzenie spisu treści zawierającego hiperłącza	362
Synchronizowanie arkuszy	363
Techniki programowania w języku VBA	364
Przełączanie wartości właściwości typu logicznego	364
Wyświetlanie daty i czasu	365
Wyświetlanie czasu w formie przyjaznej dla użytkownika	367
Pobieranie listy czcionek	368
Sortowanie tablicy	368
Przetwarzanie grupy plików	370
Ciekawe funkcje, których możesz użyć w swoich projektach	372
Funkcja FileExists	372
Funkcja FileNameOnly	372
Funkcja PathExists	373
Funkcja RangeNameExists	373
Funkcja SheetExists	374
Funkcja WorkbookIsOpen	375
Pobieranie wartości z zamkniętego skoroszytu	375
Użyteczne, niestandardowe funkcje arkuszowe	376
Funkcje zwracające informacje o formatowaniu komórk	377
Gadający arkusz?	378
Wyświetlanie daty zapisania lub wydrukowania pliku	379
Obiekty nadzędne	380
Zliczanie komórek, których wartości znajdują się pomiędzy dwoma wartościami	381
Wyznaczanie ostatniej niepustej komórki kolumny lub wiersza	381
Czy dany łańcuch tekstu jest zgodny ze wzorcem?	383
Wyznaczanie n-tego elementu łańcucha	384
Zamiana wartości na słowa	385
Funkcja wielofunkcyjna	386
Funkcja SHEETOFFSET	386
Zwracanie maksymalnej wartości ze wszystkich arkuszy	387
Zwracanie tablicy zawierającej unikatowe, losowo uporządkowane liczby całkowite	388
Porządkowanie zakresu w losowy sposób	389
Sortowanie zakresów	391
Wywołanie funkcji interfejsu Windows API	392
Określanie skojarzeń plików	393
Pobieranie informacji o napędach dyskowych	394
Pobieranie informacji dotyczących drukarki domyślnej	394
Pobieranie informacji o aktualnej rozdzielcości karty graficznej	395
Odczytywanie zawartości rejestru systemu Windows i zapisywanie w nim danych	397
Część III Praca z formularzami UserForm	401
Rozdział 10. Tworzenie własnych okien dialogowych	403
Zanim rozpocznesz tworzenie formularza UserForm	403
Okno wprowadzania danych	403
Funkcja InputBox języka VBA	404
Metoda InputBox Excela	406
Funkcja MsgBox języka VBA	409
Metoda GetOpenFilename programu Excel	413
Metoda GetSaveAsFilename programu Excel	416
Okno wybierania katalogu	417
Wyświetlanie wbudowanych okien dialogowych Excela	417

Wyświetlanie formularza danych	420
Wyświetlanie formularza wprowadzania danych	420
Wyświetlanie formularza wprowadzania danych za pomocą VBA	422
Rozdział 11. Wprowadzenie do formularzy UserForm	423
Jak Excel obsługuje niestandardowe okna dialogowe	423
Wstawianie nowego formularza UserForm	424
Dodawanie formantów do formularza UserForm	424
Formanty okna Toolbox	425
Formant CheckBox	426
Formant ComboBox	426
Formant CommandButton	427
Formant Frame	427
Formant Image	427
Formant Label	427
Formant ListBox	427
Formant MultiPage	427
Formant OptionButton	428
Formant RefEdit	428
Formant ScrollBar	428
Formant SpinButton	428
Formant TabStrip	428
Formant TextBox	428
Formant ToggleButton	429
Modyfikowanie formantów formularza UserForm	430
Modyfikowanie właściwości formantów	431
Zastosowanie okna Properties	432
Wspólne właściwości	433
Uwzględnienie wymagań użytkowników preferujących korzystanie z klawiatury	434
Wyświetlanie formularza UserForm	436
Zmiana położenia formularza na ekranie	436
Wyświetlanie niemodalnych okien formularzy UserForm	436
Wyświetlanie formularza UserForm na podstawie zmiennej	437
Ładowanie formularza UserForm	437
Procedury obsługi zdarzeń	437
Zamykanie formularza UserForm	438
Przykład tworzenia formularza UserForm	439
Tworzenie formularza UserForm	439
Tworzenie kodu procedury wyświetlającej okno dialogowe	442
Testowanie okna dialogowego	443
Dodawanie procedur obsługi zdarzeń	444
Sprawdzanie poprawności danych	445
Zakończenie tworzenia okna dialogowego	445
Zdarzenia powiązane z formularzem UserForm	445
Zdobywanie informacji na temat zdarzeń	446
Zdarzenia formularza UserForm	447
Zdarzenia związane z formantem SpinButton	447
Współpraca formantu SpinButton z formantem TextBox	449
Odwoływanie się do formantów formularza UserForm	451
Dostosowywanie okna Toolbox do własnych wymagań	453
Dodawanie nowych kart	453
Dostosowywanie lub łączenie formantów	453
Dodawanie nowych formantów ActiveX	455
Tworzenie szablonów formularzy UserForm	455
Lista kontrolna tworzenia i testowania formularzy UserForm	456

Rozdział 12. Przykłady formularzy UserForm	459
Tworzenie formularza UserForm pełniącego funkcję menu	459
Zastosowanie w formularzu UserForm formantów CommandButton	460
Zastosowanie w formularzu UserForm formantu ListBox	460
Zaznaczanie zakresów przy użyciu formularza UserForm	461
Tworzenie okna powitalnego	463
Wyłączanie przycisku Zamknij formularza UserForm	465
Zmiana wielkości formularza UserForm	466
Powiększanie i przewijanie arkusza przy użyciu formularza UserForm	468
Zastosowania formantu ListBox	469
Tworzenie listy elementów formantu ListBox	470
Identyfikowanie zaznaczonego elementu listy formantu ListBox	475
Identyfikowanie wielu zaznaczonych elementów listy formantu ListBox	475
Wiele list w jednym formancie ListBox	476
Przenoszenie elementów listy formantu ListBox	477
Zmiana kolejności elementów listy formantu ListBox	479
Wielokolumnowe formanty ListBox	480
Zastosowanie formantu ListBox do wybierania wierszy arkusza	482
Uaktywnianie arkusza za pomocą formantu ListBox	484
Zastosowanie formantu MultiPage na formularzach UserForm	487
Korzystanie z formantów zewnętrznych	488
Animowanie etykiet	490
Rozdział 13. Zaawansowane techniki korzystania z formularzy UserForm	495
Niemodalne okna dialogowe	495
Wyświetlanie wskaźnika postępu zadania	499
Tworzenie samodzielniego wskaźnika postępu zadania	500
Wyświetlanie wskaźnika postępu zadania za pomocą formantu MultiPage	504
Wyświetlanie wskaźnika postępu zadania bez korzystania z kontrolki MultiPage	507
Tworzenie kreatorów	508
Konfigurowanie formantu MultiPage w celu utworzenia kreatora	509
Dodawanie przycisków do formularza UserForm kreatora	510
Programowanie przycisków kreatora	510
Zależności programowe w kreatorach	512
Wykonywanie zadań za pomocą kreatorów	513
Emulacja funkcji MsgBox	514
Emulacja funkcji MsgBox: kod funkcji MyMsgBox	515
Jak działa funkcja MyMsgBox	516
Wykorzystanie funkcji MyMsgBox do emulacji funkcji MsgBox	517
Formularz UserForm z formantami, których położenie można zmieniać	518
Formularz UserForm bez paska tytułowego	519
Symulacja paska narzędzi za pomocą formularza UserForm	520
Emulowanie panelu zadań za pomocą formularza UserForm	523
Formularze UserForm z możliwością zmiany rozmiaru	524
Obsługa wielu przycisków formularza UserForm	528
za pomocą jednej procedury obsługi zdarzeń	528
Wybór koloru za pomocą formularza UserForm	531
Wyświetlanie wykresów na formularzach UserForm	532
Zapisywanie wykresu w postaci pliku GIF	533
Modyfikacja właściwości Picture formantu Image	534
Tworzenie półprzeczarzystych formularzy UserForm	534
Zaawansowane formularze danych	536
Opis ulepszonego formularza danych	537
Instalacja dodatku — ulepszonego formularza danych	537
Puzzle na formularzu UserForm	538
Video Poker na formularzu UserForm	540

Część IV Zaawansowane techniki programowania	541
Rozdział 14. Tworzenie narzędzi dla Excela w języku VBA	543
Kilka słów o narzędziach dla programu Excel	543
Zastosowanie języka VBA do tworzenia narzędzi	544
Co decyduje o przydatności narzędzia?	545
Operacje tekstowe: anatomia narzędzia	545
Kilka słów o programie Operacje tekstowe	546
Określenie wymagań dla narzędzia Operacje tekstowe	547
Skoroszyt narzędzia Operacje tekstowe	547
Jak działa narzędzie Operacje tekstowe?	548
Formularz UserForm dla narzędzia Operacje tekstowe	549
Moduł VBA Module1	550
Moduł formularza UserForm1	552
Poprawa wydajności narzędzia Operacje tekstowe	554
Zapisywanie ustawień narzędzia Operacje tekstowe	555
Implementacja procedury Cofnij	557
Wyświetlanie pliku pomocy	559
Umieszczanie polecień na Wstążce	560
Ocena realizacji projektu	560
Działanie narzędzia Operacje tekstowe	562
Dodatkowe informacje na temat narzędzi Excela	562
Rozdział 15. Tabele przestawne	563
Przykład prostej tabeli przestawnej	563
Tworzenie tabel przestawnych	564
Analiza zarejestrowanego kodu tworzenia tabeli przestawnej	565
Optymalizacja wygenerowanego kodu tworzącego tabelę przestawną	566
Tworzenie złożonych tabel przestawnych	569
Kod tworzący tabelę przestawną	570
Jak działa złożona tabela przestawna?	571
Jednoczesne tworzenie wielu tabel przestawnych	573
Tworzenie odwróconych tabel przestawnych	576
Rozdział 16. Wykresy	579
Podstawowe wiadomości o wykresach	579
Lokalizacja wykresu	579
Rejestrator makr a wykresy	580
Model obiektu Chart	581
Tworzenie wykresów osadzonych na arkuszu danych	582
Tworzenie wykresu na arkuszu wykresu	584
Modyfikowanie wykresów	584
Wykorzystanie VBA do uaktywnienia wykresu	586
Przenoszenie wykresu	587
Wykorzystanie VBA do dezaktywacji wykresu	587
Sprawdzanie, czy wykres został uaktywniony	588
Usuwanie elementów z kolekcji ChartObjects lub Charts	589
Przetwarzanie wszystkich wykresów w pętli	590
Zmiana rozmiarów i wyrównywanie obiektów ChartObject	593
Tworzenie dużej liczby wykresów	594
Eksportowanie wykresów	596
Eksportowanie wszystkich obiektów graficznych	597
Zmiana danych prezentowanych na wykresie	599
Modyfikacja danych wykresu na podstawie aktywnej komórki	599
Zastosowanie języka VBA do identyfikacji zakresu danych prezentowanych na wykresie	600

Wykorzystanie VBA do wyświetlania dowolnych etykiet danych na wykresie	605
Wyświetlanie wykresu w oknie formularza UserForm	608
Zdarzenia związane z wykresami	611
Przykład wykorzystania zdarzeń związanych z wykresami	611
Obsługa zdarzeń dla wykresów osadzonych	614
Przykład zastosowania zdarzeń dla wykresów osadzonych	616
Jak ułatwić sobie pracę z wykresami przy użyciu VBA?	618
Drukowanie wykresów osadzonych na arkuszu	618
Ukrywanie serii danych poprzez ukrywanie kolumn	618
Tworzenie wykresów, które nie są połączone z danymi	620
Wykorzystanie zdarzenia MouseOver do wyświetlania tekstu	621
Wykresy animowane	624
Przewijanie wykresów	625
Tworzenie wykresu krzywych hipocykoidalnych	627
Tworzenie wykresu-zegara	628
Tworzenie wykresu interaktywnego bez użycia VBA	629
Przygotowanie danych do utworzenia wykresu interaktywnego	630
Tworzenie przycisków opcji dla interaktywnego wykresu	631
Tworzenie listy miast dla wykresu interaktywnego	631
Tworzenie zakresów danych dla wykresu interaktywnego	632
Utworzenie wykresu interaktywnego	633
Tworzenie wykresów przebiegu w czasie	633
Rozdział 17. Obsługa zdarzeń	637
Co powinieneś wiedzieć o zdarzeniach	637
Sekwencje zdarzeń	638
Gdzie należy umieścić procedury obsługi zdarzeń?	638
Wylęczanie obsługi zdarzeń	640
Wprowadzanie kodu procedury obsługi zdarzeń	641
Procedury obsługi zdarzeń z argumentami	642
Zdarzenia poziomu skoroszytu	644
Zdarzenie Open	645
Zdarzenie Activate	646
Zdarzenie SheetActivate	646
Zdarzenie NewSheet	647
Zdarzenie BeforeSave	647
Zdarzenie Deactivate	647
Zdarzenie BeforePrint	648
Zdarzenie BeforeClose	649
Zdarzenia poziomu arkusza	651
Zdarzenie Change	652
Monitorowanie zmian w wybranym zakresie komórek	653
Zdarzenie SelectionChange	657
Zdarzenie BeforeDoubleClick	658
Zdarzenie BeforeRightClick	659
Zdarzenia dotyczące wykresów	660
Zdarzenia dotyczące aplikacji	660
Włączenie obsługi zdarzeń poziomu aplikacji	662
Sprawdzanie, czy skoroszyt jest otwarty	663
Monitorowanie zdarzeń poziomu aplikacji	664
Zdarzenia dotyczące formularzy UserForm	665
Zdarzenia niezwiązane z obiekktami	666
Zdarzenie OnTime	667
Zdarzenie OnKey	668

Rozdział 18. Interakcje z innymi aplikacjami	673
Uruchamianie innych aplikacji z poziomu Excela	673
Zastosowanie funkcji Shell języka VBA	673
Wyświetlanie okna folderu	676
Zastosowanie funkcji ShellExecute interfejsu Windows API	676
Uaktywnianie aplikacji z poziomu Excela	677
Wykorzystanie instrukcji AppActivate	677
Uaktywnianie aplikacji pakietu Microsoft Office	678
Uruchamianie okien dialogowych Panelu sterowania	678
Wykorzystanie automatyzacji w programie Excel	680
Działania z obiekta innymi aplikacji z wykorzystaniem automatyzacji	680
Wczesne i późne wiązanie	681
Prosty przykład późnego wiązania	684
Sterowanie Wordem z poziomu Excela	685
Zarządzanie Excellem z poziomu innej aplikacji	688
Wysyłanie spersonalizowanych wiadomości e-mail z wykorzystaniem Outlooka	690
Wysyłanie wiadomości e-mail z załącznikami z poziomu Excela	693
Rozdział 19. Tworzenie i wykorzystanie dodatków	697
Czym są dodatki?	697
Porównanie dodatku ze standardowym skoroszytem	697
Po co tworzy się dodatki?	698
Menedżer dodatków Excela	700
Tworzenie dodatków	702
Przykład tworzenia dodatku	703
Tworzenie opisu dla dodatku	704
Tworzenie dodatku	704
Instalowanie dodatku	705
Testowanie dodatków	707
Dystrybucja dodatków	707
Modyfikowanie dodatku	707
Porównanie plików XLAM i XLSM	709
Pliki XLAM — przynależność do kolekcji z poziomu VBA	709
Widoczność plików XLSM i XLAM	709
Arkusze i wykresy w plikach XLSM i XLAM	710
Dostęp do procedur VBA w dodatku	711
Przetwarzanie dodatków za pomocą kodu VBA	714
Dodawanie nowych elementów do kolekcji AddIns	714
Usuwanie elementów z kolekcji AddIns	715
Właściwości obiektu AddIn	716
Korzystanie z dodatku jak ze skoroszytu	719
Zdarzenia związane z obiektami AddIn	719
Optymalizacja wydajności dodatków	720
Problemy z dodatkami	721
Zapewnienie, że dodatek został zainstalowany	721
Odwoływanie się do innych plików z poziomu dodatku	723
Wykrywanie właściwej wersji Excela dla dodatku	723

Część V Tworzenie aplikacji	725
Rozdział 20. Praca ze Wstążką	727
Wprowadzenie do pracy ze Wstążką	727
VBA i Wstążka	731
Dostęp do poleceń Wstążki	731
Praca ze Wstążką	733
Aktywowanie karty	735
Dostosowywanie Wstążki do własnych potrzeb	736
Prosty przykład kodu RibbonX	737
Prosty przykład kodu RibbonX — podejście 2.	740
Kolejny przykład kodu RibbonX	745
Demo formantów Wstążki	747
Przykład użycia formantu DynamicMenu	753
Więcej wskazówek dotyczących modyfikacji Wstążki	756
Tworzenie pasków narzędzi w starym stylu	757
Ograniczenia funkcjonalności tradycyjnych pasków narzędzi w Excelu 2007 i nowszych wersjach	757
Kod tworzący pasek narzędzi	758
Rozdział 21. Praca z menu podręcznym	761
Obiekt CommandBar	761
Rodzaje obiektów CommandBar	762
Wyświetlanie menu podręcznych	762
Odwołania do elementów kolekcji CommandBars	763
Odwołania do formantów obiektu CommandBar	764
Właściwości formantów obiektu CommandBar	765
Wyświetlanie wszystkich elementów menu podręcznego	766
Wykorzystanie VBA do dostosowywania menu podręcznego	767
Co nowego w Excelu 2013	767
Resetowanie menu podręcznego	770
Wyłączanie menu podręcznego	771
Wyłączanie wybranych elementów menu podręcznego	772
Dodawanie nowego elementu do menu podręcznego Cell	772
Dodawanie nowego podmenu do menu podręcznego	774
Ograniczanie zasięgu modyfikacji menu podręcznego do jednego skoroszytu	777
Menu podręczne i zdarzenia	777
Automatyczne tworzenie i usuwanie menu podręcznego	777
Wyłączanie lub ukrywanie elementów menu podręcznego	778
Tworzenie kontekstowych menu podręcznych	778
Rozdział 22. Tworzenie systemów pomocy w aplikacjach	781
Systemy pomocy w aplikacjach Excela	781
Systemy pomocy wykorzystujące komponenty Excela	784
Wykorzystanie komentarzy do tworzenia systemów pomocy	784
Wykorzystanie pól tekstowych do wyświetlania pomocy	785
Wykorzystanie arkusza do wyświetlania tekstu pomocy	786
Wyświetlanie pomocy w oknie formularza UserForm	788
Wyświetlanie pomocy w oknie przeglądarki sieciowej	791
Zastosowanie plików w formacie HTML	791
Zastosowanie plików w formacie MHTML	792
Wykorzystanie systemu HTML Help	793
Wykorzystanie metody Help do wyświetlania pomocy w formacie HTML Help	795
Łączenie pliku pomocy z aplikacją	796
Przypisanie tematów pomocy do funkcji VBA	797

Rozdział 23. Tworzenie aplikacji przyjaznych dla użytkownika	799
Czym jest aplikacja przyjazna dla użytkownika?	799
Kreator amortyzacji pożyczek	799
Obsługa Kreatora amortyzacji pożyczek	800
Struktura skoroszytu Kreatora amortyzacji pożyczek	802
Jak działa Kreator amortyzacji pożyczek?	802
Potencjalne usprawnienia Kreatora amortyzacji pożyczek	809
Wskazówki dotyczące projektowania aplikacji	809
Część VI Inne zagadnienia	811
Rozdział 24. Problem kompatybilności aplikacji	813
Co to jest kompatybilność?	813
Rodzaje problemów ze zgodnością	814
Unikaj używania nowych funkcji i mechanizmów	815
Czy aplikacja będzie działać na komputerach Macintosh?	817
Praca z 64-bitową wersją Excela	818
Tworzenie aplikacji dla wielu wersji narodowych	819
Aplikacje obsługujące wiele języków	821
Obsługa języka w kodzie VBA	822
Wykorzystanie właściwości lokalnych	822
Identyfikacja ustawień systemu	823
Ustawienia daty i godziny	825
Rozdział 25. Operacje na plikach wykonywane za pomocą kodu VBA	827
Najczęściej wykonywane operacje na plikach	827
Zastosowanie poleceń języka VBA do wykonywania operacji na plikach	828
Zastosowanie obiektu FileSystemObject	833
Wyświetlanie rozszerzonych informacji o plikach	836
Operacje z plikami tekstowymi	838
Otwieranie plików tekstowych	838
Odczytywanie plików tekstowych	839
Zapisywanie danych do plików tekstowych	839
Przydzielanie numeru pliku	840
Określanie lub ustawianie pozycji w pliku	840
Instrukcje pozwalające na odczytywanie i zapisywanie plików	841
Przykłady wykonywania operacji na plikach	841
Importowanie danych z pliku tekstopowego	841
Eksportowanie zakresu do pliku tekstopowego	843
Importowanie pliku tekstopowego do zakresu	844
Rejestrowanie wykorzystania Excela	845
Filtrowanie zawartości pliku tekstopowego	846
Eksportowanie zakresu komórek do pliku HTML	846
Eksportowanie zakresu komórek do pliku XML	849
Pakowanie i rozpakowywanie plików	851
Pakowanie plików do formatu ZIP	852
Rozpakowywanie plików ZIP	854
Działania z obiektami danych ActiveX (ADO)	855
Rozdział 26. Operacje na składnikach języka VBA	857
Podstawowe informacje o środowisku IDE	857
Model obiektowy środowiska IDE	860
Kolekcja VBProjects	860

Wyświetlanie wszystkich składników projektu VBA	862
Wyświetlanie wszystkich procedur VBA w arkuszu	863
Zastępowanie modułu uaktualnioną wersją	864
Zastosowanie języka VBA do generowania kodu VBA	867
Zastosowanie VBA do umieszczenia formantów na formularzu UserForm	868
Operacje z formularzami UserForm w fazie projektowania i wykonania	869
Dodawanie 100 przycisków CommandButton w fazie projektowania	870
Programowe tworzenie formularzy UserForm	872
Prosty przykład formularza UserForm	872
Użyteczny (ale już nie tak prosty) przykład dynamicznego formularza UserForm	874
Rozdział 27. Moduły klas	879
Czym jest moduł klasy?	879
Przykład: utworzenie klasy NumLock	880
Wstawianie modułu klasy	881
Dodawanie kodu VBA do modułu klasy	881
Wykorzystanie klasy NumLock	883
Dodatkowe informacje na temat modułów klas	884
Programowanie właściwości obiektów	884
Programowanie metod obiektów	886
Zdarzenia definiowane w module klasy	887
Przykład: klasa CSVFileClass	887
Zmienne poziomu modułu dla klasy CSVFileClass	888
Definicje właściwości klasy CSVFileClass	888
Definicje metod klasy CSVFileClass	888
Wykorzystanie obiektów CSVFileClass	890
Rozdział 28. Praca z kolorami	893
Definiowanie kolorów	893
Model kolorów RGB	894
Model kolorów HSL	895
Konwersja kolorów	895
Skala szarości	897
Zamiana kolorów na skalę szarości	899
Eksperymenty z kolorami	900
Praca z motywami dokumentów	901
Kilka słów o motywach dokumentów	901
Kolory motywów dokumentów	902
Wyświetlanie wszystkich kolorów motywu	905
Praca z obiektami Shape	908
Kolor tła kształtu	909
Kształty i kolory motywów	911
Modyfikacja kolorów wykresów	912
Rozdział 29. Często zadawane pytania na temat programowania w Excelu	917
FAQ — czyli często zadawane pytania	917
Ogólne pytania dotyczące programu Excel	918
Pytania dotyczące edytora Visual Basic	923
Pytania dotyczące procedur	926
Pytania dotyczące funkcji	931
Pytania dotyczące obiektów, właściwości, metod i zdarzeń	934
Pytania dotyczące zagadnień związanych z bezpieczeństwem	942
Pytania dotyczące formularzy UserForm	943
Pytania dotyczące dodatków	948
Pytania dotyczące interfejsu użytkownika	950

Dodatek	Dodateki	953
Dodatek A	Instrukcje i funkcje VBA	955
	Wywoływanie funkcji Excela w instrukcjach VBA	958
Dodatek B	Kody błędów VBA	965
Dodatek C	Strona internetowa książki	969
	Skorowidz	985

O autorze

John Walkenbach jest autorem ponad 50 książek związanych tematycznie z arkuszami kalkulacyjnymi. Mieszká w południowej Arizonie. Więcej informacji znajdziesz na jego stronie internetowej <http://spreadsheetpage.com>.

Podziękowania od wydawcy oryginału

Jesteśmy bardzo dumni z tej książki. Jeżeli masz jakieś komentarze bądź uwagi, którymi chcesz się z nami podzielić, możesz je do nas przesyłać za pomocą strony internetowej <http://dummies.custhelp.com/>.

Pojawienie się tej książki na rynku było możliwe dzięki wytrwałej pracy i pomocy wielu ludzi, a w szczególności:

Acquisitions and Editorial

Project Editor: Susan Pink
Acquisitions Editor: Katie Mohr
Technical Editor: Niek Otten
Editorial Manager: Jodi Jensen
Editorial Assistant: Annie Sullivan
Sr. Editorial Assistant: Cherie Case

Composition Services

Project Coordinator: Kristie Rees
Layout and Graphics: Jennifer Henry, Andrea Hornberger, Jennifer Mayberry
Proofreader: Christine Sabooni
Indexer: BIM Indexing & Proofreading Services

Publishing and Editorial for Technology Dummies

Richard Swadley, Vice President and Executive Group Publisher
Andy Cummings, Vice President and Publisher
Mary Bednarek, Executive Acquisitions Director
Mary C. Corder, Editorial Director

Publishing for Consumer Dummies

Diane Graves Steele, Vice President and Publisher

Composition Services

Debbie Stailey, Director of Composition Services

Przedmowa

Witaj w książce *Excel 2013 PL. Programowanie w VBA. Vademecum Walkenbacha*. Jeżeli zajmujesz się tworzeniem aplikacji arkusza kalkulacyjnego, z których korzystają inni, lub po prostu zależy Ci na jak najszerzym wykorzystaniu możliwości Excela, to masz w ręku właściwą książkę.

Zakres zagadnień

Ta książka koncentruje się na zagadnieniach związanych z językiem Visual Basic for Applications (VBA) — językiem programowania wbudowanym w Excela (oraz inne aplikacje pakietu Microsoft Office). Dowiesz się tutaj, jak pisać programy, które automatyzują wykonywanie wielu zadań w Excelu. Znajdziesz tutaj wszystko, począwszy od rejestrowania prostych makr, aż do tworzenia wyrafinowanych, przyjaznych dla użytkownika narzędzi i aplikacji.

W tej książce *nie będziemy* zajmować się pakietem Microsoft Visual Studio Tools for Office (VSTO). Pakiet VSTO to względnie nowa technologia, wykorzystująca Visual Basic .NET oraz Microsoft Visual C#. VSTO może być również używane do sterowania Exceliem i innymi aplikacjami pakietu Microsoft Office.

Jak zapewne wiesz, Excel 2013 jest dostępny również na innych platformach. Na przykład korzystając z przeglądarki sieciowej, możesz używać wersji *Excel Web App*, a nawet możesz uruchomić go na urządzeniach mobilnych, działających pod kontrolą systemu Windows RT. Niestety takie wersje Excela nie obsługują VBA, stąd książka, którą trzymasz w ręku, jest dedykowana dla stacjonarnej wersji Excela 2013, działającej w systemie Windows.

Co musisz wiedzieć?

To nie jest książka dla początkujących użytkowników Excela. Jeżeli nie posiadasz doświadczenia w pracy z tą aplikacją, lepszą propozycją będzie książka *Excel 2013 PL. Biblia*, która jest przeznaczona dla wszystkich użytkowników Excela i szczegółowo opisuje jego funkcje i możliwości.

Aby w pełni skorzystać z tej książki, powinieneś być względnie doświadczonym użytkownikiem Excela. Zakładam, że posiadasz odpowiednią wiedzę na temat następujących zagadnień:

- Jak tworzyć skoroszyty, wstawiać arkusze, zapisywać pliki itp.
- Jak poruszać się w obszarze skoroszytu
- Jak posługiwać się Wstążką programu Excel
- Jak wprowadzać formuły
- Jak używać funkcji arkuszowych
- Jak nadawać nazwy komórkom i zakresom
- Jak korzystać z podstawowych funkcji systemu Windows, takich jak zarządzanie plikami i posługiwanie się schowkiem

Jeżeli nie wiesz, jak wykonać powyższe zadania, niektóre zagadnienia omawiane w książce mogą nieco wykraczać poza Twoje umiejętności, dlatego powinieneś wziąć pod uwagę moje ostrzeżenie. Jeżeli jesteś doświadczonym użytkownikiem arkuszy kalkulacyjnych, ale nie korzystałeś z Excela 2013, w rozdziale 1. znajdziesz krótki przegląd możliwości tego produktu.

Czym musisz dysponować?

Aby w pełni skorzystać z tej książki, powinieneś posiadać oryginalną kopię Excela 2013. Co prawda większość materiału opisywanego w naszej książce będzie poprawnie pracować z Exceliem 2003 i wersjami późniejszymi, ale książka została napisana z myślą o wersji 2013 tego programu. Pomimo iż Excel 2007 i jego następcy są zupełnie innymi programami niż ich poprzednicy, to jednak środowisko VBA nie zmieniło się w nich zupełnie. Jeżeli jednak chcesz tworzyć aplikacje, które mają działać z poprzednimi wersjami Excela, zdecydowanie *nie powinieneś* używać Excela 2013 do prac związkanych z ich projektowaniem. Zamiast tego, powinieneś po prostu użyć jednej z poprzednich wersji Excela (najlepiej takiej, z której korzystają przyszli użytkownicy Twoich aplikacji).

Niniejsza książka nie jest przeznaczona dla użytkowników Excela na platformie Macintosh. Aby korzystać z Excela, wystarczy dowolny komputer pracujący w systemie Windows. Oczywiście lepszy będzie szybki system dysponujący sporą ilością pamięci. Excel to duży program, dlatego uruchamianie go na komputerze powolnym lub posiadającym minimalną ilość pamięci może być wyjątkowo irytujące.

Zalecam użycie dobrej karty graficznej i monitora oferującego wysoką rozdzielczość, ponieważ bardzo często będziesz pracował z co najmniej dwoma oknami. Optymalnym rozwiązaniem będzie użycie komputera wyposażonego w dwumonitorową kartę graficzną (i oczywiście dwa monitory...), dzięki czemu będziesz mógł umieścić okno Excela na jednym monitorze, a okno edytora VBE na drugim. Szybko się przyzwyczaisz do takiego rozwiązania.

Konwencje zastosowane w książce

Powinieneś poświęcić chwilę czasu na zapoznanie się z konwencjami typograficznymi zastosowanymi w książce.

Polecenia Excela

Począwszy od Excela 2007 zaimplementowany został zupełnie nowy interfejs użytkownika, pozbawiony znanego z poprzednich wersji menu. Zamiast klasycznego menu Excel wykorzystuje teraz kontekstowy interfejs użytkownika o nazwie **Wstążka** (ang. *Ribbon*). Na górze Wstążki znajduje się szereg *kart* (takich jak *Narzędzia główne*, *Wstawianie*, *Widok* itd.). Kiedy klikniesz wybraną kartę, na Wstążce zostaną wyświetcone przyciski poleczeń, list i opcji należących do wybranej kategorii polecień. Każde polecenie ma swoją unikatową nazwę, która jest zwykle wyświetlana pod lub obok ikony polecenia. Poszczególne ikony polecień są poukładane w grupy tematyczne i każda grupa również posiada swoją nazwę, która jest wyświetlana poniżej ikon polecień.

Konwencja zapisu polecień użyta w tej książce wskazuje na nazwę karty, po której następuje nazwa grupy i wreszcie nazwa polecenia. Na przykład odwołanie do polecenia, które zawija tekst wyświetlany w komórce, może zostać zapisane następująco:

Narzędzia główne/Wyrównywanie/Zawijaj tekst

Kliknięcie pierwszej karty, o nazwie *Plik*, włącza nowy widok o nazwie *Backstage*. Polecenia menu *Backstage* znajdują się w panelu z lewej strony okna. Aby wskazać polecenie z widoku *Backstage*, będziemy używać nazwy karty *Plik*, po której będzie następowała nazwa polecenia. Na przykład wykonanie polecenia przedstawionego poniżej spowoduje wyświetlenie okna opcji programu Excel.

Plik/Opcje

Polecenia edytora VBA

Edytor VBA to okno dialogowe, w którym pracujesz z kodem języka VBA. Edytor VBA używa tradycyjnego interfejsu wyposażonego w menu i paski narzędzi. Zapis polecenia przedstawiony poniżej oznacza, że powinieneś z menu *Tools* wybrać polecenie *References*:

Tools/References

Konwencje związane z klawiaturą

Klawiatury używasz do wprowadzania danych. Możesz z niej korzystać także do obsługi menu i okien dialogowych (jest to dość wygodne w sytuacji, gdy palce znajdują się już na klawiaturze).

Wprowadzanie danych

Dane wprowadzane z klawiatury zawarte w tekście są wyróżniane pogrubieniem:

Wprowadź do komórki B51 formułę =SUMA(B2:B50).

Dłuższe łańcuchy zazwyczaj znajdujące się w oddzielnym wierszu są wyróżnione tak, jak pokazano poniżej. Dla przykładu mogę poprosić Cię o wprowadzenie następującej formuły:

=WYSZUKAJ.PIONOWO(STOCKNUMBER; PRICELIST; 2)

Kod źródłowy języka VBA

W książce zawarto wiele fragmentów kodu źródłowego języka VBA, a także pełne listingi kodu procedur. Każdy wiersz kodu znajduje się w oddzielnym wierszu tekstu (kody procedur kopiowałem bezpośrednio z modułu VBA i wklejałem w oknie edytora tekstu). Aby kod był czytelniejszy, często w celu wykonania wcięć wstawiłem jedną lub kilka spacji. Wcięcia są opcjonalne, ale pomagają pokazać powiązane z sobą instrukcje.

Jeżeli długie wiersze kodu nie mieszczą się w jednym wierszu w książce, podzielone są za pomocą standardowego w VBA systemu oznaczania kontynuacji wiersza kodu. Znak podkreślenia poprzedzony przez spację na końcu wiersza oznacza, że wiersz kodu kontynuowany jest w następnym wierszu tekstu. Oto przykład:

```
columnCount = Application.WorksheetFunction.  
    CountA(Range("A:A")) + 1
```

Ten kod może też zostać wprowadzony w jednym wierszu bez użycia znaku podkreślenia.

Funkcje, nazwy plików i zdefiniowane nazwy zakresów

Nazwy funkcji arkusza kalkulacyjnego są zapisywane przy użyciu dużych liter, na przykład: „Wprowadź w komórce C20 formułę SUMA”. Nazwy procedur, właściwości, metod i obiektów języka VBA zawarte w tekście zostały zapisane przy użyciu czcionki o stałym odstępie międzyznakowym, na przykład: „Wykonaj procedurę GetTotals”. Aby zwiększyć czytelność tego typu nazw, często posługujemy się w nazwach wielkimi i małymi literami.

Konwencje związane z myszą

Jeżeli zdecydowałeś się na przeczytanie tej książki, to zakładam, że posługiwanie się myszą nie stanowi dla Ciebie problemu. Używam standardowej terminologii — wskazywanie, klikanie, kliknięcie prawym przyciskiem, przeciąganie itp.

Znaczenie ikon

Aby zwrócić uwagę Czytelników na szczególnie istotne kwestie, używam szeregu ikon:



W ten sposób wyróżniamy nowości, które pojawiły się w Excelu 2013.



Tej ikony używamy do wskazania szczególnie istotnych informacji — może to być opis techniki ułatwiającej wykonanie zadania lub coś, co ma fundamentalne znaczenie dla zrozumienia dalszego materiału.



Ikona wskazuje wydajniejszy sposób wykonania jakiegoś zadania lub technikę, która na pierwszy rzut oka może nie być oczywista.



Ikona identyfikuje plik przykładowy znajdujący się na stronie internetowej tej książki (zapoznaj się z punktem „O stronie internetowej książki” znajdującym się w dalszej części „Przedmowy”).



Tej ikony używamy, gdy omawiana operacja przy braku ostrożności może spowodować problemy.



W ten sposób odwołujmy się do innych rozdziałów, w których znajdziesz więcej informacji na określony temat.

Struktura książki

Książkę podzielono na siedem głównych części.

Część I. Podstawowe informacje

W tej części przedstawiam podstawowe informacje stanowiące bazę dla pozostałej części książki. W rozdziale 1. znajdziesz przegląd możliwości Excela 2013. W rozdziale 2. omówilem podstawy używania formuł, włącznie z niektórymi wybranymi technikami, które mogą być dla Ciebie zupełną nowością. W rozdziale 3. będziemy zajmować się plikami używanymi i generowanymi przez Excela, a w rozdziale 4. omówimy szereg zagadnień związanych z tworzeniem aplikacji na bazie arkusza kalkulacyjnego Excela.

Część II. Język Visual Basic for Applications

Na część II składają się rozdziały od 5. do 9., gdzie omawiamy wszystko, co powinieneś wiedzieć, aby efektywnie rozpocząć pracę z językiem VBA. Znajdziesz tutaj wprowadzenie do języka VBA, omówienie podstawowych zagadnień dotyczących programowania i szczegółowy opis tworzenia procedur i funkcji w języku VBA. Na koniec w rozdziale 9. znajdziesz wiele przykładów użytecznych procedur VBA, których będziesz mógł użyć w swoich programach.

Część III. Praca z formularzami UserForm

W czterech rozdziałach omówilem niestandardowe okna dialogowe, nazywane też formularzami *UserForm*. W rozdziale 10. zaprezentowałem kilka wbudowanych narzędzi, których możesz używać zamiast tworzenia formularzy *UserForm*. W rozdziale 11. znajdziesz wprowadzenie do formularzy *UserForm* i opis formantów, z których możesz korzystać. W rozdziałach 12. i 13. znajdziesz wiele przykładów niestandardowych okien dialogowych, zarówno tych prostych, jak i tych bardziej zaawansowanych.

Część IV. Zaawansowane techniki programowania

W części IV opisujemy dodatkowe techniki programowania, które są często określane mianem zaawansowanych. W pierwszych trzech rozdziałach tej części wyjaśniamy, w jaki sposób tworzyć narzędzia i jak używać języka VBA w przypadku korzystania z tabel przestawnych i wykresów. W rozdziale 17. omówimy obsługę zdarzeń, która umożliwia automatyczne wykonywanie procedur po wystąpieniu określonych zdarzeń. W rozdziale 18. pokazujemy metody, których można użyć przy interakcji z innymi aplikacjami, takimi jak Word. Rozdział 19. kończy część IV dogłębnym omówieniem procesu tworzenia dodatków (ang. *add-ins*) do Excela.

Część V. Tworzenie aplikacji

Rozdziały wchodzące w skład tej części poświęcone są istotnym elementom procesu tworzenia aplikacji przyjaznych dla użytkownika. W rozdziale 20. omawiamy sposoby modyfikacji Wstążki. W rozdziale 21. zostały przedstawione różne sposoby modyfikacji menu podręcznych. W rozdziale 22. zaprezentowano kilka sposobów tworzenia systemów pomocy dla tworzonych aplikacji, a w rozdziale 23. omawiamy podstawowe zagadnienia związane z tworzeniem aplikacji przyjaznych dla użytkownika i zamieszczamy praktyczny przykład tworzenia takiej aplikacji.

Część VI. Inne zagadnienia

W sześciu rozdziałach VI części książki omawiamy różne zagadnienia dodatkowe. W rozdziale 24. prezentujemy informacje na temat kompatybilności różnych wersji Excela. W rozdziale 25. omawiamy różne metody użycia języka VBA do pracy z plikami. W rozdziale 26. wyjaśniamy, w jaki sposób używać języka VBA do manipulowania jego komponentami, takimi jak formularze *UserForm* i moduły. W rozdziale 27. omawiamy moduły klas. Rozdział 28. porusza zagadnienia związane z wykorzystaniem kolorów w Excelu. W ostatnim rozdziale tej części odpowiadamy na często zadawane pytania dotyczące programowania w Excelu.

Dodatek

Książka zawiera trzy dodatki. Dodatek A spełnia funkcję przewodnika po wszystkich słowach kluczowych języka VBA (instrukcje i funkcje). W dodatku B objaśniamy kody błędów języka VBA, natomiast w dodatku C — pliki dostępne na stronie internetowej książki.

O stronie internetowej książki

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz komplet skoroszytów z przykładami omawianym w książce, które powinieneś pobrać i zainstalować na dysku lokalnym swojego komputera. Gdy piszę książkę informacyjną, kładę nacisk na zdobywanie wiedzy w oparciu o przykłady. Wiem, że więcej się nauczę po zapoznaniu się z dobrze przemyślanym przykładem niż po przeczytaniu dziesięciu stron książki. Zakładam, że tak samo jest w przypadku innych osób. W konsekwencji spędziłem znacznie więcej czasu, tworząc przykłady, niż pisząc kolejne rozdziały książki.

Pliki znajdują się na stronach wydawnictwa Helion, pod adresem www.helion.pl/ksiazki/e23pvw.htm.



Opisy poszczególnych plików z przykładami do każdego rozdziału znajdziesz w dodatku C.

Narzędzie Power Utility Pak

Narzędzie *PUP* (ang. *Power Utility Pak*) jest nagrodzonym zbiorem programów i nowych funkcji arkusza, wspomagającym korzystanie z Excela. Pakiet został stworzony wyłącznie przy użyciu języka VBA.

Mam nadzieję, że to narzędzie przyda Ci się podczas codziennej pracy z Excelem. Pełny kod źródłowy napisany w języku VBA można pobrać z mojej strony internetowej (po uiszczeniu niewielkiej opłaty). Analizowanie kodu źródłowego to znakomity sposób poznawania przydatnych metod programowania.

Narzędzie *PUP* można sprawdzić, instalując jego 30-dniową wersję zamieszczoną na mojej stronie internetowej <http://spreadsheetpage.com/>.

Jak korzystać z książki?

Z książki można korzystać w dowolny sposób. Sugeruję przeczytanie jej od początku do końca. Ale ponieważ zawarłem w niej zagadnienia zarówno o średnim, jak i zaawansowanym stopniu trudności, kolejność rozdziałów często nie jest istotna. Podejrzewam, że większość Czytelników pominie niektóre rozdziały książki, wyszukując fragmenty, które ich interesują. Jeżeli zmagasz się z ambitnym zadaniem, to w celu sprawdzenia, czy książka umożliwi Ci rozwiązanie problemu, możesz najpierw skorzystać z indeksu.

Część I

Podstawowe informacje

W tej części:

Rozdział 1. „Program Excel w zarysie”

Rozdział 2. „Wybrane zasady stosowania formuł”

Rozdział 3. „Pliki programu Excel”

Rozdział 4. „Podstawy projektowania aplikacji arkusza kalkulacyjnego”

Rozdział 1.

Program Excel w zarysie

W tym rozdziale:

- Excel jako program zorientowany obiektowo
- Najważniejsze funkcje i mechanizmy programu Excel
- Co nowego w Excelu 2013?
- Przydatne wskazówki i techniki pracy

O Excelu

Excel jest jak dotąd najpopularniejszym i najczęściej używanym arkuszem kalkulacyjnym na świecie. Ponieważ czytasz te książkę, wszelkie znaki na niebie i ziemi wskazują, że również jesteś użytkownikiem tego programu i że używasz go już od co najmniej kilku lat. Jednak nawet doświadczenemu weteranowi od czasu do czasu potrzebne jest odświeżenie posiadanych informacji, zwłaszcza w sytuacji, kiedy większość Twojego doświadczenia obejmuje Excela w wersji 2003 lub nawet wcześniejszej.

W tym rozdziale zamieścimy krótkie omówienie Excela i wprowadzimy pojęcie obiektów, które są fundamentalnym zagadnieniem o kluczowym znaczeniu dla każdego, kto poważnie myśli o programowaniu w języku VBA.

Myślenie w kategoriach obiektów

Podczas tworzenia aplikacji w Excelu (zwłaszcza gdy korzystasz z języka VBA — *Visual Basic for Applications*) bardzo pomaga myślenie o Excelu w kategoriach obiektów lub inaczej mówiąc, składników arkusza kalkulacyjnego, którymi można manipulować ręcznie bądź za pośrednictwem makr. Oto kilka przykładów obiektów Excela:

- Program Excel we własnej „osobie”.
- Skoroszyt programu Excel.
- Arkusz zawarty w skoroszycie.

- Zakres komórek zdefiniowany w arkuszu.
- Formant *ListBox* umieszczany na formularzu *UserForm* (czyli w niestandardowym oknie dialogowym).
- Wykres osadzony na arkuszu danych.
- Seria danych prezentowana na wykresie.
- Wybrany punkt danych serii prezentowanej na wykresie.

Z pewnością zauważyleś, że istnieje tutaj pewna *hierarchia obiektów*. Obiekt Excel zawiera obiekty skoroszytów, które przechowują obiekty arkuszy, a te z kolei przechowują obiekty zakresów. Tego typu hierarchia tworzy *model obiektowy* programu Excel. Excel posiada ponad 200 klas obiektów, które mogą być sterowane bezpośrednio lub za pomocą języka VBA. Inne aplikacje pakietu Microsoft Office również posiadają swoje własne modele obiektowe.



Uwaga

Możliwość sterowania obiektami ma fundamentalne znaczenie przy projektowaniu aplikacji. W trakcie lektury tej książki dowieś się, w jaki sposób automatyzować zadania poprzez sterowanie obiektami Excela za pomocą języka VBA. Zagadnienie to stanie się bardziej zrozumiałe po zapoznaniu się z zawartością kolejnych rozdziałów.

Skoroszyty

Jednym z najczęściej używanych obiektów Excela jest *skoroszyt*. Wszystkie operacje wykonywane w Excelu są powiązane ze skoroszytem, przechowywanym w pliku o domyślnym rozszerzeniu *.xlsm*. Skoroszyt Excela może przechowywać dowolną liczbę arkuszy (ograniczoną jedynie przez dostępną pamięć). Możemy wyróżnić cztery typy arkuszy:

- Arkusze danych.
- Arkusze wykresów.
- Arkusze makr *XLM* z Excela 4.0 (przestarzałe, ale nadal obsługiwane).
- Arkusze dialogowe z Excela 5.0 (przestarzałe, ale nadal obsługiwane).

Pracując z Exceliem, możesz otworzyć dowolną liczbę skoroszytów (każdy w osobnym oknie), ale w danej chwili tylko jeden z nich może być *skoroszytem aktywnym*. Podobnie tylko jeden arkusz skoroszytu może być w danej chwili *arkuszem aktywnym*. Aby uaktywnić arkusz, należy kliknąć jego kartę w dolnej części ekranu. Aby zmienić nazwę arkusza, należy dwukrotnie kliknąć kartę i wpisać nową nazwę. Po kliknięciu karty prawym przyciskiem myszy zostanie wyświetlane menu podręczne, za pomocą którego możesz zmienić kolor karty, ukryć arkusz i wykonać wiele innych zadań.

W razie potrzeby możesz również ukryć całe okno skoroszytu. Aby to zrobić, przejdź na kartę *WIDOK* i naciśnij przycisk *Ukryj* znajdujący się w grupie opcji *Okno*. Ukryte okno skoroszytu pozostanie otwarte, ale będzie niewidoczne. Aby przywrócić okno skoroszytu na ekran, przejdź na kartę *WIDOK* i naciśnij przycisk *Odkryj* znajdujący się w grupie opcji *Okno*.

Pojedynczy skoroszyt może być wyświetlany w wielu oknach (aby tego dokonać, przejdź na kartę *WIDOK* i naciśnij przycisk *Nowe okno* znajdujący się w grupie opcji *Okno*). W poszczególnych oknach można wyświetlać różne arkusze skoroszytu lub różne obszary wybranego arkusza.

Arkusze

Najpopularniejszym typem arkusza jest zwykły arkusz danych, czyli to co zazwyczaj użytkownicy mają na myśli, gdy zaczynamy rozmawiać o arkuszach kalkulacyjnych. Arkusz zawiera komórki, w których są przechowywane dane i formuły.

Każdy arkusz Excela 2013 składa się z 16 384 kolumn i 1 048 576 wierszy. Liczba wierszy i kolumn jest stała — co prawda, aby zwiększyć przejrzystość arkusza, można ukryć zbędne wiersze i kolumny, ale nie można zmienić ich liczb.



Uwaga

Wersje Excela starsze od wersji Excel 2007 do zapisywania skoroszytów na dysku używały plików w binarnym formacie *XLS*, w których arkusze składały się maksymalnie z 65 536 wierszy i 256 kolumn. Jeżeli otworzysz taki plik w Excelu 2013, program automatycznie przełączy się w tzw. *tryb zgodności* pozwalający na używanie plików starego formatu. Aby dokonać konwersji takiego pliku na nowy format, zapisz go na dysku w formacie *XLSX* lub *XLSM*, a następnie zamknij i ponownie otwórz taki plik.

Prawdziwą zaletą używania wielu arkuszy w skoroszycie wbrew pozorom wcale nie jest dostęp do większej liczby komórek, a możliwość ulepszenia organizacji pracy. Jeszcze nie tak dawno, kiedy plik skoroszytu składał się z pojedynczego arkusza, projektanci tracili mnóstwo czasu, próbując tak go zorganizować, aby informacje były przechowywane w wydajny i efektywny sposób. Obecnie dane mogą być przechowywane w dowolnej liczbie arkuszy i w dalszym ciągu arkusza można uzyskać do nich natychmiastowy dostęp poprzez proste kliknięcie karty arkusza.

Jak duży jest arkusz Excela?

Zatrzymaj się teraz na chwilę i pomyśl o rzeczywistych rozmiarach arkusza programu Excel. Kiedy wykonasz proste obliczenia (16 384 kolumny × 1 048 576 wierszy), przekonasz się, że pojedynczy arkusz składa się z 17 179 869 184 komórek. Pamiętaj, że to tylko jeden arkusz — każdy skoroszyt może przechowywać przecież więcej niż jeden arkusz.

Jeżeli używasz rozdzielczości ekranu 1920×1200 punktów i domyślnej wysokości i szerokości kolumn, na ekranie możesz jednorazowo wyświetlić obszar mniej więcej o rozmiarze 29 kolumn na 47 wierszy (lub inaczej mówiąc, 1363 komórek arkusza) — co stanowi około 0,0000079% obszaru całego arkusza. Innymi słowy, w jednym arkuszu mieści się około 12,6 miliona ekranów informacji.

Jeżeli chciałbyś wpisać do każdej komórki arkusza jedną liczbę, to przy pracy w relatywnie szybkim tempie 1 komórki na sekundę wypełnienie całego arkusza zajęłoby, bagatela, ponad 500 lat ciąglej, nieprzerwanej pracy. Aby wydrukować rezultaty takiej tytanicznej pracy, musiałbyś zużyć ponad 36 milionów kartek papieru — stos o wysokości ponad 3,6 km (to więcej niż 19 budynków takich jak Pałac Kultury i Nauki w Warszawie, ustawionych jeden na drugim).

Jak zatem nietrudno się domyślić, wypełnienie wartościами całego arkusza nie jest nawet w przybliżeniu możliwe, nie mówiąc już o tym, że nawet jeżeli używałbyś 64-bitowej wersji Excela, to i tak o wiele wcześniej Twojemu komputerowi po prostu zabrakłoby pamięci i prawdopodobnie cała operacja zakończyłaby się zawieszeniem programu.

Jak wiadomo, w komórce arkusza można przechowywać wartości stałe lub wyniki wygenerowane przez formuły. Wartością może być liczba, data, wartość logiczna (Prawda lub Fałsz) lub tekst. Każdy arkusz posiada też niewidoczną warstwę rysunkową umożliwiającą wstawianie obiektów graficznych, takich jak wykresy, kształty, obiekty typu *SmartArt*, formanty formularza *UserForm*, obrazy i inne obiekty osadzone.

Jako użytkownik Excela masz pełną kontrolę nad szerokością kolumn i wysokością wierszy, a w razie potrzeby możesz nawet ukryć wybrane wiersze i kolumny, a także całe arkusze. Możesz definiować kroje i rozmiary używanych w arkuszu czcionek i masz pełną kontrolę nad kolorami. Tekst w komórce może być wyświetlany pionowo lub pod określonym kątem i może też zostać zawinięty tak, aby zajmował kilka wierszy. Oprócz tego możesz również połączyć grupę wybranych komórek tak, aby tworzyły jedną, nową, dużą komórkę arkusza.



W poprzednich wersjach Excel mógł korzystać z palety kolorów ograniczonej do 56 kolorów. Począwszy od wersji Excel 2007 liczba używanych kolorów jest praktycznie nieograniczona. Oprócz tego, od wersji 2007 Excel obsługuje również tzw. *motywy dokumentów*, dzięki którym za pomocą jednego kliknięcia możesz nadać nowy, efektowny wygląd swoim skoroszytom.

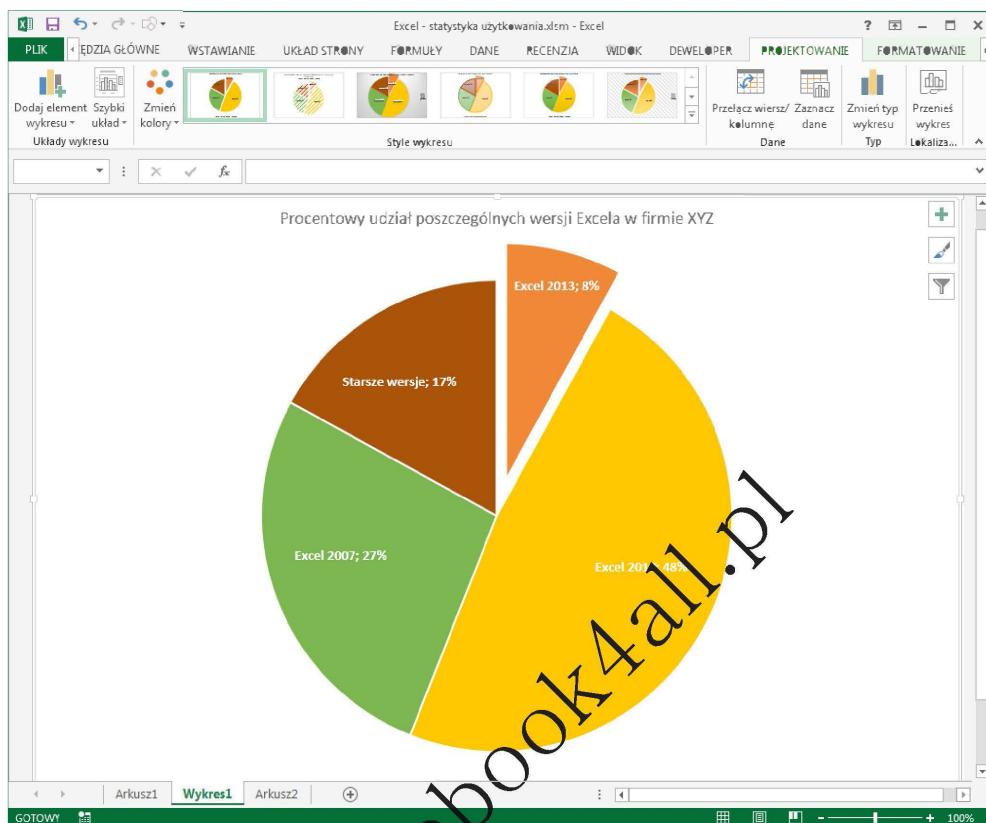
Arkusze wykresów

Standardowo arkusz wykresu przechowuje pojedynczy wykres. Wielu użytkowników ignoruje arkusze wykresów, preferując przechowywanie wykresów w warstwie rysunkowej arkusza. Co prawda użycie arkuszy wykresów jest opcjonalne, ale dzięki nim wydrukowanie na stronie samego wykresu jest trochę łatwiejsze. Tego typu arkusze są szczególnie przydatne w przypadku tworzenia prezentacji. Na rysunku 1.1 przedstawiono wykreskoły zamieszczony na arkuszu wykresu.

Arkusze makr XLM

W zasadzie arkusz makr XLM (znany również pod nazwą *arkusza makra programu MS Excel 4*) jest zwykłym arkuszem, ale posiada kilka innych ustawień domyślnych. Dokładniej mówiąc, arkusz makra XLM zamiast wyników formuł wyświetla same formuły. Oprócz tego domyślana szerokość kolumny w takim arkuszu jest większa niż w przypadku zwykłego arkusza.

Jak sama nazwa wskazuje, arkusz makr XLM został zaprojektowany do przechowywania makr XLM. System makr XLM jest pozostałością z poprzednich wersji Excela (4.0 i starszych). Excel 2013 w dalszym ciągu zachował obsługę tych makr ze względu na konieczność zachowania kompatybilności, jednak nie oferuje już możliwości ich rejestrowania. W tej książce nie będziemy zajmować się omówieniem systemu makr XLM.



Rysunek 1.1. Wykres kołowy utworzony na arkuszu wykresu

Arkusze dialogowe programów Excel 5 i 95

W wersjach Excel 5 i 95 niestandardowe okna dialogowe były tworzone za pomocą specjalnego arkusza dialogowego. Excel 97 i wersje późniejsze nadal obsługują tego typu arkusze dialogowe, ale oferują znacznie lepsze rozwiązanie w postaci formularzy *User-Form* definiowanych w edytorze Visual Basic (VBE).

Po otwarciu skoroszytu zawierającego arkusz dialogowy programów Excel 5 i 95 możesz przejść na taki arkusz, klikając jego kartę. Arkusze dialogowe programów Excel 5 i 95 nie będą omawiane w tej książce.

Interfejs użytkownika programu Excel

Interfejs użytkownika to mechanizm, za pomocą którego użytkownik może komunikować się z programem komputerowym. Interfejs użytkownika składa się z takich elementów, jak menu, paski narzędzi, okna dialogowe, skróty klawiaturowe itp.

Co nowego w Excelu 2013?

Wraz z pojawianiem się na rynku każdej kolejnej nowej wersji pakietu Microsoft Office w Excelu pojawiała się raz większa, raz mniejsza liczba nowych funkcji i mechanizmów. W przypadku pakietu Office 2013 liczba nowości jest całkiem spora, aczkolwiek nie pojawiło się tutaj nic szczególnie rewolucyjnego.

Poniżej zamieszczono listę nowości, jakie możesz znaleźć w Excelu 2013, w porównaniu do Excela 2010:

- **Możliwość przechowania danych w chmurze** — Nowy Excel jestściśle zintegrowany z usługą Microsoft SkyView.
- **Wsparcie dla urządzeń mobilnych** — Nowy Excel ma swoje wersje przeznaczone do działania na wielu różnych urządzeniach mobilnych, takich jak na przykład tablety z ekranami dotykowymi, działające pod kontrolą systemu Windows RT, czy smartfony z systemem Windows.
- **Nowy wygląd** — Projektanci firmy Microsoft nadali Excelowi nowy, „płaski” wygląd i dodali w pasku tytułowym przyciski nowych poleceń. Schematy kolorów interfejsu zostały ograniczone do trzech: białego, jasnoszarego i ciemnoszarego.
- **Jednodokumentowy interfejs użytkownika** — Excel nie obsługuje już możliwości wyświetlania wielu skoroszytów w jednym oknie programu. Każdy skoroszt ma teraz swoje osobne okno aplikacji i osobną Wstążkę.
- **Nowe mechanizmy wspomagające użytkownika** — Excel potrafi teraz zarekomendować użytkownikowi najbardziej jego zdaniem właściwy typ tabeli przestawnej i wykresów.
- **Mechanizm Fill Flash (wypełnianie błyskawiczne)** — Nowy mechanizm pozwalający na szybkie wyodrębnianie potrzebnych danych z łańcuchów tekstu, szybkie wypełnianie kolumn arkusza czy łączenie danych w wielu kolumnach.
- **Wsparcie Apps for Office (aplikacje dla pakietu Office)** — Teraz możesz kupować i pobierać specjalne aplikacje, które mogą być osadzane w plikach skoroszytów Excela.
- **Poprawiony mechanizm Slicer (fragmentator)** — Mechanizm Slicer, wprowadzony w wersji Excel 2010 do pracy z tabelami przestawnymi, w Excelu 2013 został znaczco rozbudowany i może działać ze zwykłymi tabelami.
- **Mechanizm Timeline Slicer (filtrowanie osi czasu)** — Mechanizm podobny do fragmentatora, który pozwala na szybkie filtrowanie danych według dat w tabelach przestawnych.
- **Mechanizm Quick Analysis (szybka analiza danych)** — Mechanizm, który za pomocą jednego kliknięcia myszy daje dostęp do różnych narzędzi analitycznych.
- **Rozbudowane formatowanie wykresów** — Modyfikowanie i dostosowywanie wykresów dla użytkownika jest teraz znacznie łatwiejsze.
- **Zwiększone zastosowanie paneli zadań** — W Excelu 2013 panele zadań odgrywają bardzo ważną rolę; na przykład pozwalały na modyfikację praktycznie wszystkich elementów składowych wykresów.
- **Nowe funkcje arkuszowe** — Excel 2013 ma bardzo dużo nowych funkcji arkuszowych, z których wiele to bardzo specyficzne funkcje specjalnego przeznaczenia.
- **Zmodyfikowany i ulepszony widok Backstage** — Widok Backstage został przeorganizowany i jest teraz znacznie bardziej funkcjonalny.
- **Nowe dodatki** — W pakiecie Office Professional Plus znajdziesz trzy nowe dodatki — PowerPivot, PowerView oraz Inquire.

Pojawienie się na rynku pakietu Office 2007 zasygnalizowało koniec ery tradycyjnego systemu menu i pasków narzędzi. Nowy interfejs użytkownika programu Excel składa się z następujących elementów:

- Wstążka
- Pasek narzędzi *Szybki dostęp*
- Menu podręczne, dostępne po kliknięciu prawym przyciskiem myszy
- Minipaski narzędzi
- Okna dialogowe
- Skróty klawiszowe
- Panele zadań



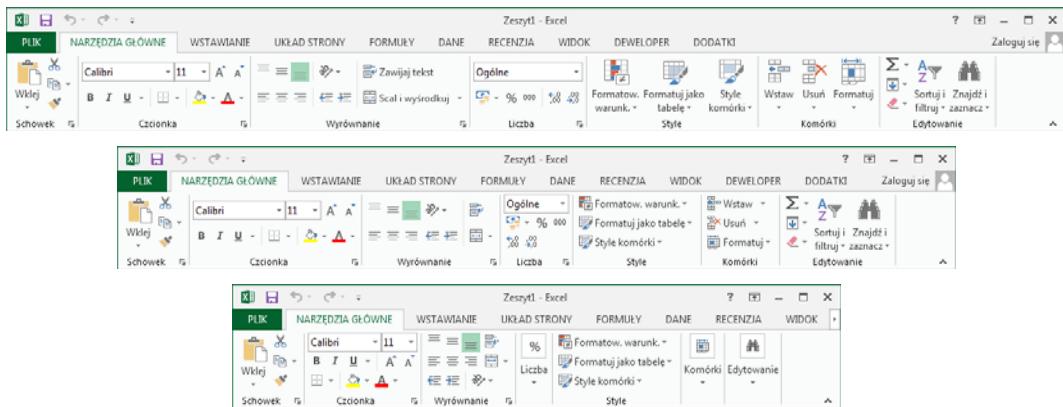
Excel 2013 może również działać na urządzeniach mobilnych wyposażonych w ekranы dotykowe. W naszej książce przyjęliśmy założenie, że korzystasz z „tradycyjnego” komputera, wyposażonego w mysz i klawiaturę, stąd nie będziemy omawiać żadnych poleceń związanych z „dotykowym” interfejsem użytkownika.

Wprowadzenie do Wstążki

W pakiecie Office 2007 firma Microsoft wprowadziła zupełnie nowy graficzny interfejs użytkownika. Menu i paski narzędzi znane z poprzednich wersji zostały całkowicie zastąpione przez interfejs o nazwie **Wstążka** (ang. *Ribbon*) i jego karty. Kiedy klikniesz wybraną kartę Wstążki (taką jak na przykład *NARZĘDZIA GŁÓWNE*, *WSTAWIANIE* czy *UKŁAD STRONY*), na Wstążce pojawią się przyciski poleceń dostępnych na danej karcie. Office 2007 było pierwszym w historii oprogramowaniem firmy Microsoft, które korzystało z takiego interfejsu, a do tej pory już kilka innych firm przyjęło podobne rozwiązania w swoich produktach.

Wygląd i liczba poleceń dostępnych na poszczególnych kartach Wstążki zależy od szerokości głównego okna programu Excel. Jeżeli okno programu jest zbyt wąskie, aby wyświetlić wszystkie dostępne polecenia, zestaw poleceń automatycznie adaptuje się do nowego otoczenia, co może sprawiać wrażenie, że niektóre polecenia znikają ze Wstążki i nie są dostępne. W praktyce wszystkie takie „zagubione” polecenia są nadal dostępne dla użytkownika. Na rysunku 1.2 przedstawiono wygląd karty *NARZĘDZIA GŁÓWNE* dla trzech różnych szerokości okna programu Excel.

Na górnym rysunku wszystkie formanty Wstążki są widoczne. Środkowy rysunek przedstawia wygląd tej samej karty, ale po zmniejszeniu szerokości okna programu Excel. Zwróc uwagę, że podpisy standardowo widoczne pod przyciskami poleceń znikają, ale same przyciski nadal pozostają dostępne. Na dolnym rysunku przedstawiono nieco ekstremalny przypadek, kiedy szerokość okna programu Excel została bardzo radykalnie zmniejszona. Niektóre grupy poleceń są zredukowane do postaci pojedynczych ikon i dopiero kliknięcie takiego przycisku rozwija menu podręczne, w którym są dostępne wszystkie polecenia danej grupy.



Rysunek 1.2. Karta NARZĘDZIA GŁÓWNE interfejsu Wstążka dla trzech różnych szerokości okna programu



Jeżeli chcesz całkowicie ukryć Wstążkę programu Excel, tak aby maksymalnie powiększyć obszar roboczy Excela, po prostu dwukrotnie kliknij lewym przyciskiem myszy dowolną kartę. Wstążka zniknie z ekranu i dzięki temu będziesz mógł wyświetlić około czterech dodatkowych wierszy arkusza. Kiedy ponownie będziesz chciał skorzystać ze Wstążki, po prostu kliknij dowolną kartę i Wstążka powróci na ekran. Wstążkę możesz również włączać bądź wyłączać, naciskając kombinację klawiszy **Ctrl+F1** lub klikając lewym przyciskiem myszy przycisk **Opcje wyświetlanego Wstążki**, znajdujący się na pasku tytułowym programu Excel, po prawej stronie przycisku **Pomoc**.

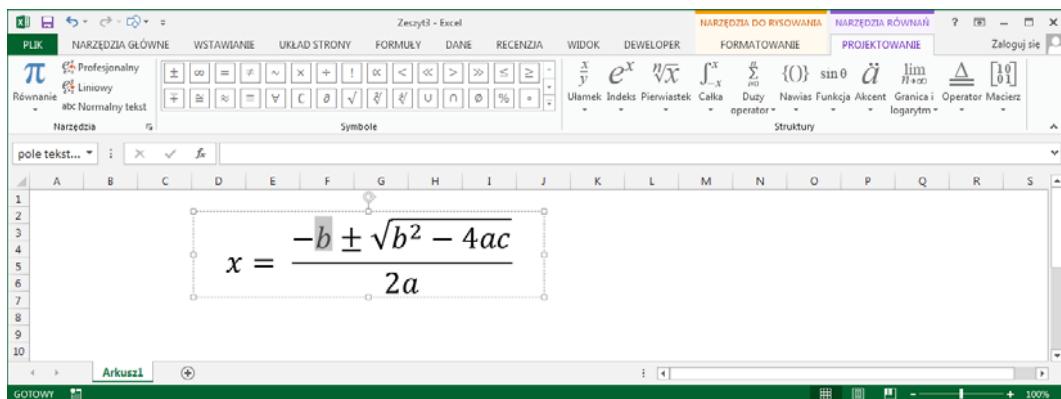
Karty kontekstowe

Oprócz standardowych kart poleceń Wstążka programu Excel posiada również tzw. *karty kontekstowe*, które pojawiają się tylko wtedy, kiedy zaznaczony zostanie obiekt (taki jak wykres, tabela, obraz czy obiekt typu *SmartArt*) posiadający specjalny zestaw poleceń przeznaczonych tylko dla tego obiektu.

Na rysunku 1.3 przedstawiono karty kontekstowe, które pojawiają się na Wstążce, kiedy zostanie zaznaczone równanie osadzone na arkuszu. W takiej sytuacji Excel wyświetla dwie karty kontekstowe: *Formatowanie* (na której znajdują się polecenia przeznaczone do pracy z obiektem) oraz *Projektowanie* (zawiera polecenia do pracy z równaniem). Zwróć uwagę, że karty kontekstowe są wyświetlane w grupach *Narzędzia do rysowania* i *Narzędzia równań* (nazwy grup kontekstowych pojawiają się na pasku tytułowym okna programu Excel). Oczywiście kiedy karty kontekstowe są widoczne na Wstążce, możesz swobodnie korzystać ze wszystkich poleceń znajdujących się na innych kartach.

Rodzaje poleceń dostępnych na Wstążce

W zdecydowanej większości przypadków polecenia dostępne na Wstążce działają tak, jak mógłbyś tego oczekiwąć. W praktyce można wyróżnić kilka różnych rodzajów poleceń dostępnych na kartach Wstążki:



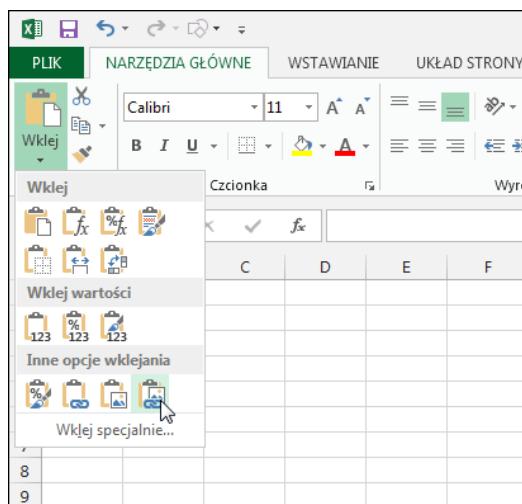
Rysunek 1.3. Kiedy zaznaczysz dany obiekt, na Wstążce pojawiają się karty kontekstowe zawierające polecenia związane z tym obiektem

- **Standardowe przyciski poleceń** — po naciśnięciu takiego przycisku wykonywane jest powiązane z nim polecenie. Przykładem takiego przycisku jest polecenie *Zwiększ rozmiar czcionki*, znajdujące się na karcie *NARZĘDZIA GŁÓWNE*, w grupie poleceń *Czcionka*. Niektóre przyciski wykonują swoje zadanie od razu po tym, jak zostaną kliknięte; naciśnięcie innych powoduje wyświetlenie okna dialogowego, w którym możesz wprowadzić dodatkowe informacje lub wybrać odpowiednie opcje. Niektóre przyciski posiadają krótkie opisy realizowanych funkcji, ale wiele polecen posiada tylko reprezentujące je przyciski bez podpisów.
- **Przyciski spełniające rolę przełączników** — przełącznik to specjalny przycisk włączający lub wyłączający daną funkcję, którego stan jest reprezentowany przez zmianę koloru przycisku. Przykładem takiego polecenia jest przycisk *Pogrubienie*, znajdujący się na karcie *NARZĘDZIA GŁÓWNE*, w grupie poleceń *Czcionka*. Jeżeli zawartość komórki nie jest pogrubiona, przycisk zachowuje swój normalny, domyślny kolor. Jeżeli jednak pogrubienie jest włączone, tło przycisku jest wyświetlane w innym kolorze. Kolejne naciśnięcia takiego przycisku odpowiednio włączają lub wyłączają powiązaną z nim funkcję.
- **Proste listy rozwijane** — jeżeli polecenie na Wstążce posiada małą, skierowaną w dół strzałkę, znajdująca się po prawej stronie przycisku, oznacza to, że taki przycisk to lista rozwijana. Po kliknięciu strzałki na ekranie rozwija się menu podrzędne (lista rozwijana) zawierające dodatkowy zestaw poleceń. Przykładem takiego polecenia jest *Scal i wyśrodkuj*, którego przycisk znajduje się na karcie *NARZĘDZIA GŁÓWNE* w grupie poleceń *Wyrównanie*. Po naciśnięciu strzałki tego przycisku na ekranie pojawia się lista czterech dodatkowych poleceń, związanych ze scalaniem i wyrównywaniem komórek.
- **Przyciski dzielone** — przycisk dzielony spełnia rolę normalnego przycisku (góra część przycisku) i listy rozwijanej (dolina część przycisku). Jeżeli klikniesz górną część przycisku, wykonywane jest polecenie powiązane bezpośrednio z tym przyciskiem. Jeżeli jednak klikniesz jego dolną część, na ekranie pojawia się lista rozwijana zawierająca szereg dodatkowych,

powiązanych poleceń. Przyciski dzielone łatwo zidentyfikować, ponieważ jeżeli ustawisz nad takim przyciskiem wskaźnik myszy, tło górnej i dolnej części przycisku jest wyświetlane w dwóch różnych kolorach. Przykładem przycisku dzielonego jest polecenie *Wklej*, znajdujące się na karcie *NARZĘDZIA GŁÓWNE*, w grupie poleceń *Schowek*. Kliknięcie górnej części przycisku po prostu wkleja do arkusza zawartość schowka systemowego. Jeżeli jednak klikniesz dolną część przycisku, na ekranie pojawi się lista rozwijana zawierająca dodatkowe polecenia powiązane z operacją wklejania (patrz rysunek 1.4).

Rysunek 1.4.

Polecenie *Wklej* jest dobrym przykładem przycisku dzielonego

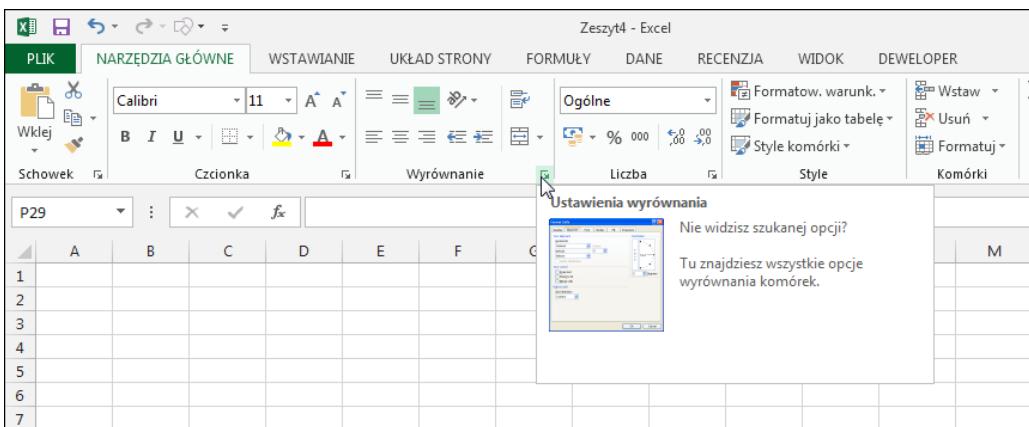


- **Pola wyboru** — pole wyboru to formant, który pozwala na włączanie bądź wyłączanie określonych opcji. Przykładem takiego pola jest opcja *Linie siatki*, znajdująca się na karcie *WIDOK* w grupie opcji *Pokazywanie*. Kiedy opcja *Linie siatki* jest zaznaczona, linie siatki w arkuszu są wyświetlane. Jeżeli usuniesz zaznaczenie tej opcji, linie siatki nie będą wyświetlane.
- **Pokrętla** (ang. *spinners*) — przykładem formantu wykorzystującego pokrętło jest polecenie *Skala*, znajdujące się w grupie poleceń *Skalowanie do rozmiaru* na karcie *UKŁAD STRONY*. Kliknięcie górnej strzałki pokrętła powoduje zwiększenie wartości powiązanego z nim elementu, odpowiednio kliknięcie dolnej strzałki powoduje zmniejszenie tej wartości.



Więcej szczegółowych informacji na temat modyfikacji i dopasowywania Wstążki do własnych potrzeb znajdziesz w rozdziale 20.

Niektóre grupy poleceń Wstążki posiadają w prawym, dolnym rogu małą ikonę polecenia *Uruchom okno dialogowe*. Ikonę tego polecenia znajdziesz na przykład w grupie poleceń *Wyrównanie*, znajdującej się na karcie *NARZĘDZIA GŁÓWNE* (patrz rysunek 1.5). Kiedy klikniesz tę ikonę, na ekranie pojawi się okno dialogowe *Formatowanie komórek*, otwarte na karcie *Wyrównanie*. W tym oknie dialogowym znajdziesz polecenia i opcje, które nie są dostępne na Wstążce.



Rysunek 1.5. Naciśnięcie przycisku Uruchom okno dialogowe wyświetla okno dialogowe zawierające dodatkowe opcje i polecenia

Pasek narzędzi Szybki dostęp

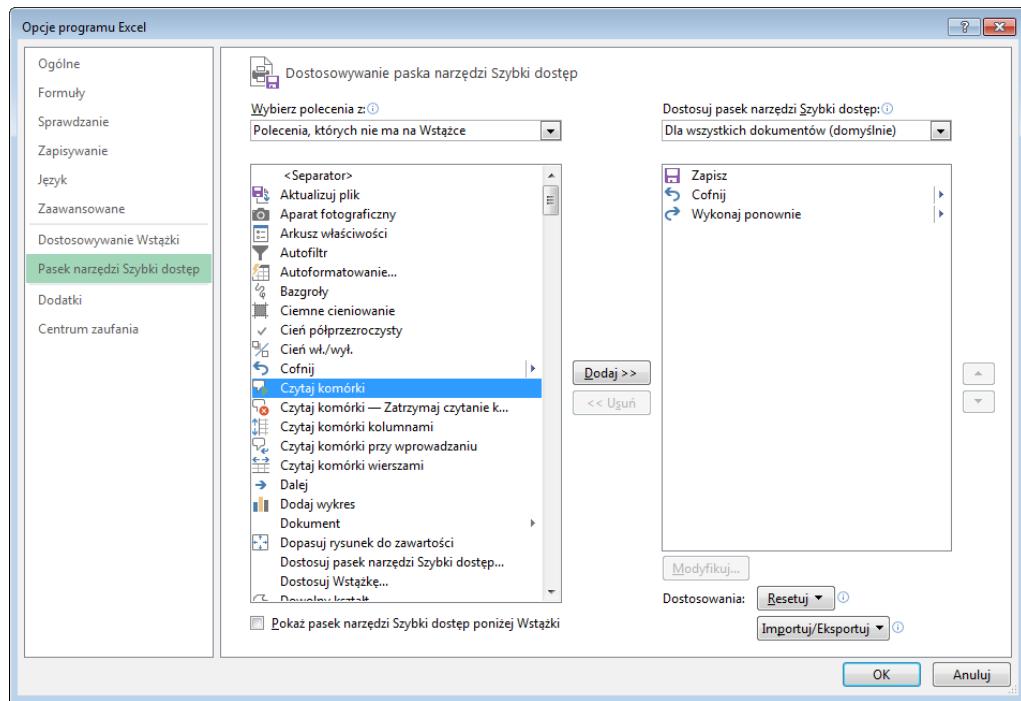
Pasek narzędzi *Szybki dostęp* to miejsce, w którym możesz umieścić przyciski najczęściej używanych poleceń. Pasek narzędzi *Szybki dostęp* jest zawsze widoczny na ekranie, niezależnie od tego, która karta Wstążki jest aktywna. Standardowo pasek narzędzi *Szybki dostęp* jest wyświetlany po lewej stronie paska tytułowego okna programu Excel, jednak w razie potrzeby możesz wyświetlić ten pasek poniżej Wstążki. Aby to zrobić, kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Pokaż pasek narzędzi Szybki dostęp poniżej Wstążki*.

Domyślnie na pasku narzędzi *Szybki dostęp* wyświetlane są przyciski trzech poleceń: *Zapisz*, *Cofnij* i *Wykonaj ponownie*. W razie potrzeby możesz umieścić na pasku przyciski najczęściej używanych poleceń. Aby to zrobić, kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Dostosuj pasek narzędzi Szybki dostęp*.

Excel posiada szereg poleceń, które nie są dostępne na Wstążce. W większości przypadków jedynym sposobem użycia takich poleceń jest umieszczenie reprezentujących je przycisków na wybranej karcie Wstążki lub na pasku narzędzi *Szybki dostęp*. Na rysunku 1.6 przedstawiono sekcję *Pasek narzędzi Szybki dostęp* okna dialogowego *Opcje programu Excel*. Jest to sekcja, która pozwala na dostosowanie paska narzędzi *Szybki dostęp* do potrzeb użytkownika. Aby szybko przywołać to okno na ekran, kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Dostosuj pasek narzędzi Szybki dostęp*.

Dostęp do poleceń Wstążki przy użyciu klawiatury

Na pierwszy rzut oka może się wydawać, że ze Wstążką można pracować tylko przy użyciu myszy. Żadne z poleceń na Wstążce nie posiada podkreślonych liter wskazujących na możliwość użycia kombinacji z klawiszem *Alt* do wywołania danego polecenia. W rzeczywistości jednak praca ze Wstążką przy użyciu klawiatury jest bardzo wygodna. Cała



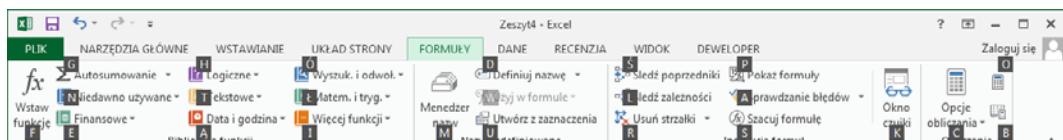
Rysunek 1.6. Sekcja Pasek narzędzi Szybki dostęp w oknie dialogowym Opcje programu Excel pozwala na dodawanie nowych poleceń do paska narzędzi Szybki dostęp

sztuczka polega na wcisnięciu klawisza *Alt*, co powoduje wyświetlenie na Wstążce klawiszy skrótu przypisanych do poszczególnych poleceń. Każde polecenie dostępne na Wstążce posiada przypisaną literę (lub sekwencję liter) pozwalających na uruchomienie takiego polecenia przy użyciu klawiatury.



Podczas wybierania poleceń Wstążki przy użyciu klawiatury klawisz *Alt* nie musi być cały czas wcisnięty.

Na rysunku 1.7 przedstawiono wygląd karty *FORMUŁY* po naciśnięciu sekwencji klawiszy *Alt*, *M* (zwróć uwagę na wyświetlone litery skrótów klawiszowych). Często naciśnięcie jednego klawisza powoduje wyświetlenie możliwych sekwencji kolejnych klawiszy. Na przykład: aby przy użyciu klawiatury wyrównać zawartość komórki do lewej strony, naciśnij klawisz *lewy Alt*, następnie klawisz *G* (karta *NARZĘDZIA GŁÓWNE*) i sekwencję klawiszy *WY* (polecenie *Wyrównaj tekst do lewej*). Jeżeli — podobnie jak ja — jesteś fanem używania klawiatury, to na zapamiętanie najważniejszych sekwencji klawiszy z pewnością nie będziesz musiał poświęcić wiele czasu.



Rysunek 1.7. Aby wyświetlić litery skrótów klawiszowych, naciśnij klawisz *Alt*

Po naciśnięciu klawisza *Alt* do przechodzenia pomiędzy kolejnymi poleceniami możesz używać klawiszy strzałek kurSORA. Kiedy przejdziesz na wybraną kartę, naciśnij klawisz *Strzałka w dół*, aby przejść do poleceń na karcie, i użyj klawiszy *Strzałka w lewo* i *Strzałka w prawo* do wybrania żądanego polecenia. Po zaznaczeniu danego polecenia możesz je uruchomić, naciskając klawisz *Enter*. Opisana technika nie jest tak efektywna, jak operowanie sekwencjami klawiszy, ale pozwala na szybkie zapoznanie się z poszczególnymi poleceniami Wstążki.



Uwaga

Excel 2013 nadal obsługuje skróty klawiszowe do obsługi poleceń menu, znane z Excela 2003, zatem jeżeli pamiętasz stare skróty klawiszowe swoich ulubionych poleceń, to nadal możesz z nich korzystać i używać ich zamiennie z nowymi sekwencjami klawiszy.

Krótki przegląd wersji Excela

Jeżeli masz zamiar tworzyć makra VBA, powinieneś zapoznać się z historią programu Excel. Do tej pory pojawiło się wiele różnych wersji Excela, a całkiem sporo z nich jest w użyciu do dnia dzisiejszego. Nietrudno sobie wyobrazić, że taka sytuacja może prowadzić do poważnych kłopotów z kompatybilnością makr między poszczególnymi wersjami. Więcej szczegółowych informacji na temat kompatybilności znajdziesz w rozdziale 24.

Poniżej zostało zamieszczone krótkie zestawienie wersji Excela, które pojawiły się na rynku do tej pory:

- **Excel 2** — Pierwszej wersji programu Excel for Windows nadano numer 2.0, tak aby zachować zgodność z wersją tego programu dla komputerów Macintosh. Excel 2 pojawił się na rynku w roku 1987.
- **Excel 3** — Pojawił się pod koniec roku 1990 i była to pierwsza wersja wyposażona w obsługę języka XLM.
- **Excel 4** — Ta wersja została opublikowana na początku roku 1992 i również obsługiwała makra w języku XLM.
- **Excel 5** — Excel 5 pojawił się na rynku na początku 1994 roku. W tej wersji po raz pierwszy został zaimplementowany język VBA (ale XLM nadal był obsługiwany). Już od dawna nie słyszałem, aby ktoś jeszcze korzystał z tej wersji.
- **Excel 95** — Technicznie rzecz biorąc, był nazywany wersją 7 (nie było wersji Excel 6). Sprzedaż tej wersji Excela rozpoczęła się latem 1995 roku. Obecnie jest rzadko używana.
- **Excel 97** — Ta wersja, znana również jako Excel 8, została wypuszczona na rynek w początkach roku 1997. Zawierała *bardzo wiele* ulepszeń, nowych funkcji i mechanizmów oraz zupełnie nowy interfejs projektowania aplikacji w oparciu o język VBA. Excel 97 używa również całkowicie innego formatu zapisu plików (którego poprzednie wersje nie mogły otwierać).
- **Excel 2000** — Była to pierwsza wersja, w której wprowadzono czterocyfrowy system numeracji. Excel 2000 (znany również jako Excel 9) miał swój debiut w czerwcu roku 1999. Pojawiło się w nim kilka znaczących zmian związanych z programowaniem i tworzeniem aplikacji. Obecnie Excel 2000 jest bardzo rzadko spotykany.
- **Excel 2002** — Wersja znana również jako Excel XP lub Excel 10, pojawiła się na rynku w połowie 2001 roku. Prawdopodobnie najważniejszą nową funkcją była możliwość naprawiania uszkodzonych plików i zapisywania wyników pracy w przypadku zawieszenia programu. Niektórzy użytkownicy nadal korzystają z tej wersji.
- **Excel 2003** — Ze wszystkich wersji Excela opublikowanych do tej pory Excel 2003 był prawdopodobnie najbardziej rozczarowującą aktualizacją tego popularnego arkusza kalkulacyjnego. W tej wersji nowości było naprawdę niewiele. Co ciekawe, do chwili obecnej jest to bardzo powszechnie spotykana i używana wersja Excela. Była to ostatnia wersja Excela wyposażona w klasyczny interfejs użytkownika.

- **Excel 2007** — Excel 2007 zasygnalizował początek nowej ery w dziedzinie arkuszy kalkulacyjnych. Znane z poprzednich wersji menu i paski narzędzi zostały zastąpione przez nowy interfejs o nazwie *Wstążka*. Zdecydowanym rozczarowaniem dla mnie był fakt, że nie można było modyfikować Wstążki za pomocą kodu VBA, jednak w tej wersji pojawiło się na tyle dużo nowych funkcji, mechanizmów, ulepszeń i modyfikacji, że byłem nią całkowicie usatysfakcjonowany. W Excelu 2007 został wprowadzony zupełnie nowy format plików skoroszytów, które od tej pory mogły zawierać ponad milion wierszy.
- **Excel 2010** — W tej wersji wprowadzono bardzo wiele nowych funkcji i mechanizmów (takich jak na przykład wykresy przebiegu w czasie, ang. *sparklines*) i ulepszono ogólną wydajność Excela. Dodatkowo, jeżeli musiałeś przetwarzanie skoroszyty o ogromnych rozmiarach, mogłeś zainstalować 64-bitową wersję tego programu. Niestety ponownie spotkało mnie pewne rozczarowanie, bo nadal nie było możliwości modyfikowania Wstążki za pomocą kodu VBA.
- **Excel 2013** — Ostatnią wersją Excela, jaka pojawiła się na rynku, jest wersja, w oparciu o którą napisałem tę książkę. Excel 2013 jest dostępny również w wersji sieciowej (za pośrednictwem przeglądarki internetowej) i może być używany na urządzeniach mobilnych, działających pod kontrolą systemu Windows RT. Oczywiście wersja ta nadal wykorzystuje Wstążkę, ale jej wygląd został znacznie zmieniony — i niestety znów nie można jej modyfikować za pomocą kodu VBA!

Menu podręczne i minipasek narzędzi

Oprócz systemu menu w edytorze VBE, jedyne „klasyczne” menu, jakie pozostało w programie Excel, to menu podręczne, które pojawia się na ekranie po kliknięciu komórki lub obiektu prawym przyciskiem myszy. Menu podręczne jest kontekstowe, co oznacza, że zestaw poleceń widocznych w menu jest uzależniony od miejsca lub obiektu, które zostały kliknięte prawym przyciskiem myszy. Możesz kliknąć dosłownie każdy element widoczny w oknie programu Excel — komórkę, krawędź wiersza lub kolumny, tytuł skoroszytu, pasek narzędzi i tak dalej.

Po kliknięciu wybranego elementu prawym przyciskiem myszy nad menu podręcznym pojawia się również minipasek narzędzi, który zapewnia szybki dostęp do najczęściej używanych poleceń formatujących. Na rysunku 1.8 przedstawiono wygląd takiego minipaska, który pojawia się po kliknięciu komórki prawym przyciskiem myszy.

Przy użyciu VBA nie możesz niestety bezpośrednio modyfikować poleceń na Wstążce, ale za to możesz użyć VBA do zmiany i dostosowania do własnych potrzeb menu podręcznego (z wyjątkiem minipaska narzędzi).



Więcej szczegółowych informacji na temat dostosowywania menu podręcznego do własnych potrzeb znajdziesz w rozdziale 21. Pamiętaj jednak, że wprowadzenie nowego, jednodokumentowego interfejsu programu Excel 2013 sprawiło, że modyfikacja menu podręcznego stała się znacznie bardziej skomplikowana.

Rysunek 1.8.

Kliknięcie wybranego obiektu prawym przyciskiem myszy powoduje wyświetlenie menu podręcznego i minipaska narzędzi

A	B	C	D	E	F	G	H
1		Poprzedni rok Bieżący rok					
2	Styczeń	78	98				
3	Luty	77	10	Calibri	11	A [▲] A [▼] A [↔] A [↕] % 000	
4	Marzec	75	10	B	I		
5	Kwiecień	82	10				
6	Maj	89	115				
7	Czerwiec	88	11				
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							

Okna dialogowe

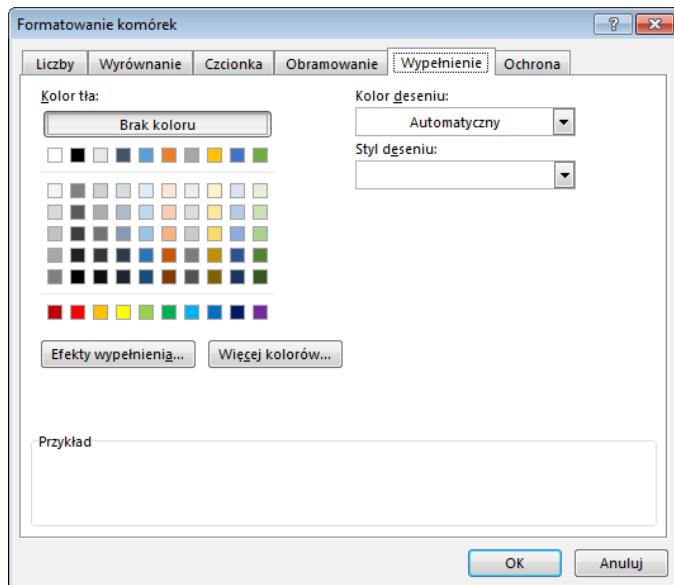
Niektóre polecenia dostępne na Wstążce wyświetlają na ekranie okna dialogowe. W wielu przypadkach w takich oknach dialogowych znajdują się polecenia i opcje, które nie są dostępne na Wstążce. Okna dialogowe wyświetlane w programie Excel można podzielić na dwie główne kategorie:

- **Modalne okna dialogowe.** Aby polecenia wybrane w takim oknie zostały wykonane, okno modalne musi zostać zamknięte. Przykładem jest okno dialogowe *Formatowanie komórek*. Żadna z opcji wybranych w tym oknie nie zostanie wykonana dopóki użytkownik nie zamknie okna, naciskając przycisk *OK*. Naciśnięcie przez użytkownika przycisku *Anuluj* również spowoduje zamknięcie okna, ale żadne zmiany nie zostaną przeprowadzone.
- **Niemodalne okna dialogowe.** Są to okna, które pozwalają użytkownikowi na interakcję z arkuszem, mimo że pozostają otwarte. Przykładem takiego okna dialogowego może być *Znajdowanie i zamienianie*. Okna niemodalne zamiast zestawu przycisków *OK* i *Anuluj* z reguły mają przycisk *Zamknij*.

Wiele okien dialogowych Excela jest wyposażonych w szereg kart, co przypomina nieco funkcjonowanie tradycyjnego notatnika, dzięki czemu pojedyncze okno dialogowe może spełniać rolę kilku różnych okien. Przykładem takiego okna jest okno dialogowe *Formatowanie komórek*, które zostało przedstawione na rysunku 1.9.

Rysunek 1.9.

Nowe okna dialogowe udostępniają wiele różnych opcji bez przeciążenia użytkownika nadmiernie „upakowanym” wyglądem okna



Projektanci aplikacji mogą tworzyć własne okna dialogowe za pomocą formularzy *UserForm*, które — jak się sam wkrótce przekonasz — pozwalają na tworzenie szerokiej gamy niestandardowych okien, począwszy od bardzo prostych, aż po złożone okna wykorzystujące karty poleceń.

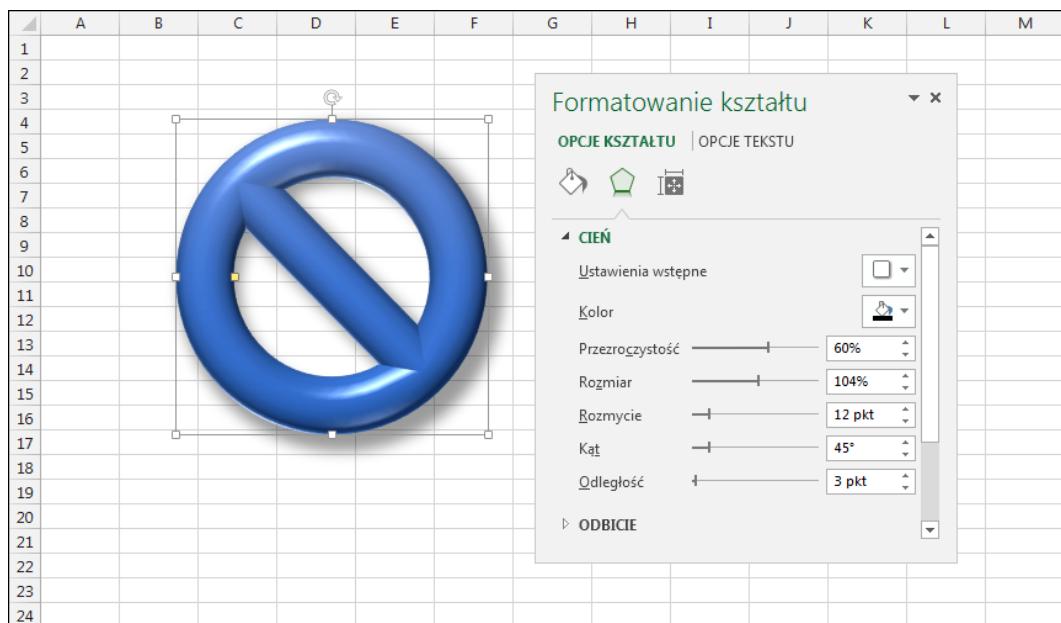


Więcej szczegółowych informacji na temat tworzenia formularzy *UserForm* znajdziesz w III części naszej książki.

Panel zadań

W Excelu 2002 wprowadzono nowy element interfejsu użytkownika, znany jako *Panel zadań*. Jest to dodatkowy, wielozadaniowy panel interfejsu użytkownika, domyślnie osadzony po prawej stronie okna programu (ale w razie potrzeby możesz go przeciągnąć w dowolne miejsce ekranu). Panel zadań jest używany do różnych celów, między innymi do wyświetlania zawartości schowka pakietu Office, wyświetlania listy pól tabel przestawnych, wstawiania obiektów typu *ClipArt*, oferowania wsparcia przy wyszukiwaniu i mapowaniu danych zapisanych w formacie XML.

W Excelu 2013 panele zadań spełniają bardzo rozszerzoną rolę. Na przykład formatowanie wykresów i wielu innych obiektów jest obecnie realizowane za pomocą paneli zadań, a nie jak do tej pory, za pomocą niemodalnych okien dialogowych. Na rysunku 1.10 przedstawiono panel *Formatowanie kształtu*.



Rysunek 1.10. Formatowanie obiektów jest jednym z zastosowań pasków zadań

Skróty klawiszowe

Excel posiada *bardzo wiele* użytecznych skrótów klawiszowych. Przykładowo aby skopiować do aktywnej komórki zawartość komórki znajdującej się bezpośrednio nad nią, możesz nacisnąć kombinację klawiszy *Ctrl+D*. Jeżeli jesteś początkującym użytkownikiem Excela — albo zależy Ci jedynie na zwiększeniu efektywności pracy — zdecydowanie powinieneś zapoznać się z zawartością pomocy systemowej (rozpocznij od przeglądania tematów pomocy w kategorii *Ułatwienia dostępu*). Opanowanie skrótów klawiszowych jest kluczowym elementem sprawnego posługiwania się programem Excel. W pliku pomocy znajdziesz tabele zawierające zestawienie przydatnych poleceń Excela wraz ze skortami klawiszowymi.

Oczywiście — jak już wspomniano wcześniej — klawiatury możesz używać do pracy z poleceniami znajdującymi się na Wstążce.

Co nowego w edytorze VBE?

Nic.

Zdecydowana większość nowych obiektów Excela 2013 jest dostępna za pośrednictwem kodu VBA, ale sam edytor VBE nie zmienił się już od wielu wersji programu Excel. Wszystkie aplikacje pakietu Microsoft Office wykorzystują nowy interfejs użytkownika, czyli Wstążkę od wersji Excel 2007, ale edytor VBE nadal korzysta ze starego, dobrego menu i tradycyjnych pasków narzędzi. Porównując go z wyglądem poszczególnych aplikacji, wielu użytkowników stwierdza, że edytor VBE wygląda już nieco staroświecko, więc może już niebawem, w kolejnej edycji pakietu Office zobaczymy jego nowe oblicze (aczkolwiek wcale nie byłbym tego taki pewny...).

Wprowadzanie danych

Wprowadzanie danych w Excelu jest całkiem proste. Program interpretuje zawartość każdej komórki jako jeden z następujących typów danych:

- Wartość liczbową (w tym data i czas)
- Tekst
- Wartość logiczną (Prawda lub Fałsz)
- Formuła

Formuły zawsze rozpoczynają się znakiem równości (=). Jednak Excel potrafi się dostosować do nawyków użytkowników aplikacji 1-2-3 i akceptuje symbol @, znak plusa (+) lub minusa (-) jako pierwszy znak formuły. Program po wcisnięciu klawisza *Enter* automatycznie modyfikuje znak formuły.

Formuły, funkcje i nazwy

Formuły sprawiają, że arkusz kalkulacyjny jest tym, czym jest. Excel posiada wiele zaawansowanych mechanizmów związanych z formułami, które umożliwiają na przykład definiowanie formuł tablicowych, stosowanie operatora przecięcia, dodawanie łączyc i tworzenie *megaformuł* (to moje własne określenie długich i złożonych — ale za to bardzo wydajnych — formuł).



W rozdziale 2. omówiono formuły i zaprezentowano wiele wskazówek i porad dotyczących ich stosowania.

Excel posiada też kilka przydatnych funkcji inspekcji pomocnych w identyfikowaniu błędów lub śledzeniu logiki w nieznanym arkuszu kalkulacyjnym. Aby skorzystać z takich możliwości, użyj poleceń, które znajdziesz na karcie *FORMUŁY*, w grupie poleceń *Inspekcja formuł*.

Jednym z bardziej użytecznych poleceń jest polecenie *Sprawdzanie błędów*, które znajdziesz na karcie *FORMUŁY*, w grupie *Inspekcja formuł*. Uruchomienie tego polecenia spowoduje przeanalizowanie arkusza i sprawdzenie go pod kątem istnienia potencjalnych błędów w formułach. Na rysunku 1.11 przedstawiono okno dialogowe sygnalizujące odnalezienie takiej potencjalnie błędnej formuły i wskazujące możliwe sposoby rozwiązania problemu.

Funkcje arkusza umożliwiają wykonywanie obliczeń lub operacji, które w przeciwnym razie byłyby niemożliwe. Excel dysponuje ogromną liczbą wbudowanych funkcji arkuszowych.

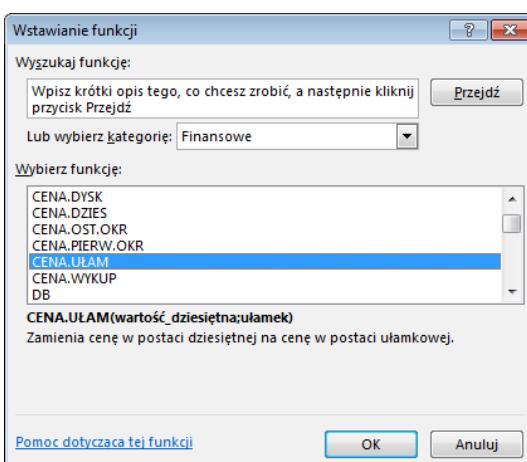
Najprostsza metoda zlokalizowania funkcji, której zamierzasz użyć, polega na zastosowaniu okna dialogowego *Wstawianie funkcji* pokazanego na rysunku 1.12. Aby przywołać to okno na ekran, naciśnij przycisk *Wstaw funkcję* znajdujący się na pasku formuły,

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		432	341	315	187	490	265							
2		577	445	216	278	573	508							
3		410	290	451	441	181	163							
4		186	229	372										
5		212	416	512										
6		477	268	267										
7		402	438	185										
8		359	177	464										
9		267	208	289										
10		335	530	457										
11		440	442	365										
12		578	389	551										
13		297	410	537										
14		568	519	338	532	367	268							
15		343	168	445	399	170	213							
16	RAZEM	5883	5270	5449	6073	5997	4230							
17														
18														

Rysunek 1.11. Excel potrafi monitorować formuły pod kątem występowania potencjalnych błędów

Rysunek 1.12.

Najlepszą metodą wstawiania funkcji w formule jest użycie okna dialogowego Wstawianie funkcji



albo zamiast tego po prostu naciśnij kombinację klawiszy *Shift+F3*. Po wybraniu funkcji na ekranie pojawi się okno dialogowe *Argumenty funkcji*, które ułatwia zdefiniowanie argumentów wybranej funkcji.



Excel pozwala też tworzyć własne funkcje arkusza za pomocą języka VBA. Aby uzyskać szczegółowe informacje na ten temat, zajrzyj do rozdziału 8.

Nazwa jest identyfikatorem umożliwiającym odwołanie się do komórki, zakresu, wartości, formuły lub graficznego obiektu. Formuły używające nazw są znacznie czytelniejsze od formuł opartych na odwołaniach do komórek, a tworzenie formuł stosujących odwołania do nazw jest znacznie łatwiejsze.



Nazwy i metody ich definiowania omówimy w rozdziale 2. Jak się sam przekonasz, w niektórych sytuacjach Excel obsługuje nazwy w dosyć szczególny sposób.

Mechanizm Flash Fill (wypełnianie błyskawiczne)

Mechanizm wypełniania błyskawicznego (ang. *Flash Fill*) wykorzystuje algorytmy rozpoznawania wzorców do wyodrębniania lub łączenia danych z wybranych kolumn. Działa to tak, że użytkownik wpisuje kilka pierwszych przykładów i Excel próbuje samodzielnie odgadnąć wzorzec i automatycznie wypełnić pozostałe komórki w kolumnie. W niektórych sytuacjach mechanizm ten może całkowicie wyeliminować konieczność stosowania formuł.

Na rysunku zamieszczonym poniżej mechanizm wypełniania błyskawicznego został użyty do wyodrębnienia nazwisk, imion i inicjałów imienia z kolumny A do osobnych kolumn.

Mechanizm wypełniania błyskawicznego znakomicie sprawdza się, kiedy przetwarzane dane mają spójny, ujednolicony format. Mimo to powinieneś zawsze zweryfikować wyniki działania tego mechanizmu, tak aby upewnić się, że rozpoznawanie wzorców zadziałało prawidłowo w każdym przypadku.

	A	B	C	D	E
1	Adamczyk Jakub	Adamczyk	Jakub	J.	
2	Adamczyk Katarzyna	Adamczyk	Katarzyna	K.	
3	Baran Kacper	Baran	Kacper	K.	
4	Górska Daria	Górska	Daria	D.	
5	Górski Filip	Górski	Filip	F.	
6	Grabowska Alicja	Grabowska	Alicja	A.	
7	Jabłoński Bartosz	Jabłoński	Bartosz	B.	
8	Jaworska Dominika	Jaworska	Dominika	D.	
9	Kaczmarek Dawid	Kaczmarek	Dawid	D.	
10	Kaczmarek Gabriela	Kaczmarek	Gabriela	G.	
11	Kamińska Amelia	Kamińska	Amelia	A.	
12	Kamiński Adam	Kamiński	Adam	A.	
13	Kowalski Jan	Kowalski	Jan	J.	
14	Kozłowski Igor	Kozłowski	Igor	I.	
15	Krawczyk Hanna	Krawczyk	Hanna	H.	
16	Lewandowski Dominik	Lewandowski	Dominik	D.	
17					
18					

Zaznaczanie obiektów

Zaznaczanie obiektów w Excelu odbywa się za pośrednictwem standardowych metod systemu Windows. Kliknięcie komórki, przytrzymanie lewego klawisza myszy, a następnie przeciągnięcie w dowolnym kierunku pozwala zaznaczyć zakres komórek (aczkolwiek korzystając z odpowiednich skrótów klawiszowych, możesz pracować znacznie bardziej efektywnie). Kliknięcie obiektu umieszczonego w warstwie rysunkowej powoduje jego zaznaczenie. Aby zaznaczyć wiele obiektów lub nieciągłych zakresów komórek, powinieneś podczas zaznaczania trzymać wcisknięty klawisz *Ctrl*.



Kliknięcie wybranego elementu wykresu powoduje jego zaznaczenie. Aby zaznaczyć sam obiekt wykresu, naciśnij klawisz *Ctrl* i kliknij dowolne miejsce wykresu.

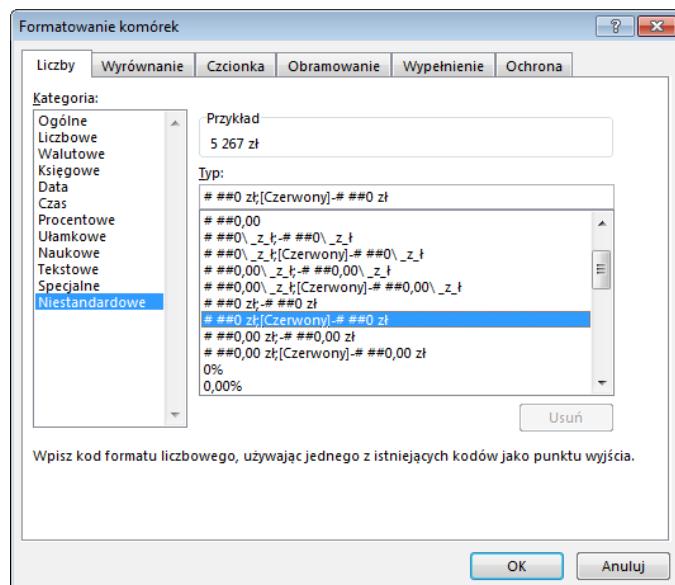
Jeżeli do obiektu zostało przypisane makro, kliknięcie obiektu spowoduje wykonanie makra. Aby zaznaczyć taki obiekt, należy kliknąć go prawym przyciskiem myszy i nacisnąć klawisz *Esc* w celu ukrycia menu podręcznego. Zamiast tego możesz również kliknąć taki obiekt po wcisnięciu i przytrzymaniu klawisza *Ctrl*.

Formatowanie

Excel oferuje dwa rodzaje formatowania: formatowanie wartości numerycznych oraz formatowanie wyglądu arkusza.

Formatowanie wartości numerycznych dotyczy sposobu reprezentacji wartości liczbowej w komórce. Możesz albo wybrać format z długiej listy predefiniowanych formatów, albo utworzyć własny sposób formatowania (patrz rysunek 1.13). Sposób postępowania podczas formatowania wartości numerycznych został dokładnie omówiony w systemie pomocy programu Excel.

Rysunek 1.13.
Excel oferuje dużą elastyczność opcji formatowania numerycznego



W zależności od wartości wpisanej do komórki, Excel automatycznie stosuje kilka typów formatowania liczb. Jeśli na przykład za liczbą wstawisz symbol waluty (np. zł w przypadku Polski), Excel automatycznie nada takiej komórce format walutowy. W razie potrzeby możesz również użyć mechanizmu formatowania warunkowego do różnicowania formatowania w zależności od wartości liczb zapisanych w poszczególnych komórkach arkusza.

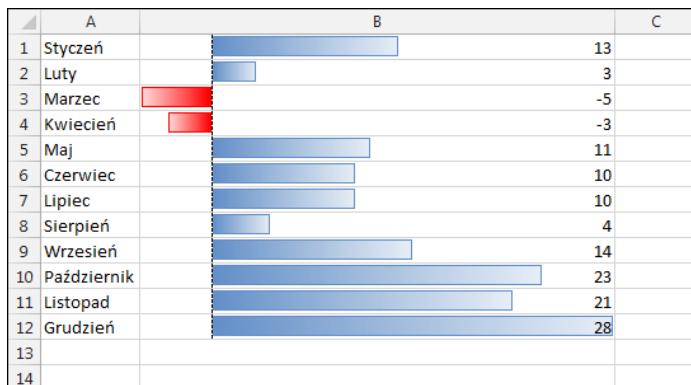
Formatowanie wyglądu arkusza odnosi się do formatowania stosowanego w celu uatrakcyjnienia wyglądu arkusza. Na Wstążce jest wiele przycisków poleceń i opcji formatowania arkusza, a pełny zestaw opcji formatowania obiektów znajdziesz na panelu zadań *Formatowanie*.

Najprostsza metoda pozwalająca otworzyć właściwe okno dialogowe i nadać obiekowi odpowiedni sposób formatowania polega na zaznaczeniu obiektu i wcisnięciu kombinacji klawiszy *Ctrl+1*. Zamiast tego możesz również kliknąć obiekt prawym przyciskiem myszy i wybrać z menu podręcznego pozycję *Formatuj xxx*, gdzie *xxx* oznacza wybrany obiekt. Rezultatem dowolnej z tych operacji będzie pojawienie się na ekranie panelu zadań, zawierającego wszystkie opcje formatowania dostępne dla zaznaczonego obiektu.

Excel nie ma panelu zadań pozwalającego na formatowanie komórek arkusza.

Funkcja formatowania warunkowego w Excelu jest szczególnie użyteczna. Aby ją uaktywnić, przejdź na kartę *NARZĘDZIA GŁÓWNE* i naciśnij przycisk *Formatów warunk.*, znajdujący się w grupie opcji *Style*. Mechanizm formatowania warunkowego umożliwia zdefiniowanie formatowania, które zostanie zastosowane tylko wtedy, gdy będą spełnione określone warunki. Mechanizm ten oferuje szereg ciekawych opcji formatowania warunkowego, takich jak paski danych, skale kolorów czy zestawy ikon. Na rysunku 1.14 przedstawiono wygląd pasków danych, za pomocą których utworzono histogram wartości bezpośrednio w komórkach arkusza.

Rysunek 1.14.
Paski danych to jedna z efektownych możliwości mechanizmu formatowania warunkowego



Opcje ochrony

Excel oferuje kilka różnych opcji ochrony arkusza i skoroszytu. Na przykład, możesz chronić formuły przed nadpisaniem lub modyfikacją, chronić strukturę skoroszytu, a także zabezpieczać skoroszyt za pomocą hasła oraz chronić kod VBA.

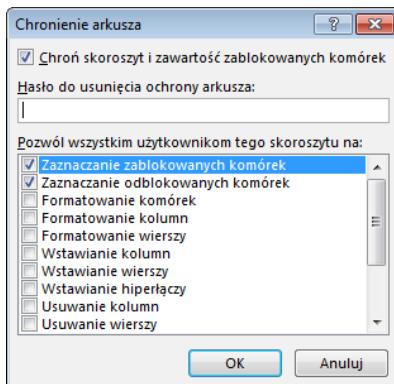
Ochrona formuł przed nadpisaniem

W wielu sytuacjach bardzo przydaje się możliwość ochrony formuły przed przypadkowym (albo celowym) nadpisaniem lub modyfikacją. Aby włączyć ochronę formuł, wykonaj następujące polecenia:

- Zaznacz komórki, które *mogą* zostać nadpisane.
- Kliknij zaznaczony obszar prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Formatuj komórki*.
- Na ekranie pojawi się okno dialogowe *Formatowanie komórek*. Przejdź na kartę *Ochrona*.
- Na karcie *Ochrona* wyłącz opcję *Zablokuj*.
- Kliknij *OK*, aby zamknąć okno dialogowe *Formatowanie komórek*.

- Przejdź na kartę *RECENZJA* i naciśnij przycisk *Chroń skoroszyt*, znajdujący się w grupie opcji *Zmiany*. Na ekranie pojawi się okno dialogowe *Chronienie arkusza*, przedstawione na rysunku 1.15.

Rysunek 1.15.
Okno dialogowe
Chronienie arkusza



- W oknie dialogowym *Chronienie arkusza* zaznacz żądaną opcję; jeżeli to konieczne, zdefiniuj hasło ochrony arkusza, a następnie naciśnij przycisk *OK*.



Domyślnie wszystkie komórki arkusza są zablokowane, aczkolwiek nie ma to żadnego wpływu na funkcjonowanie arkusza dopóty, dopóki nie zostanie włączony mechanizm ochrony arkusza.

W razie potrzeby możesz również ukryć formuły, dzięki czemu nie będą się one pojawiały na pasku formuły po zaznaczeniu komórki. Aby to zrobić, zaznacz komórki, w których chcesz ukryć formuły, i upewnij się, że na karcie *Ochrona* okna dialogowego *Formatowanie komórek* zaznaczyłeś opcję *Ukryj*.

Ochrona struktury skoroszytu

Po uaktywnieniu ochrony struktury skoroszytu nie możesz dodawać ani usuwać arkuszy. Aby wyświetlić okno dialogowe *Chronienie skoroszytu*, przejdź na kartę *RECENZJA* i naciśnij przycisk *Chroń skoroszyt*, znajdujący się w grupie opcji *Zmiany*. Upewnij się, że włączyłeś opcję *Struktura*.

Jeżeli zaznaczyłeś również opcję *Okna*, nie będzie możliwa zmiana ani wielkości okna skoroszytu, ani jego lokalizacji.



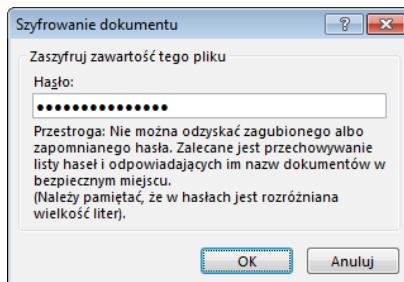
W Excelu 2013 opcja *Okna* jest zablokowana (nieaktywna). Nowy, jednodokumentowy interfejs tego programu nie pozwala na korzystanie z okien o stałym położeniu ani z okien skoroszytów o stałym rozmiarze.

Ochrona skoroszytu przy użyciu hasła

W niektórych przypadkach użytkownikowi może zależeć na ograniczeniu dostępu do skoroszytu tylko do osób znających hasło.

Aby wybrany plik skoroszytu zabezpieczyć hasłem, przejdź na kartę *PLIK*, kliknij opcję *Informacje*, a następnie naciśnij przycisk *Chroń skoroszyt* i z menu podręcznego wybierz polecenie *Szyfruj przy użyciu hasła*. Na ekranie pojawi się okno dialogowe *Szyfrowanie dokumentu* (patrz rysunek 1.16). Korzystając z tego okna dialogowego, możesz zdefiniować hasło, którego podanie będzie niezbędne do otwarcia tak zabezpieczonego skoroszytu.

Rysunek 1.16.
Okno dialogowe
Szyfrowanie dokumentu umożliwia
zabezpieczenie hasłem
pliku skoroszytu

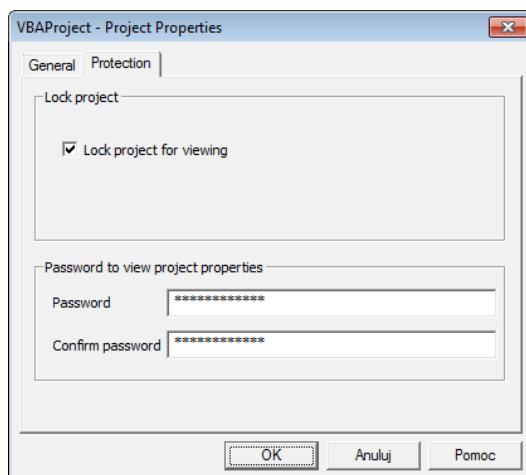


Ochrona kodu VBA przy użyciu hasła

Jeżeli Twój skoroszyt zawiera kod VBA, w niektórych sytuacjach bardzo przydatna może być możliwość zabezpieczenia tego kodu przed przeglądaniem i modyfikacją za pomocą hasła dostępu. Aby to zrobić, uaktywnij edytor *Visual Basic* (na przykład naciskając kombinację klawiszy *lewyAlt+F11*) i w oknie *Project* zaznacz wybrany projekt. Następnie z menu *Tools* wybierz polecenie *xxxx Properties*, gdzie xxxx odpowiada nazwie projektu. Na ekranie pojawi się okno dialogowe właściwości projektu.

W oknie dialogowym *Project Properties* przejdź na kartę *Protection* (patrz rysunek 1.17). Zaznacz opcję *Lock project for viewing*, dwukrotnie wprowadź hasło, naciśnij przycisk *OK* i zapisz plik skoroszytu. Po zamknięciu pliku i jego ponownym otwarciu konieczne będzie podanie hasła dostępu przed uzyskaniem możliwości przeglądnięcia kodu lub jego modyfikacji.

Rysunek 1.17.
Okno dialogowe *Project Properties* umożliwia
włączenie ochrony
projektu VBA





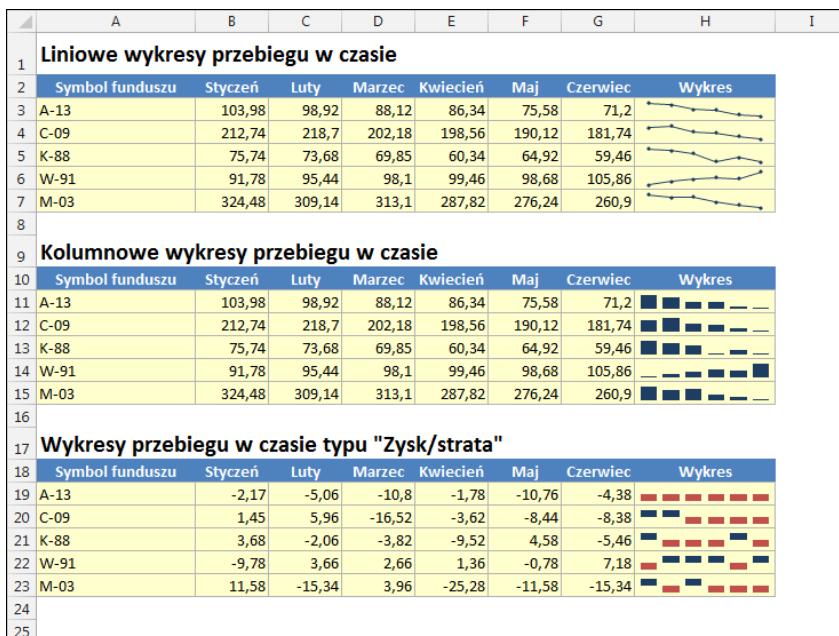
Pamiętaj, że Excel tak naprawdę nie jest szczególnie bezpieczną aplikacją. Owszem, mechanizmy ochrony, nawet te wykorzystujące hasła, blokują przypadkowym użytkownikom dostęp do różnych komponentów skoroszytu, ale każdy, komu naprawdę zależy na złamaniu zabezpieczeń, dokona tego, sięgając po łatwo dostępne narzędzia służące do łamania haseł lub wykorzystując niektóre mniej lub bardziej znane „tajemnice” Excela.

Wykresy

Excel jest aplikacją prawdopodobnie najczęściej na świecie wykorzystywaną do tworzenia wykresów. Jak już wcześniej wspominaliśmy, wykresy mogą być albo przechowywane w arkuszu wykresów, albo swobodnie osadzane na zwykłym arkuszu danych. Excel 2013 oferuje również szereg nowych narzędzi, dzięki którym modyfikacja i dostosowywanie wykresów do własnych potrzeb stało się łatwiejsze niż kiedykolwiek wcześniej.

W Excelu możesz również tworzyć *wykresy przestawne*, połączone z tabelą przestawną. Dzięki wykresom przestawnym możesz prezentować różne zestawienia danych przy użyciu takich samych metod, jak w przypadku tabeli przestawnej.

Wykresy przebiegu w czasie (ang. *Sparklines*), czyli mechanizm, który pojawił się już w Excelu 2010, to rodzaj miniwykresów mieszczących się w pojedynczej komórce danych. Ten nowy rodzaj wykresów, jest zupełnie niezależny od standardowych wykresów oferowanych przez Excela. Na rysunku 1.18 przedstawiono arkusz danych zawierający kilka przykładowych wykresów przebiegu w czasie.



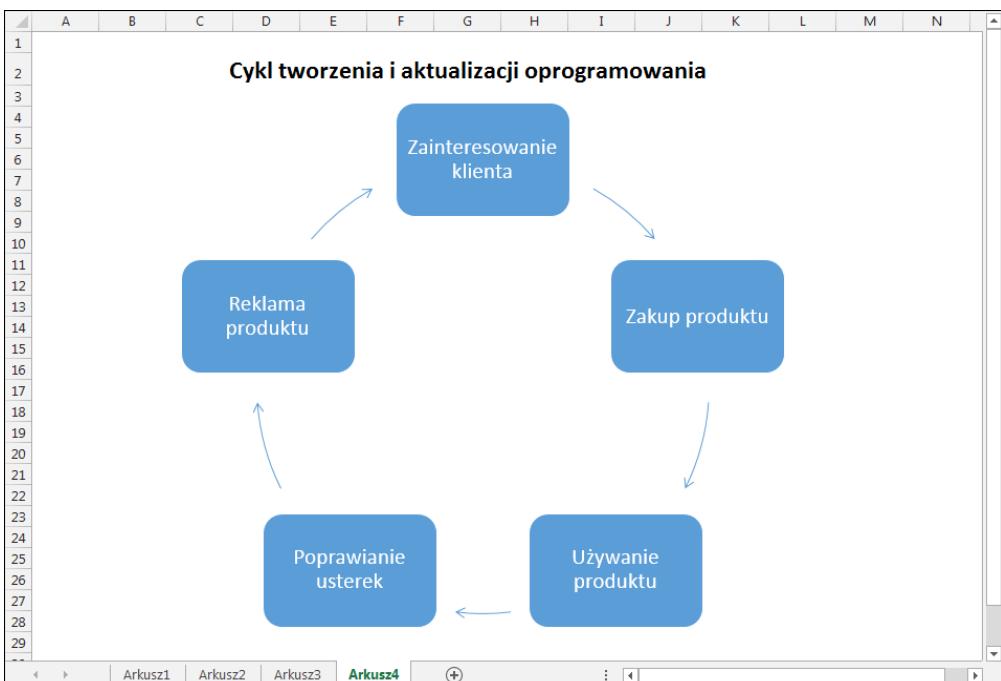
Rysunek 1.18. Przykładowe wykresy przebiegu w czasie

Kształty i obiekty typu SmartArt

Jak już wcześniej wspominaliśmy, każdy arkusz Excela posiada niewidoczną warstwę rysunkową przechowującą wykresy, mapy, obrazy, formanty (takie jak przyciski i pola listy) oraz kształty.

Excel pozwala na umieszczanie różnych kształtów bezpośrednio na arkuszu. Aby wyświetlić na ekranie galerię dostępnych kształtów, przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Kształty* znajdujący się w grupie opcji *Ilustracje*. Kształty można w łatwy sposób dostosowywać, a nawet umieszczać na nich tekst. Ponadto kształty i inne elementy graficzne możesz grupować w pojedynczy obiekt; zgrupowane obiekty łatwiej skalować, łatwiej też zmieniać ich położenie.

Nowym elementem graficznym wdrożonym już w pakiecie Office 2007, były obiekty typu SmartArt, które pozwalają na tworzenie szerokiej gamy diagramów. Przykład diagramu wykonanego przy użyciu obiektów typu SmartArt przedstawiono na rysunku 1.19.



Rysunek 1.19. Diagram wykonany przy użyciu obiektów SmartArt

Dostęp do baz danych

W miarę upływu lat w zdecydowanej większości arkuszy kalkulacyjnych zostały zaimplementowane mechanizmy pozwalające użytkownikom na korzystanie z prostych baz danych. Również Excel ma kilka przydatnych narzędzi bazodanowych, które można podzielić na dwie kategorie:

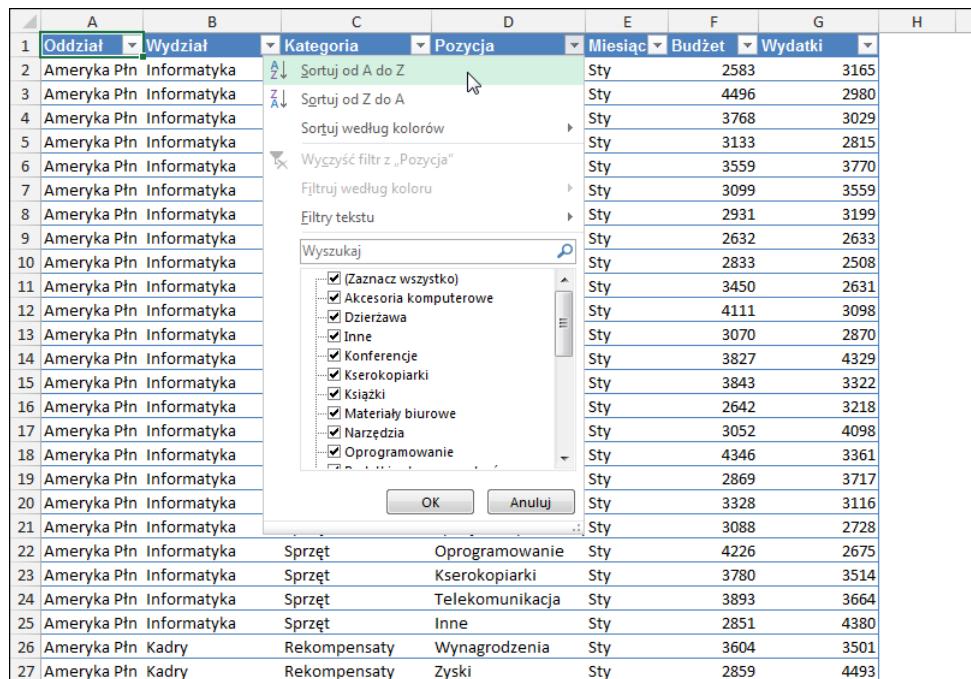
- **Arkuszowe bazy danych.** Cała baza danych jest przechowywana w arkuszu, przez co jej wielkość jest ograniczona maksymalnym rozmiarem arkusza.
- **Zewnętrzne bazy danych.** Dane są przechowywane w jednym lub kilku plikach dyskowych z dostępem na żądanie.

Arkuszowe bazy danych

W ogólnym przypadku dowolny, prostokątny obszar danych zawierający nagłówki kolumn może być uważany za arkuszową bazę danych.

Excel 2007 był pierwszą wersją, która pozwalała na zamianę określonego zakresu komórek na tabelę danych. Aby to zrobić, zaznacz dowolną komórkę znajdującej się w prostokątnym obszarze danych, przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Tabela*, znajdujący się w grupie opcji *Tabele*. Zastosowanie tabel ma wiele zalet — na przykład automatyczne podsumowania w ostatnim wierszu tabeli, ułatwione filtrowanie i sortowanie, automatyczne wypełnianie kolumn formułami czy uproszczone formatowanie. Dodatkowo, jeżeli tworzysz wykres na bazie tabeli, to po dodaniu nowych wierszy do tabeli wykres jest automatycznie rozszerzany.

Szczególnie wygodnie pracuje się z kolumnami tabeli. Nagłówki poszczególnych kolumn to tak naprawdę listy rozwijane zawierające szereg polecen ułatwiających filtrowanie i sortowanie danych (patrz rysunek 1.20). Wiersze tabeli, które nie spełniają wybranych warunków filtrowania, są tymczasowo ukrywane.



The screenshot shows a Microsoft Excel spreadsheet with data in columns A through H. The first few rows contain category names like 'Oddział' and 'Wydział'. Column C contains 'Kategoria' and column D contains 'Pozycja'. A filter menu is open over column D, showing sorting options ('Sortuj od A do Z', 'Sortuj od Z do A', 'Sortuj według kolorów'), a clear filter button ('Wyczść filtr z „Pozycja”'), a filter by color button ('Filtruj według koloru'), a filter by text button ('Filtry tekstu'), and a search bar ('Wyszukaj'). Below these are checkboxes for various categories: 'Zaznacz wszystko', 'Akcesoria komputerowe', 'Dzierżawa', 'Inne', 'Konferencje', 'Kserokopiarki', 'Książki', 'Materiały biurowe', 'Narzędzia', and 'Oprogramowanie'. At the bottom of the filter menu are 'OK' and 'Anuluj' buttons. The main body of the table lists data for each row, including 'Miesiąc' (Sty), 'Budżet', and 'Wydatki'.

	A	B	C	D	E	F	G	H
1	Oddział	Wydział	Kategoria	Pozycja	Miesiąc	Budżet	Wydatki	
2	Ameryka Płn	Informatyka		Sortuj od A do Z	Sty	2583	3165	
3	Ameryka Płn	Informatyka		Sortuj od Z do A	Sty	4496	2980	
4	Ameryka Płn	Informatyka		Sortuj według kolorów	Sty	3768	3029	
5	Ameryka Płn	Informatyka			Sty	3133	2815	
6	Ameryka Płn	Informatyka			Sty	3559	3770	
7	Ameryka Płn	Informatyka			Sty	3099	3559	
8	Ameryka Płn	Informatyka			Sty	2931	3199	
9	Ameryka Płn	Informatyka			Sty	2632	2633	
10	Ameryka Płn	Informatyka			Sty	2833	2508	
11	Ameryka Płn	Informatyka			Sty	3450	2631	
12	Ameryka Płn	Informatyka			Sty	4111	3098	
13	Ameryka Płn	Informatyka			Sty	3070	2870	
14	Ameryka Płn	Informatyka			Sty	3827	4329	
15	Ameryka Płn	Informatyka			Sty	3843	3322	
16	Ameryka Płn	Informatyka			Sty	2642	3218	
17	Ameryka Płn	Informatyka			Sty	3052	4098	
18	Ameryka Płn	Informatyka			Sty	4346	3361	
19	Ameryka Płn	Informatyka			Sty	2869	3717	
20	Ameryka Płn	Informatyka			Sty	3328	3116	
21	Ameryka Płn	Informatyka			Sty	3088	2728	
22	Ameryka Płn	Informatyka	Sprzęt	Oprogramowanie	Sty	4226	2675	
23	Ameryka Płn	Informatyka	Sprzęt	Kserokopiarki	Sty	3780	3514	
24	Ameryka Płn	Informatyka	Sprzęt	Telekomunikacja	Sty	3893	3664	
25	Ameryka Płn	Informatyka	Sprzęt	Inne	Sty	2851	4380	
26	Ameryka Płn	Kadry	Rekompensaty	Wynagrodzenia	Sty	3604	3501	
27	Ameryka Płn	Kadry	Rekompensaty	Zyski	Sty	2859	4493	

Rysunek 1.20. Tabele w Excelu ułatwiają filtrowanie i sortowanie wierszy danych

Zewnętrzne bazy danych

Aby skorzystać z tabel zewnętrznej bazy danych, przejdź na kartę *DANE* i skorzystaj z poleceń znajdujących się w grupie *Dane zewnętrzne*. Excel 2013 może korzystać z danych zewnętrznych przechowywanych w wielu różnych rodzajach baz danych.

Funkcje internetowe

Excel oferuje szereg funkcji związanych z siecią Internet. Istnieje na przykład możliwość zapisania arkusza lub całego skoroszytu jako pliku w formacie HTML, obsługiwany przez przeglądarki internetowe. Ponadto bezpośrednio w komórkach można wstawić hiperłącza (a także adresy poczty elektronicznej).



W wersjach Excela wcześniejszych niż 2007, zapis skoroszytu w formacie HTML był „podróżą tam i z powrotem” — innymi słowy, mogłeś zapisać plik w formacie HTML, zamknąć go i następnie ponownie otworzyć w Excelu, a żaden element arkusza nie ulegał zmianie. Począwszy od Excela 2007 nie ma już takiej możliwości. Format HTML jest obecnie wykorzystywany już tylko do eksportu danych w jedną stronę.

W Excelu możesz również tworzyć zapytania sieciowe, które pozwalają na pobieranie danych przechowywanych w korporacyjnych sieciach intranetowych lub w sieci Internet. Na rysunku 1.21 przedstawiono przykład takiego zapytania sieciowego.

The screenshot shows the Microsoft Excel ribbon with the 'Data' tab selected. A dropdown menu is open under the 'Get Data' button, with 'From Web' highlighted. The main area of the screen displays a web browser interface for the 'State & County QuickFacts' page from the US Census Bureau. The URL in the address bar is <http://quickfacts.census.gov/qfd/states/04000.html>. The page content includes a map of Arizona with county selection, a city search dropdown, and various demographic data tables for Arizona.

	Arizona	USA
Population, 2012 estimate	6,553,255	313,914,040
Population, 2010 (April 1) estimates base	6,392,015	308,747,508
Population, percent change, April 1, 2010 to July 1, 2012	2.5%	1.7%
Population, 2010	6,392,017	308,745,538
Persons under 5 years, percent, 2011	6.9%	6.5%
Persons under 18 years, percent, 2011	25.1%	23.7%
Persons 65 years and over, percent, 2011	14.2%	13.3%

Rysunek 1.21. Tworzenie zapytania sieciowego pozwalającego na importowanie danych do arkusza

Narzędzia analizy danych

Pod względem możliwości analizy Excel z pewnością ma się czym pochwalić. W końcu właśnie z tego powodu użytkownicy korzystają z arkusza kalkulacyjnego. Większość zadań analizy może być realizowana przy użyciu formuł, ale Excel oferuje również wiele innych narzędzi.

- **Konspekty** — Konspekt arkusza jest znakomitym narzędziem do przetwarzania danych hierarchicznych, takich jak budżet. Excel potrafi automatycznie generować konspekty danych (poziome, pionowe lub łączące oba kierunki), możesz też tworzyć je ręcznie. Po utworzeniu konspektu możesz rozwijać lub ukrywać poszczególne poziomy danych.
- **Dodatek Analysis ToolPak** — W poprzednich wersjach Excela dodatek *Analysis ToolPak* dostarczał specjalnych narzędzi i funkcji arkuszowych przeznaczonych do analizy danych (głównie statystycznej). Począwszy od wersji 2007, wszystkie funkcje i narzędzia dodatku zostały wbudowane do Excela, dzięki czemu program ten znakomicie nadaje się do wykonywania analiz statystycznych.
- **Tabele przestawne** — Jednym z najwydajniejszych i najbardziej efektywnych narzędzi Excela są *tabele przestawne*, służące do tworzenia zestawień danych w postaci poręcznych i wygodnych w użyciu tabel. Co ważne, tabele przestawne można tworzyć i modyfikować za pomocą języka VBA. Dane dla tabeli przestawnej pochodzą z arkuszowej lub zewnętrznej bazy danych i są umieszczane w specjalnej pamięci podręcznej umożliwiającej wykonywanie obliczeń natychmiast po zmodyfikowaniu zawartości tabeli przestawnej. Na rysunku 1.22 pokazano przykładowe tabele przestawne sformatowane jako raport.

The screenshot shows four pivot tables in an Excel spreadsheet:

- Lokalizacje magazynów są dogodne**: A pivot table showing the count of responses for five statements. It includes a summary row for the total number of respondents (100).
- Godziny otwarcia są dogodne**: A pivot table showing the count of responses for five statements. It includes a summary row for the total number of respondents (100).
- Magazyny są dobrze utrzymane**: A pivot table showing the count of responses for five statements. It includes a summary row for the total number of respondents (100).
- Lokalizacje magazynów są dogodne**: A second instance of the first pivot table, showing percentages instead of counts. It includes a summary row for the total number of respondents (100).

Below the tables, there are tabs labeled "Podsumowanie" (Summary), "DaneZAnkiety" (Data from Survey), and "+".

Rysunek 1.22. Tabele przestawne Excela mają wiele zastosowań



Więcej szczegółowych informacji na temat tworzenia i modyfikowania tabel przestawnych znajdziesz w rozdziale 15.

- **Dodatek Solver** — Do rozwiązywania specjalizowanych równań liniowych i nieliniowych służy dodatek Solver Excela, który w oparciu o szereg wartości wejściowych pozwala rozpatrywać scenariusze warunkowe oraz dokonywać optymalizacji, czyli poszukiwać wartości argumentów, dla których wartość funkcji celu jest największa lub najmniejsza.

Dodatki

Dodatek (ang. *add-in*) jest programem dołączanym do aplikacji w celu poszerzenia zakresu jej funkcjonalności. Aby zainstalować wybrany dodatek Excela, powinieneś użyć karty *Dodatki* okna dialogowego *Opcje programu Excel*.

Oprócz dodatków dołączonych do programu Excel, możesz używać dodatków pobranych ze strony internetowej firmy Microsoft (<http://office.microsoft.com>). Dodatki opracowane przez inne firmy można kupić lub pobrać za pośrednictwem ich serwisów internetowych, a poza tym, stworzenie własnego dodatku jest *bardzo* proste (więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 19.).

Makra i programowanie

Excel posiada dwa wbudowane języki programowania makr — XLM i VBA. Używany początkowo język programowania makr XLM jest już przestarzały i został zastąpiony przez język VBA. Excel 2013 nadal jest w stanie poprawnie wykonać większość makr XLM, a nawet tworzyć nowe makra XLM, ale nie możesz ich już rejestrować. Tworząc nowe makra, powinieneś używać języka VBA.

Zgodność formatu plików

Zgodność plików skoroszytów zapisywanych w różnych wersjach Excela jest zagadniением kluczowym. Wersje od Excela 97 do Excela 2003 używają takiego samego formatu plików, dzięki czemu w ich przypadku zgodność plików nie stanowi problemu. W wersji Excel 2007 Microsoft wprowadził jednak zupełnie nowy format zapisu plików skoroszytów, który od tej pory jest używany we wszystkich nowych wersjach Excela. Na szczęście posiadaczom poprzednich wersji Excela firma Microsoft udostępnia *Pakiet zgodności formatu plików pakietu Microsoft Office*, który pozwala użytkownikom Excela XP i Excela 2003 na odczytywanie i zapisywanie plików w nowym formacie.

Pamiętaj jednak, że istnieje poważna różnica pomiędzy zgodnością plików i zgodnością funkcji. Na przykład: pomimo iż pakiet zgodności pozwoli w Excelu 2003 otworzyć pliki utworzone przy użyciu Excela 2013, to mimo to Excel 2003 nie będzie w stanie obsługiwać wielu nowych funkcji, które zostały wprowadzone w nowszej wersji Excela.



Więcej szczegółowych informacji na temat formatów plików programu Excel znajdziesz w rozdziale 3. Więcej szczegółowych informacji na temat kompatybilności różnych wersji Excela znajdziesz w rozdziale 24.

System pomocy Excela

Jedną z najważniejszych funkcji Excela jest jego system pomocy (patrz rysunek 1.23). Gdy nie wiesz, co dalej zrobić, kliknij ikonę ze znakiem zapytania, znajdującą się po prawej stronie paska tytułowego okna skoroszytu (zamiast tego możesz po prostu nacisnąć klawisz *F1*). Na ekranie pojawi się okno dialogowe systemu pomocy programu Excel, gdzie będziesz mógł wyszukać tematy pomocy według odpowiednich słów kluczowych lub po prostu przeglądać tematy pomocy w spisie treści.

Excel — Pomoc

Odwołanie cykliczne

Znajdowanie i naprawianie odwołania cyklicznego

Po wprowadzeniu formuły nic nie działa. Zamiast tego pojawia się komunikat o „odwołaniu cyklicznym”. Mność ludzi ma ten sam problem, który wynika z tego, że formuła próbuje obliczyć własną komórkę, a funkcja o nazwie obliczanie iteracyjne jest wyłączona. Zobacz, jak wygląda takie odwołanie:

	A	B	C	D	E
1				3481	
2				4129	
3				=D1+D2+D3	
4					
5					

Formuła =D1+D2+D3 nie działa, ponieważ jest zdefiniowana w komórce D3 i próbuje obliczyć samą siebie.

Aby naprawić problem, możesz przenieść formułę do innej komórki (na pasku formuły naciśnij klawisze *Ctrl+C*, aby wyciąć formułę, zaznacz inną komórkę i naciśnij klawisz *Ctrl+V*). Możesz również wypróbować jedną z następujących technik:

- Jeśli formuła została właśnie wprowadzona, najpierw sprawdź, czy nie występuje odwołanie do komórki z definicją tej formuły. Na przykład komórka A3 może zawierać formułę = (A1+A2)/A3. Do typowych błędów należy też użycie funkcji **SUMA** z odwołaniem do samej

Rysunek 1.23. Okno pomocy systemowej programu Excel

Rozdział 2.

Wybrane zasady stosowania formuł

W tym rozdziale:

- Formuły i ich obliczanie
- Zastosowanie odwołań względnych i bezwzględnych do komórki lub zakresu w formułach
- Zastosowanie nazw
- Formuły tablicowe
- Zliczanie i sumowanie wartości w komórkach
- Używanie daty i czasu
- Tworzenie megaformuł

Formuły

Prawie każda użyteczna aplikacja oparta na arkuszu kalkulacyjnym korzysta z formuł. W praktyce tworzenie formuł może być traktowane jako swego rodzaju odmiana programowania. W tym rozdziale omówimy niektóre najbardziej popularne (i nie tylko) rodzaje formuł.



Więcej szczegółowych informacji na temat formuł i funkcji Excela znajdziesz w innej mojej książce, *Excel 2013 PL. Formuły* (wyd. Helion).

Arkusz kalkulacyjny jest tym, czym jest, głównie dzięki formułom. Bez formuł byłby tylko statycznym dokumentem, który równie dobrze można by utworzyć przy użyciu edytora tekstu obsługującego tabele.

Excel oferuje ogromną liczbę wbudowanych funkcji, znakomicie radzi sobie z obsługą zakresów mających zdefiniowane nazwy, a nawet wspiera stosowanie tzw. *formuł tablicowych* (ang. *array formulas*), czyli specjalnego typu formuł, które mogą wykonywać obliczenia na całych zakresach danych.

Formuła wprowadzona do komórki może się składać z następujących elementów:

- Operatorów takich jak znak + (dodawanie) czy * (mnożenie)
- Odwołań do komórek (w tym komórek i zakresów posiadających nazwy)
- Liczb lub łańcuchów tekstowych
- Funkcji arkusza (takich jak na przykład SUMA czy ŚREDNIA)

Formuła może się składać maksymalnie z 8192 znaków. Po zakończeniu wprowadzania formuły, w komórce pojawia się wynik jej działania, a sama formuła pojawia się na pasku formuły po uaktywnieniu komórki. Aby polepszyć sposób wyświetlania długich formuł, możesz zwiększyć wysokość paska formuły, „łapiąc” dolną krawędź paska lewym przyciskiem myszy i przeciągając w dół tak, aby pasek formuły przybrał żądane rozmiary. Zamiast tego możesz po prostu kliknąć przycisk ze strzałką skierowaną w dół, znajdujący się po prawej stronie paska formuły.

Obliczanie formuł

Prawdopodobnie zauważyłeś już, że formuły w Twoim arkuszu są automatycznie przeliczane zaraz po ich wprowadzeniu do komórki. Jeżeli zmienisz zawartość dowolnej komórki wykorzystywanej przez daną formułę, to wynik działania formuły zostanie natychmiast zaktualizowany, bez żadnych dodatkowych działań z Twojej strony. Dzieje się tak wtedy, kiedy Excel pracuje w automatycznym trybie przeliczania formuł. Jest to domyślny tryb pracy programu, w którym Excel podczas przeliczania formuł korzysta z następujących reguł:

- Po dokonaniu zmiany, na przykład po wprowadzeniu lub edycji danych bądź formuły, Excel natychmiast przelicza formuły, które są powiązane z nowymi lub zmodyfikowanymi danymi.
- Jeżeli modyfikacja arkusza ma miejsce w trakcie wykonywania złożonych obliczeń, Excel wstrzymuje obliczenia na czas realizacji zadań wynikających z modyfikacji; po zakończeniu modyfikacji Excel kontynuuje wykonywanie obliczeń.
- Formuły wykonywane są w porządku naturalnym. W praktyce oznacza to, że jeżeli formuła zawarta w komórce D12 zależy od wyniku formuły zdefiniowanej w komórce D11, komórka D11 zostanie przetworzona przed komórką D12.

Zdarzają się jednak sytuacje, kiedy będziesz chciał kontrolować proces obliczania formuł. Na przykład w arkuszu zawierającym tysiące złożonych formuł proces przeliczania arkusza może trwać naprawdę sporo czasu. W takim przypadku możesz przełączyć Excela w tryb ręcznego obliczania. Aby to zrobić, przejdź na kartę *FORMUŁY*, naciśnij przycisk *Opcje obliczania*, znajdujący się w grupie opcji *Obliczanie*, i następnie z menu podręcznego wybierz odpowiednią opcję.

Jeżeli w trybie przeliczania ręcznego Excel napotka w arkuszu jakiekolwiek formuły, które nie zostały jeszcze przeliczone, na pasku stanu pojawi się komunikat *Oblicza*. Aby ponownie przeliczyć formuły, możesz nacisnąć jedną z podanych niżej kombinacji klawiszy:

- Klawisz **F9** powoduje obliczenie formuł we wszystkich otwartych skoroszytach.
- Kombinacja klawiszy **Shift+F9** powoduje obliczenie formuł zawartych tylko w aktywnym arkuszu — inne arkusze z tego samego skoroszytu nie zostaną przeliczone.
- Kombinacja klawiszy **Ctrl+Alt+F9** wymusza ponowne obliczenie wszystkich formuł we wszystkich otwartych skoroszytach. Użyj tej kombinacji w sytuacji, kiedy z jakiegoś powodu Excel niepoprawnie wykonuje obliczenia lub gdy chcesz wymusić ponowne obliczenie formuł korzystających z niestandardowych funkcji stworzonych przy użyciu języka VBA.
- Kombinacja klawiszy **Ctrl+Alt+Shift+F9** powoduje ponowne sprawdzenie wszystkich zależności pomiędzy formułami i przelicza wartości wszystkich komórek arkusza we wszystkich skoroszytach (w tym i komórek, które nie zostały oznaczone jako wymagające przeliczenia).



Tryb obliczania, w jakim pracuje Excel, nie jest powiązany z określonym arkuszem. Jeżeli dokonasz zmiany trybu obliczania, będzie to miało wpływ na sposób przeliczania wszystkich otwartych skoroszytów, a nie tylko aktywnego.

Odwołania do komórki lub zakresu

Większość formuł odwołuje się do jednej lub kilku komórek. Odwołanie takie może być wykonane przy użyciu adresu lub nazwy komórki bądź zakresu (o ile taka nazwa została zdefiniowana). Możemy wyróżnić cztery typy odwołań do komórek:

- **Względne** — tego typu odwołanie jest całkowicie względne. Po skopiowaniu formuły odwołanie do komórki zostanie dopasowane do jej nowej lokalizacji. Przykład odwołania — A1.
- **Bezwzględne** — tego typu odwołanie jest całkowicie bezwzględne. Po skopiowaniu formuły odwołanie do komórki nie ulegnie zmianie. Przykład odwołania — \$A\$1.
- **Bezwzględne do wiersza** — tego typu odwołanie jest częściowo bezwzględne (mieszane). Po skopiowaniu formuły część kolumnowa zostanie dopasowana, natomiast część wierszowa nie ulegnie zmianie. Przykład odwołania — A\$1.
- **Bezwzględne do kolumny** — tego typu odwołanie jest częściowo bezwzględne (mieszane). Po skopiowaniu formuły część wierszowa zostanie dopasowana, natomiast część kolumnowa nie ulegnie zmianie. Przykład odwołania — \$A1.

Domyślnie wszystkie odwołania do komórek i zakresów są względne. Aby zmienić typ odwołania, konieczne jest ręczne dodanie symbolu dolara (\$). Innym sposobem zmiany rodzaju odwołania jest umieszczenie w trakcie edytowania formuły kurSORA przy adresie komórki i kolejne naciśnięcie klawisza F4 do momentu wybrania żądanego rodzaju odwołania.

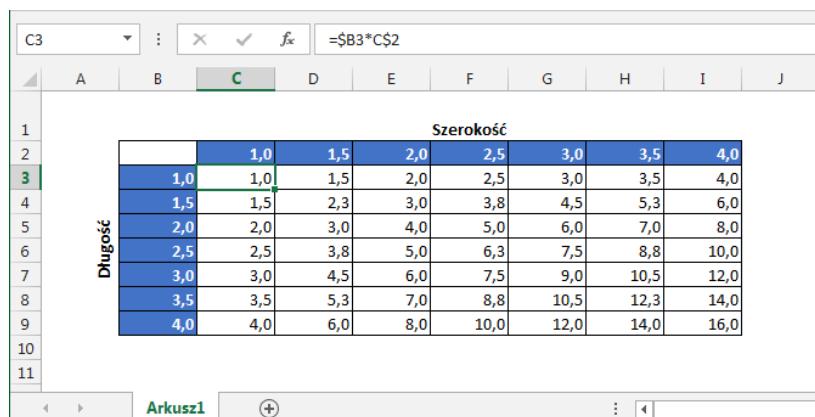
Dlaczego warto używać odwołań, które nie są względne?

Jeżeli się nad tym dobrze zastanowić, to jedynym powodem, dla którego w ogóle warto zmieniać typ odwołania, jest zamiar skopiowania formuły. Na rysunku 2.1 pokazano, dlaczego tak jest. Formuła zawarta w komórce C3 ma następującą postać:

$=\$B3*C\2

Rysunek 2.1.

Przykład zastosowania w formule odwołań mieszanych



The screenshot shows an Excel spreadsheet with a table of rectangles. The table has 'Szerokość' (Width) in row 1 and 'Długość' (Length) in column 1. The data starts from cell B3, which contains the formula $=\$B3*C\2 . The table is as follows:

		Szerokość						
		1,0	1,5	2,0	2,5	3,0	3,5	4,0
Długość	1,0	1,0	1,5	2,0	2,5	3,0	3,5	4,0
	1,5	1,5	2,3	3,0	3,8	4,5	5,3	6,0
	2,0	2,0	3,0	4,0	5,0	6,0	7,0	8,0
	2,5	2,5	3,8	5,0	6,3	7,5	8,8	10,0
	3,0	3,0	4,5	6,0	7,5	9,0	10,5	12,0
	3,5	3,5	5,3	7,0	8,8	10,5	12,3	14,0
	4,0	4,0	6,0	8,0	10,0	12,0	14,0	16,0

Powyższa formuła oblicza powierzchnie prostokątów o różnej szerokości (wartości podane w kolumnie B) i długości (wartości podane w wierszu 2). Raz opracowaną formułę można skopiować do wszystkich komórek obszaru ograniczonego przez komórki C7 i F7. Ze względu na to, że formuła w stosunku do wiersza 2 i kolumny B używa odwołań bezwzględnych, a dla pozostałych wierszy i kolumn używa odwołań względnych, każda skopiowana formuła generuje poprawny wynik. Jeśli formuła używałaby jedynie odwołań względnych, jej skopiowanie spowodowałoby zmodyfikowanie wszystkich odwołań, a tym samym uzyskano by nieprawidłowe rezultaty.

Notacja W1K1

Standardowo Excel przy adresacji komórek posługuje się tzw. *notacją A1*, w przypadku której adres każdej komórki składa się z litery kolumny i numeru wiersza. Excel obsługuje również tzw. *notację W1K1* (ang. *R1C1 notation*), gdzie komórka A1 jest identyfikowana jako komórka W1K1, komórka A2 jako W2K1 itd.

Aby przejść na notację *W1K1*, przejdź na kartę *PLIK* i z menu wybierz polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Na liście w lewej części kliknij kategorię *Formuły* i w prawej części okna, w grupie *Praca z formułami* zaznacz opcję *Styl odwołania W1K1*. Po wykonaniu tej operacji litery w nagłówkach kolumn zmienią się na liczby, a wszystkie odwołania do komórek i zakresów zawarte w formułach zostaną odpowiednio dostosowane.

W tabeli 2.1 przedstawiamy kilka przykładów formuł używających notacji standardowej i notacji W1K1. Przyjęto, że formuła znajdzie się w komórce B1 (lub W1K2).

Tabela 2.1. Porównanie prostych formuł zapisanych przy użyciu dwóch notacji

Notacja standardowa	Notacja W1K1
=A1+1	=WK[-1]+1
=\$A\$1+1	=W1K1+1
=\$A1+1	=WK1+1
=A\$1+1	=W1K[-1]+1
=SUMA(A1:A10)	=SUMA(WK[-1]:W[9]K[-1])
=SUMA(\$A\$1:\$A\$10)	=SUMA(W1K1:W10K1)

Jeżeli uznasz, że notacja W1K1 jest niezbyt przyjazna, to nie będziesz w tym poglądzie osamotniony. Trzeba jednak przyznać, że w przypadku stosowania odwołań bezwzględnych nie jest ona taka zła. Jeżeli jednak używamy odwołań względnych, nawiasy kwadratowe mogą nas niekiedy doprowadzić do szewskiej pasji.

Liczby w nawiasach kwadratowych odnoszą się do względnego położenia odwołań. Przykładowo zapis W[-5]K[-3] identyfikuje komórkę znajdującą się pięć wierszy powyżej i trzy kolumny na lewo. Z kolei zapis W[5]K[3] odnosi się do komórki położonej pięć wierszy poniżej i trzy kolumny na prawo. Jeżeli nawiasy kwadratowe zostaną pominięte, notacja dotyczyć będzie tego samego wiersza lub kolumny. Przykładowo zapis W[5]K odnosi się do komórki położonej pięć wierszy poniżej w tej samej kolumnie.

Możemy śmiało założyć, że nie będziesz na co dzień korzystał z notacji W1K1. Jeżeli jednak tworzysz w języku VBA kod generujący formuły arkusza, może się okazać, że takie zadanie będzie łatwiejsze po zastosowaniu notacji W1K1.

Odwolania do innych arkuszy lub skoroszytów

Odwolania do komórek i zakresów nie muszą dotyczyć arkusza, w którym znajduje się formuła. Aby odwołać się do komórki znajdującej się w innym arkuszu, powinieneś poprzedzić odwołanie nazwą arkusza i znakiem wykrzyknika. Oto przykład formuły używającej odwołania do komórki innego arkusza:

=Arkusz2!A1+1

Można również tworzyć formuły odwołujące się do komórek znajdujących się w innym skoroszycie. Aby to zrobić, powinieneś odwołanie poprzedzić nazwą skoroszytu (podaną w nawiasach kwadratowych), nazwą arkusza i znakiem wykrzyknika, na przykład:

=[Budżet.xls]Arkusz1!A1

Jeśli nazwa skoroszytu, do którego odwołuje się formuła, zawiera jedną lub kilka spacji, musisz umieścić je (wraz z nazwą arkusza) w apostrofach, na przykład:

='[Budżet na rok 2013.xls]Arkusz1'!A1

Jeśli skoroszyt, do którego chcesz utworzyć odwołanie jest zamknięty, musisz w odwołaniu wpisać jego pełną ścieżkę, na przykład:

='C:\Budżet\Dokumenty Excel\[Budżet na rok 2013.xls]Arkusz1'!A1

Odwołania do danych w tabelach

Począwszy od wersji 2007, w Excelu możesz definiować specjalny rodzaj zakresów komórek, spełniający rolę tabeli danych (aby zdefiniować tabelę, przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Tabela* znajdujący się w grupie opcji *Tabele*). Wprowadzenie tabel spowodowało rozszerzenie funkcjonalności formuł o kilka nowych funkcji.

Kiedy wpisujesz formułę do komórki w tabeli, Excel automatycznie kopiuje formułę do wszystkich pozostałych komórek w tej kolumnie (ale tylko wtedy, gdy kolumna była pusta). Taka kolumna jest nazywana kolumną obliczeniową (ang. *calculated column*). Jeżeli dodajesz do tabeli nowy wiersz, w kolumnie obliczeniowej automatycznie dodawane są odpowiednie formuły — w zdecydowanej większości przypadków to jest właśnie to, o co Ci chodziło. Jeżeli nie chcesz, aby Excel automatycznie wprowadzał za Ciebie formuły, możesz wyłączyć ten mechanizm za pomocą taga (ang. *SmartTag*), który pojawi się obok wstawionego wiersza.

Podczas pracy z tabelami Excel pozwala również na odwoływanie się do komórek tabeli za pomocą tzw. *odwołań strukturalnych* (ang. *structured referencing*). Aby zilustrować zasadę tworzenia takich odwołań, przyjmijmy, że tabela przedstawiona na rysunku poniżej nosi nazwę *Tabela1*.

	A	B	C	D	E	F
1	Miesiąc	Stan	Przychody	Rozchody		
2	Styczeń	Waszyngton	983	462		
3	Luty	Waszyngton	1022	549		
4	Marzec	Waszyngton	861	503		
5	Styczeń	Oregon	764	398		
6	Luty	Oregon	993	425		
7	Marzec	Oregon	882	387		
8	Suma		0	5505	2724	
9						
10						
11						

The screenshot shows a Microsoft Excel spreadsheet with a table named "Tabela1". The table has columns labeled A through F. Rows 1 and 2 serve as headers. Rows 3 through 7 contain data for Washington (Waszyngton) in January, February, and March respectively. Rows 8 through 11 contain data for Oregon (Oregon) in January, February, and March respectively. Row 12 is a summary row labeled "Suma" (Sum) with values 0, 5505, and 2724 in columns A, C, and E respectively. The table is located on "Arkusz1" (Sheet1). The status bar at the bottom shows "Arkusz2" and a small green icon.

W przypadku tabeli możesz tworzyć odwołania do jej komórek poprzez odpowiednie użycie nagłówków kolumn i wierszy. W wielu sytuacjach takie rozwiązanie może powodować, że formuły przetwarzające dane z tabeli będą łatwiejsze do zrozumienia i przeanalizowania. Jednak największą zaletą stosowania tego typu odwołań jest to, że po dodaniu lub usunięciu wierszy z tabeli wszystkie formuły nadal będą poprawne. Poniżej przedstawiamy kilka przykładów poprawnych formuł wykorzystujących odwołania strukturalne:

```
=Tabela1[[#Sumy].[Przychody]]
=SUMA(Tabela1[Przychody])
=Tabela1[[#Sumy].[Przychody]]-Tabela1[[#Sumy].[Rozchody]]
=SUMA(Tabela1[Przychody])-SUMA(Tabela1[Rozchody])
=SUMA.JEŻELI(Tabela1[Stan], "Oregon", Tabela1[Przychody])
=Tabela1[@Rozchody]
```

Ostatnia formuła, wykorzystująca symbol @, który oznacza „ten wiersz”, będzie poprawna tylko wtedy, kiedy zostanie umieszczona w komórce znajdującej się w obszarze tabeli.

Co prawda odwołania formuł łączy mogą być wprowadzane bezpośrednio, ale można też definiować je przy użyciu standardowych metod wskazywania. Aby to zrobić, musisz najpierw otworzyć plik źródłowy. Po jego otwarciu Excel utworzy bezwzględne odwołania do komórek. Aby skopiować formułę do innych komórek, powinieneś zamienić je na odwołania względne.



Przy korzystaniu z łączów powinieneś zachować ostrożność. Jeżeli na przykład w celu wykonania kopii zapasowej arkusza źródłowego przejdziesz na kartę *PLIK* i wybierzesz polecenie *Zapisz jako*, formuły łączów zostaną automatycznie zmodyfikowane, tak aby odwoływały się do nowego pliku (a zazwyczaj nie o to chodzi). Kolejna sytuacja, w której dochodzi do uszkodzenia łącz, ma miejsce wtedy, gdy nazwa skoroszytu źródłowego została zmieniona, ale powiązany z nim skoroszyt nie został wcześniej otwarty.

Zastosowanie nazw

Jednym z najprzydatniejszych mechanizmów Excela jest możliwość nadawania wybranych nazw różnym elementom arkusza, takim jak komórki, zakresy komórek, kolumny, wykresy i inne obiekty. Unikalną zaletą Excela jest możliwość nadawania nazw nawet takim wartościom lub formułom, które nie znajdują się w komórkach arkusza (patrz podpunkt „Nadawanie nazw stałym” w dalszej części tego rozdziału).

Nadawanie nazw komórkom i zakresom

Excel pozwala nadawać nazwy komórkom i zakresom na kilka różnych sposobów:

- Użyj okna dialogowego *Nowa nazwa*. Aby to zrobić, przejdź na kartę *FORMUŁY* i naciśnij przycisk *Definiuj nazwę* znajdujący się w grupie opcji *Nazwy zdefiniowane*.
- Użyj okna dialogowego *Menedżer nazw*. Aby to zrobić, przejdź na kartę *FORMUŁY* i naciśnij przycisk *Menedżer nazw* znajdujący się w grupie opcji *Nazwy zdefiniowane* (zamiast tego możesz po prostu nacisnąć kombinację klawiszy *Ctrl+F3*). Niestety nie jest to najbardziej efektywna metoda nadawania nazw, ponieważ utworzenie nowej nazwy wymaga dodatkowo naciśnięcia w oknie *Menedżer nazw* przycisku *Nowy*, co powoduje wyświetlenie na ekranie okna dialogowego *Nowa nazwa* (patrz poprzedni podpunkt).
- Zaznacz wybraną komórkę lub zakres komórek, następnie wpisz żądaną nazwę bezpośrednio w polu *Pole nazwy* i naciśnij klawisz *Enter*. *Pole nazwy* to pole listy rozwijanej znajdujące się po lewej stronie paska formuły.
- Jeżeli na arkuszu znajdują się elementy tekstowe, których chciałbyś użyć jako nazw dla sąsiadujących komórek lub zakresów komórek, zaznacz wybrane elementy oraz komórkę, przejdź na kartę *FORMUŁY* i naciśnij przycisk *Utwórz z zaznaczenia* znajdujący się w grupie opcji *Nazwy zdefiniowane*. Na rysunku 2.2 zakresowi B3:E3 nadano nazwę *Północ*, zakresowi B4:E4 nazwę *Południe* itd. W pionie zakresowi B3:B6 nadano nazwę *Kwartal_1*, zakresowi C3:C6 nazwę *Kwartal_2* itd. Zwróć uwagę na fakt, że Excel automatycznie zmodyfikował nazwy zakresów tak, aby utworzone nazwy były zgodne z zasadami tworzenia nazw (znak minus nie może być używany w nazwach zakresów).

Rysunek 2.2.

Excel ułatwia tworzenie nazw będących opisowymi łańcuchami tekstowymi

A	B	C	D	E	F	G	H	I
1								
2		Kwartal-1	Kwartal-2	Kwartal-3	Kwartal-4			
3	Północ	8973	9442	9424	9393			
4	Południe	9321	9186	8734	8595			
5	Wschód	9872	9904	10393	10978			
6	Zachód	5992	6229	6445	6287			
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								

Zastosowanie nazw jest szczególnie istotne przy pisaniu w języku VBA kodu używającego odwołań do komórek lub zakresów. Jaki jest tego powód? Po przeniesieniu komórki lub zakresu, do których odwoływano się w kodzie VBA, w inne miejsce arkusza, odwołania nie są automatycznie aktualniane. Na przykład kiedy kod języka VBA zapisuje daną wartość w komórce Range("C4"), dane zostaną umieszczone w złej komórce, gdy użytkownik wstawi nad komórką C4 nowy wiersz lub kolumnę po jej lewej stronie. Zastosowanie odwołania do nazwy komórki (np. Range("Oprocenowanie")) pozwoli na uniknięcie takich potencjalnych problemów.

Ukryte nazwy

Niektóre makra Excela i dodatki tworzą tzw. *nazwy ukryte* (ang. *hidden names*), które znajdują się w skoroszycie, ale nie są widoczne w oknie dialogowym *Definiowanie nazw*. Na przykład, dodatek Solver tworzy kilka takich ukrytych nazw. Zazwyczaj nazwy ukryte powinny być ignorowane, jednak czasami mogą być przyczyną problemów. Jeżeli skopiujesz dany arkusz do innego skoroszytu, ukryte nazwy również zostaną skopiowane, co w efekcie może przyczynić się do utworzenia trudnego do wykrycia łącza pomiędzy arkuszami.

Aby usunąć ze skoroszytu wszystkie ukryte nazwy, możesz posłużyć się poniższą procedurą języka VBA:

```
Sub DeleteHiddenNames()
    Dim n As Name
    Dim Count As Integer
    For Each n In ActiveWorkbook.Names
        If Not n.Visible Then
            n.Delete
            Count = Count + 1
        End If
    Next n
    MsgBox Count & " ukrytych nazw zostało usuniętych."
End Sub
```

Nadawanie nazw istniejącym odwołaniom

Pomimo utworzenia nazwy dla wybranej komórki lub zakresu komórek Excel nie będzie automatycznie używał nowej nazwy w miejsce odwołań znajdujących się w formułach. Dla przykładu założymy, że w komórce F10 wstawiono następującą formułę:

=A1-A2

Jeżeli dla komórki A1 zdefiniujesz nazwę *Przychód*, a dla komórki A2 nazwę *Koszty*, Excel nie dokona automatycznej zmiany formuły do postaci:

=Przychód-Koszty

Jednak zastąpienie odwołań do komórek lub zakresów powiązanymi z nimi nazwami jest zadaniem relatywnie prostym. Aby to zrobić, powinieneś najpierw zaznaczyć zakres, który zostanie zmodyfikowany. Następnie przejdź na kartę *FORMUŁY*, naciśnij strzałkę obok przycisku *Definiuj nazwę* znajdującego się w grupie opcji *Nazwy zdefiniowane* i wreszcie z menu podręcznego, które pojawi się na ekranie, wybierz polecenie *Zastosuj nazwy*. Na ekranie pojawi się okno dialogowe *Słosowanie nazw*. Zaznacz nazwy, które powinny zostać użyte, i naciśnij przycisk *OK*. W zaznaczonym wcześniej zakresie komórek Excel zastąpi odwołania do komórek ich nazwami.



Uwaga

Niestety nie istnieje metoda automatycznego wycofywania nazw. Innymi słowy, jeśli formuła używa nazwy, nie można automatycznie zamienić jej na odpowiadające jej odwołanie do komórki lub zakresu. Co gorsza, jeżeli nazwa zastosowana w formule zostanie usunięta, Excel nie będzie w stanie odtworzyć adresu komórki lub zakresu powiązanego z nazwą i zamiast tego wyświetli po prostu błąd #NAZWA?.

W skład mojego dodatku *Power Utility Pak* wchodzi narzędzie, które przegląda wszystkie formuły w zaznaczonym obszarze i automatycznie zastępuje nazwy komórek i zakresów odpowiednimi adresami.

Słosowanie nazw z operatorem przecięcia

Excel posiada specjalny operator nazywany **operatorem przecięcia** (ang. *intersection operator*), używany do przetwarzania zakresów. Symbolem operatora jest znak spacji. Odpowiednie użycie nazw wraz z operatorem przecięcia znacznie upraszcza tworzenie przejrzystych formuł. Zastosowanie takiego rozwiązania możemy zilustrować na przykładzie rysunku 2.2. Po wprowadzeniu do komórki następującej formuły:

=Kwartał_2 Południe

otrzymasz wynik 9186, będący rezultatem przecięcia zakresów *Kwartał_2* i *Południe*.

Nadawanie nazw kolumnom i wierszom

Excel umożliwia też nadawanie nazw całym wierszom i kolumnom. W poprzednim przykładzie nazwa *Kwartał_1* została nadana zakresowi B3:B6. Alternatywnie nazwa *Kwartał_1* mogłaby zostać przypisana całej kolumnie B, nazwa *Kwartał_2* kolumnie C itd. Tak samo można postąpić w przypadku wierszy. Wierszowi 3 można nadać nazwę *Północ*, wierszowi 4 nazwę *Południe* itd.

Operator przecięcia działa dokładnie tak samo, jak w poprzednim przykładzie, ale teraz możesz dodać więcej regionów lub kwartałów bez konieczności modyfikowania istniejących nazw.

Nadając nazwy kolumnom i wierszom, powinieneś pamiętać, aby nie umieszczać w nich żadnych niepotrzebnych informacji. Pamiętaj, że na przykład po wstawieniu określonej wartości w komórce C7 zostanie ona uwzględniona w danych zakresu *Kwartał_1*.

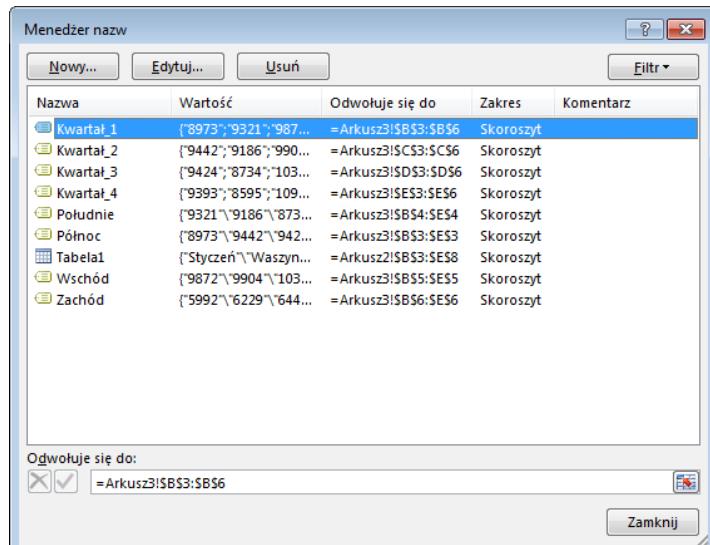
Zasięg nazw zakresów

Nazwana komórka lub zakres zwykle mają *zasięg* obejmujący cały skoroszyt. Innymi słowy, danej nazwy możesz użyć w dowolnym arkuszu skoroszytu.

Innym rozwiązaniem jest utworzenie nazw obowiązujących tylko na obszarze wybranego arkusza. Aby to zrobić, powinieneś poprzedzić definiowaną nazwę komórki lub zakresu nazwą arkusza, po której następuje znak wykryznika, na przykład Arkusz1!Sprzedaż. Jeżeli taka nazwa będzie użyta w arkuszu, w którym została zdefiniowana, to przy odwoływaniu się do niej możesz pominąć identyfikator arkusza. Jeżeli chcesz odwołać się do takiej nazwy z poziomu innego arkusza, musisz jednak poprzedzić ją kwalifikatorem arkusza, tak jak to opisaliśmy powyżej.

Okno dialogowe *Menedżer nazw* pozwala na łatwą identyfikację utworzonych nazw i ich zasięgu (patrz rysunek 2.3). Aby przywołać to okno na ekran, przejdź na kartę *FORMUŁY* i naciśnij przycisk *Menedżer nazw* znajdujący się w grupie opcji *Nazwy zdefiniowane*. Zauważ, że możesz zmieniać zarówno rozmiary okna jak i poszczególnych kolumn. Listy nazw możesz sortować według różnych kryteriów; aby posortować nazwy na przykład według ich zasięgu, kliknij nagłówek kolumny *Zakres*.

Rysunek 2.3.
W oknie Menedżera nazw wyświetlany jest zasięg poszczególnych nazw komórek i zakresów



Nadawanie nazw stałym

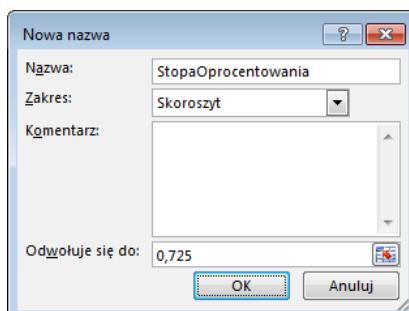
Niemal każdy doświadczony użytkownik Excela wie, w jaki sposób utworzyć nazwy dla komórek lub zakresów (choćż nie wszyscy korzystają z tej możliwości). Jednak większość użytkowników Excela nie zdaje sobie sprawy z tego, że za pomocą nazw można odwoływać się do wartości, które nie są umieszczone na arkuszu — oczywiście chodzi tutaj o *stale*.

Załóżmy, że wiele formuł arkusza wymaga zastosowania określonej stopy procentowej. Jednym z możliwych rozwiązań jest wprowadzenie tej wartości do wybranej komórki i nadanie jej nazwy np. *StopaOprocentowania*, dzięki czemu będziemy mogli użyć jej w formułach, na przykład:

=StopaOprocentowania*A3

Innym rozwiązaniem jest przywołanie na ekran okna dialogowego *Nowa nazwa* (aby to zrobić, przejdź na kartę *Formuły* i naciśnij przycisk *Definiuj nazwę*, znajdujący się w grupie opcji *Nazwy zdefiniowane*) i wprowadzenie wartości stopy procentowej bezpośrednio w polu *Odwołuje się do* (patrz rysunek 2.4). Po wykonaniu tej operacji w formułach można używać nazwy, tak jakby powiązana z nią wartość była przechowywana w komórce. Jeśli wartość stopy oprocentowania ulegnie zmianie, wystarczy odpowiednio zmodyfikować definicję nazwy *StopaOprocentowania*, a Excel automatycznie dokona aktualizacji wszystkich komórek zawierających tę nazwę.

Rysunek 2.4.
Excel umożliwia nadawanie nazw stałym, które nie znajdują się w komórkach arkusza



Opisana powyżej metoda sprawdza się też w przypadku tekstu. Przykładowo można zdefiniować nazwę HLN, odpowiadającą łańcuchowi tekstu Grupa Wydawnicza Helion, a następnie wpisać w komórce formułę =HLN, co spowoduje wyświetlenie w niej pełnej nazwy.

Nadawanie nazw formułom

Nazwy możesz nadawać nie tylko komókom, zakresom i stałym, ale też formułom. Powinieneś pamiętać, że nazwana formuła, tak jak to zostało opisane poniżej, nie jest zapisana w komórce arkusza, ale istnieje tylko i wyłącznie wirtualnie w pamięci Excela. Aby utworzyć nazwaną formułę, powinieneś wpisać ją bezpośrednio w polu *Odwołuje się do* okna dialogowego *Definiowanie nazw*.

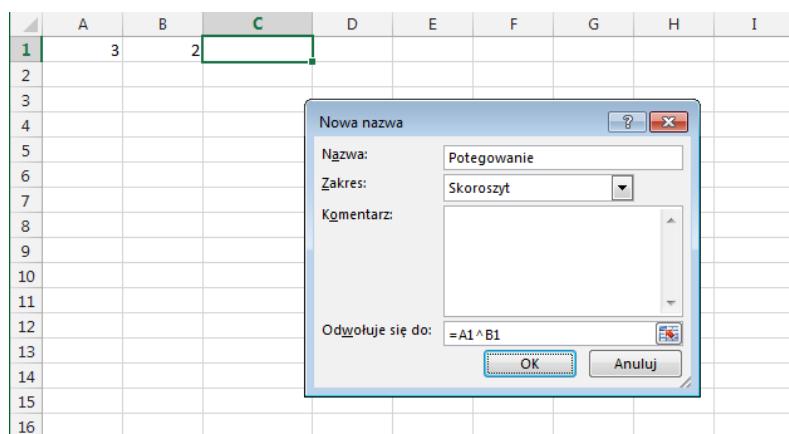


To bardzo ważne: formuła utworzona w opisany powyżej sposób tworzy odwołania do innych komórek względem *komórki aktywnej* w momencie tworzenia tej formuły.

Na rysunku 2.5 przedstawiono formułę $=A1^B1$, która została bezpośrednio wprowadzona w polu *Odwołuje się do* okna dialogowego *Nowa nazwa*. W naszym przypadku komórką aktywną była komórka C1, stąd formuła odwołuje się do dwóch komórek znajdujących się z jej lewej strony (zwróć uwagę na fakt, że odwołania do komórek są względne). Po zdefiniowaniu tej nazwy wprowadzenie formuły $=\text{Potegowanie}$ do wybranej komórki spowoduje podniesienie wartości komórki znajdującej się dwie komórki na lewo od niej do potęgi reprezentowanej przez wartość komórki znajdującej się bezpośrednio z jej lewej strony. Jeśli na przykład komórka B10 zawiera wartość 3, a komórka C10 wartość 4, to wprowadzenie formuły przedstawionej poniżej do komórki D10 spowoduje zwrócenie wartości 81 (3 podniesione do 4 potęgi).

$=\text{Potegowanie}$

Rysunek 2.5.
Definiowanie nazwy formuły, która nie występuje w żadnej komórce arkusza



Jeżeli po utworzeniu nazwanej formuły przywołasz na ekran okno dialogowe *Menedżer nazw*, w kolumnie *Odwołuje się do* zostanie wyświetlona formuła, której odwołania będą zapisane względem aktywnej komórki. Na przykład: jeżeli komórką aktywną jest D32, w polu *Odwołuje się do* pojawi się następująca formuła:

$=\text{Arkusz1}!B32^{\text{Arkusz1}!C32}$

Zauważ, że Excel w odwołaniach do komórek użytych w formule tworzy nazwy kwalifikowane, dołączając nazwę arkusza. Oczywiście jeżeli taka formuła zostanie użyta w arkuszu innym niż ten, w którym ją zdefiniowano, spowoduje to wygenerowanie nieprawidłowych wyników. Aby użyć nazwanej formuły w arkuszu innym niż arkusz *Arkusz1*, trzeba usunąć z niej odwołania do tego arkusza (pozostawiając jednak znak wykrywki), na przykład:

$=\text{!A1}^{\text{!B1}}$

Kiedy już poznasz zasady tworzenia nazwanych formuł, z pewnością znajdziesz dla nich wiele nowych, ciekawych zastosowań. Nadawanie nazw formułom niewątpliwie przynosi korzyści w niektórych sytuacjach. Wystarczy wspomnieć o sytuacji, kiedy zachodzi potrzeba zmodyfikowania formuły — aby zmodyfikować nazwaną formułę, wystarczy zmienić jej definicję zawartą w polu *Odwołuje się do*. Unikasz w ten sposób konieczności edytowania każdego jej wystąpienia.

Klucz do zrozumienia nazw komórek i zakresów

Użytkownicy Excela często używają terminów *nazwane zakresy* i *nazwane komórki*. Faktycznie, sam często się nimi posługowałem w tym rozdziale, choć tak naprawdę oba terminy nie są zbyt precyzyjne.

A oto sekret, który pozwoli Ci lepiej zrozumieć ideę nazywania komórek i zakresów: *Gdy nadajesz nazwę komórce lub zakresowi, w praktyce definiujesz nazwaną formułę, która znajduje się nie w komórce, ale w pamięci Excela.*

Gdy używasz okna dialogowego *Nowa nazwa*, pole *Odwołuje się do* zawiera formułę, natomiast w polu *Nazwa* znajduje się jej nazwa. Zawartość pola *Odwołuje się do* zawsze rozpoczyna się od znaku =, który powoduje, że wpisany dalej ciąg znaków jest traktowany jako formuła.

Nie jest to może aż tak zaskakujące odkrycie, ale zapamiętanie tej wskazówki może pomóc w zrozumieniu, co naprawdę się dzieje podczas tworzenia i stosowania nazw w skoroszytach.



W sieci

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt o nazwie *Nazwane formuły.xlsx*, prezentujący kilka przykładów zastosowania nazwanych formuł.



Wskazówka

Kiedy pracujesz z oknem dialogowym *Nowa nazwa*, pole *Odwołuje się do* domyślnie znajduje się w trybie „wskazywania”, który ułatwia wprowadzanie odwołań do zakresów komórek poprzez proste klikanie odpowiednich obszarów arkusza. Aby przełączyć to pole w tryb edycji, pozwalający na użycie strzałek kurSORA i bezpośrednie edytowanie formuły, naciśnij klawisz *F2*.

Nadawanie nazw obiektom

Oprócz tworzenia nazw komórek, zakresów i formuł, nazwy możesz również przypisywać takim obiektom, jak tabele przestawne i kształty, dzięki czemu odwoływanie się do takich obiektów może być zdecydowanie łatwiejsze, zwłaszcza z poziomu procedur języka VBA.

Jeżeli chcesz zmienić nazwę takiego obiektu, powinieneś użyć pola *Pole nazwy* znajdującego się po lewej stronie paska formuły. Aby to zrobić, zaznacz obiekt, wpisz w polu *Pole nazwy* nową nazwę i naciśnij klawisz *Enter*.



Uwaga

Jeśli po wprowadzeniu nazwy w polu *Pole nazwy* klikniesz dowolne miejsce skoroszytu, nazwa nie zostanie zachowana. Aby zachować wprowadzaną nazwę, musisz użyć klawisza *Enter*.

Nazwy obiektów to coś zupełnie innego niż nazwy zakresów i z tego powodu nie są wyświetlane w oknie dialogowym menedżera nazw.

Błędy występujące w formułach

W praktyce bardzo często zdarza się, że zamiast wyniku formuły wyświetla w komórce komunikat o błędzie. Najczęstsza przyczyna jest z reguły bardzo prozaiczna — podczas wpisywania do formuły wkradł się jakiś błąd. Formuła może również wyświetlać komunikat o błędzie, kiedy komórka, do której się odwołuje, zawiera nieprawidłową wartość. Taka sytuacja może prowadzić do czegoś, co czasami określamy mianem **efektu falowego**,

polegającego na tym, że pojedyncza, błędna wartość może spowodować wystąpienie szeregu błędów w innych komórkach przechowujących formuły odwołujące się do tej komórki. Narzędzia w grupie opcji *Inspekcja formuł* na karcie *FORMUŁY* mogą być bardzo pomocne podczas identyfikacji źródła błędów.

W tabeli 2.2 zamieszczono zestawienie błędów, które mogą pojawić się w komórce zawierającej formułę.

Tabela 2.2. Wartości błędów w Excelu

Wartość błędu	Opis
#DZIEL/0!	Formuła dokonała próby dzielenia przez zero (jak wiadomo, na naszej planecie taka operacja jest niedozwolona). Błąd taki wystąpi też, gdy formuła spróbuje dokonać dzielenia przez komórkę, w której nie została zapisana żadna wartość (komórka pusta).
#N/D!	Formuła odwołała się (pośrednio lub bezpośrednio) do komórki używającej funkcji <i>BRAK()</i> stwierdzającej niedostępność danych. Błąd #N/D! jest również zwracany przez funkcję <i>WYSZUKAJ</i> , która nie może zlokalizować wartości.
#NAZWA?	Formuła używa nazwy, której Excel nie rozpoznaje. Taka sytuacja może mieć miejsce, gdy usuniesz nazwę używaną w formule lub gdy w tekście brakuje jednego ze znaków cudzysłowu. Formuła może wygenerować ten błąd także wówczas, gdy korzysta z funkcji dodatku, który nie został zainstalowany.
#ZERO!	Formuła korzysta z przecięcia dwóch zakresów, które faktycznie się nie przecinają. Zagadnienia związane z przecinaniem się zakresów komórek omawialiśmy już we wcześniejszej części tego rozdziału.
#LICZBA!	Wystąpił problem związany z argumentem funkcji. Przykładowo funkcja <i>PIERWIASTEK</i> próbuje obliczyć pierwiastek kwadratowy dla liczby ujemnej. Taki błąd wystąpi też, gdy obliczona wartość jest zbyt duża lub zbyt mała. Excel nie obsługuje wartości różnych od zera mniejszych od 1E-307 lub większych od 1E+308.
#ADR!	Formuła odwołuje się do nieprawidłowej komórki. Taka sytuacja może mieć miejsce na przykład wtedy, kiedy komórka zostanie usunięta z arkusza.
#ARG!	Formuła używa niewłaściwego typu argumentu. <i>Argument</i> jest wartością lub odwołaniem do komórki używanym przez formułę przy obliczaniu wyniku. Taki błąd może wystąpić, gdy formuła użyje niestandardowej funkcji napisanej w języku VBA, która zawiera błąd.
#####	Ten błąd występuje wówczas, gdy kolumna ma za małą szerokość, aby wyświetlić całą zawartość komórki, albo gdy formuła zwróciła ujemną wartość daty lub godziny.

Narzędzia inspekcji

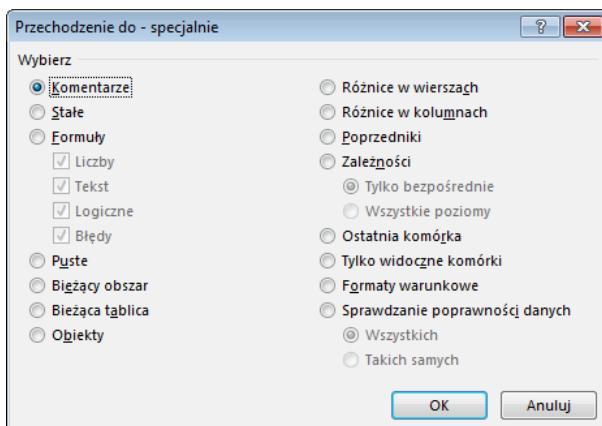
Excel ma szereg narzędzi, które wspomagają użytkownika w śledzeniu błędów w formułach arkuszowych. W tej sekcji omówimy pokrótko wbudowane narzędzia inspekcji Excela.

Identyfikowanie komórek określonego typu

Okno dialogowe *Przechodzenie do — specjalnie* (przedstawione na rysunku 2.6) jest wygodnym narzędziem, pozwalającym na szybkie zlokalizowanie komórek określonego typu. Aby wyświetlić to okno na ekranie, przejdź na kartę *NARZĘDZIA GŁÓWNE*, a następnie wybierz polecenie *Przejdz do — specjalnie*, znajdujące się w grupie opcji *Edytowanie*.

Rysunek 2.6.

Okno dialogowe
*Przechodzenie
do — specjalnie*



Jeżeli przed wyświetleniem na ekranie okna dialogowego *Przechodzenie do — specjalnie* zaznaczyłeś zakres komórek, to działanie tego polecenia będzie ograniczone wyłącznie do tego zaznaczonego obszaru. Jeżeli zaznaczona będzie pojedyncza komórka, to polecenie będzie działało na całym obszarze arkusza.

Okna dialogowego *Przechodzenie do — specjalnie* możesz używać do zaznaczania komórek określonego typu, co może być bardzo przydatne podczas wyszukiwania błędów i rozwiązywania problemów. Na przykład jeżeli wybierzesz opcję *Formuły*, Excel zaznaczy tylko takie komórki, które zawierają formuły. Jeżeli teraz zmniejszysz współczynnik powiększenia arkusza wyświetlany na ekranie, będziesz miał doskonały wgląd w sposób organizacji arkusza (patrz rysunek 2.7). Aby zmienić współczynnik powiększenia arkusza wyświetlany na ekranie, powinieneś użyć suwaka znajdującego się po prawej stronie paska statusu okna Excela lub po prostu wcisnąć i przytrzymać klawisz *Ctrl* i pokręcić w odpowiednią stronę kółkiem myszy.



Zaznaczenie komórek zawierających formuły może Ci również pomóc w wyłapaniu wielu prostych błędów, takich jak komórki, w których formuła została przypadkowo zastąpiona wartością stałą. Jeżeli w grupie komórek, w których znajdują się formuły, znajdziesz jakieś niezaznaczone komórki, to istnieje spora szansa, że poprzednio w takich komórkach znajdowała się formuła, która później przypadkowo została zastąpiona inną wartością.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Agent	Data	Obszar	Cena	Lic. syp.	taśmiki	Powierzchnia m ²	Typ	Besen	Sprzedażny	Cena za m ²	
2	Adamek	2007-10-09	Śródmieście	199 000,00 zł	3	2,5	140	Mieszkanie	FAŁSZ	FAŁSZ	1 418,55 zł	
3	Adamek	2007-08-19	Śródmieście	214 500,00 zł	4	2,5	173	Dom	PRAWDA	FAŁSZ	1 239,99 zł	
4	Adamek	2007-04-28	Śródmieście	260 000,00 zł	4	3	177	Dom	FAŁSZ	FAŁSZ	1 487,34 zł	
5	Adamek	2007-07-19	Śródmieście	268 500,00 zł	4	2,5	178	Dom	FAŁSZ	FAŁSZ	1 512,35 zł	
6	Adamek	2007-02-06	Śródmieście	273 500,00 zł	2	2	144	Dom	PRAWDA	PRAWDA	1 896,86 zł	
7	Adamek	2007-08-01	Śródmieście	309 950,00 zł	4	3	260	Dom	PRAWDA	FAŁSZ	1 191,53 zł	
8	Adamek	2007-01-15	Śródmieście	322 000,00 zł	3	2,5	163	Dom	FAŁSZ	PRAWDA	1 996,73 zł	
9	Jelonek	2007-01-29	Trynek	1 200 500,00 zł	5	5	436	Dom	PRAWDA	FAŁSZ	2 751,72 zł	
10	Roboczek	2007-04-04	Trynek	799 000,00 zł	6	5	446	Dom	FAŁSZ	FAŁSZ	1 791,74 zł	
11	Hirek	2007-02-24	Trynek	423 900,00 zł	5	3	224	Dom	PRAWDA	FAŁSZ	1 899,07 zł	
12	Refałski	2007-04-24	Trynek	403 000,00 zł	2	3	227	Dom	PRAWDA	PRAWDA	1 783,71 zł	
13	Adamek	2007-04-21	Sikornik	208 750,00 zł	4	3	205	Dom	PRAWDA	PRAWDA	1 018,11 zł	
14	Szostak	2007-03-24	Trynek	398 000,00 zł	4	2,5	243	Dom	FAŁSZ	FAŁSZ	1 635,13 zł	
15	Kraszczek	2007-06-09	Trynek	389 500,00 zł	4	2	183	Dom	FAŁSZ	FAŁSZ	2 127,11 zł	
16	Szostak	2007-08-17	Trynek	389 000,00 zł	4	3	289	Dom	FAŁSZ	FAŁSZ	1 346,79 zł	
17	Adamek	2007-06-06	Trynek	379 900,00 zł	3	2,5	219	Mieszkanie	FAŁSZ	FAŁSZ	1 656,89 zł	
18	Adamek	2007-02-03	Trynek	379 000,00 zł	3	3	219	Mieszkanie	FAŁSZ	PRAWDA	1 733,02 zł	
19	Rzeczek	2007-03-30	Trynek	379 000,00 zł	4	3	279	Dom	FAŁSZ	PRAWDA	1 359,84 zł	
20	Barnes	2007-06-26	Sikornik	208 750,00 zł	4	2	167	Dom	FAŁSZ	FAŁSZ	1 248,81 zł	
21	Benek	2007-05-12	Śródmieście	228 500,00 zł	4	3	190	Dom	FAŁSZ	PRAWDA	1 110,53 zł	
22	Benek	2007-05-09	Śródmieście	348 000,00 zł	4	3	180	Dom	PRAWDA	FAŁSZ	3 046,07 zł	
23	Szostak	2007-07-15	Trynek	374 900,00 zł	4	3	365	Dom	FAŁSZ	FAŁSZ	1 027,80 zł	
24	Lang	2007-05-03	Trynek	369 900,00 zł	3	2,5	189	Mieszkanie	PRAWDA	FAŁSZ	1 961,36 zł	
25	Roboczek	2007-01-28	Trynek	369 900,00 zł	4	3	185	Mieszkanie	FAŁSZ	PRAWDA	2 002,80 zł	
26	Benek	2007-06-26	Sikornik	229 900,00 zł	3	2,5	147	Dom	PRAWDA	FAŁSZ	1 566,32 zł	
27	Chudy	2007-07-06	Śródmieście	236 900,00 zł	3	2	158	Dom	FAŁSZ	FAŁSZ	1 489,98 zł	
28	Chudy	2007-08-27	Śródmieście	338 900,00 zł	4	2	208	Dom	FAŁSZ	FAŁSZ	1 634,79 zł	
29	Chudy	2007-04-21	Śródmieście	373 000,00 zł	4	3	229	Dom	PRAWDA	FAŁSZ	1 656,18 zł	
30	Chudy	2007-03-22	Sikornik	203 000,00 zł	3	2,5	186	Dom	PRAWDA	FAŁSZ	1 102,75 zł	
31	Chudy	2007-08-03	Sikornik	229 500,00 zł	4	2,5	212	Mieszkanie	FAŁSZ	FAŁSZ	1 081,57 zł	
32	Chudy	2007-09-30	Sikornik	233 990,00 zł	5	3	233	Mieszkanie	FAŁSZ	FAŁSZ	932,86 zł	
33	Chudy	2007-07-05	Sikornik	239 900,00 zł	4	3	210	Dom	FAŁSZ	FAŁSZ	1 142,59 zł	
34	Chudy	2007-09-02	Sikornik	242 000,00 zł	4	3	194	Dom	FAŁSZ	FAŁSZ	1 263,43 zł	
35	Chudy	2007-10-18	Sikornik	264 900,00 zł	4	2,5	231	Mieszkanie	FAŁSZ	FAŁSZ	1 146,04 zł	
36	Dobry	2007-10-01	Śródmieście	340 000,00 zł	4	2,5	234	Mieszkanie	FAŁSZ	FAŁSZ	1 454,00 zł	
37	Dobry	2007-02-20	Śródmieście	354 000,00 zł	4	2	194	Dom	FAŁSZ	FAŁSZ	1 824,91 zł	
38	Dobry	2007-03-25	Sikornik	204 900,00 zł	3	2,5	151	Dom	FAŁSZ	PRAWDA	1 353,08 zł	
39	Dobry	2007-08-12	Sikornik	223 911,00 zł	4	2,5	177	Dom	FAŁSZ	PRAWDA	1 274,47 zł	
40	Dobry	2007-01-29	Sikornik	289 000,00 zł	3	2	131	Dom	FAŁSZ	PRAWDA	1 911,97 zł	
41	Dobry	2007-05-19	Sikornik	380 000,00 zł	5	3	196	Dom	PRAWDA	PRAWDA	1 834,76 zł	
42	Hirek	2007-08-28	Śródmieście	222 911,00 zł	4	3	212	Dom	PRAWDA	FAŁSZ	1 064,19 zł	
43	Hirek	2007-01-24	Śródmieście	283 000,00 zł	2	1	189	Dom	FAŁSZ	PRAWDA	1 306,73 zł	
44	Szostak	2007-07-28	Trynek	369 900,00 zł	5	3	230	Dom	FAŁSZ	FAŁSZ	1 607,41 zł	
45	Hirek	2007-04-27	Sikornik	304 900,00 zł	4	3	218	Dom	FAŁSZ	PRAWDA	1 396,56 zł	
46	Jelonek	2007-04-11	Śródmieście	319 000,00 zł	4	2	157	Mieszkanie	PRAWDA	FAŁSZ	2 031,77 zł	
47	Rzeczek	2007-05-14	Trynek	359 900,00 zł	3	3	171	Mieszkanie	FAŁSZ	PRAWDA	2 106,34 zł	
48	Roboczek	2007-08-03	Trynek	359 900,00 zł	3	2	204	Mieszkanie	PRAWDA	FAŁSZ	1 762,48 zł	
49	Lang	2007-08-28	Trynek	359 900,00 zł	5	2,5	205	Dom	FAŁSZ	FAŁSZ	1 748,52 zł	
50	Barnes	2007-06-26	Trynek	359 000,00 zł	4	2,5	246	Mieszkanie	PRAWDA	FAŁSZ	1 443,59 zł	
51	Jelonek	2007-03-26	Sikornik	247 300,00 zł	4	3	186	Dom	FAŁSZ	FAŁSZ	1 332,03 zł	
52	Jelonek	2007-04-14	Sikornik	316 000,00 zł	3	2,5	161	Mieszkanie	FAŁSZ	PRAWDA	1 549,74 zł	

Arkusz1

Rysunek 2.7. Zmniejszenie wyświetlanego obrazu i zaznaczenie wszystkich komórek zawierających formuły pozwala na szybkie zorientowanie się w strukturze arkusza

Przeglądanie formuł

Jeżeli chcesz się zapoznać z organizacją nowego, nieznanego Ci skoroszytu, powinieneś zamiast wartości w komórkach wyświetlić formuły. Aby przełączyć wyświetlanie formuł, przejdź na kartę **FORMUŁY**, a następnie wybierz polecenie *Pokaż formuły*, znajdujące się w grupie opcji *Inspekcja formuł*. Przed wykonaniem tego polecenia dobrze jest otworzyć drugie okno skoroszytu, tak by w jednym oknie wyświetlać komórki z formułami, a w drugim komórki, w których formuły zostały zastąpione wynikami ich działania. Aby otworzyć nowe okno skoroszytu, przejdź na kartę **WIDOK** i wybierz polecenie *Nowe okno*, znajdujące się w grupie opcji *Okno*.



Żeby szybko przełączyć wyświetlanie formuł, możesz nacisnąć kombinację klawiszy **Ctrl+`** (lewy apostrof, zazwyczaj znak ten znajdziesz na klawiaturze nad klawiszem **Tab**).

Na rysunku 2.8 przedstawiono przykładowy skoroszyt wyświetlany w dwóch oknach. W górnym oknie skoroszyt jest wyświetlany w normalny sposób (wyniki działania formuł), a w dolnym oknie w komórkach arkusza są wyświetlane formuły. Wybranie polecenia *Wyświetl obok siebie*, znajdującego się w grupie opcji *Okno* na karcie *WIDOK*, pozwala na synchroniczne przewijanie zawartości obu okien.

The screenshot displays two Excel windows side-by-side. Both windows have the title 'Kalkulator prowizji.xlsx - Excel' and show the same data. The top window displays the results of the formulas, while the bottom window shows the raw formulas themselves.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Wysokość prowizji	5,50%	Standardowa wysokość prowizji														
2	Limit sprzedaży	15%	Wzrost w porównaniu do poprzedniego miesiąca														
3	Wysokość premii	6,50%	Do wypłaty, jeżeli limit sprzedaży został osiągnięty														
4																	
5	Sprzedawca	Poprzedni miesiąc	Bieżący miesiąc	Zmiana	Zmiana w %	Limit osiągnięty?	Prowizja										
6	Adamczyk	101 293	108 444	7 211	7,1%	PRAWDA	7 049										
7	Dąbrowski	120 933	108 434	-12 499	-10,3%	FALSZ	5 964										
8	Jankowski	139 832	165 901	26 069	18,6%	PRAWDA	10 784										
9	Kamiński	98 323	100 083	1 760	1,8%	FALSZ	5 505										
10	Pietrzak	78 322	79 923	1 601	2,0%	FALSZ	4 396										
11	RAZEM	538 643	562 785	24 142	4,5%		33 697										
12																	
13	Średnia wysokość prowizji:		5,99%														
14																	
15																	
16																	
	Arkusz1																

	A	B	C	D	E	F	G
1	Wysokość prowizji	0,055	Standardowa wysokość				
2	Limit sprzedaży	0,15	Wzrost w porównaniu				
3	Wysokość premii	0,065	Do wypłaty, jeżeli limit				
4							
5	Sprzedawca	Poprzedni miesiąc	Bieżący miesiąc	Zmiana	Zmiana w %	Limit osiągnięty?	Prowizja
6	Adamczyk	101233	108444	=C6-B6	=D6/B6	=E6=S\$3	=J\$ELU(F6:\$B\$3:\$B\$1)
7	Dąbrowski	120933	108434	=C7-B7	=D7/B7	=E7=S\$3	=J\$ELU(F7:\$B\$3:\$B\$1)
8	Jankowski	139832	165901	=C8-B8	=D8/B8	=E8=S\$3	=J\$ELU(F8:\$B\$3:\$B\$1)
9	Kamiński	98323	100083	=C9-B9	=D9/B9	=E9=S\$3	=J\$ELU(F9:\$B\$3:\$B\$1)
10	Pietrzak	78322	79923	=C10-B10	=D10/B10	=E10=S\$3	=J\$ELU(F10:\$B\$3:\$B\$1)
11	RAZEM	SUMA(B6:B10)	SUMA(C6:C10)	SUMA(D6:D10)	=D11/B11		SUMA(G6:G10)
12							
13	Średnia wysokość prowizji:			=G11/C11			
14							

Rysunek 2.8. Wyświetlanie formuł (dolne okno) i wyników ich działania (górnego okna)

Śledzenie zależności między komórkami arkusza

Aby sprawnie posługiwać się mechanizmami śledzenia zależności między komórkami arkusza, powinieneś najpierw poznać dwa podstawowe pojęcia:

- **Komórki poprzedni** — to pojęcie ma zastosowanie tylko do komórek zawierających formuły. Poprzednikami danej komórki są wszystkie inne komórki, których wartości mają wpływ na wynik działania formuły w tej komórce. *Poprzednikami bezpośrednimi* nazywamy takie komórki, które zostały użyte bezpośrednio w formule. *Poprzednikami pośrednimi* nazywamy takie komórki, które nie zostały bezpośrednio użyte w formule, ale są wykorzystywane przez inne komórki, do których taka formuła się odwołuje.
- **Komórki zależne** — to takie komórki, które zawierają formuły odwołujące się do innych komórek. Podobnie jak w poprzednim przypadku, komórka zawierająca formułę może być *zależna bezpośrednio* lub *zależna pośrednio* od innych komórek.

Przykładowo założymy, że w komórce A4 została wprowadzona prosta formuła:

=SUMA(A1:A3)

Komórka A4 ma trzy komórki poprzedniki (A1, A2 i A3); wszystkie trzy komórki to poprzedniki bezpośrednie. Komórki A1, A2 i A3 mają co najmniej po jednej komórce zależnej (komórka A4); w każdym przypadku jest to komórka zależna bezpośrednio.

Identyfikacja poprzedników komórki bardzo często pozwala na zorientowanie się, co poszło nie tak, jak powinno, i wyjaśnienie, dlaczego dana formuła nie działa prawidłowo. Podobnie możemy powiedzieć, że znajomość komórek zależnych danej formuły również może być bardzo przydatna. Na przykład jeżeli masz zamiar usunąć daną formułę z arkusza, to aby uniknąć problemów, możesz wcześniej upewnić się, czy ma ona jakieś komórki zależne.

Identyfikowanie poprzedników

Komórki wykorzystywane przez formułę znajdująca się w aktywnej komórce możesz zidentyfikować na kilka sposobów:

- **Naciśnij klawisz *F2*** — Komórki bezpośrednio wykorzystywane przez formułę zostaną wyróżnione kolorowymi obramowaniami, odpowiadającymi poszczególnym odwołaniom w formule. Za pomocą tej metody można identyfikować tylko komórki znajdujące się na tym samym arkuszu co formuła.
- **Użyj okna dialogowego *Przejdź do — specjalnie*** — Aby przywołać to okno na ekran, przejdź na kartę *NARZĘDZIA GŁÓWNE*, wybierz polecenie *Znajdź i zaznacz*, znajdujące się w grupie opcji *Edytowanie*, i następnie wybierz polecenie *Przejdź do — specjalnie*. W oknie dialogowym, które pojawi się na ekranie, zaznacz opcję *Poprzedniki*, a następnie wybierz opcję *Tylko bezpośrednie* (aby wyszukać wyłącznie poprzedniki bezpośrednie) lub opcję *Wszystkie poziomy* (aby wyszukać poprzedniki bezpośrednie i pośrednie). Naciśnij przycisk *OK* i Excel zaznaczy wszystkie komórki będące poprzednikami komórki aktywnej. Podobnie jak poprzednio, za pomocą tej metody można identyfikować tylko komórki znajdujące się na tym samym arkuszu co formuła.
- **Naciśnij kombinację klawiszy *Ctrl+I*** — Naciśnięcie tej kombinacji klawiszy zaznacza wszystkie bezpośrednie poprzedniki komórki, znajdującej się na aktywnym arkuszu.
- **Naciśnij kombinację klawiszy *Ctrl+Shift+{*** — Naciśnięcie tej kombinacji klawiszy zaznacza wszystkie poprzedniki komórki (pośrednie i bezpośrednie), znajdujące się na aktywnym arkuszu.
- **Przejdź na kartę *FORMUŁY* i wybierz polecenie *Inspekcja formuł/Sledź poprzedniki*** — Po wybraniu tego polecenia Excel narysuje strzałki wskazujące poprzedniki aktywnej komórki. Aby wyświetlić kolejne poziomy poprzedników, naciśnij kilka razy przycisk polecenia *Sledź poprzedniki*. By ukryć narysowane strzałki, przejdź na kartę *FORMUŁY* i wybierz polecenie *Usuń strzałki*, znajdujące się w grupie poleceń *Inspekcja formuł*. Na rysunku 2.9 przedstawiono arkusz z naniesionymi strzałkami wskazującymi poprzedniki formuły znajdującej się w komórce C13.

	A	B	C	D	E	F	G	H
1	Wysokość prowizji	5,50%	Standardowa wysokość prowizji					
2	Limit sprzedaży	15%	Wzrost w porównaniu do poprzedniego miesiąca					
3	Wysokość premii	6,50%	Do wyplaty, jeżeli limit sprzedaży został osiągnięty					
4								
5	Sprzedawca	Poprzedni miesiąc	Bieżący miesiąc	Zmiana	Limit osiągnięty?	Prowizja		
6	Adamczyk	101 233	108 444	7 211	7,1% PRAWDA	7 049		
7	Dąbrowski	120 933	108 434	-12 499	-10,3% FAŁSZ	5 964		
8	Jankowski	139 832	165 901	26 069	18,6% PRAWDA	10 784		
9	Kamiński	98 323	100 083	1 760	1,8% FAŁSZ	5 505		
10	Pietrzak	78 322	79 923	1 601	2,0% FAŁSZ	4 396		
11	RAZEM	538 643	562 785	24 142	4,5%	33 697		
12								
13	Średnia wysokość prowizji:			5,99%				
14								
15								

Rysunek 2.9. Arkusz z wyświetlonymi strzałkami wskazującymi poprzedniki formuły znajdujące się w komórce C13

Identyfikowanie komórek zależnych

Komórki zależne możesz zidentyfikować na kilka sposobów:

- **Użyj okna dialogowego *Przechodzenie do — specjalnie*** — Przywołaj okno na ekran, zaznacz opcję *Zależności*, a następnie opcję *Tylko bezpośrednie* (aby wyszukać wyłącznie zależności bezpośrednie) lub opcję *Wszystkie poziomy* (aby wyszukać zależności bezpośrednie i pośrednie). Naciśnij przycisk *OK* i Excel zaznaczy wszystkie komórki będące komórkami zależnymi. Za pomocą tej metody można identyfikować tylko komórki znajdujące się na tym samym arkuszu co formuła.
- **Naciśnij kombinację klawiszy *Ctrl+J*** — Naciśnięcie tej kombinacji klawiszy zaznacza wszystkie bezpośrednie komórki zależne, znajdujące się na aktywnym arkuszu.
- **Naciśnij kombinację klawiszy *Ctrl+Shift+}*** — Naciśnięcie tej kombinacji klawiszy zaznacza wszystkie komórki zależne (pośrednie i bezpośrednie), znajdujące się na aktywnym arkuszu.
- **Przejdź na kartę *FORMUŁY* i wybierz polecenie *Inspekcja formuł/Śledź zależności*** — Po wybraniu tego polecenia Excel narysuje strzałki wskazujące komórki zależne. Aby wyświetlić kolejne poziomy, naciśnij kilka razy przycisk polecenia *Śledź zależności*. By ukryć narysowane strzałki, przejdź na kartę *FORMUŁY* i wybierz polecenie *Usuń strzałki*, znajdujące się grupie poleceń *Inspekcja formuł*.

Śledzenie błędów

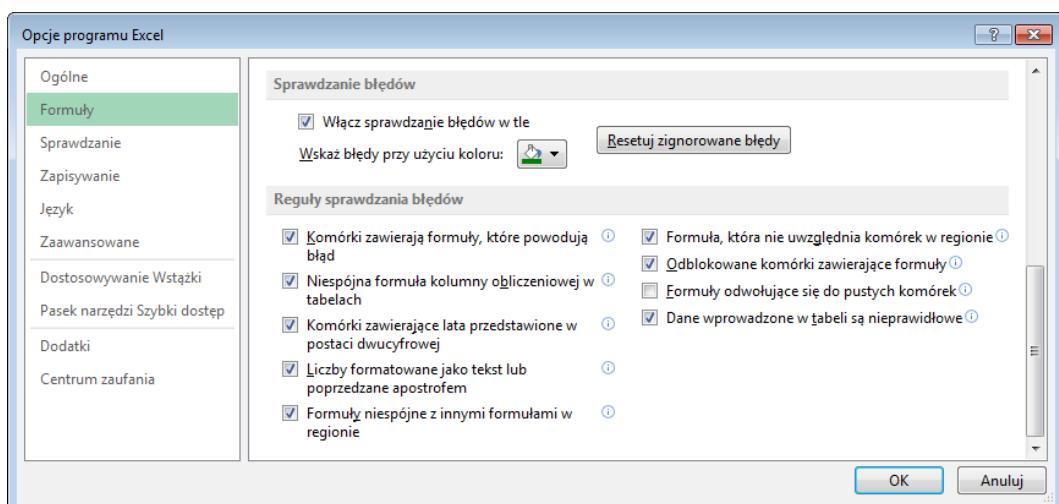
Jeżeli wynikiem działania formuły jest błąd, Excel może Ci pomóc w identyfikacji komórki, która spowodowała wystąpienie tego błędu. Błąd w jednej z komórek arkusza jest często wynikiem powstania błędu w jednej z komórek poprzedników. Aby to sprawdzić, uaktywnij komórkę zawierającą błąd, a następnie przejdź na kartę *FORMUŁY* i wybierz polecenie *Inspekcja formuł/Sprawdzanie błędów/Śledź błędy*. Excel narysuje na ekranie strzałki wskazujące komórkę(i) będącą potencjalnym źródłem problemu.

Naprawianie błędów spowodowanych odwołaniami cyklicznymi

Jeżeli przypadkowo utworzysz formułę zawierającą odwołanie cykliczne, Excel wyświetli na pasku stanu komunikat *Odwołania cykliczne* oraz adres komórki, a także narysuje na arkuszu strzałki, które pomogą Ci zidentyfikować i rozwiązać problem. Jeżeli nie potrafisz zlokalizować źródła problemu, przejdź na kartę *FORMUŁY* i wybierz polecenie *Inspekcja formuł/Sprawdzanie błędów/Odwołania cykliczne*. Na ekranie pojawi się menu podrzczne zawierające listę wszystkich komórek wchodzących w skład danego odwołania cyklicznego. Rozpocznij analizę od pierwszej komórki na liście, a następnie sprawdzaj kolejne pozycje listy aż do momentu zlokalizowania i rozwiązania problemu.

Zastosowanie mechanizmu sprawdzania błędów w tle

Wielu użytkowników chętnie korzysta z mechanizmu automatycznego sprawdzania błędów w tle, który możesz włączyć lub wyłączyć za pomocą opcji *Włącz sprawdzanie błędów w tle*, znajdującej się w oknie dialogowym *Opcje programu Excel*, na karcie *Formuły*, w grupie opcji *Sprawdzanie błędów* (patrz rysunek 2.10). Oprócz tego do zmiany ustawień możesz użyć odpowiednich opcji w grupie *Reguły sprawdzania błędów*.



Rysunek 2.10. Excel może automatycznie sprawdzać formuły pod kątem występowania błędów

Kiedy opcja sprawdzania błędów w tle jest włączona, Excel na bieżąco przelicza formuły wprowadzone do arkusza. Jeżeli potencjalny błąd zostanie zidentyfikowany, Excel umieszcza mały, trójkątny znacznik w lewym górnym rogu komórki. Kiedy taka komórka zostanie aktywowana, na ekranie pojawi się lista rozwijana, na której zestaw poleceń jest uzależniony od rodzaju błędu. Na przykład na rysunku 2.11 przedstawiono listę poleceń, która pojawia się po kliknięciu komórki zawierającej błąd #DZIEL/0.

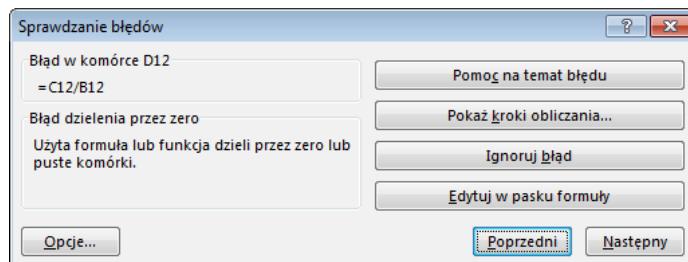
Rysunek 2.11.
Kiedy komórka zawierająca błąd zostanie aktywowana, na ekranie pojawi się lista rozwijana, na której zestaw poleceń jest uzależniony od rodzaju błędu

A	B	C	D	E	F	G
1	Wyniki telemarketingu					
3	Dzień	Ilość połączeń	Sprzedaż	Zmiana w %		
4	1	3598	74	2,06%		
5	2	3032	78	2,57%		
6	3	2987	68	2,28%		
7	4	3100	59	1,90%		
8	5	3535	43	1,22%		
9	6			#DZIEL/0!		
10	7			#DZIEL/0!		
11	8			#DZIEL/0!		
12	9				Błąd dzielenia przez zero	
13	10					
14	11					
15	12					
16					Pomoć na temat błędu	
17					Pokaż kroki obliczania...	
18						
19						
20						
21						

W wielu przypadkach będziesz mógł bezpiecznie zignorować taki błąd, wybierając polecenie *Ignoruj błąd*. Wybranie tego polecenia eliminuje komórkę z listy komórek poddawanych sprawdzaniu błędów. Jeżeli chcesz, aby ignorowane do tej pory błędy ponownie pojawiły się na ekranie, możesz użyć opcji *Resetuj zignorowane błędy*, którą znajdziesz na karcie *Formuły* okna dialogowego *Opcje programu Excel*.

Aby wyświetlić na ekranie okno dialogowe, w którym kolejno są wyświetlane opisy poszczególnych błędów (nieco podobnie jak to ma miejsce podczas sprawdzania pisowni w edytorze tekstu), przejdź na kartę *FORMUŁY* i wybierz polecenie *Sprawdzanie błędów*, znajdujące się w grupie polecień *Inspekcja formuł*. Polecenie to jest dostępne nawet wtedy, kiedy wyłączysz opcję sprawdzania błędów w tle. Na rysunku 2.12 przedstawiono wygląd okna *Sprawdzanie błędów*. Jest to okno *niemodalne* (ang. *modeless*), co oznacza, że kiedy to okno jest wyświetlone na ekranie, nadal masz pełny dostęp do arkusza.

Rysunek 2.12.
W oknie Sprawdzenie błędów wyświetlane są kolejno opisy poszczególnych błędów zidentyfikowanych przez Excela



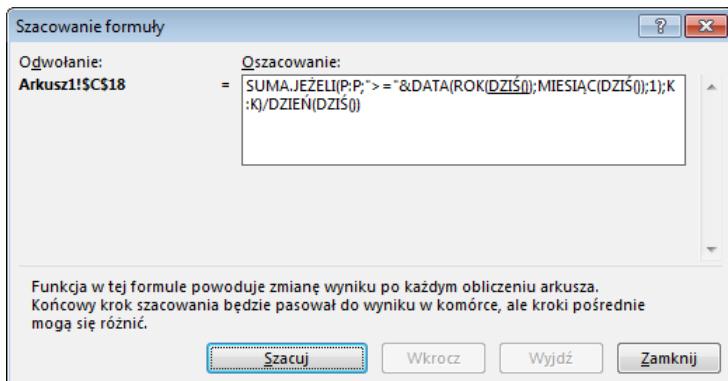


Niestety mechanizm sprawdzania błędów w Excelu nie jest doskonały. W rzeczywistości należałoby nawet powiedzieć, że jest daleki od doskonałości. Innymi słowy, nie możesz przyjąć założenia, że w Twoim arkuszu nie ma błędów, tylko dlatego, że Excel żadnego w nim nie znalazł! Co więcej, powinieneś również pamiętać, że mechanizm sprawdzania błędów nie jest w stanie wyłapać na przykład dosyć często spotykanej sytuacji, w której prawidłowo zapisana w komórce formuła jest przypadkowo zastępowana przez wartość.

Zastosowanie mechanizmu szacowania formuł

Polecenie *Szacuj formułę* pozwala na debugowanie złożonej formuły poprzez oszacowanie wartości każdej jej części osobno, w kolejności, w jakiej formuła jest w normalnych warunkach obliczana przez Excela. Aby skorzystać z mechanizmu szacowania formuł, zaznacz komórkę zawierającą wybraną formułę, przejdź na kartę *FORMUŁY* i wybierz polecenie *Szacuj formułę*, znajdujące się w grupie poleceń *Inspekcja formuł*. Po wybraniu tego polecenia na ekranie pojawi się okno dialogowe *Szacowanie formuły* (patrz rysunek 2.13).

Rysunek 2.13.
Okno dialogowe
Szacowanie formuły
pokazuje proces
obliczania formuły krok
po kroku



Aby wyświetlić wartość kolejnego obliczanego wyrażenia w formule, naciśnij przycisk *Szacuj*. Każde naciśnięcie tego przycisku oblicza kolejny element formuły. Mechanizm szacowania formuł może się na pierwszy rzut oka wydawać nieco skomplikowany, ale z pewnością po kilku próbach zrozumiesz, jak to działa, i przekonasz się, jak bardzo naprawdę to jest przydatne.

Excel oferuje również inny sposób szacowania wartości wybranych fragmentów formuły. Aby to zrobić, wykonaj polecenia opisane poniżej:

1. Zaznacz komórkę zawierającą badaną formułę.
 2. Naciśnij klawisz *F2*, aby przejść do trybu edycji zawartości komórki.
 3. Zaznacz fragment formuły, którego wartość chcesz oszacować.
- Do zaznaczania użyj myszy lub naciśnij i przytrzymaj klawisz *Shift*, a następnie użyj klawiszy kurSORA.
4. Naciśnij klawisz *F9*.

Zaznaczony fragment formuły zostanie zamieniony na obliczoną wartość. Teraz możesz w taki sam sposób oszacować wartość kolejnego fragmentu tej formuły. Jeżeli chcesz przywrócić formułę do jej oryginalnej postaci, po prostu naciśnij klawisz *Esc*.



Korzystając z tej metody, powinieneś zachować ostrożność, ponieważ jeżeli po oszacowaniu wartości wybranego fragmentu formuły naciśniesz klawisz *Enter*, formuła zostanie zmodyfikowana i taki fragment zostanie na stałe zastąpiony obliczoną wartością.

Formuły tablicowe

W terminologii Excela, *tablica* jest prostym zbiorem komórek lub wartości przetwarzanych grupowo. *Formuła tablicowa* jest specjalnego typu formułą obsługującą tablice. Formuła tablicowa może wygenerować pojedynczą wartość lub wiele wyników, z których każdy jest wyświetlany w oddzielnej komórce.

Przykładowo efektem mnożenia tablicy o wymiarach 1×5 przez inną tablicę o wymiarach 1×5 będzie trzecia tablica o wymiarach 1×5 . Innymi słowy, wynik tego typu operacji zajmie pięć komórek. Każdy element pierwszej tablicy jest mnożony przez odpowiadający mu element drugiej tablicy. W wyniku operacji uzyska się pięć nowych elementów, z których każdemu zostanie przydzielona własna komórka. Formuła tablicowa przedstawiona poniżej, mnoży wartości zakresu A1:A5 przez odpowiadające im wartości zakresu B1:B5. Taka formuła tablicowa jest jednocześnie wprowadzana do pięciu komórek:

`{=A1:A5*B1:B5}`



Aby wprowadzić formułę tablicową, powinieneś nacisnąć kombinację klawiszy *Ctrl+Shift+Enter*. Aby przypomnieć użytkownikowi, że określona formuła jest formułą tablicową, Excel umieszcza ją w pasku formuły w nawiasach klamrowych `{ }` . W naszej książce również będziemy używali nawiasów klamrowych do odróżnienia formuł tablicowych od normalnych formuł. Nie wprowadzaj samemu takich nawiasów!

Przykładowa formuła tablicowa

Formuły tablicowe Excela umożliwiają wykonywanie na każdej komórce zakresu poszczególnych operacji w prawie taki sam sposób, jak pętla języka programowania pozwala przetwarzać elementy tablicy. Jeżeli nigdy wcześniej nie używałeś formuł tablicowych, będziesz się miał teraz okazję z nimi bliżej zapoznać.

Na rysunku 2.14 przedstawiono arkusz, w którym do zakresu komórek A1:A5 wprowadzono tekst. Celem ćwiczenia jest utworzenie *pojedynczej formuły* zwracającej po zsumowaniu całkowitą liczbę znaków zawartych w tym zakresie komórek. Jeśli naszym założeniem nie byłoby dokonanie tego przy użyciu tylko jednej formuły, wynik można by uzyskać po zdefiniowaniu formuły używającej funkcji *DŁ*, przekopiowaniu jej do kolejnych komórek kolumny, a następnie zastosowaniu funkcji *SUMA* do zsumowania wyników zwróconych przez wszystkie formuły pośrednie.

Rysunek 2.14.

W komórce B1 znajduje się formuła tablicowa obliczająca całkowitą liczbę znaków z zakresu A1:A5. Zwróć uwagę na nawiasy klamrowe widoczne na pasku formuły

		B1	:	X	✓	fx	{=SUMA(DŁ(A1:A5))}
	A	B	C	D	E	F	
1	Królik	32					
2	Kojot						
3	Przepiórka						
4	Jeleń						
5	Pekari						
6							

Aby przekonać się, w jaki sposób formuła tablicowa może zajmować więcej niż jedną komórkę, utwórz arkusz pokazany na rysunku i następnie wykonaj polecenia przedstawione poniżej:

- Zaznacz zakres B1:B5.
- Wpisz następującą formułę:
=DŁ(A1:A5)
- Naciśnij kombinację klawiszy *Ctrl+Shift+Enter*.

Wykonanie powyższych poleceń powoduje wstawienie formuły tablicowej do pięciu komórek. Za pomocą dodatkowej formuły wykorzystującej funkcję SUMA możesz podsumować wartości zakresu B1:B5 (w naszym przykładzie otrzymaliśmy wynik 32).

A oto kluczowy element: *wyświetlanie* wszystkich pięciu elementów tablicy tak naprawdę nie jest konieczne, ponieważ Excel może po prostu przechowywać tablice w pamięci. Wiedząc o tym, poniższą formułę tablicową możesz umieścić w dowolnej, pustej komórce arkusza. (Zapamiętaj: tworząc formułę tablicową, nie wpisz nawiasów klamrowych — zamiast tego tworzenie formuły powinieneś zakończyć naciśnięciem kombinacji klawiszy *Ctrl+Shift+Enter*).

{=SUMA(DŁ(A1:A5))}

Powyższa formuła tworzy pięcioelementową tablicę (umieszczoną w pamięci) przechowującą długość każdego łańcucha z zakresu A1:A5. Funkcja SUMA używa tablicy jako swojego argumentu. Wynikiem zwracanym przez formułę jest wartość 32.

Kalendarz oparty na formule tablicowej

Na rysunku 2.15 przedstawiono arkusz wyświetlający kalendarz dla dowolnego miesiąca (jeżeli zmienisz miesiąc, poszczególne dni tygodni zostaną automatycznie zaktualizowane). Możesz wierzyć lub nie, ale główną część kalendarza stworzono przy użyciu jednej formuły tablicowej zajmującej 42 komórki.

Formuła tablicowa umieszczona w zakresie B5:H10 ma następującą postać:

```
=JEŻEL.I(MIESIĄC(DATA(ROK(B3);MIESIĄC(B3);1))<>MIESIĄC(DATA(ROK(B3);MIESIĄC(B3);1))-  
(DZIEN.TYG(DATA(ROK(B3);MIESIĄC(B3);1))-1)+{0;1;2;3;4;5}*7+{1\2\3\4\5\6\7}-  
1);"";DATA(ROK(B3);MIESIĄC(B3);1)-(DZIEŃ.TYG(DATA(ROK(B3);MIESIĄC(B3);1))-  
1)+{0;1;2;3;4;5}*7+{1\2\3\4\5\6\7}-1)
```

Formuła zwraca liczby seryjne kolejnych dat, stąd musisz jeszcze tylko odpowiednio sformatować komórki arkusza tak, aby poprawnie wyświetlały dzień miesiąca (format *Niestandardowy*, typ *d*).



W sieci

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt z kalendarzem, zawierający także kilka dodatkowych przykładów zastosowania formuł tablicowych. Skoroszyt z tymi przykładami nosi nazwę *Przykłady formuł tablicowych.xlsx*. Oprócz tego znajdziesz tam skoroszyt o nazwie *Kalendarz na cały rok.xlsx*, zawierający, jak sama nazwa wskazuje całoroczny kalendarz wykorzystujący formuły tablicowe.

The screenshot shows a Microsoft Excel spreadsheet. The formula bar at the top contains a complex array formula: `{=JEŻELI(MIESIĄC(DATA(ROK(B3);MIESIĄC(B3);1))<>MIESIĄC(DATA(ROK(B3);MIESIĄC(B3);1)-(DZIEŃ.TYG(DATA(ROK(B3);MIESIĄC(B3);1))-1)+{0;1;2;3;4;5}*7+{1\2\3\4\5\6\7}-1);"";DATA(ROK(B3);MIESIĄC(B3);1)-(DZIEŃ.TYG(DATA(ROK(B3);MIESIĄC(B3);1))-1)+{0;1;2;3;4;5}*7+{1\2\3\4\5\6\7}-1)}`. Below the formula bar is a table for the month of March 2013. The table has a dark green header row with days of the week: Niedziela (N), Poniedziałek (Pn), Wtorek (Wt), Środa (Sr), Czwartek (Cz), Piątek (Pt), Sobota (So). The days of the month are filled in the cells: Row 5 (3, 4, 5, 6, 7, 8, 9), Row 6 (10, 11, 12, 13, 14, 15, 16), Row 7 (17, 18, 19, 20, 21, 22, 23), Row 8 (24, 25, 26, 27, 28, 29, 30), and Row 9 (31). The table is set against a light green background. At the bottom of the screen, there is a navigation bar with buttons for back, forward, search, ranking, summary table, and calendar, followed by a status bar.

marzec 2013						
N	Pn	Wt	Sr	Cz	Pt	So
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Rysunek 2.15. Jedna formuła tablicowa wystarczy do utworzenia kalendarza dla dowolnego miesiąca dowolnego roku

Zalety i wady formuł tablicowych

Używanie formuł tablicowych zamiast formuł jednokomórkowych ma następujące zalety:

- Formuły tablicowe z reguły zajmują mniej pamięci.
- Znacznie poprawiają wydajność obliczeń.
- Pozwalają wyeliminować formuły pośrednie.
- Umożliwiają wykonywanie operacji, które w przeciwnym razie byłyby trudne do zrealizowania lub wręcz niemożliwe.

A oto kilka wad formuł tablicowych:

- Zastosowanie zbyt wielu złożonych formuł tablicowych może czasami znacznie wydłużyć czas ponownego przeliczania arkusza kalkulacyjnego.
- Formuły tablicowe powodują, że arkusz może być trudniejszy do zrozumienia dla innych użytkowników.
- Musisz pamiętać o zatwierdzaniu wszystkich formuł tablicowych poprzez naciśnięcie kombinacji klawiszy *Ctrl+Shift+Enter*.

Metody zliczania i sumowania

Jednym z najczęściej wykonywanych w Excelu zadań jest zliczanie i sumowanie warunkowe. W tym podrozdziale znajdziesz szereg przykładów formuł zliczających różne elementy występujące w arkuszu, bazujących na prostych lub złożonych kryteriach. Oczywiście możesz dostosowywać te formuły do własnych potrzeb.



Uwaga

W Excelu 2007 wprowadzono dwie nowe funkcje zliczające i sumujące, które nie były dostępne w poprzednich wersjach programu (**LICZ.JEŻELI** oraz **SUMA.JEŻELI**). Z tego właśnie powodu przedstawimy dwa warianty tej samej formuły: pierwsza wersja jest przeznaczona dla Excela 2007 i nowszych wersji, a druga to formuła tablicowa, która będzie działała w pozostałych, wcześniejszych wersjach.

Na rysunku 2.16 przedstawiono przykładowy arkusz ilustrujący kilka przydatnych formuł. W arkuszu zostały zdefiniowane następujące zakresy komórek:

- **Miesiąc:** A2:A10
- **Region:** B2:B10
- **Sprzedaż:** C2:C10

	A	B	C	D	E	F	G	H
1	Miesiąc	Region	Sprzedaż		Excel 2007+	Inne wersje		Opis
2	Styczeń	Północ	100			3	Podaje liczbę regionów	
3	Styczeń	Południe	200			2	Podaje liczbę komórek zakresu Sprzedaż, gdzie Sprzedaż=300.	
4	Styczeń	Zachód	300			2	Podaje liczbę komórek zakresu Sprzedaż, gdzie Sprzedaż>300	
5	Luty	Północ	150			8	Podaje liczbę komórek zakresu Sprzedaż, gdzie Sprzedaż<> 100	
6	Luty	Południe	250			6	Podaje liczbę regionów, których nazwa składa się z 6 liter	
7	Luty	Zachód	350			3	Podaje liczbę regionów, których nazwa zawiąra literę "h"	
8	Marzec	Północ	200			1	Podaje liczbę komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" i Sprzedaż >200	
9	Marzec	Południe	300			1	Podaje liczbę komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" i Region = "Północ"	
10	Marzec	Zachód	400			2	Podaje liczbę komórek zakresu Sprzedaż, gdzie Sprzedaż Miesiąc = "Styczeń" i Region = "Północ" lub "Południe"	
11						4	Podaje liczbę komórek zakresu Sprzedaż, gdzie Sprzedaż ma wartość między 300 a 400	
12								
13								
14								
15					Excel 2007	Inne wersje		Opis
16						1 600	Sumuje wartości komórek zakresu Sprzedaż, gdzie Sprzedaż >00	
17						600	Sumuje wartości komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń"	
18						1 350	Sumuje wartości komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" lub "Luty"	
19						100	Sumuje wartości komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" i Region = "Północ"	
20						500	Sumuje wartości komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" i Region<>"Północ"	
21						500	Sumuje wartości komórek zakresu Sprzedaż, gdzie Miesiąc = "Styczeń" i Sprzedaż >= 200	
22						1 350	Sumuje wartości komórek zakresu Sprzedaż, gdzie Sprzedaż ma wartość między 300 a 400	
23								

Rysunek 2.16. Prosty arkusz ilustrujący kilka przydatnych formuł do zliczania i sumowania



Skoroszyt z tym przykładem (*Zliczanie i sumowanie.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Przykłady formuł zliczających

W tabeli 2.3 przedstawiono przykłady formuł ilustrujących różne techniki zliczania danych.

Tabela 2.3. Przykłady formuł zliczających

Formuła	Opis
=LICZ.JEŻELI(Region;"Północ")	Oblicza liczbę wierszy, dla których spełniony jest warunek Region="Północ"
=LICZ.JEŻELI(Sprzedaż;300)	Oblicza liczbę wierszy, dla których spełniony jest warunek Sprzedaż=300
=LICZ.JEŻELI(Sprzedaż;">300")	Oblicza liczbę wierszy, dla których spełniony jest warunek Sprzedaż>300
=LICZ.JEŻELI(Sprzedaż;"<>100")	Oblicza liczbę wierszy, dla których spełniony jest warunek Sprzedaż<>100
=LICZ.JEŻELI(Region;"?????")	Oblicza liczbę wierszy, dla których nazwa regionu składa się z pięciu dowolnych znaków
=LICZ.JEŻELI(Region;"*d*")	Oblicza liczbę wierszy, dla których nazwa regionu zawiera literę <i>d</i> (bez rozróżniania wielkich i małych liter)
=LICZ.JEŻELI(Miesiąc;"Styczeń" ↳;Sprzedaż;">200")	Oblicza liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Sprzedaż>200 (tylko Excel 2007 i nowsze)
{=SUMA((Miesiąc="Styczeń")* ↳(Sprzedaż>200))}	Formuła tablicowa obliczająca liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Sprzedaż>200
=LICZ.JEŻELI(Miesiąc;"Styczeń" ↳;Region;"Północ")	Oblicza liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Region="Północ" (tylko Excel 2007 i nowsze)
{=SUMA((Miesiąc="Styczeń")* ↳(Region;"Północ"))}	Formuła tablicowa obliczająca liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Region="Północ"
=LICZ.JEŻELI(Miesiąc;"Styczeń"; ↳Region;"Północ")+LICZ.JEŻELI ↳(Miesiąc;"Styczeń";Region; ↳"Południe")	Oblicza liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Region="Północ" lub "Południe" (tylko Excel 2007 i nowsze)
{=SUMA((Miesiąc="Styczeń")* ↳((Region="Północ")+(Region= ↳"Południe")))}	Formuła tablicowa obliczająca liczbę wierszy, dla których spełnione są warunki Miesiąc="Styczeń" i Region="Północ" lub "Południe"
=LICZ.JEŻELI(Sprzedaż;">300"; ↳Sprzedaż;"<=400")	Oblicza liczbę wierszy, dla których Sprzedaż znajduje się w zakresie od 300 do 400 (tylko Excel 2007 i nowsze)
{=SUMA((Sprzedaż>=300)* ↳(Sprzedaż<=400))}	Formuła tablicowa obliczająca liczbę wierszy, dla których Sprzedaż znajduje się w zakresie od 300 do 400

Przykłady formuł sumujących

W tabeli 2.4 przedstawiono kilka przykładów formuł sumujących, ilustrujących różne techniki sumowania danych.

Tabela 2.4. Przykłady formuł sumujących

Formuła	Opis
=SUMA.JEŻELI(Sprzedaż;">>200")	Sumuje całość sprzedaży powyżej 200
=SUMA.JEŻELI(Miesiąc;"Styczeń";Sprzedaż)	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń"
=SUMA.JEŻELI(Miesiąc;"Styczeń";Sprzedaż)+ ↳SUMA.JEŻELI(Miesiąc;"Luty";Sprzedaż)	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń" lub "Luty"
{=SUMA((Miesiąc="Styczeń")*(Region="Północ")* Sprzedaż)}	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Region="Północ"
=SUMA.JEŻELI(Sprzedaż;Miesiąc;"Styczeń"; ↳Region; "Północ")	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Region="Północ" (tylko Excel 2007 i nowsze)
{=SUMA.JEŻELI((Miesiąc="Styczeń")* ↳(Region="Północ")*Sprzedaż)}	Formuła tablicowa, która sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Region="Północ"
=SUMA.JEŻELI(Sprzedaż;Miesiąc;"Styczeń"; ↳Region; "<>Północ")	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Region<>"Północ" (tylko Excel 2007 i nowsze)
{=SUMA((Miesiąc="Styczeń")*(Region<> ↳"Północ")*Sprzedaż)}	Formuła tablicowa, która sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Region<>"Północ"
=SUMA.JEŻELI(Sprzedaż;Miesiąc;"Styczeń"; ↳Sprzedaż; ">=200")	Sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Sprzedaż>=200 (tylko Excel 2007 i nowsze)
{=SUMA((Miesiąc="Styczeń")*(Sprzedaż>=200)* ↳(Sprzedaż))}	Formuła tablicowa, która sumuje Sprzedaż, gdzie Miesiąc="Styczeń" i Sprzedaż>=200
=SUMA.JEŻELI(Sprzedaż;Sprzedaż">=300"; ↳Sprzedaż; "<=400")	Sumuje Sprzedaż, która znajduje się w zakresie do 300 do 400 (tylko Excel 2007 i nowsze)
{=SUMA((Sprzedaż>=300)*(Sprzedaż<=400)* ↳(Sprzedaż))}	Formuła tablicowa sumująca Sprzedaż, która znajduje się w zakresie do 300 do 400

Inne narzędzia zliczające

Inne metody pozwalające na zliczanie bądź sumowanie komórek spełniających wybrane kryteria to na przykład:

- Filtrowanie (z wykorzystaniem tabel)
- Filtrowanie zaawansowane
- Zastosowanie funkcji BD.ILE.REKORDÓW oraz BD.SUMA
- Tabele przestawne

Więcej szczegółowych informacji na ten temat znajdziesz w pomocy systemowej programu Microsoft Excel.

Formuły wyszukiwania i adresu

W skoroszytach Excela bardzo często można spotkać formuły, które wyszukują w tablicy źródłowej poszukiwaną wartość i zwracają odpowiadającą jej wartość docelową. Dobrą analogią takiego rozwiązania jest klasyczna książka telefoniczna (czy ktoś je jeszcze pamięta?). Jeżeli chciełeś w takiej książce znaleźć numer telefonu wybranej osoby, najpierw musiałeś odszukać jej nazwisko i imię na liście i dopiero potem odczytać przypisany do niej numer telefonu.

Excel ma kilka funkcji, które są bardzo przydatne przy tworzeniu formuł wyszukiwujących dane w tablicach. W tabeli 2.5 zamieszczono zestawienie takich funkcji wraz z krótkim opisem.

Tabela 2.5. Funkcje wyszukiwania i adresu

Funkcja	Opis
WYBIERZ	Wybiera z listy wartość lub czynność do wykonania na podstawie numeru wskaźnika.
WYSZUKAJ.POZIOMO	Wyszukuje wartość w górnym wierszu tabeli lub tablicy wartości i zwraca wartość z tej samej kolumny ze wskazanego wiersza.
JEŻELI	Jeżeli warunek jest spełniony, zwraca pierwszą wartość, w przeciwnym wypadku jest zwracana druga wartość.
JEŻELI.BŁĄD	Jeżeli pierwszy argument funkcji zwraca błąd, obliczany i zwracany jest drugi argument funkcji. Jeżeli pierwszy argument funkcji nie zwraca błędu, jest obliczany i zwracany jako wynik działania funkcji.
INDEKS	Zwraca wartość lub odwołanie do komórki na przecięciu określonego wiersza i kolumny w danym zakresie.
WYSZUKAJ	Wyszukuje wartość z zakresu jednowierszowego lub jednokolumnowego albo z tablicy. Występuje w dwóch formach, tablicowej i wektorowej.
PODAJ.POZYCJĘ	Zwraca względną pozycję elementu w tablicy, odpowiadającą określonej wartości.
PRZESUNIĘCIE	Zwraca odwołanie do zakresu, który jest daną liczbą wierszy lub kolumn z danego odwołania.
WYSZUKAJ.PIONOWO	Wyszukuje wartość w pierwszej od lewej kolumnie tabeli i zwraca wartość z tego samego wiersza w kolumnie określonej przez użytkownika. Domyślnie tabela musi być posortowana w kolejności rosnącej.

Podstawowych funkcji wyszukiwania i adresu możesz używać do przeszukiwania kolumn lub wierszy pod kątem danej wartości i zwracania odpowiadającej jej innej wartości. Excel ma trzy podstawowe funkcje wyszukiwania: WYSZUKAJ.POZIOMO, WYSZUKAJ.PIONOWO oraz WYSZUKAJ. Oprócz tego funkcje PODAJ.POZYCJĘ oraz INDEKS są często używane ze sobą do zwracania komórki lub względnego odwołania do komórki zawierającej poszukiwaną wartość.

Funkcja WYSZUKAJ.PIONOWO wyszukuje wartość w pierwszej od lewej kolumnie tabeli i zwraca wartość z tego samego wiersza w kolumnie określonej przez użytkownika. Przeszukiwana tabela jest zorganizowana pionowo (co wyjaśnia nazwę funkcji). Składnia funkcji WYSZUKAJ.PIONOWO jest następująca:

`WYSZUKAJ.PIONOWO(szukana_wartość; tabela_tablica; nr_kolumny [:przeszukiwany_zakres])`

Argumenty wywołania tej funkcji są następujące:

- *szukana_wartość* — Argument wymagany. Wartość, która ma zostać odszukana w pierwszej kolumnie tabeli lub zakresu.
- *tabela_tablica* — Argument wymagany. Zakres komórek zawierający dane.
- *nr_kolumny* — Argument wymagany. Numer kolumny określonej przez argument *tabela_tablica*, z której musi zostać zwrocona znaleziona wartość.
- *przeszukiwany_zakres* — Argument opcjonalny. Wartość logiczna określająca, czy funkcja WYSZUKAJ.PIONOWO ma znaleźć dopasowanie dokładne, czy przybliżone. Jeżeli argument ma wartość *PRAWDA* lub zostanie pominięty, funkcja zwróci dopasowanie dokładne lub przybliżone — jeśli nie zostanie znalezione dokładne dopasowanie, funkcja zwróci następną największą wartość, która jest mniejsza od wartości argumentu *szukana_wartość*. Jeżeli argument *przeszukiwany_zakres* ma wartość *FAŁSZ*, funkcja WYSZUKAJ.PIONOWO wyszuka tylko dopasowanie dokładne. Jeżeli dopasowanie dokładne nie zostanie odnalezione, funkcja zwróci błąd #N/D.



Uwaga

Jeżeli parametr *przeszukiwany_zakres* ma wartość *PRAWDA* lub zostanie pominięty, wartości w pierwszej kolumnie *tabeli_tablicy* muszą być uporządkowane w kolejności rosnącej. Jeżeli wartość argumentu *szukana_wartość* jest mniejsza niż najmniejsza wartość w pierwszej kolumnie tablicy określonej za pomocą argumentu *tabela_tablica*, funkcja WYSZUKAJ.PIONOWO zwróci błąd #N/D. Jeżeli argument *przeszukiwany_zakres* ma wartość *FAŁSZ*, wartości w pierwszej kolumnie *tabeli_tablicy* nie muszą być uporządkowane. Jeżeli dopasowanie dokładne nie zostanie odnalezione, funkcja zwróci błąd #N/D.

Jednym z najczęstszych zastosowań polecenia WYSZUKAJ.PIONOWO jest sprawdzanie wysokości należnego podatku (patrz rysunek 2.17). W tabeli zamieszczono przykładowe zestawienie zakresów kwot stanowiących poszczególne progi podatkowe i wysokość podatku dochodowego (w %), jaki dla danej kwoty należy zapłacić. Formuła zamieszczona w komórce B3 zwraca wysokość podatku w %, jaki należy zapłacić dla kwoty wpisanej w komórce B2:

=WYSZUKAJ.PIONOWO(B2:D2:F7;3)

Tabela z wartościami znajduje się w zakresie komórek składającym się z trzech kolumn (D2:F7). Ponieważ ostatni argument funkcji WYSZUKAJ.PIONOWO to liczba 3, formuła zwraca odpowiednią wartość trzeciej kolumny tabeli wyszukiwania.

A	B	C	D	E	F	G
1	Podaj kwotę:	32 650 zł	Przychód większy lub równy...	...ale mniejszy lub równy	Tax Rate	
2			0 zł	2 650 zł	15,00%	
3	Wysokość podatku:	31,00%	2 651 zł	27 300 zł	28,00%	
4			27 301 zł	58 500 zł	31,00%	
5			58 501 zł	131 800 zł	36,00%	
6			131 801 zł	284 700 zł	39,60%	
7			284 701 zł		45,25%	
8						
9						
10						
	◀ ▶ ...	WYSZUKAJ.PIONOWO	WYSZUKAJ.POZIOMO	WYSZUKAJ	PODAJ.POZYCJĘ & INI ...	⊕

Rysunek 2.17. Zastosowanie funkcji WYSZUKAJ.PIONOWO do sprawdzania wysokości podatku

Zwróć uwagę, że dokładne wyszukiwanie nie jest w tym przypadku wymagane. Jeżeli dokładna wartość nie zostanie odnaleziona w pierwszej kolumnie tabeli, funkcja WYSZUKAJ. →PIONOWO wykorzystuje następną największą wartość, która jest mniejsza niż wartość w tabeli wyszukiwania. Innymi słowy, funkcja wykorzystuje wiersz, dla którego poszukiwana wartość jest większa lub równa wartości w wierszu, ale mniejsza niż wartość w kolejnym wierszu. W przypadku sprawdzania wysokości należnego podatku dochodowego jest to dokładnie takie zachowanie, jakiego oczekujemy.



Skoroszyt z tym przykładem (*Funkcje wyszukiwania i adresu.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Oprócz tego zamieszczono tam również skoroszyt Zaawansowane funkcje wyszukiwania i adresu.xlsx, zawierający dodatkowe, bardziej złożone przykłady zastosowania takich funkcji.

Przetwarzanie daty i czasu

Excel przechowuje daty przy użyciu systemu odpowiadających im liczb seryjnych. Najstarszą datą rozpoznawaną przez Excela jest 1 stycznia 1900 roku. Dacie tej odpowiada liczba 1, dacie 2 stycznia 1900 roku jest przypisana liczba 2 itd.

W większości przypadków nie musisz sobie zwracać głowy takimi liczbami seryjnymi. Po prostu wprowadzasz datę w znany Ci formacie i Excel zajmuje się całą resztą. Na przykład: jeżeli chcesz wpisać datę 15 sierpnia 2013, wystarczy, że wpiszesz w komórce dokładnie taki łańcuch znaków (lub zastosujesz dowolny inny format daty). Excel odpowiednio interpretuje wprowadzoną datę i przechowuje reprezentującą ją wartość 41501 (liczba seryjna odpowiadająca dacie 15 sierpnia 2013).



W tym rozdziale użyto formatu daty stosowanego w Polsce. Jeżeli w swoim systemie używasz innego formatu daty, prawdopodobnie będziesz musiał dokonać odpowiednich modyfikacji, na przykład daty będziesz musiał wprowadzać w formacie takim jak 2013-08-15.

Wprowadzanie daty i czasu

Dane reprezentujące czas możesz wprowadzać (w jednym ze standardowych, rozpoznawanych przez Excela formatów) bezpośrednio do komórek arkusza. System reprezentacji dat w postaci liczb seryjnych został rozszerzony o obsługę wartości dziesiętnych odpowiadających upływowi czasu w ciągu dnia. Innymi słowy, Excel reprezentuje upływ czasu przy użyciu tego samego systemu, niezależnie od tego, czy przetwarzany element dotyczy określonego dnia, godziny czy sekundy. Przykładowo liczba seryjna odpowiadająca dacie 15 sierpnia 2013 roku to 41501. Południe (czyli inaczej mówiąc, środek dnia) w wewnętrznym zapisie Excela jest reprezentowane przez wartość 41501,5. Również w tym przypadku zazwyczaj nie musisz się przejmować częścią ułamkową numerów seryjnych.

Ponieważ data i czas są zapisywane jako numery seryjne, można na nich wykonywać operacje dodawania i odejmowania. Na przykład w celu określenia liczby dni oddzielających dwie daty można zastosować odpowiednią formułę. Jeżeli zarówno w komórce A1, jak i w komórce A2 znajdują się daty, poniższa formuła zwróci liczbę dni stanowiących różnicę:

=A2-A1



Wykonując obliczenia związane z czasem, należy zachować pewną ostrożność. Jeśli do komórki zostanie wpisany czas bez powiązanej z nim daty, Excel domyślnie przyjmie, że chodzi o „dzień” 0 stycznia 1900 roku (data, dla której liczba seryjna ma wartość 0). Nie stanowi to problemu dopóty, dopóki w rezultacie obliczeń nie uzyskamy wartości ujemnej. Gdy do tego dojdzie, Excel wyświetli błąd #####. Jakie jest zatem możliwe rozwiązanie? W takim przypadku należy uaktywnić system daty 1904. Aby to zrobić, przywołaj na ekran okno *Opcje programu Excel*, kliknij kategorię *Zaawansowane*, a następnie odszukaj i zaznacz opcję *Użyj systemu daty 1904*. Pamiętaj jednak, że włączenie systemu daty 1904 może spowodować problemy z datami znajdującymi się już w arkuszu lub innych, powiązanych z nim skoroszytach.



W niektórych sytuacjach wartości czasu są częściej używane do reprezentowania okresu trwania danego zdarzenia niż do wskazywania określonego punktu w czasie. Na przykład możesz zsumować liczbę godzin, jakie przepracowałeś w danym tygodniu. W przypadku dodawania wartość reprezentujących czas, nie możesz wyświetlić więcej niż 24 godziny. Dla każdego 24-godzinnego okresu Excel po prostu do wartości sumarycznej dodaje kolejny dzień. Rozwiążanie tego problemu polega na wybraniu takiego formatu liczb, który umieszcza część godzinową w nawiasach kwadratowych. Na przykład poniższy format liczbowy pozwala na wyświetlanie więcej niż 24 godzin:

[gg]:mm

Przetwarzanie dat sprzed roku 1900

Nie jest chyba dla nikogo zaskoczeniem, że świat, jaki znamy, nie rozpoczął się 1 stycznia 1900 roku. Osoby przetwarzające w Excelu dane historyczne, bardzo często zatem muszą operować na datach wcześniejszych. Niestety jedyna metoda pozwalająca na używanie takich dat polega na wprowadzaniu ich do komórki jako tekstu, tak jak w poniższym przykładzie:

4 lipca 1776

Niestety Excel nie pozwala na manipulowanie datami, które zostały wprowadzone jako tekst. Na przykład nie można zmienić formatowania daty, określić, na który dzień tygodnia przypada dana data, czy wyznaczyć daty wypadającej siedem dni później.

Warto tutaj jednak zauważyc, że VBA pozwala na operowanie znacznie szerszym zakresem dat. Autor tej książki utworzył przy użyciu VBA szereg funkcji arkuszowych pozwalających na przetwarzanie dat wcześniejszych niż 1 stycznia 1900 roku. Na rysunku 2.18 przedstawiono przykłady zastosowania takich funkcji w arkuszu. Jest to doskonały przykład tego, jak VBA może znaczco rozszerzyć funkcjonalność samego Excela.

A	B	C	D	E	F	G	H	I
4	Przykłady: urodziny prezydentów							
6	Prezydent	Rok	Miesiąc	Dzień	XDATE	XDATEDIF	XDATEYEARIDF	XDATEDOW
7	George Washington	1732	2	22	22 lutego 1732	102 641	281	Piątek
8	John Adams	1735	10	30	30 października 1735	101 295	277	Niedziela
9	Thomas Jefferson	1743	4	13	13 kwietnia 1743	98 573	269	Sobota
10	James Madison	1751	3	16	16 marca 1751	95 679	261	Wtorek
11	James Monroe	1758	4	28	28 kwietnia 1758	93 079	254	Piątek
12	John Quincy Adams	1767	7	11	11 lipca 1767	89 718	245	Sobota
13	Andrew Jackson	1767	3	15	15 marca 1767	89 836	245	Niedziela
14	Martin Van Buren	1782	12	5	5 grudnia 1782	84 092	230	Czwartek
15	William Henry Harrison	1773	2	9	9 lutego 1773	87 678	240	Wtorek
16	John Tyler	1790	3	29	29 marca 1790	81 421	222	Poniedziałek
17	James K. Polk	1795	11	2	2 listopada 1795	79 377	217	Poniedziałek
18	Zachary Taylor	1784	11	24	24 listopada 1784	83 372	228	Środa
19	Millard Fillmore	1800	1	7	7 stycznia 1800	77 850	213	Wtorek
20	Franklin Pierce	1804	11	23	23 listopada 1804	76 069	208	Piątek
21	James Buchanan	1791	4	23	23 kwietnia 1791	81 031	221	Sobota
22	Abraham Lincoln	1809	2	12	12 lutego 1809	74 527	204	Niedziela
23	Andrew Johnson	1808	12	29	29 grudnia 1808	74 572	204	Czwartek
24	Ulysses S. Grant	1822	4	27	27 kwietnia 1822	69 705	190	Sobota
25	Rutherford B. Hayes	1822	10	4	4 października 1822	69 545	190	Piątek
26	James A. Garfield	1831	11	19	19 listopada 1831	66 212	181	Sobota
27	Chester A. Arthur	1829	10	5	5 października 1829	66 987	183	Poniedziałek
28	Grover Cleveland	1837	3	18	18 marca 1837	64 266	175	Sobota

Rysunek 2.18. Dodatek Extended Date Functions pozwala na przetwarzanie dat wcześniejszych niż rok 1900



Więcej informacji na temat rozszerzonych funkcji przetwarzania dat znajdziesz w rozdziale 8.

Tworzenie megaformuł

W celu wygenerowania wyniku arkusze kalkulacyjne często wymagają zastosowania formuł pośrednich. Innymi słowy, dana formuła może być zależna od innych formuł, które z kolei opierają się na jeszcze innych formułach. Gdy już wszystkie te formuły poprawnie działają, często można wyeliminować formuły pośrednie i zamiast nich zastosować pojedynczą megaformułę. Jakie uzyskuje się korzyści? Używasz mniejszej liczby komórek (a co za tym idzie, zwiększa się przejrzystość arkusza), plik skoroszytu jest zazwyczaj mniejszy, a ponadto ponowne przeliczanie arkusza może być wykonywane

szycie. Główną wadą takiego rozwiązań jest stopień złożoności takich formuł, które mogą łatwo stać się zupełnie zagmatwane i niemal niemożliwe do „rozszyfrowania” czy modyfikacji.

A oto przykład. Wyobraź sobie arkusz zawierający kolumnę z danymi osobowymi tysięcy osób. Założymy, że zostałeś poproszony o usunięcie z danych drugiego imienia i drugiego inicjału. Jednak nie wszystkie personalia zawierają drugie imię lub inicjał. Ręczne przetwarzanie takiego arkusza zajęłoby wiele godzin i nawet zamiana danych tekstowych na tabelę niewiele by tutaj pomogła; co więcej, także polecenie *Tekst jako kolumny*, znajdujące się w grupie *Narzędzia danych* na karcie *DANE*, nie będzie w takiej sytuacji zbyt przydatne. Z tego powodu zdecydowałeś się na rozwiązanie wykorzystujące odpowiednie formuły. Nie jest to zbyt trudne zadanie, ale zwykle wiąże się z użyciem kilku formuł pośrednich.



Zastosowanie nowego mechanizmu błyskawicznego wypełniania (ang. *Flash Fill*), zaimplementowanego w Excelu 2013, jest innym sposobem wykonania takiego zadania.

Na rysunku 2.19 pokazano wyniki zastosowania nieco bardziej konwencjonalnego rozwiązania, w przypadku którego wymagane było użycie sześciu formuł pośrednich przedstawionych w tabeli 2.6. Personalia znajdują się w kolumnie A, natomiast wyniki w kolumnie H. W kolumnach od B do G są przechowywane formuły pośrednie.

	A	B	C	D	E	F	G	H	I
1	Dane	Formula-1	Formula-2	Formula-3	Formula-4	Formula-5	Formula-6	Wynik	
2	Bogdan Nowak	Bogdan Nowak	7	#ARG!	7	Bogdan	Nowak	Bogdan Nowak	
3	Michał A. Jarocki	Michał A. Jarocki	7	10	10	Michał	Jarocki	Michał Jarocki	
4	Jerzy Roman Jop	Jerzy Roman Jop	6	12	12	Jerzy	Jop	Jerzy Jop	
5	Tomasz Aleksy Jarosz	Tomasz Aleksy Jarosz	7	14	14	Tomasz	Jarosz	Tomasz Jarosz	
6	Jan A. Petelski	Jan A. Petelski	4	7	7	Jan	Petelski	Jan Petelski	
7	R.J Siwiec	R.J Siwiec	4	#ARG!	4	R.J	Siwiec	R.J Siwiec	
8	R. Jan Szyman	R. Jan Szyman	3	7	7	R.	Szyman	R. Szyman	
9	Adam Kowalski	Adam Kowalski	5	#ARG!	5	Adam	Kowalski	Adam Kowalski	
10									

Rysunek 2.19. Usunięcie drugich imion i inicjałów wymaga zastosowania kilku formuł pośrednich

Tabela 2.6. Formuły pośrednie umieszczone w drugim wierszu arkusza pokazanego na rysunku 2.19

Kolumna	Formula pośrednia	Przeznaczenie
B	=USUŃ.ZBĘDNE.ODSTĘPY(A2)	Usuwa zbędne spacje.
C	=ZNAJDŹ(" ";B2;1)	Lokalizuje pierwszą spację.
D	=ZNAJDŹ(" ";B2;C2+1)	Lokalizuje drugą spację. Jeśli takiej spacji nie znajdzie, zwraca błąd #ARG!.
E	=JEŻELI(CZY.BŁĄD(D2);C2;D2)	Jeśli druga spacja nie istnieje, używa pierwszej.
F	=LEWY(B2;C2)	Zwraca imię.
G	=PRAWY(B2:DŁ(B2)-E2)	Zwraca nazwisko.
H	=F2&G2	Łączy imię z nazwiskiem.

Poprzez utworzenie megaformuły można wyeliminować wszystkie formuły pośrednie. W tym celu należy utworzyć wszystkie formuły pośrednie, a następnie przejść do formuły generującej ostateczny wynik i w miejscu każdego odwołania do kolejnej komórki wstawić kopię formuły znajdującej się w komórce, do której stworzono odwołanie (bez znaku równości). Na szczęście do kopiowania i wklejania można użyć schowka. Taką operację powinieneś powtarzać do momentu, gdy w komórce H2 będą się znajdowały jedynie odwołania do komórki A2. Po zakończeniu cyklu w ostatniej komórce znajdziesz się następująca megaformuła:

```
=LEWY(USUŃ.ZBĘDNE.ODSTĘPY(A2);ZNAJDŹ(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);1))  
↳&PRAWY(USUŃ.ZBĘDNE.ODSTĘPY(A2);DŁ(USUŃ.ZBĘDNE.ODSTĘPY(A2))-JEŻELI(CZY.BŁĄD(ZNAJDŹ  
↳(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);ZNAJDŹ(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);1)+1));  
↳ZNAJDŹ(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);1);ZNAJDŹ(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);  
↳ZNAJDŹ(" "; USUŃ.ZBĘDNE.ODSTĘPY(A2);1)+1)))
```

Gdy uznasz, że megaformuła działa poprawnie, możesz usunąć kolumny z formułami pośrednimi, ponieważ nie będą już potrzebne.

Megaformuła wykonuje dokładnie takie same operacje jak poszczególne, użyte wcześniej formuły pośrednie — aczkolwiek prześledzenie sposobu działania megaformuły jest prawie niemożliwie (nawet dla autora). Zanim zaczniesz tworzyć megaformułę, sprawdź, czy formuły pośrednie wykonują poprawne obliczenia. Jeszcze lepszym rozwiązaniem jest przechowywanie kopii formuł pośrednich w innym miejscu na wypadek wykrycia błędu lub w razie konieczności dokonania modyfikacji.

Innym sposobem rozwiązania takiego zadania jest utworzenie własnej funkcji arkuszowej w języku VBA, dzięki czemu będziesz mógł zastąpić złożoną megaformułę pojedynczą, prostą funkcją, na przykład:

```
=NOMIDDLE(A1)
```

Tak naprawdę utworzyłem funkcję NOMIDDLE po to, aby porównać efektywność jej działania z formułami pośrednimi i megaformułami. Kod funkcji wygląda następująco:

```
Function NOMIDDLE(n) As String  
    Dim FirstName As String, LastName As String  
    n = Application.WorksheetFunction.Trim(n)  
    FirstName = Left(n, InStr(1, n, " "))  
    LastName = Right(n, Len(n) - InStrRev(n, " "))  
    NOMIDDLE = FirstName & LastName  
End Function
```



Skoroszyt z tym przykładem (*Megaformuła.xlsxm*), zawierający formuły pośrednie, megaformułę oraz funkcję NOMIDDLE VBA, znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Ze względu na tak dużą złożoność megaformuły możesz dojść do wniosku, że jej użycie spowolni wykonywanie ponownych obliczeń, ale tak się nie dzieje. Aby to sprawdzić, utworzyłem arkusz, który używa megaformuły do przetwarzania 175 000 wierszy. Następnie utworzyłem kolejny arkusz, korzystający z sześciu formuł pośrednich. Porównanie czasów przetwarzania i rozmiarów skoroszytów znajdziesz poniżej:

- Formuły pośrednie: czas przeliczania 5,8 sekundy; rozmiar pliku 12,60 MB
- Megaformuła: czas przeliczania 3,9 sekundy; rozmiar pliku 2,95 MB

Wyniki testów na różnych komputerach mogą znaczco różnić się od siebie i zależą głównie od wydajności procesora, ilości zainstalowanej pamięci operacyjnej oraz budowy samej formuły.

Funkcja napisana w języku VBA była znacznie wolniejsza — po upływie 5 minut przewałem testowanie. Tak mała wydajność to dosyć typowe zachowanie dla funkcji napisanych w języku VBA — zawsze są wolniejsze od wbudowanych funkcji Excela.

Rozdział 3.

Pliki programu Excel

W tym rozdziale:

- Uruchamianie programu Excel
- Otwieranie i zapisywanie plików w różnych formatach
- Omówienie formatu plików używanych przez Excela
- Jak Excel korzysta z rejestrów systemu Windows

Uruchamianie Excela

Excel może być uruchamiany na różne sposoby w zależności od tego, jak został zainstalowany. Możesz kliknąć ikonę programu znajdująca się na pulpicie, użyć menu *Start* czy też po prostu dwukrotne kliknąć lewym przyciskiem myszy plik powiązany z Excellem (na przykład plik skoroszytu). Wszystkie metody ostatecznie powodują załadowanie pliku wykonywalnego o nazwie *excel.exe*.

Podczas uruchamiania Excel 2013 wykonuje następujące operacje:

- Odczytuje ustawienia przechowywane w rejestrze systemu Windows.
- Odczytuje i aplikuje wszelkie niestandardowe konfiguracje paska narzędzi *Szybki dostęp* oraz Wstäżki, przechowywane w pliku *Excel.officeUI*.
- Otwiera plik **.xlb*, w którym przechowywane są informacje o modyfikacjach menu i pasków narzędzi.
- Otwiera listy automatycznej korekty i pliki **.ACL*, o ile istnieją.
- Otwiera wszystkie zainstalowane dodatki (czyli takie, które zostały uaktywnione w oknie dialogowym *Dodatki*).
- Otwiera wszystkie skoroszyty przechowywane w folderze *XLStart*.
- Otwiera skoroszyt makr osobistych (*personal.xlsb*), o ile istnieje.
- Otwiera wszystkie skoroszyty przechowywane w alternatywnym folderze startowym (zdefiniowanym na karcie *Zaawansowane* okna dialogowego *Opcje programu Excel*).

- Sprawdza, czy poprzednia sesja pracy z Exceliem nie zakończyła się przedwcześnie (na przykład na skutek zawieszenia się systemu). Jeżeli tak, to Excel odczytuje i wyświetla listę automatycznie odzyskanych skoroszytów.
- Tworzy nowy, pusty skoroszyt — o ile wcześniej nie zostały otwarte inne skoroszyty zdefiniowane przez użytkownika bądź przechowywane w folderze *XLStart* lub alternatywnym folderze startowym.

Excel może być zainstalowany w dowolnym katalogu, ale w większości przypadków plik wykonywalny Excela możesz znaleźć w domyślnym katalogu instalacyjnym dla systemu 64-bitowego:

C:\Program Files (x86)\Microsoft Office\Office15\EXCEL.EXE

lub dla systemu 32-bitowego:

C:\Program Files\Microsoft Office\Office15\EXCEL.EXE

Aby sprawdzić ścieżkę, gdzie Excel został zainstalowany, możesz wykonać następujące polecenie VBA:

```
MsgBox Application.Path
```

W przeciwieństwie do poprzednich wersji Excel 2013 ma nowy, jednodokumentowy interfejs użytkownika. Inaczej mówiąc, każdy skoroszyt jest teraz otwierany w osobnym, własnym oknie aplikacji, ma własną Wstążkę i jest traktowany jako osobne zadanie. W poprzednich wersjach Excela wszystkie skoroszyty były otwierane w jednym oknie aplikacji (interfejs wielodokumentowy).

Z punktu widzenia deweloperów tworzących aplikacje VBA interfejs jednodokumentowy zmienia sposób działania niemodalnych formularzy *UserForms* oraz niestandardowych menu podręcznych. Zagadnienia z tym związane zostały omówione w dalszej części tej książki.



Jeżeli podczas uruchamiania Excela będziesz trzymał wciśnięty klawisz *Ctrl*, program zostanie uruchomiony w trybie awaryjnym (ang. *Safe Mode*). Taki tryb pracy Excela jest wykorzystywany głównie do diagnozowania przyczyn awarii i błędów, które pojawiają się podczas normalnego uruchamiania Excela.

Formaty plików

Domyślnym typem pliku programu Excel 2013 jest plik skoroszytu w formacie *XLSX*, ale Excel nadal potrafi również otwierać i zapisywać pliki w wielu innych formatach. W tym podrozdziale dokonamy szybkiego przeglądu typów plików, które Excel 2013 potrafi obsługiwać.



Począwszy od wersji 2007, Excel nie obsługuje już plików zapisanych w formatach arkuszy kalkulacyjnych Lotus ani Quattro Pro.

Formaty plików obsługiwane w programie Excel

W wersji 2007 programu Excel wprowadzono nowy, domyślny format zapisu skoroszytów i taki sam format jest używany w wersjach Excel 2010 i Excel 2013. Nie zmienia to jednak w niczym faktu, że najnowszy Excel nadal może otwierać i zapisywać pliki skoroszytów w formatach używanych przez poprzednie wersje programu.



Aby zmienić domyślny typ zapisywanych plików, przejdź na kartę *PLIK* i z menu wybierz polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Zapisywanie* i z listy rozwijanej *Zapisz pliki w następującym formacie* wybierz nowy, domyślny format zapisywania plików.

W tabeli 3.1 przedstawiono zestawienie typów plików obsługiwanych w programie Excel 2010. Pamiętaj, że zarówno pliki skoroszytów, jak i dodatków Excela mogą mieć dowolne rozszerzenia — innymi słowy, do zapisywania takich plików nie musisz używać rozszerzeń przedstawionych w poniższej tabeli. Warto jednak zauważyć, że podczas próby otwarcia pliku, którego rozszerzenie nie odpowiada zawartości, Excel może wyświetlić na ekranie odpowiednie ostrzeżenie.

Tabela 3.1. Typy plików obsługiwanych w programie Excel

Typ pliku	Rozszerzenie	Odczytywanie/ zapisywanie	Uwagi
<i>Skoroszyt programu Excel</i>	<i>.xlsx</i>	Tak/Tak	Domyślny format plików skoroszytów Excela. Nie może przechowywać makr VBA ani XLM. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.
<i>Skoroszyt programu Excel z obsługą makr</i>	<i>.xlsm</i>	Tak/Tak	Domyślny format plików skoroszytów Excela, w którym można przechowywać makra VBA i XLM. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.
<i>Skoroszyt binarny programu Excel</i>	<i>.xlsb</i>	Tak/Tak	Binarny format plików skoroszytów Excela. Jest to zaktualizowana wersja poprzedniego formatu XLS. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.
<i>Szablon programu Excel</i>	<i>.xltx</i>	Tak/Tak	Domyślny format szablonów Excela. Nie może przechowywać makr VBA ani XLM. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.
<i>Szablon programy Excel z obsługą makr</i>	<i>.xltxm</i>	Tak/Tak	Domyślny format szablonów Excela, w którym można przechowywać makra VBA i XLM. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.
<i>Dodatek programu Excel</i>	<i>.xlam</i>	Tak/Tak	Domyślny format dodatków Excela, w którym można przechowywać makra VBA i XLM. Kompatybilny z Exceliem 2007 i wersjami późniejszymi.

Tabela 3.1. Typy plików obsługiwanych w programie Excel (ciąg dalszy)

Typ pliku	Rozszerzenie	Odczytywanie/ zapisywanie	Uwagi
<i>Skoroszyt programu Excel 97-2003</i>	<i>.xls</i>	Tak/Tak	Binarny format plików skoroszytów (BIFF8), kompatybilny z Exceliem 97 i wersjami późniejszymi.
<i>Szablon programu Excel 97-2003</i>	<i>.xlt</i>	Tak/Tak	Binarny format plików szablonów (BIFF8), kompatybilny z Exceliem 97 i wersjami późniejszymi.
<i>Dodatek programu Excel 97-2003</i>	<i>.xla</i>	Tak/Tak	Binarny format plików dodatków (BIFF8), kompatybilny z Exceliem 97 i wersjami późniejszymi.
<i>Skoroszyt Microsoft Excel 5.0/95</i>	<i>.xls</i>	Tak/Tak	Binarny format plików skoroszytów (BIFF5), kompatybilny z Exceliem 5.0 i wersjami późniejszymi.
<i>Arkusz kalkulacyjny XML 2003</i>	<i>.xml</i>	Tak/Tak	Format skoroszytów Microsoft XML 2003 (XMLSS).



Użytkownicy pakietów Microsoft Office XP oraz Microsoft Office 2003 mogą zainstalować *Pakiet zgodności formatu plików pakietu Microsoft Office*, który pozwala na otwieranie i zapisywanie dokumentów w formatach używanych przez pakiet Office 2007 i wersje późniejsze. *Pakiet zgodności formatu plików pakietu Microsoft Office* możesz pobrać ze strony <http://office.microsoft.com>.

Formaty plików tekstowych

Gdy spróbujesz otworzyć w Excelu plik tekstowy, na ekranie może pojawić się okno *Kreatora importu tekstu*, który pomoże w określeniu sposobu załadowania pliku.



Aby pominąć *Kreatora importu tekstu*, należy przed kliknięciem *OK* w oknie dialogowym *Otwieranie* nacisnąć i przytrzymać klawisz *Shift*.

W tabeli 3.2 przedstawiono zestawienie typów plików tekstowych obsługiwanych w programie Excel 2013. W plikach tekstowych można przechowywać tylko dane z jednego arkusza.

Kiedy Excel nie potrafi otworzyć pliku

Jeżeli Excel nie potrafi otworzyć pliku zapisanego w określonym formacie, nie poddawaj się tak łatwo. Istnieje bardzo duże prawdopodobieństwo, że ktoś inny kiedyś spotkał się już z takim problemem. Spróbuj poszukać w sieci Internet informacji na temat plików o takim rozszerzeniu i dodać do wzorca wyszukiwania słowo *excel*. Być może jest gdzieś dostępny odpowiedni konwerter formatów plików, lub być może ktoś już odkrył inny sposób na zaimportowanie zawartości takiego pliku do Excela.

Tabela 3.2. Formaty plików tekstowych obsługiwanych przez Excela

Typ pliku	Rozszerzenie	Odczytywanie/ zapisywanie	Uwagi
CSV (rozdzielany przecinkami)	csv	Tak/Tak	Kolumny danych są rozdzielane przecinkami, a wiersze znakami powrotu karetki. Excel może eksportować pliki CSV w subformacie Macintosh lub MS-DOS.
Tekst z formatowaniem	prn	Tak/Tak	Kolumny danych są rozdzielane znakami spacji, a wiersze znakami powrotu karetki.
Tekst	txt	Tak/Tak	Kolumny danych są rozdzielane znakami tabulacji, a wiersze znakami powrotu karetki. Excel może eksportować pliki tekstowe w subformacie Macintosh, MS-DOS lub Unicode.
DIF (Format wymiany danych)	dif	Tak/Tak	Format plików używany początkowo przez program VisiCalc.
SYLK (Łącze symboliczne)	slk	Tak/Tak	Format plików używany początkowo przez program Multiplan.

Formaty plików baz danych

W tabeli 3.3 zamieszczono zestawienie typów plików baz danych obsługiwanych przez Excela 2013. W plikach baz danych można przechowywać tylko dane z jednego arkusza.

Tabela 3.3. Formaty plików baz danych obsługiwanych przez Excela

Typ pliku	Rozszerzenie	Odczytywanie/ zapisywanie	Uwagi
Access	mdb, mde, accdb, accde	Tak/Nie	W Excelu możesz otworzyć wybraną jedną tabelę bazy danych.
dBASE	dbf	Tak/Nie	Format plików opracowany przez firmę Ashton-Tate, twórcę baz danych dBASE.
Inne	Różne	Tak/Nie	Poprzez użycie poleceń znajdujących się w grupie <i>Dane zewnętrzne</i> na karcie <i>DANE</i> możesz importować dane z różnych źródeł zewnętrznych.

Inne formaty plików

W tabeli 3.4 zamieszczono zestawienie innych formatów plików baz danych obsługiwanych przez Excela 2013.

Tabela 3.4. Inne typy plików obsługiwane przez Excela

Typ pliku	Rozszerzenie	Odczytywanie/ zapisywanie	Uwagi
Plik HTML (<i>Hypertext Markup Language</i>)	<i>htm, html</i>	Tak/Tak	Począwszy od Excela 2007, format HTML już nie obsługuje „podróży danych w obie strony” ¹ . Jeżeli zapiszesz dane w tym formacie, a następnie ponownie otworzysz taki plik, musisz się liczyć z tym, że pewne informacje mogą zostać utracone.
Archiwum sieci Web, osobny plik	<i>mht, mhtml</i>	Tak/Tak	Znane również jako pliki archiwum stron sieci Web. Pliki zapisane w tym formacie mogą być odczytywane tylko w przeglądarce Microsoft Internet Explorer lub Opera ² .
Arkusz kalkulacyjny <i>OpenDocument</i>	<i>ods</i>	Tak/Tak	Format pliku opracowany przez firmy Sun Microsystems oraz OASIS. Pozwala na odczytywanie i zapisywanie skoroszytów w formacie takich pakietów jak na przykład OpenOffice.
Plik PDF (<i>Portable Document Format</i>)	<i>pdf</i>	Nie/Tak	Format pliku opracowany przez firmę Adobe.
Plik XPS (<i>XML Paper Specification</i>)	<i>xps</i>	Nie/Tak	Alternatywa firmy Microsoft dla plików PDF firmy Adobe.

Pliki obszaru roboczego

Plik obszaru roboczego (ang. *workspace file*) jest specjalnym plikiem zawierającym informacje na temat obszaru roboczego Excela. Jeżeli na przykład stworzyłeś projekt korzystający z dwóch skoroszytów i chciałbyś mieć ustawione ich okna w określony sposób, możesz zapisać ich układ w specjalnym pliku *XLW*. Później po każdym otwarciu takiego pliku *XLW* Excel przywróci żądany obszar roboczy.

Aby zapisać plik obszaru roboczego, przejdź na kartę *WIDOK*, naciśnij przycisk *Zapisz obszar roboczy*, znajdujący się w grupie opcji *Okno*, i kiedy Excel o to poprosi, podaj nazwę zapisywanej pliku.

Aby otworzyć plik obszaru roboczego, przejdź na kartę *PLIK*, z menu wybierz polecenie *Otwórz* i w oknie dialogowym *Otwieranie* z listy rozwijanej *Pliki typu* wybierz opcję *Obszary robocze (*.xlw)*.

Pamiętaj, że pliki obszaru roboczego *nie* zawierają żadnych skoroszytów, a jedynie informacje konfiguracyjne o układzie i widoczności okien skoroszytów. A zatem jeżeli zamierzasz przesyłać konfigurację obszaru roboczego komuś innemu, nie zapomnij o wysłaniu wraz z plikiem *XLW* odpowiednich skoroszytów.

¹ Oznacza to, że nie można zapisać dokumentu w formacie HTML i następnie odczytać go w Excelu bez utraty funkcjonalności skoroszytu — *przyp. tłum.*

² ...a także w przeglądarce Firefox, po zainstalowaniu np. dodatku UnMHT — *przyp. tłum.*

Kompatybilność plików Excela

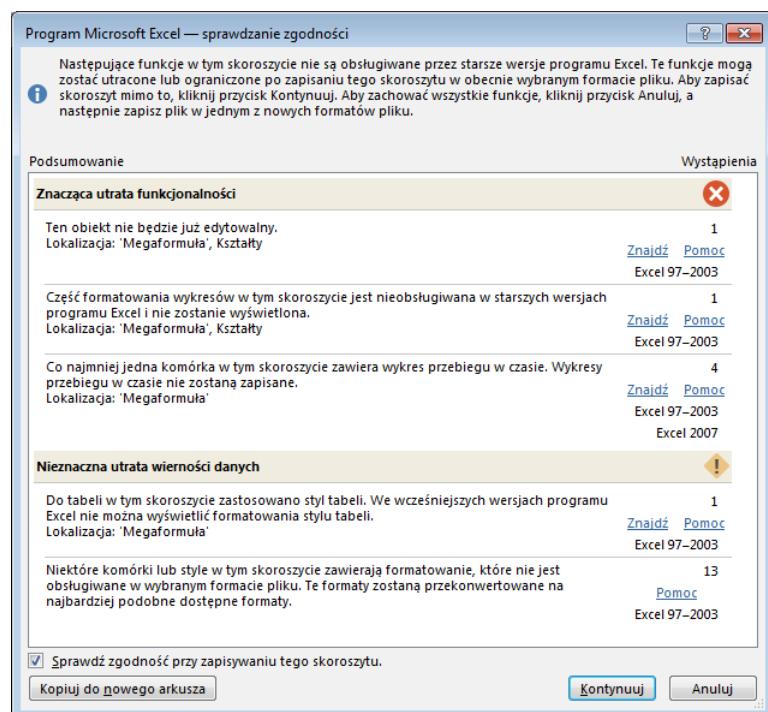
Bardzo ważną sprawą jest zrozumienie ograniczeń, jakie narzuca zagadnienie kompatybilności plików między różnymi wersjami Excela. Nawet jeżeli inny użytkownik będzie w stanie otworzyć skoroszyt, który mu podesłałeś, to nie masz żadnej gwarancji, że wszystko będzie działać poprawnie lub że wszystko będzie wyglądało dokładnie tak, jak w Twoim systemie.

Jeżeli spróbujesz zapisać skoroszyt w jednym ze starszych formatów plików (takich jak *XLS* dla wersji Excela wcześniejszych niż 2007), to Excel automatycznie uruchomi kreatora sprawdzania zgodności. Spróbuje on zidentyfikować wszystkie elementy arkusza, które zostaną utracone (lub których funkcjonalność albo wygląd zostaną w większym czy mniejszym stopniu zredukowane) na skutek zapisania w takim formacie.

Na rysunku 3.1 przedstawiono wygląd okna kreatora sprawdzania zgodności. Aby ograniczyć sprawdzanie kompatybilności formatu pliku do wybranej wersji Excela, naciśnij przycisk *Wybierz wersję do pokazania*.

Rysunek 3.1.

Kreator sprawdzania zgodności jest bardzo użytecznym narzędziem dla tych, którzy chcą udostępniać swoje skoroszyty innym użytkownikom



W głównej części okna kreatora sprawdzania zgodności jest wyświetlana lista zidentyfikowanych potencjalnych problemów ze zgodnością skoroszytu. Aby wyświetlić ją w nieco bardziej przyjaznym dla użytkownika formacie, naciśnij przycisk *Kopiuj do nowego arkusza*.

Pamiętaj, że problemy z kompatybilnością mogą wystąpić nawet pomiędzy Exceliem 2007 a Exceliem 2010, i to pomimo tego, że obie wersje używają tego samego formatu plików co Excel 2013. Nie możesz również oczekiwać, że nowe mechanizmy i funkcje wprowadzone w Excelu 2013 będą działać w poprzednich wersjach. Na przykład jeżeli w danym skoroszycie korzystasz z fragmentatorów (ang. *Slicers*), które są zupełnie nowym mechanizmem zaimplementowanym w Excelu 2013, i następnie wyślesz taki arkusz do kolegi korzystającego z Excela 2010, to oczywiście nie będą one u niego wyświetlane. Oprócz tego wszystkie formuły w skoroszycie, które wykorzystują nowe funkcje arkuszowe, w starszych wersjach Excela będą wyświetlały błędy. Na szczęście kreator sprawdzania zgody jest w stanie zidentyfikować takie problemy i wyświetlić odpowiednie ostrzeżenie.

Widok chroniony

Już w Excelu 2010 został wprowadzony nowy mechanizm bezpieczeństwa, znany jako *Widok chroniony*. Choć na pierwszy rzut oka wydawać by się mogło, że Excel próbuje uniemożliwić Ci otwieranie Twoich własnych plików, to jednak prawdziwym przeznaczeniem tego mechanizmu jest ochrona przed złośliwym oprogramowaniem. Termin *złośliwe oprogramowanie* (ang. *Malware*) odnosi się do wszystkich programów, które w jakiś sposób mogłyby wyrządzić „krzywdę” Twóemu systemowi lub danym znajdującym się na dyskach. Hakerzy już dawno znaleźli sposoby na wprowadzenie do plików Excela takich modyfikacji, aby po ich otwarciu automatycznie wykonywał się ukryty głęboko w skoroszytach złośliwy kod. Widok chroniony generalnie zapobiega tego typu atakom poprzez otwieranie „podejrzanych” plików w środowisku chronionym, czyli tzw. *piaskownicy* (ang. *Sandbox*).

Jeżeli otwierasz skoroszyt Excela, który został pobrany z sieci Internet, na pasku tytułowym okna skoroszytu pojawi się odpowiedni komunikat. Oprócz tego Excel na pasku statusu wyświetli *[Widok chroniony]*. Aby dowiedzieć się, dlaczego dany plik został otwarty w widoku chronionym, przejdź na kartę *PLIK* i wybierz polecenie *Informacje*.

Jeżeli jesteś pewny, że plik jest całkowicie bezpieczny, naciśnij przycisk *Włącz edytowanie*. Jeżeli nie zgodzisz się na edycję pliku, będziesz mógł tylko przeglądać zawartość skoroszytu, ale nie będziesz mógł wprowadzać do niego żadnych zmian.

Jeżeli skoroszyt zawiera makra, na ekranie zobaczysz jeszcze inny komunikat: *OSTRZEŻENIE O ZABEZPIECZENIACH. Makra zostały wyłączone*. Jeżeli jesteś pewny, że makra w takim skoroszycie są bezpieczne, naciśnij przycisk *Włącz zawartość*.

Domyślnie widok chroniony jest włączany dla następujących rodzajów skoroszytów:

- Plików pobranych z sieci Internet.
- Załączników otwieranych z poziomu wiadomości poczty elektronicznej w programie Outlook.
- Plików otwieranych z potencjalnie niebezpiecznych lokalizacji, takich jak na przykład folder *Temporary Internet Files*.

- Plików, które są blokowane poprzez reguły blokowania plików (ang. *File Block Policy*), czyli mechanizm systemu Windows pozwalający administratorom na zdefiniowanie listy potencjalnie niebezpiecznych lub niepożądanych plików.
- Plików, które zostały podpisane cyfrowo, ale ważność certyfikatu już wygasła.

W niektórych sytuacjach nie masz potrzeby edytowania dokumentu i chcesz go tylko na przykład wydrukować. W takiej sytuacji powinieneś po otwarciu dokumentu w trybie chronionym przejść na kartę *PLIK*, wybrać opcję *Drukuj* i następnie nacisnąć przycisk *Zezwól na drukowanie*.

Zwróć uwagę, że możesz skopiować zakres komórek ze skoroszytu otwartego w trybie chronionym i następnie wkleić je do innego skoroszytu.

Co ciekawe, masz pewną kontrolę nad typami plików, których próba otwarcia będzie wymuszała włączenie widoku chronionego. Aby zmienić takie ustawienia, przejdź na kartę *PLIK*, wybierz polecenie *Opcje* i potem, w oknie dialogowym *Opcje programu Excel*, kliknij opcję *Centrum zaufania*. Następnie naciśnij przycisk *Ustawienia Centrum zaufania* i przejdź na kartę *Widok chroniony*.

Zastosowanie mechanizmu Autoodzyskiwania

Jeżeli korzystasz z komputerów już od pewnego czasu, to z pewnością przytrafiła Ci się sytuacja, w której w niezamierzony sposób utraciłeś jakieś dane. Mogło to być skutkiem na przykład tego, że zapomniałeś zapisać wprowadzone zmiany lub nagle zabrakło prądu, a Ty nie zdążyłeś zapisać edytowanego dokumentu. Być może było tak, że pracowałeś nad czymś, co nie wydawało się takie ważne, i po prostu skończyłeś pracę, nie zapisując pliku na dysku, a później okazało się, że to jednak *było* ważne. Mechanizm o nazwie *Autoodzyskiwanie*, wprowadzony już w Excelu 2010, może spowodować, że takie sytuacje będą zdarzać się zdecydowanie rzadziej.

Kiedy pracujesz w Excelu, Twój skoroszyt jest automatycznie co jakiś czas zapisywany na dysku — najczęściej nawet nie zdajesz sobie sprawy z tego, że taka operacja właśnie miała miejsce. Excel zapisuje również kopie takich skoroszytów, którym jeszcze nie nadałeś nawet nazwy i nigdy świadomie ich nie zapisał.

Mechanizm *Autoodzyskiwanie* działa na dwa sposoby:

- Kolejne wersje aktualnie edytowanego skoroszytu są zapisywane automatycznie i możesz je później przeglądać.
- Skoroszyty, które zamknąłeś bez zapisywania, są automatycznie zapisywane jako wersje robocze (ang. *draft versions*).

Odzyskiwanie poprzednich wersji bieżącego skoroszytu

Aby sprawdzić, czy zachowały się jakieś poprzednie wersje aktualnie edytowanego skoroszytu, przejdź na kartę *PLIK* i wybierz opcję *Informacje*. Listę poprzednich wersji skoroszytu

(o ile oczywiście istnieją) znajdziesz w sekcji *Wersje*. W niektórych przypadkach na liście znajdziesz więcej niż jedną automatycznie zisaną wersję, w innych zaś takie wersje nie będą w ogóle dostępne.

Aby otworzyć automatycznie zisaną wersję, wystarczy kliknąć jej nazwę lewym przyciskiem myszy. Pamiętaj, że otwarcie automatycznie zisanej wersji skoroszytu *nie powoduje* zastąpienia aktualnie edytowanej wersji tego skoroszytu. Dzięki temu możesz łatwo podjąć decyzję, którą wersję wolisz, lub po prostu skopiować z poprzedniej wersji usunięte dane i wkleić je do aktualnie edytowanej wersji skoroszytu.

Kiedy zamkñasz skoroszyt, automatycznie zapisywane wersje są usuwane.

Odzyskiwanie niezapisanych skoroszytów

Kiedy zamkñasz skoroszyt bez uprzedniego zapisania go na dysku, Excel pyta Cię, czy jesteś pewny, że chcesz wykonać taką operację. Jeżeli taki niezapisany skoroszyt miał już swoją automatycznie zisaną wersję roboczą, to okno dialogowe *Czy jesteś pewny?* Cię o tym poinformuje.

Aby odzyskać skoroszyt, który zamknąłeś bez uprzedniego zapisania go na dysku, przejdź na kartę *PLIK*, wybierz polecenie *Informacje*, kliknij przycisk *Zarządzaj wersjami* i z menu podręcznego wybierz polecenie *Odzyskaj niezapisane skoroszyty*. Na ekranie pojawi się okno dialogowe *Otwieranie*, w którym zostanie wyświetlona lista wszystkich aktualnie przechowywanych wersji roboczych skoroszytów. Możesz teraz spróbować je otworzyć i (jeżeli będziesz miał odrobinę szczęścia) znaleźć coś, co jest Ci potrzebne. Zwróć uwagę na fakt, że wszystkie automatycznie zisanane wersje robocze są przechowywane w formacie *XLSB*.

Wersje robocze są usuwane automatycznie albo po czterech dniach od zapisania, albo w chwili, kiedy taki skoroszyt zostanie otwarty do edycji (którekolwiek z tych zdarzeń nastąpi jako pierwsze).

Konfigurowanie mechanizmu Autoodzyskiwania

Domyślnie mechanizm Autoodzyskiwania zapisuje pliki co 10 minut. Możesz zmienić to ustawienie i ustawić dowolny interwał czasowy z zakresu od 1 do 120 minut.

Jeżeli pracujesz z dokumentami poufnymi, być może nie będziesz chciał, aby ich poprzednie wersje były automatycznie zapisywane na dysku Twojego komputera. W takiej sytuacji na karcie *Zapisywanie* okna dialogowego *Opcje programu Excel* znajdziesz odpowiednie ustawienia, które pozwolą Ci na całkowite wyłączenie tego mechanizmu dla wszystkich lub tylko dla wybranych skoroszytów.

Praca z plikami szablonów

Szablon to, jak sama nazwa wskazuje, model, na bazie którego tworzymy inny dokument. Szablon programu Excel to skoroszyt, którego używasz do tworzenia innych skoroszytów.

Jako plik szablonu (z rozszerzeniem *XLTX*) może zostać zapisany dowolny skoroszyt. Szablony są szczególnie przydatne w przypadku częstego tworzenia podobnych plików. Aby na przykład co miesiąc generować podobne raporty sprzedaży i przy okazji zaoszczędzić czas, należy utworzyć szablon przechowujący formuły i wykresy wymagane do wygenerowania raportu. Po stworzeniu plików w oparciu o nowy szablon wystarczy jedynie wprowadzić odpowiednie wartości.

Przeglądanie dostępnych szablonów

Excel daje Ci do dyspozycji bardzo wiele różnych szablonów. Aby przeglądać galerię szablonów, przejdź na kartę *PLIK* i z menu wybierz polecenie *Nowe*.



Lokalizacja folderu *Szablony* zmienia się w zależności od wersji Excela. Aby znaleźć położenie tego foldera, należy wykonać następującą instrukcję języka VBA:

```
MsgBox Application.TemplatesPath
```

Tworzenie szablonów

Excel obsługuje trzy rodzaje szablonów:

- **Domyślny szablon skoroszytu** — jest wykorzystywany jako baza dla nowych skoroszytów. Plik domyślnego szablonu skoroszytu nosi nazwę *book.xltx*.
- **Domyślny szablon arkusza** — jest wykorzystywany jako baza dla nowych arkuszy wstawianych do skoroszytu. Plik domyślnego szablonu arkusza nosi nazwę *sheet.xltx*.
- **Własne szablony skoroszytów** — zazwyczaj są to gotowe do użycia skoroszyty, zawierające odpowiednie formuły, ale w praktyce szablony mogą być zarówno bardzo proste, jak i bardzo złożone. Zazwyczaj szablony są tworzone w taki sposób, że po ich otwarciu użytkownik musi tylko wprowadzić odpowiednie dane i od razu otrzymuje żądaną rezultat.

Tworzenie szablonu

zmieniającego domyślne ustawienia skoroszytów

Każdy nowy skoroszyt posiada pewne predefiniowane ustawienia domyślne. Na przykład nowy skoroszyt posiada trzy arkusze, linie siatki są wyświetlane, domyślna czcionka to Calibri o rozmiarze 11 punktów, domyślna szerokość kolumny to 8,43 punktu i tak dalej. Jeżeli któryś z tych ustawień Ci nie odpowiada, możesz je w prosty sposób zmienić.

Wprowadzanie zmian domyślnych ustawień programu Excel jest zadaniem stosunkowo prostym i na przyszłość może Ci zaoszczędzić sporo czasu. Aby zmienić ustawienia domyślne, powinieneś wykonać polecenia opisane poniżej:

- Otwórz nowy skoroszyt.
- Dodaj lub usuń odpowiednią liczbę arkuszy, tak aby osiągnąć ich żądaną ilość.

- Dokonaj odpowiednich zmian i modyfikacji ustawień skoroszytu, takich jak szerokość kolumny, style formatowania, opcje ustawienia strony i wiele innych ustawień dostępnych w oknie *Opcje programu Excel* (kategoria *Zaawansowane*).

Aby zmienić domyślny sposób formatowania komórek, przejdź na kartę *NARZĘDZIA GŁÓWNE* i naciśnij przycisk *Style komórki*, znajdujący się w grupie opcji *Style*, a następnie zmień ustawienia stylu *Normalny* (na przykład możesz zmienić domyślny krój czcionki, rozmiar czcionki czy sposób formatowania liczb).

- Po zakończeniu wprowadzania modyfikacji przejdź na kartę *PLIK* i wybierz z menu polecenie *Zapisz jako*.
- Na ekranie pojawi się okno dialogowe *Zapisywanie jako*. Z listy rozwijanej *Zapisz jako typ* wybierz opcję *Szablon programu Excel (*.xlsx)*.
- W polu *Nazwa pliku* wpisz nazwę *book.xlsx*.
- Zapisz plik w folderze *\XLStart* (a nie w domyślnym folderze *Szablony*).
- Zamknij plik.



Aby określić lokalizację foldera *XLStart*, wykonaj następujące polecenie VBA:

```
MsgBox Application.StartupPath
```

Po wykonaniu poleceń opisanych powyżej nowy skoroszyt, który będzie się pojawiał po uruchomieniu Excela, będzie oparty na ustawieniach szablonu *book.xlsx*. Aby utworzyć nowy skoroszyt oparty na tym szablonie, możesz po prostu nacisnąć kombinację klawiszy *Ctrl+N*. Jeżeli będziesz chciał kiedyś powrócić do „fabrycznych”, domyślnych ustawień skoroszytu Excela, po prostu usuń plik *book.xlsx*.



Jeżeli z menu karty *PLIK* wybierzesz polecenie *Nowe* i następnie wybierzesz opcję *Pusty skoroszyt*, utworzony skoroszyt *nie będzie* oparty na szablonie *book.xlsx*. Szczerze mówiąc, nie wiem, czy wynika to z błędu Excela, czy jest po prostu działaniem zgodnym z intencjami programistów firmy Microsoft. Niezależnie jednak od przyczyny, takie zachowanie Excela pozwala na utworzenie nowego, pustego skoroszytu z pominięciem ustawień szablonu *book.xlsx*.

Zastosowanie szablonów do zmiany domyślnych ustawień arkusza

Kiedy wstawiasz nowy arkusz do skoroszytu, Excel tworzy go na bazie predefiniowanych ustawień domyślnych, obejmujących takie elementy jak szerokość kolumn, wysokość wierszy i tak dalej. Jeżeli któreś z tych ustawień domyślnych Ci nie odpowiada, możesz je w prosty sposób zmienić. Aby to zrobić, powinieneś wykonać polecenia opisane poniżej:

1. Utwórz nowy skoroszyt i usuń z niego wszystkie arkusze z wyjątkiem jednego.
2. Dokonaj odpowiednich zmian i modyfikacji ustawień skoroszytu, takich jak szerokość kolumny, style formatowania, opcje ustawienia strony i wiele innych ustawień dostępnych w oknie *Opcje programu Excel*.
3. Po zakończeniu wprowadzania modyfikacji przejdź na kartę *PLIK* i wybierz z menu polecenie *Zapisz jako*.

4. Na ekranie pojawi się okno dialogowe *Zapisywanie jako*. Z listy rozwijanej *Zapisz jako typ* wybierz opcję *Szablon programu Excel (*.xltx)*.
5. W polu *Nazwa pliku* wpisz nazwę *sheet.xltx*.
6. Zapisz plik w folderze *\XLStart* (a nie w domyślnym folderze *Szablony*).
7. Zamknij plik.
8. Zamknij i ponownie uruchom Excela.

Po wykonaniu poleceń opisanych powyżej wszystkie nowe arkusze wstawiane do skoroszytu za pomocą polecenia *Wstaw arkusz* (przycisk tego polecenia znajdziesz po prawej stronie ostatniej karty arkusza) będą sformatowane w oparciu o szablon *sheet.xltx*. Aby wstawić nowy arkusz, możesz również nacisnąć kombinację klawiszy *Shift+F11*.

Tworzenie szablonów skoroszytu

Szablony *book.xltx* oraz *sheet.xltx*, o których była mowa w poprzedniej sekcji, to dwa specjalne rodzaje szablonów odpowiedzialne za domyślne ustawienia odpowiednio nowych skoroszytów i nowych arkuszy. W tym podrozdziale omówimy rodzaje szablonów skoroszytów, określanych po prostu jako *szablony skoroszytów* użytkownika, które są tworzone jako baza dla dokumentów, z jakimi pracujesz na co dzień.

Po co używać szablonów skoroszytów? Odpowiedź jest prosta — po to, aby zaoszczędzić sobie konieczności ciągłego wykonywania tej samej pracy. Założmy, że tworzysz comiesięczny raport sprzedaży, składający się z informacji o poziomie sprzedaży w poszczególnych regionach oraz pewnej liczby obliczeń i wykresów. W takiej sytuacji możesz utworzyć szablon zawierający wszystkie wspomniane elementy z wyjątkiem konkretnych wartości. Teraz, kiedy nadziejdziesz czas utworzenia raportu, możesz po prostu utworzyć nowy skoroszyt oparty na szablonie raportu, wprowadzić aktualne dane i raport będzie gotowy.



Oczywiście w podobnym celu możesz wykorzystać po prostu raport z poprzedniego miesiąca i tylko zmienić odpowiednie dane, ale taki sposób postępowania jest bardzo podatny na powstawanie błędów, a co gorsza, jeżeli zapomnisz użyć polecenia *Zapisz jako* i zapisać nowy raport pod inną nazwą, możesz w niezamierzony sposób nadpisać raport z poprzedniego miesiąca i utracić tamte dane. Innym rozwiązaniem jest kliknięcie nazwy skoroszytu prawym przyciskiem myszy i wybranie z menu podręcznego polecenia *Otwórz kopię*. Wykonanie tego polecenia powoduje utworzenie nowego skoroszytu na bazie skoroszytu istniejącego i nadaje mu inną nazwę, dzięki czemu oryginalny skoroszyt pozostanie nienaruszony.

Aby utworzyć skoroszyt oparty na własnym szablonie, przejdź na kartę *PLIK*, kliknij polecenie *Nowy* i następnie kliknij kategorię *OSOBISTE* (jest zlokalizowana na górze okna, tuż pod polem *Wyszukaj szablon online*).

Kiedy tworzysz skoroszyt w oparciu o szablon, domyślna nazwa skoroszytu składa się z nazwy szablonu i kolejnego numeru skoroszytu. Jeśli na przykład tworzysz nowy skoroszyt oparty na szablonie *Raport sprzedaży.xltx*, to domyślną nazwą skoroszytu będzie *Raport sprzedaży1.xlsx*. Kiedy po raz pierwszy będziesz chciał zapisać skoroszyt utworzony na bazie szablonu, Excel wyświetli na ekranie okno dialogowe *Zapisywanie jako* i będziesz mógł nadać skoroszytowi nową nazwę.

Szablon skoroszytu użytkownika jest w zasadzie normalnym skoroszytem Excela, stąd możesz w nim używać wszystkich dostępnych elementów, takich jak na przykład wykresy, formuły czy makra. Zazwyczaj szablony są tworzone w taki sposób, że po ich otwarciu użytkownik musi tylko wprowadzić odpowiednie dane i od razu otrzymuje żądzany rezultat — innymi słowy, większość szablonów zawiera wszystko z wyjątkiem danych, które musi wprowadzić użytkownik.



Jeżeli szablon zawiera makra, musi zostać zapisany w formacie *Szablon programu Excel z obsługą makr*. Takie pliki szablonów mają rozszerzenie **.xlsm*.

Budowa plików programu Excel

Jak już wspominaliśmy, Excel 2007 i nowsze wersje do zapisywania swoich skoroszytów, szablonów i dodatków używają plików w formacie XML. Co ciekawe, tak naprawdę są to pliki w formacie archiwum ZIP, a zatem ich zawartość może zostać „wypakowana” i przeglądana „na zewnątrz”, po rozpakowaniu.

Wersje Excela wcześniejsze niż 2007 do zapisu dokumentów używały plików binarnych. Pomimo iż specyfikacja tych plików jest znana, to jednak praca z nimi nie należała do najłatwiejszych. Format dokumentów XML używany przez nowe wersje Excela jest formatem *otwartym*, co oznacza, że takie pliki mogą być również otwierane i przetwarzane przy użyciu innego oprogramowania.

Zaglądamy do wnętrza pliku

W tym podrozdziale omówimy elementy składowe, które możesz znaleźć wewnętrz typowego pliku *XLSM* programu Excel (czyli pliku skoroszytu z obsługą makr). Wygląd naszego przykładowego skoroszytu, o nazwie *przykład.xlsm*, został przedstawiony na rysunku 3.2. Skoroszyt składa się z jednego arkusza danych, jednego arkusza wykresu oraz prostego makra VBA. Na arkuszu znajduje się tabela, przycisk (formant formularza), diagram składający się z obiektów *SmartArt* oraz zdjęcia.



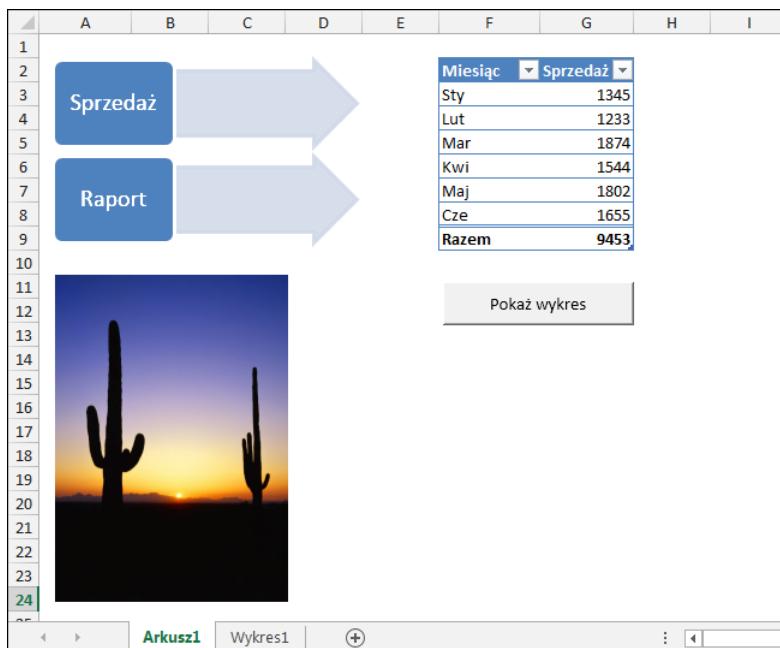
Skoroszyt z tym przykładem (*przykład.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Aby zjrzeć „pod maskę” pliku skoroszytu programu Excel (*XLSX* lub *XLSM*), będziesz musiał uruchomić program Windows Explorer, odszukać plik skoroszytu i dodać mu rozszerzenie ZIP. Po wykonaniu tej operacji zamiast pliku *przykład.xlsm* powinieneś mieć plik *przykład.xlsm.zip*. Teraz możesz otworzyć ten plik w dowolnym programie obsługującym archiwa ZIP, ale równie dobrze możesz po prostu skorzystać z obsługi plików ZIP wbudowanej w system Windows 7.



Jeżeli Twój system jest skonfigurowany tak, że rozszerzenia plików nie są domyślnie wyświetlane, powinieneś wyłączyć tę opcję. Aby to zrobić, uruchom Eksploratora Windows i z menu *Narzędzia* wybierz polecenie *Opcje folderów*. Na ekranie pojawi się okno dialogowe o tej samej nazwie. Przejdz na kartę *Widok* i usuń zaznaczenie opcji *Ukryj rozszerzenia znanych typów plików*.

Rysunek 3.2.
Przykład prostego skoroszytu



Innym rozwiązaniem może być wypakowanie zawartości archiwum do osobnego, nieskompresowanego foldera na dysku, co powinno zdecydowanie ułatwić przeglądanie poszczególnych plików składowych. Aby to zrobić, kliknij plik skoroszytu prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Wyodrębnić wszystkie*.

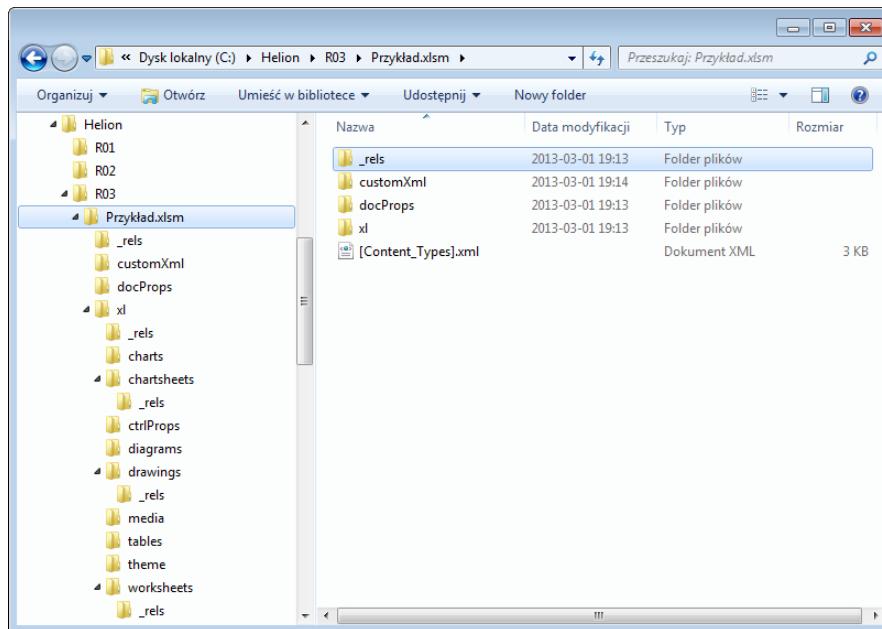
Pierwszą rzeczą, która rzuca się w oczy, jest to, że plik posiada strukturę katalogu. Na rysunku 3.3 w lewym panelu widać pełną strukturę wewnętrzną pliku skoroszytu. Oczywiście dokładna struktura pliku i jego zawartość będzie się różniła w zależności od skoroszytu.

Oprócz kilku wyjątków zdecydowana większość plików w skoroszycie to pliki tekstowe, a dokładniej mówiąc, pliki w formacie XML. Ich zawartość możesz przeglądać w dowolnym edytorze tekstu, edytorze XML, przeglądarce sieciowej, a nawet w Excelu. Na rysunku 3.4 przedstawiono zawartość jednego z plików, wyświetlzoną w oknie przeglądarki Internet Explorer. Pliki w formatach innych niż XML są powiązane z obiektami graficznymi i projektami VBA (które są zapisywane w formacie binarnym).

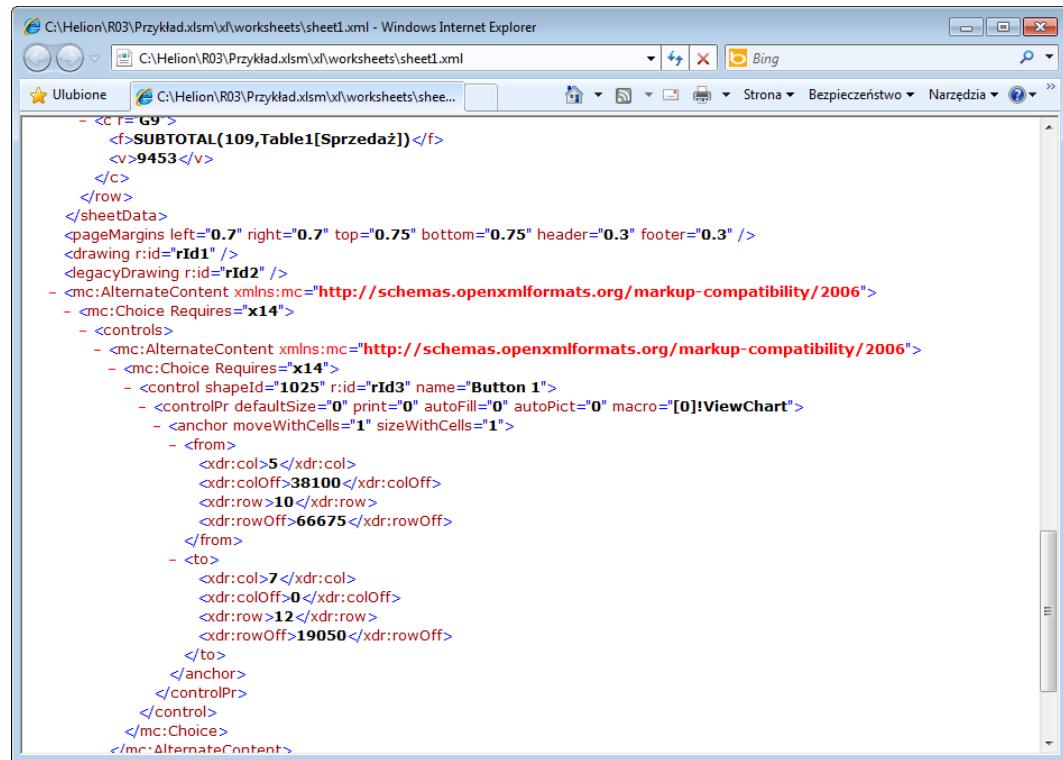
Ten plik *XML* posiada trzy foldery główne i kilka podfolderów. Zwrót uwagi, że wiele folderów w skoroszycie posiada podkatalog o nazwie *_rels*. W tych podkatalogach znajdują się pliki *XML*, które definiują powiązania danego elementu z innymi elementami składowymi pakietu.

Poniżej zamieszczono listę folderów, które możesz znaleźć w naszym skoroszycie *przykład.xlsxm*:

- ***_rels*** — zawiera informacje o relacjach pomiędzy elementami składowymi skoroszytu.
- ***customXml*** — zawiera informacje o rozszerzeniach Wstążki zapisanych w tym skoroszycie.



Rysunek 3.3. Wewnętrzna struktura katalogów pliku skoroszytu



Rysunek 3.4. Przeglądanie zawartości pliku XML w przeglądarce sieciowej

- **docProps** — zawiera plik *XML* opisujący właściwości plików i ustawienia aplikacji.
- **xl** — w tym folderze przechowywane są główne elementy składowe skoroszytu. Nazwa tego foldera zmienia się w zależności od aplikacji (*xl, ppt, word* i tak dalej). Znajdziesz tutaj kilka plików *XML* zawierających ustawienia skoroszytu, a jeżeli skoroszyt zawiera kod VBA, w tym folderze znajdziesz również odpowiedni plik binarny z rozszerzeniem *BIN*. W folderze *xl* znajdziesz kilka podkatalogów (ich nazwy i liczba mogą się zmieniać w zależności od zawartości skoroszytu):
 - **charts** — zawiera pliki *XML* opisujące wykresy znajdujące się w skoroszycie (po jednym pliku dla każdego wykresu). W plikach *XML* znajdują się ustawienia poszczególnych wykresów.
 - **chartsheets** — zawiera pliki *XML* z danymi dla wykresów umieszczonych na poszczególnych arkuszach wykresów.
 - **diagrams** — zawiera pliki *XML* opisujące diagramy (utworzone przy użyciu obiektów *SmartArt*) w skoroszycie.
 - **drawings** — zawiera pliki *XML* opisujące poszczególne „rysunki”, takie jak przyciski, wykresy czy obrazy.
 - **media** — zawiera osadzone pliki multimedialne, takie jak na przykład pliki *GIF* czy *JPEG*.
 - **tables** — zawiera pliki *XML* opisujące poszczególne tabele danych.
 - **theme** — zawiera plik *XML* z danymi o motywie skoroszytu.
 - **worksheets** — zawiera pliki *XML* opisujące poszczególne arkusze skoroszytu.



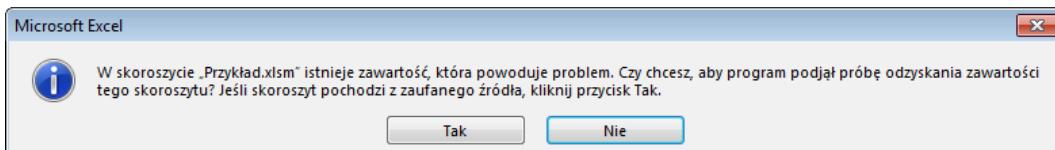
Jeżeli dodasz do pliku skoroszytu rozszerzenie *ZIP*, to i tak nadal będziesz mógł otworzyć taki plik w programie Excel (aczkolwiek na ekranie pojawi się ostrzeżenie o niezgodności rozszerzenia z zawartością pliku). W razie potrzeby możesz od razu zapisać skoroszyt na dysku w postaci pliku z rozszerzeniem *ZIP*. Aby to zrobić, po prostu dodaj w oknie dialogowym *Zapisywanie jako* do nazwy pliku rozszerzenie *ZIP* i następnie umieść całą nazwę w cudzysłowie, na przykład "MójSkoroszyt.xlsx.zip".

Dlaczego format pliku jest taki ważny?

Nowe, „otwarte” formaty *XML* plików wprowadzone w pakiecie Microsoft Office 2007 reprezentują znaczący postęp dla całej społeczności użytkowników komputerów. Po raz pierwszy w historii można względnie łatwo odczytywać i zapisywać skoroszyty Excela przy użyciu oprogramowania innego niż sam Excel. Na przykład możesz napisać program, który będzie modyfikował zawartość tysięcy skoroszytów Excela, bez potrzeby uruchamiania samego programu Excel. Taki program mógłby na przykład wstawać do każdego ze skoroszytów nowy arkusz. Programista tworzący taką aplikację musiałby oczywiście wykazać się dogłębną znajomością struktur *XML* plików skoroszytów, ale mimo to takie zadanie jest jak najbardziej do zrealizowania.

Co ważne, nowe formaty plików są znacznie mniej podatne na uszkodzenia (w porównaniu ze starymi formatami binarnymi). Aby to przetestować, zapisałem skoroszyt i następnie usunąłem z niego jeden z plików *XML* opisujących arkusze. Kiedy spróbowałem

otworzyć taki uszkodzony plik w Excelu, na ekranie pojawiło się okno z komunikatem, przedstawione na rysunku 3.5. Dzięki analizie danych zapisanych w plikach *.res* Excel był w stanie stwierdzić, że plik został uszkodzony. W tym przypadku Excel był w stanie naprawić zawartość pliku i poprawnie go otworzyć. Usunięty arkusz został wstawiony do skoroszytu, ale zniknęły znajdujące się na nim dane.



Rysunek 3.5. Excel bardzo często jest w stanie samodzielnie naprawić uszkodzone pliki

Kolejną zaletą nowego formatu plików Excela jest to, że spakowane pliki *XML* są zazwyczaj znacznie mniejsze niż odpowiadające im pliki binarne w starym formacie. Dodatkowo strukturalna budowa plików skoroszytu pozwala na wyodrębnianie z nich poszczególnych elementów (na przykład wszystkich elementów graficznych).

Typowy użytkownik Excela prawdopodobnie nigdy nie będzie musiał przeglądać ani modyfikować elementów składowych *XML* pliku skoroszytu. Programiści tworzący aplikacje Excela mogą jednak wykorzystywać tę wiedzę do utworzenia kodu modyfikującego Wstążkę, czyli graficzny interfejs użytkownika programu Excel. Aby można było to zrobić, dogłębiańska znajomość struktur *XML* pliku skoroszytu jest po prostu nieodzowna.



Więcej szczegółowych informacji na temat modyfikacji Wstążki Excela znajdziesz w rozdziale 20.

Plik *OfficeUI*

Informacje o zmianach konfiguracji paska narzędzi *Szybki dostęp* oraz Wstążki są zapisywane w pliku XML o nazwie *Excel.officeUI*, który jest przechowywany w następującym folderze:

C:\Users\<nazwa_użytkownika>\AppData\Local\Microsoft\Office

Plik *Excel.officeUI* jest aktualizowany za każdym razem, kiedy użytkownik zmienia konfigurację paska narzędzi *Szybki dostęp* lub Wstążki. Zmiany w pliku są dokonywane natychmiast, a nie tylko w momencie, kiedy Excel jest zamknięty. Plik *Excel.officeUI* domyślnie nie istnieje; jest tworzony dopiero w momencie, kiedy dokonasz pierwszej modyfikacji interfejsu użytkownika programu Excel.

Excel.officeUI jest plikiem w formacie *XML*, a zatem możesz przeglądać jego zawartość przy użyciu dowolnego edytora *XML*, przeglądarki sieciowej, a nawet samego Excela. Aby wyświetlić zawartość tego pliku w Excelu, powinieneś wykonać następujące polecenia:

1. Utwórz kopię pliku *Excel.officeUI*.
2. Dodaj do nazwy kopii pliku rozszerzenie *XML*, aby jego nazwa wyglądała na przykład tak: *Excel.officeUI.xml*.

3. Przejdź na kartę *PLIK*, wybierz z menu polecenie *Otwórz* i wybierz plik. Zamiast tego możesz po prostu przeciągnąć go do okna Excela.
4. Na ekranie pojawi się okno dialogowe *Otwieranie pliku XML*, zawierające kilka opcji; wybierz opcję *Jako tabelę XML* i naciśnij przycisk *OK*.

Na rysunku 3.6 przedstawiono zawartość pliku *Excel.officeUI*, wyświetlanego w formie tabeli w Excelu. W tym przypadku na pasku narzędzi *Szybki dostęp* aktywnych jest pięć poleceń (wskaazywanych przez wartość TRUE w kolumnie B), a dodatkowo do Wstążki dodana została nowa karta, zawierająca nową grupę, w której zostały zdefiniowane pięć nowych poleceń (wiersze 14 – 18 w tabeli).

A	B	C	D	E	F	G	H	I	
1	idQ	visible	insertBeforeQ	id	label	insertBeforeQ2	id3	label4	autoScale
2	mso:FileNewDefault	FALSZ							
3	mso:FileOpenUsingBackstage	FALSZ							
4	mso:FileSave	PRAWDA							
5	mso:FileSendAsAttachment	FALSZ							
6	mso:FilePrintQuick	FALSZ							
7	mso:Spelling	FALSZ	mso:PrintPreviewAndPrint						
8	mso:Undo	PRAWDA	mso:PrintPreviewAndPrint						
9	mso:Redo	PRAWDA	mso:PrintPreviewAndPrint						
10	mso:PointerModeOptions	FALSZ	mso:PrintPreviewAndPrint						
11	mso:SortAscendingExcel	PRAWDA	mso:PrintPreviewAndPrint						
12	mso:SortDescendingExcel	PRAWDA	mso:PrintPreviewAndPrint						
13	mso:PrintPreviewAndPrint	FALSZ							
14			mso_c1.6C47E01	MOJA KARTA	mso:TabInsert	mso_c2.6C47E01	Narzędzia podręczne	PRAWDA	
15			mso_c1.6C47E01	MOJA KARTA	mso:TabInsert	mso_c2.6C47E01	Narzędzia podręczne	PRAWDA	
16			mso_c1.6C47E01	MOJA KARTA	mso:TabInsert	mso_c2.6C47E01	Narzędzia podręczne	PRAWDA	
17			mso_c1.6C47E01	MOJA KARTA	mso:TabInsert	mso_c2.6C47E01	Narzędzia podręczne	PRAWDA	
18			mso_c1.6C47E01	MOJA KARTA	mso:TabInsert	mso_c2.6C47E01	Narzędzia podręczne	PRAWDA	
19									
20									

Rysunek 3.6. Przeglądanie zawartości pliku *Excel.officeUI* w programie Excel

Plik konfiguracyjny *Excel.officeUI* możesz udostępniać innym użytkownikom. Jeżeli, na przykład, upakowałeś na pasku narzędzi *Szybki dostęp* ze dwiema tuzinami użytecznych poleceń, a dodatkowo utworzyłeś na Wstążce nową kartę, podzieliłeś na grupy i umieściłeś w nich cały szereg potrzebnych poleceń, i wreszcie to wszystko zrobiło ogromne wrażenie na Twoich kolegach z pracy, możesz po prostu przekazać im kopię swojego pliku *Excel.officeUI* i powiedzieć, gdzie mają go umieścić. Pamiętaj jednak, że zamiana istniejącego pliku *Excel.officeUI* na nowy spowoduje nadpisanie wszystkich modyfikacji, których Twoi koledzy mogli wcześniej dokonać.

Pamiętaj, nigdy nie próbuj samodzielnie modyfikować zawartości pliku *Excel.officeUI*, jeżeli nie jesteś całkowicie pewny, że wiesz, co robisz. Jeżeli Excel podczas uruchamiania zgłosi błąd w pliku *Excel.officeUI*, możesz po prostu usunąć ten plik, a Excel utworzy jego nową kopię. Jeszcze lepszym rozwiązaniem będzie oczywiście przechowywanie kopii bezpieczeństwa takiego pliku.

Plik **XLB**

Excel zapisuje konfigurację pasków narzędzi i menu w pliku *XLB*. Pomimo iż Excel 2013 oficjalnie nie obsługuje niestandardowych pasków narzędzi i menu użytkownika w sposób, jaki miał miejsce w poprzednich wersjach Excela, to w sytuacji, kiedy używasz dowolnej

aplikacji, która tworzy niestandardowe paski narzędzi bądź menu, nadal korzysta z pliku *XLB*. Jeżeli nie potrafisz odnaleźć pliku *XLB*, oznacza to po prostu, że Excel nie przechowuje do tej pory konfiguracji żadnych niestandardowych pasków narzędzi ani menu.

Po zamknięciu Excela aktualna konfiguracja pasków narzędzi jest przechowywana w pliku *Excel15.xlsb*. Plik ten (najczęściej) jest przechowywany w następującej lokalizacji:

C:\Users\<nazwa_ użytkownika>\AppData\Roaming\Microsoft\Excel

Ten plik binarny zawiera informacje o położeniu i widoczności wszystkich niestandardowych pasków narzędzi i menu wraz z opisem modyfikacji wbudowanych pasków narzędzi i menu.

Pliki dodatków

Dodatek jest plikiem skoroszytu różniącym się od standardowego skoroszytu następującymi — ważnymi — elementami:

- Właściwość *IsAddin* skoroszytu ma wartość *True* — co oznacza, że można go załadować przy użyciu okna dialogowego *Dodatki*.
- Skoroszyt dodatku jest ukryty i nie może zostać wyświetlony przez użytkownika — w konsekwencji dodatek może nigdy nie być aktywnym skoroszytem.
- Z punktu widzenia kodu VBA skoroszyt dodatku nie jest elementem kolekcji *Workbooks*.



Aby przywołać na ekran okno zarządzania dodatkami, przejdź na kartę *PLIK* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Dodatki*, następnie rozwiń listę *Zarządzaj* i wybierz z niej opcję *Dodatki programu Excel*. Jeżeli skonfigurowałeś wcześniej Excela tak, że na Wstążce widoczna jest karta *Deweloper*, to aby otworzyć okno dialogowe *Dodatki*, możesz po prostu przejść na kartę *Deweloper* i nacisnąć przycisk *Dodatki* znajdujący się w grupie opcji *Dodatki* (zamiast tego możesz nacisnąć sekwencję klawiszy *lewy Alt, Q, X*).

Dodatki wzbogacają Excel o nowe funkcje lub właściwości, z których można korzystać tak, jakby były wbudowane w program.

Własne dodatki możesz tworzyć w oparciu o pliki standardowych skoroszytów. W praktyce dodatki są preferowaną metodą dystrybucji wielu aplikacji przeznaczonych dla Excela. Domyślnie pliki dodatków programu Excel 2007 i nowszych wersji są zapisywane z rozszerzeniem *XLAM*.



Poza dodatkami o rozszerzeniu *.xlam* Excel obsługuje dodatki *XLL* i *COM*. Tego typu dodatki są tworzone przy użyciu innego oprogramowania niż Excel. W tej książce omówiono jedynie dodatki typu *XLAM*.



Więcej szczegółowych informacji na temat dodatków znajdziesz w rozdziale 19.

Ustawienia Excela w rejestrze systemu Windows

W oknie dialogowym *Opcje programu Excel* znajduje się kilka tuzinów opcji konfiguracyjnych Excela. Excel przechowuje te ustawienia w rejestrach systemu Windows i odczytuje je stamtąd podczas uruchamiania programu. W tym podrozdziale przedstawiono kilka podstawowych informacji na temat rejestrów systemu Windows i wyjaśniono, w jaki sposób Excel korzysta z niego do przechowywania własnych ustawień.

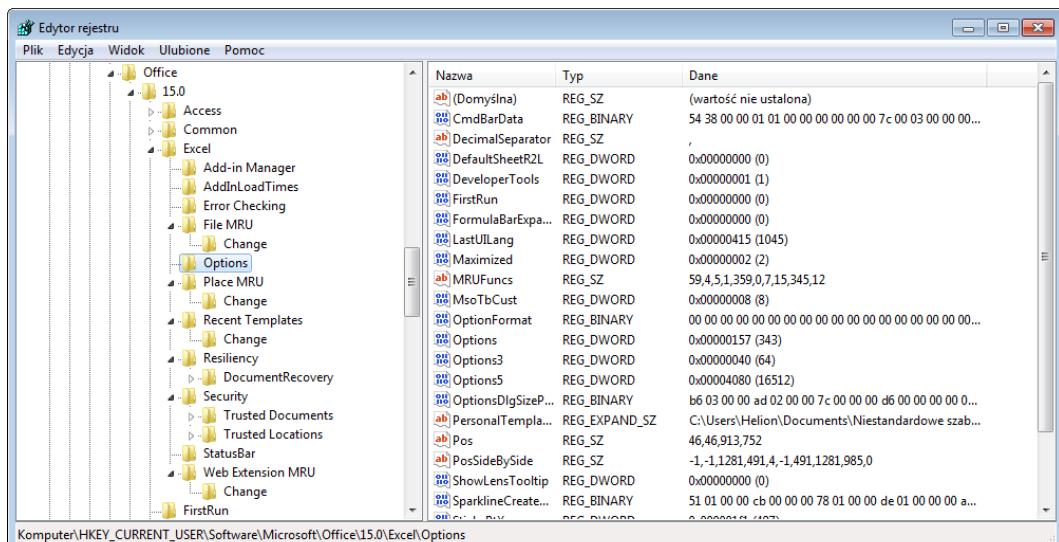
Rejestr systemu Windows

Rejestr systemu Windows to, ogólnie mówiąc, centralna, hierarchiczna baza danych wykorzystywana zarówno przez system operacyjny, jak i jego aplikacje. Rejestr po raz pierwszy pojawił się w systemie Windows 95, zastępując pliki *INI*, w których do tej pory system operacyjny i aplikacje przechowywały swoje ustawienia.



Makra języka VBA również mogą odczytywać i zapisywać informacje w rejestrze systemu Windows. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 9.

Do przeglądania zawartości rejestru, a nawet modyfikacji jego zawartości (o ile wiesz, co robisz...) możesz użyć programu o nazwie Edytor rejestru (plik *regedit.exe*). Zanim jednak rozpoczniesz eksplorację rejestru, powinieneś poświęcić chwilę na zapoznanie się z zawartością ramki „Zanim dokonasz zmian w rejestrze...”. Na rysunku 3.7 przedstawiono wygląd okna Edytora rejestru.



Rysunek 3.7. Edytor rejestru pozwala na przeglądanie rejestru i wprowadzanie do niego zmian

Zanim dokonasz zmian w rejestrze...

Za pomocą programu *regedit.exe* możesz zmienić dowolną wartość w rejestrze systemu Windows, w tym informacje mające krytyczne znaczenie dla pracy i stabilności systemu operacyjnego. Krótko mówiąc, jeżeli zmienisz nie to, co trzeba, system Windows może po prostu przestać działać poprawnie.

Powinieneś wyrobić sobie nawyk wybierania z menu *Plik* programu Edytor rejestru polecenia *Eksportuj*. Polecenie to umożliwia zapisanie w formacie ASCII zawartości całego rejestru lub tylko wybranej części. Jeżeli stwierdzisz, że zrobiłeś coś nie tak, to w celu przywrócenia rejestru do poprzedniego stanu zawsze możesz zaimportować plik ASCII (z menu *Plik* należy wybrać pozycję *Importuj*). Więcej szczegółowych informacji na temat korzystania z edytora rejestru znajdziesz w jego systemie pomocy.

Rejestr ma strukturę hierarchiczną i składa się z kluczy i wartości. Główne klucze rejestru systemu Windows są następujące:

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE
- HKEY_USERS
- HKEY_CURRENT_CONFIG

Ustawienia Excela

Informacje wykorzystywane przez Excela 2013 są przechowywane m.in. w następującej sekcji rejestru:

HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel

W tej sekcji rejestru znajduje się szereg kluczy zawierających specyficzne wartości wpływające na sposób działania Excela.

Ustawienia rejestru są automatycznie uaktualniane przez Excela w momencie jego zamknięcia.



Excel wczytuje zawartość rejestru systemu Windows tylko raz — podczas uruchamiania programu. Ustawienia rejestru uaktualniane są też tylko raz — gdy Excel zostanie prawidłowo zamknięty. Jeżeli Excel zawiesi się (co się niestety czasami zdarza), zawartość rejestru nie zostanie uaktualniona. Jeżeli na przykład zmodyfikujesz jedno z ustawień Excela, takie jak wyświetlanie paska statusu, dokonana zmiana nie zostanie zapisana w rejestrze do momentu poprawnego zamknięcia aplikacji.

W tabeli 3.5 przedstawiono sekcje rejestru, których używa Excel 2013. Pamiętaj, że na różnych komputerach zawartość tych sekcji rejestru może się różnić od siebie.

Jeżeli masz problemy z uruchomieniem Excela, przyczyną może być uszkodzony klucz rejestrów. Możesz spróbować przywrócić funkcjonalność Excela, uruchamiając *Edytor rejestru* i usuwając całą sekcję Excela:

HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel

Tabela 3.5. Dane konfiguracyjne Excela przechowywane w rejestrze

Sekcja	Opis
<i>Add-in Manager</i>	Zawiera listę dodatków widocznych w oknie dialogowym <i>Dodatki</i> . Na liście nie są widoczne dodatki dołączone do Excela. Jeżeli na liście znajduje się wpis dotyczący dodatku, którego już nie używasz, możesz go usunąć za pomocą programu Edytor rejestru.
<i>Converters</i>	Zawiera listę dodatkowych (zewnętrznych) konwerterów plików, których nie wbudowano do Excela.
<i>Error Checking</i>	Przechowuje ustawienia dotyczące sprawdzania błędów w formułach.
<i>File MRU</i>	Przechowuje listę ostatnio używanych plików (dostępnej po przejściu na kartę <i>Plik</i> i wybraniu z menu polecenia <i>Ostatnio używane</i>).
<i>Options</i>	Sekcja do przechowywania ustawień różnych.
<i>Place MRU</i>	Przechowuje informacje o najczęściej używanych lokalizacjach (katalogi i inne lokalizacje w pamięci masowej).
<i>Recent Templates</i>	Przechowuje nazwy ostatnio używanych szablonów.
<i>Resiliency</i>	Informacje używane do przywracania dokumentów.
<i>Security</i>	Okręsła poziom bezpieczeństwa otwieranych plików zawierających makra.
<i>Spell Checker</i>	Przechowuje informacje o opcji modułu sprawdzającego pisownię.
<i>StatusBar</i>	Przechowuje ustawienia paska stanu.
<i>UserInfo</i>	Przechowuje informacje dotyczące użytkownika.

Po ponownym uruchomieniu Excela wszystkie niezbędne klucze rejestrów zostaną przebudowane i Excel powinien się uruchomić. Oczywiście po wykonaniu takiej operacji utracisz wszystkie przechowywane w rejestrze informacje na temat dostosowania Excela do Twoich potrzeb.

Rozdział 4.

Podstawy

projektowania aplikacji

arkusza kalkulacyjnego

W tym rozdziale:

- Podstawowe etapy projektowania aplikacji arkusza kalkulacyjnego
- Określenie wymagań użytkownika
- Planowanie aplikacji spełniających wymagania użytkownika
- Projektowanie i testowanie aplikacji
- Dokumentowanie aplikacji na etapie projektowania i tworzenie dokumentacji dla użytkownika

Czym jest aplikacja arkusza kalkulacyjnego?

Na potrzeby niniejszej książki terminem *aplikacja arkusza kalkulacyjnego* będziemy nazywali *plik arkusza* (lub grupę powiązanych ze sobą plików) opracowany przez projektanta w taki sposób, że inny użytkownik bez konieczności przechodzenia gruntowego szkolenia może za jego pomocą wykonać określone operacje i zadania. Zgodnie z taką definicją większość plików arkuszy kalkulacyjnych nie zostanie uznana za aplikacje arkusza kalkulacyjnego. Na dysku twardym swojego komputera możesz posiadać dziesiątki lub setki takich plików, ale założę się, że większość z nich nie została zaprojektowana do wykorzystania przez innych użytkowników.

Dobra aplikacja arkusza kalkulacyjnego wyróżnia się następującymi cechami:

- Umożliwia użytkownikowi wykonanie zadania, którego w innym wypadku nie byłby w stanie wykonać.
- Oferuje odpowiednie rozwiązywanie problemu (aczkolwiek środowisko arkusza kalkulacyjnego nie zawsze jest rozwiązaniem optymalnym).

- Realizuje założony cel (taki warunek może się wydawać oczywisty, ale w praktyce często zdarza się, że aplikacje go nie spełniają).
- Generuje dokładne wyniki i jest pozbawiona błędów.
- W celu wykonania zadania korzysta z właściwych i wydajnych metod oraz algorytmów.
- Wychwytuje błędy, zanim będzie konieczna interwencja użytkownika.
- Uniemożliwia użytkownikowi przypadkowe bądź celowe usunięcie lub zmodyfikowanie ważnych komponentów.
- Interfejs aplikacji jest przejrzysty i spójny, dzięki czemu użytkownik zawsze wie, jak postępować.
- Formuły, makra i elementy interfejsu są dobrze udokumentowane, dzięki czemu w razie konieczności można łatwo dokonać ich modyfikacji.
- Może być modyfikowana przy użyciu prostych metod bez dokonywania większych zmian. Zwykle w miarę upływu czasu potrzeba wprowadzenia różnych zmian i poprawek do aplikacji staje się coraz bardziej oczywista.
- Posiada łatwo dostępny system pomocy, oferujący przydatne informacje przynajmniej o podstawowych procedurach.
- Może być przenoszona i uruchamiana w dowolnym systemie posiadającym odpowiednie oprogramowanie (w naszym przypadku jest to kopia odpowiedniej wersji Excela).

Z pewnością nie będzie to dla nikogo zaskoczeniem, jeżeli powiem, że można tworzyć aplikacje arkusza kalkulacyjnego przeznaczone do zastosowań o różnym stopniu złożoności, począwszy od prostych szablonów do wypełnienia bądź wprowadzania danych, a skończywszy na złożonych, bardzo rozbudowanych programach korzystających z niestandardowych menu i okien dialogowych, które na pierwszy rzut oka mogą zupełnie nie przypominać arkuszy kalkulacyjnych.

Podstawowe etapy projektowania

Nie istnieje prosta i niezawodna recepta dotycząca projektowania wydajnych aplikacji arkusza kalkulacyjnego. Każdy programista posiada swój własny styl tworzenia takich aplikacji, stąd można śmiało zaryzykować stwierdzenie, że nie istnieje jedna, najlepsza metoda sprawdzająca się w każdym przypadku. Co więcej, każdy realizowany projekt jest inny, a zatem potrzebuje indywidualnego podejścia. Wreszcie wymagania i ogólna postawa osób, z którymi będziesz współpracował (lub dla których będziesz pracował), również wpływają na sposób, w jaki będzie przebiegał proces projektowania.

Projektanci aplikacji arkuszy kalkulacyjnych zazwyczaj wykonują następujące zadania:

- Określanie wymagań użytkownika.
- Planowanie aplikacji spełniającej wymagania użytkownika.
- Wybieranie odpowiedniego interfejsu użytkownika.

- Tworzenie arkusza kalkulacyjnego, formuł, makr oraz interfejsu użytkownika.
- Testowanie aplikacji oraz wykrywanie i usuwanie błędów.
- Uodpornianie aplikacji na błędy popełniane przez użytkownika.
- Nadawanie aplikacji przyjaznego, intuicyjnego i estetycznego wyglądu.
- Dokumentowanie prac projektowych.
- Tworzenie systemu pomocy i dokumentacji przeznaczonej dla użytkownika.
- Przekazanie aplikacji użytkownikom.
- Aktualizacja aplikacji (kiedy to konieczne).

Nie wszystkie wymienione czynności są wymagane w przypadku każdej aplikacji, a kolejność ich wykonywania może być inna w różnych projektach. Poszczególne zadania zostały omówione w dalszej części rozdziału, a ich aspekty techniczne (w większości przypadków) zostały omówione w kolejnych rozdziałach.

Określanie wymagań użytkownika

Kiedy rozpoczynasz przygotowania do tworzenia nowego projektu aplikacji arkusza programu Excel, jedną z pierwszych czynności powinna być dokładna identyfikacja wymagań stawianych przez użytkowników końcowych. Błędy w ocenie potrzeb użytkowników często prowadzą do niepotrzebnego zwiększenia nakładu pracy i konieczności dokonywania poprawek w gotowej aplikacji, tak aby wykonywała to, czego użytkownik naprawdę od niej oczekuje.

W niektórych sytuacjach projektant ma bliski kontakt z końcowymi użytkownikami, a nawet może być jednym z nich. W innych — na przykład dotyczy to konsultantów realizujących projekt dla nowego klienta — projektant dysponuje bardzo ograniczoną wiedzą na temat użytkowników czy charakteru ich pracy.

W jaki sposób określić wymagania użytkowników końcowych? Jeżeli to możliwe, naprawdę warto po prostu spotkać się z nimi i zadać kilka precyzyjnych pytań. Jeszcze lepszym rozwiązaniem będzie przygotowanie wymagań aplikacji w formie pisemnej, utworzenie diagramów przepływu informacji (ze zwróceniem uwagi na pozornie mniej istotne szczegóły) i dołożenie wszelkich starań, które zagwarantują, że projektowane oprogramowanie będzie zgodne z oczekiwaniami.

Poniżej wymieniono kilka wytycznych, które mogą być pomocne na tym etapie realizacji projektu:

- Nie zakładaj, że znasz wymagania użytkowników. Zgadywanie na tym etapie projektu prawie zawsze jest przyczyną późniejszych problemów.
- W miarę możliwości rozmawiaj bezpośrednio z użytkownikami końcowymi tworzonej aplikacji, a nie wyłącznie z dyrektorem lub kierownikiem projektu.
- Dowiedz się, czy aktualnie istnieją jakiekolwiek rozwiązań pozwalające na realizację wymagań stawianych przez użytkowników. Czasami poprzez proste

zaadaptowanie istniejącej aplikacji możesz sobie zaoszczędzić wiele pracy, a przyjrzenie się aktualnie stosowanym rozwiązaniom pozwoli zapoznać się z ich funkcjonowaniem.

- Zidentyfikuj zasoby i możliwości, jakimi dysponuje użytkownik. Na przykład, możesz spróbować określić, czy istnieją jakiekolwiek ograniczenia sprzętowe lub programowe, które będą musiały zostać wzięte pod uwagę.
- W miarę możliwości zorientuj się, na jakiej platformie bądź platformach sprzętowych ma pracować aplikacja. Powinieneś to rozważyć zwłaszcza w sytuacji, kiedy tworzona aplikacja ma być używana na nieco wolniejszych, starszych komputerach.
- Zidentyfikuj wersje Excela, które są lub będą używane. Co prawda Microsoft robi wszystko, co możliwe, aby skłonić użytkowników do korzystania zawsze z najnowszych wersji, ale w praktyce to się zazwyczaj nie udaje i większość użytkowników nadal używa którejś z poprzednich wersji Excela.
- Sprawdź poziom umiejętności końcowych użytkowników aplikacji. Taka informacja z pewnością pomoże Ci w odpowiednim zaprojektowaniu aplikacji.
- Spróbuj określić planowany czas „życia” aplikacji i ewentualnie czy w trakcie tego czasu będzie konieczne wprowadzanie zmian do aplikacji. Taka wiedza pomoże Ci określić nakład pracy potrzebny do zrealizowania projektu i ułatwi zaplanowanie ewentualnych zmian.

Nie bądź zaskoczony, jeżeli specyfikacja projektu zmieni się przed zakończeniem prac. Jest to dość powszechnie zjawisko, stąd jeżeli będziesz z góry oczekiwał pojawienia się takich zmian, będziesz w zdecydowanie lepszej sytuacji, niż gdybyś dał się im zaskoczyć. Na wszelki wypadek upewnij się jednak, czy umowa na wykonanie aplikacji (o ile taka umowa została podpisana) obejmuje wprowadzenie takiego zakresu zmian.

Planowanie aplikacji spełniającej wymagania użytkownika

Kiedy zakończysz proces definiowania wymagań użytkownika, perspektywa natychmiastowego uruchomienia Excela i rozpoczęcia prac nad tworzeniem aplikacji stanie się bardzo kusząca. Przyjmij jednak dobrą radę i postaraj się jeszcze z tym powstrzymać. Budowniczowie nie stawiają domu bez zestawu planów, więc również i Ty nie powinieneś rozpoczynać prac nad tworzeniem aplikacji bez jakiegoś planu. Zakres takiego planu zależy co prawda w dużej mierze od skali i stopnia złożoności realizowanego projektu, ale mimo wszystko powinieneś przynajmniej *trochę* czasu poświęcić na zastanowienie się nad tym, co masz zamiar robić, i przynajmniej zgrubnie określić poszczególne etapy postępowania.

Zatem zanim podwiniesz rękawy i zasiadziesz przy klawiaturze swojego komputera, powinieneś przez chwilę zastanowić się nad różnymi wariantami rozwiązania problemu. Właśnie na tym etapie dogłębiańska znajomość Excela okazuje się wręcz nieoceniona. Unikanie ślepych zaułków jest zawsze najlepszym sposobem postępowania.

Jeżeli poprosisz dziesięciu różnych ekspertów o zaprojektowanie aplikacji w oparciu o bardzo precyzyjną specyfikację, to w rezultacie zapewne otrzymasz dziesięć różnych implementacji projektu spełniających (bądź nie...) podane wymagania. Spośród zaproponowanych rozwiązań kilka z pewnością będzie lepszych od innych, ponieważ Excel często umożliwia wykonanie określonego zadania na kilka różnych sposobów. Jeżeli dobrze znasz ten program, będziesz wiedział, jakie metody masz do dyspozycji, będziesz więc mógł wybrać taką, która najlepiej sprawdzi się w realizowanym projekcie. Często odrobina kreatywności pozwala na wypracowanie nietypowego czy wręcz niezwykłego rozwiązania, które może być znacznie bardziej efektywne od innych.

W początkowej fazie planowania powinieneś zatem wziąć pod uwagę między innymi następujące kwestie:

- **Struktura plików.** Zastanów się, czy chcesz stworzyć jeden skoroszyt zawierający wiele arkuszy, kilka skoroszytów jednoarkuszowych czy plik szablonu.
- **Struktura danych.** Zawsze powinieneś dobrze przemyśleć strukturę danych, włączając w to dokonanie wyboru pomiędzy użyciem plików zewnętrznej bazy danych a przechowywaniem danych bezpośrednio na arkuszach skoroszytów.
- **Zastosowanie dodatków lub skoroszytów.** W niektórych przypadkach dodatek to najlepsza postać ostatecznego produktu. A może warto zastosować dodatek razem ze standardowym skoroszytem?
- **Wersja Excela.** Czy Twoja aplikacja będzie używana tylko z Exceliem 2013, czy może też z wersją 2007 i nowszymi? A co z wersjami Excel 2003 i wcześniejszymi? Czy aplikacja będzie też uruchamiana na komputerach Macintosh? Wbrew pozorom to bardzo ważne kwestie, ponieważ każda kolejna wersja Excela jest poszerzona o nowe funkcje, które nie były dostępne w poprzednich wersjach. Zupełnie nowy interfejs użytkownika wprowadzony w Excelu 2007 powoduje, że napisanie aplikacji, która będzie poprawnie działała również w starszych wersjach programu, stało się jeszcze większym wyzwaniem.
- **Obsługa błędów.** Prawidłowa obsługa błędów to jeden z kluczowych elementów każdej aplikacji. Tworząc aplikację, powinieneś określić, w jaki sposób będzie ona wykrywała i obsługiwała błędy. Na przykład: jeżeli Twoja aplikacja modyfikuje formatowanie wybranych komórek aktywnego arkusza, powinieneś pomyśleć o dodaniu obsługi sytuacji, w której zostanie uaktywniony arkusz wykresu.
- **Zastosowanie funkcji specjalnych.** Jeżeli aplikacja dokonuje podsumowań dużych zbiorów danych, powinieneś rozważyć użycie tabeli przestawnej Excela. Do sprawdzania, czy użytkownicy wprowadzają prawidłowe informacje, możesz użyć odpowiednich funkcji sprawdzania poprawności danych.
- **Kwestie związane z wydajnością.** O szybkości działania aplikacji powinieneś myśleć już na etapie jej projektowania, a nie wtedy, gdy aplikacja jest gotowa i użytkownicy uskarżają się na jej wydajność.
- **Poziom bezpieczeństwa.** Jak zapewne wiesz, Excel oferuje kilka poziomów ochrony ograniczającej dostęp do określonych elementów skoroszytu. Na przykład możesz zablokować komórki tak, że modyfikacja zawartych w nich formuł nie będzie możliwa. Możesz również zabezpieczyć arkusz za pomocą hasła

uniemożliwiającego nieautoryzowanym użytkownikom przeglądanie określonych plików i uzyskanie do nich dostępu. Praca będzie łatwiejsza, jeśli wcześniej określisz, co musi być chronione i na jakim poziomie.



Pamiętaj, że funkcje ochrony skoroszytów Excela i ich zawartości nie dają 100-procentowego zabezpieczenia przed nieautoryzowanym dostępem. Jeżeli potrzebujesz pełnego i absolutnego bezpieczeństwa aplikacji, prawdopodobnie Excel nie będzie najlepszą platformą do tego celu.

Na tym etapie projektowania będziesz już musiał rozważyć wiele złożonych zagadnień związanych z przyszłym funkcjonowaniem aplikacji. Pamiętaj, że zawsze powinieneś brać pod uwagę wszystkie możliwe opcje i nie trzymać się uparcie pierwszego rozwiązania, które przyjdzie Ci na myśl.

Kolejnym elementem, o którym powinieneś pamiętać, jest planowanie zmian i modyfikacji. Zrobisz sobie przysługę, jeżeli utworzysz aplikację tak uniwersalną, jak to tylko możliwe. Nie powinieneś na przykład pisać procedury, która przetwarza tylko ściśle określony zakres komórek. Zamiast tego powinieneś stworzyć procedurę akceptującą jako argument dowolny zakres — gdy pojawi się żądanie dokonania zmian, przeprowadzenie modyfikacji będzie wówczas prostsze. Poza tym projektowane aplikacje są często do siebie podobne. Jeśli zaplanujesz wielokrotne zastosowanie niektórych rozwiązań, wyjdzie Ci to tylko na dobre.

Z własnego doświadczenia wiem, że końcowy użytkownik nie powinien decydować o sposobie rozwiązania problemu. Na przykład założmy, że od kierownika działu dowiedziałeś się, że jego pracownicy potrzebują aplikacji tworzącej pliki tekstowe, które są importowane do innego programu. Uważaj, aby w takiej sytuacji nie pomylić wymagań użytkownika z rozwiązaniem. W rzeczywistości użytkownicy mogą np. potrzebować po prostu narzędzia umożliwiającego współdzielenie danych. Zastosowanie pośredniego pliku tekstuowego jest tylko jednym z możliwych rozwiązań, gdyż istnieją również inne metody, takie jak bezpośrednie przesłanie informacji przy użyciu technologii DDE (ang. *Dynamic Data Exchange*) lub OLE (ang. *Object Linking and Embedding*). Innymi słowy, nigdy nie powinieneś pozwolić użytkownikom, aby narzucali Ci sposób rozwiązania problemu. Wybór najlepszego rozwiązania to *Twoje zadanie*.

Wybieranie odpowiedniego interfejsu użytkownika

Projektując arkusze kalkulacyjne dla innych użytkowników, powinieneś zwrócić szczególną uwagę na interfejs użytkownika. *Interfejs użytkownika* to inaczej sposób, za pomocą którego użytkownik komunikuje się z aplikacją i uruchamia odpowiednie makra VBA.

Excel 2007 i nowsze wersje wprowadziły radykalne zmiany na tym polu. Niestandardowe menu użytkownika i paski narzędzi, powszechnie używane we wcześniejszych wersjach Excela, stały się elementami przestarzałymi. Projektanci nowych aplikacji muszą się więc teraz nauczyć, jak korzystać ze Wstążki, czyli nowego interfejsu użytkownika.

Excel posiada cały szereg mechanizmów ułatwiających tworzenie interfejsu użytkownika:

- Dostosowywanie Wstążki do potrzeb użytkownika.
- Dostosowywanie menu podręcznego do potrzeb użytkownika.
- Klawisze skrótu.
- Niestandardowe okna dialogowe (formularze *UserForm*).
- Formanty umieszczane bezpośrednio w arkuszu, takie jak *ListBox* lub *CommandButton*.

Poszczególne elementy omówimy pokrótko w kolejnych podpunktach, a bardziej szczegółowo w następnych rozdziałach.

Dostosowywanie Wstążki do potrzeb użytkownika

Wstążka, czyli nowy interfejs wprowadzony w programie Excel 2007, wymusiła wręcz dramatyczne zmiany w metodach projektowania interfejsów użytkownika. Na szczęście projektanci nadal mają całkiem spory zakres kontroli nad wyglądem i zachowaniem Wstążki, aczkolwiek pomimo iż Excel 2013 pozwala użytkownikowi na modyfikowanie Wstążki, to jednak wcale nie jest to takie proste zadanie.



Więcej szczegółowych informacji na temat pracy ze Wstążką znajdziesz w rozdziale 20.

Dostosowywanie menu podręcznego do potrzeb użytkownika

Excel 2013 nadal pozwala programistom VBA na dostosowywanie do potrzeb użytkownika menu podręcznego dostępnego po naciśnięciu prawego przycisku myszy. Na rysunku 4.1 przedstawiono wygląd niestandardowego menu podręcznego wyświetlanego na ekranie po kliknięciu numeru wiersza prawym przyciskiem myszy. Zwróć uwagę, że na dole menu znajduje się kilka dodatkowych elementów (oznaczonych ikoną z literką „P”), które nie są wyświetlane w standardowym menu.



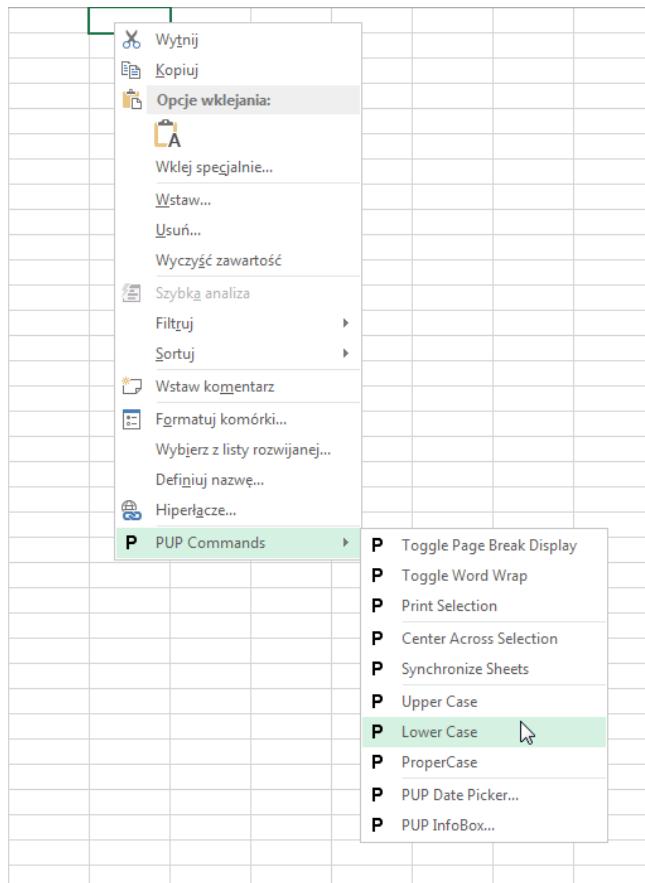
Więcej szczegółowych informacji na temat modyfikowania menu podręcznego przy użyciu VBA znajdziesz w rozdziale 21.

Tworzenie klawiszy skrótu

Kolejnym elementem pozwalającym na dostosowanie interfejsu użytkownika do swoich potrzeb jest możliwość tworzenia własnych klawiszy skrótu. Excel pozwala na przypisanie do makra wybranej kombinacji używającej klawisza *Ctrl* lub klawiszy *Shift* i *Ctrl*. Kiedy użytkownik naciśnie taką kombinację klawiszy, Excel uruchomi przypisane do niej makro.

Rysunek 4.1.

Przykład
niestandardowego
menu podręcznego



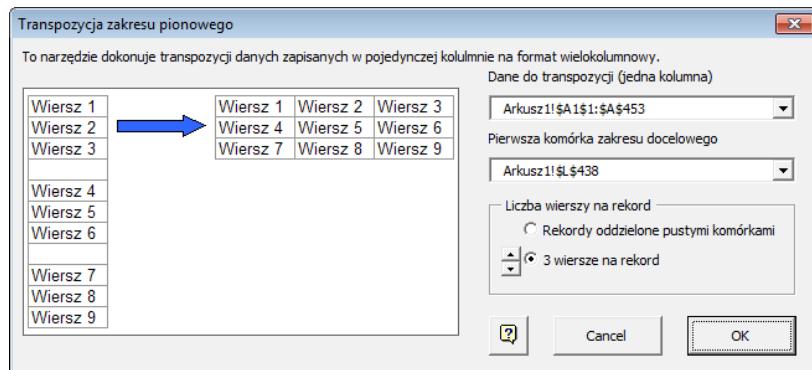
Powinieneś jednak pamiętać o dwóch sprawach. Po pierwsze, powinieneś poinformować użytkownika, które klawisze są aktywne i jaką spełniają funkcję. Po drugie, powinieneś uważać, aby nie użyć kombinacji klawiszy, która jest już używana do innych celów. Pamiętaj, że kombinacja klawiszy przypisywana do makra ma pierwszeństwo nad wbudowanymi skrótami klawiaturowymi. Na przykład kombinacja *Ctrl+S* to wbudowany klawisz skrótu Excela, służący do zapisania aktualnie otwartego pliku. Jeśli taka sama kombinacja zostanie przypisana do makra, stracisz możliwość zapisywania pliku przy użyciu tej kombinacji. Pamiętaj, że klawisze skrótu rozróżniają wielkość znaków, stąd będziesz mógł np. użyć kombinacji *Ctrl+Shift+S*.

Tworzenie niestandardowych okien dialogowych

Każdy, kto kiedykolwiek korzystał z komputera osobistego, bez wątpienia spotkał się już z oknami dialogowymi, dzięki czemu niestandardowe okna dialogowe mogą odgrywać ogromną rolę w interfejsach użytkownika aplikacji Excela. Na rysunku 4.2 przedstawiono przykład takiego niestandardowego okna dialogowego.

Rysunek 4.2.

Okno dialogowe utworzone przy użyciu formularza UserForm



Niestandardowe okna dialogowe noszą nazwę *formularzy UserForm* i pozwalają użytkownikowi na wprowadzenie danych, zaznaczenie wybranych opcji lub właściwości i sterowanie działaniem całej aplikacji. Formularze *UserForm* mogą być tworzone i modyfikowane przy użyciu edytora VBE. Elementy składowe takiego okna dialogowego (przyciski, listy rozwijane, pola wyboru itp.) są nazywane *formantami*, a dokładniej, *formantami ActiveX*. Excel oferuje standardowy zbiór formantów ActiveX, a w razie potrzeby możesz użyć formantów tworzonych przez innych programistów.

Formanty umieszczone w oknie dialogowym można skojarzyć z komórką arkusza, co nie wymaga użycia żadnego makra (z wyjątkiem prostego makra wyświetlającego okno dialogowe). Połączenie formantu z komórką jest prostą operacją, ale nie zawsze jest to najlepsza metoda pobierania danych wprowadzonych przez użytkownika w oknie dialogowym. W większości przypadków będziesz do tego celu używał odpowiednich makr języka VBA, współpracujących z niestandardowymi oknami dialogowymi.



Więcej szczegółowych informacji na temat formularzy *UserForm* znajdziesz w III części książki.

Zastosowanie formantów ActiveX w arkuszu

Excel pozwala umieszczać formanty ActiveX formularzy *UserForm* w warstwie *rysunkowej* arkusza (jest to niewidoczna warstwa znajdująca się nad arkuszem, przechowująca obrazy, wykresy i inne obiekty). Na rysunku 4.3 przedstawiono prosty model arkusza zawierający kilka formantów formularza umieszczonych bezpośrednio na arkuszu. Znajdziesz tam formanty *CheckBox*, *ScrollBar* i dwa zestawy formantów *OptionButton*. W skoroszycie nie ma żadnych makr, a formanty są przypisane bezpośrednio do wybranych komórek arkusza.



Przykładowy skoroszyt o nazwie *Formanty arkusza.xlsx* znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e13pvw.htm>).

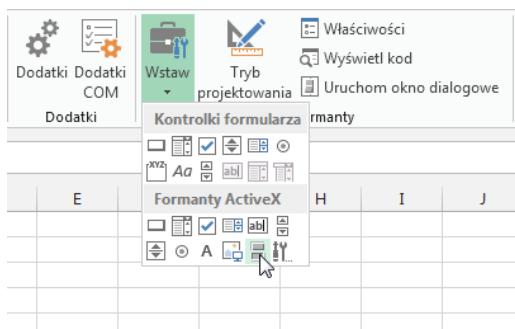
Rysunek 4.3. Formanty UserForm możesz umieścić bezpośrednio na arkuszu i przypisać do nich odpowiednie komórki

Jednym z najczęściej stosowanych formantów jest przycisk CommandButton. Sam formant CommandButton nie wykonuje żadnej operacji, stąd do każdego przycisku musisz przypisać odpowiednie makro.

Bezpośrednie zastosowanie formantów formularza na arkuszu często eliminuje konieczność tworzenia niestandardowych okien dialogowych. Często poprzez bezpośrednie umieszczenie na arkuszu kilku formantów ActiveX (lub formantów formularza) w znaczący sposób możesz uprościć działanie arkusza kalkulacyjnego. Dzięki takiemu rozwiązaniu użytkownik nie musi wprowadzać danych do komórek, a decyzje może podejmować przy użyciu dobrze znanych mu formantów.

Aby umieścić wybrany formant na arkuszu, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Wstaw* znajdujący się w grupie opcji *Formanty* (patrz rysunek 4.4). Jeżeli karta *Deweloper* nie jest widoczna, przywołaj na ekran okno *Opcje programu Excel*, kliknij kategorię *Dostosowywanie Wstążki* i na liście wyświetlanych kart zaznacz opcję *Deweloper*.

Rysunek 4.4.
Formanty formularza



Formanty są dostępne w dwóch wersjach: *Formanty formularza* oraz *Formanty ActiveX*. Oba zestawy formantów mają swoje wady i zalety. Ogólnie rzecz biorąc, formanty formularza są prostsze w użyciu, ale za to formanty ActiveX są bardziej elastyczne. W tabeli 4.1 przedstawiono zestawienie tych dwóch klas formantów.

Tabela 4.1. Porównanie formantów formularza i formantów ActiveX

	Formanty ActiveX	Formanty formularza
Wersje Excela	97, 2000, 2002, 2003, 2007, 2010, 2013	5, 95, 97, 2000, 2002, 2003, 2007, 2010, 2013
Dostępne formanty	CheckBox (pole wyboru), TextBox (pole tekstowe), CommandButton (przycisk polecenia), OptionButton (przycisk opcji), ListBox (pole listy), ComboBox (pole kombi), ToggleButton (przycisk przełącznika), SpinButton (przycisk pokrętła), ScrollBar (pasek przewijania), Label (etykieta), Image (obraz); inne mogą zostać dodane	GroupBox (pole grupy), Button (przycisk), CheckBox (pole wyboru), OptionButton (przycisk opcji), ListBox (pole listy), DropDown (ComboBox) (pole kombi), ScrollBar (pasek przewijania), Spinner (pokrętło)
Przechowywanie kodu makra	W module kodu powiązanego z arkuszem kodu makra	W dowolnym, standardowym module kodu języka VBA
Nazwa makra	Odpowiada nazwie formantu (na przykład CommandButton1_Click)	Dowolna nazwa
Odpowiedniki	Formanty formularza <i>UserForm</i>	Formanty arkusza dialogowego (w wersjach wcześniejszych niż Excel 97)
Możliwości dostosowania	Duże, przy użyciu okna <i>Properties</i>	Minimalne
Reakcja na zdarzenia	Tak	Tylko na zdarzenia Click lub Change

Rozpoczęcie prac projektowych

Po określaniu wymagań użytkownika, wybraniu rozwiązania umożliwiającego ich spełnienie oraz wybraniu komponentów, które wejdą w skład interfejsu użytkownika, najwyższa pora przejść do sedna sprawy i rozpocząć tworzenie aplikacji. Oczywiście realizacja tego etapu zajmie sporą część całkowitego czasu poświęconego na wykonanie projektu.

Sposób, w jaki będziesz pracował nad aplikacją, zależy od Twojego stylu pracy i specyfiki programu. Poza prostymi szablonami skoroszytów zawierających pola przeznaczone do wypełnienia, w aplikacji prawdopodobnie zostaną zastosowane makra. Tworzenie makr stanowi trudniejszą fazę. Z łatwością można tworzyć makra w Excelu, ale sztuką jest napisanie *dobrych* makr.

Zadania realizowane z myślą o końcowym użytkowniku

W tej części rozdziału omówimy ważne zagadnienia związane z projektowaniem aplikacji, z którymi możesz spotkać się w sytuacji, kiedy tworzona aplikacja staje się coraz bardziej funkcjonalna i wielkimi krokami zbliża się moment utworzenia jej finalnego pakietu i przesłania go użytkownikom końcowemu.

Testowanie aplikacji

Ile razy zdarzyło Ci się, że w bardzo ważnym momencie pracy komercyjne oprogramowanie uległo zawieszeniu? Prawdopodobnie powodem problemu była niewystarczająca liczba testów, w trakcie których nie wykryto wszystkich błędów. Wszystkie złożone aplikacje zawierają błędy, ale w oprogramowaniu najwyższej jakości błędy po prostu mniej rzucają się w oczy. Czasem w celu uzyskania poprawnego działania aplikacji konieczne będzie ominięcie błędów Excela poprzez modyfikację programu.

Gotową aplikację koniecznie musisz przetestować. Jest to jeden z najważniejszych etapów projektu i często zdarza się, że testowanie i usuwanie błędów aplikacji zajmuje taką samą ilość czasu, jak jej tworzenie. Właściwie sporą liczbę testów powinieneś wykonać już na etapie projektowania aplikacji — w końcu niezależnie od tego, czy piszesz procedurę w języku VBA, czy tworzysz formuły bezpośrednio w arkuszu, musisz mieć pewność, że aplikacja działa zgodnie z przyjętymi założeniami.

Podobnie jak standardowe programy komplikowane, tak i aplikacje arkusza kalkulacyjnego są również podatne na błędy. *Błąd aplikacji* może być zdefiniowany jako (1) coś, co zdarza się, a nie powinno się zdarzać w trakcie działania aplikacji, lub (2) coś, co się nie zdarza, a zdarzać się powinno. Oba rodzaje błędów są w równym stopniu nieprzyjemne i z tego powodu powinieneś z góry zaplanować poświęcenie odpowiedniej ilości czasu na testowanie aplikacji uwzględniające wszystkie możliwe przypadki i na usuwanie wszelkich wykrytych na tym etapie problemów. Niestety, czasami możesz napotkać problemy, które pojawiają się zupełnie nie z Twojej winy (patrz ramka „Błędy? W Excelu?”).

Prawdopodobnie nie muszę Ci przypominać o konieczności wykonania szczegółowych testów przynajmniej tych aplikacji arkusza, które będą używane przez innych użytkowników. W zależności od docelowej grupy użytkowników możesz również spróbować uodpornić swoją aplikację na wszelkiego rodzaju możliwe do przewidzenia błędy, niepoprawne dane i wszelkie sytuacje, które będziesz w stanie przewidzieć. Taki wysiłek nie tylko pomoże użytkownikowi, ale również z pewnością mocno przyczyni się do ochrony i umocnienia Twojej reputacji. Powinieneś również rozważyć możliwość wykonania testów wersji beta Twojej aplikacji — najlepszymi kandydatami na beta-testerów będą oczywiście użytkownicy końcowi Twojej aplikacji, ponieważ to właśnie oni będą w efekcie z tej aplikacji korzystać (patrz ramka „Testowanie wersji beta”).

Błędy? W Excelu?

Ktoś mógłby zakładać, że taki produkt jak Excel, używany przez miliony osób na całym świecie, powinien być pozbawiony błędów. Nic bardziej mylnego. Excel jest tak złożonym programem, że wręcz należy się ich spodziewać. I faktycznie — Excel zawiera błędy.

Wprowadzenie do sprzedaży takiego produktu jak Excel nie jest łatwym zadaniem nawet dla takiej firmy jak Microsoft, giganta mającego na pozór nieograniczone możliwości. Z przygotowywaniem oprogramowania związane są kompromisy i dylematy. Powszechnie wiadomo, że największą wytwórcy oprogramowania wprowadzają do sprzedaży produkty, mając pełną świadomość, że zawierają one błędy. Większość odkrywanych błędów jest na tyle niespotykana, że po prostu się je ignoruje. Firma mogłaby co prawda opóźnić wprowadzenie produktu na przykład o kilka miesięcy i usunąć wiele dodatkowych błędów, ale również w tym przypadku rykiem oprogramowania rządzą twarde prawa ekonomii. Korzyści wynikające z takiego opóźnienia często nie wyrównują poniesionych strat finansowych. Co prawda Excel posiada sporą pulę błędów, ale przypuszczam, że większość użytkowników tej aplikacji nigdy nie będzie miała z żadnym z nich do czynienia.

W tej książce omawiam tylko takie problemy z Excellem, które są mi znane. Z pewnością sam napotkasz kilka dodatkowych. Niektóre błędy pojawiają się tylko w określonej wersji Excela i w dodatku w przypadku specyficznej konfiguracji powiązanej ze sprzętem i (lub) oprogramowaniem. Spośród wszystkich błędów takie są najgorsze, ponieważ trudno je odtworzyć.

Co w takim razie może zrobić projektant? Powinien wykonać tzw. *obejście* (ang. *workaround*). Jeżeli coś nie działa, a wszystko wskazuje na to, że *powinno*, pora przejść do planu B. Frustrujące? Na pewno. Strata czasu? Zdecydowanie. Ale właśnie takie sytuacje są nieodłączną częścią pracy projektanta.

Testowanie wersji beta

Producenci oprogramowania zazwyczaj stosują rygorystyczne procedury testowania nowych produktów. Po zakończeniu intensywnych testów wewnętrznych wstępna wersja produktu jest zwykle wysyłana do grupy użytkowników zainteresowanych *testowaniem wersji beta* (czyli tzw. beta-testerów). Na tym etapie są identyfikowane dodatkowe problemy, usuwane zazwyczaj przed pojawiением się ostatecznej wersji produktu.

Jeżeli tworzysz aplikacje, z których będzie korzystać więcej niż kilka osób, możesz wziąć pod uwagę testowanie wersji beta. Dzięki temu właściwi (zazwyczaj) użytkownicy sprawdzą aplikację na różnych komputerach i z różnymi ustawieniami.

Testowanie wersji beta powinno rozpoczęć się po zakończeniu wszystkich testów wewnętrznych i uznaniu, że aplikacja jest gotowa do dystrybucji. Konieczne będzie określenie grupy użytkowników, którzy pomogą w przeprowadzeniu testów. Proces testów beta przyniesie najlepsze rezultaty, gdy użytkownikom zostaną przekazane wszystkie elementy wchodzące w skład ostatecznej aplikacji (dokumentacja użytkownika, program instalacyjny, system pomocy itp.). Wyniki testów beta można uzyskać na kilka sposobów, w tym poprzez bezpośrednie spotkania z użytkownikami, kwestionariusze i rozmowy telefoniczne.

Po zakończeniu testów beta prawie zawsze będziesz wiedział, które problemy muszą być usunięte i jakie udoskonalenia muszą zostać wykonane. Oczywiście testowanie wersji beta zajmuje dodatkowy czas i nie przy wszystkich projektach można sobie pozwolić na taki luksus.

Z pewnością nie będziesz w stanie przetestować działania aplikacji we wszystkich możliwych scenariuszach zdarzeń, ale pomimo to Twoje makra powinny obsługiwać przynajmniej najczęściej występujące rodzaje błędów. Na przykład: co się stanie, jeżeli użytkownik zamiast wartości liczbowej wprowadzi ciąg znaków? Co się stanie, jeżeli użytkownik spróbuje wykonać makro, gdy odpowiedni skoroszyt nie został otwarty? Co będzie, gdy zamknie okno dialogowe bez wybrania żadnych opcji lub wcisnie kombinację klawiszy *Ctrl+F6* i przejdzie do następnego okna? W miarę zdobywania doświadczenia bardzo dobrze poznasz takie sytuacje i bez chwili namysłu uwzględnisz w projekcie odpowiednie rozwiązania.

Uodpornianie aplikacji na błędy popełniane przez użytkownika

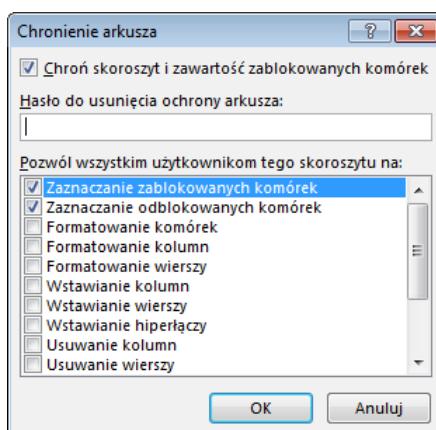
Jeżeli się nad tym dobrze zastanów, to zniszczenie integralności arkusza kalkulacyjnego jest zadaniem dosyć prostym. Usunięcie jednej istotnej formuły lub wartości często powoduje wygenerowanie błędów w całym arkuszu, a może nawet w innych powiązanych z nim arkuszach i skoroszytach. Co gorsza, jeśli uszkodzony skoroszyt zostanie zapisany, zastąpi na dysku poprawną kopię. Jeżeli wcześniej nie została wdrożona procedura wykonywania kopii zapasowych, użytkownik końcowy może znaleźć się w trudnej sytuacji i prawdopodobnie winą za to obarczy właśnie *Ciebie*.

Oczywiście nietrudno się zorientować, dlaczego stosowanie określonych mechanizmów ochrony arkuszy jest konieczne — zwłaszcza gdy będą z nich korzystali początkujący użytkownicy. Excel oferuje następujące mechanizmy ochrony arkuszy i jego elementów.

- **Blokowanie wybranych komórek.** W razie potrzeby możesz zablokować wybrane komórki, tak aby nie można było modyfikować ich zawartości. Aby to zrobić, przejdź na kartę *RECENZJA* i naciśnij przycisk *Chroni arkusz* znajdujący się w grupie opcji *Zmiany*. W oknie dialogowym *Chronienie arkusza* znajdziesz szereg opcji pozwalających na określenie akcji, jakie można wykonać w chronionym arkuszu (patrz rysunek 4.5).

Rysunek 4.5.

Okno dialogowe *Chronienie arkusza* pozwala na określenie akcji, jakie użytkownik może wykonać w chronionym arkuszu



- **Ukrywanie formuł w określonych komórkach.** Możesz ukrywać formuły zawarte w wybranych komórkach tak, aby nie były widoczne dla innych użytkowników (żeby to zrobić, przejdź na kartę *Ochrona* okna dialogowego

Formatowanie komórek). Pamiętaj, że funkcja ochrony formuł jest aktywna tylko wtedy, gdy włączona jest ochrona arkusza (aby to zrobić, przejdź na kartę *Recenzja* i naciśnij przycisk *Chroń arkusz* znajdujący się w grupie opcji *Zmiany*).

- **Ochrona całego skoroszytu.** Istnieje możliwość ochrony całego skoroszytu — jego struktury oraz pozycji lub rozmiaru okna (bądź obu tych elementów jednocześnie). Aby to zrobić, przejdź na kartę *Recenzja* i naciśnij przycisk *Chroń skoroszyt*.
- **Blokowanie obiektów arkusza.** Do ochrony obiektów arkusza (takich jak na przykład kształty) przed przenoszeniem lub modyfikacją powinieneś użyć opcji zlokalizowanych na karcie *Właściwości* okna dialogowego *Rozmiar i właściwości*. Aby przywołać na ekran okno dialogowe *Rozmiar i właściwości*, zaznacz wybrany obiekt i naciśnij przycisk *Uruchom okno dialogowe* znajdujący się w grupie opcji *Rozmiar* na karcie *Formatowanie* grupy kart *Narzędzia do rysowania*. Pamiętaj, że blokowanie obiektów jest aktywne tylko wtedy, gdy włączona jest ochrona arkusza (aby to zrobić, przejdź na kartę *Recenzja* i naciśnij przycisk *Chroń arkusz* znajdujący się w grupie opcji *Zmiany*). Domyślnie blokowane są wszystkie obiekty.
- **Ukrywanie wierszy, kolumn, arkuszy i dokumentów.** W razie potrzeby możesz ukryć wybrane wiersze, kolumny, arkusze, a nawet całe skoroszyty. Dzięki temu można poprawić wygląd arkusza, a także zapewnić ochronę przed ciekawskimi spojrzeniami niepowołanych osób trzecich.
- **Udostępnienie skoroszytu Excela w trybie tylko do odczytu.** Istnieje możliwość udostępnienia skoroszytu Excela w trybie tylko do odczytu (z użyciem hasła), dzięki czemu możesz zyskać pewność, że w pliku nie zostaną dokonane żadne przypadkowe zmiany. Aby to zrobić, przejdź na kartę *Plik* i z menu wybierz polecenie *Zapisz jako*. Na ekranie pojawi się okno dialogowe *Zapisz jako*. Naciśnij przycisk *Narzędzia* i z menu podręcznego wybierz polecenie *Opcje ogólne*.
- **Ochrona dokumentu przy użyciu hasła.** Aby uniemożliwić nieautoryzowanym użytkownikom otwarcie pliku, można zabezpieczyć go przy użyciu hasła. Aby to zrobić, przejdź na kartę *Plik* i z menu wybierz polecenie *Zapisz jako*. Następnie naciśnij przycisk *Chroń skoroszyt* i z menu podręcznego wybierz polecenie *Szyfruj przy użyciu hasła*.
- **Użycie dodatku chronionego hasłem.** W razie potrzeby możesz zastosować dodatek chroniony hasłem, który nie pozwala użytkownikowi na dokonanie w chronionych arkuszach żadnych modyfikacji.

Nadawanie aplikacji przyjaznego, intuicyjnego i estetycznego wyglądu

Jeśli miałeś do czynienia z różnymi pakietami oprogramowania, na pewno widziałeś przykłady kiepsko zaprojektowanych interfejsów użytkownika, programów trudnych w obsłudze i po prostu nieestetycznie wyglądającej zawartości ekranu. Tworząc aplikacje arkuszy kalkulacyjnych dla innych osób, powinieneś szczególną uwagę zwrócić na wygląd programu.

Jak bezpieczne są hasła programu Excel?

O ile mi wiadomo, firma Microsoft nigdy nie ogłaszała, że Excel potrafi zapewnić bezpieczeństwo przetwarzanych informacji, i miała ku temu poważne powody. Tak naprawdę ominięcie systemu zabezpieczania dokumentów programu Excel przy użyciu hasła jest zadaniem stosunkowo prostym. Na rynku dostępnych jest kilka komercyjnych programów służących do łamania haseł takich dokumentów. Co prawda Excel 2007 i późniejsze wersje wydają się oferować nieco lepszy poziom zabezpieczenia niż starsze wersje, ale mimo to odpowiednio zdeterminowany użytkownik może takie hasło złamać. Wnioski? Nigdy nie zakładaj, że ochrona dokumentu przy użyciu hasła jest wystarczająco pewna. W przypadku szeregowego użytkownika taka ochrona może być w większości przypadków wystarczająca, ale jeżeli komuś naprawdę zależy na złamaniu takiego hasła, to naprawdopodobniej będzie w stanie tego dokonać.

Wbrew pozorom wygląd programu może być dla użytkowników niezmiernie istotnym czynnikiem niemal przesądzającym o jego popularności i to samo sprawdza się w przypadku interfejsów użytkownika aplikacji projektowanych w programie Excel. Piękno — jak powiadają — jest jednak w oku patrzącego. Jeżeli czujesz, że Twoje umiejętności są bardziej związane z naukami ścisłymi, podczas projektowania interfejsu użytkownika skorzystaj z pomocy kogoś, kto posiada większe wyczucie estetyki.

Dobra informacja jest taka, że począwszy od wersji Excel 2007 wprowadzono wiele nowych mechanizmów zdecydowanie ułatwiających tworzenie estetycznie wyglądających arkuszy kalkulacyjnych. Jeżeli będziesz korzystał z predefiniowanych stylów komórek, Twoje dzieło będzie miało dużą szansę na efektowny wygląd. Co więcej, za pomocą kilku prostych kliknięć myszą możesz nałożyć zupełnie nowy styl, całkowicie zmieniający wygląd skoroszytu — przy czym arkusz nadal będzie wyglądał schludnie i efektownie.

Użytkownicy aplikacji doceniają dobrze wyglądający interfejs użytkownika. Jeżeli poświęcisz trochę czasu, aby zadbać o jego estetykę, wygląd aplikacji będzie bardziej dopracowany i profesjonalny. Dobrze prezentująca się aplikacja świadczy o tym, że projektantowi na niej zależało, więc postarał się i poświęcił dodatkowy czas. Tworząc interfejs użytkownika, warto wziąć pod uwagę kilka prostych sugestii:

- **Dążenie do spójności interfejsu.** Tworząc na przykład okna dialogowe, spróbuj wszędzie, gdzie to tylko możliwe, imitować wygląd i sposób obsługi okien dialogowych Excela. Zachowaj zgodność pod względem formatowania, czcionek, wielkości tekstu i kolorów.
- **Dążenie do prostoty.** Częstym błędem projektantów są próby pomieszczenia na pojedynczym ekranie lub w oknie dialogowym zbyt wielu informacji. Dobra zasadą jest prezentowanie w danej chwili tylko jednego lub dwóch fragmentów informacji.
- **Podział okien służących do wprowadzania danych.** Jeśli używasz okna służącego do pobrania danych od użytkownika, weź pod uwagę podzielenie go na kilka mniejszych i przejrzystszych. Jeśli korzystasz ze złożonego okna dialogowego, możesz je podzielić przy użyciu kontrolki MultiPage (umożliwia tworzenie dobrze znanych okien dialogowych podzielonych na karty).
- **Oszczędne używanie kolorów.** Rzadko używaj kolorów. Bardzo łatwo pod tym względem przesadzić i spowodować, że okno aplikacji będzie zbyt pstrokate.

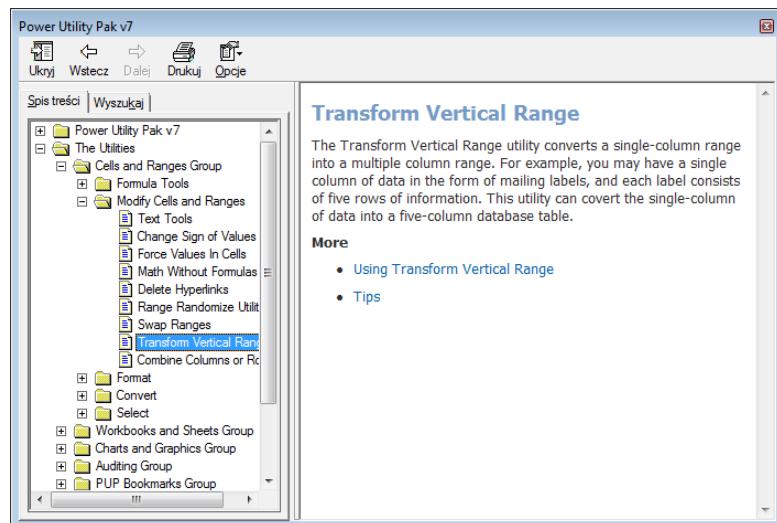
- **Kontrolowanie typografii i grafiki.** Zwróć uwagę na formaty liczbowe, a także zachowaj zgodność kroju i rozmiaru czcionek oraz ramek.

Określanie jakości walorów estetycznych jest bardzo subiektywne. W razie wątpliwości staraj się zachować prostotę i przejrzystość.

Tworzenie systemu pomocy i dokumentacji przeznaczonej dla użytkownika

Dostępne są dwie formy dokumentacji pomocy — papierowa i elektroniczna. Aplikacje systemu Windows standardowo posiadają system pomocy w wersji elektronicznej. Na szczęście aplikacje przeznaczone dla Excela też mogą posiadać system pomocy, nawet w wersji kontekstowej. Opracowanie zawartości pomocy wiąże się z dość sporym, dodatkowym nakładem pracy, ale w przypadku dużego projektu może się opłacić. Na rysunku 4.6 przedstawiono przykładowy system pomocy w postaci skompilowanego pliku w formacie HTML.

Rysunek 4.6.
Przykładowy system pomocy w postaci skompilowanego pliku w formacie HTML



Kolejnym zagadnieniem, które należy również rozważyć i ewentualnie zaplanować, jest wsparcie techniczne dla aplikacji. Innymi słowy, decydując się na takie rozwiązanie, powinieneś na przykład określić, kto odbierze telefon, gdy użytkownik będzie chciał zgłosić problem. Jeśli nie jesteś przygotowany, aby udzielać odpowiedzi na bieżące pytania, musisz znaleźć kogoś, kto się tym zajmie. W niektórych przypadkach będziesz chciał otrzymywać jedynie informacje o błędach oraz pytania typowo techniczne.



W rozdziale 22. omówionych zostało kilka alternatywnych metod oferowania pomocy dla użytkowników aplikacji.

Dokumentowanie prac projektowych

Skompletowanie aplikacji arkusza kalkulacyjnego to jedno, natomiast sprawienie, aby była zrozumiała dla innych osób, to drugie. Podobnie jak w przypadku tradycyjnego programowania, tak i tu ważne jest, aby dokładnie dokumentować prace projektowe. Taka dokumentacja przyda się, gdy trzeba będzie cofnąć się do wcześniejszego etapu projektu. Posłuży również każdemu, komu aplikacja zostanie przekazana.

W jaki sposób tworzysz dokumentację aplikacji opartej na skoroszycie? Jej zawartość można umieścić w arkuszu lub w innym pliku. Można nawet dla własnej wygody stworzyć wersję papierową. Prawdopodobnie najprostszy sposób przechowywania komentarzy i kluczowych informacji o projekcie polega na zastosowaniu oddzielnego arkusza. W kodzie źródłowego języka VBA powinieneś umieszczać dużą liczbę komentarzy (wiersz kodu źródłowego języka VBA rozpoczynający się znakiem apostrofu jest ignorowany i traktowany jako komentarz). Nawet najbardziej elegancki, niezawierający komentarzy fragment kodu, który dzisiaj wydaje Ci się oczywisty, po kilku miesiącach nadal będzie elegancki, ale jego sposób działania i przeznaczenie może nie być już takie oczywiste.

Przekazanie aplikacji użytkownikom

Ukończyłeś projekt i jesteś gotowy do przekazania go końcowym użytkownikom. W jaki sposób to zrobisz? Do wyboru masz różne metody dystrybucji. Wybór najodpowiedniejszej zależy od wielu czynników.

W najprostszym wariantie możesz po prostu przekazać użytkownikowi dysk CD-ROM z aplikacją, napisać instrukcję i mieć to z głowy. Możesz sam zainstalować aplikację, ale nie zawsze jest to możliwe. Kolejną opcją jest stworzenie oficjalnego programu instalacyjnego, który automatycznie przeprowadzi taką operację. Można go napisać przy użyciu zwykłego języka programowania, zakupić uniwersalny program instalacyjny lub napisać własny w języku VBA.

Excel korzysta z technologii umożliwiającej projektantom cyfrowe „podpisywanie” aplikacji. Ma to ułatwić użytkownikom identyfikację jej twórcy, zagwarantować, że projekt nie został zmodyfikowany, i zapobiec rozprowadzaniu wirusów makr lub innego potencjalnie destrukcyjnego kodu źródłowego. W celu „podpisania” projektu wystarczy zwrócić się do urzędu zarządzającego cyfrowymi certyfikatami z prośbą o jego przyznanie. Projekt można też „podpisać”, tworząc własny certyfikat. Więcej szczegółowych informacji na ten temat znajdziesz w systemie pomocy Excela lub na stronie internetowej firmy Microsoft.

Aktualizacja aplikacji (kiedy to konieczne)

Czy po zainstalowaniu aplikacji nastąpił oczekiwany koniec projektu? Czy możesz teraz usiąść, zrelaksować się i spróbować zapomnieć o problemach, które napotkałeś (i rozwiązałeś) w trakcie pracy nad aplikacją? Owszem, w rzadkich przypadkach może osiągnięto koniec projektu. Jednak znacznie częściej użytkownicy aplikacji nie będą całkowicie usatysfakcjonowani. Co prawda, aplikacja jest zgodna z wszystkimi *początkowymi*

specyfikacjami, ale z czasem wymagania ulegają zmianie. Użytkownik często korzystający z aplikacji zaczyna zastanawiać się nad innymi zadaniami, które mogłyby wykonywać. Aplikacja wymaga w takim przypadku *aktualizacji*.

Gdy pojawi się konieczność uaktualnienia lub modyfikowania aplikacji, z pewnością docenisz fakt, że od początku starannie ją projektowałeś i w pełni udokumentowałeś wykonaną pracę. Jeżeli jednak tego nie zrobiłem, to... No cóż, zazwyczaj najlepiej uczymy się na własnych błędach...

Pozostałe kwestie dotyczące projektowania

W trakcie projektowania aplikacji konieczne jest uwzględnienie kilku innych kwestii, zwłaszcza gdy nie wiesz dokładnie, kto ma być użytkownikiem końcowym. Jeżeli tworzyś aplikację, która będzie powszechnie używana (np. na przykład na licencji *shareware*), nie dowiesz się, w jaki sposób użytkownicy będą z niej korzystali, na jakim systemie zostanie uruchomiona lub jakie inne programy w tym samym czasie będą aktywne.

Wersja Excela zainstalowana przez użytkownika

Wraz z pojawieniem się kolejnej, nowej wersji Excela powraca bardzo ważna kwestia kompatybilności. Gdy pisałem ten rozdział, Excel 2013 właśnie pojawił się w sprzedaży, ale wiele dużych korporacji nadal korzystało z wersji 2003 (a nawet starszych).

Niestety nie ma gwarancji, że aplikacja stworzona z myślą na przykład o Excelu 2003 będzie idealnie działała w przypadku jego nowszych wersji. Jeśli zależy Ci, aby aplikacja współpracowała z różnymi wersjami Excela, najlepszym rozwiązaniem jest skorzystanie z najstarszej z nich, a następnie dokładne przetestowanie aplikacji pod kątem zgodności z wszystkimi pozostałyimi.

Sprawy skomplikują się jeszcze bardziej, gdy pod uwagę weźmie się podwersje Excela. Microsoft rozprowadza dodatki *SR* (ang. *Service Release*), *SP* (ang. *Service Pack*) i aktualizacje, które usuwają wykryte błędy. W niektórych przypadkach, Twoja aplikacja może nie pracować poprawnie dopóty, dopóki użytkownik nie zainstaluje określonego dodatku czy aktualizacji.



Więcej szczegółowych informacji na temat kompatybilności poszczególnych wersji Excela zamieszczono w rozdziale 24.

Wersje językowe

Możesz uważać się za dużego szczęściarza, jeśli wszyscy końcowi użytkownicy Twojej aplikacji korzystają z anglojęzycznej wersji Excela. Inne wersje językowe Excela nie będą z nią w pełni zgodne, co oznacza konieczność wykonania dodatkowych testów. Ponadto powinieneś mieć świadomość, że dwóch użytkowników tej samej wersji Excela może korzystać z różnych ustawień regionalnych systemu Windows, co w niektórych przypadkach może prowadzić do pojawienia się problemów z tym związanych.



Niektóre zagadnienia dotyczące wersji językowych zostały pokrótkę omówione w rozdziale 24.

Wydajność systemu

Prawdopodobnie zaliczasz się do zaawansowanych użytkowników komputera i masz tendencję do rozsądnego modernizowania sprzętu. Innymi słowy, dysponujesz dość wydajnym systemem, który prawdopodobnie jest lepszy od komputera przeciętnego użytkownika. Czasami będziesz dokładnie wiedział, z jakiego sprzętu korzystają końcowi użytkownicy aplikacji. W takim przypadku szczególnie ważne jest, aby testy aplikacji przeprowadzić przy użyciu podobnego systemu. Wykonanie danej procedury na Twoim komputerze może być niemal niezauważalne, ale na innym, starszym czy mniej wydajnym, może zająć kilka czy nawet kilkadziesiąt sekund. Pamiętaj jednak, że w niektórych sytuacjach tych kilka czy kilkadziesiąt sekund może być nie do przyjęcia.



Po zdobyciu większego doświadczenia w zakresie języka VBA przekonasz się, że istnieją metody pozwalające po prostu na wykonanie określonego zadania oraz metody pozwalające na szybkie wykonanie takiego zadania. Powinieneś wyrobić sobie nawyk tworzenia wydajnego kodu źródłowego, zoptymalizowanego pod kątem szybkości działania. Pozostałe rozdziały książek z pewnością pomogą Ci w osiągnięciu tego celu.

Tryby karty graficznej

Jak łatwo się można domyślić, użytkownicy korzystają z przeróżnych kart graficznych. Obecnie najczęściej używanymi trybami graficznymi są rozdzielczości karty graficznej wynoszące 1280×1024 oraz 1024×768 . Systemy korzystające z rozdzielczości 800×600 pomału wychodzą obecnie z użycia, aczkolwiek od czasu do czasu można je jeszcze spotkać. Monitory o wysokich rozdzielczościach oraz systemy dwumonitorowe stają się coraz popularniejsze. Pamiętaj, że tylko na postawie tego, że Ty posiadasz monitor pozwalający ustawić bardzo wysokie rozdzielczości, nie możesz zakładać, że każdy nim dysponuje.

Odpowiednia rozdzielczość karty graficznej może stanowić problemu, jeśli aplikacja korzysta z informacji wyświetlanych na pojedynczym ekranie. Na przykład, jeżeli projektujesz okno wprowadzania danych oparte na rozdzielczości 1280×1024 , to użytkownicy dysponujący monitorem o rozdzielczości 1024×768 nie będą mogli zobaczyć całego okna bez korzystania z paska przewijania.

Pamiętaj również o tym, że po *przywróceniu* okna skoroszytu na ekran (w sytuacji, kiedy nie jest ani zmaksymalizowane, ani zminimalizowane) jest ono wyświetlane w poprzednim położeniu i rozmiarze. W krańcowym przypadku okno zapisane przy wyższej rozdzielczości może całkowicie zniknąć z ekranu po otwarciu go na komputerze wyposażonym w monitor bądź kartę graficzną o niższej rozdzielczości.

Niestety nie istnieje metoda automatycznego skalowania okien tak, aby były wyświetlane w taki sam sposób niezależnie od rozdzielczości karty graficznej czy monitora. W niektórych przypadkach istnieje możliwość powiększenia arkusza (za pomocą suwaka *Powiększenie*, znajdującego się w prawej części paska stanu), ale uzyskanie stabilnych wyników

tej operacji może być trudne. Jeśli nie jesteś pewien, jaką rozdzielczość karty graficznej zastosują użytkownicy aplikacji, zaprojektuj ją w oparciu o najmniejszą możliwą rozdzielczość, czyli na przykład 800×600 lub 1024×768 .

Jak dowiesz się w dalszej części książki (patrz rozdział 8.), rozdzielczość karty graficznej użytkownika można określić za pomocą wywołań API systemu Windows, wykonywanych z poziomu kodu źródłowego języka VBA. W niektórych sytuacjach za pomocą programu można odpowiednio modyfikować ustawienia aplikacji zależne od rozdzielczości karty graficznej i monitora wykorzystywanych przez użytkownika.

Część II

Język Visual Basic for Applications

W tej części:

Rozdział 5. „Wprowadzenie do języka VBA”

Rozdział 6. „Podstawy programowania w języku VBA”

Rozdział 7. „Tworzenie procedur w języku VBA”

Rozdział 8. „Tworzenie funkcji w języku VBA”

Rozdział 9. „Przykłady i techniki programowania w języku VBA”

Rozdział 5.

Wprowadzenie do języka VBA

W tym rozdziale:

- Visual Basic for Applications — wbudowany język programowania programu Excel
- Czym VBA różni się od tradycyjnego języka makr oraz czym różni się od języka Visual Basic
- Jak korzystać z edytora VBE?
- Jak używać okna dialogowego *Code* oraz jak dostosować edytor VBE do własnych potrzeb
- Jak korzystać z rejestratora makr programu Excel
- Obiekty, kolekcje, właściwości i metody
- Tajemnice obiektu Comment
- Przykłady zastosowań obiektów Range
- Gdzie szukać informacji o obiektach, właściwościach i metodach

Podstawowe informacje o języku BASIC

Doświadczony programista zapewne parsknąłby śmiechem, gdyby zaproponowano mu programowanie w języku BASIC. Sama nazwa języka (skrót od *Beginner's All-purpose Symbolic Instruction Code* — uniwersalny kod instrukcji symbolicznych dla początkujących) sugeruje, że nie jest to język dla profesjonalistów. Tak naprawdę BASIC został stworzony na początku lat 60. na potrzeby kształcenia studentów w metodach programowania. Język szybko się przyjął i obecnie jest dostępny w kilkuset odmianach dla wielu typów komputerów.

Przez lata BASIC był rozwijany i ulepszany. We wczesnych implementacjach był to język *interpretowany*. Przed wykonaniem każdy wiersz kodu był interpretowany, co obniżało wydajność (wielokrotnie wykonywany wiersz kodu był za każdym razem powtórnie

interpretowany). Nowsze odmiany języka BASIC umożliwiają *kompilowanie* kodu (czyli inaczej mówiąc, konwersję kodu źródłowego programu na kod maszynowy), dzięki czemu cały program działa o wiele szybciej.

BASIC zdobył spore uznanie w 1991 roku, gdy Microsoft wprowadził do sprzedaży produkt o nazwie Visual Basic for Windows. Produkt ten umożliwił szerokim rzeszom programistów tworzenie własnych, efektownych aplikacji dla systemu Windows. Visual Basic co prawda nie ma już zbyt wiele wspólnego z wcześniejszymi wersjami języka BASIC, ale to właśnie Visual Basic stał się fundamentem, na bazie którego powstał Visual Basic for Applications.

Język VBA

Excel 5 był pierwszą aplikacją na rynku wyposażoną w język VBA. Język VBA należy traktować jako ogólny, skryptowy język Microsoftu służący do tworzenia aplikacji. VBA jest dołączany do większości aplikacji pakietu *Office 2013*, a często nawet wchodzi w skład pakietów aplikacji innych producentów. A zatem, jeżeli opanowałeś język VBA korzystając z Excela, będziesz mógł od razu zacząć pisać makra dla innych produktów firmy Microsoft, a nawet pakietów innych producentów oprogramowania. Co więcej, będziesz nawet w stanie tworzyć kompletnie rozwiązania wykorzystujące funkcje oferowane przez różne aplikacje.

Modele obiektowe

Kluczem do zastosowaniem języka VBA w innych aplikacjach jest zrozumienie *modelu obiektowego* każdej z nich. Język VBA pozwala po prostu na manipulowanie obiektami, a każdy produkt (Excel, Word, Access, PowerPoint itp.) posiada własny unikatowy model obiektowy. Aplikacja może zostać utworzona przy użyciu obiektów udostępnionych przez program, na którym bazuje.

Przykładowo model obiektowy Excela oferuje kilka bardzo wydajnych obiektów analizy danych, takich jak arkusze, wykresy, tabele przestawne, scenariusze, oraz sporą liczbę funkcji matematycznych, finansowych, inżynierskich i biznesowych. Język VBA umożliwia używanie tych obiektów i projektowanie zautomatyzowanych procedur. Używając języka VBA, stopniowo zapoznasz się z modelem obiektowym Excela. ***Uwaga*** — początki będą naprawdę trudne, jednak przy odrobinie wytrwałości wszystkie elementy zaczynają się układać w całość i w pewnym momencie po prostu uświadomisz sobie, że to opanowałeś!

Porównanie języka VBA z językiem XLM

Zanim na rynku pojawiła się wersja 5.0, Excel oferował bardzo bogaty w możliwości (ale jednocześnie bardzo zagadkowy...) język makr o nazwie *XLM*. Co prawda w nowszych wersjach Excela (włącznie z wersją 2013) nadal można uruchamiać makra XLM, ale począwszy od wersji Excel 97, usunięto możliwość ich rejestrowania. Jako projektant powinieneś wiedzieć o istnieniu języka XLM (na wypadek spotkania się kiedykolwiek z makrami napisanymi w tym języku), ale przy tworzeniu aplikacji powinieneś już korzystać z języka VBA.

Czy VBA staje się już przestarzały?

W ciągu kilku ostatnich lat słyszałem wiele plotek, jakoby Microsoft planował usunąć VBA z aplikacji pakietu Office i zastąpić go środowiskiem .NET (lub jeszcze czymś innym). Osobiście uważam, że te plotki są całkowicie bezpodstawne. Oczywiście firma Microsoft pracuje nad nowymi rozwiązaniami umożliwiającymi automatyzację zadań w aplikacjach pakietu Office, ale VBA będzie z nami jeszcze przez długi czas — przynajmniej w Excelu na platformie Windows. Warto zauważyć, że nie tak dawno firma Microsoft usunęła VBA z Excela na platformie Macintosh, ale po pewnym czasie przywróciła go ponownie!

Dlaczego VBA przetrwa? Ponieważ obecnie na rynku funkcjonują dosłownie miliony różnych rozwiązań i aplikacji opartych na VBA, a co więcej, VBA jest o wiele łatwiejszy do opanowania dla programisty niż obecnie proponowane rozwiązania alternatywne.



Nie powinieneś mylić języka makr XLM z językiem XML (ang. *eXtensible Markup Language*). Co prawda nazwy obu języków są bardzo podobne i używają tych samych liter, ale poza tym nie mają ze sobą nic wspólnego. XML to format zapisu danych strukturalnych. Aplikacje pakietu Office 2013 używają XML-a jako domyślnego formatu zapisu dokumentów.

Wprowadzenie do języka VBA

Zanim przejdziemy do konkretów, powinieneś dokładne zapoznać się z materiałem zawartym w tym podrozdziale, który stanowi obszerny przegląd zagadnień omawianych w pozostałej części rozdziału.

Poniższa lista to krótkie (i nieco chaotyczne) podsumowanie tego, z czym będziesz się spotykał programując w VBA:

- **Kod programu** — Wykonanie kodu napisanego w języku VBA pozwala zrealizować żądane operacje. Kod VBA wpisywany w edytorze lub tworzony za pomocą rejestratora makr jest umieszczany w module kodu VBA.
- **Moduł** — Moduły VBA są przechowywane w skoroszcycie Excela, ale do ich przeglądania lub modyfikacji musisz użyć edytora *VBE* (ang. *Visual Basic Editor*). Moduły VBA składają się z szeregu procedur.
- **Procedury** — Procedura to w uproszczeniu jednostka kodu programu realizująca określone zadanie. VBA obsługuje dwa rodzaje procedur: procedury typu *Sub* oraz funkcje (procedury typu *Function*).
 - Procedura *Sub* składa się z serii poleceń i może być wykonywana na kilka sposobów. Poniżej zamieszczono przykład procedury *Sub* o nazwie *Test*, która oblicza prostą sumę, a następnie wyświetla wynik w oknie komunikatu.

```
Sub Test()
    Sum = 1 + 1
    MsgBox "Wynik działania to: " & Sum
End Sub
```

- Procedura *Function*. Procedura *Function* zwraca pojedynczą wartość lub tablicę. Funkcja może być wywołana z innej procedury języka VBA lub być zastosowana w formule arkusza. Oto przykład prostej funkcji o nazwie *AddTwo*:

```
Function AddTwo(arg1, arg2)
    AddTwo = arg1 + arg2
End Function
```

- **Obiekty** — Język VBA operuje na obiektach powiązanej z nim aplikacji (w tym przypadku naszą aplikacją jest oczywiście Excel). Excel oferuje ponad 100 klas obiektów, na których możesz wykonywać różne operacje. Przykładami takich obiektów są skoroszyt, arkusz, zakres arkusza, wykres czy kształt. Oczywiście istnieje znacznie więcej obiektów, które możesz zmieniać przy użyciu kodu programu w języku VBA. Klasy obiektów mają strukturę hierarchiczną.

Obiekty mogą spełniać funkcję kontenerów dla innych obiektów. Przykładowo Excel jest obiektem klasy `Application` i zawiera inne obiekty, takie jak `Workbook`. Obiekt klasy `Workbook` może przechowywać inne obiekty, takie jak `Worksheet` czy `Chart`. Z kolei obiekt klasy `Worksheet` stanowi kontener dla takich obiektów, jak `Range`, `PivotTable` itp. Sposób uporządkowania takich obiektów jest nazywany *modelem obiektowym* programu Excel.

- **Kolekcje** (ang. *Collections*) — Kolekcja to po prostu zbiór podobnych do siebie obiektów. Przykładowo kolekcja `Worksheets` składa się z wszystkich arkuszy określonego skoroszytu. Kolekcje same w sobie również są obiektami.
- **Hierarchia obiektów** — Przy odwoływaniu się do obiektu podrzędnego należy określić jego lokalizację w hierarchii obiektów przy użyciu znaku *kropki*, spełniającej funkcję separatora oddzielającego kontener od obiektu podrzędnego. Na przykład do skoroszytu o nazwie `Zeszyt1.xlsx` można odwołać się w następujący sposób:

```
Application.Workbooks("Zeszyt1.xlsx")
```

Instrukcja odwołuje się do skoroszytu `Zeszyt1.xlsx` zawartego w kolekcji `Workbooks`. Kolekcja `Workbooks` jest zawarta w obiekcie `Application` programu Excel. Po przejściu na kolejny poziom hierarchii w dół, można odwołać się do arkusza `Arkusz1` znajdującego się w skoroszycie `Zeszyt1.xlsx`:

```
Application.Workbooks("Zeszyt1.xlsx").Worksheets("Arkusz1")
```

Po przejściu na kolejny poziom w podobny sposób można się odwołać do wybranej komórki:

```
Application.Workbooks("Zeszyt1.xlsx").Worksheets("Arkusz1").Range("A1")
```

- **Obiekty aktywne** — Jeżeli pominiesz pełne odwołanie do określonego obiektu, Excel domyślnie użyje obiektu aktywnego. Jeżeli aktywnym skoroszytem w danej chwili jest obiekt `Zeszyt1`, poprzednie odwołanie może zostać uproszczone do postaci:

```
Worksheets("Arkusz1").Range("A1")
```

Jeżeli wiesz, że aktywnym arkuszem w danej chwili jest arkusz `Arkusz1`, odwołanie można uprościć jeszcze bardziej:

```
Range("A1")
```

- **Właściwości obiektów** — Obiekty posiadają swoje właściwości (ang. *properties*). Właściwość może być traktowana jako *ustawienie* obiektu. Przykładowo obiekt `Range` posiada takie właściwości jak `Value` i `Address`. Obiekt `Chart` dysponuje

właściwościami HasTitle i Type. Przy użyciu języka VBA można określić i modyfikować właściwości obiektu. Niektóre właściwości są przeznaczone tylko do odczytu i za pomocą języka VBA nie można zmienić ich wartości.

Przy odwoływaniu się do właściwości obiektu nazwę właściwości oddziela się od nazwy obiektu za pomocą kropki. Na przykład do wartości komórki A1 arkusza Arkusz1 można odwołać się za pomocą następującego polecenia:

```
Worksheets("Arkusz1").Range("A1").Value
```

Analogie

Jeżeli lubisz analogie, to oto jedna z nich, która pomoże Ci zrozumieć relacje pomiędzy poszczególnymi obiektami, właściwościami i metodami w VBA. Użyjemy tutaj porównania programu Excel do sieci barów szybkiej obsługi.

Podstawowym elementem Excela jest obiekt Workbook. W przypadku sieci barów szybkiej obsługi takim elementem jest jeden bar. Excel umożliwia tworzenie i zamykanie skoroszytów. Wszystkie otwarte skoroszyty tworzą kolekcję Workbooks (złożoną z obiektów Workbook). Analogicznie, kierownictwo sieci barów może otwierać i zamykać poszczególne bary. Wszystkie bary sieci mogą być potraktowane jak kolekcja Bary złożona z obiektów Bar.

Skoroszyt Excela jest obiektem, ale sam przechowuje inne obiekty, takie jak arkusze, wykresy, moduły VBA itp. Ponadto każdy obiekt zawarty w skoroszycie może posiadać własne obiekty. Na przykład obiekt Worksheet może przechowywać takie obiekty jak Range, PivotTable, Shape itp.

Podobnie bar szybkiej obsługi (nasz odpowiednik skoroszytu) zawiera takie obiekty jak Kuchnia, Jadalnia i Stoły (kolekcja). Ponadto kierownictwo może dodawać lub usuwać obiekty zawarte w obiekcie Bar, na przykład do kolekcji Stoły może dodać więcej stołów. Każdy z tych obiektów może posiadać inne obiekty. Na przykład obiekt Kuchnia przechowuje obiekty takie jak Kuchenka, Wentylator, GłównyKucharz, Zlew itp.

Jak na razie wszystko się zgadza i wydaje się, że nasza analogia się sprawdza. Przekonajmy się, czym będzie tak dalej.

Obiekty Excela posiadają swoje właściwości. Na przykład obiekt Range dysponuje takimi właściwościami jak Value i Name, natomiast obiekt Shape oferuje takie właściwości jak Width, Height itp. Obiekty w naszym barze szybkiej obsługi również posiadają swoje właściwości. Na przykład obiekt Kuchenka ma takie właściwości jak Temperatura i LiczbaPalników. Obiekt Wentylator także posiada zbiór swoich właściwości, takich jak Włączony, LiczbaObrotówNaMinutę itd.

Oprócz właściwości obiekty Excela posiadają również swoje metody, które pozwalają na wykonywanie operacji na tych obiektach. Na przykład metoda ClearContents usuwa zawartość obiektu Range. Obiekty powiązane z barem szybkiej obsługi również posiadają metody. Z łatwością możemy sobie wyobrazić dla obiektu Kuchenka metodę ZmieńUstawieniaTermostatu czy metodę Włącz dla obiektu Wentylator.

W przypadku Excela metody mogą czasami zmieniać właściwości obiektów, na przykład metoda ClearContents obiektu Range zmienia jego właściwość Value. Podobnie metoda ZmieńUstawieniaTermostatu obiektu Kuchenka zmienia jego właściwość o nazwie Temperatura.

Język VBA umożliwia pisanie procedur wykonujących operacje na obiektach Excela. W przypadku baru szybkiej obsługi kierownictwo może wydać polecenie wykonania operacji związanych z obiektami znajdującymi się w barze (np. „włącz kuchenkę i przełącz wentylator na wysokie obroty”). Czy teraz wszystko jest już jasne?

- **Zmienne języka VBA** — Zmiennym języka VBA można przypisywać różne wartości. Zmienną możesz traktować jako nazwę służącą do przechowywania określonej wartości. Na przykład, aby przypisać zmiennej Interest wartość komórki A1 arkusza Arkusz1, powinieneś użyć poniższej instrukcji języka VBA:

```
Interest = Worksheets("Arkusz1").Range("A1").Value
```

- **Metody obiektów** — Obiekty posiadają swoje metody. *Metoda* to operacja, która jest wykonywana na obiekcie. Na przykład jedną z metod obiektu Range jest metoda ClearContents, która powoduje wyczyszczenie zawartości zakresu. Aby odwołać się do danej metody należy za nazwą obiektu wstawić kropkę, a następnie nazwę metody. Na przykład, aby wyczyścić zawartość komórki A1 aktywnego arkusza, powinieneś skorzystać z następującego polecenia:

```
Range("A1").ClearContents
```

- **Standardowe elementy języka programowania** — Język VBA posiada wiele standardowych elementów spotykanych w nowoczesnych językach programowania, takie jak tablice, pętle itd.
- **Zdarzenia** — Niektóre obiekty potrafią rozpoznawać wystąpienie określonych zdarzeń i dzięki temu możesz napisać odpowiedni kod VBA, który będzie automatycznie wykonywany po zaistnieniu danego zdarzenia. Na przykład otwarcie skoroszytu generuje zdarzenie o nazwie Workbook_Open, a zmiana zawartości komórki arkusza — zdarzenie Worksheet_Change.

Możesz wierzyć lub nie, ale powyższe zestawienie całkiem dobrze opisuje język VBA i jego sposób działania z programem Excel. Cała reszta to już tylko kwestia opanowania szczegółów.

Edytor VBE

Tworzenie i modyfikacja kodu języka VBA odbywa się przy użyciu edytora VBE (ang. *Visual Basic Editor*). Edytor Visual Basic jest co prawda oddzielną, ale bardzo ściśle zintegrowaną z Exceliem aplikacją. Pisząc „zintegrowany”, mam na myśli to, że kiedy potrzebujesz VBE, Excel bierze na siebie wszystkie działania niezbędne do jego uruchomienia. Nie możesz uruchomić edytora Visual Basic bez uprzedniego uruchomienia programu Excel.



Moduły VBA są przechowywane w plikach skoroszytów, ale są widoczne tylko i wyłącznie po uruchomieniu edytora VBE.

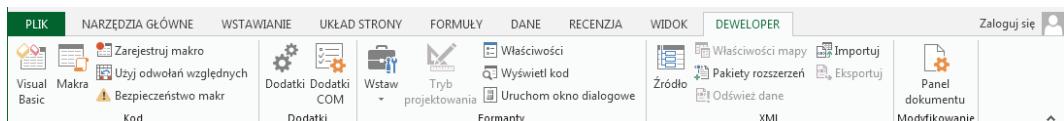
Wyświetlanie karty DEVELOPER

Karta *DEVELOPER* nie jest wyświetlana domyślnie na *Wstążce* programu Excel. Jeżeli jednak masz zamiar pracować z programami VBA, powinieneś włączyć na stałe wyświetlanie tej karty. Aby to zrobić, wykonaj polecenia przedstawione poniżej:

1. Kliknij *Wstążkę* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Dostosuj Wstążkę*.

2. Na ekranie pojawi się okno dialogowe *Opcje programu Excel* otwarte na karcie *Dostosowywanie Wstążki*.
3. Na liście znajdującej się po prawej stronie okna zaznacz opcję *Deweloper*.
4. Naciśnij przycisk *OK*.

Po wykonaniu tych poleceń Excel wyświetli na *Wstążce* nową kartę, tak jak to zostało przedstawione na rysunku 5.1.



Rysunek 5.1. Karta *DEVELOPER* domyślnie nie jest wyświetlana na *Wstążce*

Uruchamianie edytora VBE

Pracując z programem Excel, masz możliwość przełączania się do edytora VBE. Aby to zrobić, powinieneś skorzystać z jednej z metod opisanych poniżej:

- Naciśnij kombinację klawiszy *Alt+F11*.
- Przejdź na kartę *DEVELOPER* i naciśnij przycisk *Visual Basic* znajdujący się w grupie opcji *Kod*.

Na rysunku 5.2 przedstawiono wygląd głównego okna edytora VBE. Oczywiście w Twoim przypadku okno edytora VBE może wyglądać nieco inaczej. Użytkownik ma duże możliwości dopasowywania wyglądu tego okna do własnych potrzeb — możesz ukrywać poszczególne okna składowe, zmieniać ich rozmiary, zmieniać układ okien na ekranie itd.

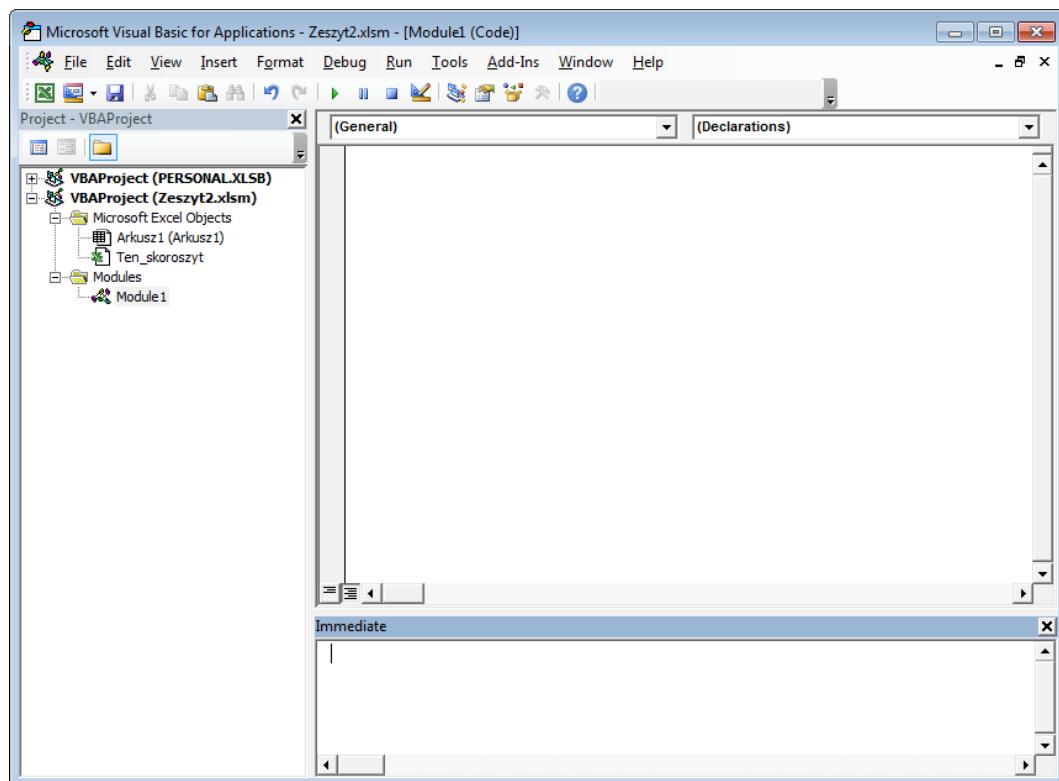
Okna edytora VBE

Edytor VBE składa się z kilku elementów — w kolejnych podpunktach w skrócie omówimy najważniejsze z nich.

- **Pasek menu edytora VBE** — Pomimo, iż Excel używa *Wstążki*, czyli nowego, efektownego interfejsu użytkownika, edytor VBA nadal mocno tkwi w świecie tradycyjnego menu i pasków narzędzi. Pasek menu edytora VBE działa jak każdy inny, tradycyjny pasek menu, z którymi się do tej pory spotykałeś. Znajdziesz tam szereg poleceń wykorzystywanych w pracy z różnymi elementami składowymi edytora. Wiele poleceń posiada powiązane z nimi skróty klawiaturowe. Na przykład: zamiast wybierać z menu głównego polecenie *View/Immediate Window*, możesz po prostu nacisnąć kombinację klawiszy *Ctrl+G*.



Edytor VBE oferuje również menu podrzenne, w którym znajdziesz często używane polecenia. Aby przywołać menu na ekran, kliknij prawym przyciskiem myszy w dowolnym miejscu okna edytora VBE.



Rysunek 5.2. Okno edytora VBE

- **Paski narzędzi edytora VBE** — Pasek narzędzi *Standard*, domyślnie znajdujący się bezpośrednio pod paskiem menu, jest jednym z sześciu dostępnych pasków narzędzi edytora VBE (pasek menu jest traktowany jako pasek narzędzi). Paski narzędzi edytora VBE możesz dostosowywać do własnych wymagań, przemieszczać w inne miejsca, wyświetlać i ukrywać w zależności od potrzeb. Aby wyświetlić lub ukryć wybrany pasek narzędzi, z menu głównego wybierz polecenie *View/Toolbars*.
- **Okno Project Explorer** — W oknie *Project Explorer* znajdziesz diagram zawierający wszystkie skoroszyty aktualnie otwarte w Excelu (w tym także dodatki i skoroszyty ukryte). Każdy skoroszyt jest określany mianem *projektu*. Okno *Project Explorer* bardziej szczegółowo zostanie omówione w podrozdziale zatytułowanym „*Tajemnice okna Project Explorer*” w dalszej części tego rozdziału.
Jeżeli okno *Project Explorer* nie jest widoczne, naciśnij kombinację klawiszy *Ctrl+R*. Aby ukryć okno *Project Explorer*, kliknij przycisk *Zamknij* znajdujący się na jego pasku tytułowym. Zamiast tego możesz również kliknąć prawym przyciskiem myszy dowolne miejsce okna *Project Explorer* i z menu podręcznego wybrać polecenie *Hide*.
- **Okno Code** — Okno *Code* (czasem nazywane oknem *Module*) zawiera kod źródłowy języka VBA. Każdy element projektu posiada powiązane okno zawierające kod źródłowy. Aby dla określonego obiektu otworzyć okno *Code*, należy go

dwukrotnie kliknąć w oknie *Project Explorer*. Na przykład: aby otworzyć okno *Code* dla obiektu *Arkusz1*, powinieneś po prostu dwukrotnie kliknąć go w oknie *Project Explorer*. Jeżeli dany obiekt nie posiada przypisanego kodu VBA, okno *Code* będzie puste.

Kolejna metoda umożliwiająca otworzenie okna *Code* dla obiektu polega na jego zaznaczeniu w oknie *Project Explorer* i kliknięciu przycisku *View Code* znajdującego się na pasku narzędzi w górnej części okna *Project Explorer*.

Okno *Code* zostanie bardziej szczegółowo omówione w podrozdziale „Tajemnice okna *Code*” w dalszej części tego rozdziału.

- **Okno *Immediate*** — Okno *Immediate* jest bardzo użyteczne w przypadku bezpośredniego wykonywania i testowania poleceń języka VBA oraz usuwania błędów z kodu źródłowego. Okno może być widoczne lub ukryte. Jeżeli okno *Immediate* jest niewidoczne, powinieneś nacisnąć kombinację klawiszy *Ctrl+G*. Aby je zamknąć, kliknij przycisk *Zamknij* na pasku tytułowym. Zamiast tego możesz również prawym przyciskiem myszy kliknąć w dowolnym miejscu okna *Immediate* i z menu podręcznego wybrać polecenie *Hide*.

Tajemnice okna *Project Explorer*

Kiedy pracujesz z edytorem VBE, każdy otwarty skoroszyt i dodatek Excela jest traktowany jako projekt. *Projekt* można potraktować jako zbiór obiektów uporządkowanych na wzór konspektu. Aby rozwinąć zawartość projektu, należy kliknąć ikonę ze znakiem plus, znajdująca się z lewej strony nazwy projektu w oknie *Project Explorer*. W celu zwinięcia zawartości projektu należy kliknąć ikonę ze znakiem minus, widoczną z lewej strony nazwy projektu. Jeżeli spróbowajesz rozwinąć zawartość projektu chronionego hasłem, zostaniesz poproszony o wpisanie hasła.



W górnej części okna *Project Explorer* znajdują się trzy przyciski. Trzeci przycisk, o nazwie *Toggle Folders*, decyduje o tym, czy obiekty projektu będą wyświetlane jako hierarchia, czy w postaci pojedynczej, prostej listy.

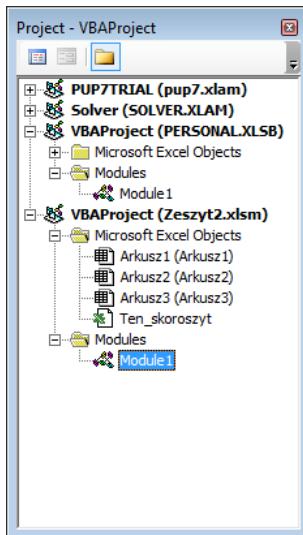
Na rysunku 5.3 pokazano okno *Project Explorer* zawierające cztery projekty (dwa dodatki XLAM i dwa skoroszyty).



Często zdarza się sytuacja, że po uruchomieniu edytora VBE wyświetlony moduł kodu źródłowego nie odpowiada obiektowi zaznaczonemu w oknie *Project Explorer*. Aby upewnić się, że przeglądasz i modyfikujesz odpowiedni moduł kodu źródłowego, zawsze powinieneś najpierw w oknie *Project Explorer* dwukrotnie kliknąć wybrany obiekt lewym przyciskiem myszy.

Jeżeli otwarłeś jednocześnie wiele skoroszytów i dodatków, wygląd okna *Project Explorer* może być nieco przytłaczający. Niestety nie ma możliwości ukrycia projektów wyświetlanych w tym oknie; jedyne co możesz zrobić, to po prostu zwinać (zminimalizować) te projekty, których w danej chwili nie będziesz używać.

Rysunek 5.3.
Okno Project Explorer
wyświetlające
cztery projekty



Dla każdego projektu rozwiniętego w oknie *Project Explorer* wyświetlany jest przynajmniej jeden węzeł o nazwie *Microsoft Excel Objects*. Po jego rozwinięciu będą widoczne poszczególne arkusze i arkusze wykresu zawarte w danym skoroszycie (każdy arkusz jest traktowany jak osobny obiekt). Dodatkowo wyświetlany jest również obiekt o nazwie *ThisWorkbook*, który reprezentuje obiekt *Workbook*. Jeżeli projekt zawiera moduły VBA, na liście pojawi się też węzeł *Modules* reprezentujący moduły kodu VBA. Projekt może również zawierać węzeł o nazwie *Forms*, przechowujący obiekty *UserForm* (inaczej nazywane niestandardowymi oknami dialogowymi). Jeżeli w projekcie są jakieś moduły klas, na liście pojawi się węzeł o nazwie *Class Modules*. Jeżeli projekt posiada jakieś kolwiek odwołania, na liście pojawi się węzeł o nazwie *References*. Węzeł *References* może być nieco mylący, ponieważ odwołania z definicji nie mogą zawierać żadnego kodu VBA.

Dodawanie nowego modułu VBA

Aby do projektu dodać nowy moduł VBA, zaznacz w oknie *Project Explorer* nazwę projektu, a następnie z menu *Insert* wybierz pozycję *Module*. Zamiast tego możesz również prawym przyciskiem myszy kliknąć nazwę projektu i z menu podręcznego wybrać pozycję *Insert*, a następnie *Module*.

Kiedy rejestrujesz makro, Excel automatycznie wstawia odpowiedni moduł VBA zawierający wygenerowany kod źródłowy.

Usuwanie modułu VBA

Jeżeli chcesz usunąć z danego projektu moduł VBA lub moduł klasy, powinieneś w oknie *Project Explorer* zaznaczyć nazwę modułu, a następnie z menu *File* wybrać polecenie *Remove xxx*, gdzie *xxx* to nazwa modułu. Zamiast tego możesz również kliknąć nazwę modułu prawym przyciskiem myszy i z menu podręcznego wybrać polecenie *Remove xxx*. Excel zapyta Cię, czy przed usunięciem modułu chcesz go wyeksportować. Więcej informacji na temat eksportowania modułów znajdziesz w następującym podrozdziale.

Pamiętaj, że nie możesz usuwać modułów kodu źródłowego, które są powiązanych ze skoroszytem (moduł kodu `ThisWorkbook`) lub arkuszem (na przykład moduł kodu dla obiektu `Arkusz1`).

Eksportowanie i importowanie obiektów

Wyjawszy elementy przynależne do węzła `References`, każdy obiekt projektu może zostać zapisany w oddzielnym pliku. Operacja zapisu jednego obiektu jest nazywana *eksportowaniem*. Oczywiście obiekty mogą być również *importowane*. Eksportowanie i importowanie obiektów przydaje się, gdy chcesz użyć określonego obiektu, takiego jak moduł VBA lub formularz `UserForm`, w innym projekcie.

Aby wyeksportować obiekt, powinieneś najpierw zaznaczyć go w oknie *Project Explorer*, a następnie z menu *File* wybrać pozycję *Export File* (lub wcisnąć kombinację klawiszy *Ctrl+E*). Na ekranie pojawi się okno dialogowe *Export File*, w którym zostaniesz poproszony o podanie nazwy pliku. Zwróć uwagę na fakt, że wyeksportowany obiekt nadal pozostaje w projekcie (eksportowana jest jedynie jego kopia). Jeżeli eksportujesz obiekt `UserForm`, wyeksportowany zostanie także każdy kod źródłowy z nim powiązany.

Aby do projektu zaimportować wybrany plik, w oknie *Project Explorer* powinieneś zaznaczyć nazwę projektu, a następnie z menu *File* wybrać pozycję *Import File*. Na ekranie pojawi się okno dialogowe, w którym zostaniesz poproszony o podanie nazwy pliku. Zaimportować możesz tylko taki plik, który został wcześniej wyeksportowany przy użyciu polecenia *File/Export File*.

Wyeksportowane pliki zostają zapisane z rozszerzeniami nazw reprezentującymi typy eksportowanych obiektów. Wyeksportowanie modułu formularzy `UserForm` powoduje wygenerowanie dwóch plików.



Aby skopiować moduł lub obiekt `UserForm` do innego projektu, tak naprawdę nie musisz go eksportować, a następnie importować. Aby to zrobić, wystarczy po prostu otworzyć oba projekty, uaktywnić okno *Project Explorer* i przeciągnąć obiekt z jednego projektu do drugiego. Oryginalny moduł pozostaje na miejscu, a w drugim projekcie tworzona jest jego kopia.

Tajemnice okna Code

Gdy już nabierzesz wprawy w korzystaniu z języka VBA, będziesz spędzał *mnóstwo czasu* na pracy z oknami *Code* (okno kodu źródłowego). Z każdym obiektem projektu jest powiązane osobne okno *Code*. Okna kodu mogą być powiązane z takimi obiektami jak:

- Obiekt skoroszytu (obiekt `ThisWorkbook` w oknie *Project Explorer*).
- Arkusz lub arkusz wykresu zawarty w skoroszycie (na przykład obiekty `Arkusz1` lub `Wykres1` zawarte w oknie *Project Explorer*).
- Moduł VBA.
- *Moduł klasy* (ang. *class module* — specjalny rodzaj modułu umożliwiający tworzenie nowych klas obiektów).
- Formularz `UserForm`.

Minimalizacja i maksymalizacja okien

Kiedy w edytorze Visual Basic otworzysz jednocześnie wiele okien *Code*, może powstać małe zamieszanie. Okna *Code* zachowują się bardzo podobnie jak okna arkuszy Excela. Możesz je minimalizować, maksymalizować, ukrywać, zmieniać ich położenie itd. Wiele osób twierdzi, że najefektywniej pracuje się z oknami *Code* rozciągniętymi do maksymalnych rozmiarów. Dzięki takiemu rozwiążaniu możesz zobaczyć większą ilość kodu źródłowego i skupić się na wykonywanym zadaniu. Aby zmaksymalizować okno *Code*, naciśnij przycisk *Maksymalizuj* znajdujący się na pasku tytułowym okna lub dwukrotnie kliknij pasek tytułu. Aby przywrócić poprzednią wielkość okna *Code*, naciśnij przycisk *Przywróć okno* znajdujący się poniżej paska tytuловego okna aplikacji.

Czasem jednak bardzo korzystne może być jednokrotne wyświetlenie dwóch lub większej liczby okien *Code*, na przykład kiedy chcesz porównać kod źródłowy zawarty w dwóch modułach lub przekopiować kod z jednego modułu do drugiego. Aby wyświetlić więcej okien *Code* jednocześnie, upewnij się, że aktywne okno kodu nie jest zmaksymalizowane, a następnie po prostu zmień odpowiednio rozmiary i położenie wybranych okien kodu.

Minimalizacja okna *Code* spowoduje jego ukrycie. Aby całkowicie zamknąć okno *Code*, naciśnij przycisk *Zamknij* znajdujący się na pasku tytułowym okna. Aby ponownie je otworzyć, wystarczy dwukrotnie kliknąć odpowiedni obiekt w oknie *Project Explorer*.

Z poziomu edytora VBE możesz zapisać otwarty skoroszyt. Żeby to zrobić, zaznacz wybrany skoroszyt w oknie *Project* i następnie z menu głównego wybierz polecenie *File/Save*. Oczywiście zamiast tego możesz również nacisnąć kombinację klawiszy *Ctrl+S* lub kliknąć lewym przyciskiem myszy ikonę polecenia *Save*, znajdującą się na standardowym pasku narzędzi.

Edytor *Visual Basic* nie posiada polecenia pozwalającego na zamknięcie skoroszytu. Aby wykonać taką operację, musisz powrócić do Excela. W razie potrzeby możesz jednak do zamknięcia skoroszytu lub dodatku użyć okna *Immediate*. Aby to zrobić, uaktywnij to okno (jeżeli nie jest widoczne, naciśnij kombinację klawiszy *Ctrl+G*), a następnie wpisz odpowiednie polecenie języka VBA i naciśnij klawisz *Enter*, na przykład

```
Workbooks("mój_dodatek.xls").Close
```

Wykonanie takiego polecenia spowoduje wywołanie metody *Close* obiektu *Workbook*, która zamknie skoroszyt. W tym przypadku skoroszytem jest dodatek.

Przechowywanie kodu źródłowego języka VBA

W oknie kodu możesz przechowywać cztery typy kodu języka VBA:

- **Procedury** Sub. Jak pamiętasz, *procedura* to inaczej zbiór poleceń, które wykonują określone operacje.
- **Procedury Function** (funkcje). *Funkcja* jest zbiorem instrukcji zwracających pojedynczą wartość lub tablicę (pod względem działania przypominają funkcje arkusza, na przykład funkcję *SUMA*).

■ **Deklaracje.** Deklaracja to informacja na temat zmiennej zastosowanej w kodzie źródłowym języka VBA. Na przykład możesz zadeklarować typ danych dla zmiennych, których planujesz użyć w procedurze czy funkcji.

■ **Procedury Property.** To specjalne procedury stosowane w modułach klas.

W pojedynczym module VBA możesz przechowywać dowolną liczbę procedur, funkcji oraz deklaracji. Sposób zorganizowania modułu VBA jest całkowicie zależny od Ciebie. Niektórzy programiści umieszczają cały kod VBA aplikacji w jednym module, z kolei inni preferują podział i przechowanie poszczególnych części kodu w kilku różnych modułach.



Co prawda w zakresie wyboru miejsca przechowywania kodu źródłowego języka VBA dysponujesz dużą swobodą, ale zawsze powinieneś pamiętać, że istnieją jednak pewne ograniczenia. Procedury obsługujące zdarzeń muszą znajdować się w oknie *Code* obiektu, który reaguje na wystąpienie danego zdarzenia. Na przykład, jeżeli napiszesz procedurę wykonywaną podczas otwierania skoroszytu, musi ona zostać umieszczona w oknie *Code* obiektu *ThisWorkbook*, a ponadto taka procedura musi mieć specjalną nazwę. Zagadnienie to stanie się bardziej zrozumiałe po omówieniu zdarzeń (patrz rozdział 17.) i formularzy *UserForm* (patrz III część książki).

Wprowadzanie kodu źródłowego języka VBA

Zanim będziesz mógł czegokolwiek dokonać za pomocą języka VBA, musisz w oknie *Code* umieścić odpowiedni kod programu, mający postać procedury. Procedura składa się z szeregu instrukcji języka VBA. Na razie skupimy się na jednym typie okna *Code* — module VBA.

Kod programu możesz umieścić w module VBA na trzy sposoby:

- **Metoda tradycyjna.** Wprowadzanie kodu źródłowego ręcznie, przy użyciu klawiatury.
- **Kopiowanie i wklejanie.** Kopiowanie kodu z innego modułu (czy listingów zamieszczonych na przykład na stronach WWW) i wklejanie go do modułu, nad którym pracujesz.
- **Rejestrowanie makra.** Korzystanie z rejestratora makr programu Excel, który umożliwia zarejestrowanie wykonywanych operacji i automatyczną zamianę na odpowiedni kod źródłowy języka VBA.

Przerwa na odrobinę terminologii

W tej książce posługujemy się terminami takimi jak *program*, *procedura* czy *makro*. Zazwyczaj programiści używają terminu *procedura* w celu opisania zautomatyzowanego zadania. W przypadku Excela procedura jest również czasami określana mianem *makra*. Z technicznego punktu widzenia procedura może być procedurą typu Sub lub Function, wobec których czasami używa się również określenia *podprogram*. Zazwyczaj tych terminów będziemy używać zamiennie, ale zawsze powinieneś pamiętać, że pomiędzy procedurami Sub i Function występuje istotna różnica. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziałach 7. i 8.

Ręczne wprowadzanie kodu źródłowego

Czasami najprostsza metoda okazuje się najlepsza. Bezpośrednie wprowadzanie kodu z klawiatury wymaga... no cóż, bezpośredniego wpisywania kodu programu przy użyciu klawiatury. Innymi słowy, musisz po prostu ręcznie „wklepać” odpowiedni kod programu. Aby pogrupować elementy logicznie z sobą powiązane, odpowiednie wiersze kodu można poprzedzić wcięciami wykonanymi za pomocą klawisza *Tab*. Przykładem są instrukcje warunkowe pomiędzy instrukcjami *If* i *End If*. Tak naprawdę tworzenie wcięć nie jest konieczne, ale znakomicie ułatwia analizowanie kodu, stąd warto wyrobić sobie nawyk korzystania z tabulatora.

Wprowadzanie i edytowanie kodu źródłowego modułu VBA wygląda dokładnie tak, jak mogłeś się tego spodziewać — kod programu możesz zaznaczać, kopiować lub wycinać, a następnie wklejać w inne miejsce.

Pojedyncze polecenie języka VBA może być tak długie, jak tylko będzie to potrzebne, aczkolwiek dla zachowania czytelności kodu długą instrukcję warto podzielić na dwie lub większą liczbę wierszy. Aby to zrobić, na końcu wiersza po spacji powinieneś wstawić znak podkreślenia, nacisnąć klawisz *Enter* i ciąg dalszy instrukcji umieścić w następnym wierszu. Oto przykład pojedynczego polecenia umieszczonego w czterech wierszach:

```
MsgBox "Nie można znaleźć " & UCASE(SHORTCUTMENUFILE) _  
& vbCrLF & vbCrLF & "Plik powinien być umieszczony w " _  
& ThisWorkbook.Path & vbCrLF & vbCrLF  
& "Aby naprawić błąd, należy ponownie zainstalować aplikację BudgetMan", vbCritical,  
APPNAME
```

Zwróć uwagę, że ostatnie wiersze polecenia zostały wcięte. Wcięcia są opcjonalne, ale pozwalają szybko zorientować się, że te wszystkie wiersze rzeczywiście stanowią jedno wyrażenie.

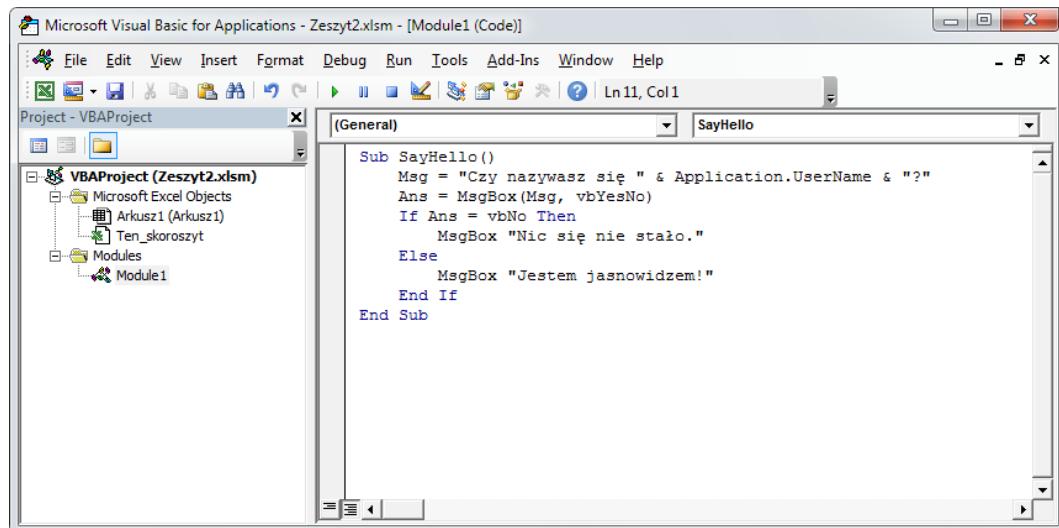


Podobnie jak sam Excel, również edytor *Visual Basic* oferuje wiele poziomów wycofywania i powtarzania operacji realizowanych przez polecenia *Undo* i *Redo*. A zatem jeżeli stwierdzisz, że usunąłeś instrukcję, której nie powinieneś był usuwać, możesz kolejno kliknąć przycisk *Undo* (lub nacisnąć kombinację klawiszy *Ctrl+Z*) aż do momentu jej przywrócenia. Po wykonaniu operacji przywracania możesz wycofać jej efekty, klikając przycisk *Redo* (lub naciskając kombinację klawiszy *Ctrl+Y*). Funkcja ta może „ocalić życie” programisty, stąd warto z nią poeksperymentować tak, aby w pełni zrozumieć zasady jej działania.

Aby przekonać się, jak w praktyce wygląda tworzenie kodu procedury VBA, wstaw nowy moduł VBA do projektu, a następnie wpisz poniższą procedurę w jego oknie *Code*:

```
Sub SayHello()  
    Msg = "Czy nazywasz się " & Application.UserName & "?"  
    Ans = MsgBox(Msg, vbYesNo)  
    If Ans = vbNo Then  
        MsgBox "Nic się nie stało."  
    Else  
        MsgBox "Jestem jasnowidzem!"  
    End If  
End Sub
```

Na rysunku 5.4 przedstawiono wygląd tego modułu VBA.



```
Sub SayHello()
    Msg = "Czy nazywasz się " & Application.UserName & "?"
    Ans = MsgBox(Msg, vbYesNo)
    If Ans = vbNo Then
        MsgBox "Nic się nie stało."
    Else
        MsgBox "Jestem jasnowidzem!"
    End If
End Sub
```

Rysunek 5.4. Twoja pierwsza procedura napisana w języku VBA



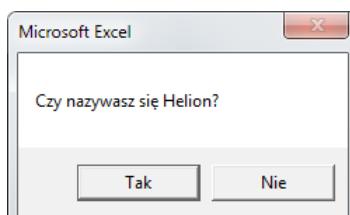
Podczas wprowadzania kodu edytor Visual Basic dokonuje pewnych modyfikacji wpisywanego kodu. Jeżeli na przykład przed lub po znaku równości (=) pominiesz spację, edytor Visual Basic wstawi ją za Ciebie. Dodatkowo, słowa kluczowe języka VBA wyróżniane są innym kolorem. To zupełnie normalne, domyślne zachowanie edytora, które z czasem z pewnością docenisz.

Aby uruchomić procedurę *SayHello*, należy sprawdzić, czy kursor znajduje się w dowolnym miejscu wprowadzonego kodu programu, a następnie wykonać jedną z następujących operacji:

- Nacisnąć klawisz *F5*.
- Z menu *Run* wybrać opcję *Run Sub/User Form*.
- Kliknąć przycisk *Run Sub/UserForm* znajdujący się na pasku narzędzi *Standard*.

Jeżeli kod programu został poprawnie wprowadzony, procedura zostanie wykonana i w efekcie na ekranie pojawi się proste okno dialogowe (patrz rysunek 5.5) zawierające nazwę użytkownika programu Excel (która możesz zdefiniować w oknie dialogowym *Opcje programu Excel*). Zwróć uwagę, że po uruchomieniu makra uaktywniany jest program Excel. Na tym etapie nie jest istotne, czy dokładnie rozumiesz sposób działania kodu naszego programu — wszystko stanie się jasne niebawem, po zakończeniu lektury tego i kolejnych rozdziałów.

Rysunek 5.5.
Wynik wykonania
procedury
przedstawionej
na rysunku 5.4





W większości przypadków makra będą wykonywane z poziomu Excela, aczkolwiek zwykle warto je wcześniej przetestować poprzez bezpośrednie uruchomienie w edytorze Visual Basic.

Przed chwilą utworzyłeś procedurę Sub języka VBA (nazywaną też *makrem*). Po wykonaniu polecenia uaktywniającego makro edytor Visual Basic szybko skompilował i uruchomił jego kod źródłowy. Innymi słowy, poszczególne polecenia zostały przetworzone i Excel po prostu wykonał to, co mu nakazano. Nasze przykładowe makro możesz uruchomić dowolną liczbę razy, chociaż po jakimś czasie cała zabawa z pewnością nie będzie już tak atrakcyjna...

Dla porządku warto zauważyc, że nasza prosta procedura składa się z następujących elementów:

- Deklaracja procedur (pierwszy wiersz).
- Przypisywanie wartości do zmiennych (zmienne Msg i Ans).
- Wykonanie operacji łączeniałańcuchów tekstu (przy użyciu operatora &).
- Korzystanie z wbudowanych funkcji języka VBA (MsgBox).
- Używanie wbudowanych stałych języka VBA (vbYesNo i vbNo).
- Używanie instrukcji warunkowej If-Then-Else.
- Zakończenie procedury (ostatni wiersz).

Nieźle jak na pierwszy raz, prawda?

Kopiowanie kodu źródłowego języka VBA

Inną metodą umieszczania kodu w module VBA jest proste skopiowanie kodu źródłowego z innego modułu. Na przykład wyobraź sobie, że w ramach jednego projektu utworzyłeś procedurę, która może być przydatna również w innym projekcie. Zamiast ponownie mozolnie wprowadzać kod źródłowy, możesz po prostu otworzyć odpowiedni skoroszyt, uaktywnić wybrany moduł, skopiować kod wybranej procedury i następnie wkleić go do aktualnie tworzonego modułu VBA. W razie konieczności po wklejeniu kodu można go zmodyfikować i dopasować do wymagań bieżącego projektu.



Jak już wcześniej wspomniano, w razie potrzeby możesz zaimportować do bieżącego projektu dowolny moduł, który wcześniej został wyeksportowany.

Nie zapominaj również o sieci Internet. Na niezliczonych stronach internetowych, forach dyskusyjnych i blogach, znajdziesz tysiące przykładów kodu VBA, które wystarczy po prostu skopiować do swojego projektu i w razie potrzeby odpowiednio zmodyfikować.

Zwrót uwagę na fakt, że czasami w kodzie kopiowanym ze stron WWW, forów internetowych czy blogów znajdują się cudzysłów drukarskie, a nie proste. Próba uruchomienia programu, w którego kodzie źródłowym znalazły się cudzysłów drukarskie, spowoduje pojawienie się błędu i przerwanie działania programu, stąd powinieneś zawsze upewnić się, że odpowiednio wcześniej zamieniłeś je na cudzysłów proste.

Zastosowanie rejestratora makr

Ostatnia metoda umieszczania kodu źródłowego w module VBA polega na rejestrowaniu wykonywanych operacji przy użyciu rejestratora makr programu Excel.

Niezależnie jednak od tego, jak bardzo będziesz się starał, w żadnym przypadku nie jest możliwe zarejestrowanie kodu takiej procedury jak procedura SayHello, którą utworzyliśmy przed chwilą. Jak widać, rejestrowanie makr jest bardzo przydatną funkcją, ale posiada pewne ograniczenia. W praktyce po zarejestrowaniu makra prawie zawsze trzeba zmodyfikować kod lub ręcznie wprowadzić dodatkowe fragmenty.

Następny przykład demonstruje, w jaki sposób zarejestrować makro, które zmienia orientację strony na poziomą. Jeżeli chcesz samodzielnie zrealizować ten przykład, otwórz pusty skoroszyt i wykonaj poniższe polecenia.

1. Uaktywnij dowolny arkusz skoroszytu.
2. Przejdź na kartę *DEVELOPER* i naciśnij przycisk *Zarejestruj makro* znajdujący się w grupie opcji *Kod*.
3. Na ekranie pojawi się okno dialogowe *Rejestrowanie makra*.
4. Naciśnij przycisk *OK*, aby zaakceptować domyślne ustawienia rejestrowanego makra.

Excel automatycznie umieszcza w projekcie nowy moduł VBA. Od tego momentu wykonywane operacje są zamieniane na kod języka VBA. W trakcie rejestrowania Excel na pasku stanu wyświetla szary kwadracik, którego kliknięcie spowoduje zakończenie rejestrowania makra.

5. Przejdź na kartę *UKLAD STRONY*, następnie do grupy opcji *Ustawienia strony*, naciśnij przycisk *Orientacja* i wybierz opcję *Pozioma*.
6. Przejdź na kartę *DEVELOPER* i naciśnij przycisk *Zatrzymaj rejestrowanie* znajdujący się w grupie opcji *Kod*. Zamiast tego możesz po prostu kliknąć szary kwadracik znajdujący się na pasku stanu.
7. Excel zakończy rejestrowanie wykonywanych operacji.

Aby przyjrzeć się kodowi wygenerowanego makra, uaktywnij edytor Visual Basic (najszybszą metodą jest po prostu naciśnięcie kombinacji klawiszy *Alt+F11*) i zlokalizuj projekt VBA w oknie *Project Explorer*. Rozwiń węzeł *Modules*, dwukrotnie klikając go lewym przyciskiem myszy, a następnie dwukrotnie kliknij lewym przyciskiem myszy moduł *Module1*, aby wyświetlić jego zawartość (jeżeli w projekcie znajdował się już wcześniej moduł *Module1*, nowe makro zostanie umieszczone w module o nazwie *Module2* itd.). Kod źródłowy wygenerowany po wykonaniu prostej zmiany układu strony został przedstawiony na rysunku 5.6.

Możesz być nieco zaskoczony ilością kodu źródłowego wygenerowanego po wykonaniu jednego polecenia (tak właśnie było ze mną, gdy po raz pierwszy utworzyłem takie makro). Pomimo że zmieniłeś tylko jedno proste ustawienie układu strony, Excel wygenerował ponad 50 wierszy kodu źródłowego, zawierających informacje o ustawieniach wszystkich parametrów układu strony.

Rysunek 5.6.

Ogromna ilość kodu wygenerowana przez rejestrator makr programu Excel

```

Sub Makro1()
    '
    ' Makro1 Makro
    '

    Application.PrintCommunication = False
    With ActiveSheet.PageSetup
        .PrintTitleRows = ""
        .PrintTitleColumns = ""
    End With
    Application.PrintCommunication = True
    ActiveSheet.PageSetup.PrintArea = ""
    Application.PrintCommunication = False
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
        .CenterFooter = ""
        .RightFooter = ""
        .LeftMargin = Application.InchesToPoints(0.7)
        .RightMargin = Application.InchesToPoints(0.7)
        .TopMargin = Application.InchesToPoints(0.75)
        .BottomMargin = Application.InchesToPoints(0.75)
        .HeaderMargin = Application.InchesToPoints(0.3)
        .FooterMargin = Application.InchesToPoints(0.3)
        .PrintHeadings = False
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = 600
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
        .Draft = False
        .PaperSize = xlPaperA4
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
        .PrintErrors = xlPrintErrorsDisplayed
        .OddAndEvenPagesHeaderFooter = False
        .DifferentFirstPageHeaderFooter = False
        .ScaleWithDocHeaderFooter = True
        .AlignMarginsHeaderFooter = True
        .EvenPage.LeftHeader.Text = ""
        .EvenPage.CenterHeader.Text = ""
        .EvenPage.RightHeader.Text = ""
        .EvenPage.LeftFooter.Text = ""
        .EvenPage.CenterFooter.Text = ""
        .EvenPage.RightFooter.Text = ""
        .FirstPage.LeftHeader.Text = ""
        .FirstPage.CenterHeader.Text = ""
        .FirstPage.RightHeader.Text = ""
        .FirstPage.LeftFooter.Text = ""
        .FirstPage.CenterFooter.Text = ""
        .FirstPage.RightFooter.Text = ""
    End With
    Application.PrintCommunication = True
End Sub

```

Nasze doświadczenie prowadzi do ważnego wniosku. Użycie rejestratora makr programu Excel nie jest najbardziej efektywnym sposobem tworzenia kodu VBA, ponieważ w zdecydowanej większości przypadków wygenerowany kod źródłowy będzie mocno nadmiarowy. Kiedy przyjrzyisz się uważnie wygenerowanemu kodowi programu, z pewnością zauważysz, że większość polecień jest po prostu niepotrzebna, możesz więc znacznie uprościć nasze makro, po prostu usuwając nadmiarowe polecenia. Dzięki takiemu rady-

kalnemu posunięciu kod makra będzie bardziej przejrzysty, a samo makro będzie działało szybciej, ponieważ nie będzie wykonywało zbędnych operacji. W praktyce nasze makro można uprościć do postaci przedstawionej poniżej:

```
Sub Makrol()
    With ActiveSheet.PageSetup
        .Orientation = xlLandscape
    End With
End Sub
```

Z poprzedniego przykładu usunąłem wszystkie wiersze kodu źródłowego, za wyjątkiem polecenia ustawiającego wartość właściwości `Orientation`. Co ciekawe, nasze makro może zostać jeszcze bardziej uproszczone, ponieważ przy modyfikowaniu tylko jednej właściwości nie jest wymagane stosowanie konstrukcji opartej na instrukcjach `With` i `End With`.

```
Sub Makrol()
    ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

W powyższym przykładzie makro zmienia wartość właściwości `Orientation` obiektu `PageSetup` aktywnego arkusza. Przy okazji należy wspomnieć, że `xlLandscape` jest wbudowaną stałą, której zadaniem jest ułatwienie życia programisty. Stała `xlLandscape` ma wartość 2, natomiast stała `xlPortrait` wartość 1. Poniższe makro daje taki sam wynik jak poprzednie o nazwie `Macrol`.

```
Sub Macrola()
    ActiveSheet.PageSetup.Orientation = 2
End Sub
```

Z pewnością zgodzisz się, że łatwiej zapamiętać nazwy stałych niż różne wartości liczbowe. Więcej szczegółowych informacji na temat stałych zdefiniowanych dla poszczególnych poleceń znajdziesz w systemie pomocy edytora VBE.

Powyższą procedurę możesz bezpośrednio wprowadzić do modułu VBA. Aby to zrobić, powinieneś jednak wiedzieć, jakich obiektów, właściwości i metod należy użyć. Oczywiście zarejestrowanie takiego prostego makra jest znacznie szybsze, a przy okazji dowiedziałeś się też, że obiekt `PageSetup` posiada między innymi właściwość `Orientation`.



W tej książce zdecydowanie podkreślam, że prawdopodobnie jedną z najlepszych metod nauki języka VBA jest rejestrowanie wykonywanych operacji i przeglądanie wygenerowanego kodu. Jeżeli masz wątpliwości, spróbuj zarejestrować wykonywane czynności i przeanalizować wygenerowany kod. Co prawda uzyskane wyniki mogą nie być dokładne takie, jakich oczekiwaleś, ale jest szansa, że dzięki nim Twoja praca będzie postępowała we właściwym kierunku. Aby zapoznać się z obiektami, właściwościami i metodami pojawiającymi się w zarejestrowanym kodzie źródłowym, powinieneś zajrzeć do systemu pomocy edytora VBE.



Rejestrator makr zostanie bardziej szczegółowo omówiony w dalszej części tego rozdziału (patrz podrozdział „Rejestrator makr Excela”).

Dostosowywanie środowiska edytora Visual Basic

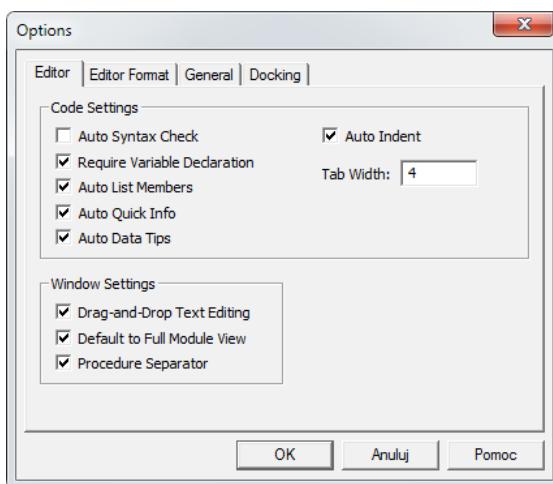
Jeżeli naprawdę chcesz zostać programistą tworzącym aplikacje Excela, musisz przyzwyczaić się do myśli, że spędzasz mnóstwo czasu pracując z edytorem Visual Basic. Aby ułatwić życie programisty, edytor VBE posiada całkiem sporo opcji pozwalających na dostosowanie go do indywidualnych wymagań użytkownika.

Po uruchomieniu edytora Visual Basic z menu *Tools* wybierz polecenie *Options*. Na ekranie pojawi się okno dialogowe zawierające cztery karty — *Editor*, *Editor Format*, *General* oraz *Docking*. W kolejnych podrozdziałach omówimy kilka najważniejszych opcji znajdujących się na tych kartach. Nie pomyl tego okna z oknem dialogowym *Opcje programu Excel*, które jest wyświetlane po przejściu na kartę *Plik* i wybraniu z menu polecenia *Opcje*.

Karta Editor

Na rysunku 5.7 przedstawiono opcje dostępne po przejściu na kartę *Editor* okna dialogowego *Options*.

Rysunek 5.7.
Karta *Editor* okna dialogowego *Options*



Opcja Auto Syntax Check

Opcja *Auto Syntax Check* określa, czy edytor Visual Basic wyświetli okno dialogowe po wykryciu błędu składni w trakcie wprowadzania kodu źródłowego języka VBA. Okno dialogowe zawiera przybliżoną informację o zaistniałym problemie. Jeżeli opcja *Auto Syntax Check* zostanie wyłączona, edytor Visual Basic oznaczy niepoprawny kod innym kolorem niż pozostała część kodu, a okno dialogowe z komunikatem o błędzie nie będzie pojawiało się na ekranie.

Zazwyczaj wyłączam tę opcję, ponieważ wyskakujące niespodziewanie okna dialogowe mnie irytują i zazwyczaj wolę sam sprawdzić, jaki problem wystąpił. Jeżeli jednak dopiero zaczynasz uczyć się języka VBA, opcja *Auto Syntax Check* może okazać się bardzo pomocna.

Opcja Require Variable Declaration

Jeżeli opcja *Require Variable Declaration* zostanie włączona, edytor Visual Basic na początku każdego tworzonego modułu VBA wstawi następujące polecenie:

```
Option Explicit
```

Jeżeli takie polecenie pojawi się w module, będziesz musiał jawnie zadeklarować wszystkie zmienne używane w tym module. Jest to znakomity nawyk, który warto w sobie wyrobić pomimo tego, że wymaga trochę dodatkowych starań. Jeżeli nie zadeklarujesz zmiennych, zostanie im przypisany typ Variant, który jest elastyczny, ale mało wydajny pod względem zajmowanej pamięci i szybkości działania. Więcej szczegółowych informacji na temat deklarowania zmiennych znajdziesz w rozdziale 6.



Zmiana ustawienia opcji *Require Variable Declaration* będzie miała wpływ tylko na ustawienia nowych modułów. Moduły już istniejące nie zostaną zmodyfikowane.

Opcja Auto List Members

Po uaktywnieniu opcji *Auto List Members* edytor Visual Basic będzie oferował pomoc w trakcie wprowadzania kodu źródłowego języka VBA. Pomoc edytora polega na wyświetlanym listy elementów powiązanych z obiektem, takich jak metody i właściwości użytego obiektu.

Jest to bardzo przydatna opcja i dlatego zawsze ją włączam. Na rysunku 5.8 przedstawiono przykład działania opcji *Auto List Members*, który stanie się znacznie bardziej zrozumiałym, kiedy już zaczniesz samodzielnie pisać programy w języku VBA. W naszym przykładzie edytor Visual Basic wyświetla listę elementów powiązanych z obiektem Application. Teraz wystarczy już tylko wybrać z listy odpowiedni element i nacisnąć klawisz Tab (lub po prostu dwukrotnie kliknąć wybrany element lewym przyciskiem myszy). Dzięki takiemu rozwiążaniu możesz uniknąć mozołnego wpisywania nazw metod, właściwości i obiektów, a co więcej takie rozwiązanie gwarantuje też poprawność składni.

Rysunek 5.8.
Przykład działania mechanizmu
Auto List Members

A screenshot of the Microsoft Visual Basic Editor window titled "Zeszyt2.xlsx - Module1 (Code)". The code pane contains the following VBA code:

```
Sub GetUserName()
    Dim TheName As String
    TheName = application.username
End Sub
```

The cursor is positioned at the end of the line "TheName = application.username". In the immediate window pane to the right, a list of members is displayed, with "UserName" highlighted in blue. The list includes: UserControl, UserLibraryPath, UserName, UseSystemSeparators, Value, VBE, and Version. This demonstrates the "Auto List Members" feature, which provides a quick way to complete object names and their properties and methods.

Opcja Auto Quick Info

Po włączeniu opcji *Auto Quick Info* edytor Visual Basic wyświetla informacje o argumentach wprowadzanych funkcji, właściwości i metod. Może to być bardzo przydatne, dlatego zawsze pozostawiam tę opcję włączoną. Na rysunku 5.9 przedstawiono działanie opcji *Auto Quick Info*, która na naszym przykładzie podpowiada poprawną składnię właściwości *Cells*.

Rysunek 5.9.

Przykład działania opcji *Auto Quick Info* oferującej pomoc na temat właściwości *Cells*

```
Sub VBATest()
    Dim InputArea As Range
    Set InputArea = Cells()
End Sub
```

Opcja Auto Data Tips

Po włączeniu opcji *Auto Data Tips* możesz ustawić wskaźnik myszy nad wybraną zmienną, a edytor VBE wyświetli na ekranie jej wartość. Pamiętaj jednak, że mechanizm ten działa tylko wtedy, kiedy wystąpi błąd, działanie procedury zostanie wstrzymane i wejdziesz w tryb wyszukiwania i usuwania błędów (ang. *debugging*). Jest to bardzo użyteczna opcja i sam z pewnością nieraz docenisz jej pomoc. Z tego właśnie powodu zawsze włączam tę opcję.

Opcja Auto Indent

Opcja *Auto Indent* określa, czy edytor Visual Basic automatycznie wstawi na początku każdego nowego wiersza identyczne wcięcie jak w przypadku wcześniej wprowadzonego. Jestem wielkim zwolennikiem wcięć w kodzie źródłowym, dlatego korzystam z tej opcji. Domyslna wartość wcięcia to cztery znaki, ale w razie potrzeby możesz zdefiniować inną szerokość wcięcia.



Do tworzenia wcięć w kodzie źródłowym zamiast spacji powinieneś używać klawisza *Tab*. Dzięki temu kod programu będzie miał bardziej regularne wcięcia. Ponadto w celu usunięcia wcięcia wystarczy nacisnąć kombinację klawiszy *Shift+Tab*. Ta kombinacja klawiszy działa również w przypadku zaznaczenia więcej niż jednego polecenia.

Opcja Drag-and-Drop Text Editing

Włączenie opcji *Drag-and-Drop Text Editing* umożliwia kopiowanie i przenoszenie tekstu za pomocą metody przeciągnij i upuść. W moim przypadku zwykle ta opcja jest włączona, ale szczerze mówiąc, chyba nigdy nie modyfikowałem kodu programu za pomocą przeciągania i upuszczania. Po prostu podczas wycinania, kopiowania i wklejania tekstu wolę korzystać z odpowiednich skrótów klawiszowych.

Opcja Default to Full Module View

Opcja *Default to Full Module View* określa sposób przeglądania procedur. Po jej włączeniu procedury zawarte w oknie kodu będą wyświetlane w pojedynczym, przewijanym oknie. Po wyłączeniu tej opcji w danej chwili widoczna będzie tylko jedna procedura. Zawsze pozostawiam tę opcję włączoną.

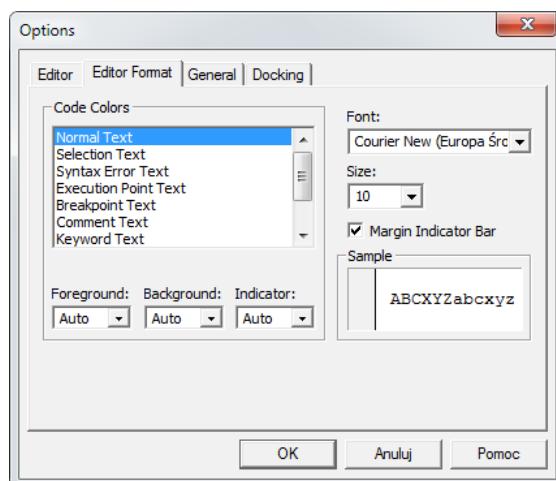
Opcja Procedure Separator

Kiedy opcja *Procedure Separator* jest aktywna, na końcu każdej procedury zawartej w oknie *Code* wyświetlany będzie pasek separatora (przy założeniu oczywiście, że włączona jest również opcja *Default to Full Module View*). Zwykle dobrze jest widzieć, gdzie kończy się kod procedury, dlatego zawsze mam tę opcję włączoną.

Karta Editor Format

Na rysunku 5.10 przedstawiono kartę *Editor Format* okna dialogowego *Options*. Opcje zamieszczone na tej karcie odpowiadają za wygląd edytora VBE.

Rysunek 5.10.
Karta *Editor Format*
okna dialogowego
Options



- **Opcja *Code Colors*** — Opcja *Code Colors* umożliwia wybór koloru tekstu (tła i pierwszego planu) oraz kolorów powiązanych z różnymi elementami kodu źródłowego języka VBA. W tym przypadku główną rolę odgrywają indywidualne upodobania każdego użytkownika. Ze swojej strony uważam, że kolory domyślne

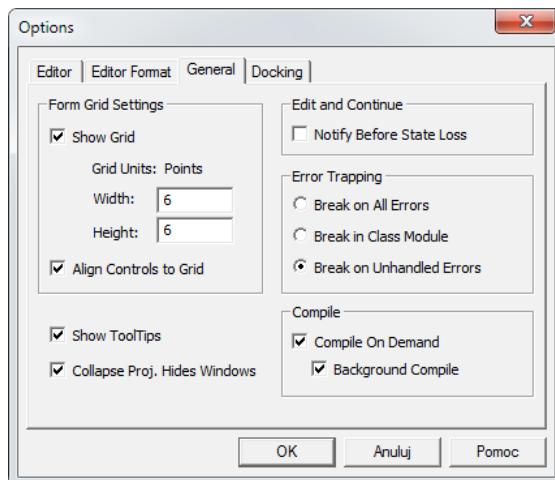
są całkiem odpowiednie, ale pomimo to od czasu do czasu, chcąc zmienić scenerię, możesz zmienić ustawienia tej opcji.

- **Opcja *Font*** — Opcja *Font* umożliwia wybranie czcionki, która zostanie użyta w modułach VBA. Aby uzyskać jak najlepsze wyniki, powinieneś raczej pozostać przy *czcionce o stałej szerokości*, takiej jak na przykład Courier New. W takich czcionkach wszystkie znaki mają dokładnie taką samą szerokość. Takie rozwiązanie powoduje, że kod źródłowy jest bardziej czytelny, ponieważ znaki są ładnie wyrównane w pionie i bardzo łatwo można odnaleźć powielane spacje.
- **Pole *Size*** — Pole *Size* pozwala określić rozmiar czcionki stosowanej w modułach VBA. Wielkość czcionki zależy zwykle od indywidualnych upodobań użytkownika oraz używanej rozdzielczości obrazu. Domyślnie rozmiar czcionki jest ustawiony na 10 punktów, co powinno być w zupełności wystarczające w większości przypadków.
- **Opcja *Margin Indicator Bar*** — Opcja *Margin Indicator Bar* decyduje o tym, czy będzie wyświetlany pasek wskaźnika pionowego marginesu. Opcja powinna być włączona. W przeciwnym razie podczas pracy w trybie wykrywania i usuwania błędów w kodzie źródłowym nie zobaczysz przydatnych graficznych wskaźników szerokości marginesu.

Karta General

Na rysunku 5.11 przedstawiono opcje znajdujące się na karcie *General* okna dialogowego *Options*.

Rysunek 5.11.
Zakładka *General* okna dialogowego *Options*



- **Grupa opcji *Form Grid Settings*** — Opcje w tej grupie są dedykowane do obsługi formularzy *UserForms* i pozwalają na zdefiniowanie rozmiarów siatki ułatwiającej wyrównywanie formantów umieszczanych na formularzach *UserForm*. Jeżeli posiadasz już doświadczenie w projektowaniu takich formularzy, możesz sam zadecydować, czy wyświetlanie siatki jest Ci potrzebne.

- **Opcja *Show ToolTips*** — Opcja *Show ToolTips* pozwala na wyświetlanie podpowiedzi dla przycisków na paskach narzędzi. W zasadzie nie ma potrzeby wyłączenia tej opcji.
- **Opcja *Collapse Proj. Hides Windows*** — Zaznaczenie tej opcji powoduje, że po zminimalizowaniu projektu w oknie *Project Explorer* wszystkie okna kodu powiązane z tym projektem są automatycznie ukrywane. Zawsze pozostawiam tę opcję włączoną.
- **Grupa opcji *Edit and Continue*** — W tej grupie znajdziesz tylko jedną opcję, która może być użyteczna podczas wyszukiwania błędów w kodzie. Po zaznaczeniu tej opcji VBA wyświetla odpowiedni komunikat za każdym razem, kiedy ze względu na jakiś problem dowolna zmienna może utracić wartość.
- **Grupa opcji *Error Trapping*** — Opcje w tej grupie odpowiadają za to, co się wydarzy po wystąpieniu błędu. Jeżeli w swojej aplikacji chcesz umieścić kod, którego zadaniem będzie obsługa błędów, upewnij się, że opcja *Break on Unhandled Errors* jest włączona. Jeżeli wybierzesz opcję *Break on All Errors*, to kod obsługi błędów będzie po prostu ignorowany (co zwykle nie jest tym, czego będziesz oczekiwac). Więcej szczegółowych informacji na temat technik obsługi błędów znajdziesz w rozdziale 7.
- **Grupa opcji *Compile*** — Znajdziesz tutaj dwie opcje odpowiedzialne za komplikację kodu. Zawsze pozostawiam obie opcje włączone. Sam proces komplikacji kodu jest niemal natychmiastowy (oczywiście pod warunkiem, że komplikowany projekt nie ma jakichś gigantycznych rozmiarów).



Odnośnik

Zastosowanie karty Docking

Na rysunku 5.12 przedstawiono kartę *Docking* okna dialogowego *Options*. Opcje znajdujące się na tej karcie pozwalają określić zachowanie różnych okien edytora Visual Basic. Po zadokowaniu okno znajduje się przy jednej z krawędzi głównego okna edytora Visual Basic, dzięki czemu znacznie łatwiej można je zidentyfikować i zlokalizować. Gdy wszystkie opcje znajdują się na karcie *Docking* zostaną wyłączone, okna będą nieuporządkowane i na ekranie powstanie niezłe zamieszanie. Zazwyczaj domyślne ustawienia są uznawane za odpowiednie.

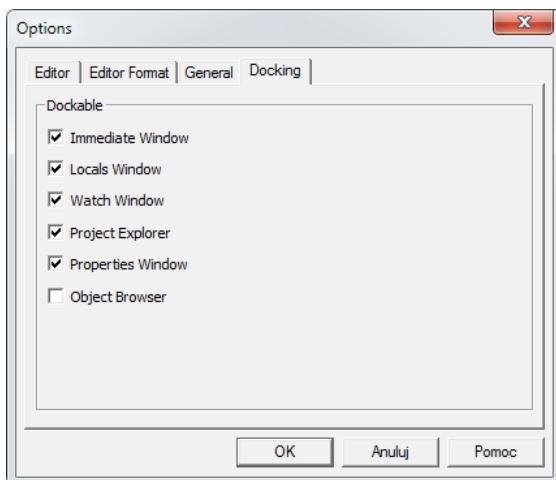
Aby zadokować dane okno, wystarczy przeciągnąć je w żądane miejsce. Na przykład możesz przy lewej krawędzi ekranu zadokować okno *Project Explorer*. Aby to zrobić, złap okno za pasek tytułowy i przeciągnij w lewo aż do momentu, kiedy na ekranie pojawi się obrys zadokowanego okna. Zwolnij przycisk myszy i okno pozostanie na swojej nowej, zadokowanej pozycji.



Uwaga

Dokowanie okien w edytorze Visual Basic zawsze było trochę problematyczne. Niektóre okna nie zawsze dają się od razu zablokować, choć po kilku próbach zwykle się to udaje. Niestety nie znam żadnych sekretów, które wyjaśniałyby takie dziwne zachowanie okien VBE podczas dokowania.

Rysunek 5.12.
Karta Docking okna dialogowego Options



Rejestrator makr Excela

Rejestrator makr — o którym wspomniano już wcześniej — zamienia operacje wykonywane w Excelu na kod źródłowy języka VBA. W tym podrozdziale szczegółowo omówimy zagadnienia związane z rejestratorem makr.



Upewnij się, czy na Wstążce programu Excel jest widoczna karta *DEVELOPER*. Jeżeli nie, powinieneś zająrzeć do podrozdziału „Wyświetlanie karty DEVELOPER” we wcześniejszej części tego rozdziału.

Rejestrator makr jest *wyjątkowo* przydatnym narzędziem, ale zawsze powinieneś pamiętać, że:

- Nadaje się tylko do rejestrowania prostych makr lub niewielkiego fragmentu bardziej złożonego makra.
- Nie wszystkie operacje wykonywane w Excelu dają się zarejestrować.
- Przy użyciu rejestratora makr nie możesz generować kodu źródłowego wykonującego *pętle* (powtarzające się instrukcje), przypisującego zmienne, przetwarzającego instrukcje w oparciu o warunki, wyświetlającego okna dialogowe itp.
- Rejestrator makr zawsze generuje procedury Sub. Nie możesz przy jego użyciu tworzyć funkcji.
- Generowany kod źródłowy zależy od konfiguracji i ustawień rejestratora makr.
- Wygenerowany kod zwykle warto zmodyfikować tak, aby usunąć z niego niepotrzebne, nadmiarowe instrukcje.

Co właściwie zapisuje rejestrator makr?

Rejestrator makr Excela zamienia operacje wykonane za pomocą myszy i klawiatury na kod źródłowy języka VBA. Można by napisać tutaj co najmniej kilka stron opisujących ten proces, ale najlepszą metodą (jak zwykle) będzie zaprezentowanie tego w praktyce. Aby się o tym przekonać, wykonaj polecenia opisane poniżej:

1. Utwórz nowy, pusty skoroszyt.
2. Upewnij się, że okno Excela nie jest zmaksymalizowane — nie powinno zajmować całego ekranu.
3. Uruchom edytor VBE, naciskając kombinację klawiszy *Alt+F11*.

Uwaga: Ponownie upewnij się, że okno edytora VBE nie jest zmaksymalizowane — w przeciwnym razie nie będziesz mógł zobaczyć jednocześnie okien Excela i edytora Visual Basic.

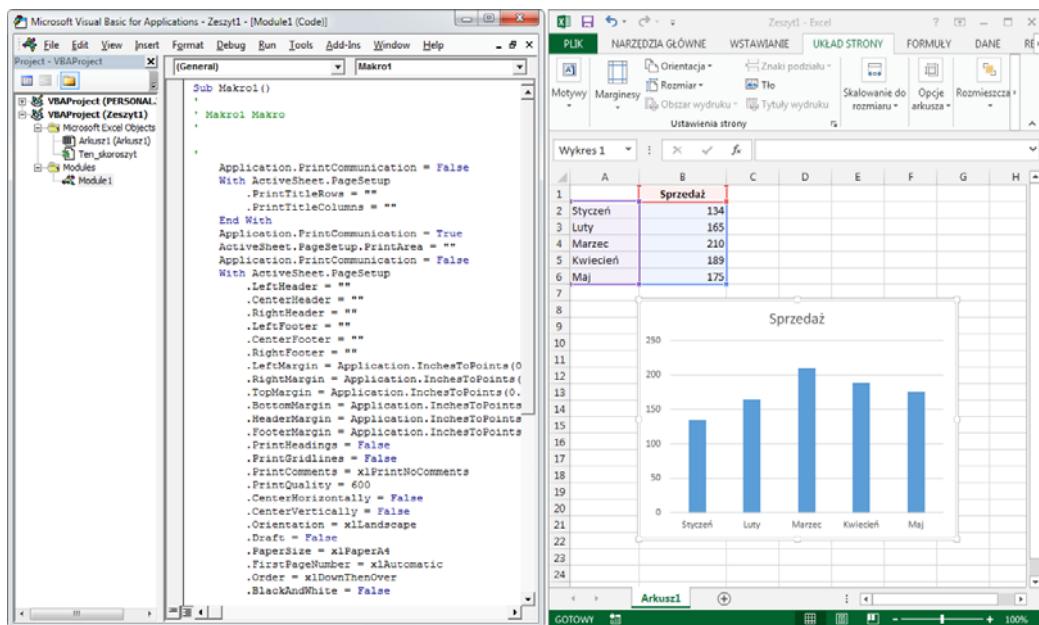
4. Zmień wielkość i lokalizację okien Excela i edytora Visual Basic, tak aby oba były widoczne (w celu uzyskania optymalnego efektu powinieneś zminimalizować wszystkie inne uruchomione aplikacje).
5. Przejdź do Excela, kliknij kartę *DEVELOPER* i naciśnij przycisk *Zarejestruj makro* znajdujący się w grupie opcji *Kod*. Na ekranie pojawi się okno dialogowe *Rejestrowanie makra*. Aby rozpocząć rejestrowanie, naciśnij przycisk *OK*.
6. Przejdź do okna edytora Visual Basic.
7. W oknie *Project Explorer* dwukrotnie kliknij lewym przyciskiem myszy moduł *Module1*.
8. Zamknij okno *Project Explorer*, tak aby okno kodu było wyświetcone na całym ekranie.

Układ okien na ekranie powinien być podobny do pokazanego na rysunku 5.13. Rzeczywisty rozmiar okien będzie zależał od rozdzielczości karty graficznej Twojego komputera. Jeżeli używasz systemu dwumonitorowego, możesz po prostu umieścić okno VBA na pierwszym monitorze, a okno Excela na drugim.

Teraz w arkuszu Excela wykonaj kilka różnych poleceń. Zaznacz i sformatuj komórki, wprowadź dane, użyj wybranych poleceń ze *Wstążki*, utwórz wykres, zmodyfikuj obiekty graficzne itp. W oknie modułu VBA możesz obserwować proces generowania kodu źródłowego.

Odwołania względne czy bezwzględne?

Podczas rejestrowania makr Excel domyślnie używa *odwołań bezwzględnych*. Innymi słowy, kiedy zaznaczasz daną komórkę, program zapamiętuje bezwzględną lokalizację tej komórki, a nie lokalizację odniesioną do aktualnie aktywnej komórki. Aby zrozumieć, jak to działa, wykonaj poniższe polecenia i przeanalizuj otrzymany kod źródłowy.



Rysunek 5.13. Uporządkowanie okien umożliwiające wygodne obserwowanie procesu rejestrowania makra

1. Uaktywnij arkusz i rozpoczęź rejestrowanie makra.
2. Kliknij komórkę B1.
3. W komórce B1 wpisz Styczeń.
4. Przejdz do komórki C1 i wprowadź łańcuch Luty.
5. Kontynuuj proces aż do wprowadzenia w komórkach z zakresu B1:G1 nazw pierwszych sześciu miesięcy roku.
6. Kliknij komórkę B1, aby ją ponownie uaktywnić.
7. Zakończ rejestrowanie makra.

Excel wygeneruje następujący kod źródłowy:

```

Sub Makrol()
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Styczeń"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "Luty"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Marzec"
    Range("E1").Select
    ActiveCell.FormulaR1C1 = "Kwiecień"
    Range("F1").Select
    ActiveCell.FormulaR1C1 = "Maj"
    Range("G1").Select
    ActiveCell.FormulaR1C1 = "Czerwiec"
    Range("B1").Select
End Sub

```

Aby wykonać zarejestrowane makro, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Makra* znajdujący się w grupie opcji *Kod* (zamiast tego możesz nacisnąć kombinację klawiszy *Alt+F8*). Na ekranie pojawi się okno dialogowe *Makro*. Wybierz z listy makro, które chcesz uruchomić (w naszym przypadku *Makro1*), i naciśnij przycisk *Uruchom*.

Po uruchomieniu makra ponownie zostaną powtórzone operacje wykonane w trakcie jego rejestrowania, niezależnie od tego, która komórka była aktywna w momencie uruchomienia makra. Użycie odwołań bezwzględnych podczas rejestrowania zawsze powoduje uzyskanie takich samych wyników.

Zdarzają się jednak sytuacje, w których chciałbyś utworzyć makro, które będzie się odwoływało do nowych komórek *wzgółdem* komórki aktywnej w momencie uruchomienia makra. Na przykład chcesz, aby nasze makro wpisywało nazwy sześciu pierwszych miesięcy roku w sześciu kolejnych, położonych obok siebie komórkach, począwszy od komórki aktywnej. W takiej sytuacji przed zarejestrowaniem makra musisz włączyć opcję korzystania z odwołań względnych.

Rodzaj użytych odwołań możesz zmienić, przechodząc na kartę *DEVELOPER* i naciśkając przycisk *Użyj odwołań względnych*, znajdujący się w grupie opcji *Kod*. Przycisk ten działa jak przełącznik. Kiedy przycisk ten jest wyświetlany w innym kolorze (domyślnie jest to kolor pomarańczowy), rejestrator będzie używał odwołań względnych. Jeżeli przycisk *Użyj odwołań względnych* ma kolor taki, jak pozostałe przyciski, oznacza to, że rejestrator używa odwołań bezwzględnych. Typ używanych odwołań możesz zmienić w dowolnej chwili, nawet w czasie rejestrowania makra.

Aby sprawdzić, jak to działa, wyczyszć zawartość komórek z zakresu B1:G1, a następnie wykonaj polecenia opisane poniżej:

1. Uaktywnij komórkę B1.
2. Przejdź na kartę *Deweloper* i naciśnij przycisk *Zarejestruj makro* znajdujący się w grupie opcji *Kod*.
3. Naciśnij przycisk *OK*, aby rozpocząć proces rejestrowania.
4. Aby zmienić typ używanych odwołań z bezwzględnych na względne, naciśnij przycisk *Użyj odwołań względnych*.

Po naciśnięciu przycisk zostanie wyróżniony innym kolorem.

5. W komórkach z zakresu B1:G1 wprowadź nazwy pierwszych sześciu miesięcy, podobnie jak w poprzednim przykładzie.
6. Zaznacz komórkę B1.
7. Zakończ rejestrowanie makra.

Po zarejestrowaniu makra w trybie odwołań względnych kod źródłowy będzie miał całkiem inną postać:

```
Sub Makro2()
    ActiveCell.FormulaR1C1 = "Styczeń"
    ActiveCell.Offset(0, 1).Range("A1").Select
    ActiveCell.FormulaR1C1 = "Luty"
    ActiveCell.Offset(0, 1).Range("A1").Select
```

```

ActiveCell.FormulaR1C1 = "Marzec"
ActiveCell.Offset(0, 1).Range("A1").Select
ActiveCell.FormulaR1C1 = "Kwiecień"
ActiveCell.Offset(0, 1).Range("A1").Select
ActiveCell.FormulaR1C1 = "Maj"
ActiveCell.Offset(0, 1).Range("A1").Select
ActiveCell.FormulaR1C1 = "Czerwiec"
ActiveCell.Offset(0, -5).Range("A1").Select
End Sub

```

Aby wykonać powyższe makro, uaktywnij arkusz, przejdź na kartę *Deweloper* i naciśnij przycisk *Makra* znajdujący się w grupie opcji *Kod*. Na ekranie pojawi się okno dialogowe *Makro*. Wybierz nazwę makra, które chcesz wykonać, i naciśnij przycisk *Uruchom*.

Zwrót uwagę, że w tym przykładzie procedura została nieznacznie zmieniona. Komórka początkowa została uaktywniona *przed* rozpoczęciem rejestracji makra. Ma to olbrzymie znaczenie, gdy rejestrujesz makra bazujące na aktywnej komórce.

Makro na pierwszy rzut oka sprawia wrażenie raczej skomplikowanego, ale tak naprawdę jest dość proste. Pierwsza instrukcja jedynie wprowadza ciąg znaków Styczeń do aktywnej komórki. Instrukcja korzysta z aktywnej komórki, ponieważ przed nią nie występuje instrukcja, która ją zaznacza. Następna instrukcja przy użyciu właściwości *Offset* przenosi zaznaczenie o jedną komórkę w prawo. Kolejna instrukcja wstawia kolejny ciąg znaków itd. Na końcu początkowa komórka jest ponownie zaznaczana. W tym celu zamiast stosowania odwołania bezwzględnego obliczane jest przesunięcie względne. W przeciwieństwie do poprzedniego przykładu to makro zawsze rozpoczyna wprowadzanie tekstu od aktywnej komórki.



Zwrót uwagę na fakt, że nasze makro generuje kod źródłowy, który odwołuje się do komórki A1 — co może wydawać się dziwne, ponieważ ta komórka nie była używana przy rejestracji makra. Takie zachowanie wynika po prostu ze sposobu działania rejestratora makr (właściwość *Offset* zostanie bardziej szczegółowo omówiona w dalszej części rozdziału). Na chwilę obecną w zupełności wystarczy że powiemy, iż nasze makro działa zgodnie z oczekiwaniemi.

Kluczowym elementem naszych dotychczasowych rozważań jest fakt, że rejestrator makr ma dwa odrębne tryby działania, i zawsze powinieneś dokładnie wiedzieć, w którym trybie rejestrujesz nowe makro — w przeciwnym razie osiągnięte rezultaty mogą być zgoła różne od oczekiwanych.

Nawiasem mówiąc, kod źródłowy automatycznie generowany przez rejestrator makr jest zbyt złożony i z pewnością użycie rejestratora nie jest najbardziej optymalną metodą tworzenia kodu programu. Kolejne makro, którego kod znajdziesz poniżej, zostało napisane ręcznie i realizuje dokładnie taką samą operację jak poprzednio. Ten przykład demonstruje, że w kodzie VBA wcale nie trzeba zaznaczać komórki przed umieszczeniem w niej danych — to bardzo ważne stwierdzenie, które pozwala na znaczące przyspieszenie działania kodu makra.

```

Sub Makro3()
    ActiveCell.Offset(0, 0) = "Styczeń"
    ActiveCell.Offset(0, 1) = "Luty"
    ActiveCell.Offset(0, 2) = "Marzec"
    ActiveCell.Offset(0, 3) = "Kwiecień"

```

```
ActiveCell.Offset(0, 4) = "Maj"  
ActiveCell.Offset(0, 5) = "Czerwiec"  
End Sub
```

Po zastosowaniu konstrukcji `With ... End With` makro może być jeszcze wydajniejsze:

```
Sub Makro4()  
    With ActiveCell  
        .Offset(0, 0) = "Styczeń"  
        .Offset(0, 1) = "Luty"  
        .Offset(0, 2) = "Marzec"  
        .Offset(0, 3) = "Kwiecień"  
        .Offset(0, 4) = "Maj"  
        .Offset(0, 5) = "Czerwiec"  
    End With  
End Sub
```

Jeżeli zaliczasz się do ekspertów z zakresu języka VBA, możesz zrobić wrażenie na kolegach i sprowadzić makro do jednej instrukcji:

```
Sub Makro5()  
    ActiveCell.Resize(.6) = Array("Styczeń", "Luty", "Marzec", "Kwiecień", "Maj",  
    ↴"Czerwiec")  
End Sub
```

Opcje związane z rejestraniem makr

Przed rozpoczęciem rejestrowania nowego makra na ekranie pojawia się okno dialogowe *Rejestrowanie makra* zawierające kilka opcji, które omówimy poniżej.

- **Nazwa makra** — W tym polu możesz wpisać nazwę rejestrowanego makra. Domyślnie Excel dla kolejnych rejestrowanych makr używa nazw Makro1, Makro2 itd. Zazwyczaj akceptuję domyślną nazwę makra, a później ją zmieniam. Jednak można podać własną nazwę makra przed rozpoczęciem jego rejestracji. Wybór należy do Ciebie.
- **Klawisz skrótu** — Opcja *Klawisz skrótu* umożliwia przypisanie do makra kombinacji klawiszy, których wciśnięcie spowoduje uruchomienie makra. Na przykład: jeżeli wpiszesz w tym polu małą literę *w*, makro zostanie wykonane po wciśnięciu kombinacji klawiszy *Ctrl+W*. Jeżeli wpiszesz dużą literę *W*, makro zostanie uruchomione po wciśnięciu kombinacji klawiszy *Ctrl+Shift+W*. Pamiętaj, że zdefiniowany w ten sposób skrót klawiszowy nadpisuje wbudowane skróty klawiszowe Excela (jeżeli taki skrót istnieje). Na przykład: jeżeli do swojego makra przypiszesz kombinację klawiszy *Ctrl+B*, to nie będziesz już mógł użyć tego skrótu klawiszowego do pogrubiania tekstu w komórce.

Skrót klawiszowy możesz dodać lub zmienić w dowolnej chwili, dlatego nie musisz tego robić w trakcie rejestracji makra.

- **Przechowuj makro w** — Opcja *Przechowuj makro w* informuje Excela, gdzie ma zostać zapisane rejestrowane makro. Domyślnie Excel umieszcza makra w module kodu aktywnego skoroszytu. Tworzone makra możesz również zapisać w nowym skoroszytce (Excel utworzy w tym celu nowy, pusty skoroszyt) lub

w skoroszycie makr osobistych (więcej szczegółowych informacji na temat tego skorosztytu znajdziesz w ramce „Skoroszyt makr osobistych”).



Excel zapamiętuje dokonany wybór, dzięki czemu kiedy następnym razem będziesz rejestrował nowe makro, Excel domyślnie użyje tej samej lokalizacji co poprzednio.

- **Opis** — Jeżeli chcesz, możesz w tym polu umieścić mniej lub bardziej szczegółowy opis rejestrowanego makra. Tekst wpisany w tym polu pojawi się w postaci komentarza na początku rejestrowanego makra.

Modyfikowanie zarejestrowanych makr

Jak pamiętasz, wynikiem zarejestrowania nawet jednego, prostego polecenia (polecenie *UKŁAD STRONY/Ustawienia strony/Pozioma*) może być całkiem spora ilość kodu języka VBA. Nasze doświadczenie było znakomitym przykładem tego, że w wielu przypadkach po zarejestrowaniu makra wiele zbędnych poleceń kodu źródłowego można po prostu usunąć.

Pamiętaj, że rejestrator makr nie zawsze generuje najwydajniejszy kod. Jeżeli dokładnie przyjrzyisz się wygenerowanemu kodowi, zobacysz, że Excel zazwyczaj rejestruje to, co jest zaznaczane (czyli obiekt), a następnie w kolejnych poleceniach posługuje się obiektem *Selection*. Poniżej przedstawiono przykładowy kod źródłowy, który zostanie wygenerowany przez rejestrator makr po zaznaczeniu zakresu komórek, a następnie użyciu kilku przycisków z karty *Narzędzia główne* do zmiany sposobu formatowania liczb oraz zastosowania pogrubienia i kursyw.

```
Range("A1:C5").Select
Selection.Style = "Comma"
Selection.Font.Bold = True
Selection.Font.Italic = True
```

Wygenerowany kod VBA oczywiście działa, ale to tylko jeden ze sposobów wykonania takich operacji. Zamiast tego możesz użyć bardziej efektywnej konstrukcji z poleceniami *With* i *End With*, tak jak to zostało przedstawione na poniższym przykładzie:

```
Range ("A1:C5").Select
With Selection
    .Style = "Comma"
    .Font.Bold = True
    .Font.Italic = True
End With
```

Skoroszyt makr osobistych

Szczególnie przydatne i ważne makra języka VBA możesz zapisać w skoroszycie makr osobistych (ang. *Personal Macro Workbook*). Jest to plik skorosztytu o nazwie *Personal.xlsb*, umieszczony w katalogu *XLStart*. Ten skoroszyt jest automatycznie ładowany za każdym razem, kiedy uruchamiasz program Excel i dzięki temu od razu masz dostęp do zapisanych w nim makr. Skoroszyt *Personal.xlsb* jest domyślnie ukryty, tak, aby nie przeszkadzać Ci w pracy.

Plik *Personal.xls* jest tworzony dopiero w momencie zapisania w nim pierwszego makra, a wcześniej po prostu nie istnieje.

Jeżeli pominiesz metodę `Select`, możesz stworzyć jeszcze bardziej wydajny kod:

```
With Range ("A1:C5")
    .Style = "Comma"
    .Font.Bold = True
    .Font.Italic = True
End With
```

Jeżeli szybkość działania ma krytyczne znaczenie dla Twojej aplikacji, zawsze powinieneś dokładnie przeanalizować zarejestrowany kod źródłowy języka VBA i upewnić się, że jest tak wydajny, jak to tylko możliwe.

Oczywiście zanim rozpocznesz modyfikowanie zarejestrowanych makr, powinieneś dogłębnie poznać tajniki programowania w języku VBA. Na chwilę obecną powinieneś po prostu zapamiętać, że kod VBA wygenerowany przez rejestrator makr nie zawsze jest najlepszym i najbardziej efektywnym rozwiązaniem problemu.

Kilka słów o przykładach kodu

W książce znajdziesz wiele fragmentów kodu źródłowego języka VBA, które mają na celu zilustrowanie danego zagadnienia czy problemu. Bardzo często takie przykłady kodu mogą składać się tylko z jednego polecenia. W niektórych przypadkach cytowany przykład składa się tylko z pojedynczego wyrażenia, które samo w sobie nie jest jeszcze poprawnym poleceniem języka VBA.

Na przykład kod przedstawiony poniżej jest wyrażeniem:

```
Range("A1").Value
```

Aby przetestować wyrażenie, musisz je obliczyć (wykonać). W tym przypadku dobrym rozwiązaniem będzie użycie funkcji `MsgBox`:

```
MsgBox Range("A1").Value
```

Jeżeli chcesz samodzielnie wypróbować podane przykłady, powinieneś umieścić polecenie w procedurze utworzonej w module kodu VBA, na przykład:

```
Sub Test()
    ' tutaj wstaw testowane polecenie
End Sub
```

Następnie umieść kursor w dowolnym miejscu kodu procedury i naciśnij klawisz `F5` aby ją wykonać. Dodatkowo powinieneś się upewnić, czy kod procedury jest wykonywany we właściwym kontekście. Na przykład: jeżeli testowane polecenie odnosi się do arkusza Arkusz1, powinieneś upewnić się, czy aktywny skoroszyt faktycznie zawiera arkusz o takiej nazwie.

Jeżeli kod źródłowy składa się tylko z jednej instrukcji, możesz użyć okna *Immediate* edytora Visual Basic. Okno to jest bardzo przydatne w przypadku natychmiastowego wykonywania instrukcji bez konieczności tworzenia procedury. Jeżeli okno *Immediate* nie jest widoczne, w edytorze Visual Basic powinieneś nacisnąć kombinację klawiszy `Ctrl+G`.

Po wprowadzeniu instrukcji w oknie *Immediate* wystarczy nacisnąć klawisz `Enter`. Aby obliczyć (wykonać) dane wyrażenie, powinieneś na jego początku wstawić znak zapytania (?), który pełni funkcję odpowiednika metody `Print`. Na przykład w oknie *Immediate* możesz wprowadzić następujące wyrażenie:

```
? Range("A1").Value
```

Wynik wyrażenia zostanie wyświetlony w następnym wierszu okna *Immediate*.

Obiekty i kolekcje

Jeżeli uważnie przeczytałeś pierwszą część rozdziału, masz już ogólny pogląd na to, czym jest VBA, i znasz podstawy stosowania modułów VBA w edytorze *Visual Basic*. Zapoznałeś się również z kilkoma przykładowymi kodami programów w języku VBA oraz takimi pojęciami, jak obiekty i właściwości. W tym podrozdziale znajdziesz dodatkowe informacje o obiektach oraz kolekcjach obiektów.

Kiedy pracujesz z językiem VBA, musisz znać koncepcję obiektów i poznać model obiektowy Excela. Bardzo pomocne będzie tutaj myślenie o obiektach w kategorii *hierarchii*. Na szczytce modelu znajduje się obiekt Application, którym w tym przypadku jest sam Excel. Jeżeli jednak tworzysz aplikacje w języku VBA oparte na programie Microsoft Word, obiektem Application będzie oczywiście edytor tekstu Word.

Hierarchia obiektów

Obiekt Application (czyli w naszym przypadku Excel) zawiera inne obiekty. Oto przykłady obiektów zawartych w obiekcie Application:

- Workbooks (kolekcja wszystkich obiektów Workbook — *skoroszyty*)
- Windows (kolekcja wszystkich obiektów Window — *okna*)
- AddIns (kolekcja wszystkich obiektów AddIn — *dodatki*)

Niektóre obiekty są kontenerami dla innych obiektów. Przykładowo zbiór Workbooks składa się z wszystkich otwartych obiektów Workbook. Z kolei obiekt Workbook zawiera inne obiekty. Oto niektóre z nich:

- Worksheets (kolekcja obiektów Worksheet — *arkusze*)
- Charts (kolekcja obiektów Chart — *wykresy*)
- Names (kolekcja obiektów Name — *nazwy*)

Z kolei każdy z powyższych obiektów przechowuje inne obiekty. Kolekcja Worksheets składa się z wszystkich obiektów Worksheet zawartych w obiekcie Workbook. Obiekt Worksheet zawiera wiele innych obiektów. Oto niektóre z nich:

- ChartObjects (kolekcja obiektów ChartObject)
- Range
- PageSetup
- PivotTables (kolekcja obiektów PivotTable — tabela przestawna)

Na razie wydaje się to zagmatwane, ale na pewno z czasem uświadomisz sobie, że hierarchia obiektów jest dość logiczna i dobrze zbudowana.

Kolekcje

Kolejnym kluczowym zagadnieniem związanym z programowaniem w języku VBA są kolekcje. *Kolekcja* jest grupą obiektów tej samej klasy, a sama w sobie też jest obiektem. Jak wspomniałem wcześniej, Workbooks (skoroszyty) jest kolekcją wszystkich aktualnie otwartych obiektów Workbook (skoroszyt). Z kolei Worksheets (arkusze) jest kolekcją wszystkich obiektów Worksheet (arkusz) znajdujących się w określonym obiekcie Workbook (skoroszyt). Istnieje możliwość przetwarzania całej kolekcji obiektów lub tylko jednego wybranego obiektu. Aby odwołać się do pojedynczego obiektu kolekcji, należy jego nazwę lub numer indeksu umieścić w nawiasach okrągłych za nazwą kolekcji. Oto przykład:

```
Worksheets("Arkusz1")
```

Jeżeli arkusz Arkusz1 jest pierwszym arkuszem kolekcji, możesz posłużyć się odwołaniem o następującej postaci:

```
Worksheets(1)
```

Aby odwołać się do drugiego arkusza obiektu Workbook, należy użyć odwołania `WorkSheets(2)` itd.

Istnieje też kolekcja o nazwie Sheets złożona ze wszystkich arkuszy skoroszytu, niezależnie od tego, czy są to arkusze zwykłe, czy arkusze wykresu. Jeżeli arkusz Arkusz1 jest pierwszym arkuszem skoroszytu, możesz odwołać się do niego w następujący sposób:

```
Sheets(1)
```

Odwoływanie się do obiektów

Aby odwołać się do obiektu przy użyciu polecień języka VBA, należy podawać nazwy kwalifikowane, łącząc za pomocą kropki (nazywanej też *operatorem kropki*) nazwy poszczególnych obiektów. Co się stanie w sytuacji, kiedy otworzysz dwa skoroszyty, a w każdym z nich znajduje się arkusz o nazwie Arkusz1? Rozwiążanie polega na utworzeniu odwołania do obiektu poprzez zawarcie w nim nazwy jego kontenera. Oto przykład:

```
Workbooks("Zeszyt1").Worksheets("Arkusz1")
```

Bez użycia identyfikatora skoroszytu VBA szukałby arkusza Arkusz1 w aktywnym skoroszycie.

Aby odwołać się do określonego zakresu (np. komórki A1) arkusza o nazwie Arkusz1 zawartego w skoroszycie Zeszyt1, należy zastosować następujące wyrażenie:

```
Workbooks("Zeszyt1").Worksheets("Arkusz1").Range("A1")
```

W pełnej, kwalifikowanej postaci powyższego odwołania zostanie też uwzględniony obiekt Application. Ma ono następującą postać:

```
Application.Workbooks("Zeszyt1").Worksheets("Arkusz1").Range("A1")
```

W większości przypadków możesz pominąć obiekt Application w odwołaniach, gdyż jest on przyjmowany domyślnie. Jeżeli obiekt Zeszyt1 jest aktywnym skoroszytem, w odwołaniu można go również pominąć i zastosować następujące wyrażenie:

```
Worksheets("Arkusz1").Range("A1")
```

Chyba już zorientowałeś się, do czego zmierzam. Jeżeli obiekt Arkusz1 jest aktywnym arkuszem, wyrażenie można jeszcze bardziej uprościć. Oto ono:

```
Range("A1")
```



W przeciwnieństwie do tego, czego mógłbyś oczekwać, Excel nie posiada obiektu o nazwie Cell odwołującego się do pojedynczej komórki. Pojedyncza komórka jest po prostu obiektem klasy Range składającym się tylko z jednego elementu.

Samo odwoływanie się do obiektów (jak w powyższych przykładach) niczego nie wykonuje. Aby wykonać jakąkolwiek znaczącą operację, musisz odczytać lub zmodyfikować właściwości obiektu bądź użyć metody, która przetworzy obiekt.

Właściwości i metody

Bardzo łatwo możesz poczuć się przytłoczony mnogością właściwości i metod obiektów, ponieważ istnieją ich dosłownie tysiące. W tym podrozdziale wyjaśnimy, w jaki sposób z nich korzystać.

Właściwości obiektów

Każdy obiekt ma swoje właściwości. Przykładowo obiekt Range posiada właściwość o nazwie Value. Możesz utworzyć kod źródłowy języka VBA odczytujący lub ustawiający wartość właściwości Value tego obiektu. Poniżej pokazano procedurę używającą funkcji języka VBA, o nazwie MsgBox. Wyświetla ona okno zawierające wartość komórki A1 arkusza Arkusz1 aktywnego skoroszytu:

```
Sub ShowValue()
    MsgBox Worksheets("Arkusz1").Range("A1").Value
End Sub
```



MsgBox jest bardzo użyteczną funkcją, która pozwala na wyświetlanie wyników w trakcie działania kodu programu VBA. W przykładach z tej książki bardzo często będziemy z niej korzystać.

Kod źródłowy zawarty w poprzednim przykładzie wyświetla aktualną wartość właściwości Value powiązanej z określoną komórką, którą jest komórka A1 arkusza Arkusz1 aktywnego skoroszytu. Jeżeli aktywny skoroszyt nie zawiera arkusza o nazwie Arkusz1, makro wygeneruje błąd.

A co należy zrobić, aby zmienić wartość właściwości Value? Poprzez modyfikację właściwości Value komórki A1 poniższa procedura zmienia wyświetlana w niej wartość:

```
Sub ChangeValue()
    Worksheets("Arkusz1").Range("A1").Value = 123.45
End Sub
```

Po wykonaniu tej procedury w komórce A1 arkusza Arkusz1 pojawi się wartość 123.45.

Możesz wprowadzić kod źródłowy tej procedury do modułu i poeksperymentować z nim.

Pamiętaj, że właściwość Value możesz odczytywać tylko dla obiektów klasy Range składających się z jednej komórki. Z drugiej strony kod Twojego programu może *ustawiać* wartości właściwości Value obiektów Range składających się z wielu komórek. W przykładzie przedstawionym poniżej pierwsze polecenie jest poprawne, ale drugie polecenie już nie:

```
Range("A1:C12").Value = 99  
MsgBox Range("A1:C12").Value
```



Większość obiektów posiada domyślną właściwość. W przypadku obiektu Range taką właściwością jest Value, stąd jeżeli w kodzie programu pominięty zostanie element .Value, uzyskany efekt będzie dokładnie taki sam. Jednak za dobry nawyk uważa się umieszczanie w kodzie programu pełnych nazw właściwości, nawet jeżeli są to właściwości domyślne.

Polecenie przedstawione poniżej korzysta z właściwości HasFormula oraz Formula obiektu Range:

```
If Range("A1").HasFormula Then MsgBox Range("A1").Formula
```

W naszym przykładzie konstrukcja If-Then została użyta do warunkowego wyświetlania okna dialogowego. Jeżeli w komórce znajduje się jakaś formuła, to zostanie ona wyświetlona w oknie dialogowym za pomocą właściwości Formula obiektu Range. Jeżeli jednak w komórce nie ma formuły, nic się nie dzieje (nic nie jest wyświetlane).

Właściwość Formula obiektu Range jest właściwością do odczytu i zapisu, stąd możesz umieścić daną formułę w wybranej komórce, korzystając z następującego polecenia VBA:

```
Range("D12").Formula = "=RAND()*100"
```

Metody obiektowe

Oprócz wspomnianych wcześniej właściwości obiekty posiadają również swoje metody. **Metoda** to inaczej mówiąc operacja wykonywana na obiekcie. Poniżej pokazano prosty przykład, w którym zastosowano metodę Clear obiektu Range. Po wykonaniu tej procedury zawartość zakresu komórek A1:C3 arkusza Arkusz1 będzie pusta i usunięte z niego zostanie całe formatowanie.

```
Sub ZapRange()  
    Worksheets("Arkusz1").Range("A1:C3").Clear  
End Sub
```

Aby usunąć wartości z zakresu i pozostawić formatowanie komórek, należy zastosować metodę ClearContents obiektu Range.

Określanie argumentów metod i właściwości

Jednym z zagadnień, które często wywołują zamieszanie wśród niedoświadczonych programistów używających języka VBA, są argumenty metod i właściwości. Niektóre metody korzystają z argumentów w celu dokładniejszego zdefiniowania operacji. Z kolei niektóre właściwości korzystają z argumentów w celu uzyskania możliwości dodatkowego określenia wartości. W niektórych przypadkach argumenty są opcjonalne.

Jeżeli metoda używa argumentów, są one wstawiane za jej nazwą i oddzielane przecinkami. Jeżeli metoda posługuje się opcjonalnymi argumentami, zamiast nich można pozostawić puste miejsca. W dalszej części ramki wyjaśnimy, jak to zrobić.

Weźmy dla przykładu metodę **Protect** obiektu **Workbook**. Jeżeli sprawdzisz w systemie pomocy opis tej metody, to zobaczysz, że metoda **Protect** pobiera trzy argumenty, reprezentujące odpowiednio hasło, status ochrony struktury skoroszytu i status ochrony okna. Argumenty odpowiadają opcjom zawartym w oknie dialogowym *Chronienie struktury i systemu Windows*.

Jeżeli na przykład zależy Ci na ochronie skoroszytu o nazwie *MójSkoroszyt.xlsx*, możesz użyć następującego polecenia:

```
Workbooks("MójSkoroszyt.xlsx").Protect "xyzzy", True, False
```

W tym przypadku skoroszyt jest chroniony hasłem (argument 1). Dodatkowo jest chroniona jego struktura (argument 2), ale okna skoroszytu już nie (argument 3).

Jeżeli nie chcesz przypisywać hasła, możesz użyć następującej instrukcji:

```
Workbooks("MójSkoroszyt.xlsx").Protect , True, False
```

Pierwszy argument został pominięty, a zamiast niego pozostawiłem puste miejsce identyfikowane przy użyciu przecinka.

Kolejna metoda zwiększąca czytelność kodu źródłowego polega na zastosowaniu argumentów, którym nadano nazwy. Oto przykład demonstrujący, w jaki sposób w poprzedniej instrukcji zastosować nazwane argumenty:

```
Workbooks("MójSkoroszyt.xlsx").Protect Structure:=True, Windows:=False
```

Zastosowanie nazwanych argumentów jest dobrym pomysłem, zwłaszcza w przypadku metod mających wiele argumentów opcjonalnych, a także w sytuacji, gdy konieczne jest użycie jedynie kilku z nich. Przy korzystaniu z nazwanych argumentów nie ma potrzeby pozostawiania pustych miejsc identyfikujących pominięte argumenty.

W przypadku właściwości i metod zwracających wartość konieczne jest umieszczenie argumentów w nawiasach okrągłych. Przykładowo właściwość **Address** obiektu **Range** pobiera pięć argumentów, z których wszystkie są opcjonalne. Ze względu na to, że właściwość **Address** zwraca wartość, poniższa instrukcja nie jest poprawna, ponieważ pominięto nawiasy okrągłe:

```
MsgBox Range("A1").Address False ' nieprawidłowa instrukcja
```

Aby składnia takiej instrukcji była poprawna, konieczne jest zastosowanie nawiasów okrągłych:

```
MsgBox Range("A1").Address(False)
```

Instrukcja może też zostać zapisana przy użyciu nazwanego argumentu w następujący sposób:

```
MsgBox Range("A1").Address(rowAbsolute:=False)
```

Tego typu niuanse stają się bardziej zrozumiałe w miarę zdobywania doświadczenia z zakresu języka VBA.

Większość metod pobiera też argumenty umożliwiające dokładniejsze zdefiniowanie operacji. Poniżej zamieszczono przykład procedury kopiącej przy użyciu metody `Copy` obiektu `Range` zawartość komórki A1 do komórki B1. W tym przypadku metoda `Copy` posiada jeden argument (miejscie przeznaczenia skopiowanych danych).

```
Sub CopyOne()
    Range("A1").Copy Range("B1")
End Sub
```

Tajemnice obiektu Comment

Aby lepiej zrozumieć właściwości i metody obiektu, zwrócimy uwagę na szczególny obiekt o nazwie `Comment`. Obiekt ten jest tworzony w momencie wprowadzania w komórce komentarza. Aby wstawić komentarz, przejdź na kartę *RECENZJA* i naciśnij przycisk *Nowy komentarz* znajdujący się w grupie opcji *Komentarze*. W kolejnych podrozdziałach będziesz miał okazję bliżej zapoznać się ze specyfiką pracy z obiekttami. Na nasze potrzeby wybrałem obiekt `Comment`, ponieważ nie ma on zbyt wielu właściwości ani metod.

Celem tego podrozdziału jest nie tyle szczegółowe omówienie obiektu `Comment`, ile po prostu przedstawienie sposobu pracy z obiekttami w ogóle. Praktycznie każde zagadnienie omawiane w tym podrozdziale będziesz mógł wykorzystać podczas pracy z innymi obiekttami.

Pomoc dla obiektu Comment

Jedna z metod uzyskiwania informacji na temat obiektów polega na skorzystaniu z systemu pomocy. Na rysunku 5.14 pokazano okno pomocy systemowej dotyczące obiektu `Comment`. Zwróć uwagę na fakt, że w spisie treści pomocy znajdziesz również opis wszystkich właściwości i metod tego obiektu.



W Excelu 2013 tematy pomocy są wyświetlane w oknie Twojej domyślnej przeglądarki sieciowej, stąd aby skorzystać z takiej pomocy, musisz być połączony do sieci Internet.

Właściwości obiektu Comment

Obiekt `Comment` ma sześć właściwości. W tabeli 5.1 zawarto ich listę wraz z krótkim opisem każdej z nich. Jeżeli właściwość jest *tylko do odczytu*, z poziomu kodu języka VBA możesz odczytywać jej wartość, ale nie możesz jej modyfikować.

Metody obiektu Comment

W tabeli 5.2 zawarto metody, których można używać w przypadku obiektu `Comment`. Jak już wspominaliśmy, są to metody realizujące operacje, które zapewne już niejednokrotnie wykonywałeś ręcznie na obiekcie `Comment`... ale prawdopodobnie nigdy nie myślałeś o nich jak o metodach.

The screenshot shows the Microsoft Office Dev Center page for the Comment Object (Excel). The page has a navigation bar at the top with links for Home, Products, Office 365, Docs, Samples, Downloads, Training, How To, Support, and Forums. The breadcrumb trail indicates the path: Office and SharePoint development > Office client development > Office 2013 > Excel 2013 > Welcome to the Excel 2013 developer reference > Object model reference (Excel 2013 developer reference) > Comment Object (Excel). The main content area is titled "Comment Object (Excel)". It includes a "Remarks" section stating that the Comment object is a member of the Comments collection, and an "Example" section showing VBA code to change comment text and hide comment two. There are also "VBA" code snippets for changing comment text and hiding comment two.

Rysunek 5.14. Głównie okno pomocy systemowej dla obiektu Comment

Korzystanie z systemu pomocy

Najprostszą metodą uzyskania pomocy na temat określonego obiektu, właściwości lub metody jest wprowadzenie nazwy szukanego elementu w oknie *Code* i naciśnięcie klawisza *F1*. Jeżeli wystąpi jednakże niejednoznaczność dotycząca wpisanego słowa, na ekranie pojawi się okno dialogowe podobne do przedstawionego na rysunku poniżej.

Niestety znaczenie pozycji pokazanych w oknie dialogowym nie zawsze jest oczywiste, dlatego zlokalizowanie właściwego tematu pomocy może wymagać wykonania kilku iteracji. Okno dialogowe przedstawione poniżej pojawia się po wprowadzeniu słowa *Comment* i wcisnięciu przycisku *F1*. W tym przypadku chodziło nam oczywiście o obiekt o nazwie *Comment*, ale jak się możemy łatwo przekonać, jest to również nazwa właściwości obiektu *Scenario*. Kliknięcie pierwszej pozycji wyświetla tematy pomocy dla obiektu *Comment*. Z kolei kliknięcie drugiej pozycji wyświetla tematy pomocy dotyczące właściwości *Comment* obiektu *Scenario*.

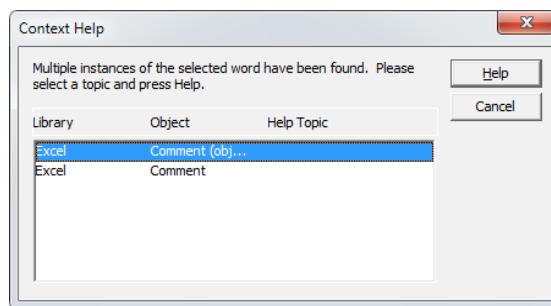


Tabela 5.1. Właściwości obiektu Comment

Właściwość	Tylko do odczytu	Opis
Application	Tak	Zwraca obiekt reprezentujący aplikację, która utworzyła komentarz (w naszym przypadku jest to oczywiście Excel).
Author	Tak	Zwraca nazwę osoby, która utworzyła komentarz.
Creator	Tak	Zwraca liczbę całkowitą wskazującą aplikację, w której został utworzony obiekt.
Parent	Tak	Zwraca nadzędny obiekt komentarza (obiektem nadzędnym w tym przypadku jest nim zawsze obiekt Range).
Shape	Tak	Zwraca obiekt Shape reprezentujący kształt powiązany z komentarzem.
Visible	Nie	Jeżeli komentarz jest widoczny, ta właściwość ma wartość True.

Tabela 5.2. Metody obiektu Comment

Metoda	Opis
Delete	Usuwa komentarz.
Next	Zwraca obiekt Comment reprezentujący następny komentarz w arkuszu.
Previous	Zwraca obiekt Comment reprezentujący poprzedni komentarz w arkuszu.
Text	Zwraca lub wstawia tekst komentarza (pobiera trzy argumenty).



Możesz być zaskoczony stwierdzeniem, że Text jest właściwością a nie metodą, ale rozróżnienie pomiędzy właściwościami i metodami nie zawsze jest oczywiste, a oprócz tego model obiektowy Excela nie jest idealnie spójny. W praktyce rozróżnianie właściwości od metod nie jest tak bardzo istotne — dopóki używasz poprawnej składni, nie ma większego znaczenia, czy słowo zawarte w kodzie źródłowym identyfikuje właściwość, czy metodę.

Kolekcja Comments

Jak pamiętasz, kolekcja jest grupą podobnych do siebie obiektów. Każdy arkusz posiada kolekcję Comments złożoną ze wszystkich jego obiektów Comment. Jeżeli arkusz nie zawiera komentarzy, kolekcja Comments jest pusta. Komentarze pojawiają się w kolekcji w porządku opartym na pozycji w arkuszu — kolejno od lewej do prawej i z góry na dół.

Na przykład poniższe polecenie odwołuje się do pierwszego komentarza w arkuszu Arkusz1 znajdującego się w aktywnym skoroszycie:

```
Worksheets("Arkusz1").Comments(1)
```

Kolejne polecenie wyświetla tekst pierwszego komentarza w arkuszu Arkusz1:

```
MsgBox Worksheets("Arkusz1").Comments(1).Text
```

W przeciwieństwie do większości obiektów, obiekt Comment nie posiada właściwości Name. Z tego powodu, aby odwołać się do określonego komentarza, musisz użyć odpowiedniego numeru indeksu lub (częściej) użyć właściwości Comment obiektu Range, która zwróci wybrany komentarz.

Kolekcja `Comments` jest również obiektem i posiada własny zestaw właściwości oraz metod. Na przykład kolekcja `Comments` posiada właściwość `Count`, która przechowuje liczbę elementów kolekcji — w tym przypadku jest to liczba obiektów `Comment` utworzonych w aktywnym arkuszu. Przykładowo poniższe polecenie wyświetla całkowitą liczbę komentarzy znajdujących się w aktywnym arkuszu:

```
MsgBox ActiveSheet.Comments.Count
```

Następny przykład wyświetla adres komórki zawierającej pierwszy komentarz:

```
MsgBox ActiveSheet.Comments(1).Parent.Address
```

W przypadku tej instrukcji `Comments(1)` zwraca pierwszy obiekt `Comment` kolekcji `Comments`. Właściwość `Parent` obiektu `Comment` zwraca jego kontener, którym jest obiekt `Range`. W oknie komunikatu jest wyświetlana wartość właściwości `Address` obiektu `Range`, co w efekcie powoduje wyświetlenie adresu komórki zawierającej pierwszy komentarz.

Przy użyciu konstrukcji `For Each ... Next` możesz w pętli przetwarzać wszystkie komentarze arkusza (więcej szczegółowych informacji na temat petli znajdziesz w rozdziale 6.). Oto przykład kodu źródłowego wyświetlającego oddzielne okno komunikatu dla każdego komentarza z aktywnego arkusza:

```
For Each cmt In ActiveSheet.Comments  
    MsgBox cmt.Text  
Next cmt
```

Jeżeli nie chcesz borykać się z serią okien dialogowych, możesz wyświetlić wszystkie komentarze w oknie *Immediate* edytora Visual Basic przy użyciu następującej procedury:

```
For Each cmt In ActiveSheet.Comments  
    Debug.Print cmt.Text  
Next cmt
```

Właściwość `Comment`

Do tej pory omawialiśmy jedynie obiekt `Comment`. Jeżeli jednak poszukasz trochę w pomocy systemowej, dowiesz się, że obiekt `Range` posiada właściwość o nazwie `Comment`. Jeżeli komórka zawiera komentarz, właściwość `Comment` zwraca obiekt `Comment`. Na przykład poniższa instrukcja odwołuje się do obiektu `Comment` powiązanego z komórką A1:

```
Range("A1").Comment
```

Jeżeli byłby to pierwszy komentarz w arkuszu, do tego samego obiektu `Comment` mógłbyś się odwołać się w następujący sposób:

```
ActiveSheet.Comments(1)
```

Aby w oknie komunikatu wyświetlić komentarz umieszczony w komórce A1, powinieneś posłużyć się następującą instrukcją:

```
MsgBox Range("A1").Comment.Text
```

Jeżeli komórka A1 nie zawiera komentarza, próba wykonania powyższego polecenia wygeneruje błąd.



Fakt, że właściwość może zwracać obiekt, jest niezmiernie ważny — być może na początku trudno to zrozumieć, ale jest to niezbędne do opanowania tajników programowania w języku VBA.

Obiekty zawarte w obiekcie Comment

Ponieważ niektóre z właściwości mogą zwracać obiekty, praca z nimi może na początku sprawiać pewne trudności. Założymy, że chciałbyś określić kolor tła wybranego komentarza w arkuszu Arkusz1. Jeżeli przejrzyesz listę właściwości obiektu Comment, nie znajdziesz żadnej właściwości związanej z kolorami. Aby zatem osiągnąć swój cel, powinieneś wykonać następujące operacje:

1. Użyj właściwości Shape obiektu Comment, która zwróci obiekt Shape zawarty w komentarzu.
2. Użyj właściwości Fill obiektu Shape, która zwróci obiekt FillFormat.
3. Użyj właściwości ForeColor obiektu FillFormat, która zwróci obiekt ColorFormat.
4. W sprawdzeniu koloru użyj właściwości RGB obiektu ColorFormat.

Kolejna metoda sprawdzenia koloru tła obiektu Comment polega na użyciu innych obiektów zawartych w obiekcie Comment. Oto hierarchia obiektów związanych z tą operacją:

```
Application (Excel)
  obiekt Workbook
    obiekt Worksheet
      obiekt Comment
        obiekt Shape
          obiekt FillFormat
            obiekt ColorFormat
```

Muszę jednak przyznać, że może się to wydawać bardzo zagmatwane! Aby zademonstrować pewną elegancję języka VBA, poniżej zamieściłem kod źródłowy jednego, co prawda nieco złożonego, polecenia, które zmienia kolor tła komentarza:

```
Worksheets("Arkusz1").Comments(1).Shape.Fill.ForeColor _  
    .RGB = RGB(0, 255, 0)
```

Początkowo tego typu odwoływanie zapewne nie będą dla Ciebie zbyt intuicyjne, ale ostatecznie z pewnością docenisz ich zalety. Na szczęście rejestrowanie operacji wykonywanych w Excelu prawie zawsze pokazuje sporą liczbę szczegółów dotyczących hierarchii zastosowanych obiektów.

Nawiasem mówiąc, aby zmienić kolor tekstu komentarza, konieczne jest użycie obiektu TextFrame będącego obiektem potomnym obiektu Comment. Obiekt TextFrame zawiera obiekt Characters, będący obiektem nadzawanym obiektu Font. Mając dostęp do obiektu Font

możesz użyć jego właściwości `Color` lub `ColorIndex`. Oto przykład instrukcji ustawiającej właściwość `ColorIndex` na wartość 5:

```
Worksheets("Arkusz1").Comments(1)  
    .Shape.TextFrame.Characters.Font.ColorIndex = 5
```



Więcej szczegółowych informacji na temat kolorów znajdziesz w rozdziale 28.

Sprawdzanie, czy komórka posiada komentarz

Instrukcja przedstawiona poniżej wyświetla zawartość komentarza znajdującego się w komórce A1 aktywnego arkusza:

```
MsgBox Range("A1").Comment.Text
```

Jeżeli w komórce A1 nie ma komentarza, wykonanie instrukcji spowoduje wygenerowanie nieco tajemniczego komunikatu o wystąpieniu błędu: *Object variable or With block variable not set* (Nie zdefiniowano zmiennej obiektowej lub zmiennej powiązanej z instrukcją `With`).

Aby stwierdzić, czy określona komórka zawiera komentarz, można napisać polecenie sprawdzające, czy obiekt `Comment` jest typu `Nothing` (zgadza się, `Nothing` jest poprawnym słowem kluczowym). Poniższa instrukcja wyświetli wartość `True`, jeżeli w komórce A1 nie zostanie znaleziony komentarz:

```
MsgBox Range("A1").Comment Is Nothing
```

Zwróć uwagę, że zamiast znaku równości użyłem słowa kluczowego `Is`.

Idąc krok dalej, możesz utworzyć polecenie wyświetlające komentarz zawarty w komórce tylko wtedy, gdy faktycznie on się w niej znajduje (w przypadku braku komentarza nie zostanie wygenerowany błąd). Takie polecenie ma następującą postać:

```
If Not Range("A1").Comment Is Nothing Then  
    MsgBox Range("A1").Comment.Text
```

Zwróć uwagę, że użyłem słowa kluczowego `Not`, które neguje wartość `True` zwracaną w sytuacji, kiedy w komórce nie ma komentarza. Nasze polecenie do sprawdzania warunku używa podwójnej negacji: jeżeli nieprawdą jest, że komentarz nie istnieje, to go wyświetlimy. Jeżeli wydaje się to nieco zagmatwane, spróbuj się nad tym przez chwilę spokojnie zastanowić i wszystko stanie się jasne.

Dodawanie nowego obiektu `Comment`

Być może zauważyleś, że lista metod obiektu `Comment` nie zawiera metody umożliwiającej dodanie nowego komentarza. Jest to spowodowane tym, że metoda `AddComment` należy po prostu do obiektu `Range`. Poniższa instrukcja umieszcza komentarz (pusty!) w komórce A1 aktywnego arkusza:

```
Range("A1").AddComment
```

Jeżeli poszukasz nieco w pomocy systemowej, odkryjesz, że metoda AddComment pobiera argument reprezentujący zawartość komentarza. A zatem można za pomocą jednego polecenia utworzyć komentarz i dodać tekst stanowiący jego zawartość:

```
Range("A1").AddComment "Formuła utworzona przez JW."
```



Jeżeli w danej komórce znajduje się już jakiś komentarz, użycie metody AddComment spowoduje wygenerowanie błędu. Aby tego uniknąć, przed dodaniem komentarza powinieneś umieścić kod sprawdzający, czy w danej komórce jest już komentarz.



Aby zobaczyć, jakie efekty można uzyskać po zastosowaniu właściwości i metod obiektu Comment, powinieneś zapoznać się z przykładami w skoroszycie zamieszczonym na stronie internetowej tej książki (<http://www.helion.pl/ksiazki/e13pvw.htm>). Skoroszyt *Obiekt Comment.xlsm* zawiera przykłady wykorzystujące obiekt Comment przy użyciu instrukcji języka VBA. Prawdopodobnie nie cały kod źródłowy będzie dla Ciebie od razu zrozumiały, ale dzięki niemu zorientujesz się, w jaki sposób pracujemy z obiektami w języku VBA.

Kilka przydatnych właściwości obiektu Application

Podczas pracy z Exceliem w danej chwili aktywny może być tylko i wyłącznie jeden skoroszyt. W takim skoroszycie aktywny może być tylko jeden arkusz, a w danym arkuszu aktywna może być tylko jedna komórka naraz (nawet jeżeli zaznaczony jest zakres złożony z wielu komórek). Język VBA „jest tego świadom” i umożliwia odwoływanie się do aktywnych obiektów w uproszczony sposób.

Takie rozwiązanie często się przydaje, ponieważ nie zawsze będziesz dokładnie wiedział, który konkretnie skoroszyt, arkusz lub zakres zostanie użyty. Język VBA znamienne ułatwia takie odwołania dzięki wykorzystaniu właściwości obiektu Application. Przykładowo obiekt Application posiada właściwość ActiveCell zwracającą odwołanie do aktywnej komórki. Poniższa instrukcja przypisuje aktywnej komórce wartość 1:

```
ActiveCell.Value = 1
```

Zwróci uwagę na fakt, że w tym przykładzie pominięte zostało zarówno odwołanie do obiektu Application, jak i do aktywnego arkusza, ponieważ oba obiekty zostały zastosowane domyślnie. Powinieneś jednak zdawać sobie sprawę z tego, że takie polecenie nie zadziała, gdy aktywna karta skoroszytu nie będzie zwykłym arkuszem. Jeżeli na przykład interpreter języka VBA wykona polecenie, gdy aktywny będzie arkusz wykresu, działanie procedury zostanie wstrzymane i zostanie wyświetlony komunikat o wystąpieniu błędu.

Jeżeli w arkuszu zaznaczysz zakres komórek, komórką aktywną będzie zawsze komórka z tego zakresu. Innymi słowy, komórka aktywna to zawsze pojedyncza komórka (nigdy zakres komórek).

Obiekt Application posiada też właściwość Selection zwracającą odwołanie do dowolnego zaznaczonego elementu, który jest pojedynczą komórką (aktywną), zakresem komórek lub takim obiektem, jak ChartObject, TextBox lub Shape.

W tabeli 5.3 przedstawiono listę wybranych właściwości obiektu Application, które są bardzo przydatne podczas pracy z komórkami i zakresami.

Tabela 5.3. Niektóre przydatne właściwości obiektu Application

Właściwość	Zwracany obiekt
ActiveCell	Aktywna komórka
ActiveChart	Aktywny arkusz wykresu lub wykres powiązany z obiektem ChartObject znajdującym się w arkuszu. Jeżeli wykres nie jest aktywny, właściwość będzie miała wartość Nothing
ActiveSheet	Aktywny arkusz (zwykły lub wykresu)
ActiveWindow	Aktywne okno
ActiveWorkbook	Aktywny skoroszyt
Selection	Zaznaczony obiekt (mogą to być takie obiekty jak Range, Shape, ChartObject itp.).
ThisWorkbook	Skoroszyt zawierający wykonywaną procedurę VBA. Ten obiekt może (ale nie musi) być taki sam jak obiekt ActiveWorkbook.

Zaletą stosowania wymienionych właściwości zwracających obiekty jest to, że nie musisz wiedzieć, która komórka, arkusz lub skoroszyt są aktualnie aktywne, ani też nie musisz tworzyć do nich odwołania. Dzięki temu możesz tworzyć kod źródłowy języka VBA, który nie jest ściśle powiązany z określonym skoroszytem, arkuszem lub zakresem. Na przykład poniższa instrukcja czyści zawartość aktywnej komórki, pomimo że nie jest znany jej adres:

```
ActiveCell.ClearContents
```

Kolejna instrukcja wyświetla komunikat zawierający nazwę aktywnego arkusza:

```
MsgBox ActiveSheet.Name
```

Aby poznać nazwę i ścieżkę aktywnego skoroszytu, należy użyć następującej instrukcji:

```
MsgBox ActiveWorkbook.FullName
```

Gdy zaznaczyłeś w arkuszu zakres komórek, za pomocą jednego polecenia możesz go w całości wypełnić wybraną wartością. W poniższym przykładzie właściwość Selection obiektu Application zwraca obiekt Range odpowiadający zaznaczonym komórkom. Polecenie po prostu modyfikuje właściwość Value obiektu Range, a wynikiem takiej operacji jest zakres komórek wypełnionych daną wartością:

```
Selection.Value = 12
```

Jeżeli zostanie zaznaczone coś innego niż zakres (np. obiekt ChartObject lub Shape), próba wykonania takiego polecenia spowoduje wygenerowanie błędu, ponieważ obiekty ChartObject i Shape nie posiadają właściwości Value.

Zwróć jednak uwagę na fakt, że kolejne polecenie wpisuje wartość 12 do komórek reprezentowanych przez obiekt Range, który został zaznaczony jeszcze przed zaznaczeniem innego obiektu, niebędącego obiektem typu Range. Jeżeli przyjrzyisz się opisowi właściwości RangeSelection w pomocy systemowej, przekonasz się, że właściwość ta ma zastosowanie wyłącznie do obiektu Window.

```
ActiveWindow.RangeSelection.Value = 12
```

Aby dowiedzieć się, ile komórek zostało zaznaczonych w aktywnym oknie, należy użyć właściwości Count. Oto przykład:

```
MsgBox ActiveWindow.RangeSelection.Count
```

Tajemnice obiektów Range

Wiele zadań realizowanych przy użyciu instrukcji języka VBA jest powiązanych z komórkami i zakresami arkuszy — w końcu w tym właśnie celu stworzono arkusze kalkulatory. Co prawda we wcześniejszym porównaniu trybów rejestrowania makr (patrz podrozdział „Odwołania względne czy bezwzględne?”) poznaleś już niektóre zagadnienia związane z przetwarzaniem komórek przy użyciu języka VBA, ale musisz zdobyć znacznie większą wiedzę na ten temat.

Obiekt Range zawarty w obiekcie Worksheet składa się z pojedynczej komórki arkusza lub ich zakresu. W kolejnych podpunktach omówimy trzy metody odwoływania się do obiektu Range przy użyciu instrukcji języka VBA. Oto one:

- właściwość Range obiektu klasy Worksheet lub Range,
- właściwość Cells obiektu Worksheet,
- właściwość Offset obiektu Range.

Właściwość Range

Właściwość Range zwraca obiekt Range. Jeżeli poszukasz w pomocy systemowej, to dowieś się, że ta właściwość posiada dwie wersje składni:

```
obiekt.Range(komórka1)  
obiekt.Range(komórka1, komórka2)
```

Właściwość Range jest powiązana z dwoma typami obiektów — Worksheet lub Range. Słowa komórka1 i komórka2 są odpowiednikami terminów, które Excel traktuje jako identyfikator zakresu (w pierwszym przypadku) lub jego granice (drugi przypadek). Poniżej zamieszczono kilka przykładów użycia właściwości Range.

Do tej pory poznaleś przykłady podobne do zamieszczonego poniżej. Polecenie takie po prostu wpisuje wybraną wartość do określonej komórki arkusza. W tym konkretnym przypadku w komórce A1 arkusza Arkusz1 jest umieszczana wartość 12.3.

```
Worksheets("Arkusz1").Range("A1").Value = 12.3
```

Właściwość Range rozpoznaje też nazwy zdefiniowane w skoroszytach. A zatem jeżeli komórce nadano nazwę Input, można wpisać do niej wartość przy użyciu następującego polecenia:

```
Worksheets("Arkusz1").Range("Input").Value = 100
```

W kolejnym przykładzie do zakresu złożonego z 20 komórek aktywnego arkusza jest wstawiana taka sama wartość (jeżeli aktywna karta Excela nie jest zwykłym arkuszem, zostanie wygenerowany komunikat błędu):

```
ActiveSheet.Range("A1:B10").Value = 2
```

Kolejne polecenie daje takie same wyniki, jak poprzednio:

```
Range("A1", "B10") = 2
```

Co prawda w powyższym poleceniu pominięto odwołanie do aktywnego arkusza, ale jest ono używane domyślnie. Poza tym pominięto właściwość `Value` obiektu `Range`, ponieważ jest to właściwość domyślna. W powyższym przykładzie użyto drugiej składni właściwości `Range`. W jej przypadku pierwszym argumentem jest górną lewą komórką, natomiast drugim — dolna prawa komórka zakresu.

W kolejnym przykładzie zastosowano operator przecięcia zakresów Excela (znak spacji), który zwraca przecięcie dwóch zakresów. W tym przypadku w miejscu przecięcia zakresów znajduje się jedna komórka, C6, stąd następująca instrukcja wpisuje do tej komórki wartość 3:

```
Range("C1:C10 A6:E6") = 3
```

Na koniec, kolejne polecenie wpisuje wartość 4 do pięciu komórek tworzących nieciągły zakres. Przecinek spełnia funkcję operatora złączenia (zwróć uwagę, że ciąg odwołań do poszczególnych komórek, oddzielonych od siebie przecinkami, został ujęty w znaki cudzysłowy):

```
Range("A1, A3, A5, A7, A9") = 4
```

Dotychczas we wszystkich przykładach używano właściwości `Range` obiektu `Worksheet`. Jak już wspomniałem, możesz też używać właściwości `Range` obiektu `Range`. Na początku może Ci się to wydawać nieco zagmatwane, ale powinieneś się już do tego przyzwyczaić.

Poniżej zamieszczono przykład takiego użycia właściwości `Range` obiektu `Range`. W tym przypadku obiektem `Range` jest aktywna komórka. Poniższe polecenie traktuje obiekt `Range` tak, jakby komórka znajdowała się w górnym lewym narożniku arkusza, i wstawia wartość 5 do komórki, którą *będzie* B2. Innymi słowy, zwrócone odwołanie jest względne w stosunku do górnego lewego narożnika obiektu `Range`. A zatem instrukcja wpisze wartość 5 bezpośrednio do komórki znajdującej się na prawo od aktywnej komórki i jeden wiersz poniżej:

```
ActiveCell.Range("B2") = 5
```

Mówilem już, że trochę to zagmatwane... Na szczęście istnieje o wiele bardziej zrozumiała metoda uzyskania dostępu do komórki względnej w stosunku do zakresu. Opiera się na właściwości `Offset`, którą omówimy w jednym z kolejnych podrozdziałów.

Właściwość Cells

Kolejna metoda odwoływanego się do zakresu polega na użyciu właściwości `Cells`. Podobnie jak właściwość `Range`, tak i właściwość `Cells` jest używana w stosunku do obiektów `Worksheet` i `Range`. Kiedy zajrzesz do pomocy systemowej programu Excel, przekonasz się, że właściwość `Cells` ma trzy warianty składni:

```
obiekt.Cells(rowIndex, columnIndex)  
obiekt.Cells(rowIndex)  
obiekt.Cells
```

Używanie scalonych komórek

Praca ze scalonymi komórkami może być niezłyym wyzwaniem. Jeżeli w zakresie komórek znajdują się komórki scalone, prawdopodobnie będziesz musiał wykonać w makrach kilka specjalnych operacji. Na przykład: jeżeli komórki A1:D1 zostały scalone, wykonanie poniższego polecenia spowoduje zaznaczenie kolumn od A do D (a nie tylko kolumny B, jak można by się było spodziewać):

```
Columns("B:B").Select
```

Szczerze mówiąc, nie wiem, czy takie nieoczekiwane zachowanie jest zamierzone, czy też po prostu jest wynikiem błędu. Nie zmienia to jednak w niczym faktu, że może to spowodować zupełnie nieoczekiwane zachowanie Twojego makra. Scalone komórki arkusza sprawiają również sporo kłopotów podczas sortowania.

Aby określić, czy w danym zakresie arkusza znajdują się jakieś scalone komórki, możesz użyć funkcji VBA przedstawionej poniżej. Funkcja zwraca wartość True, jeżeli w podanym zakresie znajduje się jakakolwiek komórka scalona (więcej szczegółowych informacji na temat funkcji znajdziesz w rozdziale 8.).

```
Function ContainsMergedCells(rng As Range)
    Dim cell As Range
    For Each cell In rng
        If cell.MergeCells Then
            ContainsMergedCells = True
            Exit Function
        End If
    Next cell
    ContainsMergedCells = False
End Function
```

Aby odwołać się do scalonych komórek, powinieneś odwołać się do całego scalonego zakresu lub po prostu do lewej, górnej komórki scalonego zakresu. Na przykład: jeżeli w naszym arkuszu zostały scalone cztery komórki o adresach A1, B1, A2 oraz B2, do scalonych komórek możesz odwołać się, używając jednego z poniższych poleceń:

```
Range("A1:B2")
Range("A1")
```

Jeżeli spróbujesz przypisać wartość komórce znajdującej się w scalonym zakresie, innej niż lewa, górna komórka tego zakresu, VBA zignoruje takie polecenie i nie wyświetli żadnego błędu. Na przykład polecenie przedstawione poniżej nie przyniesie żadnego efektu, jeżeli zakres A1:B2 został scalony:

```
Range("B2").Value = 43
```

Próba wykonania niektórych operacji spowoduje, że Excel wyświetli na ekranie okno dialogowe z prośbą o potwierdzenie zamiaru wykonania takiej operacji. Na przykład: jeżeli zakres A1:B2 został scalony, wykonanie polecenia przedstawionego poniżej spowoduje wyświetlenie następującego komunikatu: *Ta operacja spowoduje, że scalenie niektórych komórek zostanie cofnięte. Czy chcesz kontynuować?*

```
Range("B2").Delete
```

Wnioski? Podczas pracy ze scalonymi komórkami powinieneś zachować szczególną ostrożność. Niektórzy użytkownicy twierdzą, że mechanizm ten nie został zbyt dobrze przemyślany przed implementacją, i ja zdecydowanie jestem skłonny zgodzić się z tą opinią.

Poniżej podano kilka przykładów zastosowania właściwości Cells. W pierwszym z nich do komórki A1 arkusza Arkusz1 zostanie wpisana wartość 9. W tym przypadku posłużę się pierwszą składnią akceptującą numer indeksu wiersza (wartości z przedziału od 1 do 1048576) i numer indeksu kolumny (wartości z przedziału od 1 do 16384):

```
Worksheets("Arkusz1").Cells(1, 1) = 9
```

Oto instrukcja wpisująca do komórki D3 (czyli komórki znajdującej się na przecięciu wiersza numer 3 i kolumny numer 4 aktywnego arkusza) wartość 7:

```
ActiveSheet.Cells(3, 4) = 7
```

Można też użyć właściwości Cells obiektu Range. W tym przypadku obiekt Range zwracany przez właściwość Cells jest względny w stosunku do górnej lewej komórki obiektu Range, do którego jest wykonywane odwołanie. Zagmatwane? Raczej tak. Kolejny przykład może sprawić, że stanie się to bardziej przejrzyste. Instrukcja wpisuje do aktywnej komórki wartość 5. Pamiętaj, że aktywna komórka jest traktowana tak, jakby była komórką A1 arkusza.

```
ActiveCell.Cells(1, 1) = 5
```



Prawdziwa korzyść wynikająca z zastosowania tego typu odwołania do komórek stanie się widoczna przy omawianiu zmiennych i pętli (patrz rozdział 6.). W większości przypadków zamiast faktycznych wartości argumentów będziesz raczej używać zmiennych.

Aby do komórki znajdującej się bezpośrednio poniżej aktywnej komórki wpisać wartość 5, należy użyć następującej instrukcji:

```
ActiveCell.Cells(2, 1) = 5
```

Instrukcję tę można opisać za pomocą następującego zdania: „Rozpocznij od aktywnej komórki i traktuj ją tak, jakby to była komórka A1. Następnie wpisz wartość 5 w komórce znajdującej się w drugim wierszu i pierwszej kolumnie”.

W drugim wariantie składni właściwości Cells jest używany jeden argument, który może przyjmować wartości z przedziału od 1 do 17 179 869 184. Wartość ta jest równa liczbie komórek arkusza programu Excel 2013. Komórki są numerowane, począwszy od A1, kolejno w prawą stronę, a następnie od początku następnego wiersza. Na przykład komórka numer 16 384 ma adres XFD1, natomiast komórka o numerze 16 385 ma już adres A2.

W kolejnym przykładzie do komórki S71 aktywnego arkusza (która jest 520. komórką tego arkusza) wprowadzono wartość 2.

```
ActiveSheet.Cells(520) = 2
```

Aby wyświetlić wartość ostatniej komórki arkusza (o adresie XFD1048576), należy użyć następującej instrukcji:

```
MsgBox ActiveSheet.Cells(17179869184)
```

Taka składnia może też zostać użyta w przypadku obiektu Range. W tym przypadku zwrócona komórka jest względna w stosunku do obiektu Range, do którego się odwoano. Jeżeli na przykład obiekt Range reprezentuje zakres A1:D10 (40 komórek), właściwość Cells może pobrać argument o wartości z przedziału od 1 do 40 i zwrócić jedną z komórek obiektu Range. W poniższym przykładzie wartość 2000 jest wpisywana do komórki A2, ponieważ jest to piąta komórka zakresu (licząc od góry i w prawą stronę, a następnie od początku następnego wiersza), do którego się odwoano:

```
Range("A1:D10").Cells(5) = 2000
```



W poprzednim przykładzie argument właściwości Cells nie jest ograniczony do wartości z przedziału od 1 do 40. Jeżeli wartość argumentu przekracza liczbę komórek zakresu, zliczanie jest kontynuowane, tak jakby zakres był większy niż w rzeczywistości. A zatem polecenie podobne do powyższego może zmienić wartość komórki spoza zakresu A1:D10. Na przykład polecenie przedstawione poniżej zmienia wartość komórki A11:

```
Range("A1:D10").Cells(41) = 2000
```

Trzeci wariant składni właściwości Cells po prostu zwraca wszystkie komórki arkusza, do którego jest wykonywane odwołanie. W przeciwieństwie do dwóch pozostałych składni w tym przypadku nie jest zwracana pojedyncza komórka. Poniższa instrukcja używa metody ClearContents w stosunku do zakresu zwróconego przez właściwość Cells aktywnego arkusza. W wyniku wykonania instrukcji czyszczona jest zawartość wszystkich komórek arkusza.

```
ActiveSheet.Cells.ClearContents
```

Pobieranie informacji z komórki

Jeżeli chcesz odczytać zawartość wybranej komórki, to VBA udostępnia Ci kilka właściwości. Poniżej zamieszczamy krótkie zestawienie najczęściej używanych właściwości:

- Właściwość Formula zwraca formułę zapisaną w pojedynczej komórce (pod warunkiem oczywiście, że taka formuła istnieje). Jeżeli dana komórka nie posiada formuły, zwracana jest wartość komórki. Właściwość Formula jest właściwością przeznaczoną do odczytu i zapisu i posiada kilka podobnych odmian, takich jak FormulaR1C1, FormulaLocal czy FormulaArray (więcej szczegółowych informacji na ich temat znajdziesz w systemie pomocy).
- Właściwość Value zwraca nieformatowaną wartość komórki. Jest to właściwość przeznaczona do odczytu i zapisu.
- Właściwość Text zwraca tekst, który jest wyświetlany w komórce. Jeżeli w danej komórce znajduje się wartość numeryczna, właściwość zwraca wartość razem z formatowaniem, takim jak przecinki czy symbole walut. Właściwość Text jest przeznaczona tylko do odczytu.
- Właściwość Value2 jest nieco podobna do właściwości Value, z wyjątkiem tego, że nie korzysta z danych typu Date oraz Currency. W przypadku napotkania takich typów danych, właściwość dokonuje ich konwersji na zmienne typu Variant zawierające wartości typu Double. Na przykład, jeżeli w komórce została zapisana data 2013-03-16, to właściwość Value zwróci ją jako wartość typu Date, podczas gdy właściwość Value2 zwróci wartość typu Double (w tym wypadku 41349).

Właściwość Offset

Właściwość **Offset** (podobnie jak właściwości **Range** i **Cells**) również zwraca obiekt typu **Range**. Jednak w przeciwieństwie do dwóch poprzednio omawianych metod, właściwość **Offset** jest powiązana tylko i wyłącznie z obiektem **Range**. Oto jej składnia:

```
obiekt.Offset(rowOffset, columnOffset)
```

Właściwość **Offset** pobiera dwa argumenty odpowiadające względnemu położeniu odniesieniemu do górnej lewej komórki określonego obiektu **Range**. Argumentami mogą być wartości dodatnie (przesunięcie w dół lub w prawo), ujemne (przesunięcie w góre lub w lewo) lub zero. Poniższa instrukcja wpisuje wartość 12 do komórki znajdującej się bezpośrednio poniżej komórki aktywnej:

```
ActiveCell.Offset(1, 0).Value = 12
```

Kolejna instrukcja wstawia wartość 15 do komórki znajdującej się bezpośrednio powyżej aktywnej komórki:

```
ActiveCell.Offset(-1, 0).Value = 15
```

Nawiąsem mówiąc, jeżeli aktywna komórka znajduje się w wierszu 1, właściwość **Offset** z powyższej instrukcji spowoduje wygenerowanie błędu, ponieważ nie będzie mogła zwrócić obiektu **Range**, który nie istnieje.

Właściwość **Offset** jest dość przydatna, zwłaszcza w przypadku stosowania zmiennych w procedurach wykonujących pętle. Więcej szczegółowych informacji na ten temat znajdziesz w kolejnym rozdziale.

Po zarejestrowaniu makra w trybie odwołań względnych, Excel używa właściwości **Offset** w celu odwołania się do komórek względnych w stosunku do pozycji początkowej (komórki, która jest aktywna w momencie rozpoczęcia rejestrowania makra). Na przykład do wygenerowania poniższego kodu użyłem rejestratora makr. Najpierw umieściłem kursor w komórce B1, a następnie wprowadziłem wartości do komórek zakresu B1:B3 i powróciłem do komórki B1.

```
Sub Makrol()
    ActiveCell.FormulaR1C1 = "1"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "2"
    ActiveCell.Offset(1, 0).Range("A1").Select
    ActiveCell.FormulaR1C1 = "3"
    ActiveCell.Offset(-2, 0).Range("A1").Select
End Sub
```

Zwróć uwagę, że rejestrator makr użył właściwości **FormulaR1C1**. Zazwyczaj w celu wprowadzenia wartości do komórki będziemy używać właściwości **Value**, co nie zmienia jednak faktu, że po zastosowaniu właściwości **FormulaR1C1** lub nawet **Formula** uzyskasz takie same wyniki.

Warto również zauważyc, że wygenerowany kod odwołuje się do komórki A1, co może wydawać się trochę dziwne, ponieważ komórka ta nie była nawet użyta w trakcie rejestrowania makra. Jest to nietypowy element procedury rejestrowania, który sprawia, że kod

źródłowy jest niepotrzebnie aż tak złożony. Możesz śmiało usunąć wszystkie odwołania do Range("A1"), a makro nadal będzie znakomicie działać:

```
Sub Zmodyfikowane Makrol()
    ActiveCell.FormulaR1C1 = "1"
    ActiveCell.Offset(1, 0).Select
    ActiveCell.FormulaR1C1 = "2"
    ActiveCell.Offset(1, 0).Select
    ActiveCell.FormulaR1C1 = "3"
    ActiveCell.Offset(-2, 0).Select
End Sub
```

Poniżej zamieszczamy jeszcze bardziej efektywną wersję makra, która nie używa metody Select:

```
Sub Makrol()
    ActiveCell = 1
    ActiveCell.Offset(1, 0) = 2
    ActiveCell.Offset(2, 0) = 3
End Sub
```

Co należy wiedzieć o obiektach?

W poprzednich punktach zapoznałeś się z obiektami, kolekcjami obiektów, właściwościami i metodami. Ale to zaledwie wierzchołek góry lodowej.

Podstawowe zagadnienia, które należy zapamiętać

W tym podrozdziale przedstawimy kilka dodatkowych zagadnień, których opanowanie jest niezbędne dla każdego, kto chce na poważnie zajmować się programowaniem w języku VBA. Poszczególne zagadnienia staną się bardziej zrozumiałe w miarę zdobywania doświadczenia i czytania kolejnych rozdziałów.

- **Obiekty posiadają unikatowe właściwości i metody** — Każdy obiekt dysponuje własnym zestawem właściwości i metod. Jednak niektóre obiekty mają wspólne niektóre właściwości (np. Name) i metody (np. Delete).
- **Możesz manipulować obiektami bez ich zaznaczania** — Może to być niezgodne ze standardowym wyobrażeniem przetwarzania obiektów w Excelu. W praktyce jednak wykonywanie operacji na obiektach bez ich uprzedniego zaznaczenia jest zazwyczaj efektywniejsze. Kiedy rejestrujesz nowe makro, Excel najpierw zaznacza obiekt, a dopiero potem wykonuje na nim operacje. Nie jest to konieczne i właściwie spowalnia działanie makra.
- **Powinieneś dokładnie zrozumieć koncepcję i zasady pracy z kolekcjami** — W większości przypadków odwołujesz się do obiektu w sposób pośredni za pośrednictwem kolekcji, w której dany obiekt się znajduje. Aby odwołać się na przykład do obiektu klasy Workbook o nazwie Mój_plik, powinieneś użyć następującego odwołania do kolekcji Workbooks:

```
Workbooks("Mój_plik.xlsx")
```

Powyższe odwołanie zwraca obiekt, który jest żądany przez Ciebie skoroszytem.

- **Właściwości mogą zwracać odwołanie do innego obiektu** — Przykładowo w poniższej instrukcji właściwość Font zwraca obiekt Font zawarty w obiekcie Range. Bold to właściwość obiektu Font, a nie obiektu Range.

```
Range("A1").Font.Bold = True
```

- **Istnieje wiele różnych metod odwoływania się do tego samego obiektu**

— Założymy, że dysponujesz skoroszytem o nazwie Sprzedaż i jest to jedyny otwarty w danej chwili skoroszyt. Przyjmijmy też, że skoroszyt zawiera jeden arkusz o nazwie Zestawienie. Do arkusza można odwołać się przy użyciu następujących sposobów:

```
Workbooks("Sprzedaż.xlsx").Worksheets("Zestawienie")  
Workbooks(1).Worksheets(1)  
Workbooks(1).Sheets(1)  
Application.ActiveWorkbook.ActiveSheet  
ActiveWorkbook.ActiveSheet  
ActiveSheet
```

To, która metoda zostanie wybrana, przeważnie zależy od wiedzy użytkownika na temat obszaru roboczego. Jeżeli na przykład otwartych jest więcej skoroszytów niż jeden, nieodpowiednia będzie druga lub trzecia metoda. Jeżeli używasz aktywnego arkusza (dowolnego typu), każda z trzech ostatnich metod będzie skuteczna. Aby mieć całkowitą pewność, że odwołujesz się do właściwego arkusza określonego skoroszytu, zastosuj pierwszą metodę.

Dodatkowe informacje na temat obiektów i właściwości

Jeżeli po raz pierwszy masz styczność z językiem VBA, prawdopodobnie będziesz trochę przytłoczony mnogością obiektów, właściwości i metod. Nie ma w tym żadnej Twojej winy. Jeżeli spróbujesz użyć właściwości, której obiekt nie posiada, zostanie wygenerowany błąd *runtime* i wykonywanie kodu języka VBA zostanie wstrzymane do czasu usunięcia problemu.

Na szczęście istnieje kilka dobrych sposobów zdobycia informacji na temat obiektów, właściwości i metod.

Przeczytaj resztę książki

Nie zapomnij o tym, że tytuł tego rozdziału brzmi „Wprowadzenie do języka VBA”. W pozostałej części książki znajdziesz wiele dodatkowych szczegółów oraz przydatnych i pouczających przykładów.

Rejestruj wykonywane operacje

Bez wątpienia najlepszą metodą zaznajomienia się z językiem VBA jest po prostu uaktywnienie rejestratora makr i zapisanie kilku operacji wykonywanych w Excelu. Jest to szybki sposób zdobycia wiedzy na temat obiektów, właściwości i metod stosowanych w określonym zadaniu. Jeszcze lepszym rozwiązaniem jest obserwowanie podczas rejestrowania makra modułu VBA, w którym generowany jest kod.

Korzystaj z systemu pomocy

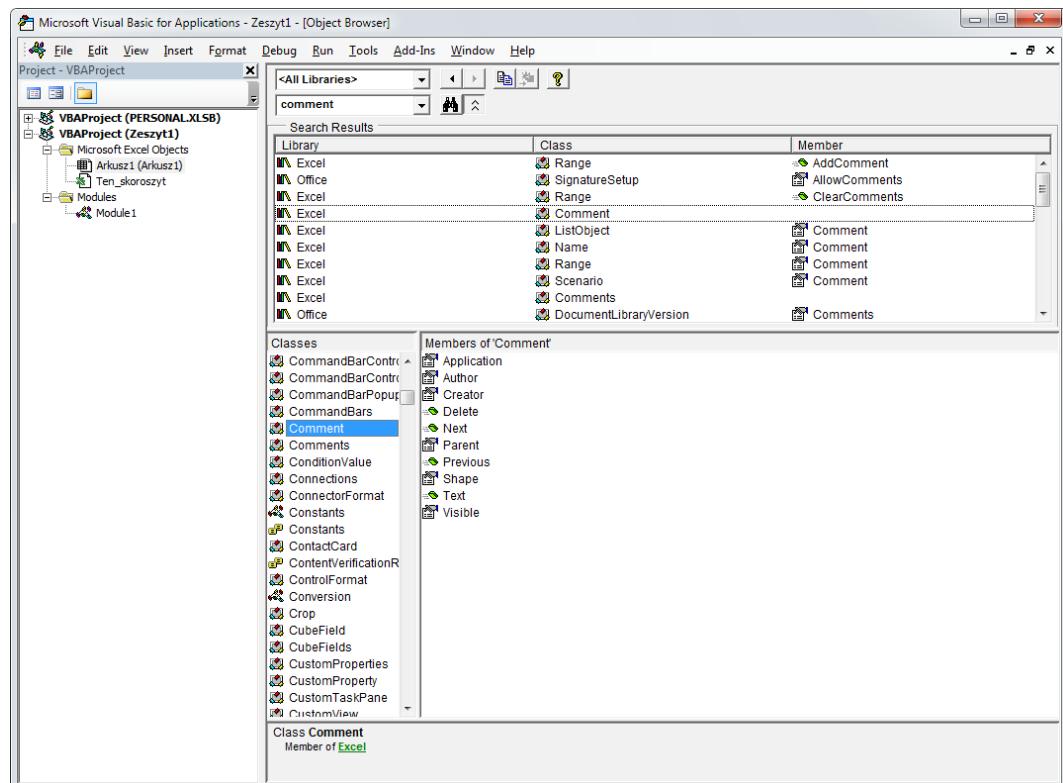
Podstawowym źródłem szczegółowych informacji na temat obiektów, metod i procedur stosowanych w Excelu jest jego system pomocy. Wielu użytkowników po prostu zapomina o tym zasobie.

Używaj przeglądarki obiektów

Object Browser, czyli przeglądarka obiektów, jest bardzo przydatnym narzędziem wyświetlającym wszystkie właściwości i metody dostępne dla danego obiektu. Po uaktywnieniu edytora *Visual Basic* przeglądarkę *Object Browser* można uruchomić na jeden z trzech sposobów:

- Naciśnij klawisz *F2*.
- Wybierz z menu *View* polecenie *Object Browser*.
- Kliknij przycisk *Object Browser* znajdujący się na pasku narzędzi *Standard*.

Okno przeglądarki *Object Browser* zostało przedstawione na rysunku 5.15.



Rysunek 5.15. Przeglądarka *Object Browser* jest znakomitym źródłem informacji o obiektach

Lista rozwijana w górnym lewym narożniku przeglądarki *Object Browser* zawiera wszystkie dostępne biblioteki obiektów:

- Program Excel.
- *MSForms* (używana do tworzenia niestandardowych okien dialogowych).
- *Office* (obiekty wspólne dla wszystkich aplikacji pakietu Microsoft Office).
- *Stdole* (obiekty automatyzacji OLE).
- VBA.
- Bieżący projekt (wybrany w oknie *Project Explorer*) i wszystkie skoroszyty, do których się odwołuje.

Zawartość okna *Classes* zależy od elementu wybranego z listy rozwijanej. Z kolei zawartość okna *Members of* zależy od elementu, który zostanie zaznaczony w oknie *Classes*.

Jeśli po wybraniu biblioteki chcesz uzyskać listę właściwości i metod zawierających określony tekst, możesz go wprowadzić w polu szukania *Search Text* drugiej listy rozwijanej, a następnie kliknąć ikonę *Search* (lornetka). Dla przykładu założmy, że realizujesz projekt, w którym są przetwarzane komentarze zawarte w komórce. Wykonaj następujące polecenia:

1. Zaznacz interesującą Cię bibliotekę.
2. Jeżeli nie jesteś pewien, która biblioteka obiektów jest właściwa, wybierz pozycję *All Libraries*.
3. W polu listy rozwijanej znajdującej się poniżej listy bibliotek wpisz słowo *Comment*.
4. Aby rozpocząć wyszukiwanie, kliknij ikonę lornetki.

Wyniki wyszukiwania zostaną wyświetcone w oknie *Search Results*. Aby w oknie *Classes* wyświetlić klasy obiektu, zaznacz go na liście. Aby uzyskać listę składników klasy (właściwości, metody i stałe), kliknij nazwę klasy. Zwróć uwagę na dolny panel wyświetlający dodatkowe informacje o obiekcie. Aby bezpośrednio przejść do właściwego tematu pomocy, naciśnij klawisz *F1*.

Na pierwszy rzut oka korzystanie z przeglądarki *Object Browser* może się wydawać trudne, ale z czasem się do niej przyzwyczaisz i docenisz jej przydatność.

Eksperymentuj z oknem Immediate

Jak już wspomniano wcześniej w ramce „Kilka słów o przykładach kodu”, okno *Immediate* edytora *Visual Basic* jest bardzo przydatne do testowania poleceń i wyrażeń języka VBA. W moim przypadku jest ono włączone niemal przez cały czas, gdyż bardzo często posługuję się nim przy testowaniu różnych wyrażeń oraz w trakcie identyfikowania i wykrywania błędów w kodzie źródłowym.

Rozdział 6.

Podstawy programowania

w języku VBA

W tym rozdziale:

- Przegląd elementów języka VBA, takich jak zmienne, typy danych, stałe i tablice
- Korzystanie z wbudowanych funkcji VBA
- Praca z obiektami i kolekcjami
- Sterowanie sposobem wykonywania procedur

Przegląd elementów języka VBA

Jeżeli pracowałeś już z innymi językami programowania, wiele informacji będzie Ci znanych. Język VBA posiada jednak kilka unikatowych cech, dlatego też nawet doświadczeni programiści z pewnoścą znajdą w tym rozdziale wiele nowych i ciekawych informacji.

W rozdziale 5. dokonaliśmy przeglądu obiektów, właściwości i metod, choć niewiele znalazło się tam informacji o tym, w jaki sposób manipułować obiektami tak, aby wykonywały potrzebne operacje. W tym rozdziale omówiono podstawowe *elementy języka VBA*, takie jak słowa kluczowe i struktury sterujące, których będziesz na co dzień używał podczas pisania aplikacji w języku VBA.

Na dobry początek zaprezentuję prostą procedurę Sub języka VBA. Program, którego kod VBA zamieszczono poniżej, oblicza sumę pierwszych 100 liczb całkowitych. Po zakończeniu obliczeń procedura wyświetla komunikat zawierający wynik.

```
Sub VBA_Demo()
    ' Przykład prostej procedury języka VBA
    Dim Total As Long, i As Long
    Total = 0
    For i = 1 To 100
        Total = Total + i
    Next i
    MsgBox Total
End Sub
```

Procedura zawiera kilka często stosowanych elementów języka VBA, takich jak:

- komentarz (wiersz rozpoczynający się apostrofem);
- deklaracje zmiennych (wiersz rozpoczynający się polecienniem Dim);
- dwie zmienne (Total oraz i);
- dwie instrukcje przypisania wartości do zmiennych (Total = 0 i Total = Total + i);
- struktura pętli (For ... Next);
- wbudowana funkcja VBA (MsgBox).

Wszystkie wymienione elementy języka VBA zostaną omówione w kolejnych punktach tego rozdziału.



Procedury języka VBA nie muszą wykonywać żadnych operacji na obiektach. Procedura omawiana w poprzednim przykładzie nie wykonuje żadnych operacji na obiektach i przetwarza jedynie liczby.

Wprowadzanie kodu źródłowego języka VBA

Kod źródłowy języka VBA znajdujący się w module VBA składa się z szeregu poleceń. W praktyce przyjęło się umieszczanie w wierszu jednej instrukcji, ale nie jest to konieczne. W celu oddzielenia wielu instrukcji zawartych w jednym wierszu należy użyć dwukropka. W poniższym przykładzie w jednym wierszu umieszczone zostały instrukcje.

```
Sub OneLine()
    x = 1: y = 2: z = 3: MsgBox x + y + z
End Sub
```

Łatwo jednak zauważyc, że kod źródłowy jest znacznie czytelniejszy po umieszczeniu w każdym wierszu tylko jednej instrukcji:

```
Sub OneLine()
    x = 1
    y = 2
    z = 3
    MsgBox x + y + z
End Sub
```

Poszczególne wiersze mogą mieć dowolną długość. W przypadku długich poleceń można użyć sekwencji złożonej ze spacji i znaku podkreślenia (_), oznaczającej kontynuację polecenia w kolejnym wierszu. Oto przykład:

```
Sub LongLine()
    SummedValue =
        Worksheets("Arkusz1").Range("A1").Value +
        Worksheets("Arkusz2").Range("A1").Value
End Sub
```

Przy rejestrowaniu makr Excel często używa znaków podkreślenia w celu podzielenia długich instrukcji na wiele wierszy.

Podczas wprowadzania kodu edytor VBE w celu zwiększenia czytelności wykonuje następujące operacje:

- **Pomiędzy operatorami wstawia spację** — Przykładowo po wprowadzeniu polecenia `Ans=1+2` (bez spacji) edytor VBE zamieni je na następującą postać:

`Ans = 1 + 2`

- **Edytor VBE zmienia odpowiednio wielkość znaków zawartych w słowach kluczowych oraz w nazwach właściwości i metod** — Po wprowadzeniu następującego polecenia:

`Result=activesheet.range("a1").value=12`

Edytor VBE zamieni je na postać:

`Result = ActiveSheet.Range("a1").Value=12`

Zwróć uwagę na to, że tekst zawarty w znakach cudzysłowu (w tym przypadku "a1") nie jest modyfikowany.

- **Ponieważ w nazwach zmiennych języka VBA nie jest rozróżniana wielkość znaków, edytor domyślnie używa w nich liter takiej samej wielkości jak w nazwie, którą wprowadzono jako ostatnią** — Jeżeli na przykład na początku wprowadziłeś zmienną `myvalue` (tylko małe litery), a następnie zmienną `MyValue` (połączenie dużych i małych liter), edytor VBE w przypadku wszystkich kolejnych wystąpień tej zmiennej użyje nazwy `MyValue`. Wyjątkiem jest sytuacja, gdy zmienna zostanie zadeklarowana przy użyciu słowa kluczowego `Dim` lub podobnej instrukcji. W tym przypadku nazwa zmiennej będzie miała taką postać, jak w wierszu deklaracji.
- **Edytor VBE sprawdza polecenia pod kątem błędów składni** — Jeżeli VBA znajdzie błąd, zmieni kolor wiersza i może wyświetlić komunikat opisujący problem. Okno dialogowe `Options` (uruchamiane w edytorze Visual Basic przez wybranie opcji `Options` w menu `Tools`) umożliwia określenie koloru błędnego wiersza (zakładka `Editor Format`) i tego, czy zostanie wyświetlony komunikat błędu (użyj opcji `Auto Syntax Check` w zakładce `Editor`).

Komentarze

Komentarz jest tekstem spełniającym funkcję opisu zawartego w kodzie źródłowym. VBA całkowicie ignoruje zawartość komentarza. Warto często stosować komentarze do opisu przeznaczenia poszczególnych poleceń, ponieważ nie zawsze jest to oczywiste.

Komentarz może zostać umieszczony w oddzielnym wierszu lub za instrukcją w tym samym wierszu. Komentarz jest rozpoczęty znakiem apostrofu. VBA ignoruje tekst znajdujący się za znakiem apostrofu do końca wiersza, z wyjątkiem apostrofu umieszczonego pomiędzy znakami cudzysłowu. Przykładowo poniższa instrukcja nie zawiera komentarza, mimo że w wierszu znajdują się apostrofy.

```
Msg = "'Nie mogę kontynuować'"
```

Kolejny przykład demonstruje procedurę języka VBA zawierającą trzy komentarze:

```
Sub Comments()
    ' Procedura nie wykonuje żadnej wartościowej operacji
    x = 0      ' wartość x niczego nie reprezentuje
    ' Wyświetlenie wyniku
    MsgBox x
End Sub
```

Co prawda apostrof jest preferowanym znakiem identyfikującym komentarz, ale w razie potrzeby możesz również użyć słowa kluczowego Rem. Oto przykład:

Rem -- Następna instrukcja prosi użytkownika o podanie nazwy pliku

Słowo kluczowe Rem (od ang. *Remark* — komentarz) jest pozostałością ze starszych wersji języka BASIC. W celu utrzymania kompatybilności słowo zostało uwzględnione w języku VBA. W przeciwieństwie do apostrofu, słowo kluczowe Rem musi być umieszczone na początku wiersza polecenia.

Poniżej zamieszczono kilka ogólnych wskazówek dotyczących poprawnego korzystania z komentarzy:

- Używaj komentarzy do krótkiego opisu przeznaczenia każdej procedury.
- Używaj komentarzy do opisu zmian dokonanych w procedurze.
- Używaj komentarzy do zaznaczenia, że funkcje lub konstrukcje są używane w nietypowy lub niestandardowy sposób.
- Używaj komentarzy do opisania przeznaczenia zmiennych, dzięki czemu użytkownicy będą w stanie rozszyfrować ich czasem niewiele mówiące nazwy.
- Używaj komentarzy do opisania obejść, które zastosowałaś w celu poradzenia sobie z błędami lub ograniczeniami Excela.
- Pamiętaj, aby dołączać komentarze *podczas* pisania kodu programu, a nie dopiero po jego zakończeniu.



W niektórych przypadkach istnieje możliwość przetestowania procedury bez dołączania określonego polecenia lub grupy poleceń. Zamiast usuwać instrukcję, wystarczy wstawić na początku wiersza apostrof i zamienić ją w ten sposób na komentarz. W efekcie podczas wykonywania procedury takie polecenie zostanie zignorowane. Aby z powrotem zamienić komentarz na instrukcję, należy usunąć znak apostrofu.

Pasek narzędzi Edyt edytora Visual Basic posiada dwa bardzo przydatne przyciski. (Pasek narzędzi Edyt nie jest domyślnie wyświetlanym na ekranie; aby go przywołać, z menu głównego edytora VBE wybierz polecenie View/Toolbars/Edit). Po zaznaczeniu grupy poleceń możesz je szybko zamienić na komentarze, naciskając przycisk *Comment Block*. Przycisk *Uncomment Block* zamienia grupę komentarzy z powrotem na polecenia.

Zmienne, typy danych i stałe

Podstawowym zadaniem języka VBA jest przetwarzanie danych. Niektóre dane znajdują się w obiektach, takich jak zakresy arkusza. Inne są przechowywane w utworzonych przez Ciebie zmiennych.

Zmienna jest po prostu nazwanym obszarem przechowywania danych zawartym w pamięci komputera. Zmienne mogą przechowywać wiele różnych *typów danych*, począwszy od prostego typu Boolean (wartości True lub False), a skończywszy na dużych wartościach typu Double (patrz następny podrozdział). Wartość jest przypisywana zmiennej przy użyciu operatora przypisania mającego postać znaku równości (więcej na ten temat w kolejnym punkcie „Instrukcje przypisania”).

Tworzenie kodu źródłowego stanie się prostsze, jeżeli wyrobisz w sobie nawyk nadawania zmiennym jak najbardziej zrozumiałych nazw. Język VBA narzuca jednak w stosunku do nazw zmiennych kilka zasad:

- W nazwach zmiennych możesz stosować znaki alfanumeryczne, liczby i niektóre znaki interpunkcji, ale pierwszy znak musi być literą.
- Język VBA nie rozróżnia wielkości znaków. Aby nazwy zmiennych były bardziej czytelne, programiści często stosują znaki różnej wielkości (na przykład InterestRate zamiast interestrate).
- W nazwach zmiennych nie możesz stosować spacji lub kropek. Aby nazwy były bardziej czytelne, programiści często używają znaku podkreślenia (Interest_Rate).
- W nazwach zmiennych nie możesz umieszczać znaków specjalnego typu używanych w deklaracjach (#, \$, %, & lub !).
- Nazwy zmiennych mogą zawierać maksymalnie 254 znaki, ale ze względów praktycznych tworzenie zmiennych o tak długich nazwach nie jest zalecane.

Poniżej zawarto kilka przykładów instrukcji przypisania stosujących różnych typu zmienne. Nazwy zmiennych znajdują się po lewej stronie znaku równości. W każdej instrukcji zmiennej umieszczonej po lewej stronie wyrażenia przypisywana jest wartość widoczna po prawej stronie.

```
x = 1  
InterestRate = 0.075  
LoanPayoffAmount = 243089  
DataEntered = False  
x = x + 1  
MyNum = YourNum * 1.25  
UserName = "Jan Nowak"  
DateStarted = #12/14/2009#
```

Język VBA posiada wiele *zastrzeżonych słów kluczowych*, których nie można używać w roli nazw zmiennych lub procedur. Jeżeli będziesz próbował użyć jednego z tych słów, zostanie wygenerowany komunikat błędu. Na przykład pomimo tego, że zastrzeżone słowo kluczowe Next może być bardzo oczywistą nazwą zmiennej, to jednak po zastosowaniu poniższej instrukcji zostanie wygenerowany błąd składni:

```
Next = 132
```

Niestety komunikaty błędu składni nie zawsze są jasne. Powyższa instrukcja generuje komunikat błędu o treści: *Compile error: Expected variable* (Błąd komplikacji: oczekiwana zmienna). Wcale by mnie nie zmartwiło, gdyby zamiast tego komunikat błędu miał na przykład taką oto treść: *Reserved word used as a variable* (Jako nazwy zmiennej użyto zastrzeżonego słowa). Jeżeli zatem instrukcja generuje dziwny komunikat błędu, powinieneś zatrzymać się i sprawdzić, czy użyta nazwa zmiennej nie spełnia w języku VBA jakiejś specjalnej funkcji.

Definiowanie typów danych

VBA ułatwia życie programistom, ponieważ automatycznie zajmuje się wszystkimi szczegółami związanymi z przetwarzaniem danych. Nie wszystkie języki programowania są tak wygodne w użytkowaniu. Niektóre języki programowania są *ściśle deklarowane*, co oznacza, że programista musi jawnie zdefiniować typ danych dla każdej zastosowanej zmiennej.

Typ danych określa, w jakiej postaci dane są zapisywane w pamięci (np. liczby całkowite, liczby rzeczywiste,łańcuchy znaków itd.). Co prawda, VBA może automatycznie wstawiać typy danych w tracie realizacji programu, ale odbywa się to kosztem szybkości i powoduje mniej efektywne wykorzystanie pamięci. Zezwolenie VBA na samodzielne obsługiwanie typów danych może wywołać problemy z wydajnością po uruchomieniu dużej lub złożonej aplikacji. Dodatkową zaletą jawnego deklarowania poprawnych typów zmiennych jest to, że VBA już na etapie komplikacji może dokonać dodatkowego sprawdzenia poprawności danych — w przeciwnym wypadku błędy związane z niepoprawnymi typami danych mogą być niezwykle trudne do zlokalizowania.

W tabeli 6.1 zawarto zestawienie wbudowanych typów danych języka VBA (pamiętaj, że możesz również zdefiniować niestandardowe typy danych, które zostały omówione w dalszej części rozdziału w podrozdziale „Typy danych definiowane przez użytkownika”).



Typ danych `Decimal` jest dosyć nietypowy, ponieważ właściwie nie możesz zadeklarować zmiennej tego typu. Tak naprawdę `Decimal` to podtyp typu `Variant`. Aby zamienić typ `Variant` na typ `Decimal`, powinieneś użyć funkcji `CDec` języka VBA.

Zazwyczaj najlepiej posłużyć się typem danych, który zajmuje najmniejszą liczbę bajtów i jednocześnie obsługuje wszystkie dane, które zostaną z nim powiązane. W trakcie przetwarzania danych przez VBA szybkość działania zależy od liczby bajtów będących do dyspozycji. Innymi słowy, im mniej bajtów jest wykorzystywanych do przechowywania danych, tym szybciej VBA uzyska do nich dostęp i je przetworzy.

Ponieważ Excel przy obliczeniach wykonywanych w arkuszach posługuje się typem danych `Double`, w celu uniknięcia utraty dokładności warto przy przetwarzaniu wartości liczbowych za pomocą języka VBA używać tego typu danych. W przypadku przetwarzania liczb całkowitych można użyć typu `Integer`, jeżeli masz pewność, że wartości nie przekroczą liczby 32 767. W przeciwnym razie powinieneś zastosować dane typu `Long`. Co więcej, użycie zmiennych typu `Long` jest zalecane nawet dla wartości mniejszych niż 32 767, ponieważ dane typu `Long` są przetwarzane nieco szybciej niż dane typu `Integer`. Podczas przetwarzania numerów wierszy arkusza Excela również należy używać typu `Long`, ponieważ liczba wierszy przekracza maksymalną wartość zakresu danych typu `Integer`.

Tabela 6.1. Wbudowane typy danych języka VBA

Typ danych	Liczba używanych bajtów	Zakres wartości
Byte	1 bajt	Od 0 do 255
Boolean	2 bajty	True lub False
Integer	2 bajty	Od -32 768 do 32 767
Long	4 bajty	Od -2 147 483 648 do 2 147 483 647
Single	4 bajty	Od -3,402823E38 do -1,401298E-45 (dla wartości ujemnych) Od 1,401298E-45 do 3,402823E38 (dla wartości dodatnich)
Double	8 bajtów	Od -1,79769313486232E308 do -4,94065645841247E-324 (dla wartości ujemnych) Od 4,94065645841247E-324 do 1,79769313486232E308 (dla wartości dodatnich)
Currency	8 bajtów	Od -922 337 203 685 477,5808 do 922 337 203 685 477,5807
Decimal	12 bajtów	+/-79 228 162 514 264 337 593 543 950 335 (bez części dziesiętnej) +/-7,9228162514264337593543950335 (z 28 miejscami po przecinku)
Date	8 bajtów	Od 1 stycznia 0100 roku do 31 grudnia 9999 roku
Object	4 bajty	Dowolne odwołanie do obiektu
String (zmienna długość)	10 bajtów+długość łańcucha	Od 0 do w przybliżeniu 2 miliardów znaków
String (stała długość)	Długość łańcucha	Od 1 do w przybliżeniu 65 400 znaków
Variant (z liczbami)	16 bajtów	Dowolna wartość numeryczna aż do maksymalnej wartości zakresu typu danych Double. Dodatkowo może przechowywać wartości specjalne, takie jak Empty, Error, Nothing oraz Null.
Variant (ze znakami)	22 bajty+długość łańcucha	Od 0 do w przybliżeniu 2 miliardów
Zdefiniowany przez użytkownika	Zmienna	Zmienia się zależnie od elementu

Deklarowanie zmiennych

Jeżeli nie zadeklarujesz typu danych dla zmiennej, VBA użyje domyślnego typu Variant. Dane typu Variant zachowują się trochę jak kameleon: zmieniają swój typ w zależności od tego, w jaki sposób są przetwarzane.

Testowanie wydajności typu danych Variant

Aby przekonać się, czy typ danych odgrywa ważną rolę, utworzyłem poniższą procedurę, która w pętli wykonuje 300 milionów prostych obliczeń i na zakończenie wyświetla całkowity czas przetwarzania:

```
Sub TimeTest()
    Dim x As Long, y As Long
    Dim A As Double, B As Double, C As Double
    Dim i As Long, j As Long
    Dim StartTime As Date, EndTime As Date
    Zapisz czas uruchomienia
    StartTime = Timer
    Wykonaj kilka obliczeń
    x = 0
    y = 0
    For i = 1 To 10000
        x = x + 1
        y = y + 1
        For j = 1 To 10000
            A = x + y + i
            B = y - x - i
            C = x / y * i
        Next j
    Next i
    Pobierz czas zakończenia
    EndTime = Timer
    Wyświetl całkowity czas działania wyrażony w sekundach
    MsgBox Format(EndTime - StartTime, "0.0")
End Sub
```

W moim systemie wykonanie procedury zajęło 6,8 sekundy (czas może być inny w zależności od szybkości procesora Twojego komputera). Następnie wyłączyłem instrukcje Dim deklarujące typy danych. Oznacza to, że zamieniłem je na komentarze poprzez umieszczenie apostrofu na początku każdego wiersza. W efekcie VBA użył domyślnego typu danych Variant. Ponowne wykonanie procedury zajęło 22,1 sekundy, czyli trwało ponad trzy razy dłużej niż poprzednio.

Wniosek jest prosty. Jeżeli chcesz, aby aplikacje napisane w języku VBA działały szybko i efektywnie, powinieneś zadeklarować wszystkie zmienne!

Skoroszyt z tym programem (*miar czasu.xlsm*) znajdziesz na stronie internetowej tej książki (patrz <http://www.helion.pl/ksiazki/e13pvw.htm>).

Poniższa procedura demonstruje, w jaki sposób zmienna może przyjmować różne typy danych:

```
Sub VariantDemo()
    MyVar = True
    MyVar = MyVar * 100
    MyVar = MyVar / 4
    MyVar = "Wynik: " & MyVar
    MsgBox MyVar
End Sub
```

W procedurze VariantDemo zmiennej MyVar początkowo jest przypisywana wartość logiczna typu Boolean. Następnie zostaje na niej wykonana operacja mnożenia, która zamienia typ zmiennej na Integer. W kolejnym wierszu wartość tej zmiennej zostaje podzielona

przez 4, co zamienia typ zmiennej na Double, i wreszcie na koniec wartość zmiennej zostaje połączona za pomocą operatora konkatenacji z łańcuchem tekstu, co w efekcie zamienia typ zmiennej na String. Polecenie MsgBox wyświetla końcowy wynik działania programu, czyli ciąg znaków *Wynik: -25*.

Aby na własnej skórze przekonać się, jakie potencjalne problemy są związane z wykorzystywaniem danych typu Variant, spróbuj wykonać następującą procedurę:

```
Sub VariantDemo2()
    MyVar = "123"
    MyVar = MyVar + MyVar
    MyVar = "Wynik: " & MyVar
    MsgBox MyVar
End Sub
```

W oknie komunikatu zostanie wyświetlony łańcuch *Wynik: 123123*. Prawdopodobnie *nie tego* się spodziewałeś. W przypadku typu danych Variant przechowującego łańcuchy tekstowe operator + dokonuje ich połączenia.

Określanie typu danych

Do określenia typu danych zmiennej możesz użyć funkcji TypeName języka VBA. Poniżej zawarto zmodyfikowaną wersję procedury VariantDemo. W tej wersji po każdej instrukcji przypisania jest wyświetlany typ danych zmiennej MyVar.

```
Sub VariantDemo3()
    MyVar = True
    MsgBox TypeName(MyVar)
    MyVar = MyVar * 100
    MsgBox TypeName(MyVar)
    MyVar = MyVar / 4
    MsgBox TypeName(MyVar)
    MyVar = "Wynik: " & MyVar
    MsgBox TypeName(MyVar)
    MsgBox MyVar
End Sub
```

Dzięki językowi VBA konwersja typów danych zmiennych, które nie zostały zadeklarowane, odbywa się automatycznie. Proces ten może się wydać prostym rozwiążaniem problemu, ale należy pamiętać, że odbywa się to kosztem szybkości działania programu i zwiększonej zajętości pamięci — a co więcej, zwiększasz w ten sposób ryzyko powstania błędów, których wystąpienia zupełnie się nie spodziewałeś.

Deklarowanie każdej zmiennej przed jej użyciem jest naprawdę znakomitym nawykkiem. Deklaracja zmiennej przekazuje VBA jej nazwę i typ danych. Dzięki deklarowaniu zmiennych uzyskujemy dwie podstawowe korzyści:

- **Programy działają szybciej i efektywniej korzystają z pamięci** — Domyślny typ Variant powoduje, że VBA wielokrotnie wykonuje czasochłonne sprawdzenie typu danych i zużywa więcej pamięci, niż jest to konieczne. Jeżeli VBA z góry zna typ danych, nie musi go określać i może od razu zarezerwować taką ilość pamięci, która będzie optymalna do przechowywania zmiennej danego typu.

- **Unikasz problemów związanych z błędnie wprowadzonymi nazwami zmiennych** — W tym przypadku zakładamy, że używasz opcji Option Explicit wymuszającej deklarowanie wszystkich zmiennych (więcej szczegółowych informacji na ten temat znajdziesz w kolejnym podrozdziale). Załóżmy, że używasz zmiennej o nazwie CurrentRate, która nie została zadeklarowana. W wybranym miejscu procedury wstawiasz instrukcję przypisania CurrentRate = .075. Ten bardzo trudny do wykrycia błąd w nazwie zmiennej prawdopodobnie spowoduje, że rezultaty działania procedury nie będą prawidłowe.

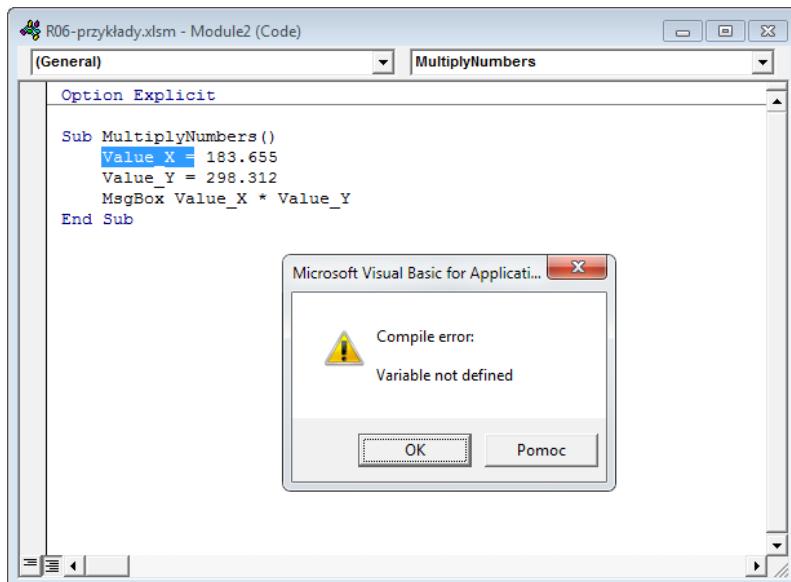
Wymuszanie deklarowania wszystkich zmiennych

Aby wymusić konieczność deklarowania wszystkich używanych zmiennych, w module VBA w pierwszym wierszu powinieneś umieścić następujące polecenie:

```
Option Explicit
```

Takie rozwiązanie spowoduje zatrzymanie programu za każdym razem, kiedy VBA napotka zmienną, która nie została zadeklarowana. Dodatkowo VBA wygeneruje komunikat błędu (patrz rysunek 6.1) i przed wznowieniem działania programu będziesz musiał poprawić kod i zadeklarować taką zmienną.

Rysunek 6.1.
Komunikat, który VBA informuje, że procedura zawiera niezadeklarowaną zmienną



Uwaga dotycząca przykładów zawartych w tym rozdziale

W tym rozdziale zamieszczono wiele przykładów kodu źródłowego języka VBA, zazwyczaj mających postać prostych procedur. Przykłady ilustrują omawiane zagadnienia w możliwie prosty sposób. Większość z nich nie wykonuje żadnych szczególnie użytecznych zadań i takie operacje często mogłyby być wykonane w inny (często bardziej efektywny) sposób. Innymi słowy, tych przykładów nie powinieneś raczej używać podczas tworzenia własnych aplikacji. W kolejnych rozdziałach znajdziesz znacznie więcej naprawdę przydatnych kodów procedur i funkcji VBA.



Aby upewnić się, że instrukcja Option Explicit zostanie automatycznie wstawiona w każdym nowym module VBA, powinieneś włączyć opcję *Require Variable Declaration* znajdującej się na karcie *Editor* okna dialogowego *Options* edytora Visual Basic (aby przywołać to okno na ekran, wybierz z menu głównego edytora VBE polecenie *Tools/Options*). Naprawdę warto to zrobić, ale przy okazji powinieneś pamiętać, że włączenie tej opcji nie ma żadnego wpływu na moduły już istniejące.

Zasięg zmiennych

Zasięg zmiennej określa moduły i procedury, w których daną zmienną można zastosować. W tabeli 6.2 zamieszczono zestawienie typów zasięgu zmiennych, z których możesz korzystać.

Tabela 6.2. Typy zasięgu zmiennych

Zasięg	Sposób deklarowania zmiennej o takim zasięgu
Pojedyncza procedura	Użyj polecenia <code>Dim</code> lub <code>Static</code>
Pojedynczy moduł	Użyj polecenia <code>Dim</code> lub <code>Private</code> przed pierwszą procedurą modułu
Wszystkie moduły	Użyj polecenia <code>Public</code> przed pierwszą procedurą modułu

W kolejnych podrozdziałach bardziej szczegółowo omówiono poszczególne typy zasięgu zmiennych.

Zmienne lokalne

Zmienna lokalna jest zmienną deklarowaną wewnętrz procedure. Zmienne lokalne mogą być używane wyłącznie w procedurze, w której zostały zadeklarowane. Po zakończeniu działania procedury zmienna lokalna przestaje istnieć i Excel zwalnia zajmowaną przez nią pamięć. Jeżeli chcesz, aby po zakończeniu działania procedury zmienna zachowywała swoją wartość, powinieneś podczas deklaracji zmiennej użyć słowa kluczowego `Static` (więcej szczegółowych informacji na ten temat znajdziesz w podrozdziale „Zmienne statyczne” w dalszej części tego rozdziału).

Najczęstszy sposób deklarowania zmiennej lokalnej polega na umieszczeniu instrukcji `Dim` pomiędzy instrukcjami `Sub` i `End Sub`. Instrukcje `Dim` zazwyczaj są umieszczane bezpośrednio po instrukcji `Sub` i przed kodem źródłowym procedury.



Słowo kluczowe `Dim` jest skrótem od angielskiego słowa *Dimension* (rozmiar). W starszych wersjach języka BASIC polecenie `Dim` stosowane było wyłącznie do deklarowania rozmiarów tablicy. W języku VBA słowo kluczowe `Dim` służy do deklarowania dowolnej zmiennej, a nie tylko tablic.

Poniższa procedura używa sześciu lokalnych zmiennych zadeklarowanych przy użyciu instrukcji `Dim`:

```
Sub MySub()
    Dim x As Integer
    Dim First As Long
```

```

Dim InterestRate As Single
Dim TodaysDate As Date
Dim UserName As String
Dim MyValue
' - [W tym miejscu powinieneś umieścić kod procedury] -
End Sub

```

Zauważ, że ostatnia instrukcja Dim nie deklaruje typu danych, a jedynie nazwę zmiennej. W efekcie tak zadeklarowana zmienna staje się zmienną typu Variant.

Przy użyciu instrukcji Dim można też zadeklarować od razu kilka zmiennych. Na przykład:

```

Dim x As Integer, y As Integer, z As Integer
Dim First As Long, Last As Double

```



W przeciwieństwie do wielu innych języków, VBA nie pozwala na deklarowanie grupy zmiennych danego typu poprzez oddzielenie ich przecinkami. Przykładowo poniższa instrukcja, pomimo że jest poprawna, *nie* powoduje zadeklarowania wszystkich zmiennych jako zmiennych typu danych Integer:

```
Dim i, j, k As Integer
```

W języku VBA tylko zmienna k będzie typu Integer. Pozostałe zmienne zostaną zadeklarowane przy użyciu typu Variant. Aby zadeklarować zmienne i, j i k jako zmienne typu Integer, należy użyć następującej instrukcji:

```
Dim i As Integer, j As Integer, k As Integer
```

Zmienne dostępne w obszarze całego modułu

Jeżeli chcesz, aby zmienna była dostępna dla wszystkich procedur modułu, wystarczy deklarację zmiennej umieścić *przed* pierwszą procedurą modułu (na zewnątrz wszelkich procedur lub funkcji).

W poniższym przykładzie instrukcja Dim jest pierwszą instrukcją modułu, dzięki czemu zarówno procedura Procedure1, jak i procedura Procedure2 mają dostęp do zmiennej CurrentValue.

```

Dim CurrentValue as Integer
Sub Procedure1()
    ' [W tym miejscu znajduje się kod procedury]
End Sub
Sub Procedure2()
    ' [W tym miejscu znajduje się kod procedury]
End Sub

```

Wartość zmiennej dostępnej w całym module zostaje zachowana po normalnym zakończeniu działania procedury (tzn. kiedy zostanie wykonane polecenie End Sub lub End Function). Wyjątkiem jest sytuacja, gdy działanie procedury jest zatrzymywane przy użyciu instrukcji End. Gdy VBA napotka instrukcję End, wszystkie zmienne zadeklarowane w tym module tracą swoje wartości.

Inna metoda określania typu zmiennych

Podobnie jak większość odmian języka BASIC, także VBA umożliwia dodanie do nazwy zmiennej znaku identyfikującego typ danych. Na przykład poprzez dołączenie do nazwy zmiennej MyVar znaku % możesz zadeklarować ją jako zmienną typu Integer:

```
Dim MyVar%
```

Znaki deklarujące typ danych dostępne są dla większości typów danych języka VBA. Typy danych, które nie zostały wymienione na liście, nie posiadają znaku deklaracji typu danych.

Typ danych	Znak deklarujący typ danych
Integer	%
Long	&
Single	!
Double	#
Currency	@
String	\$

Taka metoda deklarowania typów danych właściwie jest pozostałością z języka BASIC, stąd oczywiście lepiej deklarować zmienne przy użyciu metod omówionych w tym rozdziale. Znaki deklarujące typ danych zostały wymienione tylko na wypadek, gdybyś spotkał się z nimi w starszych programach.

Jeżeli zmienna zostanie zadeklarowana jako lokalna, inne procedury zawarte w tym samym module będą mogły użyć zmiennej o identycznej nazwie, ale każdy jej egzemplarz będzie unikatowy w ramach procedury, w której go zdefiniowano.

Zazwyczaj zmienne lokalne są najefektywniejsze, ponieważ po zakończeniu działania procedury VBA zwalnia pamięć używaną przez zmienne.

Zmienne globalne

Aby udostępnić zmienną wszystkim procedurom zawartym we wszystkich modułach VBA projektu, powinieneś na poziomie modułu (przed pierwszą deklaracją procedury) zadeklarować zmienną przy użyciu słowa kluczowego Public (zamiast Dim). Oto przykład:

```
Public CurrentRate as Long
```

Słowo kluczowe Public powoduje, że zmienna CurrentRate jest dostępna dla wszystkich procedur projektu, nawet zawartych w innych modułach. Tego typu deklaracja musi pojawić się w standardowym module VBA, a nie w kodzie źródłowym modułu arkusza lub formularza *UserForm*.

Zmienne statyczne

Zmienne statyczne to szczególny przypadek zmiennych. Są deklarowane na poziomie procedury i utrzymują swoje wartości po zakończeniu jej wykonywania. Pamiętaj jednak, że jeżeli procedura zakończy działanie po wykonaniu polecenia End, zmienne statyczne również *utracą* swoje wartości.

Zmienne statyczne są deklarowane za pomocą słowa kluczowego Static:

```
Sub MySub()
    Static Counter as Integer
    ' [W tym miejscu znajduje się kod procedury]
End Sub
```

Zastosowanie stałych

Wartości zmiennych często się zmieniają w trakcie wykonywania procedury (stąd nazwa *zmienna...*). Czasami jednak odwołujemy się w kodzie programu do wartości lub łańcucha tekstu, który nie zmienia się przez cały czas działania programu — inaczej mówiąc, odwołujemy się w ten sposób do *stałej* (ang. *constant*).

Używanie w kodzie programu stałych zamiast wpisanych na sztywno wartości lub łańcuchów tekstu jest znakomitą praktyką programistyczną. Jeżeli na przykład procedura musi kilka razy odwołać się do określonej wartości, takiej jak stopa procentowa, lepiej zadeklarować wartość jako stałą i zamiast jej wartości w wyrażeniach używać nazwy. Dzięki temu nie tylko kod źródłowy będzie czytelniejszy, ale w razie potrzeby będzie również łatwiejszy do modyfikacji — zamiast kilku czy kilkudziesięciu wartości w różnych miejscach programu będziesz musiał zmienić tylko jedną wartość stałej.

Konwencje dotyczące nazw zmiennych

Niektórzy programiści nadają zmiennym nazwy w taki sposób, aby sama nazwa pozwalała na szybką identyfikację typu zmiennej. Osobiście niezbyt często posługuję się taką metodą, ponieważ uważam, że pogarsza to czytelność kodu źródłowego (ale oczywiście możesz się z tym nie zgodzić).

Konwencja nazewnictwa zmiennych wymaga zastosowania w nich standardowego przedrostka złożonego z małych liter. Jeżeli na przykład masz do czynienia ze zmienną typu Boolean określającą, czy skoroszyt został zapisany, możesz nadać jej nazwę bJestZapisany. Tym sposobem oczywiste będzie, że zmienna jest typu Boolean. W poniżej tabeli wymieniono kilka standardowych przedrostków typów danych.

Typ danych	Przedrostek
Boolean	b
Integer	i
Long	l
Single	s
Double	d
Currency	c
Date/Time	dt
String	str
Object	obj
Variant	v
Zdefiniowany przez użytkownika	u

Deklarowanie stałych

Stałe są deklarowane przy użyciu instrukcji Const. Oto kilka przykładów:

```
Const NumQuarters as Integer = 4
Const Rate = .0725, Period = 12
Const ModName as String = "Makra budżetowe"
Public Const AppName as String = "Aplikacja budżetowa"
```

W drugim przykładzie nie został zadeklarowany typ danych, stąd w konsekwencji VBA określa go na podstawie wartości. Typem danych zmiennej Rate jest Double, natomiast zmiennej Period — Integer. Ponieważ wartość stałej nigdy się nie zmienia, zwykle deklarujemy je przypisując z góry określony typ danych.

Stałe, podobnie jak zmienne, posiadają swój zasięg. Jeżeli stała ma być dostępna na obszarze pojedynczej procedury, to w celu przypisania jej roli stałej lokalnej należy zadeklarować ją po instrukcji Sub lub Function. Aby udostępnić stałą wszystkim procedurom modułu, należy zadeklarować ją przed pierwszą procedurą modułu. Aby udostępnić stałą wszystkim modułom skoroszytu, należy użyć słowa kluczowego Public i zadeklarować ją przed pierwszą procedurą modułu. Oto przykład:

```
Public Const InterestRate As Double = 0.0725
```



Jeżeli w procedurze VBA spróbujesz przypisać stałej inną wartość, na ekranie pojawi się następujący komunikat o błędzie: *Assignment to constant not permitted* (Przypisanie do stałej jest niedozwolone). W sumie tego chyba właściwie oczekiwaliśmy — w końcu stała to stała i jej wartość nie może się zmieniać.

Zastosowanie stałych predefiniowanych

Excel i język VBA oferują wiele predefiniowanych stałych, których można używać bez uprzedniego deklarowania. Co więcej, aby z nich korzystać, nie musisz nawet znać ich wartości. Rejestrator makr zazwyczaj zamiast wartości używa predefiniowanych stałych. Przykładowo, poniższa procedura w celu ustawienia poziomej orientacji strony dla aktywnego arkusza korzysta z wbudowanej stałej xlLandscape:

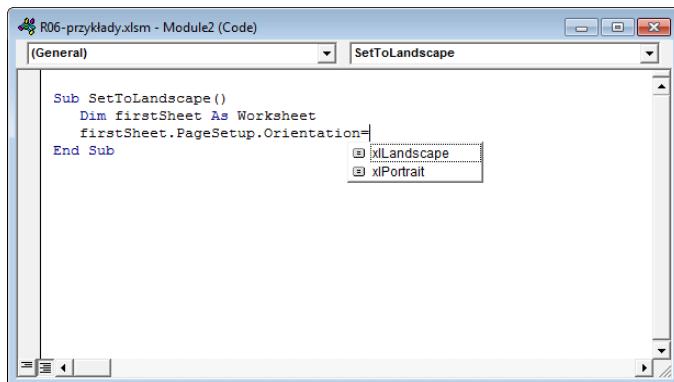
```
Sub SetToLandscape()
    ActiveSheet.PageSetup.Orientation = xlLandscape
End Sub
```

Stałą xlLandscape odkryłem w trakcie rejestracji makra. Poza tym informacje na jej temat znalazłem w systemie pomocy. Po włączeniu opcji *Auto List Members* przy wprowadzaniu kodu źródłowego często można uzyskać również informacje o stałych (patrz rysunek 6.2). W wielu przypadkach VBA wyświetla listę wszystkich stałych, które mogą być przypisane do danej właściwości.

Rzeczywista wartość reprezentowana przez stałą xlLandscape to 2 (o czym możesz się przekonać przy użyciu okna *Immediate*). Inną wbudowaną stałą, zmieniającą orientację strony, jest stała xlPortrait, mająca wartość 1. Oczywiście jeżeli zastosujesz wbudowane stałe, nie będziesz musiał znać ani pamiętać ich wartości.

Rysunek 6.2.

VBA wyświetla listę wszystkich stałych, które mogą być przypisane do właściwości



Za pomocą przeglądarki *Object Browser*, którą pokrótkę omówiliśmy w rozdziale 5., możesz wyświetlić listę wszystkich stałych Excela i języka VBA. Aby otworzyć okno przeglądarki obiektów, przejdź do edytora Visual Basic i naciśnij klawisz F2.

Praca z łańcuchami tekstu

Język VBA, podobnie jak Excel, umożliwia manipulowanie zarówno liczbami, jak i tekstem (łańcuchami tekstu). W przypadku języka VBA można wyróżnić dwa typy łańcuchów tekstu:

- *Łańcuchy o stałej długości*, deklarowane przy użyciu ściśle określonej liczby znaków (maksymalna długość takiego łańcucha tekstu wynosi 65 535 znaków).
- *Łańcuchy o zmiennej długości*, które teoretycznie mogą przechowywać do 2 miliardów znaków.

Każdy znak łańcucha wymaga 1 bajta pamięci. Dodatkowa niewielka przestrzeń potrzebna jest do przechowywania nagłówka każdego łańcucha tekstu. Podczas deklarowania zmiennej łańcuchowej przy użyciu instrukcji Dim możesz zdefiniować długość łańcucha (jeżeli ją znasz i tworzysz łańcuch o stałej długości) lub możesz pozwolić, aby VBA tworzył łańcuch tekstu dynamicznie (łańcuch o zmiennej długości).

W kolejnym przykładzie zmiennej MyString przypisano typ String i maksymalną długość wynoszącą 50 znaków. Typ danych String jest też definiowany dla zmiennej YourString, ale długość łańcucha nie jest stała:

```
Dim MyString As String * 50
Dim YourString As String
```

Przetwarzanie dat

Oczywiście do przechowywania dat możesz użyć zmiennej łańcuchowej, ale w takim przypadku nie będziesz mógł wykonywać na nich obliczeń. Do przetwarzania dat lepiej zastosować typ danych Date.

Błąd Excela związany z datą

Powszechnie wiadomo, że w Excelu jest pewien błąd związany z przetwarzaniem dat. Aplikacja niepoprawnie zakłada, że rok 1900 jest przestępny. Mimo że w roku 1900 nie było daty 29 lutego, Excel akceptuje poniższą formułę i jako wynik wyświetla datę 29 lutego 1900 roku:

```
=DATA(1900;2;29)
```

W języku VBA nie ma już tego błędu. Odpowiednikiem funkcji DATA programu Excel jest w języku VBA funkcja DateSerial. Poniższe wyrażenie poprawnie zwraca datę 1 marca 1900 roku:

```
DateSerial(1900, 2, 29)
```

Wynika stąd jasno, że system numerów seryjnych związany z datami nie jest taki sam w Excelu i języku VBA. Oba systemy zwracają różne wartości dla dat z przedziału od 1 stycznia do 28 lutego 1900 roku.

Zmienna, dla której zdefiniowano typ Date, zajmuje 8 bajtów pamięci i jest w stanie przechowywać daty z przedziału od 1 stycznia 100 roku do 31 grudnia 9999 roku. Jest to prawie 10 000 lat, czyli okres w zupełności wystarczający nawet w przypadku najbardziej agresywnych prognoz finansowych! Typ danych Date nadaje się też do przechowywania danych związanych z czasem. W języku VBA daty i czas są umieszczane pomiędzy dwoma znakami #.



Uwaga

Przedział dat obsługiwany przez język VBA jest znacznie większy niż w przypadku samego Excela (gdzie rozpoczyna się od 1 stycznia 1900 roku) i rozciąga się aż do 31 grudnia 9999 roku. Aby uniknąć przykrych niespodzianek, powinieneś zachować ostrożność i nie umieszczać w arkuszu dat wykraczających poza przedział dat obsługiwany przez program Excel.



W rozdziale 8. omówimy kilka relatywnie prostych funkcji VBA, które pozwalają na tworzenie formuł operujących na datach wcześniejszych niż 1 stycznia 1900 roku.

Oto kilka przykładów deklarowania zmiennych i stałych typu Date:

```
Dim Today As Date  
Dim StartTime As Date  
Const FirstDay As Date = #1/1/2013#  
Const Noon = #12:00:00#
```



Daty w języku VBA zawsze są definiowane przy użyciu formatu miesiąc/dzień/rok, nawet jeśli w systemie są wyświetlane w innym formacie (np. dzień/miesiąc/rok).

Jeżeli do wyświetlania daty użyjesz okna komunikatu, data zostanie pokazana zgodnie z systemowym formatem krótkiej daty. Podobnie czas jest wyświetlany z uwzględnieniem systemowego formatu czasu (12- lub 24-godzinny). Aby zmienić systemowe ustawienia wyświetlania daty, powinieneś skorzystać z opcji Zegar, język i region w Panelu sterowania systemu Windows.

Instrukcje przypisania

Instrukcja przypisania jest instrukcją języka VBA wykonującą obliczenia matematyczne i przypisującą wynik zmiennej lub obiektowi. System pomocy Excela definiuje *wyrażenie* jako „kombinację słów kluczowych, operatorów, zmiennych i stałych zwracających łańcuch, liczbę lub obiekt. Wyrażenie takie może wykonywać obliczenia, przetwarzać znaki lub testować dane”.

Nie mógłbym tego lepiej zdefiniować. Większość prac programistycznych opartych na języku VBA jest związana z tworzeniem wyrażeń (oraz wykrywaniem i usuwaniem w nich błędów). Jeżeli umiesz tworzyć formuły w Excelu, bez kłopotu zdefiniujesz wyrażenia w języku VBA. W przypadku formuły arkuszowej Excel wyświetla wynik w komórce. Z kolei wyrażenie języka VBA może zostać przypisane zmiennej lub użyte jako wartość właściwości.

W języku VBA funkcję operatora przypisania spełnia znak równości (=). Poniżej podano kilka przykładowych instrukcji przypisania (wyrażenia znajdują się po prawej stronie znaku równości):

```
x = 1  
x = x + 1  
x = (y * 2) / (z * 2)  
FileOpen = True  
FileOpen = Not FileOpen  
Range("Rok").Value = 2010
```



Wyrażenia mogą być bardzo złożone. Aby zwiększyć czytelność długich wyrażeń, można zastosować sekwencję kontynuacji, przenoszącą je do następnego wiersza (jak pamiętasz, taka sekwencja składa się ze spacji i znaku podkreślenia).

Bardzo często wyrażenia używają funkcji. Mogą to być wbudowane funkcje języka VBA, funkcje arkusza Excel lub niestandardowe funkcje języka VBA. Wbudowane funkcje języka VBA zostaną omówione w dalszej części rozdziału, w podrozdziale „Wbudowane funkcje VBA”.

W języku VBA kluczową rolę odgrywają operatory. Dobrze znane operatory opisują matematyczne operacje, w tym dodawanie (+), mnożenie (*), dzielenie (/), odejmowanie (-), potęgowanie (^) i łączenie łańcuchów (&). Do mniej znanych operatorów należy znak \ (stosowany przy dzieleniu liczb całkowitych) i operator Mod (używany w arytmetyce modulo). Operator Mod zwraca resztę z dzielenia jednej liczby przez drugą. Przykładowo następujące wyrażenie zwraca wartość 2:

```
17 Mod 3
```

Język VBA obsługuje też takie same operatory porównania, jak formuły Excela. Są to operator równości (=), większy niż (>), mniejszy niż (<), większy lub równy (>=), mniejszy lub równy (<=) i nierówności (<>).

Z jednym wyjątkiem, kolejność stosowania operatorów w języku VBA jest dokładnie taka sama, jak w Excelu (patrz tabela 6.3). Oczywiście w celu zmiany naturalnej kolejności wykonywania działań możesz zastosować nawiasy.

Tabela 6.3. Kolejność operatorów w języku VBA

Operator	Działanie	Kolejność wykonywania
$^$	Potęgowanie	1
$* \text{ i } /$	Mnożenie i dzielenie	2
$+ \text{ i } -$	Dodawanie i odejmowanie	3
$\&$	Konkatenacja	4
$=, <, >, <=, >=, <>$	Porównywanie	5



Operator negacji (znak minus) jest inaczej traktowany w języku VBA. W Excelu poniższa formula zwróci wartość 25:

$$=-5^2$$

W języku VBA, po obliczeniu tego wyrażenia zmienna x będzie miała wartość -25:

$$x = -5 ^ 2$$

VBA najpierw wykonuje operację potęgowania, a następnie wykonuje negację.

Abytrzymać wartość 25 powinieneś użyć następującego wyrażenia:

$$x = (-5) ^ 2$$

Na przykładzie przedstawionym poniżej zmiennej x przypisywana jest wartość 10, ponieważ operator mnożenia ma wyższy priorytet niż operator dodawania.

$$x = 4 + 3 * 2$$

Aby uniknąć dwuznaczności, możesz zapisać powyższe wyrażenie w następujący sposób:

$$x = 4 + (3 * 2)$$

Oprócz operatorów matematycznych język VBA oferuje pełny zestaw operatorów logicznych, przedstawiony w tabeli 6.4. Aby uzyskać szczegółowe informacje o tych operatorach, skorzystaj z systemu pomocy języka VBA.

Tabela 6.4. Operatory logiczne języka VBA

Operator	Przeznaczenie
Not	Negacja logiczna wyrażenia
And	Koniunkcja logiczna dwóch wyrażeń
Or	Suma logiczna dwóch wyrażeń
Xor	Nierównoważność logiczna dwóch wyrażeń
Eqv	Równoważność logiczna dwóch wyrażeń
Imp	Implikacja logiczna dwóch wyrażeń

Poniższa instrukcja używa operatora Not w celu wyświetlania i ukrywania linii siatki w aktywnym oknie. Właściwość DisplayGridlines pobiera wartość True lub False, stąd zatem zastosowanie operatora Not spowoduje zmianę wartości z False na True lub z True na False.

```
ActiveWindow.DisplayGridlines = Not ActiveWindow.DisplayGridlines
```

Kolejne wyrażenie wykonuje operację używającą logicznego operatora And. Instrukcja MsgBox wyświetli wartość True tylko wtedy, gdy arkusz Arkusz1 jest aktywny i aktywna komórka znajduje się w wierszu 1. Jeżeli jeden z dwóch warunków nie zostanie spełniony, instrukcja MsgBox wyświetli wartość False.

```
MsgBox ActiveSheet.Name = "Arkusz1" And ActiveCell.Row = 1
```

Poniższe wyrażenie wykonuje operację z logicznym operatorem Or. Instrukcja MsgBox wyświetli wartość True tylko wtedy, gdy aktywnym arkuszem będzie arkusz Arkusz1 lub Arkusz2.

```
MsgBox ActiveSheet.Name = "Arkusz1" Or ActiveSheet.Name = "Arkusz2"
```

Tablice

Tablica jest grupą elementów tego samego typu, posiadających wspólną nazwę. W celu odwołania się do określonego elementu tablicy należy użyć jej nazwy i numeru indeksu. Przykładowo można zdefiniować tablicę złożoną z 12 zmiennych łańcuchowych, z których każda odpowiada nazwie miesiąca. Jeżeli tablica zostanie nazwana MonthNames, to w celu odwołania się do jej pierwszego elementu należy użyć instrukcji MonthNames(0), do drugiego elementu MonthNames(1) i tak dalej aż do instrukcji MonthNames(11).

Deklarowanie tablic

Podobnie jak zwykła zmienna, tak i tablica może zostać zadeklarowana przy użyciu instrukcji Dim lub Public. Istnieje też możliwość określenia liczby elementów tablicy. W tym celu w nawiasie okrągłym należy podać numer pierwszego indeksu, słowo kluczowe To i numer ostatniego indeksu. Oto przykład demonstrujący, w jaki sposób zadeklarować tablicę złożoną dokładnie ze 100 liczb całkowitych:

```
Dim MyArray(1 To 100) As Integer
```



Przy deklarowaniu tablicy konieczne jest podanie jedynie górnego indeksu. VBA przyjmuje domyślnie, że dolny indeks ma wartość 0. A zatem efekt działania dwóch poniższych instrukcji jest taki sam:

```
Dim MyArray(0 To 100) As Integer  
Dim MyArray(100) As Integer
```

W obu przypadkach tablica składa się ze 101 elementów.

VBA domyślnie zakłada numerację elementów tablicy od 0. Jeżeli chciałbyś, aby w przypadku wszystkich tablic, w których deklarowany jest tylko górny indeks, VBA stosował dolny indeks o wartości 1, przed pierwszą procedurą modułu należy wstawić następującą instrukcję:

```
Option Base 1
```

Deklarowanie tablic wielowymiarowych

Przykłady z poprzedniego podrozdziału demonstrowały tablice jednowymiarowe. Tablice języka VBA mogą mieć maksymalnie 60 wymiarów, ale rzadko stosuje się więcej niż 3 wymiary (tablice trójwymiarowe). Poniższa instrukcja deklaruje dwuwymiarową tablicę złożoną ze 100 liczb całkowitych:

```
Dim MyArray(1 To 10, 1 To 10) As Integer
```

Taką tablicę można traktować jak macierz 10×10 . Aby odwołać się do określonego elementu tablicy dwuwymiarowej, konieczne jest podanie dwóch numerów indeksu. Poniżej pokazano, w jaki sposób uzyskać dostęp do wartości zawartej w powyższej tablicy:

```
MyArray(3, 4) = 125
```

Poniżej przedstawiono deklarację tablicy trójwymiarowej, która zawierać będzie 1000 elementów (możesz sobie wyobrazić taką tablicę jako sześciąn).

```
Dim MyArray(1 To 10, 1 To 10, 1 To 10) As Integer
```

Aby odwołać się do wybranego elementu takiej tablicy, musisz podać wartości trzech indeksów tablicy, na przykład:

```
MyArray(4, 8, 2) = 0
```

Deklarowanie tablic dynamicznych

Tablica dynamiczna nie posiada z góry określonej liczby elementów. Jest deklarowana przy użyciu pary pustych nawiasów okrągłych:

```
Dim MyArray() As Integer
```

Zanim jednak użyjesz w kodzie programu tablicy dynamicznej konieczne jest użycie instrukcji `ReDim`, która informuje VBA o liczbie elementów tablicy. Często taką operację wykonuje się przy użyciu zmiennej, której wartość nie jest znana aż do momentu wykonania procedury. Na przykład: jeżeli zmienna `x` zawiera liczbę, możesz zdefiniować rozmiar tablicy w następujący sposób:

```
ReDim MyArray(1 To x)
```

Polecenia `ReDim` możesz użyć dowolną liczbę razy, zmieniając rozmiar tablicy tak często, jak to będzie potrzebne. Kiedy zmieniasz rozmiar tablicy, jej zawartość jest usuwana. Jeżeli chcesz zachować istniejące wartości, powinieneś użyć polecenia `ReDim Preserve`. Na przykład:

```
ReDim Preserve MyArray(1 to y)
```

Tablice pojawią się jeszcze w dalszej części rozdziału przy okazji omawiania pętli (patrz podrozdział „Wykonywanie bloku instrukcji w ramach pętli”).

Zmienne obiektowe

Zmienna obiektowa jest zmienną reprezentującą cały obiekt, na przykład zakres lub arkusz. Zmienne obiektowe są ważne z dwóch powodów:

- W znaczący sposób mogą uprościć kod źródłowy.
- Mogą zwiększyć szybkość wykonywania kodu źródłowego.

Zmienne obiektowe, podobnie jak zwykłe zmienne, są deklarowane przy użyciu instrukcji `Dim` lub `Public`. Na przykład poniższe polecenie deklaruje zmienną `InputArea` jako obiekt klasy `Range`:

```
Dim InputArea as Range
```

Aby przypisać obiekt do zmiennej, powinieneś użyć słowa kluczowego `Set`, na przykład:

```
Set InputArea = Range("C16:E16")
```

Aby przekonać się, w jaki sposób zmienne obiektowe mogą uprościć kod źródłowy, najpierw zapoznaj się z poniższą procedurą, utworzoną bez używania zmiennych obiektowych:

```
Sub NoObjVar()
    Worksheets("Arkusz1").Range("A1").Value = 124
    Worksheets("Arkusz1").Range("A1").Font.Bold = True
    Worksheets("Arkusz1").Range("A1").Font.Italic = True
    Worksheets("Arkusz1").Range("A1").Font.Size = 14
    Worksheets("Arkusz1").Range("A1").Font.Name = "Cambria"
End Sub
```

Powyższa procedura wpisuje wartość 124 do komórki A1 arkusza Arkusz1 aktywnego skoroszytu, a następnie formatuje zawartość komórki i zmienia krój oraz rozmiar czcionki. Jak widać, całkiem sporo klikania po klawiaturze. Aby oszczędzić biednym palcom klikania (a przy okazji zwiększyć efektywność programu), możesz nieco skondensować kod używając odpowiedniej zmiennej obiektowej:

```
Sub ObjVar()
    Dim MyCell As Range
    Set MyCell = Worksheets("Arkusz1").Range("A1")
    MyCell.Value = 124
    MyCell.Font.Bold = True
    MyCell.Font.Italic = True
    MyCell.Font.Size = 14
    MyCell.Font.Name = Cambria
End Sub
```



Przy przypisaniu obiektu do zmiennej VBA może znacznie łatwiej z niego korzystać niż w przypadku normalnego odwołania o dłuższej postaci, które musi zostać przeanalizowane. Jeżeli szybkość działania aplikacji jest czynnikiem bardzo istotnym, powinieneś korzystać ze zmiennych obiektowych. Jednym z elementów mających wpływ na szybkość jest *przetwarzanie znaku kropki*. Każdorazowo, gdy VBA napotka znak kropki, tak jak w przypadku odwołania `Sheets(1).Range("A1")`, traci trochę czasu na przeprowadzenie analizy odwołania. Zastosowanie zmiennej obiektowej pozwala zredukować liczbę kropek. Im mniej kropek, tym krótszy czas przetwarzania. Kolejna metoda zwiększenia szybkości wykonywania kodu źródłowego polega na zastosowaniu konstrukcji `With ... End With`, która także zmniejsza liczbę przetwarzanych kropek. Konstrukcja zostanie omówiona w dalszej części rozdziału.

Po zadeklarowaniu zmiennej MyCell jako obiektu Range instrukcja Set przypisuje do niej obiekt. Zamiast dłuższego odwołania o postaci Worksheets("Arkusz1").Range("A1") w kolejnych instrukcjach będzie można zastosować jej prostszą wersję, czyli MyCell.

Przydatność zmiennych obiektowych stanie się widoczna przy okazji omawiania pętli w dalszej części tego rozdziału.

Typy danych definiowane przez użytkownika

Język VBA umożliwia tworzenie niestandardowych typów danych lub inaczej — *typów danych definiowanych przez użytkownika*. Typ danych definiowany przez użytkownika może ułatwić pracę z niektórymi rodzajami danych. Jeżeli na przykład aplikacja przetwarza dane na temat klientów, można stworzyć niestandardowy typ danych o nazwie CustomerInfo o następującej postaci:

```
Type CustomerInfo  
    Company As String  
    Company As String  
    RegionCode As Long  
    Sales As Double  
End Type
```



Niestandardowe typy danych muszą być definiowane na początku modułu przed kodem pierwszej procedury.

Po utworzeniu niestandardowego typu danych w celu zadeklarowania zmiennej tego typu powinieneś użyć instrukcji Dim. Zazwyczaj w taki sposób definiowane są tablice. Oto przykład:

```
Dim Customers(1 To 100) As CustomerInfo
```

Każdy ze 100 elementów tablicy składa się z czterech komponentów (określonych w niestandardowym typie danych CustomerInfo). Do określonego komponentu rekordu można się odwołać w następujący sposób:

```
Customers(1).Company = "Narzędzia Acme"  
Customers(1).Contact = "Jan Nowak"  
Customers(1).RegionCode = 3  
Customers(1).Sales = 150674.98
```

Można też przetwarzać każdy element tablicy jako całość. Przykładowo w celu skopiowania informacji z Customers(1) do Customers(2) należy użyć następującego polecenia:

```
Customers(2) = Customers(1)
```

Powyższy przykład jest równoważny następującemu blokowi instrukcji:

```
Customers(2).Company = Customers(1).Company  
Customers(2).Contact = Customers(1).Contact  
Customers(2).RegionCode = Customers(1).RegionCode  
Customers(2).Sales = Customers(1).Sales
```

Wbudowane funkcje VBA

Język VBA, podobnie jak większość języków programowania, posiada szereg wbudowanych funkcji upraszczających obliczenia i wykonywanie wielu operacji. Wiele funkcji języka VBA jest podobnych do funkcji arkuszowych Excela (a czasami są one identyczne). Na przykład funkcja Ucase() języka VBA, która zamienia w łańcuchu tekstu małe litery na wielkie jest odpowiednikiem funkcji LITERY.WIELKIE() programu Excel.



W dodatku A zamieszczono kompletną listę funkcji języka VBA wraz z krótkim opisem każdej z nich. Dokładny opis funkcji znajdziesz w systemie pomocy języka VBA.



Aby w czasie wprowadzania kodu programu wyświetlić listę funkcji VBA, powinieneś wpisać słowo kluczowe VBA, a za nim kropkę. Edytor Visual Basic wyświetli listę wszystkich dostępnych elementów, w tym i funkcje (patrz rysunek 6.3). Obok funkcji jest widoczna zielona ikona. Jeżeli lista się nie pojawia, powinieneś sprawdzić, czy opcja *Auto List Members* jest włączona. Aby to zrobić, z menu *Tools* wybierz polecenie *Options* i naciśnij klawisz *Enter*.

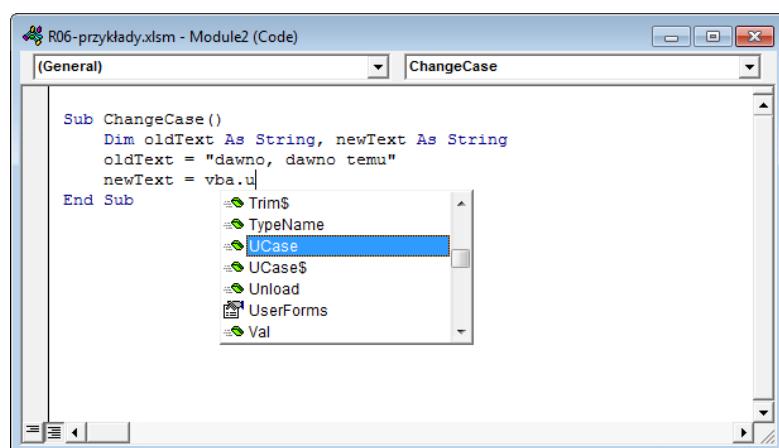
Funkcje języka VBA są stosowane w wyrażeniach w bardzo podobny sposób, jak funkcje formuł arkuszowych. Oto przykład prostej procedury, która oblicza pierwiastek kwadratowy dla zmiennej (przy użyciu funkcji Sqr języka VBA), zapisuje wynik w innej zmiennej i wreszcie wyświetla wynik obliczeń na ekranie:

```
Sub ShowRoot()
    Dim MyValue As Double
    Dim SquareRoot as Double
    MyValue = 25
    SquareRoot = Sqr(MyValue)
    MsgBox SquareRoot
End Sub
```

Funkcja Sqr języka VBA jest odpowiednikiem arkuszowej funkcji PIERWIATEK.

Rysunek 6.3.

Wyświetlanie listy funkcji języka VBA w oknie edytora Visual Basic



W instrukcjach języka VBA można używać wielu, ale nie wszystkich funkcji arkusza Excela. Obiekt WorksheetFunction zawarty w obiekcie Application przechowuje wszystkie funkcje arkusza, które można wywołać z procedur języka VBA.

Aby w instrukcji języka VBA użyć funkcji arkusza, wystarczy poprzedzić jej nazwę następującym odwołaniem:

```
Application.WorksheetFunction
```

Przykład przedstawiony poniżej pokazuje, w jaki sposób w procedurze języka VBA zastosować funkcję arkuszową Excela. Rzadko używana funkcja arkuszowa ROMAN zamienia liczbę arabską na rzymską:

```
Sub ShowRoman()
    Dim DecValue As Long
    Dim RomanValue As String
    DecValue = 1939
    RomanValue = Application.WorksheetFunction.Roman(DecValue)
    MsgBox RomanValue
End Sub
```

Po wykonaniu powyższej procedury funkcja MsgBox wyświetli ciąg znaków MCMXXXIX.

Swoją drogą, miłośnicy starych filmów często wpadali w przerażenie, gdy dowiadywali się, że Excel niestety nie posiada funkcji zamieniającej liczby rzymskie na ich odpowiedniki arabskie. Formuła przedstawiona poniżej wyświetla wartość 1939:

```
=ARABSKIE("MCMXXXIX")
```

Pamiętaj jednak, że nie możesz używać funkcji arkuszowych, które mają swoje odpowiedniki w języku VBA. Przykładowo w kodzie VBA nie możesz zastosować funkcji arkuszowej PIERWIATEK Excela, ponieważ język VBA posiada jej własną wersję o nazwie Sqr, stąd próba wykonania polecenia przedstawionego poniżej spowoduje wygenerowanie błędu:

```
MsgBox Application.WorksheetFunction.Sqrt(123) ' błąd
```



Język VBA umożliwia tworzenie niestandardowych funkcji działających podobnie jak wbudowane funkcje arkuszowe Excela. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 8.

Praca z obiektami i kolekcjami

Jako programista używający Excela mnóstwo czasu będziesz spędzał pracując z obiektami i kolekcjami. Z tego prostego powodu z pewnością warto poznać najwydajniejsze metody pisania kodu programów wykorzystujących obiekty i kolekcje. Język VBA oferuje dwie ważne konstrukcje upraszczające pracę z obiektami i kolekcjami:

- Polecenie With ... End With.
- Polecenie For Each ... Next.

Funkcja MsgBox

Funkcja MsgBox jest jedną z najprzydatniejszych funkcji języka VBA. W wielu przykładach zamieszczonych w tym rozdziale używamy jej do wyświetlania wartości zmiennej.

Funkcja ta często jest dobrym odpowiednikiem prostego, niestandardowego okna dialogowego, a poza tym świetnie sprawdza się przy wykrywaniu i usuwaniu błędów, ponieważ możesz ją wstawić w niemal dowolnym miejscu kodu w celu wstrzymania działania programu i wyświetlenia wyniku obliczenia lub instrukcji przypisania.

Większość funkcji zwraca jedną wartość, która jest przypisywana do zmiennej. Funkcja MsgBox nie tylko zwraca wartość, ale też wyświetla okno dialogowe, dzięki któremu użytkownik może odpowiednio zareagować. Wartość zwracana przez funkcję MsgBox reprezentuje akcję podjętą przez użytkownika w wyświetlnym oknie dialogowym. Funkcja MsgBox może zostać użyta nawet wtedy, gdy nie zależy Ci na odpowiedzi użytkownika, ale na możliwości wyświetlenia komunikatu.

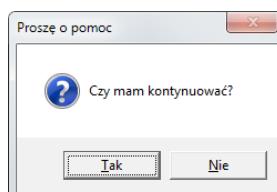
Oficjalna składnia funkcji MsgBox zawiera pięć argumentów (te w nawiasach kwadratowych są opcjonalne):

```
MsgBox(komunikat[, przyciski][, tytuł][, plik_pomocy][, kontekst])
```

- **komunikat** (wymagany): komunikat wyświetlany w oknie dialogowym;
- **przyciski** (opcjonalny): wartość określająca, jakie przyciski i ikony (jeżeli są potrzebne) pojawią się w oknie komunikatu (do definiowania przycisków oraz ikon powinieneś użyć wbudowanych stałych, np. vbYesNo);
- **tytuł** (opcjonalny): tekst pojawiający się na pasku tytułu okna komunikatu — domyślnym tytułem jest łańcuch tekstu *Microsoft Excel*;
- **plik_pomocy** (opcjonalny): nazwa pliku pomocy powiązanego z oknem komunikatu;
- **kontekst** (opcjonalny): identyfikator kontekstu tematu pomocy, który zostanie wyświetlony. Jeżeli użyjesz argumentu **kontekst**, musisz również użyć argumentu **plik_pomocy**.

Zwróconą wartość można przypisać do zmiennej, ale możesz również użyć samej funkcji, bez przypisania zwracanej wartości. W przykładzie przedstawionym poniżej wartość zwrócona przez funkcję jest przypisywana zmiennej Ans:

```
Dim Ans As Long  
Ans = MsgBox ("Czy mam kontynuować?", vbYesNo + vbQuestion, "Proszę o pomoc")  
If Ans = vbNo Then Exit Sub
```



Zauważ, że dla argumentu **przyciski** użyliśmy sumy dwóch wbudowanych stałych (**vbYesNo + vbQuestion**). Zastosowanie stałej **vbYesNo** spowoduje wyświetlenie dwóch przycisków o nazwach **Tak** i **Nie**. Dodanie stałej **vbQuestion** spowoduje wyświetlenie ikony pytajnika. Okno dialogowe będące wynikiem działania funkcji przedstawione na rysunku poniżej. Po wykonaniu pierwszej instrukcji zmienność Ans będzie zawierać jedną z dwóch wartości reprezentowanych przez stałe **vbYes** i **vbNo**. W przypadku, gdy użytkownik naciśnie przycisk **Nie**, program zakończy działanie.

Więcej szczegółowych informacji na temat funkcji MsgBox znajdziesz w rozdziale 10.

Konstrukcja With ... End With

Konstrukcja oparta na poleceniach With i End With umożliwia wykonywanie wielu operacji na pojedynczym obiekcie. Aby zrozumieć zasady działania konstrukcji With ... End With, na początek przyjrzyjmy się procedurze modyfikującej wartość pięciu właściwości związanych z formatowaniem zaznaczonego obszaru (reprezentowanego przez obiekt klasy Range):

```
Sub ChangeFont1()
    Selection.Font.Name = "Cambria"
    Selection.Font.Bold = True
    Selection.Font.Italic = True
    Selection.Font.Size = 12
    Selection.Font.Underline = xlUnderlineStyleSingle
    Selection.Font.ThemeColor = xlThemeColorAccent1
End Sub
```

Taką procedurę możemy zapisać w inny sposób, przy użyciu konstrukcji With ... End With, a jej działanie pozostanie niezmienione:

```
Sub ChangeFont2()
    With Selection.Font
        .Name = "Cambria"
        .Bold = True
        .Italic = True
        .Size = 12
        .Underline = xlUnderlineStyleSingle
        .ThemeColor = xlThemeColorAccent1
    End With
End Sub
```

Niektórzy programiści uważają, że druga wersja procedury jest mniej czytelna. Należy jednak pamiętać, że nadzrodnym celem wprowadzenia modyfikacji było zwiększenie szybkości. Procedura z konstrukcją With ... End With znacznie szybciej modyfikuje wartości kilku właściwości obiektu w porównaniu z procedurą, która w każdej instrukcji jawnie odwołuje się do obiektu.



W przypadku rejestrowania makra VBA, Excel używa konstrukcji With ... End With zawsze, gdy jest to możliwe. Aby otrzymać dobry przykład zastosowania takiej konstrukcji, spróbuj zarejestrować nowe makro podczas zmiany ustawień strony (przejdz na kartę UKŁAD STRONY i naciśnij przycisk Orientacja znajdujący się w grupie opcji Ustawienia strony).

Konstrukcja For Each ... Next

Wiesz już, że *kolekcja* to grupa powiązanych z sobą obiektów. Na przykład Workbooks jest kolekcją wszystkich otwartych obiektów Workbook, a w języku VBA możesz korzystać z wielu innych kolekcji.

Załóżmy, że chcesz wykonać określona operację na wszystkich obiektach z danej kolekcji. Albo przypuśćmy, że chcesz sprawdzić wszystkie obiekty kolekcji i wykonać określona operację tylko na obiektach spełniających wybrane kryterium. Takie zadania są wprost wymarzone do zastosowania konstrukcji For Each ... Next, ponieważ używając jej, nie musisz wiedzieć, ile elementów znajduje się w kolekcji.

Składnia konstrukcji For Each ... Next jest następująca:

```
For Each element In grupa
    [instrukcje]
    [Exit For]
    [instrukcje]
Next [element]
```

Poniższa procedura używa konstrukcji For Each ... Next do przetwarzania kolekcji Worksheets w aktywnym skoroszycie. Po uruchomieniu procedury funkcja MsgBox wyświetla wartość właściwości Name poszczególnych arkuszy (jeżeli w skoroszycie znajduje się na przykład pięć arkuszy, funkcja MsgBox zostanie wywołana pięć razy).

```
Sub CountSheets()
    Dim Item as Worksheet
    For Each Item In ActiveWorkbook.WorkSheets
        MsgBox Item.Name
    Next Item
End Sub
```



W powyższym przykładzie Item jest zmienną obiektową (a dokładniej mówiąc — obiektem klasy Worksheet). W nazwie Item nie ma nic nadzwyczajnego — zamiast niej możesz użyć dowolnej innej, poprawnej nazwy zmiennej.

W następnym przykładzie konstrukcja For Each ... Next została użyta do przetwarzania wszystkich obiektów kolekcji Windows i zliczania okien, które są ukryte.

```
Sub HiddenWindows()
    Dim Cnt As Integer
    Dim Win As Window
    Cnt = 0
    For Each Win In Windows
        If Not Win.Visible Then Cnt = Cnt + 1
    Next Win
    MsgBox Cnt & " ukrytych okien."
End Sub
```

Dla każdego okna, które jest ukryte, wartość zmiennej Cnt jest inkrementowana. Kiedy pętla kończy działanie, w oknie dialogowym wyświetlana jest wartość zmiennej Cnt.

Poniżej zamieszczono przykład procedury, która zamknie wszystkie skoroszyty z wyjątkiem aktywnego. Procedura do sprawdzenia poszczególnych skoroszytów w kolekcji Workbooks używa konstrukcji If ... Then:

```
Sub CloseInActive()
    Dim Book as Workbook
    For Each Book In Workbooks
        If Book.Name <> ActiveWorkbook.Name Then Book.Close
    Next Book
End Sub
```

Typowym zastosowaniem pętli For Each ... Next jest przetwarzanie wszystkich komórek z danego zakresu. Procedura przedstawiona poniżej powinna być wywoływana po tym, jak użytkownik zaznaczy wybrany zakres komórek. W tym przypadku obiekt Selection spełnia rolę kolekcji obiektów Range, ponieważ każda komórka w zaznaczonym obszarze

jest obiektem klasy Range. Procedura sprawdza każdą komórkę i przy użyciu funkcji UCase języka VBA zamienia małe litery na wielkie (działanie procedury nie ma żadnego wpływu na komórki zawierające wartości liczbowe).

```
Sub MakeUpperCase()
    Dim Cell as Range
    For Each Cell In Selection
        Cell.Value = UCase(Cell.Value)
    Next Cell
End Sub
```

VBA posiada mechanizm pozwalający na zakończenie działania pętli For Each ... Next przed zakończeniem przetwarzania wszystkich elementów kolekcji. Możesz tego dokonać przy użyciu polecenia Exit For, co zostało zilustrowane poniżej. Nasza kolejna procedura przechodzi przez kolejne komórki pierwszego wiersza aktywnego arkusza i zaznacza pierwszą komórkę, w której znajduje się wartość ujemna.

```
Sub SelectNegative()
    Dim Cell As Range
    For Each Cell In Range("1:1")
        If Cell.Value < 0 Then
            Cell.Select
            Exit For
        End If
    Next Cell
End Sub
```

Powyższa procedura do sprawdzania wartości kolejnych komórek wykorzystuje konstrukcję If ... Then. Kiedy pętla natrafi na pierwszą wartość ujemną, komórka jest zaznaczana i pętla kończy działanie po wykonaniu polecenia Exit For.

Sterowanie sposobem wykonywania procedur

W niektórych procedurach instrukcje są przetwarzane po kolei — od pierwszej do ostatniej. Na przykład rejestrowane makra zawsze działają w ten sposób. Jednak często konieczne jest sterowanie wykonywaniem procedur poprzez pominięcie kilku instrukcji, wielokrotne przetworzenie niektórych z nich i przetestowanie warunków określających, jaką czynność procedura zrealizuje jako następną.

W poprzednim podpunkcie omówiono konstrukcję For Each ... Next, będącą odmianą pętli. W tym punkcie zostaną omówione dodatkowe metody sterowania wykonywaniem procedur języka VBA. Oto one:

- polecenie GoTo
- konstrukcja If ... Then
- konstrukcja Select Case
- pętla For ... Next
- pętla Do While
- pętla Do Until

Polecenie GoTo

Najprostsza metoda sterowania programem polega na zastosowaniu polecenia GoTo, przynoszącego sterowanie programu do nowego polecenia, przed którym musi znajdować się odpowiednia etykieta (łańcuch tekstu z dwukropkiem lub liczba bez dwukropka). Procedury języka VBA mogą zawierać dowolną liczbę etykiet. Polecenie GoTo nie może przekazywać wykonywania programu poza obszar procedury.

Procedura przedstawiona poniżej do pobrania imienia użytkownika używa funkcji InputBox języka VBA. Jeżeli imieniem nie jest Jan, procedura przechodzi do etykiety WrongName i kończy działanie. W przeciwnym razie procedura wykonuje dodatkowe polecenia. Działanie procedury kończy polecenie Exit Sub.

```
Sub GoToDemo()
    UserName = InputBox("Podaj imię:")
    If UserName <> "Jan" Then GoTo WrongName
    MsgBox ("Witaj Janku!")
    ' [W tym miejscu możesz umieścić dodatkowy kod]
    Exit Sub
WrongName:
    MsgBox "Przykro mi, ale tylko Jan może uruchomić tę procedurę."
End Sub
```

Ta prosta procedura działa, ale z pewnością nie jest przykładem dobrego programowania. W praktyce instrukcja GoTo powinna być używana tylko i wyłącznie wtedy, kiedy danej operacji nie można wykonać w inny sposób. Tak naprawdę jedyną sytuacją, w której naprawdę konieczne jest użycie w kodzie VBA polecenia GoTo, jest przechwytywanie i obsługa błędów (patrz rozdział 7.).

Nawiasem mówiąc, przedstawiona procedura wcale nie jest dobrym przykładem efektywnej i bezpiecznej metody autoryzacji!

Konstrukcja If ... Then

Prawdopodobnie najczęściej stosowaną konstrukcją grupującą języka VBA jest konstrukcja złożona z instrukcji If i Then. Jest to jeden ze sposobów wyposażania aplikacji w możliwości podejmowania decyzji. Dobry mechanizm podejmowania decyzji to klucz do tworzenia efektywnych aplikacji.

Podstawowa składnia konstrukcji If ... Then jest następująca:

```
If warunek Then instrukcje_jeżeli_warunek_prawdziwy [Else instrukcje_jeżeli_warunek_nieprawdziwy]
```

Konstrukcja If ... Then służy do warunkowego wykonania jednej lub większej liczby instrukcji. Klauzula Else jest opcjonalna. Jeżeli zostanie zastosowana, pozwala na wykonanie jednej lub kilku instrukcji w sytuacji, kiedy testowany warunek nie zostanie spełniony.

Poniższa procedura demonstruje zastosowanie konstrukcji If ... Then bez klauzuli Else do przetwarzania wartości czasowych. Język VBA, podobnie jak Excel, używa systemu numerów seryjnych powiązanych z datą. Porę dnia wyraża wartość ułamkową — przy-

kładowo południe jest reprezentowane przez wartość 0.5. Funkcja Time języka VBA zwraca wartość reprezentującą porę dnia odpowiadającą czasowi wskazywanemu przez zegar systemowy. W naszym przykładzie komunikat jest wyświetlany tylko przed południem. Jeżeli liczba seryjna reprezentująca aktualny czas systemowy ma wartość większą lub równą 0.5, procedura zakończy działanie i nie zostanie wykonana żadna operacja.

```
Sub GreetMe1()
    If Time < 0.5 Then MsgBox "Witam przed południem!"
End Sub
```

Innym sposobem zapisania tej procedury jest umieszczenie poleceń w osobnych wierszach, tak jak to przedstawiono poniżej:

```
Sub GreetMe1a()
    If Time < 0.5 Then
        MsgBox " Witam przed południem!"
    End If
End Sub
```

Zwróć uwagę, że polecenie If posiada odpowiadające mu polecenie End If. W naszym przykładzie, kiedy warunek jest spełniony, wykonywane jest tylko jedno polecenie, aczkolwiek w praktyce pomiędzy poleceniami If oraz End If możesz umieścić dowolną liczbę poleceń.

Aby wyświetlić miłe powitanie również po południu, powinieneś dodać kolejną konstrukcję If ... Then, tak jak to zrobiono poniżej:

```
Sub GreetMe2()
    If Time < 0.5 Then MsgBox "Witam przed południem!"
    If Time >= 0.5 Then MsgBox "Witaj! Już po południu!"
End Sub
```

Zwróć uwagę, że w drugiej konstrukcji If ... Then użyto operatora \geq (większy lub równy). Dzięki takiemu rozwiążaniu nasz program uwzględnia przypadek, że w chwili uruchomienia będzie dokładnie samo południe.

Innym rozwiązaniem jest zastosowanie konstrukcji If ... Then wraz z klauzulą Else, na przykład:

```
Sub GreetMe3()
    If Time < 0.5 Then MsgBox "Witam przed południem!" Else _
        MsgBox "Witaj! Już po południu!"
End Sub
```

Zwróć uwagę, że użyty tutaj został znak przeniesienia instrukcji do drugiego wiersza, ale tak naprawdę konstrukcja If ... Then ... Else to jedno polecenie.

Jeżeli w zależności od spełnienia warunku musisz wykonać kilka poleceń, powinieneś użyć takiej formy:

```
Sub GreetMe3a()
    If Time < 0.5 Then
        MsgBox "Witam przed południem!"
        ' Reszta poleceń tutaj
    Else
```

```

    MsgBox "Witaj! Już po południu!"
    ' Reszta poleceń tutaj
End If
End Sub

```

Aby rozbudować procedurę o obsługę trzech warunków (np. poranek, popołudnie i wieczór), można zastosować trzy konstrukcje If ... Then lub zagnieżdżoną strukturę If ... Then ... Else. Pierwsza metoda jest prostsza:

```

Sub GreetMe4()
    If Time < 0.5 Then MsgBox "Witam przed południem!"
    If Time >= 0.5 And Time < 0.75 Then MsgBox "Witaj! Już po południu!"
    If Time >= 0.75 Then MsgBox "Dobry wieczór!"
End Sub

```

Wartość 0.75 reprezentuje godzinę 18, czyli $\frac{3}{4}$ doby. Taką porę dnia można już nazywać wieczorem.

W poprzednich przykładach wykonywane są wszystkie polecenia procedury, nawet gdy spełniony został od razu pierwszy warunek (czyli zegar wskazuje ranną porę). Wydajniejsza procedura będzie zawierać strukturę kończącą wykonywanie procedury, gdy warunek będzie miał wartość True. Przykładowo procedura może przed południem wyświetlać komunikat "Witam przed południem!", a następnie kończyć działanie bez sprawdzania innych, zbędnych już w takiej chwili warunków. Oczywiście w przypadku tak małej i prostej procedury różnice szybkości są bez znaczenia. Jednak w bardziej złożonych aplikacjach konieczne jest zastosowanie innej składni konstrukcji If:

```

If warunek Then
    [instrukcje_wykonywane_jeżeli_spełniono_warunek]
[ElseIf warunek Then
    [alternatywne_instrukcje]]
[Else
    [domyslnie_instrukcje]]
End If

```

Poniżej pokazano, w jaki sposób należy użyć powyższej składni do przebudowania procedury GreetMe:

```

Sub GreetMe5()
    If Time < 0.5 Then
        MsgBox "Witam przed południem!"
    ElseIf Time >= 0.5 And Time < 0.75 Then
        MsgBox "Witaj! Już po południu!"
    Else
        MsgBox "Dobry wieczór!"
    End If
End Sub

```

Gdy w przypadku tej składni warunek będzie miał wartość True, zostaną wykonane instrukcje warunkowe i konstrukcja If ... Then zakończy działanie. Innymi słowy, pozostałe warunki nie są sprawdzane. Co prawda taka składnia umożliwia zwiększenie wydajności, ale niektórzy mogą uznać, że oparty na niej kod źródłowy jest trudniejszy do zrozumienia.

Poniższa procedura demonstruje kolejną metodę kodowania powyższego przykładu. Zastosowano w niej konstrukcję If ... Then ... Else (bez instrukcji ElseIf). Procedura jest

wydajna i łatwa do zrozumienia. Należy zauważyc, że każdej instrukcji If odpowiada instrukcja End If.

```
Sub GreetMe6()
    If Time < 0.5 Then
        MsgBox "Witam przed południem!"
    Else
        If Time >= 0.5 And Time < 0.75 Then
            MsgBox "Witaj! Już po południu!"
        Else
            If Time >= 0.75 Then
                MsgBox "Dobry wieczór!"
            End If
        End If
    End If
End Sub
```

Poniżej zamieszczono kolejny prosty przykład z konstrukcją If ... Then. Procedura prosi użytkownika o wprowadzenie wartości dla zmiennej Quantity, a następnie w zależności od niej wyświetla odpowiedni rabat. Zmienna Quantity została zadeklarowana jako zmienna typu Variant. Wynika to stąd, że jeżeli funkcji InputBox nie zostanie przekazany argument, zmienna Quantity będzie zawierała pusty łańcuch, a nie wartość liczbową. Procedura nie wykonuje żadnej innej kontroli błędów. Nie sprawdza na przykład, czy wprowadzona wartość liczbową zmiennej Quantity jest dodatnia.

```
Sub Discount1()
    Dim Quantity As Variant
    Dim Discount As Double
    Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")
    If Quantity = "" Then Exit Sub
    If Quantity >= 0 Then Discount = 0.1
    If Quantity >= 25 Then Discount = 0.15
    If Quantity >= 50 Then Discount = 0.2
    If Quantity >= 75 Then Discount = 0.25
    MsgBox "Rabat: " & Discount
End Sub
```

Każda konstrukcja If ... Then zawarta w procedurze jest zawsze wykonywana, dlatego wartość zmiennej Discount może się zmieniać, aczkolwiek jej końcowa wartość jest zgodna z oczekiwaniami.

Poniższa procedura, która jest modyfikacją poprzedniej, używa alternatywnej składni. W tym przypadku kończy działanie po wykonaniu bloku instrukcji realizowanych po spełnieniu warunku.

```
Sub Discount2()
    Dim Quantity As Variant
    Dim Discount As Double
    Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")
    If Quantity = "" Then Exit Sub
    If Quantity >= 0 And Quantity < 25 Then
        Discount = 0.1
    ElseIf Quantity < 50 Then
        Discount = 0.15
    ElseIf Quantity < 75 Then
        Discount = 0.2
    End If
End Sub
```

```

    Else
        Discount = 0.25
    End If
    MsgBox "Rabat: " & Discount
End Sub

```

Szczerze mówiąc, zagnieździone struktury If ... Then uważam za bardzo nieporęczne. Z tego powodu używam ich tylko w przypadku podejmowania prostych decyzji o charakterze dwuwariantowym. Gdy trzeba wybrać jedną z trzech lub większej liczby alternatywnych struktur, często lepszą propozycją będzie konstrukcja Select Case (która opiszę w kolejnym podrozdziale).

Konstrukcja Select Case

Konstrukcja Select Case przydaje się, gdy trzeba dokonać wyboru spośród trzech lub większej liczby opcji. Sprawdza się też w przypadku dwóch opcji i stanowi alternatywną propozycję dla konstrukcji If ... Then ... Else. Składnia konstrukcji Select Case jest następująca:

```

Select Case sprawdzane_wyrażenie
    [Case lista_wyrażeń
        [instrukcje]]
    [Case Else
        [domyślne_instrukcje]]
End Select

```

Funkcja IIf języka VBA

Język VBA oferuje funkcję IIf, która stanowi ciekawą alternatywę dla konstrukcji If ... Then. Funkcja pobiera trzy argumenty i działa w sposób bardzo podobny do funkcja arkuszowej JEŻELI. Składnia funkcji jest następująca:

IIf(wyrażenie, część_True, część_False)

- *wyrażenie* (wymagany): wyrażenie, które zostanie sprawdzone;
- *część_True* (wymagany): wartość lub wyrażenie zwracane, gdy wartością *wyrażenia* będzie True;
- *część_False* (wymagany): wartość lub wyrażenie zwracane, gdy wartością *wyrażenia* będzie False.

Poniższy przykład demonstruje zastosowanie funkcji IIf. Okno komunikatu wyświetla ciąg znaków Zero, gdy komórka A1 jest pusta lub gdy znajduje się w niej zero, albo ciąg znaków Różne od zera, jeżeli w komórce A1 znajdzie się dowolna inną wartość.

```
MsgBox IIf(Range("A1") = 0, "Zero", "Różne od zera")
```

Pamiętaj, że trzeci argument (*część_False*) jest również zawsze obliczany, nawet jeżeli wartością pierwszego argumentu (*wyrażenie*) będzie True, stąd poniższa instrukcja wygeneruje błąd, gdy wartością n będzie zero:

```
MsgBox IIf(n = 0, 0, 1 / n)
```

Oto kolejny wariant procedury GreetMe z poprzedniego podrozdziału, w którym zastosowano konstrukcję Select Case:

```
Sub GreetMe()
    Dim Msg As String
    Select Case Time
        Case Is < 0.5
            Msg = "Witam przed południem!"
        Case 0.5 To 0.75
            Msg = "Witaj! Już po południu!"
        Case Else
            Msg = "Dobry wieczór!"
    End Select
    MsgBox Msg
End Sub
```

Poniżej zamieszczono zmodyfikowaną wersję procedury Discount, w której użyto konstrukcji Select Case. Procedura zakłada, że wartość zmiennej Quantity zawsze będzie liczbą całkowitą. Aby uprościć kod przykładu, w procedurze nie zastosowano żadnej obsługi błędów.

```
Sub Discount3()
    Dim Quantity As Variant
    Dim Discount As Double
    Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")
    Select Case Quantity
        Case ""
            Exit Sub
        Case 0 To 24
            Discount = 0.1
        Case 25 To 49
            Discount = 0.15
        Case 50 To 74
            Discount = 0.2
        Case Is >= 75
            Discount = 0.25
    End Select
    MsgBox "Rabat: " & Discount
End Sub
```

Do oddzielenia wielu wartości dla danego warunku instrukcji Case można użyć znaku przecinka. W poniższej procedurze zastosowano funkcję WeekDay języka VBA, określającą, czy aktualny dzień jest sobotą lub niedzielą (czyli kiedy funkcja WeekDay zwróci wartość 1 lub 7). Na końcu procedura wyświetli odpowiedni komunikat.

```
Sub GreetUser1()
    Select Case WeekDay(Now)
        Case 1, 7
            MsgBox "Mamy już weekend."
        Case Else
            MsgBox "To jeszcze nie weekend."
    End Select
End Sub
```

A oto kolejny wariant tej samej procedury:

```
Sub GreetUser2()
    Select Case WeekDay(Now)
```

```
Case 2, 3, 4, 5, 6
    MsgBox "To jeszcze nie weekend."
Case Else
    MsgBox "Mamy już weekend."
End Select
End Sub
```

Poniżej przedstawiono jeszcze inny sposób implementacji takiej procedury, tym razem wykorzystujący słowo kluczowe To do zdefiniowania zakresu wartości:

```
Sub GreetUser3()
    Select Case Weekday(Now)
        Case 2 To 6
            MsgBox "To jeszcze nie weekend"
        Case Else
            MsgBox "Mamy już weekend"
    End Select
End Sub
```

Aby zademonstrować wielką elastyczność języka VBA, pokażemy jeszcze jeden, finalny przykład implementacji tej procedury, gdzie warunki są sprawdzane kolejno aż do momentu, kiedy któryś z nich zostanie spełniony (będzie miał wartość True):

```
Sub GreetUser4()
    Select Case True
        Case Weekday(Now) = 1
            MsgBox "Mamy już weekend"
        Case Weekday(Now) = 7
            MsgBox "Mamy już weekend"
        Case Else
            MsgBox "To jeszcze nie weekend"
    End Select
End Sub
```

Pod każdą instrukcją Case można umieścić dowolną liczbę instrukcji, które zostaną wykonane, jeżeli dany warunek zostanie spełniony (będzie miał wartość True). Jeżeli dla danego warunku chcesz użyć tylko jednego polecenia (tak jak w poprzednim przykładzie), to możesz je umieścić w tym samym wierszu, co słowo kluczowe Case (nie należy jednak zapominać o wstawieniu separatora instrukcji języka VBA, czyli dwukropka). Dzięki takiemu rozwiązaniu kod źródłowy jest bardziej zwarty. Oto przykład:

```
Sub Discount3()
    Dim Quantity As Variant
    Dim Discount As Double
    Quantity = InputBox("Wprowadź liczbę kupowanych książek: ")
    Select Case Quantity
        Case "": Exit Sub
        Case 0 To 24: Discount = 0.1
        Case 25 To 49: Discount = 0.15
        Case 50 To 74: Discount = 0.2
        Case Is >= 75: Discount = 0.25
    End Select
    MsgBox "Rabat: " & Discount
End Sub
```



VBA kończy przetwarzanie konstrukcji Select Case natychmiast po spełnieniu któregoś z warunków. Aby uzyskać maksymalną efektywność takiego kodu, powinieneś jako pierwszy do sprawdzenia umieścić najbardziej prawdopodobny przypadek.

Struktury Select Case też mogą być zagnieżdżane. Procedura przedstawiona poniżej używa funkcji TypeName VBA do sprawdzenia, jaki element został zaznaczony w skoroszycie (zakres komórek, nic nie jest zaznaczone itd.). Jeżeli zaznaczony został zakres, procedura wykonuje zagnieżdżone polecenie Select Case i sprawdza liczbę zaznaczonych komórek. Jeżeli zaznaczona jest tylko jedna komórka, wyświetlany jest komunikat *Tylko jedna komórka jest zaznaczona!*. W przeciwnym wypadku na ekranie wyświetlana jest liczba zaznaczonych wierszy.

```
Sub SelectionType()
    Select Case TypeName(Selection)
        Case "Range"
            Select Case Selection.Count
                Case 1
                    MsgBox "Tylko jedna komórka jest zaznaczona!"
                Case Else
                    MsgBox Selection.Rows.Count & " wierszy"
            End Select
        Case "Nothing"
            MsgBox "Nic nie jest zaznaczone!"
        Case Else
            MsgBox "Coś innego niż zakres!"
    End Select
End Sub
```

Przedstawiona procedura ilustruje również sposób użycia klauzuli Case Else, która obsługuje wszystkie pozostałe przypadki. Stopień zagnieżdżenia konstrukcji Select Case może być dowolny, ale powinieneś upewnić się, że każdej instrukcji Select Case odpowiada instrukcja End Select.

Powyższa procedura demonstruje również przydatność wcięć, które zwiększą przejrzystość struktury kodu źródłowego. Spójrz na tę samą procedurę pozbawioną wcięć:

```
Sub SelectionType()
    Select Case TypeName(Selection)
        Case "Range"
            Select Case Selection.Count
                Case 1
                    MsgBox "Tylko jedna komórka jest zaznaczona!"
                Case Else
                    MsgBox Selection.Rows.Count & " wierszy"
            End Select
        Case "Nothing"
            MsgBox "Nic nie jest zaznaczone!"
        Case Else
            MsgBox "Coś innego niż zakres!"
    End Select
End Sub
```

Sam przyznasz, że to niezły pasztet...

Wykonywanie bloku instrukcji w ramach pętli

Pętla jest procesem wielokrotnego wykonywania określonego bloku instrukcji. Liczba wykonań pętli może być z góry znana lub określana przez wartość zmiennych programu.

Poniższa procedura wstawiająca do zakresu kolejne wartości liczbowe demonstruje to, co określам mianem **zły pętli**. Procedura do przechowania wartości początkowej i całkowitej liczby wypełnianych komórek używa dwóch zmiennych (są to odpowiednio StartVal i NumToFill). Pętla używa instrukcji GoTo do sterowania wykonywaniem programu. Jeżeli wartość zmiennej Cnt przechowującej liczbę wypełnianych komórek jest mniejsza od wartości zmiennej NumToFill, program ponownie przechodzi do etykiety DoAnother.

```
Sub BadLoop()
    Dim StartVal As Integer
    Dim NumToFill As Integer
    Dim Cnt As Integer
    StartVal = 1
    NumToFill = 100
    ActiveCell.Value = StartVal
    Cnt = 1
    DoAnother:
        ActiveCell.Offset(Cnt, 0).Value = StartVal + Cnt
        Cnt = Cnt + 1
        If Cnt < NumToFill Then GoTo DoAnother Else Exit Sub
End Sub
```

Powyższa procedura działa zgodnie z oczekiwaniami, zatem dlaczego jest przykładem **zły pętli**? Programiści zazwyczaj krzywią się na widok instrukcji GoTo, gdy jej użycie nie jest absolutnie konieczne. Zastosowanie instrukcji GoTo w roli pętli jest zaprzeczeniem programowania strukturalnego (zapoznaj się z zawartością ramki „Czym jest programowanie strukturalne?”). Instrukcja GoTo znacznie pogarsza czytelność kodu źródłowego, ponieważ zapisanie takiej pętli przy użyciu wcięć wierszy jest prawie niemożliwe. Ponadto tego typu niestrukturalna pętla sprawia, że procedura jest podatna na błędy. Co więcej, zastosowanie wielu etykiet generuje kod „spaghetti”, który posiada znikomą lub wręcz nie posiada żadnej struktury i jest podatny na powstawanie niekontrolowanych hazardów.

Ze względu na to, że język VBA oferuje kilka instrukcji spełniających funkcję strukturalnych pętli, przy tworzeniu mechanizmów podejmowania decyzji praktycznie nigdy nie będziesz musiał korzystać z instrukcji GoTo.

Pętla For ... Next

Najprostszym typem dobrej pętli jest pętla For ... Next. Oto jej składnia:

```
For [licznik] = [wartość_początkowa] To [wartość_końcowa] [Step [wartość_kroku]]
    [instrukcje]
    [Exit For]
    [instrukcje]
Next [licznik]
```

Czym jest programowanie strukturalne?

Mając kontakt z programistami, wcześniej czy później poznasz pojęcie *programowania strukturalnego*. Dowiesz się też, że programy strukturalne są lepsze od programów pozbawionych struktury.

A zatem czym jest programowanie strukturalne i czy język VBA jest językiem strukturalnym?

Zgodnie z podstawowym założeniem programowania strukturalnego, procedura lub segment kodu źródłowego powinny posiadać tylko jeden punkt wejściowy i jeden wyjściowy. Innymi słowy, zawartość kodu źródłowego powinna być niezależną jednostką i sterowanie programem nie powinno przechodzić do jej środka lub z niego wychodzić. W rezultacie takich założeń, programowanie strukturalne wyklucza stosowanie instrukcji GoTo. Strukturalny kod źródłowy jest wykonywany w sposób uporządkowany i łatwy do przeanalizowania, w przeciwieństwie do ryzykownie przetwarzanego kodu „spaghetti”, w którym sterowanie jest przekazywane chaotycznie do różnych części programu.

Program strukturalny jest też łatwiejszy do przeglądania i zrozumienia, a co ważniejsze, znacznie łatwiej można wprowadzać do niego wymagane modyfikacje.

Język VBA jest językiem strukturalnym. Oferuje standardowe konstrukcje strukturalne, takie jak If ... Then ... Else i Select Case oraz pętle For ... Next, Do Until i Do While. Ponadto w pełni obsługuje tworzenie kodu źródłowego dzielonego na moduły.

Jeżeli jesteś początkującym programistą, powinieneś od razu starać się wyrobić sobie nawyki dobrego programowania strukturalnego.

Poniżej zamieszczono przykład pętli For ... Next, która nie używa opcjonalnego słowa kluczowego Step lub instrukcji Exit For. Procedura sto razy wykonuje instrukcję Sum = Sum + Sqr(Count) i wyświetla wynik, czyli sumę pierwiastków kwadratowych pierwszych stu liczb całkowitych.

```
Sub SumSquareRoots()
    Dim Sum As Double
    Dim Count As Integer
    Sum = 0
    For Count = 1 To 100
        Sum Sum + Sqr(Count)
    Next Count
    MsgBox Sum
End Sub
```

W tym przykładzie początkową wartością zmiennej Count (licznik pętli) jest 1. Po każdej iteracji wartość ta jest zwiększana o jeden. Zmienna Sum jedynie przechowuje sumę pierwiastków kwadratowych obliczonych dla każdej wartości zmiennej Count.



Korzystając z pętli For ... Next powinieneś pamiętać, że licznik pętli jest zwykłą zmienną i nie ma żadnych specjalnych właściwości. Z tego powodu można zmienić jego wartość wewnątrz wykonywanego kodu źródłowego, zawartego pomiędzy instrukcjami For i Next. Bezpośrednia modyfikacja zmiennej licznika pętli nie jest jednak zalecaną praktyką, gdyż może wywołać nieprzewidywalne efekty. W praktyce powinieneś zawsze upewnić się, że kod źródłowy nie modyfikuje bezpośrednio wartości licznika pętli.

Aby pominąć niektóre wartości pętli, można zastosować słowo kluczowe Step. Poniżej procedurę z poprzedniego przykładu zmodyfikowano w taki sposób, że oblicza sumę pierwiastków kwadratowych dla nieparzystych liczb z przedziału od 1 do 100:

```
Sub SumOddSquareRoots()
    Dim Sum As Double
    Dim Count As Integer
    Sum = 0
    For Count = 1 To 100 Step 2
        Sum Sum + Sqr(Count)
    Next Count
    MsgBox Sum
End Sub
```

Wartością początkową zmiennej Count jest 1, a następnie liczby nieparzyste 3, 5, 7 itd. Ostatnią wartością zmiennej Count jest 99. Po zakończeniu wykonywania pętli zmienna Count przyjmie wartość 101.

Parametr Step pętli For...Next może również przyjmować wartości ujemne. Procedura przedstawiona poniżej usuwa z aktywnego skoroszytu 2, 4, 6, 8 oraz 10 wiersz.

```
Sub DeleteRows()
    Dim RowNum As Long
    For RowNum = 10 To 2 Step -2
        Rows(RowNum).Delete
    Next RowNum
End Sub
```

Zastanawiasz się pewnie, dlaczego w procedurze DeleteRows użyto ujemnej wartości parametru Step. Jeżeli użyjemy dodatniej wartości tego parametru, tak jak to przedstawiono w procedurze poniżej, usunięte zostaną niewłaściwe wiersze. Dzieje się tak, ponieważ wiersz znajdujący się poniżej usuwanego wiersza automatycznie otrzymuje nowy numer. Na przykład: jeżeli usuniemy wiersz numer 2, dotychczasowy wiersz numer 3 staje się nowym wierszem numer 2. Użycie ujemnej wartości parametru Step powoduje, że usuwane są wiersze o poprawnych numerach.

```
Sub DeleteRows2()
    Dim RowNum As Long
    For RowNum = 2 To 10 Step 2
        Rows(RowNum).Delete
    Next RowNum
End Sub
```

Poniższa procedura wykonuje to samo zadanie, co procedura BadLoop zamieszczona na początku podrozdziału „Wykonywanie bloku instrukcji w ramach pętli”. Usunięto z niej instrukcję GoTo i zamieniono złą pętlę na dobrą, opartą na strukturze pętli For ... Next:

```
Sub GoodLoop()
    Dim StartVal As Integer
    Dim NumToFill As Integer
    Dim Cnt As Integer
    StartVal = 1
    NumToFill = 100
    For Cnt = 0 To NumFill - 1
        ActiveCell.Offset(Cnt, 0).Value = StartVal + Cnt
    Next Cnt
End Sub
```

Pętle For ... Next mogą też zawierać jedną lub kilka instrukcji Exit For. Po napotkaniu takiej instrukcji pętla natychmiast kończy działanie i przekazuje sterowanie do pierwszej instrukcji znajdującej się za instrukcją Next aktualnej pętli For ... Next. Poniższy przykład ilustruje zastosowanie instrukcji Exit For. Procedura określa, w której komórce kolumny A aktywnego arkusza znajduje się największa wartość.

```
Sub ExitForDemo()
    Dim MaxVal As Double
    Dim Row As Long
    MaxVal = Application.WorksheetFunction.Max(Range("A:A"))
    For Row = 1 To 1048576
        If Cells(Row, 1).Value = MaxVal Then
            Exit For
        End If
    Next Row
    MsgBox "Największa wartość znajduje się w wierszu nr: " & Row
    Cells(Row, 1).Activate
End Sub
```

Maksymalna wartość kolumny jest wyliczana przy użyciu funkcji MAX Excela i następnie przypisywana do zmiennej MaxVal. Pętla For ... Next sprawdza każdą komórkę kolumny. Jeżeli wartość sprawdzanej komórki jest równa wartości zmiennej MaxVal, instrukcja Exit For kończy działanie pętli i wykonywane są polecenia znajdującej się po instrukcji Next, które wyświetlają na ekranie największą wartość i aktywują ją komórkę.



Procedura ExitForDemo została utworzona po to, aby zademonstrować sposób wyjścia z pętli For ... Next. Oczywiście nie jest to najbardziej efektywna metoda wyszukiwania i aktywacji komórki zawierającej największą wartość w zakresie. W praktyce taką operację można wykonać za pomocą zaledwie jednego polecenia:

```
Range("A:A").Find(Application.WorksheetFunction.Max _  
(Range("A:A"))).Activate
```

W poprzednich przykładach zastosowano stosunkowo proste pętle. Jednak można w nich zatrzymać dowolną liczbę instrukcji, a nawet zagnieździć pętle For ... Next wewnętrznie w innych pętlach For ... Next. Oto przykład zawierający zagnieżdżone pętle For ... Next inicjujące tablicę o wymiarach 10×10×10 przy użyciu wartości -1. Po zakończeniu działania tej procedury każdy z tysiąca elementów tablicy MyArray będzie miał wartość -1.

```
Sub NestedLoops()
    Dim MyArray(1 To 10, 1 To 10, 1 To 10)
    Dim i As Integer, j As Integer, k As Integer
    For i = 1 To 10
        For j = 1 To 10
            For k = 1 To 10
                MyArray(i, j, k) = -1
            Next k
        Next j
    Next i
    ' Tutaj możesz wstawić pozostałą część kodu
End Sub
```

Pętla Do While

W tym podrozdziale omówimy kolejny rodzaj pętli dostępnej w języku VBA. Pętla Do While, w przeciwieństwie do pętli For ... Next, jest wykonywana dopóty, dopóki jest spełniony warunek pętli.

Pętla Do While może występować w dwóch wariantach składni:

```
Do [While warunek]
    [instrukcje]
    [Exit Do]
    [instrukcje]
```

Loop

lub

```
Do
    [instrukcje]
    [Exit Do]
    [instrukcje]
Loop [While warunek]
```

Jak łatwo zauważyc, język VBA umożliwia umieszczenie instrukcji warunkowej While na początku lub na końcu pętli. Różnica pomiędzy obiema składniami polega na momencie sprawdzania warunku. W przypadku pierwszej składni zawartość pętli może nigdy nie być wykonana. Z kolei w przypadku drugiej składni zawartość pętli zawsze będzie wykonyana przynajmniej raz.

Przykłady zamieszczone poniżej wstawiają serie dat do aktywnego arkusza. Daty odpowiadają dniom bieżącego miesiąca i są wstawiane do arkusza, począwszy od aktywnej komórki.



W omawianych przykładach wykorzystujemy kilka funkcji języka VBA przetwarzających daty:

- Funkcja Date zwraca bieżącą datę.
- Funkcja Month zwraca numer miesiąca odpowiadający dacie przekazanej jako argument wywołania funkcji.
- Funkcja DateSerial zwraca datę odpowiadającą rokowi, miesiącowi i dniu przekazanym jako argumenty wywołania funkcji.

Pierwszy przykład prezentuje pętlę Do While, która testuje warunek pętli przed rozpoczęciem działania. Procedura EnterDates1 zapisuje kolejne daty w kolumnie arkusza, począwszy od aktywnej komórki.

```
Sub EnterDates1()
    ' Pętla Do While, warunek pętli jest testowany przed rozpoczęciem działania
    Dim TheDate As Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do While Month(TheDate) = Month(Date)
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Loop
End Sub
```

Powyższa procedura wykorzystuje zmienną TheDate, która zawiera kolejne daty zapisywane do komórek arkusza. Zmienna jest inicjowana wartością reprezentującą pierwszy dzień bieżącego miesiąca. Po wejściu do pętli wartość zmiennej TheDate jest zapisywana w aktywnej komórce, a następnie wartość zmiennej TheDate jest inkrementowana i aktywowana jest następna komórka. Pętla kontynuuje działanie tak długo, aż wartość miesiąca w zmiennej TheDate będzie taka sama jak wartość miesiąca bieżącej daty.

Procedura przedstawiona poniżej daje identyczne rezultaty jak procedura EnterDates1, ale wykorzystuje drugą postać składni pętli Do While, gdzie warunek jest sprawdzany na końcu pętli.

```
Sub EnterDates2()
    ' Pętla Do While, warunek testowany na końcu pętli
    Dim TheDate As Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Loop While Month(TheDate) = Month(Date)
End Sub
```

Poniżej zamieszczono kolejny przykład użycia pętli Do While. Procedura otwiera plik tekstowy, odczytuje kolejne wiersze, zamienia małe litery na wielkie i zapisuje je w aktywnym arkuszu, począwszy od komórki A1 w dół kolumny. Procedura używa funkcji EOF języka VBA zwracającej wartość True po osiągnięciu końca pliku. Ostatnia instrukcja zamknuje plik tekstowy.

```
Sub DoWhileDemo1()
    Dim LineCt As Long
    Dim LineOfText As String
    Open "c:\data\plik_tekstowy.txt" For Input As #1
    LineCt = 0
    Do While Not EOF(1)
        Line Input #1, LineOfText
        Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
        LineCt = LineCt + 1
    Loop
    Close #1
End Sub
```



Więcej szczegółowych informacji na temat odczytywania i zapisywania plików tekstowych przy użyciu VBA znajdziesz w rozdziale 25.

Pętle Do While również mogą zawierać jedno lub nawet kilka poleceń Exit Do. Po napotkaniu polecenia Exit Do pętla natychmiast kończy działanie i przekazuje sterowanie do polecenia znajdującego się bezpośrednio za poleceniem Loop.

Pętla Do Until

Struktura pętli Do Until bardzo przypomina strukturę pętli Do While. Różnica jest widoczna jedynie przy sprawdzaniu warunku. Pętla Do While jest wykonywana, gdy warunek pętli jest spełniony (ma wartość True). Z kolei pętla Do Until jest wykonywana tak długo, jak długo warunek pętli nie jest spełniony.

Pętla Do Until również posiada dwa warianty składni:

```
Do [Until warunek]
    [instrukcje]
    [Exit Do]
    [instrukcje]
Loop
```

lub

```
Do
    [instrukcje]
    [Exit Do]
    [instrukcje]
Loop [Until warunek]
```

Dwa przykłady zamieszczone poniżej przedstawiają procedury, które wykonują dokładnie takie same operacje jak przykłady z pętlami Do While z poprzedniej sekcji. Jedyna różnica pomiędzy tymi procedurami to miejsce, w którym sprawdzany jest warunek pętli (na początku lub na końcu pętli).

```
Sub EnterDates3()
    ' Pętla Do Until, warunek pętli jest testowany przed rozpoczęciem działania
    Dim TheDate As Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do Until Month(TheDate) <> Month(Date)
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Loop
End Sub

Sub EnterDates4()
    ' Pętla Do Until, warunek testowany na końcu pętli
    Dim TheDate As Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    Do
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Loop Until Month(TheDate) <> Month(Date)
End Sub
```

Poniższy przykład był już prezentowany przy okazji omawiania pętli Do While, ale po zmodyfikowaniu wykorzystano w nim pętlę Do Until. Jedyna różnica występuje w wierszu zawierającym instrukcję Do. Kod źródłowy jest ponadto trochę bardziej przejrzysty, ponieważ uniknięto w nim negacji obecnej w procedurze DoWhileDemo1.

```
Sub DoUntilDemo1()
    Dim LineCt As Long
    Dim LineOfText As String
    Open "c:\dane\plik_tekstowy.txt" For Input As #1
    LineCt = 0
    Do Until EOF(1)
        Line Input #1, LineOfText
        Range("A1").Offset(LineCt, 0) = UCase(LineOfText)
        LineCt = LineCt + 1
    Loop
    Close #1
End Sub
```



W języku VBA występuje jeszcze jeden rodzaj pętli, `While Wend`, która została zaimplementowana głównie ze względu na konieczność zachowania kompatybilności ze starszymi wersjami tego języka. Wspominam o tej pętli tylko dlatego, że możesz się z nią jeszcze spotkać w kodach starszych programów i aplikacji. Poniżej zamieszczono kod procedury wprowadzającej kolejne daty do komórek arkusza, napisany przy użyciu pętli `While Wend`.

```
Sub EnterDates5()
    Dim TheDate As Date
    TheDate = DateSerial(Year(Date), Month(Date), 1)
    While Month(TheDate) = Month(Date)
        ActiveCell = TheDate
        TheDate = TheDate + 1
        ActiveCell.Offset(1, 0).Activate
    Wend
End Sub
```


Rozdział 7.

Tworzenie procedur w języku VBA

W tym rozdziale:

- Deklarowanie i tworzenie procedur Sub języka VBA
- Uruchamianie procedur Sub
- Przekazywanie argumentów do procedury
- Metody obsługi błędów
- Przykłady tworzenia procedur Sub

Kilka słów o procedurach

Procedura to zgrupowana seria poleceń języka VBA znajdujących się w module kodu VBA, do których masz dostęp z poziomu edytora Visual Basic (VBE). Moduł kodu może przechowywać dowolną liczbę procedur. Procedura przechowuje grupę instrukcji języka VBA realizujących określone zadanie. Większość kodu VBA przechowywana jest w procedurach.

Istnieje kilka metod *wywoływania* lub inaczej mówiąc, uruchamiania procedur. Polecenia procedury są wykonywane kolejno, od początku do końca, ale procedura może również zostać zakończona wcześniej.



Procedura może mieć dowolną długość, ale programiści wolą raczej unikać tworzenia bardzo rozbudowanych procedur wykonujących wiele złożonych operacji. Lepszym rozwiązaniem jest utworzenie kilku mniejszych procedur, z których każda realizuje jedno zadanie, a następnie napisanie głównej procedury wywołującej w odpowiedniej kolejności procedury składowe. Dzięki takiemu rozwiązaniu zarządzanie kodem programu jest zdecydowanie łatwiejsze.

Niektóre procedury do poprawnego działania wymagają przekazania odpowiednich argumentów. *Argument* to po prostu informacja wykorzystywana przez procedurę i przekazywana jej po uruchomieniu. Argumenty procedury odgrywają podobną rolę, jak argumenty stosowane w funkcjach arkusza Excela. Kolejne instrukcje procedury zazwyczaj wykonyują na argumentach różnego rodzaju operacje, a wyniki generowane przez procedurę często w dużej mierze zależą od przekazanych argumentów.



Pomimo, iż w tym rozdziale skoncentrujemy się głównie na procedurach typu Sub, to powinieneś pamiętać, że w języku VBA wykorzystywane są również funkcje (procedury typu Function), które bardziej szczegółowo zostaną omówione w rozdziale 8. W rozdziale 9. natomiast znajdziesz wiele dodatkowych przykładów procedur i funkcji, które możesz wykorzystać podczas tworzenia własnych aplikacji.

Deklarowanie procedury Sub

Procedura deklarowana przy użyciu słowa kluczowego Sub musi posiadać następującą składnię:

```
[Private | Public][Static] Sub nazwa ([lista_argumentów])
    [instrukcje]
    [Exit Sub]
    [instrukcje]
End Sub
```

Poniżej zamieszczamy zestawienie poszczególnych elementów składni procedury Sub:

- Private (opcjonalne słowo kluczowe) — wskazuje, że procedura będzie dostępna tylko dla innych procedur z tego samego modułu.
- Public (opcjonalne słowo kluczowe) — wskazuje procedurę dostępną dla wszystkich innych procedur we wszystkich modułach skoroszytu. Procedura zastosowana w module zawierającym instrukcję Option Private Module nie będzie dostępna na zewnątrz projektu nawet pomimo zastosowania słowa kluczowego Public.
- Static (opcjonalne słowo kluczowe) — wskazuje, że zmienne zadeklarowane w tej procedurze nie zostaną usunięte po zakończeniu działania procedury, ale będą przechowywane w pamięci do jej kolejnego wywołania.
- Sub (wymagane słowo kluczowe) — wskazuje początek procedury.
- *nazwa* (wymagana) — dowolna poprawna nazwa procedury.
- *lista_argumentów* (opcjonalna) — reprezentuje listę zmiennych zawartych w nawiasach okrągłych, pobierających argumenty przekazywane procedurze. Poszczególne argumenty powinny być oddzielone od siebie przecinkami. Jeżeli procedura nie zawiera argumentów, konieczne jest użycie pustej pary nawiasów okrągłych.
- *instrukcje* (opcjonalne) — zestaw poprawnych instrukcji języka VBA.
- Exit Sub (klauzula opcjonalna) — wymusza natychmiastowe zakończenie procedury jeszcze przed osiągnięciem jej formalnego końca.
- End Sub (wymagane słowo kluczowe) — wskazuje koniec procedury.



Poza kilkoma wyjątkami, wszystkie instrukcje języka VBA znajdujące się w module muszą być umieszczone w procedurach. Odstępstwem od tej reguły są deklaracje zmiennych na poziomie modułu, definicje niestandardowych typów danych i kilka innych instrukcji określających opcje obowiązujące dla całego modułu (np. opcja Option Explicit).

Nazwy procedur

Każda procedura musi posiadać nazwę. Zasady nadawania procedurom nazw są takie same jak zasady tworzenia nazw zmiennych. W idealnej sytuacji nazwa procedury powinna nawiązywać do operacji realizowanej przez procedurę. Dobrym nawykiem jest używanie nazw zawierających czasownik i rzeczownik (np. PrzetwarzajDane, DrukujRaport, SprawdźNazwęPliku czy Sortuj_Tablice). Tworząc nazwy procedur, powinieneś unikać nic nieznaczących nazw, takich jak ZróbTo, Aktualizuj czy Napraw.

Niektórzy programiści nadają procedurom długie nazwy będące w istocie wyrażeniami opisującymi zadania realizowane przez procedurę (np. Ustaw_Opcje_Drukowania_i_Wydrukuj_Raport czy ZapiszRaportWPlikuTekstowym).

Zasięg procedury

W poprzednim rozdziale wspominaliśmy, że *zasięg* zmiennej określa moduły i procedury, w których można ją zastosować. Analogicznie, zasięg procedury określa inne procedury, z poziomu których może zostać wywołana.

Procedury publiczne

Domyślnie wszystkie procedury są *publiczne*, co oznacza, że mogą być wywoływane przez inne procedury zawarte w dowolnym module skoroszytu. Do określenia takiej procedury nie musisz używać słowa kluczowego *Public*, ale programiści często je stosują dla uniknięcia nieporozumień i zwiększenia przejrzystości kodu. Poniżej przedstawiono dwa przykłady procedur publicznych:

```
Sub First()  
    ' ...[w tym miejscu umieść kod procedury] ...  
End Sub  
  
Public Sub Second()  
    ' ...[w tym miejscu umieść kod procedury] ...  
End Sub
```

Procedury prywatne

Procedury *prywatne* mogą być wywoływane przez inne procedury zawarte w tym samym module, ale nie przez procedury z innych modułów.



Kiedy użytkownik przywoła na ekran okno dialogowe *Makro* (możesz to zrobić, naciśkając kombinację klawiszy *Alt+F8*), Excel wyświetla tylko listę procedur publicznych, stąd jeżeli utworzyłeś procedury, które mają być wywoływane tylko przez inne procedury z tego samego modułu, powinieneś pamiętać, aby przy ich deklarowaniu użyć słowa kluczowego *Private*. Dzięki temu taka procedura nie będzie mogła być uruchomiona przez użytkownika z poziomu okna dialogowego *Makro*.

W poniższym przykładzie zadeklarowano prywatną procedurę o nazwie *MySub*:

```
Private Sub MySub()  
    ' ...[w tym miejscu umieść kod procedury ...]  
End Sub
```



W razie potrzeby możesz wymusić, aby wszystkie procedury w danym module — nawet te zadeklarowane przy użyciu słowa kluczowego `Public` — były prywatne. Aby to zrobić, przed pierwszą procedurą modułu powinieneś umieścić następującą klawułkę:

Option Private Module

Jeżeli umieścisz tę instrukcję w module, w deklaracjach procedur `Sub` będzie można pominąć słowo kluczowe `Private`.

Rejestrator makr Excela standardowo tworzy nowe procedury `Sub` o nazwach `Makro1`, `Makro2` itd. Tak utworzone procedury są zawsze publiczne i nigdy nie pobierają argumentów; aby to zmienić, musisz ręcznie dokonać odpowiednich modyfikacji automatycznie zarejestrowanego kodu źródłowego.

Wykonywanie procedur `Sub`

W tym punkcie omówimy różne sposoby *uruchamiania*, czy inaczej mówiąc, wywoływania procedur `Sub` języka VBA:

- Uruchamianie procedury przy użyciu polecenia *Run Sub/UserForm* z menu *Run* edytora VBE (zamiast tego możesz nacisnąć klawisz *F5* lub nacisnąć przycisk *Run Sub/UserForm* znajdujący się na standardowym pasku narzędzi).
- Uruchamianie procedury z poziomu okna dialogowego *Makro*.
- Uruchamianie procedury przy użyciu skrótu z klawiszem *Ctrl* (zakładając, że utworzyłeś taki skrót).
- Uruchamianie procedury poprzez naciśnięcie przycisku lub kształtu na arkuszu (oczywiście taka procedura musi zostać wcześniej przypisana do arkusza lub kształtu).
- Uruchamianie procedury z poziomu innej procedury. Zarówno procedury `Sub`, jak i funkcje VBA mogą wywoływać inne procedury i funkcje.
- Uruchamianie procedury poprzez naciśnięcie własnego, zdefiniowanego przez użytkownika przycisku na pasku narzędzi *Szybki dostęp*.
- Uruchamianie procedury poprzez naciśnięcie własnego, zdefiniowanego przez użytkownika przycisku na Wstążce
- Uruchamianie procedury za pośrednictwem niestandardowego menu podręcznego.
- Uruchamianie procedury po wystąpieniu określonego zdarzenia, takiego jak otwarcie, zapisanie i zamknięcie skoroszytu, zmodyfikowanie komórki, uaktywnienie arkusza i wiele innych.
- Uruchamianie procedury z poziomu okna *Immediate* edytora Visual Basic. Aby to zrobić, wystarczy w oknie *Immediate* wpisać nazwę procedury (razem z odpowiednimi argumentami, o ile ich podanie jest wymagane) i nacisnąć klawisz *Enter*.

Poszczególne sposoby uruchamiania procedur zostaną omówione w kolejnych podrozdziałach.



W wielu przypadkach procedura nie będzie działała poprawnie do momentu zastosowania jej w odpowiednim kontekście. Na przykład, jeżeli dana procedura została stworzona z myślą o przetwarzaniu aktywnego zwykłego arkusza, nie zadziała, gdy aktywny będzie arkusz wykresu. Dobrze napisana procedura zawiera kod źródłowy sprawdzający, czy jest używana w odpowiednim kontekście. Jeżeli „stwierdzi”, że nie może kontynuować przetwarzania, zakończy działanie w poprawny, kontrolowany sposób.

Uruchamianie procedury przy użyciu polecenia Run Sub/UserForm

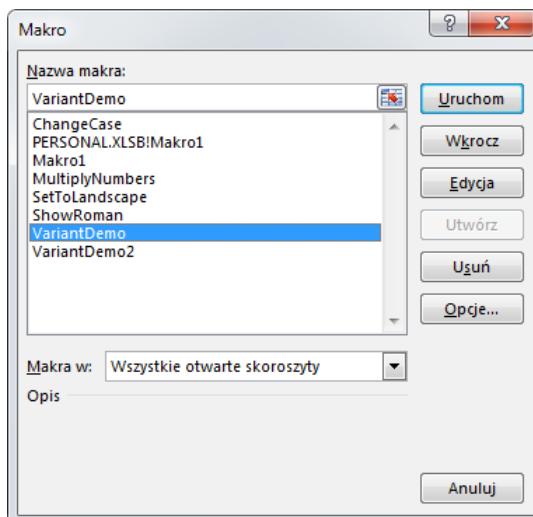
Polecenie *Run Sub/UserForm* dostępne w menu edytora Visual Basic służy przede wszystkim do testowania procedury w trakcie jej tworzenia — nie można przecież wymagać od użytkownika, aby w celu wykonania procedury za każdym razem uruchamiał edytor VBE. Aby uruchomić bieżącą procedurę (czyli procedurę, w obrębie której aktualnie znajduje się kursor), z menu *Run* edytora Visual Basic powinieneś wybrać polecenie *Run Sub/UserForm*. Zamiast tego możesz również nacisnąć klawisz *F5* lub nacisnąć przycisk *Run Sub/UserForm* znajdujący się na standardowym pasku narzędzi edytora VBE.

Jeżeli w momencie wykonywania polecenia *Run Sub/UserForm* kursor nie znajduje się na obszarze kodu źródłowego procedury, edytor Visual Basic wyświetli okno dialogowe *Macros* umożliwiające wybranie procedury, która zostanie uruchomiona.

Uruchamianie procedury z poziomu okna dialogowego Makro

Aby przywołać na ekran okno dialogowe *Makro* (patrz rysunek 7.1), przejdź na kartę *DEVELOPER* i naciśnij przycisk *Makra* znajdujący się w grupie opcji *Kod* (zamiast tego możesz po prostu nacisnąć kombinację klawiszy *Alt+F8*). Aby ograniczyć zakres wyświetlanych makr, użyj listy rozwijanej *Makra w* (na przykład aby wyświetlić tylko makra znajdujące się w aktywnym skoroszycie).

Rysunek 7.1.
Okno dialogowe *Makro*



Okno dialogowe *Makro* *nie* wyświetla następujących elementów:

- Funkcji (procedur typu Function).
- Procedur Sub zadeklarowanych przy użyciu słowa kluczowego Private (procedur prywatnych).
- Procedur Sub wymagających przekazania co najmniej jednego argumentu.
- Procedur Sub znajdujących się w dodatkach (ang. add-ins).
- Procedur obsługi zdarzeń, zapisanych w modułach kodu powiązanych z takimi obiektami jak ThisWorkbook, Sheet1 czy UserForm1.



Procedury zapisane w dodatkach nie są wyświetlane w oknie dialogowym *Makro*, ale jeżeli znasz ich nazwy, to w dalszym ciągu możesz je uruchomić. Aby to zrobić, wystarczy w polu *Nazwa makra* okna dialogowego *Makro* wpisać nazwę procedury, a następnie nacisnąć przycisk *Uruchom*.

Uruchamianie procedury przy użyciu skrótu z klawiszem Ctrl

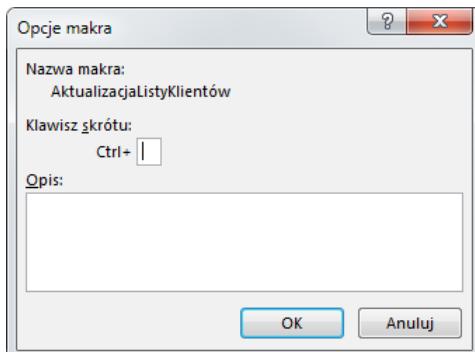
Skrót oparty na klawiszu *Ctrl* może zostać przypisany do dowolnej procedury, która nie korzysta z argumentów. Jeżeli na przykład do procedury o nazwie UpdateCustomerList zostanie przypisana kombinacja klawiszy *Ctrl+U*, jej wcisnięcie spowoduje wykonanie procedury.

W początkowej fazie rejestracji makra, okno dialogowe *Rejestrowanie makra* pozwala na przypisanie do makra skrótu klawiszowego (inaczej klawisz skrótu). W razie potrzeby jednak taki skrót możesz zdefiniować w dowolnej chwili. Aby utworzyć lub zmodyfikować skrót klawiszowy do procedury, powinieneś wykonać następujące polecenia:

1. Przejdź do okna programu Excel i przywołaj na ekran okno dialogowe *Makro* (możesz to zrobić, na przykład naciskając kombinację klawiszy *lewy Alt+F8*).
2. Wybierz żądaną procedurę z listy widocznej w oknie dialogowym *Makro*.
3. Naciśnij przycisk *Opcje* (patrz rysunek 7.2). Na ekranie pojawi się okno dialogowe *Opcje makra*.

Rysunek 7.2.

Okno dialogowe
Opcje makra pozwala
na przypisanie
do makra skrótu
klawiszowego
i wprowadzenie
opisu makra



4. Wprowadź wybrany znak w polu *Ctrl+*.

Uwaga: Wielkość znaku jest rozróżniana. Na przykład, jeżeli wprowadzisz literę s (mała litera), skrót klawiszowy będzie miał postać *Ctrl+S*. Jeżeli jednak wprowadzisz literę S (wielka litera), skrót klawiszowy będzie miał postać *Ctrl+Shift+S*.

5. Wprowadź opis (opcjonalnie), który zostanie wyświetlony w dolnej części okna dialogowego *Makro* po wybraniu procedury z listy.

6. Naciśnij przycisk *OK*, aby zamknąć okno dialogowe *Opcje makra*. Aby zamknąć okno dialogowe *Makro*, naciśnij przycisk *Zamknij*.



Po przypisaniu do procedury jednej z predefiniowanych kombinacji klawiszy stosowanych w Excelu pierwszeństwo będzie miała kombinacja zdefiniowana przez Ciebie. Przykładowo kombinacja *Ctrl+S* jest predefiniowanym skrótem klawiszowym Excela zapisującym aktywny skoroszyt. Jeżeli jednak taka sama kombinacja zostanie przypisana do procedury, jej wcisnięcie nie spowoduje już zapisania aktywnego skoroszytu.

Uruchamianie procedury za pomocą Wstążki

Wstążka, czyli nowy interfejs programu Excel, została wprowadzona w wersji Excel 2007. W tej wersji modyfikacja i dostosowywanie Wstążki do własnych potrzeb wymagało napisania odpowiedniego kodu XML, który tworzył nowe przyciski (lub inne formanty) na Wstążce i następnie przypisywał do nich odpowiednie makra. Zauważ, że modyfikacja Wstążki dokonywana w ten sposób, odbywała się niejako „na zewnątrz” Excela, stąd nie można tego było dokonać przy użyciu VBA.

Począwszy od wersji Excel 2010, użytkownicy mogą modyfikować zawartość Wstążki bezpośrednio z poziomu programu. Aby to zrobić, wystarczy kliknąć prawym przyciskiem myszy dowolne miejsce Wstążki i z menu podręcznego, które pojawi się na ekranie, wybrać polecenie *Dostosuj Wstążkę*. Teraz możesz po prostu umieścić na Wstążce nowy formant i przypisać do niego odpowiednie makro VBA. Taka operacja musi być jednak wykonana ręcznie — innymi słowy, nadal nie ma możliwości automatycznego dodawania nowych formantów do Wstążki za pomocą języka VBA.



Więcej szczegółowych informacji na temat dopasowywania Wstążki do własnych potrzeb znajdziesz w rozdziale 20.

Uruchamianie procedur za pośrednictwem niestandardowego menu podręcznego

Makro może być uruchamiane poprzez wybranie odpowiedniego polecenia z niestandardowego menu podręcznego, które pojawia się na ekranie, kiedy klikniesz prawym przyciskiem myszy wybrany obiekt lub zakres komórek. Napisanie odpowiedniej procedury w języku VBA, która będzie dodawała nowe polecenia do menu podręcznego, jest bardzo prostym zadaniem.



Więcej szczegółowych informacji na temat tworzenia niestandardowego menu podręcznego znajdziesz w rozdziale 21.

Wywoływanie procedury z poziomu innej procedury

Jedna z najczęściej stosowanych metod uruchamiania procedury polega na wywołaniu jej z poziomu innej procedury. Możesz tego dokonać na trzy sposoby:

- Wprowadź nazwę procedury oraz jej argumenty, o ile są wymagane (poszczególne argumenty powinny być od siebie oddzielone przecinkami).
- Użyj słowa kluczowego Call wraz z nazwą procedury i jej argumentami umieszczonymi w nawiasach okrągłych i oddzielonymi przecinkami (o ile procedura wymaga podania takich argumentów).
- Użyj metody Run obiektu Application. Metoda Run jest szczególnie użyteczna w sytuacji, kiedy musisz uruchomić procedurę, której nazwa została przypisana do zmiennej. Taką zmienną możesz przekazać jak argument wywołania metody Run.

Poniżej znajdziesz przykład prostej procedury Sub, która pobiera dwa argumenty i wyświetla na ekranie wartość ich iloczynu.

```
Sub AddTwo (arg1, arg2)
    MsgBox arg1 * arg2
End Sub
```

Trzy kolejne wyrażenia przedstawione poniżej ilustrują trzy różne sposoby wywołania procedury AddTwo i przekazania jej dwóch argumentów. Wyniki działania procedury, niezależnie od sposobu jej wywołania, będą zawsze takie same.

```
AddTwo 12, 6
Call AddTwo (12, 6)
Run "AddTwo", 12, 6
```

Mimo że użycie słowa kluczowego Call jest opcjonalne, niektórzy programiści zawsze je dodają, aby jednoznacznie podkreślić fakt wywoływania innej procedury.

Najlepszą okazją do użycia metody Run jest sytuacja, w której nazwa procedury jest przypisywana do zmiennej — a tak naprawdę to jedyny sposób na wywołanie takiej procedury. Poniżej przedstawiamy przykład bardzo uproszczonej procedury, która dobrze ilustruje właśnie taką sytuację. Procedura Main używa funkcji WeekDay języka VBA określającej dzień tygodnia (dni tygodnia ponumerowane są od 1 do 7, począwszy od niedzieli). Zmiennej SubToCall jest przypisywany łańcuch reprezentujący nazwę procedury. Metoda Run wywołuje następnie odpowiednią procedurę (WeekEnd lub NormalnyDzieńTygodnia).

```
Sub Main()
    Dim SubToCall As String
    Select Case WeekDay(Now)
        Case 1, 7: SubToCall = "WeekEnd"
        Case Else: SubToCall = "NormalnyDzieńTygodnia"
    End Select
    Application.Run SubToCall
End Sub

Sub Weekend()
    MsgBox "Mamy już weekend."
    ' [w tym miejscu znajduje się kod procedury wykonywany w czasie weekendu]
End Sub
```

```
Sub NormalnyDzieńTygodnia()
    MsgBox "Niestety to jeszcze nie weekend."
    '   [w tym miejscu znajduje się kod procedury wykonywany w dni robocze]
End Sub
```

Wywoływanie procedury zawartej w innym module

Jeżeli interpreter języka VBA nie może zlokalizować procedury wywoływanej w aktualnym module, poszuka procedur publicznych w innych modułach tego samego projektu.

Jeżeli chcesz wywołać procedurę prywatną z poziomu innej procedury, obie procedury muszą znajdować się w tym samym module.

W tym samym module nie mogą znajdować się dwie procedury o jednakowej nazwie, natomiast możesz utworzyć dwie procedury o identycznych nazwach ale umieszczone w różnych modułach. Istnieje możliwość zmuszenia interpretera języka VBA do wykonania procedury posiadającej *niejednoznaczną nazwę* (ang. *ambiguously named procedure*), czyli innej procedury o takiej samej nazwie zawartej w innym module. W tym celu przed nazwą procedury należy wstawić nazwę modułu i kropkę.

Dla przykładu założmy, że w modułach Module1 i Module2 zdefiniowano procedury o nazwie MySub. Jeżeli procedura z modułu Module2 ma wywoływać procedurę MySub znajdującą się w module Module1, należy użyć jednego z poniższych poleceń:

```
Module1.MySub
Call Module1.MySub
```

Jeżeli nie odróżnisz dwóch procedur o takich samych nazwach, zostanie wygenerowany komunikat błędu *Ambiguous name detected* (wykryto niejednoznaczną nazwę).

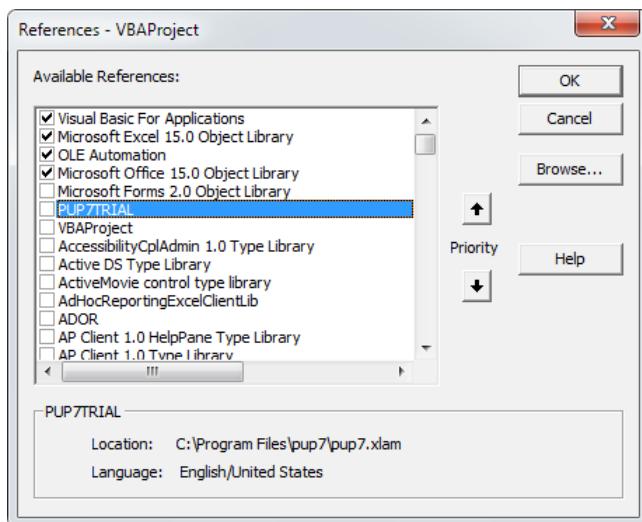
Wywoływanie procedury zawartej w innym skoroszycie

W niektórych przypadkach może być konieczne wywołanie z procedury drugiej procedury zdefiniowanej w innym skoroszycie. W takiej sytuacji masz dwie drogi postępowania: możesz utworzyć odwołanie do innego skoroszytu lub użyć metody Run i jawnie zdefiniować nazwę skoroszytu.

Aby utworzyć odwołanie do innego skoroszytu, z menu *Tools* edytora Visual Basic należy wybrać polecenie *References*. Na ekranie pojawi się okno dialogowe *References* (patrz rysunek 7.3), w którym będą widoczne wszystkie dostępne odwołania powiązane z wszystkimi otwartymi skorosztami. Aby stworzyć odwołanie do określonego skoroszytu, wystarczy go zaznaczyć, a następnie nacisnąć przycisk *OK*. Po utworzeniu odwołania można wywoływać procedury zawarte w skoroszycie, tak jakby znajdowały się w tym samym skoroszycie, co procedura wywołująca.

Skoroszyt, do którego jest wykonywane odwołanie, nie musi być otwarty — jest po prostu traktowany jak oddzielna biblioteka obiektów. Aby utworzyć odwołanie do skoroszytu, który nie jest otwarty, powinieneś w oknie dialogowym *References* nacisnąć przycisk *Browse*.

Rysunek 7.3.
Okno dialogowe
References umożliwia
utworzenie odwołania
do innego skoroszytu



Kiedy otworzysz skoroszyt zawierający odwołanie do innego skoroszytu, taki dodatkowy skoroszyt zostanie otwarty automatycznie.



Uwaga

Nazwy skoroszytów widoczne na liście odwołań są uporządkowane według nazw projektów otwartych w edytorze Visual Basic. Domyślnie każdy projekt początkowo otrzymuje nazwę **VBAPrj**, stąd lista może zawierać kilka identycznie nazwanych pozycji. Aby wyróżnić poszczególne projekty, powinieneś w oknie właściwości projektu zmienić jego nazwę. Aby to zrobić, zaznacz nazwę projektu, którą chcesz zmienić, i z menu głównego edytora VBE wybierz polecenia *Tools/<nazwa> Properties* (gdzie <nazwa> to aktualna nazwa projektu). Na ekranie pojawi się okno właściwości projektu. Przejdz na kartę *General* i w polu *Project Name* wpisz żądaną nazwę.

Lista odwołań wyświetlona w oknie dialogowym *References* zawiera również biblioteki obiektów i formanty ActiveX zarejestrowane w systemie. Skoroszyty Excela zawsze zawierają odwołania do następujących bibliotek obiektów:

- Visual Basic for Applications,
- Microsoft Excel 15.0 Object Library,
- OLE Automation,
- Microsoft Office 15.0 Object Library,
- Microsoft Forms 2.0 Object Library (opcjonalna, dołączana tylko wtedy, gdy w projekcie zastosowano formularze *UserForm*).



Uwaga

W strukturze projektu wyświetlanej w oknie *Project Explorer* edytora Visual Basic są też ujęte wszystkie utworzone wcześniej odwołania do innych skoroszytów. Znajdziesz je po rozwinięciu węzła o nazwie *References*.

Jeżeli utworzyłeś odwołanie do skoroszytu zawierającego procedurę *YourSub*, to w celu jej wywołania możesz użyć jednej z poniższych instrukcji:

```
YourSub
Call YourSub
```

Aby precyzyjnie zidentyfikować procedurę zawartą w innym skoroszycie, powinieneś dokładnie określić nazwę projektu, modułu i procedury, na przykład:

```
MyProject.MyModule.MySub
```

Oczywiście możesz również użyć słowa kluczowego Call. Oto przykład:

```
Call MyProject.MyModule.MySub
```

Kolejna metoda wywoływania procedury zawartej w innym, otwartym skoroszycie polega na zastosowaniu metody Run obiektu Application. Metoda nie wymaga tworzenia odwołania, ale skoroszyt zawierający wywoływaną procedurę musi być otwarty. Poniższa instrukcja wykonuje procedurę Consolidate zawartą w skoroszycie o nazwie *budżet makra.xlsxm*:

```
Application.Run "budżet makra.xlsxm"!Consolidate"
```

Zwróć uwagę, że nazwa skoroszytu została ujęta w znaki apostrofu. Taka składnia jest niezbędna tylko w sytuacji, kiedy nazwa pliku zawiera jedną lub więcej spacji. Poniżej znajdziesz przykład wywołania takiej procedury ze skoroszytu, którego nazwa nie zawiera żadnych spacji:

```
Application.Run "budgetmacros.xlsxm!Consolidate"
```

Uruchamianie procedury poprzez kliknięcie obiektu

Excel udostępnia wiele różnego rodzaju obiektów, które mogą być umieszczane na zwykłym arkuszu lub arkuszu wykresu. Do takich obiektów możesz również przypisać makro. Obiekty, do których możesz przypisać makro, dzielą się na kilka kategorii:

- Formanty ActiveX
- Formanty formularza
- Obiekty wstawiane do arkusza (kształty, obiekty typu SmartArt i WordArt, wykresy i obrazy)



Jeżeli przejdziesz na kartę *DEVELOPER* i naciśniesz przycisk *Wstaw* znajdujący się w grupie opcji *Formanty*, na ekranie zostanie wyświetlona lista rozwijana zawierająca dwa rodzaje formantów, które możesz umieścić na arkuszu: formanty formularza oraz formanty ActiveX. Formanty ActiveX są podobne do formantów używanych na formularzach *UserForm*. Formanty formularza zostały początkowo zaprojektowane dla Excela w wersji 5 oraz 95, ale pomimo upływu czasu nadal mogą być używane w nowszych wersjach Excela (a czasami są wręcz bardziej preferowanym rozwiązaniem).

W przeciwieństwie do formantów formularzy formanty ActiveX nie mogą być użyte do uruchomienia dowolnie wybranego makra. Formanty ActiveX uruchamiają tylko makra o specjalnie dobranych nazwach. Na przykład: jeżeli umieścisz na arkuszu przycisk (formant CommandButton) i nadasz mu nazwę *CommandButton1*, kliknięcie tego przycisku spowoduje automatyczne uruchomienie makra o nazwie *CommandButton1_Click*. Kod tego makra musi być umieszczony w module kodu arkusza, do którego taki formant został wstawiony.

Więcej szczegółowych informacji na temat używania formantów na arkuszach znajdziesz w rozdziale 11.

Dlaczego wywołujemy inne procedury?

Jeżeli jesteś początkującym programistą, możesz zastanawiać się, dlaczego ktoś w ogóle wywołuje jedną procedurę z poziomu innej procedury. Możesz zapytać: „Dlaczego po prostu nie umieścić całego kodu wywoływanej procedury w procedurze wywołującej?”.

Jednym z powodów jest przejrzystość kodu źródłowego. Im prostszy kod programu, tym łatwiejsze będzie zarządzanie i modyfikacja kodu. Mniejsze procedury są prostsze do przeanalizowania, a następnie sprawdzenia. Kod poniżej procedury, która jedynie wywołuje inne procedury, jest prosty, przejrzysty i łatwy do zrozumienia:

```
Sub Main()
    Call GetUserOptions
    Call ProcessData
    Call CleanUp
    Call CloseItDown
End Sub
```

Wywoływanie innych procedur pozwala również na eliminowanie niepotrzebnej nadmiarowości kodu. Założmy, że chcesz wykonać identyczną operację w dziesięciu różnych miejscach programu. Zamiast w dziesięciu miejscach wstawiać identyczny kod źródłowy, możesz napisać procedurę wykonującą taką operację, a następnie po prostu wywołać ją 10 razy w odpowiednich miejscach programu. Co więcej, jeżeli później zajdzie potrzeba zmiany kodu takiej procedury, będziesz musiał go zmienić tylko w jednym miejscu, a nie w dziesięciu.

Możesz też utworzyć kilka często stosowanych procedur ogólnego zastosowania. Po umieszczeniu ich w oddzielnym module możesz go zimportować do aktualnie realizowanego projektu, a następnie wywoływać wymagane procedury, co jest znacznie prostsze niż kopowanie i wklejanie kodu źródłowego do nowych procedur.

Tworzenie większej liczby niewielkich procedur zamiast jednej dużej jest powszechnie uważane za dobrą praktykę programistyczną. Modułowość nie tylko upraszcza pracę, ale też ułatwia życie osobom, które później zajmują się utworzonym przez Ciebie kodem źródłowym.

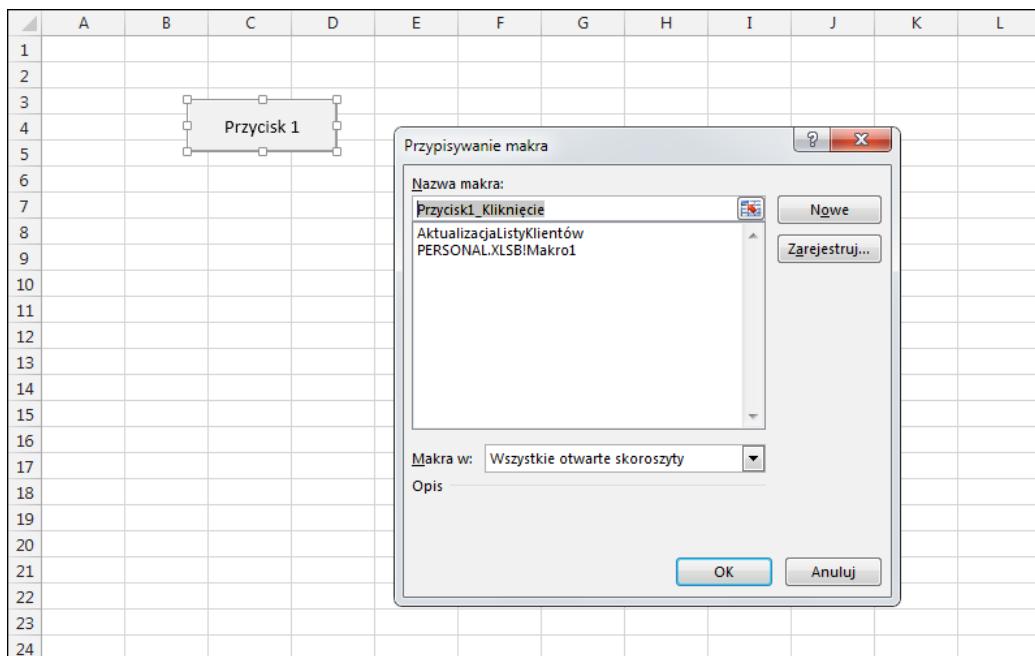
Aby do obiektu Przycisk, będącego jednym z formantów formularza, przypisać wybraną procedurę, powinieneś wykonać następujące polecenia:

1. Przejdz na kartę *DEVELOPER*, naciśnij przycisk *Wstaw*, znajdujący się w grupie opcji *Formanty*, a następnie kliknij formant przycisku.
2. Aby wstawić przycisk do arkusza, kliknij dowolne miejsce arkusza.

Zamiast tego możesz również przy użyciu myszy „narysować” przycisk, odpowiednio dopasowując jego rozmiary.

Po wstawieniu przycisku do arkusza Excel automatycznie przywołała na ekran okno dialogowe *Przypisywanie makra* (patrz rysunek 7.4). Domyślnie Excel sugeruje przypisanie makra o nazwie opartej na nazwie samego przycisku (*Przycisk1_Kliknięcie*).

3. Wybierz z listy lub wpisz nazwę makra, które chcesz przypisać do przycisku, a następnie naciśnij przycisk *OK*.



Rysunek 7.4. Przypisywanie makra do przycisku

Przypisane do formantu makro możesz zmienić w dowolnej chwili. Aby to zrobić, kliknij formant prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Przypisz makro*.

Aby przypisać makro do kształtu, obiektu typu SmartArt lub WordArt, wykresu czy obrazka, kliknij obiekt prawym przyciskiem myszy, a następnie z menu podręcznego wybierz polecenie *Przypisz makro*.

Aby przypisać makro do wykresu osadzonego na arkuszu, naciśnij i przytrzymaj klawisz *Ctrl*, kliknij wykres (tak, by go zaznaczyć jako obiekt), a następnie kliknij wykres prawym przyciskiem myszy i z menu podręcznego, które pojawi się na ekranie, wybierz polecenie *Przypisz makro*.

Wykonywanie procedury po wystąpieniu określonego zdarzenia

Procedura może zostać wykonana po wystąpieniu określonego zdarzenia. Przykładami zdarzeń są otwarcie i zapisanie skoroszytu, wprowadzenie do niego danych, kliknięcie formantu ActiveX reprezentującego przycisk (ang. CommandButton) i wiele innych. Procedura wykonywana po wystąpieniu zdarzenia jest nazywana procedurą *obsługi zdarzenia* (ang. *event handler procedure*). Procedury obsługi zdarzeń charakteryzują się następującymi cechami:

- Posiadają specjalne nazwy złożone z nazwy obiektu, znaku podkreślenia i nazwy zdarzenia (na przykład procedura wykonywana po otwarciu skoroszytu nosi nazwę *Workbook_Open*).

- Są przechowywane w module kodu, powiązanym z określonym obiektem (na przykład ThisWorkbook czy Sheet1).



Procedury obsługi zdarzeń zostały szczegółowo omówione w rozdziale 17.

Uruchamianie procedury z poziomu okna Immediate

Procedura może również zostać uruchomiona bezpośrednio poprzez wprowadzenie jej nazwy w oknie *Immediate* edytora Visual Basic. Jeżeli okno *Immediate* nie jest widoczne, powinieneś przywołać je na ekran, naciskając kombinację klawiszy *Ctrl+G*. Polecenia języka VBA wpisywane w oknie *Immediate* są wykonywane od razu po zakończeniu wprowadzania i naciśnięciu klawisza *Enter*. Aby uruchomić wybraną procedurę, wystarczy w oknie *Immediate* wprowadzić jej nazwę z uwzględnieniem wymaganych argumentów i nacisnąć klawisz *Enter*.

Metoda ta może być bardzo przydatna podczas tworzenia procedury, ponieważ w dowolnych miejscach tworzonej procedury możesz umieszczać polecenia wyświetlające w oknie *Immediate* na przykład częstekowe wyniki obliczeń czy wartości badanych zmiennych i tym samym na bieżąco testować działanie procedury oraz poszczególnych poleceń. Poniższa przykładowa procedura demonstruje tę metodę:

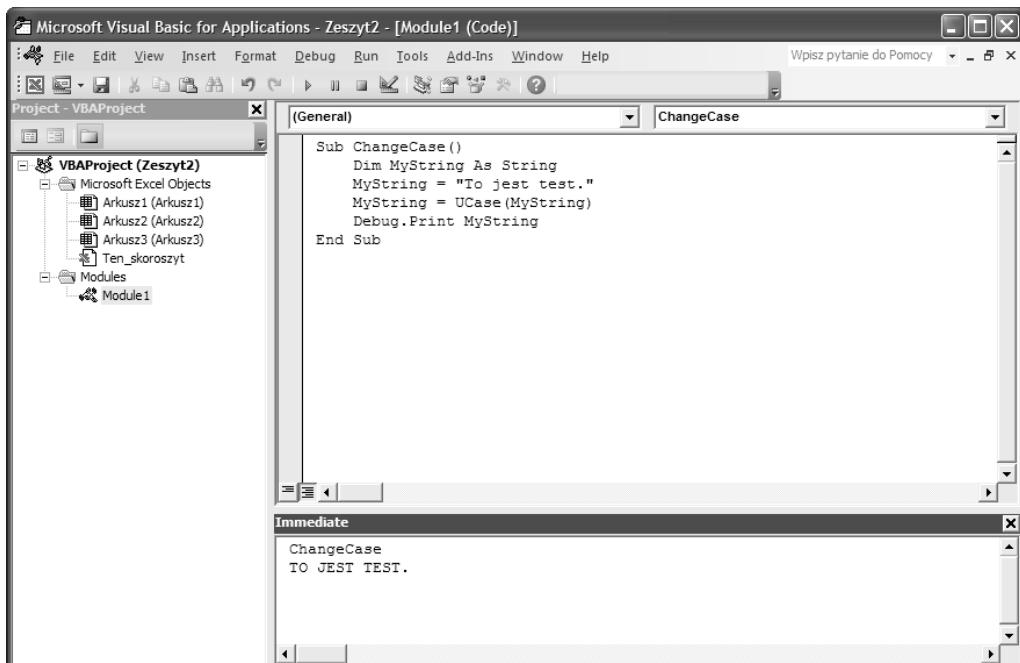
```
Sub ChangeCase()
    Dim MyString As String
    MyString = "To jest test."
    MyString = UCASE(MyString)
    Debug.Print MyString
End Sub
```

Na rysunku 7.5 pokazano, co się stanie, kiedy w oknie *Immediate* wpiszesz nazwę procedury *ChangeCase* — natychmiast zostanie wyświetlony wynik wygenerowany przez instrukcję *Debug.Print*.

Przekazywanie argumentów procedurom

Argumenty pozwalają na przekazywanie danych, których procedura używa w trakcie działania. Argumentami wywołania procedur mogą być:

- Zmienne
- Stałe
- Wyrażenia
- Tablice
- Obiekty



Rysunek 7.5. Wykonanie procedury poprzez wprowadzenie jej nazwy w oknie Immediate

Sposób wykorzystywania argumentów przez procedury jest bardzo zbliżony do sposobu ich wykorzystywania w funkcjach arkuszowych:

- Procedury mogą, ale nie muszą mieć żadnych argumentów wywołania.
- Procedury mogą wymagać podawania stałej, ściśle określonej liczby argumentów.
- Procedury mogą akceptować nieokreślona liczbę argumentów.
- Procedury mogą posiadać zarówno argumenty wymagane, jak i opcjonalne.
- Wszystkie argumenty procedury mogą być opcjonalne.

Przykładowo, niektóre funkcje arkuszowe programu Excel (np. funkcja LOS) nie wymagają podawania żadnych argumentów, inne (np. LICZ.JEŻELI) wymagają podania dwóch argumentów, a jeszcze inne (np. SUMA) mogą używać nieokreślonej liczby argumentów (maksymalnie 255). Istnieją też funkcje arkusza, których argumenty są opcjonalne. Przykładowo funkcja PMT może korzystać z pięciu argumentów (trzy są wymagane, natomiast dwa opcjonalne).

Większość procedur, z którymi miałeś do tej pory do czynienia w tej książce, była deklarowana bez argumentów. Ich deklaracja zawierała jedynie słowo kluczowe Sub, nazwę procedury i parę pustych nawiasów okrągłych. Puste nawiasy wskazują, że procedura nie przyjmuje argumentów.

W poniższym przykładzie procedura Main trzykrotnie wywołuje procedurę ProcessFile (instrukcja Call jest zawarta w pętli For ... Next). Jednak przed wywołaniem procedury ProcessFile tworzona jest trzyelementowa tablica. Wewnątrz pętli każdy element tablicy

staje się argumentem wywoływanej procedury. Procedura ProcessFile pobiera jeden argument o nazwie TheFile, który znajduje się w nawiasach okrągłych instrukcji Sub. Po zakończeniu działania procedury ProcessFile jako następna jest wykonywana instrukcja znajdująca się za instrukcją Call.

```
Sub Main()
    Dim File(1 To 3) As String
    Dim i as Integer
    File(1) = "dept1.xlsx"
    File(2) = "dept2.xlsx"
    File(3) = "dept3.xlsx"
    For i = 1 To 3
        Call ProcessFile(File(i))
    Next i
End Sub

Sub ProcessFile(TheFile)
    Workbooks.Open FileName := TheFile
    ' [...]w tym miejscu może znajdować się pozostała część kodu...]
End Sub
```

Oczywiście procedurze można też przekazywać *literały* (literały nie są zmiennymi). Oto przykład:

```
Sub Main()
    Call ProcessFile("budżet.xlsx")
End Sub
```

Argumenty można przekazywać do procedury na dwa sposoby:

- **Przez odwołanie.** Przekazywanie argumentu przez odwołanie (metoda domyślna) polega po prostu na przesłaniu procedurze adresu zmiennej w pamięci. W tym przypadku procedura, zmieniając wartość argumentu, bezpośrednio modyfikuje wartość zmiennej.
- **Przez wartość.** Przekazywanie argumentu przez wartość polega na przesłaniu kopii oryginalnej zmiennej, stąd w tym przypadku modyfikacje argumentu przez procedurę nie wpływają na wartość oryginalnej zmiennej.

Poniższy przykład demonstruje to zagadnienie. Argument procedury Process jest jej przekazywany przez odwołanie (metoda domyślna). Po przypisaniu zmiennej MyValue wartości 12 procedura Main wywołuje procedurę Process i przekazuje jej jako argument zmienną MyValue. Procedura Process mnoży wartość otrzymanego argumentu (zmienna YourValue) przez 10. Po zakończeniu działania procedury Process sterowanie powraca do procedury Main i funkcja MsgBox wyświetla wartość zmiennej MyValue wynoszącą 120.

```
Sub Main()
    Dim MyValue As Integer
    MyValue = 12
    Call Process(MyValue)
    MsgBox MyValue
End Sub

Sub Process(YourValue)
    YourValue = YourValue * 10
End Sub
```

Jeżeli nie chcesz, aby wywoływana procedura modyfikowała jakąkolwiek zmienną przekazywaną jako argument, możesz zmienić definicję argumentów tak, aby były przekazywane przez *wartość*, a nie przez *odwołanie*. W tym celu przed nazwą argumentu należy wstawić słowo kluczowe `ByVal`. Dzięki takiemu rozwiązaniu procedura będzie przetwarzala kopię przekazanej zmiennej, a nie jej oryginal. Przykładowo w poniższej procedurze modyfikacja zmiennej `YourValue` procedury `Process` nie ma wpływu na zmienną `MyValue` procedury `Main`. W efekcie funkcja `MsgBox` wyświetla wartość 12, a nie 120.

```
Sub Process(ByVal YourValue)
    YourValue = YourValue * 10
End Sub
```

W większości przypadków wystarczy zastosowanie domyślnej metody przekazywania argumentów przez odwołanie. Jeżeli jednak procedura musi użyć danych przekazywanych jej przy użyciu argumentu i absolutnie zależy Ci na zachowaniu oryginalnych danych w niezmienionej postaci, należy zastosować przekazywanie przez wartość.

Argumenty procedury mogą być zarówno przekazywane przez wartość jak i odwołanie. Argumenty, przed którymi umieszczone słowo kluczowe `ByVal`, są przekazywane przez wartość, natomiast wszystkie pozostałe — przez odwołanie.



Jeżeli jako argumentu wywołania procedury chcesz użyć zmiennej typu zdefiniowanego przez użytkownika, musisz ją przekazywać przez odwołanie. Próba przekazania takiej zmiennej przez wartość spowoduje wygenerowanie błędu.

Użycie zmiennych publicznych a przekazywanie argumentów do procedury

W rozdziale 6. pokazano, w jaki sposób wszystkim procedurom modułu udostępnić zmienną deklarowaną przy użyciu słowa kluczowego `Public` (umieszczonego na początku modułu). W niektórych przypadkach podczas wywoływania innej procedury, zamiast przekazywania zmiennej jako argumentu używa się zmiennej publicznej.

Przykładowo poniższa procedura przekazuje wartość zmiennej `MonthVal` procedurze `ProcessMonth`:

```
Sub MySub()
    Dim MonthVal as Integer
    '...w tym miejscu znajduje się kod procedury...
    MonthVal = 4
    Call ProcessMonth(MonthVal)
    '...w tym miejscu znajduje się kod procedury...
End Sub
```

Alternatywnym rozwiązaniem, które nie wykorzystuje przekazywania argumentu może być taka procedura:

```
Public MonthVal as Integer

Sub MySub()
    '...w tym miejscu znajduje się kod procedury...
    MonthVal = 4
    Call ProcessMonth2
    '...w tym miejscu znajduje się kod procedury...
End Sub
```

Ponieważ w drugim wariantie procedury zmieniona `MonthVal` jest publiczna, procedura `ProcessMonth2` może jej użyć, dzięki czemu wyeliminowana zostaje konieczność przekazywania argumentu.

Ponieważ w poprzednich przykładach nie zadeklarowaliśmy typu danych dla żadnego argumentu, wszystkie zmienne używały typu Variant. Jednak procedura pozwala na zdefiniowanie typu przekazywanych danych bezpośrednio w liście argumentów. Poniżej zamieszczamy nagłówek procedury Sub używającej dwóch argumentów różnego typu. Pierwszy z nich został zadeklarowany jako typ Integer, natomiast drugi jako typ String:

```
Sub Process(Iterations As Integer, TheFile As String)
```

Przekazując argumenty procedurze, należy pamiętać, aby ich wartości były zgodne z typami danych zdefiniowanymi dla poszczególnych argumentów. Jeżeli na przykład w poprzednim przykładzie wywołasz procedurę Process i jako pierwszy argument przekażesz jej zmienną typu String, zostanie wygenerowany błąd *ByRef argument type mismatch* (niezgodność typów dla argumentu przekazywanego przez odwołanie).



Argumenty mogą być stosowane zarówno w przypadku procedur Sub, jak i Function. W praktyce argumenty częściej są stosowane w przypadku funkcji (procedur typu Function). W rozdziale 8., który został w całości poświęcony funkcjom, znajdziesz wiele dodatkowych przykładów objaśniających sposoby definiowania i przekazywania argumentów, włączając w to również argumenty opcjonalne.

Metody obsługi błędów

W trakcie tworzenia i uruchamiania procedur języka VBA mogą — i z pewnością wystąpią — różne błędy. Należy do nich zaliczyć *błędy składni* (które muszą zostać usunięte, zanim będziesz mógł uruchomić daną procedurę lub program) i *błędy wykonania* (które pojawiają się dopiero w trakcie działania procedury). W tym podrozdziale zajmiemy się omówieniem błędów wykonania.



Aby procedury zawierające obsługę błędów mogły poprawnie działać, musisz wyłączyć opcję *Break on All Errors*. Aby to zrobić, wybierz z menu *Tools* edytora VBE polecenie *Options* i w oknie dialogowym *Options* przejdź na kartę *General*. Po włączeniu opcji *Break on All Errors* VBA ignoruje procedury obsługujące błędy. W praktyce najczęściej będziesz korzystał z opcji *Break on Unhandled Errors*.

Zazwyczaj błąd wykonania powoduje przerwanie działania programu VBA i pojawienie się na ekranie okna dialogowego wyświetlającego numer błędu i jego opis. Użytkownik dobrze napisanej aplikacji nie powinien mieć do czynienia z takimi komunikatami. Poprawne napisane aplikacje zawierają kod obsługujący błędy, który je przechwyci i podejmie odpowiednie działania, a w najgorszym razie wyświetli na ekranie bardziej zrozumiałą komunikat od tego, który domyślnie jest generowany przez VBA.



W dodatku B zawarto wszystkie kody błędów języka VBA wraz z ich opisem.

Przechwytywanie błędów

Aby określić, co się stanie po wystąpieniu błędu, powinieneś użyć instrukcji *On Error*. Do wyboru masz w zasadzie dwie możliwości:

- **Zignorowanie błędu i zezwolenie VBA na kontynuowanie.** Program może później sprawdzić obiekt Err, aby stwierdzić, jaki wystąpił błąd i w razie konieczności wykonać odpowiednią operację.
- **Przeskoczenie do specjalnej części kodu obsługującej błędy i wykonanie odpowiedniej operacji.** Segment kodu obsługi błędów znajduje się na końcu procedury i oznaczony jest odpowiednią etykietą.

Aby po wystąpieniu błędu procedura VBA była dalej wykonywana, powinieneś umieścić w niej następującą instrukcję:

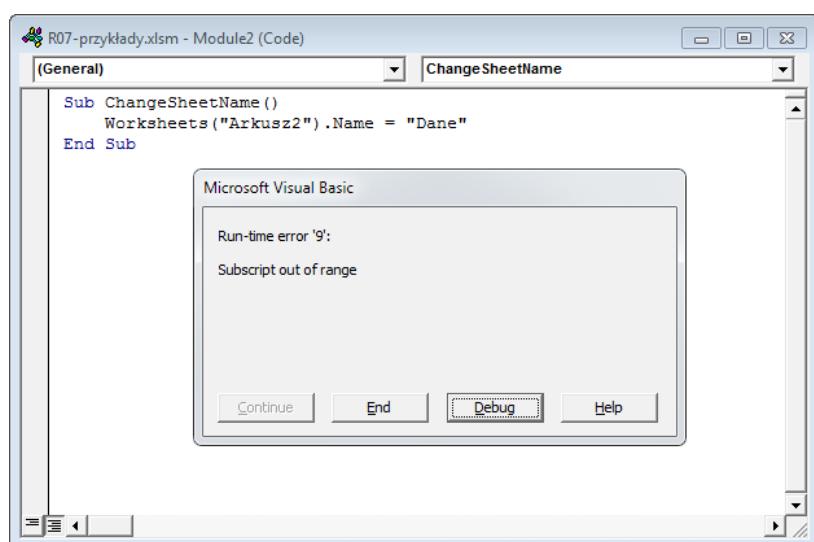
```
On Error Resume Next
```

Niektóre błędy są mało istotne, nie mają wpływu na poprawne działanie programu i po prostu bezpiecznie mogą zostać zignorowane. Zazwyczaj jednak warto wiedzieć, jaki błąd wystąpił. Aby to sprawdzić, powinieneś użyć obiektu Err, który pozwala na zidentyfikowanie numeru błędu. Do wyświetlenia tekstu odpowiadającego właściwości Err.Number możesz użyć funkcji Error języka VBA. Właściwość Number jest domyślną właściwością obiektu Err. Na przykład poniższa instrukcja wyświetla identyczną informację, jak standardowy komunikat błędu języka VBA (numer błędu i jego opis):

```
MsgBox "Error " & Err & ": " & Error(Err.Number)
```

Na rysunku 7.6 pokazano komunikat błędu języka VBA, a na rysunku 7.7 ten sam błąd wyświetlony w oknie dialogowym programu Excel. Bardziej zrozumiałą opis może oczywiście sprawić, że komunikat błędu będzie dla końcowych użytkowników przedstawiał większą wartość.

Rysunek 7.6.
Komunikaty błędów
języka VBA nie zawsze
są przyjazne
dla użytkownika

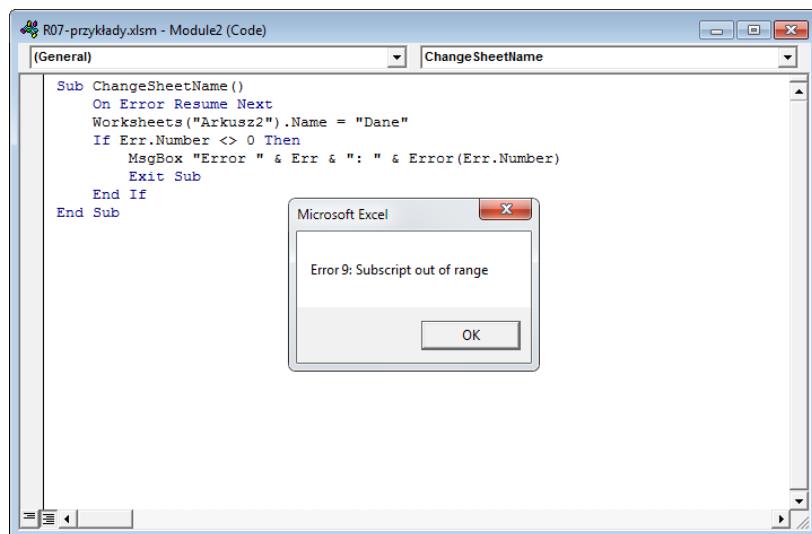


Odwołanie się do obiektu Err jest równoznaczne użyciu jego właściwości o nazwie Number, stąd poniższe dwie instrukcje dają taki sam efekt:

```
MsgBox Err  
MsgBox Err.Number
```

Rysunek 7.7.

Możesz utworzyć okno komunikatu zawierające kod błędu i jego opis



Instrukcja `On Error` może też posłużyć do zlokalizowania w procedurze miejsca, do którego ma zostać przekazane sterowanie po wystąpieniu błędu. W celu oznaczenia tego miejsca możesz użyć etykiety. Oto przykład:

```
On Error GoTo ErrorHandler
```

Przykłady kodu źródłowego obsługującego błędy

W pierwszym przykładzie przedstawiliśmy błąd, który bez żadnych obaw może zostać zignorowany. Metoda `SpecialCells` zaznacza komórki spełniające określone warunki.



Działanie metody `SpecialCells` odpowiada przejściu na kartę **NARZĘDZIA GŁÓWNE**, wybraniu polecenia **Znajdź i zaznacz** (znajdującego się w grupie opcji *Edycja*) i następnie wybraniu z menu podręcznego polecenia **Przejjdź do — specjalnie**. W oknie dialogowym **Przechodzenie do — specjalnie** znajdziesz szereg opcji. Za ich pomocą możesz wybrać na przykład tylko takie komórki, w których znajdują się stałe wartości liczbowe (nie formuły).

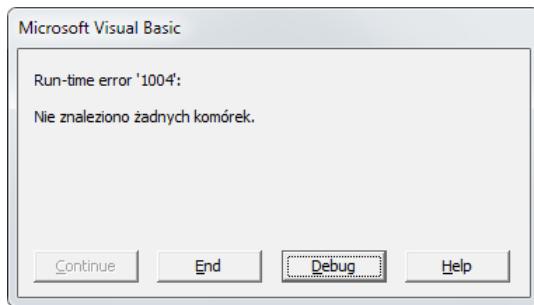
W poniższym przykładzie metoda `SpecialCells` zaznacza wszystkie komórki aktualnego zakresu, w których znajduje się formuła zwracająca wartość liczbową. Standardowo, gdy żadna z zaznaczonych komórek nie spełnia kryterium, VBA generuje komunikat o wystąpieniu błędu, przedstawiony na rysunku 7.8.

```
Sub SelectFormulas()
    Selection.SpecialCells(xlFormulas, xlNumbers).Select
    ' [...] w tym miejscu znajduje się kod procedury...
End Sub
```

W kodzie przedstawionym poniżej użyte zostało polecenie `On Error Resume Next`, które zapobiega pojawiению się komunikatu o błędzie.

Rysunek 7.8.

Metoda `SpecialCells` generuje taki błąd, jeżeli nie znajdzie żadnych komórek



```
Sub SelectFormulas2()
    On Error Resume Next
    Selection.SpecialCells(xlFormulas, xlNumbers).Select
    On Error GoTo 0
    ' [...]w tym miejscu znajduje się kod procedury...
End Sub
```

Polecenie `On Error GoTo 0` przywraca standardowy tryb obsługi błędów dla pozostałych instrukcji procedury.

Poniższa procedura używa dodatkowego polecenia do określenia, czy wystąpił błąd. Jeżeli tak, użytkownik zostaje o tym poinformowany za pomocą specjalnego komunikatu:

```
Sub SelectFormulas3()
    On Error Resume Next
    Selection.SpecialCells(xlFormulas, xlNumbers).Select
    If Err = 1004 Then MsgBox "Nie znaleziono komórek zawierających formułę."
    On Error GoTo 0
    ' [...]w tym miejscu znajduje się kod procedury...
End Sub
```

Jeżeli wartość właściwości `Err` jest różna od zera, oznacza to, że wystąpił błąd. Polecenie `If` sprawdza, czy właściwość `Err.Number` ma wartość 1004, i jeżeli tak, wyświetla na ekranie odpowiedni komunikat. Jak widać, w naszym przykładzie sprawdzamy, czy wystąpił jeden, ściśle określony błąd. Aby sprawdzić, czy w ogóle wystąpił jakiś błąd, możesz użyć następującego polecenia:

```
If Err <> 0 Then MsgBox "Wystąpił błąd!"
```

Kolejny przykład demonstruje obsługę błędów przy użyciu skoku do etykiety:

```
Sub ErrorDemo()
    On Error GoTo Handler
    Selection.Value = 123
    Exit Sub
Handler:
    MsgBox "Przypisanie wartości do zaznaczenia nie jest możliwe."
End Sub
```

Procedura próbuje przypisać wartość do aktualnego zaznaczenia. Jeżeli wystąpi błąd (np. odpowiedni zakres nie został zaznaczony lub arkusz jest chroniony), instrukcja przypisania wygeneruje błąd. Po wystąpieniu błędu instrukcja `On Error` spowoduje przekazanie sterowania do kodu oznaczonego etykietą `Handler`. Zwróć uwagę na użycie przed etykietą

instrukcji Exit Sub, która zapobiega wykonaniu kodu obsługującego błędy, w sytuacji gdy one nie wystąpią. Jeżeli ta instrukcja zostanie pominięta, komunikat błędu będzie wyświetlany nawet wtedy, gdy błąd nie wystąpi.

Czasami w celu uzyskania żądanych informacji można skorzystać z samego faktu wystąpienia błędu. W poniższym przykładzie dokonujemy prostego sprawdzenia, czy otwarto określony skoroszyt. Nasza procedura nie posiada żadnej obsługi błędów.

```
Sub CheckForFile()
    Dim FileName As String
    Dim FileExists As Boolean
    Dim book As Workbook
    FileName = "BUDŻET.XLSX"
    FileExists = False

    ' Sprawdzenie wszystkich skoroszytów
    For Each book In Workbooks
        If UCASE(book.Name) = FileName Then FileExists = True
    Next book

    ' Wyświetlenie odpowiedniego komunikatu
    If FileExists Then
        MsgBox FileName & " jest otwarty."
    Else
        MsgBox FileName & " nie jest otwarty."
    End If
End Sub
```

W procedurze zastosowano pętlę For Each ... Next, która sprawdza wszystkie obiekty z kolekcji Workbooks. Znalezienie otwartego skoroszytu spowoduje ustawienie dla zmiennej FileExists wartości True. Na końcu jest wyświetlany komunikat informujący użytkownika o tym, czy skoroszyt jest otwarty.

Poprzednia procedura może zostać zmodyfikowana tak, aby wykorzystywała obsługę błędów do stwierdzenia, czy plik został otwarty. W poniższym przykładzie instrukcja On Error Resume Next powoduje, że VBA ignoruje wszystkie błędy. Kolejna instrukcja próbuje odwołać się do skoroszytu poprzez przypisanie go do zmiennej obiektowej (przy użyciu słowa kluczowego Set). Jeżeli skoroszyt jest zamknięty, zostanie wygenerowany błąd. Struktura If ... Then ... Else sprawdza wartość właściwości obiektu Err i wyświetla odpowiedni komunikat. W tej wersji procedury nie używamy pętli, przez co jest ona nieco bardziej efektywna niż jej poprzednia wersja.

```
Sub CheckForFile()
    Dim FileName As String
    Dim x As Workbook
    FileName = "BUDŻET.XLSX"
    On Error Resume Next
    Set x = Workbooks(FileName)
    If Err = 0 Then
        MsgBox FileName & " jest otwarty."
    Else
        MsgBox FileName & " nie jest otwarty."
    End If
    On Error GoTo 0
End Sub
```



W rozdziale 9. znajdziesz kilka dodatkowych przykładów procedur obsługi błędów.

Praktyczny przykład wykorzystujący procedury Sub

W tym rozdziale zajmujemy się podstawowymi zagadnieniami związanymi z tworzeniem procedur Sub, ale trzeba przyznać, że większość przykładów, jakie zostały zaprezentowane do tej pory, była raczej mało przydatna w praktyce. W tym podrozdziale przedstawimy konkretny przykład demonstrujący w praktyczny sposób zagadnienia omawiane w tym i w poprzednich dwóch rozdziałach.

W tym podrozdziale omówimy krok po kroku proces tworzenia bardzo użytecznego, praktycznego narzędzia. Zademonstrujemy tutaj również *proces* analizy problemu, a następnie rozwiązywania go przy użyciu języka VBA. Kod programu został napisany z myślą o początkujących programistach używających języka VBA, stąd oprócz samego kodu źródłowego znajdują się tu również obszerne wyjaśnienia, jak zdobyć informacje niezbędne do jego utworzenia.



Skoroszyt z pełnym kodem aplikacji (*Sortowanie arkuszy.xlsxm*) znajdziesz na stronie internetowej tej książki (patrz <http://www.helion.pl/ksiazki/e13pvw.htm>).

W sieci

Cel

Celem przykładu jest zaprojektowanie narzędzia modyfikującego skoroszyt poprzez sortowanie jego arkuszy w kolejności alfabetycznej (Excel nie posiada takiej funkcji). Jeżeli masz tendencję do tworzenia wieloarkuszowych skoroszytów, to wiesz, że zlokalizowanie jednego z nich może być trudnym zadaniem. Jeżeli arkusze zostaną posortowane w kolejności alfabetycznej, o wiele łatwiej będzie znaleźć żądaną arkusz.

Wymagania projektowe

Od czego zacząć? Jednym ze sposobów na rozpoczęcie pracy jest przygotowanie listy wymagań dotyczących aplikacji. W trakcie projektowania możesz sprawdzać tę listę, aby mieć pewność, że uwzględniałeś wszystkie założenia.

Poniżej przedstawiono listę podstawowych wymagań, którą przygotowaliśmy dla naszej przykładowej aplikacji.

- Aplikacja powinna sortować arkusze (zwykłe i wykresu) aktywnego skoroszytu w kolejności alfabetycznej.
- Powinna być prosta do uruchomienia.
- Powinna być zawsze dostępna (innymi słowy, użytkownik nie może być zmuszony do otwierania skoroszytu, aby móc skorzystać z tego narzędzia).

- Aplikacja powinna działać poprawnie w przypadku każdego otwartego skoroszytu.
- Aplikacja powinna obsługiwać pojawiające się błędy i nie powinna wyświetlać na ekranie żadnych tajemniczych komunikatów o błędach VBA.

Co już wiesz

Często najtrudniejszą fazą projektu jest stwierdzenie, od czego rozpocząć jego realizację. W naszym przypadku rozpoczniemy od zebrania takich informacji o Excelu, które mogą mieć jakiś związek z wymaganiami projektowanej aplikacji:

- Excel nie posiada polecenia sortującego arkusze, zatem mamy pewność, że nie będziemy ponownie wymyślać koła.
- Nie możemy utworzyć takiego makra za pośrednictwem rejestratora makr, aczkolwiek może się on przydać do zdobywania niektórych kluczowych informacji.
- Karty arkuszy mogą być łatwo przenoszone za pomocą metody przeciagnij i upuść.

Notatka dodatkowa: Pamiętaj, aby uruchomić rejestrator makr i sprawdzić, jaki kod źródłowy zostanie wygenerowany po przeciągnięciu karty arkusza w inne miejsce.

- Excel dysponuje poleceniem *Przenieś lub kopij*, które jest dostępne w menu podręcznym po kliknięciu karty arkusza prawym przyciskiem myszy. Czy zarejestrowanie makra z użyciem tego polecenia spowoduje wygenerowanie innego kodu niż w przypadku przenoszenia arkusza metodą przeciagnij i upuść?
- Musimy znać liczbę arkuszy aktywnego skoroszytu. Informację taką możemy uzyskać przy użyciu języka VBA.
- Musimy znać nazwy wszystkich arkuszy. I tym razem odpowiednie informacje możemy uzyskać przy użyciu języka VBA.
- Excel posiada polecenie sortujące dane zawarte w komórkach arkusza.

Notatka dodatkowa: Sprawdzić, czy można przenieść nazwy arkuszy do zakresu i użyć tego polecenia, lub sprawdzić, czy język VBA dysponuje metodą sortującą z której można skorzystać.

- Okno dialogowe *Opcje makra* pozwala na łatwe przypisanie skrótu klawiszowego do makra.
- Jeżeli makro zostanie zapisane w skoroszycie makr osobistych, będzie zawsze dostępne.
- Będziemy potrzebować metody umożliwiającej testowanie aplikacji w trakcie jej tworzenia. Na pewno nie będziemy jej testować przy użyciu tego samego skoroszytu, w którym wprowadzamy kod źródłowy.

Notatka dodatkowa: Utworzyć dodatkowy arkusz roboczy, przeznaczony do testowania kodu aplikacji.

- Jeżeli utworzymy poprawny kod źródłowy, VBA nie wygeneruje żadnych błędów.

Notatka dodatkowa: Pobożne życzenia...

Podejście do zagadnienia

Co prawda nadal nie mamy dokładnego planu działania, ale możemy opracować jego wstępny zarys opisujący ogólne zadania, które musimy wykonać:

1. Identyfikacja aktywnego skoroszytu.
2. Pobranie listy nazw arkuszy skoroszytu.
3. Policzenie arkuszy.
4. Posortowanie nazw arkuszy (w dowolny sposób).
5. Uporządkowanie arkuszy w kolejności alfabetycznej.

Co musimy wiedzieć?

Zapewne zauważyleś kilka „dziur” w naszym planie. Aby go zrealizować, będziemy musieli wyjaśnić następujące kwestie:

- Jak zidentyfikować aktywny skoroszyt?
- Jak policzyć arkusze w aktywnym skoroszycie?
- Jak pobrać listę nazw arkuszy?
- Jak posortować listę nazw arkuszy?
- Jak uporządkować arkusze w kolejności określonej posortowaną listą nazw?



Gdy nie dysponujesz wszystkimi informacjami na temat określonych metod lub właściwości, możesz ich poszukać w tej książce lub w systemie pomocy języka VBA i zapewne wcześniej czy później znajdziesz to, czego szukasz. Jednak najlepszym rozwiązaniem jest zwykle uruchomienie rejestratora makr i przeanalizowanie kodu źródłowego, który zostanie wygenerowany w trakcie wykonywania kilku powiązanych z sobą operacji. Niemal zawsze takie rozwiązanie dostarczy Ci wartościowych wskazówek, które będziesz mógł następnie wykorzystać w swoim projekcie.

Wstępne rejestrowanie makr

Poniżej przedstawiono przykład użycia rejestratora makr do zdobycia wiedzy na temat języka VBA. Na początku użyliśmy skoroszytu zawierającego trzy arkusze. Później uaktywniony został rejestrator makr i jako miejsce przechowywania makra określiliśmy skoroszyt makr osobistych. Po uruchomieniu rejestratora makr trzeci arkusz został przeciągnięty przed dotychczasowy pierwszy. Oto kod źródłowy wygenerowany przez rejestrator makr:

```
Sub Makrol()
    Sheets("Arkusz3").Select
    Sheets("Arkusz3").Move Before := Sheets(1)
End Sub
```

Kiedy poszukamy w systemie pomocy języka VBA informacji o metodzie Move, odkryjemy, że służy ona do przenoszenia arkusza w nowe miejsce skoroszytu i w dużym stopniu jest związana z wykonywanym zadaniem. Metoda pobiera argument identyfikujący położenie arkusza. Jak widać, są to informacje bardzo przydatne do realizacji naszego zadania. Następnie uruchomiliśmy rejestrator makr, aby sprawdzić, czy polecenie *Przenieś lub kopij arkusz* z menu podręcznego wygeneruje inny kod źródłowy. Okazuje się, że nie — wygenerowany kod był identyczny.

Następnie chcemy się dowiedzieć, ile arkuszy znajduje się w aktywnym skoroszycie. W systemie pomocy języka VBA znajdziemy informację, że słowo Count jest właściwością kolekcji. Po uaktywnieniu w edytorze Visual Basic okna *Immediate* wprowadźmy następującą instrukcję:

```
? ActiveWorkbook.Count
```

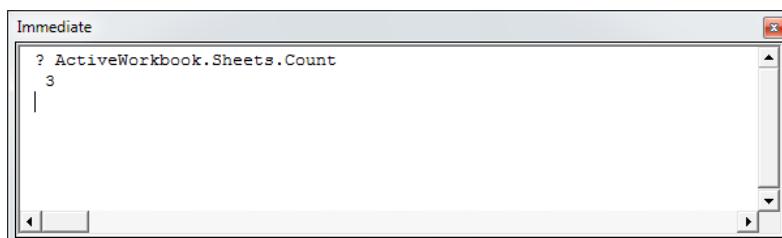
Błąd! Po chwili namysłu uświadomimy sobie, że musimy policzyć arkusze zawarte w skoroszycie. Użyjmy zatem takiej instrukcji:

```
? ActiveWorkbook.Sheets.Count
```

Sukces. Na rysunku 7.9 przedstawiono uzyskany rezultat. Bardzo przydatna informacja.

Rysunek 7.9.

Aby przetestować działanie polecenia, możesz użyć okna *Immediate* edytora VBE



A co z nazwami arkuszy? Pora wykonać inny test. W oknie *Immediate* wprowadź następującą instrukcję:

```
? ActiveWorkbook.Sheets(1).Name
```

Okazuje się, że pierwszy arkusz ma nazwę Arkusz3, co by się zgadzało (ponieważ wcześniej przenieśliśmy go przed pierwszy arkusz). Więcej dobrych informacji do zapamiętania.

Teraz spróbuj sobie przypomnieć, jak wspominaliśmy, że konstrukcja For Each ... Next jest przydatna w przypadku przetwarzania poszczególnych elementów kolekcji. Po krótkim sprawdzeniu w systemie pomocy, możemy utworzyć następującą procedurę, którą wykorzystamy do przetestowania naszej pętli:

```
Sub Test()
    For Each Sht In ActiveWorkbook.Sheets
        MsgBox Sht.Name
    Next Sht
End Sub
```

Kolejny sukces. Makro wyświetla trzy okna komunikatów, z których każde zawiera inną nazwę arkusza.

Wreszcie przyszła pora na opcje sortowania. Z systemu pomocy dowiemy się, że metoda Sort jest powiązana z obiektem Range. A zatem jednym z rozwiązań może być przeniesienie nazw arkuszy do zakresu, a następnie posortowanie ich. Jednak takie rozwiązanie wydaje się niepotrzebnie zbyt skomplikowane jak na tę aplikację. Lepiej będzie umieścić nazwy arkuszy w tablicy łańcuchów, a następnie posortować je przy użyciu instrukcji języka VBA.

Wstępne przygotowania

Teraz mamy już wystarczającą wiedzę, aby rozpocząć tworzenie kodu źródłowego z prawdziwego zdarzenia. Jednak zanim to nastąpi, musimy przeprowadzić pewne przygotowania wstępne. Aby odtworzyć to, co już do tej pory zrobiliśmy, wykonaj następujące operacje:

1. Utwórz pusty skoroszyt zawierający pięć arkuszy o nazwach Arkusz1, Arkusz2, Arkusz3, Arkusz4 i Arkusz5.
2. Przenieś losowo arkusze, tak aby nie były ułożone w żaden uporządkowany sposób.
3. Zapisz skoroszyt pod nazwą *Test.xlsx*.
4. Uaktywnij edytor *Visual Basic* i w oknie *Project Explorer* wybierz projekt *Personal.xlsb*.

Jeżeli *Personal.xlsb* nie jest widoczny w oknie *Project Explorer* edytora VBE, oznacza to, że nigdy nie używałeś skoroszytu makr osobistych do zapisania tworzonego makra. Aby Excel utworzył ten skoroszyt, powinieneś po prostu zarejestrować dowolne makro i jako miejsce przechowywania wybrać skoroszyt makr osobistych.

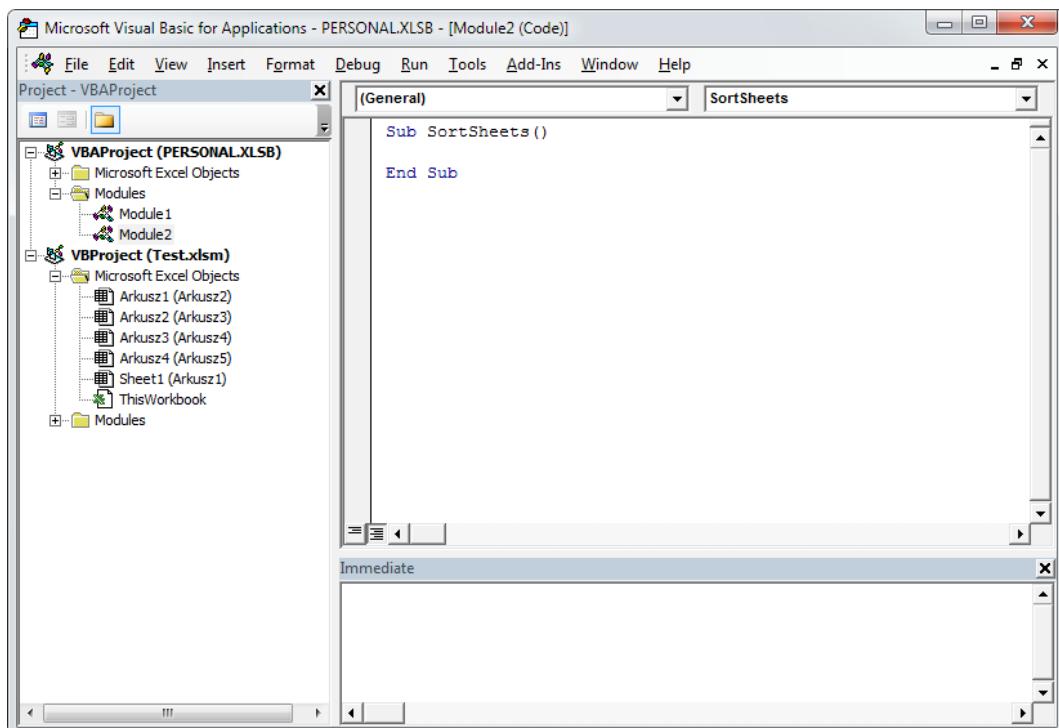
5. Wstaw nowy moduł VBA (z menu *Insert* wybierz polecenie *Module*).
6. Utwórz pustą procedurę o nazwie *SortSheets* (patrz rysunek 7.10).

Właściwie to makro może zostać zapisane w dowolnym module skoroszytu makr osobistych. Jednak zalecane jest przechowywanie każdego makra (lub każdej grupy powiązanych ze sobą makr) w oddzielnym module. Dzięki temu później z łatwością można taki moduł eksportować i importować.

7. Uaktywnij okno Excela. Przejdz na kartę *DEVELOPER* i naciśnij przycisk *Makra* znajdujący się w grupie opcji *Kod*. Na ekranie pojawi się okno dialogowe *Makro*.
8. W oknie dialogowym *Makro* wybierz procedurę *SortSheets* i naciśnij przycisk *Opcje*, aby przypisać do niej klawisz skrótu. Kombinacja klawiszy *Ctrl+Shift+S* będzie dobrym wyborem.

Tworzenie kodu źródłowego

Przyszła pora na stworzenie kodu źródłowego. Wiemy już, że musimy umieścić nazwy arkuszy w tablicy łańcuchów tekstu. Ponieważ nie znamy jeszcze liczby arkuszy aktywnego skoroszytu, to deklarując tablicę przy użyciu instrukcji *Dim*, posłużymy się para nawiasów okrągłych. Wiemy, że używając instrukcji *ReDim*, możemy później zmienić rozmiar tablicy, tak aby przechowywała właściwą liczbę elementów.



Rysunek 7.10. Pusta procedura umieszczona w module znajdującym się w skoroszycie makr osobistych

Teraz wpisz kod źródłowy przedstawiony poniżej, który umieszcza nazwy arkuszy w tablicy SheetNames. W pętli zastosujemy funkcję MsgBox tylko po to, aby uzyskać pewność, że nazwy arkuszy rzeczywiście zostały umieszczone w tablicy.

```

Sub SortSheets()
    ' Sortuje arkusze aktywnego skoroszytu
    Dim SheetNames() as String
    Dim i as Long
    Dim SheetCount as Long
    SheetCount = ActiveWorkbook.Sheets.Count
    ReDim SheetNames(1 To SheetCount)
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
        MsgBox SheetNames(i)
    Next i
End Sub

```

Aby przetestować procedurę, otwórz skoroszyt *Test.xlsx* i naciśnij kombinację klawiszy *Ctrl+Shift+S*. Na ekranie zostanie wyświetlonych pięć okien komunikatów zawierających nazwy kolejnych arkuszy aktywnego skoroszytu. Jak dotąd wszystko idzie zgodnie z planem.

Nawiasem mówiąc, jestem gorącym zwolennikiem testowania kodu źródłowego już w trakcie jego tworzenia. Zazwyczaj staram się pracować metodą małych kroków i zanim przejdę do tworzenia kolejnych, upewniam się, że właśnie utworzona procedura

działa tak, jak tego oczekowałem. Po uzyskaniu pewności, że kod źródłowy działa prawnie, możesz usunąć funkcję MsgBox (po jakimś czasie takie okna komunikatów stają się irytujące).



Zamiast funkcji MsgBox możesz w trakcie testowania tworzonej aplikacji zastosować metodę Print obiektu Debug, która powoduje wyświetlanie informacji w oknie Immediate. Przykładowo zamiast funkcji MsgBox użyj instrukcji:

```
Debug.Print SheetNames(i)
```

Metoda Print jest o wiele mniej irytująca niż korzystanie z polecenia MsgBox. Pamiętaj jednak, aby usunąć wywołanie metody Print z kodu programu, kiedy nie będzie już potrzebne.

Na tym etapie procedura SortSheets tworzy jedynie tablicę nazw odpowiadających arkuszom aktywnego skoroszytu. Pozostają dwa kroki do wykonania. Pierwszym jest posortowanie wartości tablicy SheetNames, natomiast drugim ponowne uporządkowanie arkuszy zgodnie z kolejnością elementów tablicy.

Tworzenie procedury sortującej

Nadeszła pora posortowania elementów tablicy SheetNames. Jedno z możliwych rozwiązań polega na umieszczeniu kodu źródłowego odpowiedzialnego za sortowanie w procedurze SortSheets, ale pomyślałem, że lepiej będzie stworzyć procedurę sortującą ogólnego zastosowania, której użyjemy również w innych projektach (sortowanie tablic jest częstą operacją).

Pomysł napisania własnej procedury sortującej nie powinien Cię zniechęcić, gdyż dość łatwo można znaleźć powszechnie używane procedury, a następnie przystosować je do własnych wymagań. Oczywiście sieć Internet to znakomite źródło tego typu rozwiązań.

Tablice można sortować na wiele sposobów. W naszym przypadku zdecydowaliśmy się użyć metody *sortowania bąbelkowego*. Nie jest szczególnie szybka, ale łatwa do kodowania. W przypadku naszej aplikacji duża szybkość sortowania nie jest wymagana.

Metoda sortowania bąbelkowego w celu sprawdzenia poszczególnych elementów tablicy wykorzystuje zagnieżdżoną pętlę For ... Next. Jeżeli określony element tablicy jest większy od następnego, oba elementy zamieniają się miejscami. Operacja jest powtarzana dla każdej pary elementów, czyli $n-1$ razy.



W rozdziale 9. omówiono kilka innych procedur sortujących i porównano je pod względem szybkości działania.

Oto kod źródłowy naszej procedury sortującej (który powstał po przeanalizowaniu zawartości kilku stron internetowych poświęconych programowaniu):

```
Sub BubbleSort(List() As String)
    ' Sortuje zawartość tablicy List w porządku rosnącym
    Dim First As Long, Last As Long
    Dim i As Long, j As Long
    Dim Temp As String
    First = LBound(List)
    Last = UBound(List)
```

```

For i = First To Last - 1
    For j = i + 1 To Last
        If List(i) > List(j) Then
            Temp = List(j)
            List(j) = List(i)
            List(i) = Temp
        End If
    Next j
Next i
End Sub

```

Procedura pobiera jeden argument, którym jest jednowymiarowa tablica List. Tablica przekazywana procedurze może mieć dowolną długość. W celu określenia dolnej i górnej granicy tablicy zastosowaliśmy odpowiednio funkcję LBound i Ubound, które przypisują odpowiednie wartości zmiennym First i Last.

Poniżej przedstawiono kod źródłowy małej, roboczej procedury, której użyjemy do testowania poprawności działania procedury BubbleSort:

```

Sub SortTester()
    Dim x(1 To 5) As String
    Dim i As Long
    x(1) = "pies"
    x(2) = "kot"
    x(3) = "słoń"
    x(4) = "mrówkojad"
    x(5) = "ptak"
    Call BubbleSort(x)
    For i = 1 To 5
        Debug.Print i, x(i)
    Next i
End Sub

```

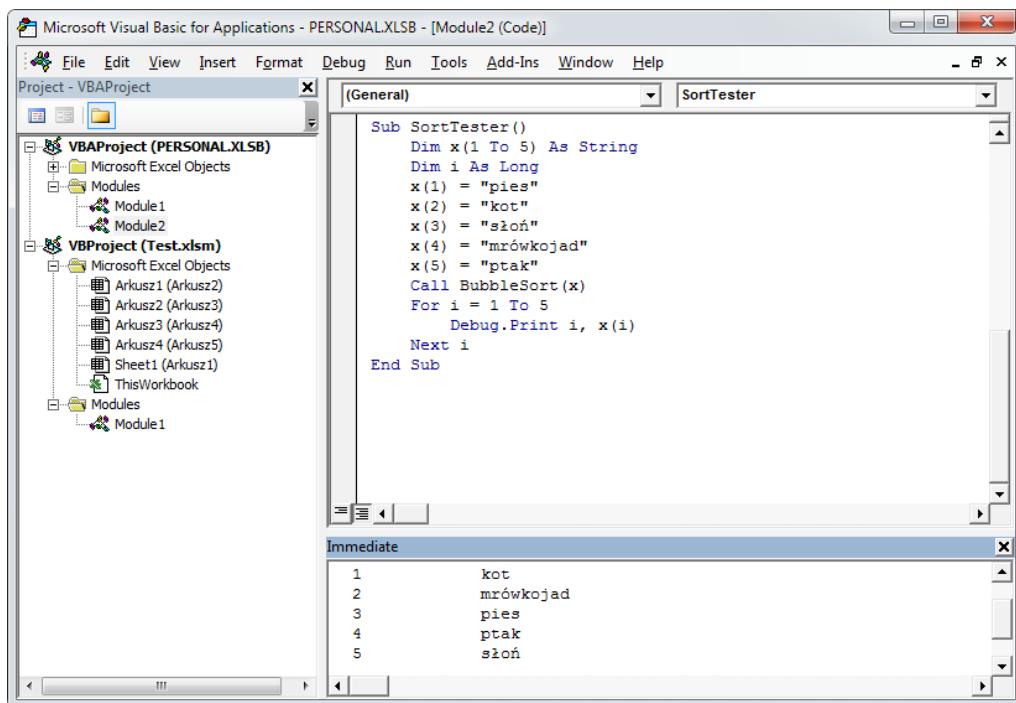
Procedura SortTester tworzy tablicę pięciu łańcuchów tekstu, przekazuje ją do procedury BubbleSort i następnie wyświetla zawartość posortowanej tablicy w oknie *Immediate* (patrz rysunek 7.11). Po pomyślnym przetestowaniu procedury sortującej możesz oczywiście usunąć kod procedury testującej.

Po stwierdzeniu, że procedura BubbleSort działała niezawodnie, zmodyfikowaliśmy procedurę SortSheets, dodając do niej wywołania procedury BubbleSort, której jako argument jest przekazywana tablica SheetNames. W tym momencie zawartość modułu jest następująca:

```

Sub SortSheets()
    Dim SheetNames() As String
    Dim SheetCount as Long
    Dim i as Long
    SheetCount = ActiveWorkbook.Sheets.Count
    ReDim SheetNames(1 To SheetCount)
    For i = 1 To SheetCount
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name
    Next i
    Call BubbleSort(SheetNames)
End Sub

```



Rysunek 7.11. Kod procedury tymczasowej sprawdzającej poprawność działania procedury BubbleSort

```
Sub BubbleSort(List() As String)
    ' Sortuje zawartość tablicy List w porządku rosnącym
    Dim First As Long, Last As Long
    Dim i As Long, j As Long
    Dim Temp As String
    First = LBound(List)
    Last = UBound(List)
    For i = First To Last - 1
        For j = i + 1 To Last
            If List(i) > List(j) Then
                Temp = List(j)
                List(j) = List(i)
                List(i) = Temp
            End If
        Next j
    Next i
End Sub
```

Kiedy procedura SheetSort kończy działanie, zawiera tablicę złożoną z posortowanych nazw arkuszy aktywnego skoroszytu. Aby się o tym przekonać, możesz wyświetlić zawartość tablicy w oknie *Immediate* edytora Visual Basic. Aby to zrobić, powinieneś na końcu procedury SortSheets umieścić następujący kod źródłowy (jeżeli okno *Immediate* nie jest widoczne, naciśnij kombinację klawiszy *Ctrl+G*):

```
For i = 1 To SheetCount
    Debug.Print SheetNames(i)
Next i
```

Na razie wszystko idzie dobrze. Następny krok: musimy utworzyć procedurę porządkującą arkusze zgodnie z kolejnością elementów tablicy SheetNames.

W tym momencie przydatny okaże się kod, który zarejestrowaliśmy już wcześniej. Czy pamiętasz polecenie, które zostało zarejestrowane, kiedy przenosiłeś arkusz skoroszytu, tak aby stał się jego pierwszym arkuszem? Oto ono:

```
Sheets("Arkusz3").Move Before := Sheets(1)
```

Po chwili namysłu możemy bez trudu utworzyć pętlę For ... Next przetwarzającą każdy arkusz i przenoszącą go w odpowiednie miejsce określone przez położenie elementu tablicy SheetNames:

```
For i = 1 To SheetCount  
    Sheets(SheetNames(i)).Move Before := Sheets(i)  
Next i
```

Licznik pętli (i) przy jej pierwszej iteracji ma wartość 1. Założmy, że pierwszym elementem tablicy SheetNames (w naszym przykładzie) jest Arkusz1, stąd wyrażenie zawierające metodę Move znajdujące się w pętli przyjmie następującą postać:

```
Sheets("Arkusz1").Move Before := Sheets(1)
```

Przy drugiej iteracji pętli, wyrażenie będzie miało następującą postać:

```
Sheets("Arkusz2").Move Before := Sheets(2)
```

Po dodaniu nowego kodu źródłowego procedura SortSheets wygląda następująco:

```
Sub SortSheets()  
    Dim SheetNames() As String  
    Dim SheetCount as Long  
    Dim i as Long  
    SheetCount = ActiveWorkbook.Sheets.Count  
    ReDim SheetNames(1 To SheetCount)  
    For i = 1 To SheetCount  
        SheetNames(i) = ActiveWorkbook.Sheets(i).Name  
    Next i  
    Call BubbleSort(SheetNames)  
    For i = 1 To SheetCount  
        ActiveWorkbook.Sheets(SheetNames(i)).Move _  
            Before:=ActiveWorkbook.Sheets(i)  
    Next i  
End Sub
```

Po wykonaniu kilku testów okazało się, że w przypadku skoroszytu *Test.xlsx* procedura działa znakomicie.

Pora na ostatnie poprawki. Upewnijmy się, czy wszystkie zmienne zastosowane w procedurach zostały zadeklarowane, a następnie dodamy kilka komentarzy i pustych wierszy, aby zwiększyć jego czytelność. Procedura SortSheets wygląda teraz następująco:

```
Sub SortSheets()  
    ' Procedura sortuje arkusze aktywnego skoroszytu w porządku rosnącym  
    ' Aby uruchomić procedurę,  
    ' naciśnij kombinację klawiszy Ctrl+Shift+S
```

```
Dim SheetNames() As String
Dim SheetCount As Long
Dim i As Long

' Określa liczbę arkuszy i zmienia rozmiar tablicy przy użyciu instrukcji ReDim
SheetCount = ActiveWorkbook.Sheets.Count
ReDim SheetNames(1 To SheetCount)

' Wypełnia tablicę nazwami arkuszy
For i = 1 To SheetCount
    SheetNames(i) = ActiveWorkbook.Sheets(i).Name
Next i

' Sortuje tablicę w porządku rosnącym
Call BubbleSort(SheetNames)

' Przenosi (sortuje) arkusze
For i = 1 To SheetCount
    ActiveWorkbook.Sheets(SheetNames(i)).Move _
        Before:= ActiveWorkbook.Sheets(i)
Next i
End Sub
```

Wszystko wydaje się działać poprawnie. W celu dodatkowego sprawdzenia kodu źródłowego dodaj do skoroszytu *Test.xlsx* kilka arkuszy i zmień kilka nazw. Procedura powinna działać bez zarzutu!

Dodatkowe testy

Wszystko działa, więc w zasadzie moglibyśmy zakończyć testowanie. Z drugiej strony jednak, to że nasza procedura znakomicie sprawdziła się w przypadku skoroszytu *Test.xlsx* nie oznacza wcale, że będzie równie dobrze działać dla innych skoroszytów. Aby przeprowadzić dodatkowe testy, załadowaliśmy więc kilka innych skoroszytów i ponownie uruchomiliśmy naszą procedurę. Wkrótce okazało się, że procedura nie jest doskonała. Co więcej, tak naprawdę jest daleka od doskonałości. Podczas testowania zidentyfikowaliśmy następujące problemy:

- Sortowanie skoroszytów zawierających wiele arkuszy zajmuje dużo czasu, ponieważ w trakcie wykonywania operacji przenoszenia zawartość ekranu jest cały czas aktualniana.
- Sortowanie nie zawsze działa poprawnie. Przykładowo w jednym z testów arkusz o nazwie **PODSUMOWANIE** (zawiera same duże litery) znalazł się przed arkuszem **Parametry**. Problem ten został spowodowany sposobem działania procedury **BubbleSort**, dla której litera *O* jest „*większa*” od litery *a*.
- Jeżeli w Excelu nie będzie widoczne żadne okno skoroszytu, wciśnięcie kombinacji klawiszy *Ctrl+Shift+S* spowoduje błąd działania makra.
- Jeżeli struktura skoroszytu jest chroniona, metoda **Move** nie zadziała.
- Po wykonaniu sortowania ostatni arkusz skoroszytu staje się arkuszem aktywnym. Zmiana aktywnego arkusza wybranego przez użytkownika nie jest zalecanym rozwiązaniem, w praktyce lepiej oczywiście pozostawić aktywny arkusz bez zmian.

- Po przerwaniu pracy makra poprzez wciśnięcie kombinacji klawiszy *Ctrl+Break* VBA wyświetla komunikat o błędzie.
- Działanie makra nie może być wycofane (inaczej mówiąc, polecenie *Cofnij* nie jest dostępne). Jeżeli użytkownik przypadkowo naciśnie kombinację klawiszy *Ctrl+Shift+S*, arkusze skoroszytu zostaną posortowane i jedynym sposobem na przywrócenie ich początkowego układu będzie przenoszenie ręczne.

Usuwanie problemów

Usunięcie problemu związanego z uaktualnianiem zawartości ekranu jest bardzo proste. Aby wyłączyć funkcję odświeżania zawartości ekranu wystarczy na początku procedury SortSheets umieścić następującą instrukcję:

```
Application.ScreenUpdating = False
```

Instrukcja powoduje „zamrożenie” okien Excela na czas działania makra. Dodatkową korzyścią takiej operacji jest też znaczące zwiększenie szybkości działania makra. Po zakończeniu działania makra, odświeżanie ekranu jest przywracane automatycznie.

Proste jest również usunięcie problemu z procedurą BubbleSort. Aby w nazwach arkuszy występowały same wielkie litery, użyjemy funkcji *UCase* języka VBA, co powoduje, że sortowane nazwy arkuszy są wszystkie „zapisane” przy użyciu wielkich liter. Poprawiony wiersz kodu źródłowego ma następującą postać:

```
If UCASE(List(i)) > UCASE(List(j)) Then
```



Inna metoda rozwiązywania problemu z wielkimi znakami polega na umieszczeniu na początku modułu następującej klauzuli:

```
Option Compare Text
```

Klauzula ta powoduje, że podczas porównywania łańcuchów tekstu VBA nie rozróżnia wielkości znaków. Innymi słowy, litera A jest traktowana tak jak litera a.

Aby wyeliminować komunikat błędu pojawiający się, gdy nie jest widoczne okno żadnego skoroszytu (żaden skoroszyt nie jest aktywny), dodamy mechanizm kontroli błędów. Aby zignorować błąd, użyjemy polecenia *On Error Resume Next*, a następnie sprawdzimy wartość obiektu Err. Jeżeli wartość jest różna od zera, oznacza to, że wystąpił błąd i procedura zakończy działanie. Kod źródłowy spełniający funkcję mechanizmu sprawdzającego błędy ma następującą postać:

```
On Error Resume Next
SheetCount = ActiveWorkbook.Sheets.Count
If Err <> 0 Then Exit Sub ' Brak aktywnego skoroszytu
```

Stwierdziliśmy jednak, że możemy uniknąć stosowania polecenia *On Error Resume Next*. Poniższa instrukcja jest bardziej bezpośrednią metodą określania, czy skoroszyt jest widoczny, a ponadto nie wymaga stosowania żadnego mechanizmu obsługi błędów. Polecenie powinno zostać umieszczone na początku procedury SortSheets:

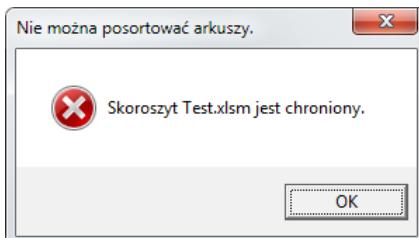
```
If ActiveWorkbook Is Nothing Then Exit Sub
```

Jeżeli struktura skoroszytu jest chroniona, zwykle istnieją ku temu poważne powody. Z tego powodu próba usunięcia ochrony skoroszytu nie będzie najlepszym rozwiązaniem. Zamiast tego kod źródłowy powinien wyświetlić okno komunikatu ostrzegające użytkownika oraz umożliwiające mu usunięcie ochrony skoroszytu i ponowne uruchomienie makra. Sprawdzenie, czy struktura skoroszytu jest chroniona, jest prostym zadaniem. Właściwość `ProtectStructure` obiektu `Workbook` zwraca wartość `True`, jeżeli skoroszyt jest chroniony. Do naszej procedury dodajemy zatem następujący kod źródłowy:

```
' Sprawdź, czy struktura skoroszytu jest chroniona
If ActiveWorkbook.ProtectStructure Then
    MsgBox "Skoroszyt " & ActiveWorkbook.Name & " jest chroniony.", _
        vbCritical, "Nie można posortować arkuszy."
    Exit Sub
End If
```

Jeżeli struktura skoroszytu jest chroniona, na ekranie zostanie wyświetcone okno dialogowe z odpowiednim komunikatem, przedstawione na rysunku 7.12.

Rysunek 7.12.
Okno dialogowe informujące użytkownika, że struktura arkusza jest chroniona i nie można posortować arkuszy



Aby po zakończeniu sortowania przywrócić oryginalny aktywny arkusz, dodamy kod, który przypisze nazwę aktywnego arkusza do zmiennej obiektowej (`OldActive`) i ponownie uaktywni ten arkusz po zakończeniu działania makra. Polecenie przypisujące nazwę aktywnego arkusza do zmiennej wygląda następująco:

```
Set OldActive = ActiveSheet
```

Polecenie przedstawione poniżej aktywuje oryginalny arkusz:

```
OldActive.Activate
```

Naciśnięcie kombinacji klawiszy *Ctrl+Break* normalnie powoduje wstrzymanie działania makra i wyświetlenie przez VBA odpowiedniego komunikatu o błędzie. Ponieważ jednak jednym z założeń naszej aplikacji jest unikanie wyświetlania komunikatów o błędach, musimy dodać odpowiedni kod programu. Korzystając z pomocy systemowej, możesz się przekonać, że obiekt `Application` posiada właściwość `EnableCancelKey`, która może zablokować działanie kombinacji klawiszy *Ctrl+Break*, stąd na początku procedury umieścimy następujące polecenie:

```
Application.EnableCancelKey = xlDisabled
```



Po wyłączeniu kombinacji *Ctrl+Break* należy zachować dużą ostrożność. Jeżeli kod źródłowy zacznie wykonywać nieskończoną pętlę, nie będzie możliwe przerwanie jego działania. Aby uniknąć problemów, powyższej instrukcji powinieneś używać tylko po upewnieniu się, że wszystko działa poprawnie.

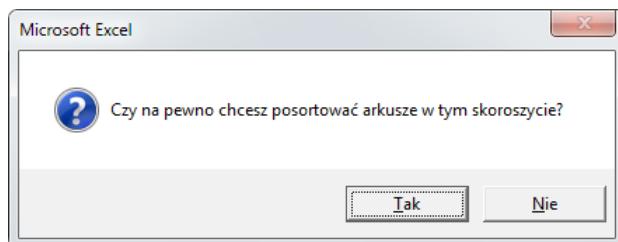
Aby uniknąć niebezpieczeństwa, że użytkownik przypadkowo uruchomi naszą procedurę i posortuje arkusze w skoroszycie, przed zablokowaniem kombinacji klawiszy *Ctrl+Break* dodajemy następujące polecenie:

```
If MsgBox("Czy na pewno chcesz posortować arkusze w tym skoroszycie?", _
vbQuestion + vbYesNo) <> vbYes Then Exit Sub
```

Kiedy użytkownik spróbuje uruchomić procedurę *SortSheets*, na ekranie pojawi się okno dialogowe przedstawione na rysunku 7.13.

Rysunek 7.13.

Okno dialogowe, które pojawia się na ekranie przed rozpoczęciem sortowania



Po wykonaniu wszystkich poprawek nasza procedura *SortSheets* wygląda tak, jak to przedstawiono poniżej:

```
Option Explicit
Sub SortSheets()
    ' Procedura sortuje arkusze aktywnego skoroszytu w porządku rosnącym
    ' Aby uruchomić procedurę, naciśnij kombinację klawiszy Ctrl+Shift+S

    Dim SheetNames() As String
    Dim i As Long
    Dim SheetCount As Long
    Dim OldActiveSheet As Object

    If ActiveWorkbook Is Nothing Then Exit Sub ' Brak aktywnego skoroszytu
    SheetCount = ActiveWorkbook.Sheets.Count

    ' Sprawdź, czy struktura skoroszytu jest chroniona
    If ActiveWorkbook.ProtectStructure Then
        MsgBox "Skoroszyt " & ActiveWorkbook.Name & " jest chroniony.". _
            vbCritical, "Nie można posortować arkuszy!"
        Exit Sub
    End If
    ' Sprawdź, czy użytkownik na pewno chce posortować arkusze
    If MsgBox("Czy na pewno chcesz posortować arkusze w tym skoroszycie?", _
        vbQuestion + vbYesNo) <> vbYes Then Exit Sub
    ' Zablokuj działanie Ctrl+Break
    Application.EnableCancelKey = xlDisabled

    ' Pobierz liczbę arkuszy
    SheetCount = ActiveWorkbook.Sheets.Count

    ' Zmień wymiary tablicy
    ReDim SheetNames(1 To SheetCount)
    ' Zapamiętaj odwołanie do aktywnego skoroszytu
    Set OldActiveSheet = ActiveSheet
    ' Wypełnij tablicę nazwami arkuszy
```

```
For i = 1 To SheetCount
    SheetNames(i) = ActiveWorkbook.Sheets(i).Name
Next i

' Posortuj zawartość tablicy w porządku rosnącym
Call BubbleSort(SheetNames)

' Wyłącz aktualizację ekranu
Application.ScreenUpdating = False

' Przenieś arkusze
For i = 1 To SheetCount
    ActiveWorkbook.Sheets(SheetNames(i)).Move _
        Before:=ActiveWorkbook.Sheets(i)
Next i

' Reaktywuj oryginalny arkusz
OldActiveSheet.Activate
End Sub
```

Dostępność narzędzia

Ponieważ makro SortSheets zostało zapisane w skoroszycie makr osobistych, dostępne jest zawsze po uruchomieniu Excela. Można je uruchomić, wybierając jego nazwę w oknie dialogowym *Makro* (w celu jego otwarcia należy wcisnąć kombinację klawiszy *Alt+F8*) lub naciskając kombinację klawiszy *Ctrl+Shift+S*. Innym rozwiązaniem może być dodanie osobnego polecenia sortującego arkusze do Wstążki.

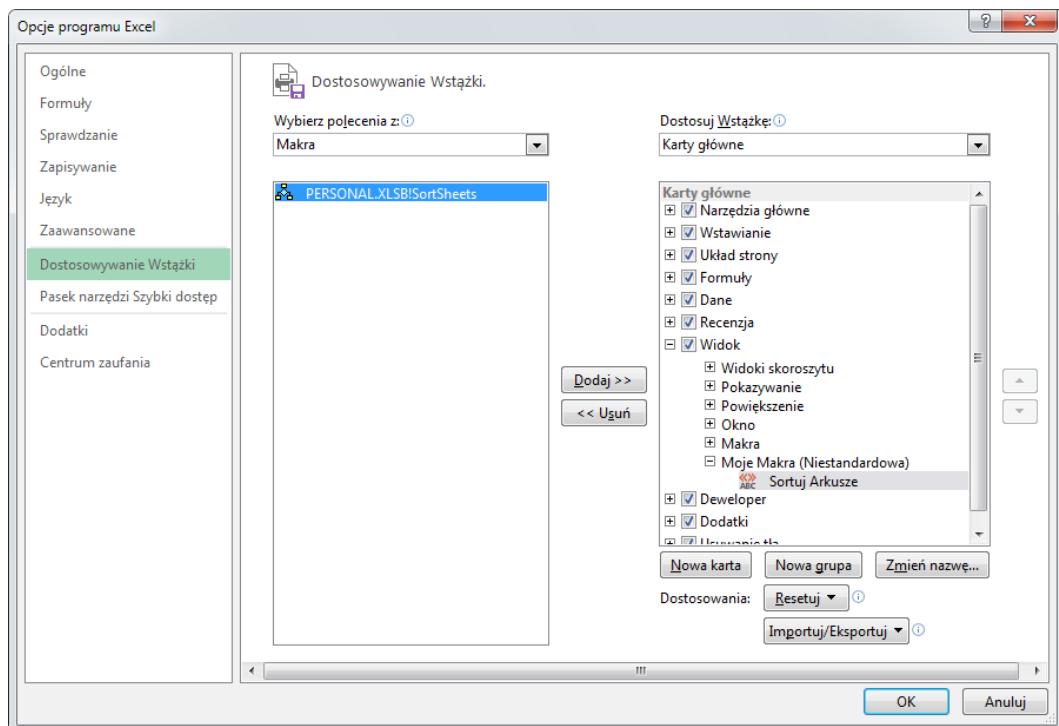
Aby to zrobić, powinieneś wykonać polecenia przedstawione poniżej.

1. Kliknij prawym przyciskiem myszy dowolny, pusty obszar Wstążki i z menu podręcznego, które pojawi się na ekranie wybierz polecenie *Dostosuj Wstążkę*.
2. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Z listy rozwijanej *Wybierz polecenia* z wybierz opcję *Makra*.
3. Na liście dostępnych elementów zaznacz makro *PERSONAL.XLSB!SortSheets*.
4. Korzystając z opcji w prawej części okna wybierz docelową kartę Wstążki i utwórz nową grupę (nie możesz dodać nowego polecenia do istniejącej grupy).

W naszym przypadku utworzyliśmy na karcie *Widok* grupę o nazwie *Arkusze*, umieściliśmy na niej przycisk z naszym makrem i zmieniliśmy jego nazwę na *Sortuj arkusze* (patrz rysunek 7.14).

Ocena projektu

Krok po kroku i dotarliśmy do końca. Nasze narzędzie spełnia wszystkie początkowe założenia projektowe. Sortuje arkusze aktywnego skoroszytu, można je łatwo uruchomić, jest zawsze dostępne, wydaje się działać poprawnie w przypadku dowolnego skoroszytu i jeszcze nigdy nie zdarzyło się, żeby to makro wyświetliło jakikolwiek komunikat o błędzie.



Rysunek 7.14. Dodawanie nowego polecenia do Wstążki



Nasza procedura w dalszym ciągu ma pewien problem. Polega on na tym, że sortowanie jest proste i może nie zawsze „logiczne”. Na przykład po wykonaniu sortowania arkusz Arkusz11 jest umieszczany przed arkuszem o nazwie Arkusz2. Większość osób chciałaby, aby to arkusz Arkusz2 znalazł się przed arkuszem Arkusz11. Rozwiążanie tego problemu, choć niezbyt skomplikowane, wykracza jednak nieco poza ramy tego prostego przykładu.

Rozdział 8.

Tworzenie funkcji

w języku VBA

W tym rozdziale:

- Różnice pomiędzy procedurami Sub i Function (funkcjami)
- Tworzenie własnych funkcji
- Procedury Function i ich argumenty
- Tworzenie funkcji, która emuluje działanie funkcji arkuszowej SUMA
- Korzystanie z funkcji, które pozwalają na przetwarzanie dat wcześniejszych niż rok 1900
- Wyszukiwanie i usuwanie błędów w kodzie funkcji, korzystanie z okna dialogowego *Wstawianie funkcji* oraz używanie dodatków przechowujących funkcje zdefiniowane przez użytkownika
- Wywoływanie funkcji Windows API z poziomu procedur VBA w celu wykonania operacji, których nie można zrealizować w inny sposób

Porównanie procedur Sub i Function

Funkcja to rodzaj procedury VBA, która wykonuje określone obliczenia i zwraca ich rezultat. Funkcji można używać w kodzie programów VBA oraz formułach arkuszowych.

VBA pozwala na tworzenie procedur Sub oraz Function. Procedury Sub możesz sobie wyobrazić jako zestaw operacji, których wykonanie może być wywołane bezpośrednio przez użytkownika lub z poziomu innych procedur. Z kolei procedury Function (funkcje) zazwyczaj zwracają pojedynczą wartość lub tablicę, podobnie jak funkcje arkusza Excela i wbudowane funkcje języka VBA. Procedury Function, podobnie jak funkcje wbudowane, mogą używać argumentów.

Procedury Function są dość uniwersalne i możesz ich używać w dwóch sytuacjach:

- jako część wyrażenia zawartego w procedurze języka VBA,
- w formułach tworzonych w arkuszu.

W praktyce procedury Function można zastosować wszędzie tam, gdzie używasz funkcji arkuszowych Excela lub wbudowanych funkcji języka VBA. O ile mi wiadomo, jedynym miejscem, w którym nie możesz wykorzystywać funkcji VBA, są formuły sprawdzania poprawności danych.

Procedury Sub zostały omówione w poprzednim rozdziale, natomiast ten rozdział poświęcony w całości procedurom Function.



W rozdziale 9. znajdziesz wiele praktycznych przykładów zastosowania procedur Function, które będziesz mógł wykorzystać we własnych projektach.

Dlaczego tworzymy funkcje niestandardowe?

Z pewnością jesteś zaznajomiony z funkcjami arkuszowymi Excela. Nawet poczatkujący wiedzą, jak posługiwać się najczęściej stosowanymi funkcjami, takimi jak SUMA, ŚREDNIA czy JEŻELI. Excel 2013 posiada ponad 450 predefiniowanych funkcji arkuszowych, których możesz używać w formułach. Jeżeli jednak to nie wystarczy, możesz przy użyciu VBA tworzyć własne funkcje.

Mając do dyspozycji wszystkie funkcje Excela i języka VBA, można się zastanawiać, po co w ogóle tworzyć nowe funkcje. Odpowiedź jest prosta: aby ułatwić sobie życie. Dobrze zaprojektowane, niestandardowe funkcje są bardzo przydatne w formułach arkuszy i procedurach języka VBA.

Przykładowo, własne funkcje mogą bardzo często znacząco uprościć tworzone formuły, a krótsze formuły są czytelniejsze i łatwiej z nich korzystać. Warto tutaj jednak zaznaczyć, że funkcje niestandardowe używane w formułach są zazwyczaj nieco wolniejsze od funkcji wbudowanych, a poza tym, aby skorzystać z własnych funkcji, musisz zezwolić na wykonywanie makr.

Kiedy będziesz tworzył własne aplikacje, z pewnością zauważysz, że pewne obliczenia często powtarzają się w procedurach. W takim przypadku warto rozważyć utworzenie niestandardowej funkcji odpowiedzialnej za obliczenia, która będzie można po prostu wywołać z procedury. Niestandardowa funkcja eliminuje konieczność powielania tego samego kodu w wielu procedurach, a tym samym redukuje liczbę potencjalnych błędów.

Wiele osób na myśl o konieczności tworzenia niestandardowych funkcji arkusza jest przerażonych, ale w praktyce to zadanie nie jest takie trudne. Tak naprawdę tworzenie niestandardowych funkcji naprawdę sprawia mi dużą przyjemność. Szczególnie jestem zdowolony, gdy utworzone przeze mnie funkcje niestandardowe pojawiają się w oknie dialogowym *Wstawianie funkcji* obok wbudowanych funkcji Excela, ponieważ czuję się, jakbym pod jakimś względem dokonał przebudowy tej aplikacji.

W tym rozdziale dowiesz się, jak tworzyć własne, niestandardowe funkcje VBA, jak również znajdziesz wiele przykładów takich funkcji.

Twoja pierwsza funkcja

Bez zbędnych rozważyń zaprezentujemy teraz przykład tworzenia prostej funkcji języka VBA.

Poniżej zamieszczamy kod jednoargumentowej, niestandardowej funkcji zdefiniowanej w module VBA. Funkcja o nazwie REMOVEVOWELS zwraca ciąg znaków będący jej argumentem, ale pozbawiony wszystkich samogłosek.

```
Function REMOVEVOWELS(Txt) As String
    ' Usuwa wszystkie samogłoski z łańcucha znaków Txt
    Dim i As Long
    REMOVEVOWELS = ""
    For i = 1 To Len(Txt)
        If Not UCASE(Mid(Txt, i, 1)) Like "[AEIOUY]" Then
            REMOVEVOWELS = REMOVEVOWELS & Mid(Txt, i, 1)
        End If
    Next i
End Function
```

Oczywiście nie jest to może zbyt użyteczna funkcja, ale za to znakomicie ilustruje pewne kluczowe zagadnienia związane z funkcjami. Sposób działania tej funkcji wyjaśnimy nieco dalej, w podrozdziale „Analiza funkcji niestandardowej”.



Tworząc niestandardowe funkcje, które będą stosowane w formule arkusza, upewnij się, że ich kod źródłowy został umieszczony w zwykłym module VBA (aby utworzyć normalny moduł kodu VBA, powinieneś z menu głównego edytora VBE wybrać polecenie *Insert/VBE*). Jeżeli umieścisz kod funkcji w module UserForm, Sheet lub ThisWorkbook, to takie funkcje nie będą działały w formułach arkusza. Próba użycia takiej funkcji w formule zakończy się wyświetleniem błędu #NAZWA?.

Zastosowanie funkcji w arkuszu

Po zdefiniowaniu formuły używającej funkcji REMOVEVOWELS Excel podczas obliczania formuły wykona kod funkcji. Oto przykład zastosowania takiej funkcji w formule:

=REMOVEVOWELS(A1)

Na rysunku 8.1 pokazano przykład działania funkcji. Formuły zawarte w kolumnie B jako argumentów używają łańcuchów tekstu, umieszczonych w kolumnie A. Jak widać, funkcja zwraca ciąg znaków będący argumentem funkcji, ale pozbawiony samogłosek.

Rysunek 8.1.

Przykład zastosowania
w formule arkusza
funkcji niestandardowej

	A	B	C
1	Prawdziwych przyjaciół poznaje się w biedzie	Prwdzwch przjciól pznj sę w bdz	
2	Konstantynopolitańczykiewiczówna	Knstntplńtńczkwówn	
3	Microsoft Excel	Mcrsft xcl	
4	abcdefghijklmnoprstuvwxyz	bcdflghijklmnoprstvwxz	
5	Transmisja została zakończona powodzeniem	Trnsmsj zstł zkńczn pwdznm	
6	W tym zdaniu nie będzie samogłosek	W tm zdñ n będż smglsk	
7	Samogłoski: AEIOUY	Smglsk:	
8	Humuhumunukunukuapua'a to słynna hawajska rybka	Hmhmnknkp't slnn hwjsk rbk	
9			
10			

Właściwie funkcja działa bardzo podobnie do dowolnej innej, wbudowanej funkcji arkusza. Aby wstawić taką funkcję do formuły, przejdź na kartę *FORMUŁY* i naciśnij przycisk *Wstaw funkcję*, znajdujący się w grupie opcji *Biblioteka funkcji*. Możesz też po prostu naciśnąć przycisk *Wstaw funkcję* znajdujący się z lewej strony paska formuły. Po wykonaniu jednej z tych operacji na ekranie pojawi się okno dialogowe *Wstawianie funkcji*, w którym funkcje niestandardowe wyświetlane są po wybraniu kategorii *Zdefiniowane przez użytkownika*.

Funkcje niestandardowe mogą być zagnieżdżane i łączone z innymi elementami formuł. Na przykład poniższa formuła zagnieżdża wywołanie funkcji REMOVEVOWELS w funkcji LITERY.WIELKIE. Wynikiem działania takiego wyrażenia jest wejściowy ciąg znaków, pozbawiony samogłosek i napisany przy użyciu wielkich liter.

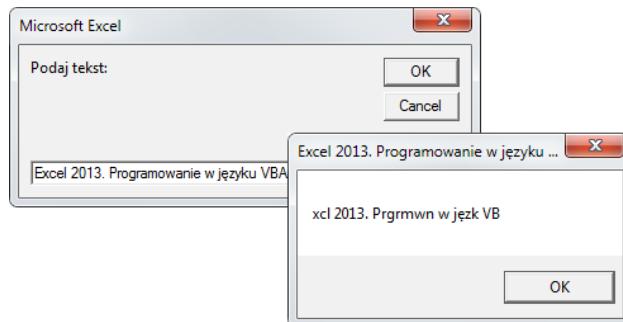
= LITERY.WIELKIE(REMOVEVOWELS(A1))

Zastosowanie funkcji w procedurze języka VBA

Niestandardowe funkcje mogą być używane nie tylko w formułach arkusza, ale również w procedurach języka VBA. Procedura przedstawiona poniżej została zdefiniowana w tym samym module co niestandardowa funkcja REMOVEVOWELS. Po uruchomieniu procedury najpierw wyświetla okno umożliwiające użytkownikowi wprowadzenie łańcucha tekstuowego, a następnie używa wbudowanej funkcji MsgBox języka VBA do wyświetlenia tego samego tekstu, ale już po przetworzeniu go przez funkcję REMOVEVOWELS (patrz rysunek 8.2). Oryginalny tekst zostaje wyświetlony na pasku tytułu okna komunikatu.

```
Sub ZapTheVowels()
    Dim UserInput as String
    UserInput = InputBox("Podaj tekst:")
    MsgBox REMOVEVOWELS(UserInput), , UserInput
End Sub
```

Rysunek 8.2.
Zastosowanie niestandardowej funkcji w procedurze języka VBA



W przykładzie pokazanym na rysunku 8.2 ciąg znaków wprowadzony po wywołaniu funkcji InputBox ma postać *Excel 2013. Programowanie w języku VBA*. Funkcja MsgBox wyświetla tekst po usunięciu z niego wszystkich samogłosek.

Analiza funkcji niestandardowej

Procedury Function mogą być dowolnie złożone. Oczywiście, zazwyczaj funkcje są bardziej złożone i bardziej użyteczne od przedstawionej funkcji przykładowej, co nie zmienia w niczym faktu, że analiza naszej prostej funkcji może być pomocna w zrozumieniu sposobu działania bardziej złożonych procedur.

Dla przypomnienia ponownie zamieszczamy kod naszej funkcji:

```
Function REMOVEVOWELS(Txt) As String
    ' Usuwa samogloski z lancucha znakow Txt
    Dim i As Long
    REMOVEVOWELS = ""
    For i = 1 To Len(Txt)
        If Not UCASE(Mid(Txt, i, 1)) Like "[AEIOUY]" Then
            REMOVEVOWELS = REMOVEVOWELS & Mid(Txt, i, 1)
        End If
    Next i
End Function
```

Zwróć uwagę, że nasza procedura rozpoczyna się słowem kluczowym Function zamiast Sub, po którym następuje nazwa funkcji (REMOVEVOWELS). Funkcja używa tylko jednego argumentu (Txt) zawartego w nawiasach okrągłych. Klauzula As String definiuje typ danych wartości zwracanej przez funkcję. Jeżeli nie zadeklarujesz typu zwracanej przez funkcję wartości, Excel domyślnie używa typu Variant.

Drugi wiersz jest zwykłym komentarzem opisującym przeznaczenie funkcji (oczywiście komentarz jest opcjonalny i funkcja będzie działać znakomicie i bez niego). W kolejnym wierszu znajduje się instrukcja Dim, za pomocą której deklarujemy zmienną i typu Long, wykorzystywaną w dalszej części naszej procedury.



Zauważ, że nazwa funkcji została tutaj użyta jako zmienna, której wartość została zainicjowana pustym ciągiem znaków. Kiedy funkcja kończy działanie, zawsze zwraca bieżącą wartość zmiennej o nazwie odpowiadającej nazwie funkcji.

Kolejnych pięć poleceń tworzy strukturę pętli For ... Next. Procedura przechodzi w pętli przez kolejne znaki łańcucha tekstu przekazanego jako argument wywołania funkcji i tworzy ciąg znaków będący wynikiem działania funkcji. Pierwsze polecenie w pętli używa funkcji Mid języka VBA do pobrania kolejnego znaku z łańcucha wejściowego i zamienia go na wielką literę. Następnie taki znak jest porównywany z listą samogłosek przy użyciu operatora Like. Inaczej mówiąc, warunek polecenia If jest prawdziwy, kiedy pobrany znak nie jest literą A, E, I, O, U ani Y. Jeżeli warunek jest prawdziwy, bieżący znak jest dodawany do ciągu znaków będącego wartością zmiennej REMOVEVOWELS.

Po zakończeniu działania pętli zmienna REMOVEVOWELS zawiera ciąg znaków odpowiadający łańcuchowi wejściowemu, ale pozbawionemu samogłosek, i to jest właśnie wartość zwracana przez funkcję.

Kod procedury kończy się poleceniem End Function.

Pamiętaj, że funkcję realizującą opisane zadanie można napisać na wiele różnych sposobów. Poniżej przedstawiono kod funkcji realizującej identyczną operację, ale w zupełnie inny sposób:

```

Function REMOVEVOWELS(txt) As String
    ' Usuwa samogłoski z łańcucha znaków Txt
    Dim i As Long
    Dim TempString As String
    TempString = ""
    For i = 1 To Len(txt)
        Select Case UCase(Mid(txt, i, 1))
            Case "A", "E", "I", "O", "U", "Y"
                'Nic nie robimy
            Case Else
                TempString = TempString & Mid(txt, i, 1)
        End Select
    Next i
    REMOVEVOWELS = TempString
End Function

```

W tej wersji do tworzenia i przechowywania pozbawionego samogłosek ciągu znaków używamy zmiennej tekstowej TempString. Kiedy procedura kończy działanie, wartość zmiennej TempString jest przypisywana do zmiennej reprezentującej nazwę funkcji. Jak łatwo zauważyc, w tej wersji funkcji zamiast instrukcji If ... Then używamy instrukcji Select ... Case.



Skoroszyt zawierający oba przykłady (*RemoveVowels.xlsxm*) znajdziesz na stronie internetowej tej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

W sieci

Czego nie potrafią niestandardowe funkcje arkusza?

Jeżeli tworzysz niestandardowe funkcje, musisz zrozumieć kluczową różnicę pomiędzy funkcjami wywoływanymi z innych procedur języka VBA i funkcjami stosowanymi w formułach arkusza. Procedury Function w formułach muszą być pasywne, co oznacza, że na przykład nie mogą manipulować zakresami lub zmieniać zawartości arkusza. Spróbujmy to zilustrować na przykładzie poniżej.

Załóżmy, że chcesz stworzyć niestandardową funkcję arkusza zmieniającą formatowanie komórki i użyć jej w formule, która w oparciu o wartość komórki zmienia kolor zawartego w niej tekstu. Możesz próbować różnych sposobów, ale z pewnością Ci się nie uda — po prostu napisanie takiej funkcji nie jest możliwe. Niezależnie od tego, jak bardzo będziesz próbował, funkcja nigdy nie będzie w stanie zmienić arkusza. Pamiętaj — funkcja po prostu zwraca obliczoną wartość i nie potrafi wykonywać operacji na obiektach.

Warto tutaj jednak wspomnieć o jednym wyjątku. Otóż za pomocą własnej, niestandardowej funkcji VBA możesz zmieniać treść komentarza załączonego do komórki. Nie jestem co prawda pewny, czy jest to efekt zamierzony, czy po prostu błąd Excela. Niezależnie jednak od przyczyny, modyfikacja komentarzy za pomocą funkcji wydaje się działać całkiem sprawnie i stabilnie. A oto przykład takiej funkcji:

```

Function ModifyComment(Cell As Range, Cmt As String)
    Cell.Comment.Text Cmt
End Function

```

Poniżej przedstawiono przykład zastosowania takiej funkcji w formule. Nasza formuła zastępuje komentarz w komórce A1 nowym tekstem. Pamiętaj, że taka funkcja nie będzie działała, jeżeli w komórce A1 nie będzie wcześniej żadnego komentarza.

```
=ModifyComment(A1, "Hej, właśnie zmieniłem komentarz!")
```

Procedury Function

Funkcje, a ściślej mówiąc, procedury typu Function, mają wiele wspólnego z procedurami Sub (więcej szczegółowych informacji na temat procedur Sub znajdziesz w rozdziale 7.).

Składnia deklaracji funkcji ma następującą postać:

```
[Public | Private][Static] Function nazwa ([lista_argumentów]) [As typ]
    [instrukcje]
    [nazwa = wyrażenie]
    [Exit Function]
    [instrukcje]
    [nazwa = wyrażenie]
End Function
```

Deklaracja funkcji składa się z następujących elementów:

- **Public** (opcjonalne słowo kluczowe) — wskazuje, że procedura Function będzie dostępna dla wszystkich innych procedur zawartych we wszystkich pozostałych modułach wszystkich aktywnych projektów VBA Excela.
- **Private** (opcjonalne słowo kluczowe) — wskazuje, że procedura Function będzie dostępna tylko dla innych procedur tego samego modułu.
- **Static** (opcjonalne słowo kluczowe) — wskazuje, że wartości zmiennych zadeklarowanych w procedurze Function są zachowywane pomiędzy kolejnymi wywołaniami.
- **Function** (wymagane słowo kluczowe) — wskazuje początek procedury zwracającej wartość lub dane innego typu.
- **nazwa** (wymagana) — reprezentuje dowolną poprawną nazwę procedury Function. Nazwa musi zostać nadana zgodnie z zasadami stosowanymi przy nadawaniu nazw zmiennym.
- **lista_argumentów** (opcjonalna) — lista składająca się z jednej lub kilku zmiennych będących argumentami przekazywanymi procedurze. Argumenty są umieszczone w nawiasach okrągłych. W celu oddzielenia od siebie kolejnych argumentów stosuje się przecinek.
- **typ** (opcjonalny) — identyfikuje typ danych wartości zwracanej przez funkcję (procedurę Function).
- **instrukcje** (opcjonalne) — dowolna liczba poprawnych instrukcji języka VBA.
- **Exit Function** (klauzula opcjonalna) — wymusza natychmiastowe zakończenie procedury Function jeszcze przed osiągnięciem jej formalnego końca.
- **End Function** (wymagane słowo kluczowe) — wskazuje koniec procedury Function.

Należy pamiętać, że w kodzie funkcji musi się znaleźć przynajmniej jedno przypisanie wartości do zmiennej reprezentującej nazwę funkcji. Zwykle ma to miejsce, gdy funkcja kończy działanie.

Aby utworzyć niestandardową funkcję, powinieneś najpierw utworzyć moduł kodu VBA. Oczywiście możesz również posłużyć się już istniejącym modelem. Następnie wprowadź słowo kluczowe Function, a za nim nazwę funkcji i listę argumentów (jeżeli są konieczne) w nawiasach okrągłych. Za pomocą słowa kluczowego As (opcjonalne, ale zalecane) możesz zadeklarować typ danych zwracanej wartości. Teraz wprowadź kod źródłowy języka VBA realizujący zadanie, upewniając się, że wewnętrz procedury Function przynajmniej raz odpowiednia wartość została przypisana zmiennej mającej nazwę funkcji. Zakończ funkcję, używając instrukcji End Function.

Nazwy funkcji muszą spełniać takie same wymagania, jak nazwy zmiennych. Jeżeli planujesz użycie w formule arkusza funkcji niestandardowej, upewnij się, że jej nazwa nie ma postaci adresu komórki. Na przykład jeżeli jako nazwy funkcji użyjesz czegoś w rodzaju ABC123, to nie będziesz mógł używać takiej funkcji w formułach arkuszowych, ponieważ jej nazwa będzie taka sama jak adres jednej z komórek arkusza. Jeżeli spróbujesz użyć funkcji o takiej nazwie, Excel wyświetli błąd #ADRES!.

Najlepszym rozwiązaniem jest jednak unikanie takich nazw funkcji, które mogą być traktowane jako odwołania do komórek czy zakresów (włączając w to nazwy zakresów). Poza tym powinieneś unikać używana nazw wbudowanych funkcji Excela. Pamiętaj, że jeżeli wystąpi konflikt nazw, Excel zawsze użyje funkcji wbudowanej.

Zasięg funkcji

W rozdziale 7. omawialiśmy zagadnienia związane z zasięgiem procedur (publiczne lub prywatne). To samo dotyczy funkcji — zasięg funkcji określa, czy funkcja może być wywoływana przez procedury z innych modułów lub arkuszy.

Oto kilka informacji o zasięgu funkcji, o których należy pamiętać.

- Jeżeli nie zadeklarujesz zasięgu funkcji, domyślnie użyta zostanie deklaracja Public.
- Funkcje zadeklarowane przy użyciu słowa kluczowego Private nie są wyświetlane na liście funkcji w oknie dialogowym *Wstawianie funkcji* Excela. Z tego powodu w deklaracjach funkcji, które powinny być używana tylko w procedurach języka VBA, należy zastosować słowo kluczowe Private, tak aby użytkownicy nie mogli używać takich funkcji w formułach arkuszowych.
- Jeżeli kod źródłowy języka VBA wymaga wywołania funkcji zdefiniowanej w innym skoroszycie, powinieneś utworzyć do niego odwołanie za pomocą polecenia *References* zawartego w menu *Tools* edytora Visual Basic.
- Jeżeli funkcja jest zdefiniowana w aktywnym dodatku, nie musisz tworzyć do niego odwołania — taka funkcja będzie automatycznie dostępna dla wszystkich skoroszytów.

Wywoływanie procedur Function

Jak pamiętasz, procedura Sub może zostać uruchomiona na wiele sposobów, jednak w przypadku procedury Function dostępne są tylko cztery metody:

- Wywołanie funkcji z poziomu innej procedury.
- Użycie funkcji w formule arkuszowej.
- Użycie w formule wykorzystywanej do określenia formatowania warunkowego.
- Wywołanie funkcji z poziomu okna *Immediate* edytora VBE.

Wywołanie funkcji z poziomu innej procedury

Funkcje niestandardowe mogą być wywoływane z poziomu procedury w taki sam sposób, jak funkcje wbudowane. Przykładowo po zdefiniowaniu funkcji o nazwie SUMARRAY można zastosować następującą instrukcję:

```
Total = SUMARRAY(MyArray)
```

Instrukcja wykonuje funkcję SUMARRAY, pobierającą argument MyArray, zwraca jej wynik i przypisuje go zmiennej Total.

W razie potrzeby możesz również użyć metody Run obiektu Application. Oto przykład:

```
Total = Application.Run("SUMARRAY", "MyArray")
```

Pierwszy argument metody Run jest nazwą funkcji. Kolejne argumenty reprezentują argument(y) funkcji. Argumenty metody Run mogą być literałami (jak w powyższym przykładzie), liczbami lub zmiennymi.

Wywołanie funkcji z poziomu formuły arkusza

Stosowanie niestandardowych funkcji w formule arkusza jest podobne, jak w przypadku wbudowanych funkcji, z tą różnicą, że musisz się upewnić, czy Excel potrafi zlokalizować procedurę Function. Jeżeli procedura Function znajduje się w tym samym skoroszycie, nie trzeba podejmować żadnych dodatkowych kroków. Jeżeli jest umieszczona w innym skoroszycie, musisz poinformować Excela o jej lokalizacji.

Möesz to zrobić na trzy sposoby:

- **Poprzez umieszczenie przed nazwą funkcji odwołania do pliku** — na przykład aby skorzystać z funkcji o nazwie COUNTNAMES zdefiniowanej w otwartym skoroszycie Myfuncs.xlsm, możesz użyć następującego odwołania:
`=Myfuncs.xls!COUNTNAMES(A1:A1000)`
- Jeżeli wstawisz funkcję do formuły za pomocą okna dialogowego *Wstawianie funkcji*, odwołanie do skoroszytu zostanie wstawione automatycznie.
- **Poprzez zdefiniowanie odwołania do skoroszytu** — aby to zrobić, z menu *Tools* edytora Visual Basic powinieneś wybrać polecenie *References*. Jeżeli funkcja znajduje się w skoroszycie, do którego zdefiniowano odwołanie, nie będzie konieczne użycie nazwy arkusza. Nawet jeżeli utworzyłeś odwołanie do skoroszytu zależnego, po wybraniu funkcji w oknie dialogowym *Wstawianie funkcji* Excel i tak utworzy odwołanie do skoroszytu (choć jego użycie nie jest konieczne).

- **Poprzez utworzenie dodatku** — Jeżeli w trakcie tworzenia dodatku na bazie skoroszytu zawierającego procedury Function w formule zostanie użyta jedna z tych funkcji, nie jest konieczne definiowanie odwołania do pliku skoroszytu. Oczywiście w takiej sytuacji dodatek musi wcześniej zostać zainstalowany. Więcej szczegółowych informacji na temat dodatków znajdziesz w rozdziale 19.

W przeciwnieństwie do procedur Sub procedury Function nie pojawiają się w oknie dialogowym *Makro* (aby przywołać to okno dialogowe na ekran, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Makra*, znajdujący się w grupie opcji *Kod*). Co więcej, jeżeli w edytorze VBE umieścisz kurSOR w obszarze procedury Function, to i tak nie będziesz mógł jej uruchomić po wykonaniu polecenia *Run Sub/UserForm* lub wcisnięciu klawisza *F5* (na ekranie pojawi się po prostu okno dialogowe *Macro*, umożliwiające wybranie makra, które zostanie uruchomione). Jak widać, w celu przetestowania tworzonych funkcji konieczne będzie wykonanie dodatkowych operacji. Jedna z metod polega na stworzeniu prostej procedury wywołującej funkcję. Jeżeli funkcja została stworzona z myślą o użyciu jej w formułach arkusza, do sprawdzenia poprawności działania funkcji możesz użyć prostej formuły.

Wywołanie funkcji z poziomu formuły formatowania warunkowego

Kiedy korzystasz z mechanizmu formatowania warunkowego, jedną z dostępnych opcji jest utworzenie formuły określającej komórki, które należy formatować. Taka formuła musi zwracać wartość logiczną (PRAWDA lub FAŁSZ). Jeżeli formuła zwraca wartość PRAWDA, warunek zostaje spełniony i komórce nadawane jest odpowiednie formatowanie.

W formułach formatowania warunkowego możesz używać własnych, niestandardowych funkcji VBA. Poniżej przedstawiamy przykład prostej funkcji VBA, która zwraca wartość PRAWDA, jeżeli jej argumentem jest adres komórki zawierającej formułę:

```
Function CELLHASFORMULA(cell) As Boolean  
    CELLHASFORMULA = cell.HasFormula  
End Function
```

Po zdefiniowaniu funkcji w module VBA możesz utworzyć regułę formatowania warunkowego, tak aby komórki, w których przechowywane są formuły, otrzymywały inne formatowanie:

1. Zaznacz zakres komórek, któremu chcesz nadać formatowanie warunkowe, na przykład zaznacz zakres A1:G20.
2. Przejdź na kartę *NARZĘDZIA GŁÓWNE* i wybierz polecenie *Formatów warunk.*, znajdujące się w grupie opcji *Style*, a następnie z menu podręcznego, które pojawi się na ekranie, wybierz polecenie *Nowa reguła*.
3. Na ekranie pojawi się okno dialogowe *Nowa reguła formatowania*. Z listy *Wybierz typ reguły* wybierz opcję *Użyj formuły do określenia komórek, które należy sformatować*.

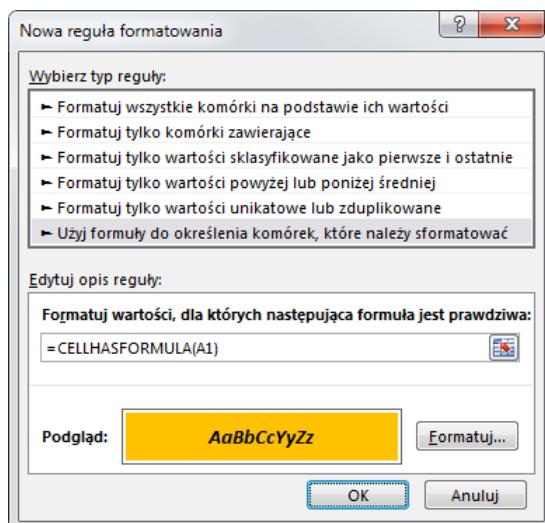
- 4.** W polu *Formatuj wartości, dla których następująca formuła jest prawdziwa* wprowadź formułę przedstawioną poniżej — upewnij się, że adres komórki będący argumentem funkcji odpowiada górnjej, lewej komórce zakresu zaznaczonego w punkcie 1.

=CELLHASFORMULA(A1)

- 5.** Naciśnij przycisk *Formatuj* i określ sposób formatowania komórek spełniających warunek.
- 6.** Naciśnij przycisk *OK*, aby nadać formatowanie zaznaczonemu zakresowi.

Komórki w zaznaczonym zakresie, w których przechowywane są formuły, otrzymają formatowanie takie, jak zdefiniowałeś. Na rysunku 8.3 przedstawiono okno dialogowe *Nowa reguła formatowania*, w którym w formule formatowania warunkowego użyta została nasza niestandardowa funkcja VBA.

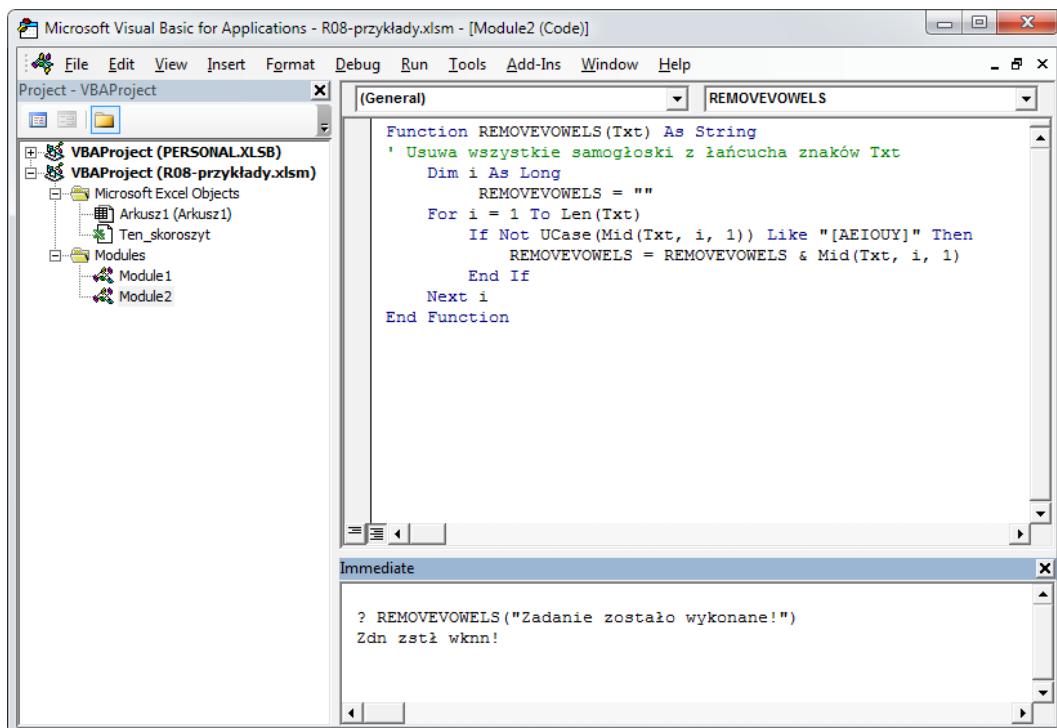
Rysunek 8.3.
Wykorzystanie niestandardowej funkcji VBA w formule formatowania warunkowego



Funkcja ISFORMULA to jedna z nowych funkcji arkuszowych, wprowadzonych w Excelu 2013, która działa w dokładnie taki sam sposób jak omawiana przed chwilą niestandardowa funkcja CELLHASFORMULA. Nie zmienia to jednak w niczym faktu, że funkcja CELLHASFORMULA nadal może być bardzo użyteczna, zwłaszcza w sytuacji, kiedy planujesz udostępnić dany skoroszyt użytkownikom, którzy nie korzystają jeszcze z Excela 2013.

Wywołanie funkcji z poziomu okna Immediate

Ostatnim sposobem uruchamiania funkcji jest wywołanie jej z poziomu okna *Immediate* edytora VBE. Metoda ta używana jest najczęściej do celów testowania poprawności działania funkcji. Na rysunku 8.4 przedstawiono przykład takiego wywołania funkcji. Znak ? reprezentuje polecenie drukowania (wyświetlania na ekranie) danej wartości.



Rysunek 8.4. Wywoływanie funkcji z poziomu okna Immediate edytora VBE

Argumenty funkcji

Pamiętaj o następujących kwestiach związanych z argumentami procedury Function:

- Argumenty mogą być zmiennymi (w tym tablicami), stałymi, literałami lub wyrażeniami.
- Niektóre funkcje nie posiadają argumentów.
- Niektóre funkcje posiadają stałą liczbę wymaganych argumentów (od 1 do 60).
- Funkcje mogą posiadać zarówno wymagane, jak i opcjonalne argumenty.



Jeżeli formuła używa niestandardowej funkcji i zwraca wartość błędu #ARG!, oznacza to, że w funkcji wystąpił błąd na skutek błędów kodu źródłowego, przekazania funkcji niepoprawnych argumentów lub wykonania niedozwolonej operacji, takiej jak próba zmiany formatowania komórki. Więcej informacji na ten temat znajdziesz w podrozdziale „Wykrywanie i usuwanie błędów w funkcjach” w dalszej części rozdziału.

Wyważanie otwartych drzwi...

W zasadzie dla zabawy napisałem własną wersję funkcji arkuszowej LITERY.WIELKIE (która zamienia wszystkie znaki łańcucha tekstu na wielkie litery) i nazwałem ją UPCASE. Oto kod tej funkcji:

```
Function UPCASE(InString As String) As String
    ' Zamienia małe litery na wielkie (z wyjątkiem polskich liter) w ciągu znaków będącym argumentem funkcji
    Dim StringLength As Long
    Dim i As Long
    Dim ASCIIIVal As Long
    Dim CharVal As Long

    StringLength = Len(InString)
    UPCASE = InString
    For i = 1 To StringLength
        ASCIIIVal = Asc(Mid(InString, i, 1))
        CharVal = 0
        If ASCIIIVal >= 97 And ASCIIIVal <= 122 Then
            CharVal = -32
            Mid(UPCASE, i, 1) = Chr(ASCIIIVal + CharVal)
        End If
    Next i
End Function
```



Skoroszyt z tym przykładem (*Funkcja UpCase.xls*) znajdziesz na stronie internetowej tej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zauważ, że oparłem się pokusie pójścia na łatwiznę i nie użyłem funkcji UCASE języka VBA.

Ponieważ zastanawiałem się, czym funkcja niestandardowa różni się od funkcji wbudowanej, utworzyłem arkusz wywołujący 20 000 razy moją nową funkcję UPCASE, której argumentem były losowe łańcuchy znaków. Obliczenia w arkuszu zajęły około 20 sekund. Następnie wstawiłem do arkusza standardową funkcję LITERY.WIELKIE i ponownie wykonałem taki sam test. Tym razem obliczenia zostały wykonane prawie natychmiast. Nie twierdzę, że moja funkcja UPCASE jest optymalnym rozwiązaniem tego zadania, ale bez obaw mogę powiedzieć, że funkcja niestandardowa pod względem szybkości nigdy nie dorówna wbudowanym funkcjom Excela.

Aby zapoznać się z kolejnym przykładem wyważania otwartych drzwi, zajrzyj do podrozdziału „Emulacja funkcji arkuszowej SUMA”, znajdującego się w dalszej części rozdziału.

Przykłady funkcji

W tym podrozdziale zaprezentujemy kilka przykładów efektywnego używania argumentów funkcji. Nawiąsem mówiąc, omawiane zagadnienia mają zastosowanie również dla procedur Sub.

Funkcja bezargumentowa

Podobnie jak w przypadku procedur Sub, nie wszystkie procedury Function wymagają stosowania argumentów. Excel posiada wiele wbudowanych funkcji, które nie posiadają żadnych argumentów wywołania, na przykład LOS(), DZIŚ() czy TERAZ(). W języku VBA możesz tworzyć podobne funkcje.

W tym podrozdziale przedstawimy przykłady funkcji, które nie używają argumentów.



Skoroszyt z przykładami (*Funkcje bez argumentów.xlsx*) znajdziesz na stronie internetowej tej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Oto prosty przykład funkcji, która nie używa argumentów. Funkcja zwraca właściwość `UserName` obiektu `Application`. Wartość tej właściwości jest przechowywana w rejestrze systemu Windows i można ją również wyświetlić w oknie dialogowym *Opcje programu Excel* (kategoria *Ogólne*)

```
Function USER()
    ' Zwraca nazwę aktualnego użytkownika programu Excel
    User = Application.UserName
End Function
```

Po wprowadzeniu poniższej formuły w komórce pojawi się nazwa aktualnego użytkownika (oczywiście przy założeniu, że nazwa jest poprawnie przechowywana w rejestrze systemu Windows):

```
=USER()
```



W przypadku użycia w formule arkusza funkcji bezargumentowej, konieczne jest dołączenie pary pustych nawiasów okrągłych. Co prawda dodawanie nawiasów nie jest konieczne, gdy funkcja jest wywoływana z procedury języka VBA, ale użycie nawiasów po prostu podkreśla fakt, że wywołujemy funkcję.

Oczywiście w praktyce nie ma potrzeby stosowania takiej funkcji w programach VBA, ponieważ zamiast tego możesz po prostu w odpowiednim miejscu kodu bezpośrednio odczytywać wartość właściwości `UserName`.

Funkcja `USER` pokazuje, w jaki sposób możesz utworzyć *funkcję osłonową* (ang. *wrapper function*), która po prostu zwraca właściwość lub wynik działania funkcji VBA. Poniżej zamieszczono kody trzech innych funkcji osłonowych, które nie pobierają żadnych argumentów.

```
Function EXCELDIR() As String
    ' Zwraca nazwę katalogu, w którym jest zainstalowany Excel
    EXCELDIR = Application.Path
End Function

Function SHEETCOUNT()
    ' Zwraca liczbę arkuszy skoroszytu
    SHEETCOUNT = Application.Caller.Parent.Parent.Sheets.Count
End Function

Function SHEETNAME()
    ' Zwraca nazwę skoroszytu
    SHEETNAME = Application.Caller.Parent.Name
End Function
```

Nietrudno sobie wyobrazić inne, potencjalnie bardzo użyteczne funkcje osłonowe. Na przykład możesz napisać funkcję, która będzie wyświetlała lokalizację szablonów (właściwość `Application.TemplatesPath`), domyślną lokalizację plików (właściwość `Application.DefaultFilePath`) czy wersję Excela (właściwość `Application.Version`). Warto rów-

nież zwrócić uwagę na fakt, że w Excelu 2013 pojawiła się nowa funkcja SHEETS, która powoduje, że nasza niestandardowa funkcja SHEETCOUNT staje się zupełnie niepotrzebna.

A oto kolejny przykład funkcji, która nie pobiera żadnych argumentów. Do wypełnienia losowymi wartościami wybranego zakresu komórek, użyłem funkcji arkuszowej LOS() Excela. Jednak nie podoba mi się fakt, że za każdym razem po przeliczeniu arkusza wartości były losowo zmieniane. Aby tego uniknąć, zwykle musiałem zamieniać formuły na wartości.

Później uświadomiłem sobie, że mogę stworzyć funkcję niestandardową zwracającą losowe wartości, które nie będą później zmieniane. W kodzie funkcji zastosowałem wbudowaną funkcję Rnd języka VBA, która zwraca losowe wartości z przedziału od 0 do 1. Kod takiej niestandardowej funkcji ma następującą postać:

```
Function STATICRAND()
    ' Zwraca losową wartość, która nie jest zmieniana po przeliczeniu arkusza
    STATICRAND = Rnd()
End Function
```

Sterowanie ponownym przeliczaniem funkcji

Jeżeli użyjesz niestandardowej funkcji w formule, to kiedy taka funkcja będzie przeliczana?

Funkcje niestandardowe zachowują się podobnie, jak wbudowane funkcje arkuszowe Excela. Zwykłe niestandardowa funkcja ponownie wykonuje obliczenia wtedy, gdy jest to konieczne, czyli na przykład kiedy zostanie zmodyfikowany dowolny jej argument. W razie potrzeby możesz jednak wymusić częstsze ponowne wykonywanie funkcji. Umieszczenie w procedurze Function poniżej instrukcji spowoduje, że funkcja zostanie ponownie uruchomiona za każdym razem, kiedy przeliczany jest arkusz. Jeżeli używasz trybu automatycznego przeliczania, przeliczanie arkusza jest wykonywane po każdej modyfikacji dowolnej komórki:

```
Application.Volatile True
```

Metoda Volatile obiektu Application posiada jeden argument (True lub False). Przypisanie procedurze Function roli funkcji nietrwałej powoduje, że zostanie wykonana każdorazowo przy ponownym obliczaniu dowolnej komórki arkusza.

Na przykład niestandardowa funkcja STATICRAND może zostać przy użyciu metody Volatile tak zmieniona, aby emulowała działanie funkcji arkuszowej LOS()Excela:

```
Function NONSTATICRAND()
    ' Zwraca losową wartość, która zmienia się po każdym przeliczeniu arkusza
    Application.Volatile True
    NONSTATICRAND = Rnd()
End Function
```

Jeżeli argumentem metody Volatile będzie False, funkcja zostanie ponownie wykonana tylko wtedy, gdy jeden lub więcej jej argumentów zmieni się w wyniku tej operacji (jeżeli funkcja nie posiada argumentów, to nie będzie powtórnie wykonana).

Aby wymusić ponowne wykonanie wszystkich obliczeń uwzględniających niestandardowe funkcje trwałe, naciśnij kombinację klawiszy **Ctrl+Alt+F9**. Jeżeli użyjesz tej kombinacji dla naszego przykładu, to funkcja STATICRAND wygeneruje nowe liczby losowe.

Aby wygenerować serię losowych liczb całkowitych z przedziału od 0 do 1000, możesz teraz zastosować następującą formułę:

```
=INT(STATICRAND()*1000)
```

W przeciwnieństwie do zastosowania wbudowanej funkcji LOS(), wartości uzyskane po wykonaniu formuły nigdy się nie zmienią. W razie potrzeby jednak możesz wymusić przeliczenie tej formuły, naciskając kombinację klawiszy *Ctrl+Alt+F9*.

Funkcje jednoargumentowe

W tym podrozdziale omówimy funkcję przeznaczoną dla kierowników sprzedaży, którzy muszą obliczać prowizję uzyskiwaną przez podlegających im przedstawicieli handlowych. Nasze obliczenia opierają się na następującej tabeli:

Miesięczna sprzedaż	Stawka prowizji
0 – 9999 złotych	8,0%
10 000 – 19 999 złotych	10,5%
20 000 – 39 999 złotych	12,0%
powyżej 40 000 złotych	14,0%

Zauważ, że wysokość prowizji zwiększa się w sposób nieliniowy i zależy od całkowitej sprzedaży miesięcznej. Pracownicy osiągający większą sprzedaż otrzymują wyższe prowizje.

Istnieje kilka metod obliczenia prowizji dla różnych wartości sprzedaży wprowadzanych do arkusza. Jeżeli nie przyłożyisz się do opracowania dobrego rozwiązania, możesz niepotrzebnie stracić masę czasu, a co gorsza, w efekcie utworzyć niezrozumiałą, złożoną formułę, jak na przykład:

```
=JEŻELI(ORAZ(A1>=0;A1<=9999,99);A1*0,08;
JEŻELI(ORAZ(A1>=10000;A1<=19999,99);A1*0,105;
JEŻELI(ORAZ(A1>=20000;A1<=39999,99);A1*0,12;
JEŻELI (A1>=40000;A1*0,14)))
```

Z kilku powodów nie jest to właściwe podejście do zagadnienia. Po pierwsze, formuła jest zbyt złożona, przez co trudno ją zrozumieć. Po drugie, wartości są na stałe wprowadzone do formuły, na skutek czego trudno ją zmodyfikować.

Lepszą metodą (która nie wykorzystuje języka VBA) jest zastosowanie do obliczenia prowizji funkcji przeszukującej tabelę. Przykładowo poniższa formuła w celu pobrania wartości prowizji z zakresu o nazwie Table używa funkcji WYSZUKAJ.PIONOWO, a następnie mnoży ją przez wartość komórki A1:

```
=WYSZUKAJ.PIONOWO(A1;Table;2)*A1
```

Kolejna metoda eliminująca konieczność stosowania funkcji przeszukującej tabelę polega na stworzeniu niestandardowej funkcji, na przykład takiej, jak to przedstawiamy poniżej:

```

Function COMMISSION(Sales)
    Const Tier1 = 0.08
    Const Tier2 = 0.105
    Const Tier3 = 0.12
    Const Tier4 = 0.14
    ' Obliczanie prowizji od sprzedaży
    Select Case Sales
        Case 0 To 9999.99: COMMISSION = Sales * Tier1
        Case 10000 To 19999.99: COMMISSION = Sales * Tier2
        Case 20000 To 39999.99: COMMISSION = Sales * Tier3
        Case Is >= 40000: COMMISSION = Sales * Tier4
    End Select
End Function

```

Po umieszczeniu funkcji w module VBA możesz jej użyć w formule arkusza lub wywołać z innych procedur języka VBA.

Gdy wprowadzisz do komórki poniższą formułę, uzyskasz wartość 3000, ponieważ sprzedaż o wartości 25000 pozwala uzyskać prowizję wynoszącą 12%.

```
=COMMISSION(25000)
```

Nawet jeżeli nie zamierzasz stosować w arkuszu funkcji niestandardowych, utworzenie procedury Function może znacznie uprościć kod źródłowy napisany w języku VBA. Jeżeli na przykład Twoja procedura języka VBA oblicza prowizje od sprzedaży, można użyć dokładnie takiej samej funkcji i wywołać ją z procedury VBA. Poniżej zamieszczono procedurę Sub proszającą użytkownika o wprowadzenie wartości sprzedaży, a następnie używającą funkcji COMMISSION do obliczenia prowizji:

```

Sub CalcComm()
    Dim Sales As Long
    Sales = InputBox("Wprowadź wartość sprzedaży:")
    MsgBox "Prowizja wynosi " & COMMISSION(Sales)
End Sub

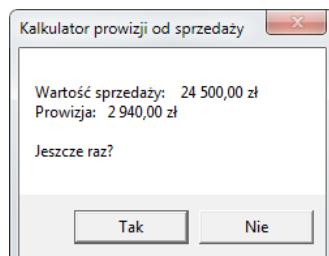
```

Po uruchomieniu, procedura o nazwie CalcComm wyświetla okno dialogowe, prosząc o wprowadzenie wartości sprzedaży. Następnie wyświetla okno komunikatu zawierające obliczoną wysokość prowizji dla podanej kwoty.

Procedura CalcComm działa, ale jest dosyć niestandardowo napisana. Poniżej znajduje się jej rozszerzona wersja, zawierająca prostą obsługę błędów, wyświetlającą sformatowane wartości i wykonującą pętlę do momentu kliknięcia przez użytkownika przycisku *Nie* (patrz rysunek 8.5).

Rysunek 8.5.

Zastosowanie funkcji do wyświetlenia wyniku obliczeń



```

Sub CalcComm()
    Dim Sales As Long
    Dim Msg As String, Ans As String

    ' Prośba o podanie wartości sprzedaży
    Sales = Val(InputBox("Wprowadź wartość sprzedaży:", _
        "Kalkulator prowizji od sprzedaży"))

    ' Jeżeli użytkownik nacisnął przycisk Cancel, kończymy działanie
    If Sales = 0 Then Exit Sub

    ' Tworzenie komunikatu
    Msg = "Wartość sprzedaży:" & vbTab & Format(Sales, "#.##0.00 zł")
    Msg = Msg & vbCrLf & "Prowizja:" & vbTab
    Msg = Msg & Format(COMMISSION(Sales), "#.##0.00 zł")
    Msg = Msg & vbCrLf & vbCrLf & "Jeszcze raz?"

    ' Wyświetla wynik i prosi o podanie kolejnej wartości sprzedaży
    Ans = MsgBox(Msg, vbYesNo, "Kalkulator prowizji od sprzedaży")
    If Ans = vbYes Then CalcComm
End Sub

```

Funkcja używa dwóch wbudowanych stałych języka VBA. Stała `vbTab` reprezentuje tabulator, umożliwiający tworzenie odstępów, natomiast stała `vbCrLf` wprowadza znaki powrotu karetki i wysuwu wiersza (powoduje przejście do następnego wiersza). Funkcja `Format` języka VBA wyświetla wartość w określonym formacie (w tym przypadku złożonym z przecinka, dwóch miejsc dziesiętnych i łańcucha `zł`).

W obu przykładach funkcja `Commission` musi być dostępna w aktywnym skoroszycie, w przeciwnym razie Excel wyświetli komunikat błędu informujący, że funkcja nie została zdefiniowana.

Używaj argumentów, a nie odwołań do komórek

Wszystkie zakresy używane w funkcjach niestandardowych powinny być przekazywane jako argumenty. Przyjrzyjmy się następującej funkcji, która zwraca wartość komórki A1 pomnożoną przez 2:

```

Function DOUBLECELL()
    DOUBLECELL = Range("A1") * 2
End Function

```

Tak zbudowana funkcja co prawda działa, ale w pewnych sytuacjach może zwracać niepoprawne wyniki. Silnik obliczeń Excela nie zawsze poprawnie odwołuje się w kodzie programu do zakresów, które nie są przekazywane jako argumenty. Z tego powodu w pewnych sytuacjach obliczenia poprzedzające wywołanie funkcji mogą nie być realizowane aż do momentu, kiedy funkcja zwróci swoją wartość. Inaczej mówiąc, funkcja `DOUBLECELL` powinna być napisana w następujący sposób, z odwołaniem do komórki A1 przekazywanym jako argument:

```

Function DOUBLECELL(cell)
    DOUBLECELL = cell * 2
End Function

```

Funkcje z dwoma argumentami

Wyobraź sobie, że wspomniani wcześniej hipotetyczni kierownicy sprzedazy z poprzedniego przykładu wprowadzili nową zasadę pozwalającą zwiększyć obroty. Polega ona na tym, że całkowita prowizja jest zwiększana o jeden procent po każdym kolejnym roku przepracowanym przez przedstawiciela w firmie.

Dla naszych potrzeb zmodyfikowałem niestandardową funkcję COMMISSION zdefiniowaną w poprzednim podpunkcie tak, aby pobierała dwa argumenty. Nowy argument reprezentuje liczbę lat pracy. Wywołaj nową funkcję COMMISSION2, która wygląda następująco:

```
Function COMMISSION2(Sales, Years)
    ' Oblicza prowizje od sprzedaży w oparciu o lata pracy
    Const Tier1 = 0.08
    Const Tier2 = 0.105
    Const Tier3 = 0.12
    Const Tier4 = 0.14
    Select Case Sales
        Case 0 To 9999.99: COMMISSION2 = Sales * Tier1
        Case 10000 To 19999.99: COMMISSION2 = Sales * Tier2
        Case 20000 To 39999.99: COMMISSION2 = Sales * Tier3
        Case Is >= 40000: COMMISSION2 = Sales * Tier4
    End Select
    COMMISSION2 = COMMISSION2 + (COMMISSION2 * Years / 100)
End Function
```

Całkiem proste, prawda? W deklaracji funkcji dodaliśmy jedynie drugi argument (Years) oraz dołączylismy do kodu funkcji instrukcje odpowiednio modyfikujące sposób obliczania prowizji.

Poniżej zamieszczamy przykład formuły używającej funkcji COMMISSION2 (zakładamy, że wartość sprzedaży znajduje się w komórce A1, natomiast liczba lat przepracowanych przez przedstawiciela w komórce B1):

=COMMISSION2(A1, B1)



Skoroszyt z tymi przykładami (*Obliczanie prowizji.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Funkcja pobierająca tablicę jako argument

Procedura Function może jako argument pobierać jedną lub kilka tablic, przetwarzać je i jako wynik zwracać pojedynczą wartość. Tablica może również składać się z zakresu komórek.

Poniższa funkcja jako argument pobiera tablicę i zwraca sumę jej elementów:

```
Function SUMARRAY(List) As Double
    Dim Item As Variant
    SUMARRAY = 0
    For Each Item In List
        If WorksheetFunction.IsNumber(Item) Then _
```

```

    SUMARRAY = SUMARRAY + Item
Next Item
End Function

```

Dzięki funkcji arkuszowej ISNUMBER (CZY.LICZBA), która przed dodaniem każdego elementu tablicy do sumy całkowitej sprawdza, czy element jest liczbą, można wyeliminować błąd pomieszanego typów, występujący podczas prób wykonania operacji arytmetycznej na danych innego typu niż liczby.

Poniższa procedura demonstruje, w jaki sposób wywołać tę funkcję z poziomu procedury Sub. Procedura MakeList tworzy tablicę złożoną ze 100 elementów i każdemu z nich przypisuje losową wartość. Później poprzez wywołanie funkcji SUMARRAY funkcja MsgBox wyświetla sumę wartości elementów tablicy.

```

Sub MakeList()
    Dim Nums(1 To 100) As Double
    Dim i as Integer
    For i = 1 To 100
        Nums(i) = Rnd * 1000
    Next i
    MsgBox SUMARRAY(Nums)
End Sub

```

Zauważ, że funkcja SUMARRAY nie deklaruje typu danych swoich argumentów (stosowany jest typ Variant). Dzięki takiemu rozwiązaniu może zostać użyta w formułach arkusza, gdzie argumentami będą obiekty klasy Range. Na przykład poniższa formuła zwraca sumę wartości komórek zakresu A1:C10:

=SUMARRAY(A1:C10)

Być może zauważysz, że funkcja SUMARRAY zastosowana w formule arkusza działa bardzo podobnie do funkcji arkuszowej SUMA Excela. Jedyna różnica polega na tym, że funkcja SUMARRAY akceptuje tylko jeden argument. Pamiętaj jednak, że powyższy przykład został przytoczony tylko dla celów edukacyjnych. Zastosowanie w formułach funkcji SUMARRAY zamiast funkcji SUMA Excela nie daje absolutnie żadnych korzyści.



Skoroszyt z tym przykładem (*Argumenty tablicowe.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Funkcje z argumentami opcjonalnymi

Wiele wbudowanych funkcji arkuszowych Excela używa argumentów opcjonalnych. Przykładem jest funkcja LEWY, która zwraca znaki, począwszy od lewej strony łańcucha. Oto jej składnia:

`LEWY(tekst;liczba_znaków)`

Pierwszy argument jest wymagany, natomiast drugi jest już opcjonalny. Jeżeli argument opcjonalny zostanie pominięty, Excel użyje wartości 1. A zatem poniższe dwie formuły zwrócią taki sam wynik:

`=LEWY(A1;1)`
`=LEWY(A1)`

Nazwy funkcji pisane wielkimi literami?

Zapewne zwróciłeś uwagę na fakt, że nazwy wbudowanych funkcji Excela są zawsze pisane wielkimi literami. Nawet jeśli wpisujesz nazwę takiej funkcji małymi literami, Excel automatycznie dokonuje jej konwersji na wielkie litery.

Tworząc własne, niestandardowe funkcje arkuszowe, możesz w ich nazwach używać wielkich liter, małych liter lub nawet jednych i drugich jednocześnie. Kiedy tworzę własne funkcje, które będą używane w formułach arkusza, zazwyczaj nadaję im nazwy pisane wielkimi literami, tak aby pasowały do stylu nazw wbudowanych funkcji Excela.

Czasami jednak, kiedy wpisuję formułę wykorzystującą własne, niestandardowe funkcje, Excel nie zachowuje oryginalnej pisowni wielkich i małych liter, która została użyta w kodzie VBA. Przyczyna takiego zachowania Excela pozostaje niestety nieznana, ale udało mi się znaleźć sposób na omijanie tego problemu. Założymy, że utworzyłeś funkcję o nazwie MOJAFUNKCJA, której nazwa w kodzie VBA została zapisana wielkimi literami. Kiedy jednak spróbujesz wpisać małymi literami nazwę tej funkcji w formule arkuszowej, Excel *nie wyświetli jej wielkimi literami*. A oto jak uniknąć takiego zachowania.

Przejdź do okna Excela, kliknij kartę *FORMUŁY*, wybierz polecenie *Definiuj nazwę* znajdujące się w grupie poleceń *Nazwy zdefiniowane*, a następnie utwórz nazwę *MOJAFUNKCJA* (pisząc ją oczywiście wielkimi literami). Obszar, do którego będzie się odwoływała taka nazwa, nie ma żadnego znaczenia. Od tego momentu wszystkie formuły, które wykorzystują funkcję *MOJAFUNKCJA*, będą wyświetlały błędy (jak można było tego oczekiwać). Zwróć jednak uwagę, że od tej chwili wszystkie formuły wyświetlają nazwę funkcji wielkimi literami!

Teraz ponownie przejdź na kartę *FORMUŁY*, wybierz polecenie *MENEDŻER NAZW* znajdujące się w grupie poleceń *Nazwy zdefiniowane i usuń nazwę* *MOJAFUNKCJA*. Od tego momentu formuły zawierające tę funkcję przestaną wyświetlać błędy, a nazwa funkcji będzie nadal wyświetlana wielkimi literami!

Nie mam pojęcia, dlaczego tak się dzieje — po prostu musimy przyjąć do wiadomości, że to działa ☺.

Niestandardowe funkcje tworzone w języku VBA też mogą używać opcjonalnych argumentów. W celu zdefiniowania opcjonalnego argumentu przed jego nazwą należy umieścić słowo kluczowe *Optional*. Na liście argumenty opcjonalne muszą znaleźć się za wymaganymi.

Poniżej zamieszczono przykład prostej funkcji zwracającej nazwę użytkownika (argument funkcji jest opcjonalny):

```
Function USER(Optional UpperCase As Variant)
    If IsMissing(UpperCase) Then UpperCase = False
    USER = Application.UserName
    If UpperCase Then USER = UCase(USER)
End Function
```

Jeżeli wartością argumentu jest *False* lub jeżeli argument zostanie pominięty, nazwa użytkownika będzie zwrocona w niezmienionej postaci. Jeżeli argument ma wartość *True*, przed zwróceniem nazwy użytkownika znaki w niej zawarte zostaną zamienione na wielkie (przy użyciu funkcji *UCase* języka VBA). Pierwsza instrukcja procedury używa funkcji *IsMissing* języka VBA, aby stwierdzić, czy podczas wywołania przekazany został argument. Jeżeli argument nie został użyty, instrukcja przypisze zmiennej *UpperCase* domyślną wartość *False*.

Wszystkie poniższe formuły są poprawne (dwie pierwsze dają taki sam wynik):

```
=USER()
=USER(False)
=USER(True)
```



Aby stwierdzić, czy podczas wywołania do funkcji został przekazany opcjonalny argument, trzeba go zadeklarować przy użyciu typu Variant. Dopiero po takiej deklaracji można w procedurze użyć funkcji IsMissing zaprezentowanej w powyższym przykładzie. Inaczej mówiąc, argumenty funkcji IsMissing muszą zawsze być typu Variant.

Poniżej zamieszczono kolejny przykład niestandardowej funkcji stosującej argument opcjonalny. Z wprowadzonego zakresu funkcja losowo wybiera jedną komórkę i zwraca jej zawartość. Jeżeli drugi argument ma wartość True, wybrana wartość zmieni się po każdorazowym ponowieniu obliczeń wykonywanych w arkuszu (oznacza to, że funkcja jest nietrwała). Jeżeli drugi argument będzie miał wartość False lub zostanie pominięty, funkcja nie zostanie ponownie uruchomiona do momentu zmodyfikowania zawartości jednej z komórek wprowadzonego zakresu.

```
Function DRAWONE(Rng As Variant, Optional Recalc As Variant = False)
    ' Losowo wybiera z zakresu jedną komórkę
    ' Kiedy opcjonalny argument Recalc ma wartość True, funkcja staje się nietrwała
    Application.Volatile Recalc
    ' Określa losowo wybraną komórkę
    DRAWONE = Rng(Int((Rng.Count) * Rnd + 1))
End Function
```

Należy zauważyć, że drugi argument funkcji DRAWONE zawiera słowo kluczowe Optional wraz z wartością domyślną.

Wszystkie poniższe formuły są poprawne (dwie pierwsze dają taki sam wynik):

```
=DRAWONE(A1:A100)
=DRAWONE(A1:A100, False)
=DRAWONE(A1:A100, True)
```

Funkcja może być przydatna w przypadku losowania liczb, wybierania zwycięzcy losowania z listy nazwisk itp.



Skoroszyt z tym przykładem (*Losowanie.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

W sieci

Funkcje zwracające tablice VBA

Język VBA posiada bardzo użyteczną funkcję Array, która zwraca zmienną typu Variant, zawierającą tablicę (nie mylić z tablicą zawierającą elementy typu Variant). Jeżeli znasz formuły tablicowe Excela, łatwiej zrozumiesz funkcję Array języka VBA. Formuła tablicowa jest wprowadzana do komórki poprzez wciśnięcie kombinacji klawiszy *Ctrl+Shift+Enter*. Aby odróżnić formułę tablicową od innych, Excel umieszcza ją w nawiasach klamrowych.



Więcej szczegółowych informacji na temat formuł tablicowych znajdziesz w rozdziale 2.



Ważne jest, abyś zrozumiał, że tablica zwrocona przez funkcję Array nie jest tym samym, co zwykła tablica złożona z elementów typu Variant. Innymi słowy, zmienna typu Variant zawierająca tablicę to nie jest to samo, co tablica zmiennych typu Variant.

Poniższa funkcja MONTHNAMES jest prostym przykładem zastosowania funkcji Array języka VBA w funkcji niestandardowej:

```
Function MONTHNAMES()
    MONTHNAMES = Array("Styczeń", "Luty", "Marzec",
        "Kwiecień", "Maj", "Czerwiec", "Lipiec", "Sierpień",
        "Wrzesień", "Październik", "Listopad", "Grudzień")
End Function
```

Funkcja MONTHNAMES zwraca poziomą tablicę zawierającą nazwy miesięcy. Możesz utworzyć wielokomórkową formułę tablicową wykorzystującą funkcję MONTHNAMES. Aby użyć takiej funkcji najpierw sprawdź, czy jej kod źródłowy znajduje się w module kodu VBA. Następnie w arkuszu zaznacz zakres komórek z wiersza (na początek wybierz 12 komórek). Na końcu wprowadź poniższą formułę (bez nawiasów klamrowych) i naciśnij kombinację klawiszy *Ctrl+Shift+Enter*:

```
{=MONTHNAMES()}
```

A co należy zrobić, aby wygenerować pionową listę nazw miesięcy? Nie stanowi to problemu. W tym celu wystarczy zaznaczyć pionowy zakres, wprowadzić poniższą formułę (bez nawiasów klamrowych) i nacisnąć kombinację klawiszy *Ctrl+Shift+Enter*:

```
{=TRANSPONUJ(MONTHNAMES())}
```

Powyzsza formuła za pomocą funkcji arkuszowej TRANSPONUJ zamienia poziomą tablicę na pionową.

Poniższy przykład jest zmodyfikowaną wersją funkcji MONTHNAMES:

```
Function MONTHNAMES(Optional MIndex)
    Dim AllNames As Variant
    Dim MonthVal As Long
    AllNames = Array("Styczeń", "Luty", "Marzec",
        "Kwiecień", "Maj", "Czerwiec", "Lipiec", "Sierpień",
        "Wrzesień", "Październik", "Listopad", "Grudzień")
    If IsMissing(MIndex) Then
        MONTHNAMES = AllNames
    Else
        Select Case MIndex
            Case Is >= 1
                Określa wartość miesiąca (na przykład, 13=1)
                MonthVal = ((MIndex - 1) Mod 12)
                MONTHNAMES = AllNames(MonthVal)
            Case Is <= 0 ' Tablica pionowa
                MONTHNAMES = Application.Transpose(AllNames)
        End Select
    End If
End Function
```

Zauważ, że do sprawdzenia, czy argument opcjonalny został pominięty, użyłem funkcji IsMissing języka VBA. W tej sytuacji określenie na liście argumentów funkcji domyślnej wartości brakującego argumentu nie jest możliwe, ponieważ taka wartość domyślna została zdefiniowana we wnętrzu funkcji. Funkcja IsMissing może zostać użyta tylko wtedy, gdy argument opcjonalny jest typu Variant.

Rozszerzona funkcja MONTHNAMES, używająca argumentu opcjonalnego działa w następujący sposób:

- **Jeżeli argument zostanie pominięty** — funkcja zwraca poziomą tablicę zawierającą nazwy miesięcy.
- **Jeżeli wartość argumentu jest mniejsza lub równa zeru** — funkcja zwraca pionową tablicę zawierającą nazwy miesięcy. Funkcja używa funkcji Transpose (TRANSPONUJ) Excela, która dokonuje odpowiedniej konwersji tablicy.
- **Jeżeli wartość argumentu jest większa lub równa 1** — funkcja zwraca nazwę miesiąca odpowiadającą wartości argumentu.



Uwaga

Do określenia wartości miesiąca procedura używa operatora Mod. Operator Mod zwraca resztę z dzielenia pierwszego argumentu przez drugi. Przykładowo dla argumentu o wartości 13 jest zwracana liczba 1. Dla argumentu o wartości 23 jest zwracana liczba 11 itd. Numerowanie elementów tablicy AllNames rozpoczyna się od zera i kończy na liczbie 11. W instrukcji używającej operatora Mod od argumentu funkcji jest odejmowana liczba 1. A zatem dla argumentu o wartości 13 zostanie zwrócone 0 (odpowiada nazwie Styczeń), natomiast dla argumentu o wartości 24 liczba 11 (odpowiada nazwie Grudzień).

Funkcja MONTHNAMES może zostać użyta na kilka sposobów, tak jak to zostało przedstawione na rysunku 8.6.

=MONTHNAMES()											
A	B	C	D	E	F	G	H	I	J	K	L
1	Styczeń	Luty	Marzec	Kwiecień	Maj	Czerwiec	Lipiec	Sierpień	Wrzesień	Październik	Listopad
2											
3	1 Styczeń			Styczeń		Marzec					
4	2 Luty			Luty							
5	3 Marzec			Marzec							
6	4 Kwiecień			Kwiecień							
7	5 Maj			Maj							
8	6 Czerwiec			Czerwiec							
9	7 Lipiec			Lipiec							
10	8 Sierpień			Sierpień							
11	9 Wrzesień			Wrzesień							
12	10 Październik			Październik							
13	11 Listopad			Listopad							
14	12 Grudzień			Grudzień							
15											
16											

Rysunek 8.6. Różne sposoby przekazywania tablicy lub pojedynczej wartości do arkusza

Zakres A1:L1 zawiera poniższą formułę tablicową. Najpierw zaznacz zakres A1:L1, wstaw formułę (bez nawiasów klamrowych), a następnie naciśnij kombinację klawiszy **Ctrl+Shift+Enter**:

```
{=MONTHNAMES( )}
```

Zakres A3:A14 zawiera liczby całkowite z zakresu od 1 do 12. Komórka B3 zawiera następującą formułę, którą skopiowano do pozostałych 11 komórek znajdujących się niżej:

=MONTHNAMES(A3)

Zakres D3:D14 zawiera następującą formułę tablicową:

{=MONTHNAMES(-1)}

W komórce F3 znajduje się następująca formuła (to nie jest formuła tablicowa):

=MONTHNAMES(3)



Aby wprowadzić formułę tablicową, naciśnij kombinację klawiszy *Ctrl+Shift+Enter* (nie wprowadzaj ręcznie nawiasów klamrowych).



Dolna granica tablicy utworzonej przy użyciu funkcji Array jest określana przez wartość podaną w klauzuli Option Base umieszczonej na początku modułu. Jeżeli klauzula Option Base nie zostanie użyta, dolna granica domyślnie będzie miała wartość 0.



Skoroszyt z tym przykładem (*Nazwy miesięcy.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Funkcje zwracające wartość błędu

Czasami chcemy, aby niestandardowa funkcja zwracała określoną wartość błędu. Przyrzyj się ponownie funkcji REMOVEVOWELS, którą omawialiśmy na początku rozdziału.

```
Function REMOVEVOWELS(Txt) As String
    ' Usuwa samogłoski z łańcucha znaków Txt
    Dim i As Long
    REMOVEVOWELS = ""
    For i = 1 To Len(Txt)
        If Not UCASE(Mid(Txt, i, 1)) Like "[AEIOUY]" Then
            REMOVEVOWELS = REMOVEVOWELS & Mid(Txt, i, 1)
        End If
    Next i
End Function
```

Kiedy użyjemy funkcji REMOVEVOWELS w formule arkuszowej, nasza funkcja usunie samogłoski z pojedynczej komórki będącej jej argumentem. Jeżeli argument jest wartością numeryczną, funkcja zwróci ją w postaci łańcucha tekstu. W pewnych sytuacjach jednak lepiej będzie, aby funkcja zamiast wartości numerycznej zamienionej na postać tekstową zwracała odpowiedni kod błędu (#N/D!).

W takiej sytuacji można się pokusić po prostu o przypisanie zmiennej REMOVEVOWELS łańcucha tekstu wyglądającego jak wartość błędu formuły Excela. Oto przykład:

REMOVEVOWELS = "#N/D!"

Niestety taki łańcuch tekstu tylko *wygląda* jak wartość błędu, ale przez inne formuły, które mogą się do niego odwołać, nie jest traktowany jak błąd. Aby zwrócić rzeczywistą wartość błędu funkcji, należy użyć funkcji CVErr języka VBA, która zamienia numer błędu na prawdziwy komunikat o błędzie.

Na szczęście język VBA posiada wbudowane stałe odpowiadające poszczególnym kodom błędów, które może zwracać niestandardowa funkcja. Stałe reprezentują wartości błędów formuł Excela, a nie błędów uruchomieniowych interpretera języka VBA. Listę stałych zamieszczamy poniżej:

- xlErrDiv0 (#DZIEL/0!),
- xlErrNA (#N/D!),
- xlErrName (#NAZWA?),
- xlErrNull (#ZERO!),
- xlErrNum (#LICZBA!),
- xlErrRef (#ADRES!),
- xlErrValue (#ARG!).

Aby niestandardowa funkcja zwróciła wartość błędu #N/D!, należy użyć następującego polecenia:

```
REMOVEVOWELS = CVErr(xlErrNA)
```

Poniżej zamieszczono zmodyfikowaną wersję funkcji REMOVEVOWELS. Nowa funkcja używa struktury If ... Then do wykonania innej operacji w sytuacji, kiedy argument nie jest tekstem. W celu określenia, czy argument ma postać łańcucha tekstu, funkcja używa funkcji IsText (CZY.TEKST) Excela. Jeżeli argument jest tekstem, funkcja jest wykonywana w standardowy sposób. Jeżeli komórka nie zawiera tekstu lub jest pusta, funkcja zwraca wartość błędu #N/D!.

```
Function REMOVEVOWELS(Txt) As Variant
    ' Usuwa samogloski z łańcucha znaków Txt
    ' Jeżeli argument Txt nie jest łańcuchem tekstu, funkcja zwraca błąd #N/D!
    Dim i As Long
    REMOVEVOWELS = ""
    If Application.WorksheetFunction.IsText(Txt) Then
        For i = 1 To Len(Txt)
            If Not UCASE(Mid(Txt, i, 1)) Like "[AEIOUY]" Then
                REMOVEVOWELS = REMOVEVOWELS & Mid(Txt, i, 1)
            End If
        Next i
    Else
        REMOVEVOWELS = CVErr(xlErrNA)
    End If
End Function
```



Zauważ, że w nowej wersji kodu został również zmieniony typ wartości zwracanej przez funkcję. Ponieważ funkcja może teraz zwracać nie tylko łańcuch tekstu ale również wartość reprezentującą kod błędu, zmieniłem typ wartości zwracanej przez funkcję na Variant.

Funkcje o nieokreślonej liczbie argumentów

Niektóre funkcje arkuszowe Excela mogą pobierać nieokreśloną liczbę argumentów. Przykładem takiej funkcji jest popularna funkcja SUMA, której składnia jest następująca:

```
SUMA(wartość1; wartość2...)
```

Pierwszy argument jest wymagany, ale dodatkowo funkcja może mieć jeszcze maksymalnie 254 dodatkowe argumenty. Oto przykład funkcji SUMA posiadającej 4 argumenty będące zakresami:

```
=SUMA(A1:A5;C1:C5;E1:E5;G1:G5)
```

Istnieje nawet możliwość stosowania argumentów różnych typów. Przykładowo poniższa funkcja zawiera trzy argumenty — pierwszym jest zakres, drugim wartość, a trzecim wyrażenie:

```
=SUMA(A1:A5, 12, 24*3)
```

Możesz również tworzyć funkcje o nieokreślonej liczbie argumentów. Cała sztuczka polega na tym, aby jako ostatniego (lub jedynego) argumentu użyć tablicy, poprzedzonej słowem kluczowym ParamArray.



Słowo kluczowe ParamArray może być użyte tylko dla ostatniego argumentu listy argumentów procedury. Argument ten zawsze ma typ Variant i zawsze jest opcjonalny (choć nie stosuje się słowa kluczowego Optional).

Poniższa funkcja może mieć dowolną liczbę jednowartościowych argumentów, nie obsługuje argumentów będących zakresami i po prostu zwraca sumę wartości argumentów.

```
Function SIMPLESUM(ParamArray arglist() As Variant) As Double
    For Each arg In arglist
        SIMPLESUM = SIMPLESUM + arg
    Next arg
End Function
```

Aby zmodyfikować naszą przykładową funkcję tak, aby mogła działać z argumentami będącymi zakresami komórek, musimy dodać kolejną pętlę, która będzie przetwarzala poszczególne komórki w kolejnych argumentach (zakresach):

```
Function SIMPLESUM(ParamArray arglist() As Variant) As Double
    Dim cell As Range
    For Each arg In arglist
        For Each cell In arg
            SIMPLESUM = SIMPLESUM + cell
        Next cell
    Next arg
End Function
```

Funkcja SIMPLESUM jest co prawda nieco podobna do funkcji arkuszowej SUMA, ale nie jest nawet w przybliżeniu tak elastyczna. Wypróbuj ją przy użyciu różnych typów argumentów i przekonasz się na przykład, że nie działa poprawnie, jeżeli dowolna z komórek będących jej argumentem zawiera wartość inną niż numeryczną lub jej argumentem zamiast adresu będzie literał.

Emulacja funkcji arkuszowej SUMA

W tym podrozdziale zaprezentujemy niestandardową funkcję o nazwie MYSUM. W przeciwieństwie do funkcji SIMPLESUM zamieszczonej w poprzednim podrozdziale funkcja MYSUM (niemal) doskonale emuluje funkcję arkuszową SUMA Excela.

Zanim przyjrzysz się kodowi źródłowemu funkcji MySum, przez chwilę zastanów się nad funkcją SUMA Excela. Jest bardzo uniwersalna. Może posiadać maksymalnie 255 argumentów (włączając w to nawet „puste” argumenty), którymi mogą być wartości liczbowe, komórki, zakresy, tekstowe reprezentacje liczb, wartości logiczne, a nawet inne funkcje. Na początek przyjrzyj się następującej, przykładowej formule:

```
=SUMA(B1; 5; "6"; ; PRAWDA; PIERWIASTEK(4); A1:A5; D:D; C2*C3)
```

Powyższa, perfekcyjnie poprawna formula zawiera wszystkie wymienione poniżej typy argumentów, przedstawione zgodnie z kolejnością ich użycia:

- Odwołanie do pojedynczej komórki.
- Literał.
- Łańcuch dający się przekształcić na wartość liczbową.
- Brakujący (pusty) argument.
- Wartość logiczna PRAWDA.
- Wyrażenie używające innej funkcji.
- Proste odwołanie do zakresu.
- Odwołanie do zakresu obejmującego całą kolumnę.
- Wyrażenie obliczające iloczyn wartości dwóch komórek.

Funkcja MYSUM, której kod zamieszczamy na listingu 8.1 poniżej, obsługuje wszystkie wymienione typy argumentów.

Listing 8.1. Funkcja MySum

```
Function MYSUM(ParamArray args() As Variant) As Variant
    ' Emuluje funkcję arkuszową SUMA

    ' Deklaracje zmiennych
    Dim i As Variant
    Dim TempRange As Range, cell As Range
    Dim ECode As String
    Dim m, n
    MYSUM = 0

    ' Przetwarzanie poszczególnych argumentów
    For i = 0 To UBound(args)
        ' Pomiń brakujące argumenty
        If Not IsMissing(args(i)) Then
            ' Określanie typu argumentów
            Select Case TypeName(args(i))
                Case "Range"
                    ' Utwórz roboczy zakres obsługujący zakresy typu cały wiersz lub cała kolumna
                    Set TempRange = Intersect(args(i).Parent.UsedRange, args(i))
                    For Each cell In TempRange
                        If IsError(cell) Then
                            MYSUM = cell ' Zwróć błąd
                            Exit Function
                        End If
                        If cell = True Or cell = False Then
                            MYSUM = MYSUM + cell.Value
                        Else
                            MYSUM = MYSUM + cell
                        End If
                    Next cell
                Case "String"
                    MYSUM = MYSUM + args(i)
                Case "Boolean"
                    MYSUM = MYSUM + IIf(args(i), 1, 0)
                Case "Variant"
                    MYSUM = MYSUM + args(i)
                Case "Double"
                    MYSUM = MYSUM + args(i)
                Case "Integer"
                    MYSUM = MYSUM + args(i)
                Case "Long"
                    MYSUM = MYSUM + args(i)
                Case "Currency"
                    MYSUM = MYSUM + args(i)
                Case "Date"
                    MYSUM = MYSUM + args(i)
                Case "Error"
                    MYSUM = args(i)
                Case "Object"
                    MYSUM = args(i)
                Case "Nothing"
                    MYSUM = Nothing
                Case Else
                    MYSUM = MYSUM + args(i)
            End Select
        End If
    Next i
End Function
```

```

MYSUM = MYSUM + 0
Else
    If IsNumeric(cell) Or IsDate(cell) Then =
        MYSUM = MYSUM + cell
    End If
Next cell
Case "Variant()"
    n = args(i)
    For m = LBound(n) To UBound(n)
        MYSUM = MYSUM(MYSUM, n(m)) 'wywołanie rekurencyjne
    Next m
Case "Null" 'zignoruj
Case "Error" 'zwroc blad
    MYSUM = args(i)
    Exit Function
Case "Boolean"
    Sprawdz, czy argument to literal PRAWDA, i uwzglednij go w obliczeniach
    If args(i) = "PRAWDA" Then MYSUM = MYSUM + 1
Case "Date"
    MYSUM = MYSUM + args(i)
Case Else
    MYSUM = MYSUM + args(i)
End Select
End If
Next i
End Function

```



Skoroszyt z tym przykładem (*Funkcja MySum.xls*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 8.7 przedstawiono skoroszyt zawierający różne formuły wykorzystujące funkcje SUMA i MySum. Jak sam możesz się przekonać, obie funkcje zwracają identyczne wyniki obliczeń.

Funkcja MYSUM to bardzo dokładna, ale nie do końca perfekcyjna emulacja wbudowanej funkcji arkuszowej SUMA — nasza funkcja nie potrafi działać na tablicach. Na przykład formuła przedstawiona poniżej zwraca sumę kwadratów wartości komórek z zakresu A1:A4.

`{=SUM(A:A4^2)}`

Nasza formuła w takiej sytuacji powoduje wygenerowanie błędu #WARTOŚĆ!:

`{=MYSUM(A1:A4^2)}`

Aby dowiedzieć się, jak działa funkcja MYSUM, powinieneś utworzyć formułę testową wykorzystującą tę funkcję. Następnie ustaw pułapkę w wybranym miejscu kodu źródłowego i wykonuj kolejno polecenie po poleceniu (patrz podrozdział „Wykrywanie i usuwanie błędów w funkcjach” w dalszej części tego rozdziału). Wypróbuj kilka różnych typów argumentów i po pewnym czasie będziesz miał całkiem obszerną wiedzę na temat sposobu działania tej funkcji.

	A	B	C	D	E	F	G	H	I
1	Skoroszyt ilustruje działanie niestandardowej funkcji MYSUM,								
2	która emuluje zachowanie wbudowanej funkcji arkuszowej SUMA.								
3									
4									
5	1	4	2		SUMA	7	MYSUM	7	
6	PRAWDA	FAŁSZ	PRAWDA			0		0	
7	Pierwszy	"2"	3			3		3	
8	1	#N/D!	3		#N/D!		#N/D!		
9	Jeden	Dwa	Trzy			0		0	
10	17:00:00	4 styczeń	5 styczeń		2126-01-11 17:00:00		2126-01-11 17:00:00		
11	1					100		100	
12						1		1	
13						1		1	
14					11,48912529		11,48912529		
15						31		31	
16					#DZIEL/0!		#DZIEL/0!		
17					#ARG!		#ARG!		
18					3,708333333		3,708333333		
19						24		24	
20									
21									
22									

Rysunek 8.7. Porównanie wyników działania funkcji SUMA i MySum

Analizując kod źródłowy funkcji MYSUM, powinieneś pamiętać o kilku istotnych sprawach:

- Brakujące argumenty (rozpoznawane przez funkcję IsMissing) są po prostu ignorowane.
- W celu określenia typu argumentu (Range, Error itp.) procedura stosuje funkcję TypeName języka VBA. Każdy typ argumentu jest obsługiwany w inny sposób.
- W przypadku argumentu typu Range funkcja przy użyciu pętli przetwarza każdą komórkę zakresu, określa typ danych w nich zawartych i w razie konieczności dodaje je do całkowitej obliczanej sumy.
- Funkcja zwraca wartość typu Variant, ponieważ musi mieć możliwość zwracania błędu, jeżeli dowolny jej argument będzie wartością błędu.
- Jeżeli argument zawiera błąd (na przykład #DZIEL/0!), funkcja MYSUM, podobnie jak funkcja SUMA Excela, po prostu zwraca błąd.
- Jeżeli argument funkcji SUMA nie jest literałem (faktyczna wartość zamiast zmiennej), łańcuch tekstowy jest traktowany przez nią jak wartość zero. Funkcja MySum dodaje wartość danej komórki tylko wtedy, gdy zidentyfikuje ją jako liczbę. Do rozpoznawania używana jest funkcja IsNumeric języka VBA.
- W przypadku argumentów będących zakresami, funkcja używa metody Intersect, która tworzy tymczasowy zakres będący częścią wspólną dwóch zakresów arkusza. Taka metoda obsługuje przypadki, kiedy argument typu Range zawiera pełny wiersz lub kolumnę, których przetworzenie trwałoby wyjątkowo długo.

Możesz być ciekawy, jak wygląda porównanie względnej szybkości działania funkcji arkuszowej SUMA i naszej funkcji MYSUM. Oczywiście z góry wiadomo, że funkcja MYSUM jest znacznie wolniejsza od funkcji SUMA, ale jej szybkość działania zależy też w dużej mierze

od wydajności komputera i konstrukcji samych formuł. W przypadku mojego komputera arkusz z 5000 formuł używających funkcji SUMA został przeliczony natychmiast. Po zastąpieniu funkcji SUMA funkcją MYSUM obliczenia zajęły mniej więcej osiem sekund. Funkcja MYSUM może zostać jeszcze trochę udoskonalona, ale pod względem szybkości nigdy nie zbliży się do funkcji SUMA.

Pamiętaj, że celem tego przykładu *nie* jest tworzenie nowej wersji funkcji SUMA. Przykład demonstruje sposób tworzenia niestandardowej funkcji arkusza, która wygląda i działa jak wbudowane funkcje Excela.

Rozszerzone funkcje daty

Bardzo często użytkownicy Excela narzekają, że nie mogą swobodnie pracować z datami wcześniejszymi niż 1 stycznia 1900 roku. Na przykład, tworząc drzewo genealogiczne, możemy wpisywać w Excelu daty narodzin i śmierci naszych przodków. Jeżeli któraś z takich dat będzie wcześniejsza niż rok 1900, proste obliczenie liczby przeżytych przez daną osobę lat nie będzie możliwe.

Na szczęście VBA potrafi przetwarzać znacznie szerszy zakres dat, dzięki czemu udało mi się utworzyć cały szereg funkcji przeznaczonych do pracy z datami. Najwcześniejszą datą rozpoznawaną przez VBA jest 1 stycznia 100 roku.



Pracując z datami, powinieneś zwracać uwagę na zmiany kalendarza, jakie miały miejsce na przestrzeni wieków (przed rokiem 1752). Różnice pomiędzy kalendarzami amerykańskimi, brytyjskimi, gregoriańskim i juliańskim mogą spowodować, że obliczenia dat nie będą dokładne.

Lista dostępnych funkcji jest następująca:

- **XDATE(*y;m;d;fmt*)** — zwraca datę dla danego roku, miesiąca i dnia. Opcjonalnie możesz podać również żądany format daty.
- **XDATEADD(*xdate1;days;fmt*)** — dodaje podaną liczbę dni do określonej daty. Opcjonalnie możesz podać również żądany format daty.
- **XDATEDIF(*xdate1;xdate2*)** — zwraca liczbę dni pomiędzy podanymi datami.
- **XDATEYEARIF(*xdate1;xdate2*)** — zwraca liczbę pełnych lat pomiędzy podanymi datami (przydatne do obliczania wieku osób, przedmiotów itp.).
- **XDATEYEAR(*xdate1*)** — zwraca rok z podanej daty.
- **XDATEMONTH(*xdate1*)** — zwraca miesiąc z podanej daty.
- **XDATEDAY(*xdate1*)** — zwraca dzień z podanej daty.
- **ZDATEDOW(*xdate1*)** — zwraca dzień tygodnia z podanej daty (jako liczbę od 1 do 7).

Na rysunku 8.8 przedstawiono wygląd skoroszytu wykorzystującego niektóre spośród wymienionych funkcji.

	E11	:	X	✓	f(x)	=xdate(B11;C11;D11;"mmmm d, yyyy")			
	A	B	C	D	E	F	G	H	I
5	Prezydent	Rok	Miesiąc	Dzień	XDATE	XDATEDIF	XDATEYEARIF	XDATEDOW	
7	George Washington	1732	2	22	lutym 22, 1732	102 641	281	Piątek	
8	John Adams	1735	10	30	październik 30, 1735	101 295	277	Niedziela	
9	Thomas Jefferson	1743	4	13	kwiecień 13, 1743	98 573	269	Sobota	
10	James Madison	1751	3	16	marzec 16, 1751	95 679	261	Wtorek	
11	James Monroe	1758	4	28	kwiecień 28, 1758	93 079	254	Piątek	
12	John Quincy Adams	1767	7	11	lipiec 11, 1767	89 718	245	Sobota	
13	Andrew Jackson	1767	3	15	marzec 15, 1767	89 836	245	Niedziela	
14	Martin Van Buren	1782	12	5	grudzień 5, 1782	84 092	230	Czwartek	
15	William Henry Harrison	1773	2	9	lutym 9, 1773	87 678	240	Wtorek	
16	John Tyler	1790	3	29	marzec 29, 1790	81 421	222	Poniedziałek	
17	James K. Polk	1795	11	2	listopad 2, 1795	79 377	217	Poniedziałek	
18	Zachary Taylor	1784	11	24	listopad 24, 1784	83 372	228	Środa	
19	Millard Fillmore	1800	1	7	styczeń 7, 1800	77 850	213	Wtorek	
20	Franklin Pierce	1804	11	23	listopad 23, 1804	76 069	208	Piątek	
21	James Buchanan	1791	4	23	kwiecień 23, 1791	81 031	221	Sobota	
22	Abraham Lincoln	1809	2	12	lutym 12, 1809	74 527	204	Niedziela	
23	Andrew Johnson	1808	12	29	grudzień 29, 1808	74 572	204	Czwartek	
24	Ulysses S. Grant	1822	4	27	kwiecień 27, 1822	69 705	190	Sobota	

Rysunek 8.8. Zastosowanie rozszerzonych funkcji daty w formułach arkuszowych

Pamiętaj, że daty zwracane przez powyższe funkcje mają postać *łańcuchów tekstu*, a nie prawdziwych dat. Z tego powodu nie możesz wykonywać tradycyjnych obliczeń matematycznych na wartościach zwracanych przez funkcje, ale za to możesz takich wyników używać jako argumentów dla innych rozszerzonych funkcji daty.

Kod funkcji jest zaskakująco prosty. Na przykład poniżej zamieszczamy pełny kod funkcji XDATE:

```
Function XDATE(y, m, d, Optional fmt As String) As String
    If IsMissing(fmt) Then fmt = "Short Date"
    XDATE = Format(DateSerial(y, m, d), fmt)
End Function
```

Argumenty funkcji XDATE są następujące:

- y — cztery cyfry reprezentujące rok z zakresu od 0100 do 9999;
- m — numer miesiąca (1 – 12);
- d — numer dnia (1 – 31);
- fmt — (argument opcjonalny) — łańcuch tekstu reprezentujący format daty.

Jeżeli argument fmt zostanie pominięty, data jest wyświetlana zgodnie z ustawieniami *krótkiego formatu daty systemowej* (możesz je zmienić za pośrednictwem ustawień regionalnych w *Panelu sterowania*).

Jeżeli argumenty m lub d mają wartość przekraczającą dopuszczalny, standardowy zakres, to są automatycznie przeliczane na kolejne lata lub miesiące. Na przykład, jeżeli jako wartość miesiąca podasz liczbę 13, to zostanie ona zinterpretowana jako miesiąc styczeń kolejnego roku.



Skoroszyt zawierający rozszerzone funkcje daty (*Rozszerzone funkcje daty.xlsx*) został zamieszczony na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Znajdziesz tam również dokument PDF zawierający krótką dokumentację tych funkcji (*Rozszerzone funkcje daty.PDF*).

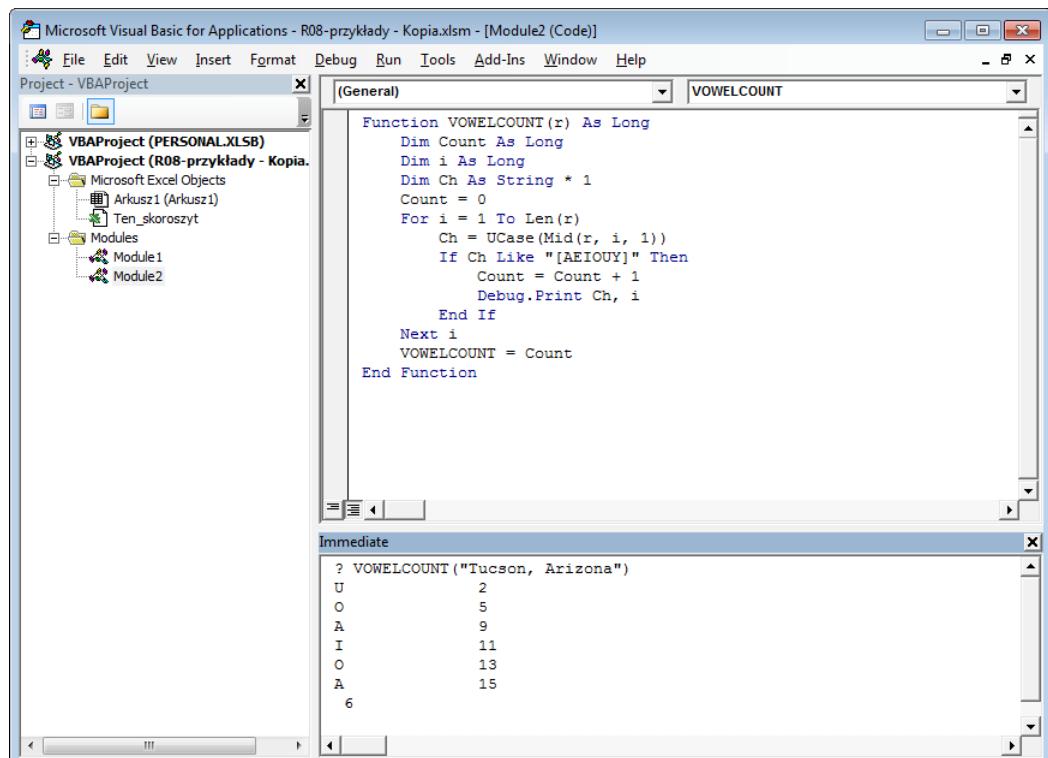
Wykrywanie i usuwanie błędów w funkcjach

Gdy w celu sprawdzenia procedury Function umieszczasz w arkuszu formułę, ewentualne błędy uruchomieniowe nie pojawiają się w dobrze Ci znanych oknach komunikatów. Zamiast tego po wystąpieniu błędu formuła po prostu zwraca błąd #WARTOŚĆ!. Na szcze- ście nie stanowi to zbyt wielkiego problemu podczas wykrywania i usuwania błędów, po- nieważ zawsze masz do dyspozycji kilka sztuczek:

- **Umieszczenie — w celu monitorowania wartości określonych zmiennych — funkcji MsgBox w strategicznych lokalizacjach.** Polecenie MsgBox umieszczone w kodzie funkcji powoduje, że po jej wywołaniu na ekranie zostaje wyświetcone odpowiednie okno dialogowe z komunikatem. Korzystając z tej metody powinieneś się upewnić, że w arkuszu znajduje się tylko jedna formuła używająca testowanej funkcji. W przeciwnym razie okna komunikatów będą się pojawiały dla każdej przetwarzanej formuły, co szybko może się okazać bardzo irytujące.
- **Testowanie procedury poprzez wywołanie jej z procedury Sub, a nie z formuły arkusza.** Błędy wykonania są wyświetlane w standardowy sposób, dzięki czemu możesz szybko usunąć problem (jeżeli wiesz, jak to zrobić) lub skorzystać z debugera.
- **Ustawienie w kodzie funkcji pułapki i wykonanie jej w trybie krokowym.** Po wykonaniu tej operacji dostępne będą wszystkie standardowe narzędzia wykrywające i usuwające błędy. Aby ustawić pułapkę, należy umieścić cursor w wierszu instrukcji, przy której ma zostać zatrzymane wykonywanie funkcji, a następnie z menu Debug wybrać Toggle Breakpoint lub wcisnąć klawisz F9. W trakcie działania funkcji możesz nacisnąć klawisz F8, który pozwala na krokowe wykonywanie kolejnych poleceń.
- **Użycie w kodzie źródłowym jednej lub kilku tymczasowych instrukcji Debug.Print wyświetlających wartości w oknie Immediate edytora Visual Basic.** Aby na przykład monitorować wartości zmiennych wewnętrz pętli, możesz użyć następującego rozwiązania:

```
Function VOWELCOUNT(r) As Long
    Dim Count As Long
    Dim i As Long
    Dim Ch As String * 1
    Count = 0
    For i = 1 To Len(r)
        Ch = UCase(Mid(r, i, 1))
        If Ch Like "[AEIOUY]" Then
            Count = Count +1
            Debug.Print Ch, i
        End If
    Next i
    VOWELCOUNT = Count
End Function
```

W tym przypadku w oknie *Immediate* są wyświetlane wartości zmiennej Ch i i każdorazowo po napotkaniu instrukcji Debug.Print. Na rysunku 8.9 pokazano wynik działania funkcji, gdy jej argumentem był łańcuch znaków Tucson Arizona.

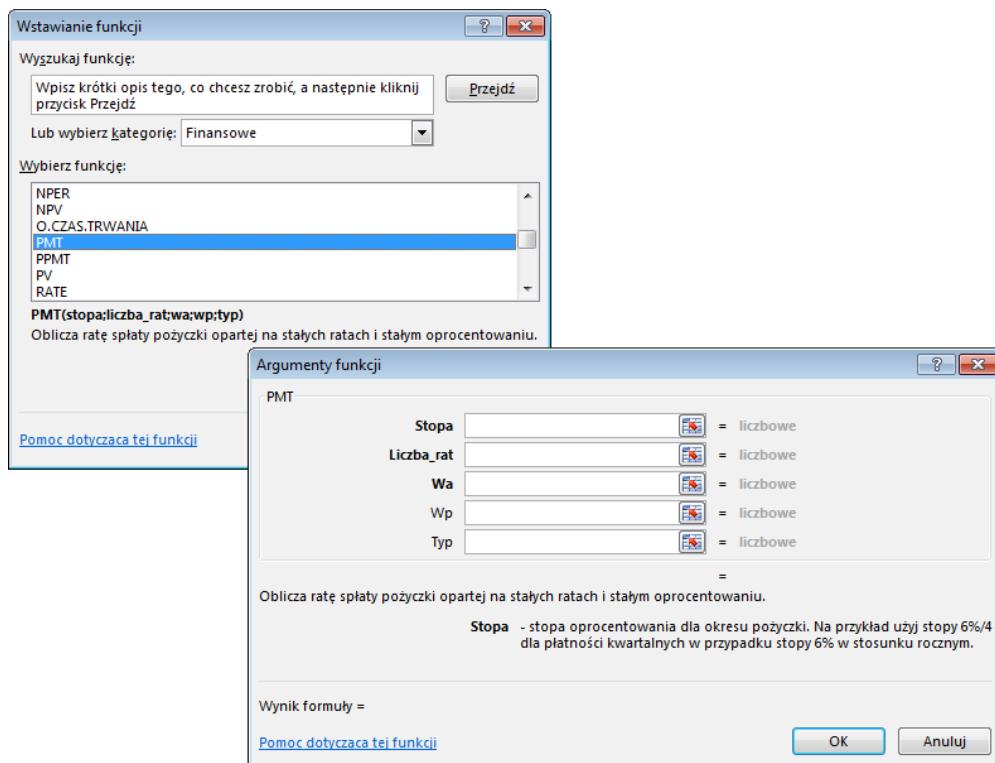


Rysunek 8.9. Okno *Immediate* umożliwia wyświetlanie wyników w trakcie wykonywania funkcji

Okno dialogowe Wstawianie funkcji

Okno dialogowe *Wstawianie funkcji* Excela jest bardzo wygodnym narzędziem. W trakcie tworzenia formuły arkuszowej umożliwia wybranie z listy żądanej funkcji arkuszowej. Lista funkcji została pogrupowana według kategorii, co zdecydowanie ułatwia odnalezienie określonej funkcji. Kiedy wybierzesz funkcję i naciśniesz przycisk *OK*, na ekranie pojawi się okno dialogowe *Argumenty funkcji*, która ułatwia wprowadzenie odpowiednich argumentów wywołania funkcji. Na rysunku 8.10 przedstawiono wygląd obu wspomnianych okien.

W oknie dialogowym *Wstawianie funkcji* znajdziesz również niestandardowe funkcje arkuszowe. Domyślnie funkcje niestandardowe są wyświetlane w kategorii *Zdefiniowane przez użytkownika*. Okno dialogowe *Argumenty funkcji* pozwala również na wprowadzanie argumentów dla takich funkcji.



Rysunek 8.10. Okno Wstawianie funkcji pozwala na wybranie żąanej funkcji z listy, a okno Argumenty funkcji ułatwia wprowadzanie argumentów

Okno dialogowe *Wstawianie funkcji* pozwala na wyszukiwanie funkcji według słów kluczowych. Niestety mechanizm ten nie może być wykorzystywany do wyszukiwania funkcji VBA zdefiniowanych przez użytkownika.



Niestandardowe funkcje Function definiowane przy użyciu słowa kluczowego Private nie pojawiają się w oknie dialogowym *Wstawianie funkcji*. Jeżeli tworzona funkcja ma być dostępna wyłącznie dla innych procedur języka VBA, powinieneś przy jej deklarowaniu użyć słowa kluczowego Private. Pamiętaj jednak, że zadeklarowanie funkcji jako Private nie zapobiega możliwości jej użycia w formule, a tylko uniemożliwia jej wyświetlenie w oknie dialogowym *Wstawianie funkcji*.

Zastosowanie metody MacroOptions

Jeżeli chcesz, aby Twoja funkcja wyglądała tak, jak wbudowane funkcje arkuszowe, możesz skorzystać z metody MacroOptions obiektu Applications. Metoda ta pozwala na:

- Zdefiniowanie opisu funkcji.
- Przydzielenie funkcji do wybranej kategorii.
- Zdefiniowanie opisów argumentów funkcji.

Poniżej zamieszczamy kod procedury, która używa metody MacroOptions do odpowiedniego zdefiniowania informacji na temat funkcji.

```
Sub DescribeFunction()
    Dim FuncName As String
    Dim FuncDesc As String
    Dim FuncCat As Long
    Dim Arg1Desc As String, Arg2Desc As String

    FuncName = "DRAWONE"
    FuncDesc = "Wyświetla zawartość losowej komórki z podanego zakresu"
    FuncCat = 5
    Arg1Desc = "Zakres komórek zawierających wartości"
    Arg2Desc = "(Opcjonalny) Jeżeli ma wartość FAŁSZ lub jest pominięty, nowa komórka"
    Arg2Desc = Arg2Desc & " nie zostanie zaznaczona podczas przeliczania arkusza."
    Arg2Desc = Arg2Desc & "Jeżeli argument ma wartość PRAWDA, nowa komórka"
    Arg2Desc = Arg2Desc & "zostanie zaznaczona podczas przeliczania arkusza."

    Application.MacroOptions _
        Macro:=FuncName, _
        Description:=FuncDesc, _
        Category:=FuncCat, _
        ArgumentDescriptions:=Array(Arg1Desc, Arg2Desc)
End Sub
```

Procedura zaprezentowana powyżej używa zmiennych do przechowywania informacji, które następnie zostają użyte jako argumenty wywołania metody MacroOptions. Funkcja zostanie przypisana do kategorii 5 (czyli inaczej Wyszukiwania i adresu). Zauważ, że opisy argumentów funkcji są przekazywane w postaci tablicy, będącej ostatnim argumentem metody MacroOptions.



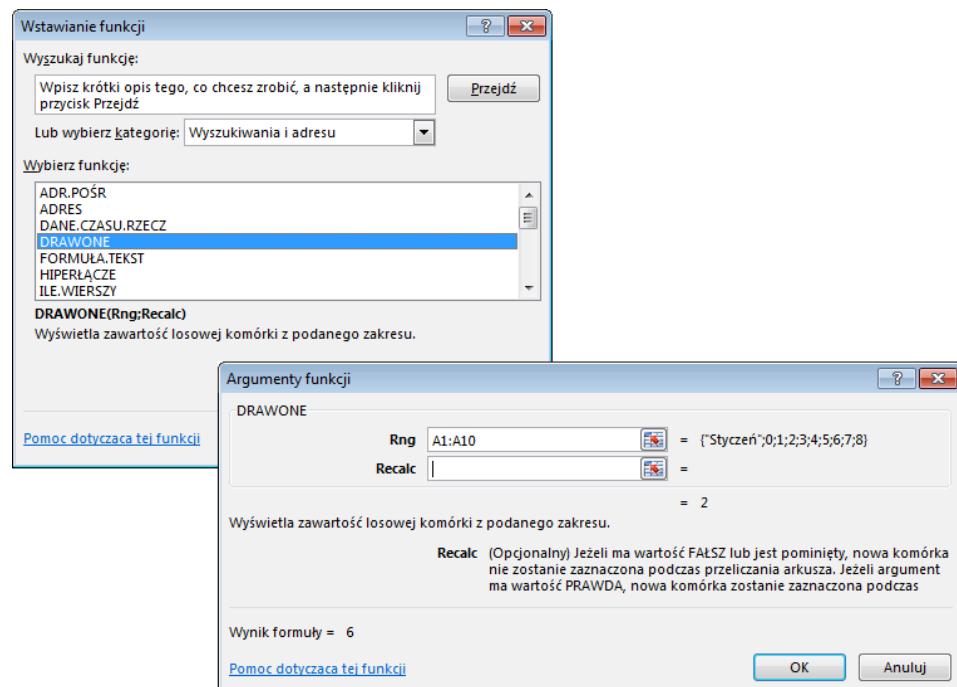
Możliwość definiowania opisów argumentów funkcji to zupełnie nowość, wprowadzona po raz pierwszy w Excelu 2010. Pamiętaj jednak, że jeżeli skoroszyt zawierający funkcje z opisami zostanie otwarty w starszej wersji Excela, to opisy argumentów nie będą widoczne.

Na rysunku 8.11 przedstawiono wygląd okien *Wstawianie funkcji* oraz *Argumenty funkcji* po uruchomieniu powyższej procedury.

Procedurę DescribeFunction musisz wywołać tylko raz. Po zakończeniu działania informacje przypisane do funkcji będą na stałe zapisane w skoroszycie. W razie potrzeby możesz również pominąć wybrane argumenty. Na przykład, jeżeli nie potrzebujesz szczegółowego opisu poszczególnych argumentów wywołania funkcji, możesz całkowicie pominąć argument ArgumentDescription.



Więcej szczegółowych informacji na temat tworzenia systemu pomocy dla niestandardowych funkcji, dostępnego z poziomu okna dialogowego *Wstawianie funkcji*, znajdziesz w rozdziale 22.



Rysunek 8.11. Okna Wstawianie funkcji oraz Argumenty funkcji dla naszej funkcji niestandardowej

Definiowanie kategorii funkcji

Jeżeli nie używasz metody MacroOptions do zdefiniowania kategorii funkcji, Twoje funkcje niestandardowe będą pojawiały się w oknie *Wstawianie funkcji* w kategorii *Zdefiniowane przez użytkownika*. W razie potrzeby możesz oczywiście przypisać taką funkcję do innej kategorii. Przypisanie funkcji do wybranej kategorii powoduje również, że taka funkcja pojawi się na liście funkcji dostępnej po naciśnięciu przycisku danej kategorii na karcie *FORMUŁY*, w grupie opcji *Biblioteka funkcji*.

W tabeli 8.1 zamieszczono listę numerów kategorii funkcji, których możesz użyć jako argumentu dla metody MacroOptions. Warto zauważyć, że niektóre z tych kategorii (od 10 do 13) nie są domyślnie wyświetlane w oknie dialogowym *Wstawianie funkcji*. Każda z tych kategorii pojawia się na liście dopiero po przypisaniu do niej pierwszej funkcji.



W razie potrzeby możesz tworzyć własne kategorie funkcji. Aby to zrobić, powinieneś zamiast odpowiedniej wartości liczbowej, reprezentującej wybraną kategorię funkcji, użyć łańcucha tekstu jako argumentu wywołania metody MacroOptions. Przykładowe polecenie przedstawione poniżej tworzy nową kategorię o nazwie *Funkcje VBA* i przypisuje do niej funkcję *COMMISSION*:

```
Application.MacroOptions Macro:="COMMISSION", Category:="Funkcje VBA"
```

Tabela 8.1. Kategorie funkcji

Numer kategorii	Nazwa kategorii
0	Wszystkie (żadna konkretna kategoria)
1	<i>Finansowe</i>
2	<i>Daty i czasu</i>
3	<i>Matematyczne</i>
4	<i>Statystyczne</i>
5	<i>Wyszukiwania i adresu</i>
6	<i>Bazy danych</i>
7	<i>Tekstowe</i>
8	<i>Logiczne</i>
9	<i>Informacyjne</i>
10	<i>Polecenia</i>
11	<i>Dostosowywanie</i>
12	<i>Sterowanie makrami</i>
13	<i>DDE/Zewnętrzne</i>
14	<i>Użytkownika</i>
15	<i>Inżynierskie</i>
16	<i>Moduł</i>
17	<i>Zgodność</i> ¹
18	<i>Sieć Web</i> ²

Dodawanie opisu funkcji

Oprócz metody MacroOptions do utworzenia opisu funkcji możesz użyć okna dialogowego Makra.



Jeżeli nie utworzysz opisu niestandardowej funkcji, w oknie dialogowym *Wstawianie funkcji* pojawi się dla tej funkcji komunikat *Pomoc niedostępna*.

Aby zdefiniować opis niestandardowej funkcji, wykonaj następujące polecenia:

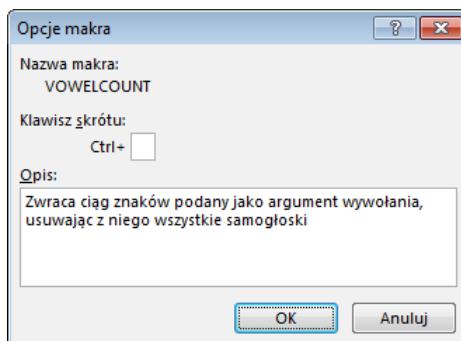
1. Utwórz funkcję w edytorze Visual Basic.
2. Przejdź do okna programu Excel i sprawdź, czy skoroszyt zawierający funkcję jest aktywny.

¹ Kategoria *Zgodność* to nowość, która pojawiła się w Excelu 2010.

² Kategoria *Sieć Web* to nowość, która pojawiła się w Excelu 2013.

3. Przejdź na kartę *DEVELOPER* i naciśnij przycisk *Makra*, znajdujący się w grupie opcji *Kod* (zamiast tego możesz nacisnąć kombinację klawiszy *Alt+F8*).
4. W oknie dialogowym *Makro* zostanie wyświetlona lista dostępnych procedur, ale Twojej funkcji na niej nie będzie.
5. W polu *Nazwa makra* wprowadź nazwę funkcji.
6. Naciśnij przycisk *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje makra*.
7. W polu *Opis* wprowadź opis funkcji (patrz rysunek 8.12).

Rysunek 8.12.
Tworzenie opisu funkcji
w oknie dialogowym
Opcje makra



8. Pole *Klawisz skrótu* nie ma zastosowania w przypadku funkcji.
9. Naciśnij przycisk *OK*, a następnie *Anuluj*.

Po wykonaniu poleceń opisanych powyżej i po wybraniu danej funkcji w oknie dialogowym *Wstawianie funkcji* będzie wyświetlany opis wprowadzony w kroku 6.

Zastosowanie dodatków do przechowywania funkcji niestandardowych

Często używane funkcje niestandardowe można przechowywać w pliku dodatku. Podstawową zaletą takiego rozwiązania jest dostępność takich funkcji w dowolnym skoroszycie.

Co więcej, możesz używać takich funkcji w formułach bez konieczności stosowania kwalifikatora nazwy pliku. Założmy, że dysponujesz niestandardową funkcją o nazwie *ZAPSPACES*, przechowywaną w pliku *Myfuncs.xlsm*. Aby użyć tej funkcji w formule skorszytu innego niż *Myfuncs.xlsm* konieczne będzie zastosowanie następującej formuły:

=Myfuncs.xlsm!ZAPSPACES(A1:C12)

Jeżeli w oparciu o plik *Myfuncs.xlsm* utworzysz dodatek i go załadujesz, możesz pominąć odwołanie do pliku i wprowadzić następującą formułę:

=ZAPSPACES(A1:C12)



Więcej szczegółowych informacji na temat dodatków znajdziesz w rozdziale 19.



Jednym z potencjalnych problemów związanych z przechowywaniem niestandardowych funkcji w dodatkach jest to, że tworzone skoroszyty są zależne od obecności pliku dodatku. Jeżeli chcesz taki skoroszyt przekazać innym użytkownikom, musisz pamiętać również o przesłaniu odpowiedniego pliku dodatków zawierającego funkcje niestandardowe.

Korzystanie z Windows API

Język VBA potrafi używać metod zdefiniowanych w innych plikach, które nie mają nic wspólnego z VBA ani Exceliem. Dobrym przykładem są biblioteki DLL (ang. *Dynamic Link Library*) używane przez system Windows i inne aplikacje. Dzięki temu przy użyciu języka VBA można wykonywać operacje, które w przeciwnym razie byłyby niemożliwe.

Interfejs API systemu Windows (ang. *Windows Application Programming Interface*) jest zbiorem funkcji dostępnych dla programistów aplikacji Windows. Przez wywołanie w kodzie źródłowym języka VBA funkcji systemu Windows uzyskuje się dostęp do jego interfejsu API. Wiele zasobów systemu Windows używanych przez programistów jest dostępnych z poziomu bibliotek DLL przechowujących procedury i funkcje oraz konsolidowanych na etapie uruchamiania aplikacji, a nie w czasie komplikacji.

64-bitowa wersja Excela a funkcje API

Począwszy od Excela 2010, korzystanie z funkcji Windows API stało się zupełnie nowym wyzwaniem, ponieważ zarówno Excel 2010, jak i 2013 są dostępne również w wersji 64-bitowej. Jeżeli chcesz, aby Twój kod był kompatybilny z wersjami 32-bitowymi oraz 64-bitowymi wersjami Excela, musisz wykorzystywać podwójne deklaracje funkcji API, używając do tego odpowiednich dyrektyw kompilatora, tak aby upewnić się, że użyta zostanie poprawna deklaracja.

Na przykład przedstawiona poniżej deklaracja funkcji działa poprawnie z 32-bitową wersją Excela, ale próba jej skompilowania na wersji 64-bitowej Excela 2010 czy 2013 zakończy się wyświetleniem komunikatu o błędzie:

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

W wielu przypadkach utworzenie deklaracji kompatybilnej z 64-bitową wersją Excela jest banalnie proste i polega w zasadzie tylko na dodaniu po słowie kluczowym `Declare klauzuli PtrSafe`. Deklaracja przedstawiona poniżej jest kompatybilna zarówno z 32-, jak i 64-bitową wersją Excela 2010 i 2013:

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Z drugiej jednak strony próba skompilowania takiego kodu w Excelu 2007 (i wersjach wcześniejszych) zakończy się niepowodzeniem, ponieważ klauzula `PtrSafe` będzie nieznana.

W rozdziale 24. wyjaśnimy, jak spowodować, aby deklaracje funkcji API były kompatybilne ze wszystkimi 32-bitowymi wersjami Excela oraz z 64-bitowymi wersjami Excela 2010 i 2013.

Przykłady zastosowania funkcji interfejsu API systemu Windows

Przed użyciem funkcji interfejsu API systemu Windows konieczne jest jej zadeklarowanie na początku modułu kodu źródłowego. Jeżeli moduł kodu źródłowego jest standaryзовym modelem VBA, takim jak *UserForm*, *Arkusz1* czy *ThisWorkbook*, będziesz musiał zadeklarować taką funkcję interfejsu API przy użyciu słowa kluczowego *Private*.

Deklaracja funkcji interfejsu API musi być precyzyjna i informować interpretera języka VBA o:

- funkcji interfejsu API, która zostanie użyta;
- bibliotecę, w której znajduje się funkcja interfejsu API;
- argumentach funkcji interfejsu API.

Po zadeklarowaniu funkcji interfejsu API można jej użyć w kodzie źródłowym języka VBA.

Identyfikacja katalogu domowego systemu Windows

Poniżej zamieszczono przykład deklaracji funkcji interfejsu API, która wyświetla nazwę katalogu domowego systemu Windows — czyli jest to coś, czego normalnie nie można osiągnąć przy użyciu języka VBA. Przedstawiony kod będzie działał poprawnie w Excelu 2010 i w wersjach późniejszych.

Deklaracja funkcji API wygląda następująco:

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Nasza funkcja posiada dwa argumenty i zwraca ścieżkę katalogu, w którym zainstalowano system Windows. Po wywołaniu funkcji ścieżka katalogu systemu Windows jest przypisywana zmiennej *lpBuffer*, natomiast długość jej łańcucha — zmiennej *nSize*.

Po umieszczeniu na początku modułu odpowiedniej deklaracji, możesz w kodzie programu wywołać funkcję *GetWindowsDirectoryA*. Poniżej zamieszczamy przykład wywołania funkcji przez procedurę wyświetlającą wynik w oknie komunikatu:

```
Sub ShowWindowsDir()  
    Dim WinPath As String * 255  
    Dim WinDir As String  
    WinPath = Space(255)  
    WinDir = Left(WinPath, GetWindowsDirectoryA(WinPath, Len(WinPath)))  
    MsgBox WinDir, vbInformation, "Ścieżka katalogu domowego systemu Windows"  
End Sub
```

Wykonanie procedury *ShowWindowsDir* spowoduje wyświetlenie okna komunikatu zawierającego ścieżkę katalogu systemu Windows.

Bardzo często dla funkcji interfejsu API tworzy się *funkcje osłonowe* (ang. *wrapper functions*). Innymi słowy, tworzysz własną funkcję, która wykorzystuje funkcje interfejsu API. Dzięki temu zastosowanie funkcji interfejsu API jest znacznie prostsze. Oto przykład funkcji osłonowej napisanej w języku VBA:

```

Function WINDOWSDIR() As String
    ' Zwraca ścieżkę katalogu domowego systemu Windows
    Dim WinPath As String * 255
    WinPath = Space(255)
    WINDOWSDIR = Left(WinPath, GetWindowsDirectoryA(WinPath, Len(WinPath)))
End Function

```

Po zadeklarowaniu funkcji można ją wywołać z innej procedury. Oto przykład:

```
MsgBox WINDOWSDIR()
```

Można nawet umieścić funkcję w formule arkusza:

```
=WINDOWSDIR()
```



Skoroszyt z tym przykładem (*Katalog Windows.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Deklaracje funkcji API są kompatybilne z Exceliem 2007 i wersjami późniejszymi.

Powodem, dla którego zwykle używamy funkcji interfejsu API, jest to, że w przeciwnym razie zrealizowanie operacji wykonywanych przez takie funkcje nie byłoby możliwe lub przynajmniej byłoby bardzo utrudnione. Jeżeli Twoja aplikacja musi odnaleźć ścieżkę katalogu systemu Windows, to niezależnie od tego, jak długo będziesz szukał, i tak nie znajdziesz odpowiedniej funkcji ani w Excelu, ani w języku VBA. Jednak wiedząc, w jaki sposób użyć funkcji interfejsu API systemu Windows, dysponujesz eleganckim rozwiązaniem takiego problemu.



Podczas korzystania z wywołań funkcji interfejsu API i ich testowania zawieszenia systemu wcale nie należą do rzadkości, dlatego pamiętaj, aby często zapisywać tworzony projekt.

Wykrywanie wcisnięcia klawisza Shift

Oto kolejny przykład zastosowania wywołań funkcji API. Założymy, że stworzyłeś makro języka VBA uruchamiane przy użyciu przycisku paska narzędzi. Dodatkowo przyjmijmy, że makro ma zadziałać inaczej, gdy użytkownik przy klikaniu przycisku wcisnie klawisz *Shift*. VBA nie posiada żadnych mechanizmów, pozwalających na wykrycie, czy wcisnięto klawisz *Shift*. Jednak w takiej sytuacji umożliwia to funkcja interfejsu API o nazwie *GetKeyState*. Funkcja pobiera jeden argument *nVirtKey* reprezentujący kod klawisza, którym jesteś zainteresowany.

Poniższy kod źródłowy demonstruje metodę wykrywania, czy w momencie uruchamiania procedury obsługi zdarzenia *Button_Click* wcisnięto klawisz *Shift*. Dla klawisza *Shift* zdefiniowałem stałą (przy użyciu wartości szesnastkowej), a następnie posłużyłem się nią jako argumentem funkcji *GetKeyState*. Jeżeli funkcja *GetKeyState* zwróci wartość mniejszą od zera, będzie to oznaczało, że wcisnięto klawisz *Shift*. W przeciwnym razie oznacza to, że klawisz *Shift* nie został wcisnięty. Kod programu nie jest kompatybilny z Exceliem 2007.

```

Declare Function GetKeyState Lib "user32" -
    (ByVal nVirtKey As Long) As Integer
Sub Button_Click()
    Const VK_SHIFT As Integer = &H10

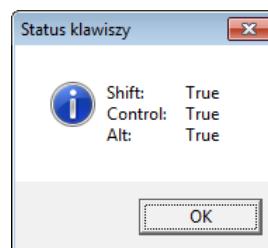
```

```
If GetKeyState(VK_SHIFT) < 0 Then  
    MsgBox "Klawisz Shift został wcisnięty."  
Else  
    MsgBox "Klawisz Shift nie został wcisnięty."  
End If  
End Sub
```



Skoroszyt *Klawisze.xls* zamieszczony na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) demonstruje sposób wykrywania wcisnięć takich klawiszy jak *Ctrl*, *Shift* i *Alt*, a także ich dowolnych kombinacji. Deklaracje funkcji API zapisanych w tym skoroszycie są kompatybilne z Exceliem 2007 i nowszymi. Na rysunku 8.13 przedstawiono okno dialogowe będące wynikiem działania tej procedury.

Rysunek 8.13.
*Przykład zastosowania
funkcji API
do wykrywania
wcisniętych klawiszy*



Dodatkowe informacje na temat funkcji interfejsu API

Korzystanie z funkcji interfejsu API może okazać się trudne. Wiele książek poświęconych programowaniu zawiera listę deklaracji powszechnie używanych funkcji interfejsu API i często też przykłady. Zwykle wystarczy przekopiować deklaracje i użyć funkcji bez zagłębiania się w szczegóły. W rzeczywistości (przynajmniej w tej, z którą się osobiście zetknąłem) większość programistów tworzących aplikacje w Excelu posługuje się funkcjami interfejsu API jak gotowymi przepisami. W sieci Internet dostępnych jest kilkaset przykładów, które można pobrać, wkleić do tworzonego kodu i będą one działały poprawnie. W razie potrzeby powinieneś poszukać w sieci pliku o nazwie *Win32API_PtrSafe.txt*. Jest to plik udostępniony przez firmę Microsoft, zawierający wiele przykładów deklaracji funkcji API, które możesz wykorzystać we własnych programach.



W rozdziale 9. znajdziesz kilka dodatkowych przykładów zastosowania funkcji interfejsu API systemu Windows.

Rozdział 9.

Przykłady

i techniki programowania

w języku VBA

W tym rozdziale:

- Zastosowanie VBA do pracy z zakresami
- Zastosowanie VBA do pracy ze skoroszytami i arkuszami
- Tworzenie własnych funkcji i używanie ich w formułach arkusza i procedurach VBA
- Przykłady technik programowania w języku VBA
- Przykłady zastosowania funkcji interfejsu API

Nauka poprzez praktykę

Wierzę, że nauka programowania odbywa się znacznie szybciej, kiedy pracujemy na konkretnych przykładach, a Czytelnicy poprzednich wydań książki zdecydowanie utwierdzą mnie w tym przekonaniu. Takie podejście sprawdza się zwłaszcza dla programistów pracujących z językiem VBA. Dobrze opracowany przykład o wiele lepiej objaśnia zagadnienie niż teoretyczny opis. W związku z tym zrezygnowałem z materiału referencyjnego, w którym dokładnie opisywano wszystkie, nawet najdrobniejsze aspekty języka VBA, a zamiast tego przygotowałem przykłady demonstrujące użytkowe, praktyczne techniki programowania przy użyciu Excela.

Poprzednie rozdziały tej części książki odpowiednio przygotowały Czytelników do poznawania zagadnień omawianych w tym rozdziale, natomiast w systemie pomocy programu Excel znajdziesz wszystkie informacje, które tutaj zostały pominione. W tym rozdziale zwiększy się nieco tempo i zaprezentowanych zostanie sporo przykładów rozwiązujących problemy spotykane w praktyce i pozwalających pogłębić wiedzę na temat języka VBA.

Jak korzystać z przykładów zamieszczonych w tym rozdziale?

Nie wszystkie przykłady zamieszczone w tym rozdziale mogą spełniać rolę samodzielnych programów, ale zawsze mają postać wykonywalnych procedur, które możesz dostosować do własnych potrzeb i użyć w swoich aplikacjach.

W trakcie lektury powinieneś na bieżąco pracować z komputerem i samodzielnie testować opisywane w tym rozdziale przykłady (i nie tylko). Jeszcze lepiej będzie, jeżeli będziesz próbował samodzielnie modyfikować przykłady i sprawdzać, jaki będzie efekt tych modyfikacji. Mogę Ci zagwarantować, że takie praktyczne doświadczenia będą o wiele bardziej pomocne niż przeczytanie od deski do deski książki zawierającej tylko teoretyczną stronę programowania w języku VBA.

Przykłady omawiane w tym rozdziale zostały podzielone na sześć kategorii:

- Praca z zakresami
- Praca ze skoroszytami i arkuszami
- Techniki programowania w języku VBA
- Użyteczne funkcje, których warto używać w procedurach VBA
- Użyteczne funkcje, których możesz używać w formułach arkuszowych
- Wywołania funkcji i procedur Windows API



W kolejnych rozdziałach naszej książki znajdziesz szereg przykładów procedur dotyczących również innych zagadnień, takich jak wykresy, tabele przestawne, zdarzenia, formularze *UserForm* i inne.

Przetwarzanie zakresów

Przykłady zamieszczone w tym podrozdziale demonstrują, w jaki sposób za pomocą języka VBA można manipulować zakresami arkusza.

W szczególności znajdziesz tutaj przykłady procedur, które pozwalają na kopowanie i przenoszenie zakresów komórek, zaznaczanie zakresów komórek, identyfikację typów danych przechowywanych w danym zakresie komórek, wprowadzanie wartości do komórek przez użytkownika, wyszukiwanie pierwszej pustej komórki w kolumnie, zatrzymywanie makra w celu umożliwienia użytkownikowi zaznaczenia zakresu, zliczanie komórek w zakresie, przechodzenie w pętli i przetwarzanie kolejnych komórek zakresu oraz kilka innych operacji, często wykonywanych na zakresach komórek arkusza.

Kopiowanie zakresów

Rejestrator makr Excela jest bardzo przydatny nie tyle do generowania wydajnego, użytecznego kodu źródłowego, co do „odkrywania” nazw odpowiednich obiektów, metod i właściwości. Kod źródłowy generowany przez rejestrator makr nie zawsze jest optymalny i efektywny, ale zwykle pozwala uzyskać sporo przydatnych informacji.

Przykładowo po zarejestrowaniu prostej operacji kopiowania i wklejania, generowanych jest pięć wierszy kodu źródłowego języka VBA:

```
Sub Makrol()
    Range("A1").Select
    Selection.Copy
    Range("B1").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
End Sub
```

Wygenerowany kod najpierw powoduje zaznaczenie i skopiowanie komórki A1, a następnie, po zaznaczeniu komórki B1, procedura wykonuje operację wklejania. Jednak w języku VBA nie jest konieczne zaznaczanie obiektu, który będzie przetwarzany. O tej istotnej sprawie nie dowiedziałbyś się nigdy, gdybyś wzorował się tylko na kodzie źródłowym zarejestrowanego makra, w którym w dwóch instrukcjach została użyta metoda Select. Zamiast tego, możesz posłużyć się znacznie prostszą procedurą, która nie zaznacza żadnych komórek i korzysta z tego, że metoda Copy może użyć argumentu reprezentującego miejsce docelowe kopowanego zakresu.

```
Sub CopyRange()
    Range("A1").Copy Range("B1")
End Sub
```

W obu powyższych makrach przyjęto założenie, że arkusz, w którym wykonywana jest operacja, jest aktywny. Aby skopiować zakres do innego arkusza lub skoroszytu, wystarczy odpowiednio zdefiniować odwołanie do zakresu docelowego. W poniższym przykładzie zakres jest kopowany z arkusza Arkusz1 skoroszytu *Plik1.xlsxm* do arkusza Arkusz2 skoroszytu *Plik2.xlsxm*. Ponieważ odwołania są w pełni kwalifikowane, procedura zadziała niezależnie od tego, który skoroszyt będzie aktywny.

```
Sub CopyRange2()
    Workbooks("Plik1.xlsxm").Sheets("Arkusz1").Range("A1").Copy _
        Workbooks("Plik2.xlsxm").Sheets("Arkusz2").Range("A1")
End Sub
```

Kolejna metoda wykonania tej operacji polega na zastosowaniu zmiennych obiektowych reprezentujących zakresy, tak jak to zostało zilustrowane w przykładzie poniżej. Zastosowanie zmiennych obiektowych jest bardzo użyteczne zwłaszcza w sytuacji, kiedy w kodzie programu wykorzystujesz odwołania do zakresów komórek:

```
Sub CopyRange3()
    Dim Rng1 As Range, Rng2 As Range
    Set Rng1 = Workbooks("Plik1.xlsxm").Sheets("Arkusz1").Range("A1")
    Set Rng2 = Workbooks("Plik2.xlsxm").Sheets("Arkusz2").Range("A1")
    Rng1.Copy Rng2
End Sub
```

Jak się zapewne domyślasz, kopianie nie jest ograniczone tylko do jednej komórki na raz. Przykładowo, procedura przedstawiona poniżej kopiuje duży zakres komórek. Zwróć uwagę na fakt, że miejsce docelowe jest tutaj identyfikowane tylko przez jedną komórkę — górną lewą komórkę wklejanego zakresu. Użycie jednej komórki działa dokładnie tak, jak podczas ręcznego kopowania i wklejania komórek arkusza.

```
Sub CopyRange4()
    Range("A1:C800").Copy Range("D1")
End Sub
```

Przenoszenie zakresów

Instrukcje języka VBA służące do przenoszenia zakresu są bardzo podobne do instrukcji używanych podczas kopiowania zakresów, tak jak to zostało zaprezentowane na poniższym przykładzie. Różnica polega na tym, że zamiast metody `Copy` użyta została metoda `Cut`. Pamiętaj, że musisz podać tylko lokalizację górnej, lewej komórki zakresu docelowego.

W przykładzie przedstawionym poniżej 18 komórek (z zakresu A1:C6) przenosimy w nowy obszar, rozpoczynający się od adresu H1.

```
Sub MoveRange1()
    Range("A1:C6").Cut Range("H1")
End Sub
```

Kopiowanie zakresu o zmiennej wielkości

W wielu przypadkach konieczne jest skopiowanie zakresu komórek, dla którego dokładna liczba wierszy i kolumn określających jego wielkość nie jest z góry znana. Przykładowo możesz dysponować skoroszytem śledzącym tygodniową sprzedaż, w którym liczba wierszy zmienia się każdego tygodnia po wprowadzeniu nowych danych.

Na rysunku 9.1 pokazano bardzo często spotykany typ arkusza. Zawarty w nim zakres składa się z kilku wierszy, których liczba zmienia się każdego tygodnia. Ponieważ nie wiesz, jaki jest adres zakresu w danej chwili, podczas pisania makra kopiującego zakres będziesz uwzględnić nieco dodatkowego kodu źródłowego.

Rysunek 9.1.
Liczba wierszy zakresu danych zmienia się każdego tygodnia

	A	B	C	D	E	F	G
1	Tydzień	Sprzedaż	Liczba nowych klientów				
2	1	21 094	45				
3	2	20 309	64				
4	3	19 374	80				
5	4	21 050	72				
6	5	22 227	92				
7	6	24 970	99				
8	7	27 259	111				
9							
10							

Poniższe makro ilustruje sposób kopiowania zakresu komórek z arkusza Arkusz1 do arkusza Arkusz2 (począwszy od komórki A1). Makro wykorzystuje właściwość `CurrentRegion`, która zwraca obiekt Range odpowiadający blokowi komórek otaczających określoną komórkę (w tym przypadku o adresie A1).

```
Sub CopyCurrentRegion2()
    Range("A1").CurrentRegion.Copy Sheets("Arkusz2").Range("A1")
End Sub
```

Wskazówki dotyczące przetwarzania zakresów

W trakcie przetwarzania zakresów powinieneś pamiętać o kilku ważnych kwestiach.

- W języku VBA, do przetwarzania zakresu nie jest konieczne jego uprzednie zaznaczenie.
- Nie możesz zaznaczyć zakresu, który znajduje się na nieaktywnym arkuszu, stąd jeżeli Twoja procedura zaznacza zakres, to powiązany z nim arkusz musi być aktywny. W celu uaktywnienia określonego arkusza można użyć metody Activate kolekcji Worksheets.
- Pamiętaj, że rejestrator makr nie generuje zbyt wydajnego kodu źródłowego. Najlepiej utworzyć makro przy użyciu rejestratora, a następnie jego kod źródłowy zmodyfikować w celu zwiększenia efektywności.
- W kodzie źródłowym języka VBA warto stosować nazwane zakresy. Przykładowo odwołanie Range("Total") jest znacznie bardziej czytelne niż odwołanie Range("D45"). W tym drugim przypadku dodanie wiersza powyżej wiersza 45. spowoduje zmianę adresu komórki i w konsekwencji konieczne będzie zmodyfikowanie makra tak, aby używało zakresu o poprawnym adresie (D46).
- Jeżeli w trakcie zaznaczania zakresów korzystasz z rejestratora makr, upewnij się, że makro rejestrowane jest przy użyciu odwołań względnych. Aby to zrobić, przejdź na kartę DEVELOPER i naciśnij przycisk *Użyj odwołań względnych*, znajdujący się w grupie opcji *Kod*.
- Po uruchomieniu makra przetwarzającego kolejne komórki aktualnie zaznaczonego zakresu, użytkownik może zaznaczać całe wiersze lub kolumny. W większości przypadków nie ma potrzeby przetwarzania wszystkich komórek zaznaczonego zakresu. Tworzone makro powinno definiować podzbior zaznaczenia zawierający wyłącznie niepuste komórki.Więcej szczegółowych informacji na ten temat znajdziesz w podrozdziale „Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli” w dalszej części rozdziału.
- Excel pozwala zaznaczać wiele obszarów jednocześnie. Na przykład możesz zaznaczyć pierwszy zakres, wcisnąć klawisz *Ctrl* i zaznaczyć kolejny zakres. Makro powinno dokonać sprawdzenia zakresu i podjąć odpowiednią decyzję. Zapoznaj się z zawartością punktu „Określanie typu zaznaczonego zakresu” w dalszej części rozdziału.



Zastosowanie właściwości *CurrentRegion* jest równoważne przejściu na kartę *NARZĘDZIA GŁÓWNE* i wybraniu polecenia *Znajdź i zaznacz/Przejdź do — specjalnie*, znajdującego się w grupie opcji *Edycja*, i następnie zaznaczeniu opcji *Bieżący obszar* (zamiast tego możesz również nacisnąć kombinację klawiszy *Ctrl+Shift+**).

Aby przekonać się, jak to działa, podczas wykonywania tych poleceń powinieneś zarejestrować makro. Zazwyczaj wartość właściwości *CurrentRegion* reprezentuje prostokątny blok komórek otoczony przez puste wiersze i kolumny.

Jeżeli zakres komórek, który chcesz skopiować, jest tabelą (zdefiniowaną przy użyciu polecenia *Tabela*, znajdującego się na karcie *NARZĘDZIA GŁÓWNE*, w grupie poleceń *Tabele*), możesz użyć kodu przedstawionego poniżej (który zakłada, że tabela ma nazwę *Table1*).

```
Sub CopyTable  
    Range("Table1[#A11]").Copy Sheets("Sheet 2").Range("A1")  
End Sub
```

Zaznaczanie oraz identyfikacja różnego typu zakresów

Większość operacji wykonywanych przez instrukcje języka VBA opiera się na zakresach — poprzez definiowanie zakresów lub identyfikowanie zakresów w celu wykonywania operacji na komórkach do nich należących.

Oprócz właściwości CurrentRegion (o której mówiliśmy już wcześniej) powinieneś również poznać metodę End obiektu Range. Metoda ta pobiera jeden argument określający kierunek, w którym zostanie wykonane zaznaczenie. Poniższe polecenie zaznacza zakres rozpoczętyjący się od aktywnej komórki i kończący na ostatniej niepustej dolnej komórce:

```
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Poniżej zamieszczamy kolejny przykład, w którym została zdefiniowana komórka będąca początkiem zakresu:

```
Range(Range("A2"), Range("A2").End(xlDown)).Select
```

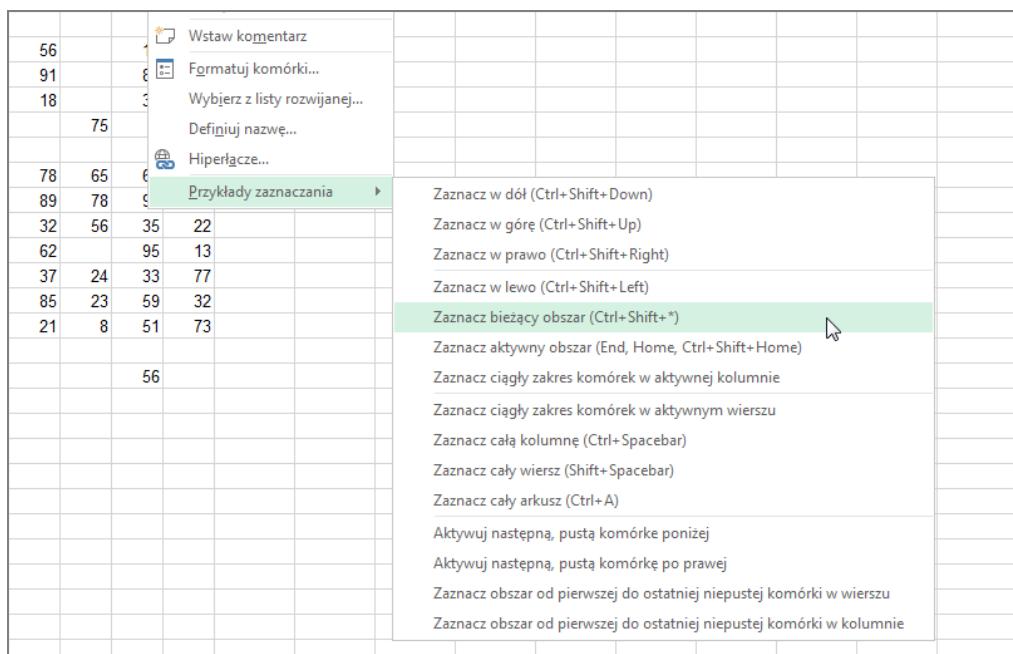
Jak można się domyślić, trzy pozostałe stałe (xlUp, xlToLeft, xlToRight) symulują kombinacje klawiszy zaznaczające komórki w innych kierunkach.



Korzystając z właściwości ActiveCell w powiązaniu z metodą End, powinieneś zachować szczególną ostrożność. Jeżeli aktywna komórka znajduje się na końcu zakresu lub jeżeli w skład zakresu wchodzi jedna lub więcej pustych komórek, wyniki działania metody End mogą być zupełnie inne od oczekiwanych.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt (*Zaznaczanie zakresów.xlsm*) ilustrujący najczęściej spotykane rodzaje zaznaczeń zakresów. Po jego otwarciu w menu podręcznym pojawi się nowe podmenu o nazwie *Przykłady zaznaczeń*. Poszczególne polecenia menu umożliwiają użytkownikowi zapoznanie się z przykładami różnych rodzajów zaznaczeń (patrz rysunek 9.2).



Rysunek 9.2. Niestandardowe menu podręczne w tym skoroszycie ilustruje zaznaczanie zakresów o różnej wielkości

Poniższe makro, o nazwie `SelectCurrentRegion` znajduje się we wspomnianym, przykładowym skoroszycie i symuluje naciśnięcie kombinacji klawiszy `Ctrl+Shift+*`:

```
Sub SelectCurrentRegion()
    ActiveCell.CurrentRegion.Select
End Sub
```

Bardzo często zaznaczanie komórek po prostu nie będzie potrzebne, a zamiast tego będziesz wykonywał na komórkach inne operacje (na przykład formatowanie). Procedurę zaznaczającą komórki można łatwo przystosować do tego celu. Procedura przedstawiona poniżej jest prostą modyfikacją makra `SelectCurrentRegion`, która nie zaznacza komórek, a jedynie formatuje zakres zdefiniowany jako bieżący obszar otaczający aktywną komórkę. Inne procedury znajdujące się w przykładowym skoroszycie też mogą zostać przy stosowane w ten sposób.

```
Sub FormatCurrentRegion()
    ActiveCell.CurrentRegion.Font.Bold = True
End Sub
```

Zmiana rozmiaru zakresu komórek

Właściwość `Resize` obiektu `Range` pozwala na łatwą zmianę rozmiaru zakresu komórek. Aby skorzystać z tej właściwości, musisz podać dwa argumenty, reprezentujące odpowiednio całkowitą liczbę wierszy i kolumn w nowym zakresie.

Inne sposoby odwoływania się do zakresów komórek

Jeżeli przyjrzysz się kodowi programów napisanych w języku VBA przez innych użytkowników, z pewnością zwróciś uwagę na różne metody odwoływania się do zakresów komórek. Na przykład polecenie przedstawione poniżej zaznacza określony zakres komórek:

```
[C2:D8].Select
```

Adres zakresu komórek znajduje się w nawiasach kwadratowych i nie jest ujęty w znaki cudzysłów. Polecenie przedstawione powyżej jest funkcjonalnym odpowiednikiem polecenia zapisanego w następujący sposób:

```
Range("C2:D8").Select
```

Nawiasy kwadratowe użyte w pierwszym poleceniu są skróconą formą wywołania metody `Evaluate` obiektu `Application`. W naszym przypadku jest to skrócona forma następującego polecenia:

```
Application.Evaluate("C2:D8").Select
```

Szczerze mówiąc, nigdy nie używałem odwołań z nawiasami kwadratowymi w swoich programach, ponieważ uważam, że taka forma zapisu może być nieco myląca. Dla mnie jedną zaletą takiej składni jest możliwość zaoszczędzenia kilku uderzeń w klawisze podczas wprowadzania kodu. Warto tutaj również zauważyc, że zgodnie z przeprowadzonymi przeze mnie testami odwoływanie do zakresów komórek wykorzystujące nawiasy kwadratowe są wykonywane około 70% wolniej niż normalne, pełne odwołania. Dzieje się tak dlatego, że interpreter VBA potrzebuje dodatkowego czasu na przeanalizowanie łańcucha tekstu zawierającego odwołanie z nawiasami kwadratowymi i odszukanie właściwego zakresu komórek.

Na przykład po wykonaniu sekwencji poleceń przedstawionych poniżej zmienna obiektowa MyRange reprezentuje zakres komórek o rozmiarach 20 wierszy na 5 kolumn (zakres A1:E20):

```
Set MyRange = Range("A1")
Set MyRange = MyRange.Resize(20, 5)
```

Po wykonaniu polecenia zamieszczonego poniżej rozmiar zakresu komórek, reprezentowanego przez zmienną MyRange, zostaje zwiększyony o jeden wiersz. Zwróć uwagę, że drugi argument został pominięty, ponieważ liczba kolumn nie ulega zmianie.

```
Set MyRange = MyRange.Resize(MyRange.Rows.Count + 1)
```

Bardziej praktyczny przykład przedstawia sytuację, w której zmieniamy definicję nazwanego zakresu komórek. Założmy, że w bieżącym skoroszycie masz zdefiniowany zakres komórek o nazwie Dane. Twoim zadaniem (a raczej zadaniem kodu, który musisz utworzyć) jest rozszerzenie tego zakresu o jeden dodatkowy wiersz. Aby to zrobić, możesz użyć na przykład następującego kodu:

```
With Range("Dane")
    .Resize(.Rows.Count + 1).Name = "Dane"
End With
```

Wprowadzanie wartości do komórki

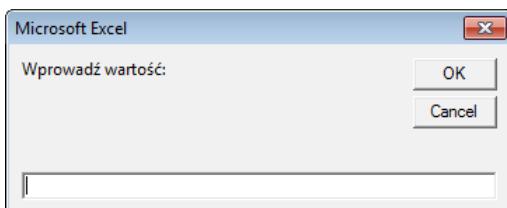
Kolejna procedura przedstawiona poniżej demonstruje, jak poprosić użytkownika o podanie wartości i wstawić ją do komórki A1 aktywnego arkusza:

```
Sub GetValue1()
    Range("A1").Value = InputBox("Wprowadź wartość:")
End Sub
```

Na rysunku 9.3 pokazano wygląd okna umożliwiającego wprowadzenie wartości.

Rysunek 9.3.

Funkcja *InputBox*
pobiera wartość,
która zostanie
umieszczona
w komórce



Przedstawiona procedura może sprawiać jednak pewien problem. Jeżeli użytkownik naciśnie w oknie dialogowym przycisk *Cancel*, procedura usunie wszelkie dane już znajdujące się w komórce. Poniższa zmodyfikowana wersja procedury sprawdza, czy został naciśnięty przycisk *Cancel* i jeżeli tak, nie dokonuje zmiany zawartości komórki:

```
Sub GetValue2()
    Dim UserEntry As String
    UserEntry = InputBox("Wprowadź wartość:")
    If UserEntry <> "" Then Range("A1").Value = UserEntry
End Sub
```

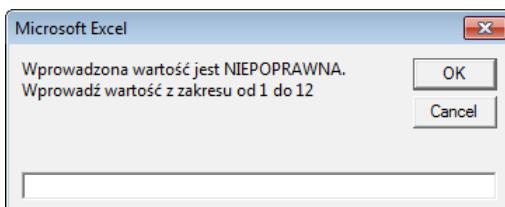
W wielu przypadkach sprawdzenie poprawności danych wprowadzonych przez użytkownika będzie konieczne. Na przykład chcesz, aby użytkownik wprowadził liczbę z zakresu od 1 do 12. W poniższym przykładzie zademonstrowano jedną z metod sprawdzania poprawności danych. Niepoprawna wartość jest ignorowana, a okno wyświetlane ponownie. Operacja jest powtarzana do momentu wprowadzenia prawidłowej wartości lub naciśnięcia przycisku *Cancel*.

```
Sub GetValue3()
    Dim UserEntry As Variant
    Dim Msg As String
    Const MinVal As Integer = 1
    Const MaxVal As Integer = 12
    Msg = "Wprowadź wartość z zakresu od " & MinVal & " do " & MaxVal
    Do
        UserEntry = InputBox(Msg)
        If UserEntry = "" Then Exit Sub
        If IsNumeric(UserEntry) Then
            If UserEntry >= MinVal And UserEntry <= MaxVal Then Exit Do
        End If
        Msg = "Wprowadzona wartość jest NIEPOPRAWNA."
        Msg = Msg & vbCrLf
        Msg = Msg & "Wprowadź wartość z zakresu od " & MinVal & " do " & MaxVal
    Loop
    ActiveSheet.Range("A1").Value = UserEntry
End Sub
```

Jeżeli użytkownik wprowadzi niepoprawną wartość, program odpowiednio zmieni treść wyświetlonego komunikatu (patrz rysunek 9.4).

Rysunek 9.4.

Sprawdzenie poprawności danych wprowadzonych przez użytkownika przy użyciu funkcji *InputBox* języka VBA



Skoroszyt zawierający wszystkie trzy przykłady (*Funkcja InputBox.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wprowadzanie wartości do następnej pustej komórki

Często wykonywaną operacją jest wprowadzenie wartości do następnej pustej komórki kolumny lub wiersza. Procedura przedstawiona poniżej prosi użytkownika o podanie imienia i wartości, a następnie wprowadza dane do następnego pustego wiersza (patrz rysunek 9.5).

```
Sub GetData()
    Dim NextRow As Long
    Dim Entry1 As String, Entry2 As String
    Do
```

Rysunek 9.5.

Makro wstawiające dane do następnego pustego wiersza arkusza

The screenshot shows a Microsoft Excel spreadsheet with two columns: 'Imię' (Name) and 'Wartość' (Value). The data starts from row 1 and continues to row 13. A new row is being inserted at row 14. An input dialog box titled 'Microsoft Excel' is displayed, asking 'Podaj imię' (Enter name) with 'OK' and 'Cancel' buttons. The status bar at the bottom indicates 'Arkusz1'.

A	B	C	D	E	F	G	H	I	J
1 Imię	Wartość								
2 Adam	983								
3 Bronek	409								
4 Czesław	773								
5 Dariusz	0								
6 Ela	412								
7 Franek	551								
8 Grzegorz	895								
9 Jacek	545								
10 Krzysiek	988								
11 Patrycja	545								
12 Piotr	344								
13 Tomasz	232								
14									
15									
16									

Odszukaj następny pusty wiersz

```
NextRow = Cells(Rows.Count, 1).End(xlUp).Row + 1
```

Poproś o wprowadzenie danych

```
Entry1 = InputBox("Podaj imię:")
```

```
If Entry1 = "" Then Exit Sub
```

```
Entry2 = InputBox("Podaj wartość:")
```

```
If Entry2 = "" Then Exit Sub
```

Zapisz dane w arkuszu

```
Cells(NextRow, 1) = Entry1
```

```
Cells(NextRow, 2) = Entry2
```

Loop

End Sub

Dla uproszczenia nasza procedura w żaden sposób nie sprawdza poprawności wprowadzanych danych. Zwróć uwagę, że w procedurze nie został określony warunek zakończenia pętli. W celu jej opuszczenia użyta została instrukcja `Exit Sub`, która jest wykonywana po naciśnięciu przycisku *Cancel* w oknie wprowadzania danych.



Skoroszyt zawierający oba przykłady (*NastępnaPustaKomórka.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zwróć uwagę na instrukcję określającą wartość zmiennej `NextRow`. Jeżeli nie rozumiesz, w jaki sposób procedura działa, spróbuj ręcznie wykonać realizowaną przez nią operację: uaktywnij ostatnią komórkę w kolumnie A (co w przypadku Excela 2010 oznacza komórkę o adresie A1048576), naciśnij klawisz *End* i następnie naciśnij klawisz ↑ (strzałka w góre). W efekcie zostanie zaznaczona ostatnia niepusta komórka kolumny A. Właściwość `Row` zwraca numer wiersza tej komórki. W celu uzyskania numeru kolejnego pustego wiersza (poniżej) wartość ta jest zwiększana o jeden. Zamiast umieszczać na „sztywno” adres ostatniej komórki wiersza A, użyta została metoda `Rows.Count`, dzięki czemu nasza procedura będzie poprawnie działała również z poprzednimi wersjami programu Excel (w których maksymalna liczba wierszy w arkuszu jest dużo mniejsza).

Z taką metodą zaznaczania następnej pustej komórki związany jest drobny problem. Jeżeli kolumna jest zupełnie pusta, jako następny pusty wiersz metoda wyznaczy wiersz 2. Na szczęście dodanie odpowiedniego kodu, który będzie zapobiegał takiemu zachowaniu, nie jest trudnym zadaniem.

Wstrzymywanie działania makra w celu umożliwienia pobrania zakresu wyznaczonego przez użytkownika

Zdarzają się sytuacje, w których makro musi być w pewien sposób interaktywne. Na przykład możesz utworzyć makro, które wstrzymuje działanie, pozwalając użytkownikowi na zaznaczenie wybranego zakresu komórek. Procedura opisana w tym punkcie demonstruje sposób wykonania zadania przy użyciu metody InputBox Excela.



Nie należy mylić metody InputBox Excela z funkcją języka VBA o takiej samej nazwie. Co prawda obie funkcje mają taką samą nazwę, ale nie są takie same.

Poniższa procedura Sub demonstruje sposób zatrzymania pracy makra i umożliwienia użytkownikowi zaznaczenia zakresu komórek. Po wznowieniu działania procedura wstawia odpowiednią formułę do wszystkich komórek zaznaczonego zakresu.

```
Sub GetUserRange()
    Dim UserRange As Range

    Prompt = "Zaznacz wybrany zakres komórek."
    Title = "Wybieranie zakresu komórek"

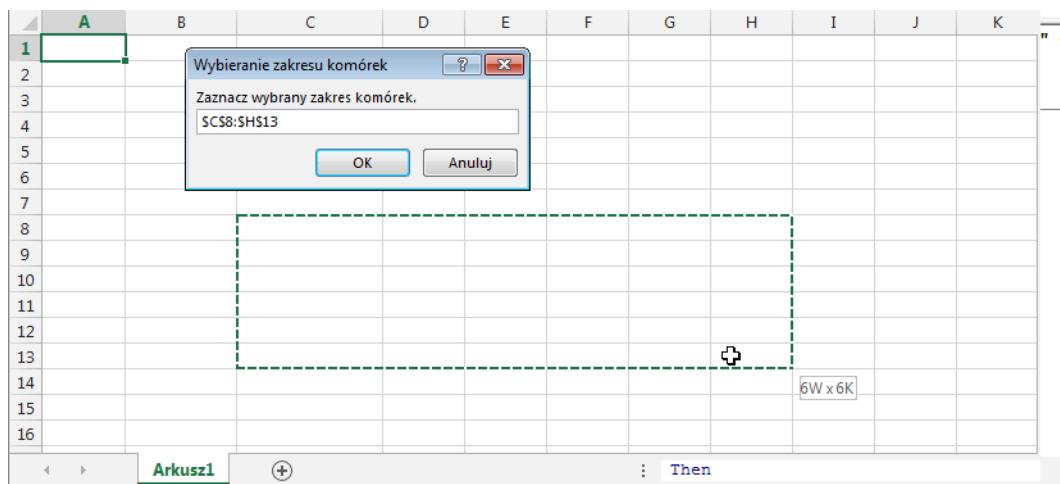
    ' Wyświetlanie okna dialogowego
    On Error Resume Next
    Set UserRange = Application.InputBox( _
        Prompt:=Prompt, _
        Title:=Title, _
        Default:=ActiveCell.Address, _
        Type:=8) 'Zaznaczanie zakresu komórek
    On Error GoTo 0

    ' Czy okno dialogowe zostało anulowane?
    If UserRange Is Nothing Then
        MsgBox "Operacja anulowana."
    Else
        UserRange.Formula = "=RAND()"
    End If
End Sub
```

Okno dialogowe zostało przedstawione na rysunku 9.6.



Skoroszyt z tym przykładem (*Zaznacz zakres.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Rysunek 9.6. Okno dialogowe użyte do wstrzymania działania makra

Podanie argumentu Type o wartości 8 ma kluczowe znaczenie dla powyższej procedury. Oprócz tego powinieneś również zwrócić uwagę na zastosowanie instrukcji On Error Resume Next. To polecenie powoduje, że błąd, który wystąpi gdy użytkownik naciśnie przycisk *Anuluj*, będzie ignorowany. Jeżeli tak się stanie, nie zostanie zdefiniowana zmienna obiektowa UserRange. W powyższym przykładzie jest wyświetlane okno zawierające komunikat o treści *Operacja anulowana*. Gdy użytkownik naciśnie przycisk *OK*, wykonywanie makra będzie kontynuowane. Instrukcja On Error GoTo 0 przywraca standardową obsługę błędów.

Nawiasem mówiąc, sprawdzanie poprawności zaznaczonego zakresu nie jest konieczne, ponieważ zajmie się tym Excel.

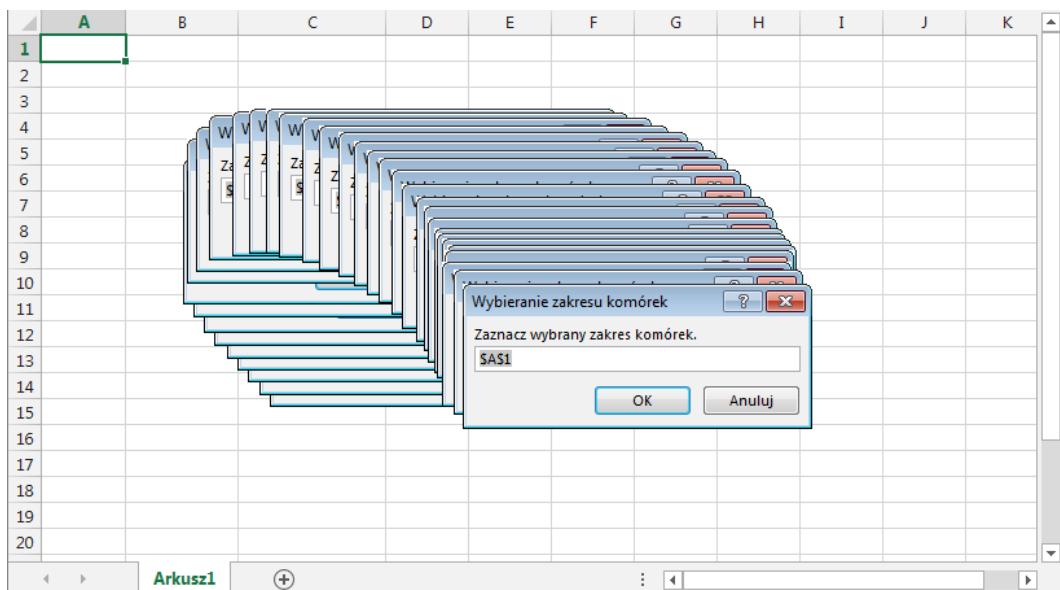


Pamiętaj, podczas używania metody InputBox do zaznaczania zakresu komórek odświeżanie ekranu powinno być zawsze włączone. W przeciwnym wypadku każde przesunięcie okna dialogowego wyboru zakresu będzie pozostawało na ekranie paskudny ślad, tak jak to zostało zaprezentowane na rysunku 9.7. Do sterowania odświeżaniem ekranu w trakcie wykonywania makra powinieneś użyć właściwości ScreenUpdating obiektu Application.

Zliczanie zaznaczonych komórek

Można stworzyć makro przetwarzające zaznaczone komórki zakresu. Aby określić liczbę komórek w zaznaczonym zakresie (lub dowolnym innym), należy użyć właściwości Count obiektu Range. Przykładowo poniższa instrukcja wyświetla w oknie komunikatu liczbę aktualnie zaznaczonych komórek:

```
MsgBox Selection.Count
```



Rysunek 9.7. Przesuwanie okna dialogowego przy wyłączonym odświeżaniu ekranu pozostawia brzydkie ślady



Ze względu na fakt, że w począwszy od Excela 2007 maksymalne rozmiary arkusza zostały znacznie powiększone, właściwość Count może w pewnych sytuacjach generować błędów. Właściwość Count wykorzystuje dane typu Long, zatem największa wartość, jaką ta właściwość może przechowywać, to 2 147 483 647, stąd jeżeli zaznaczyesz na przykład 2048 pełnych kolumn (czyli 2 147 483 648 komórek), właściwość Count wygeneruje błąd. Na szczęście firma Microsoft dodała w Excelu, począwszy od wersji 2007, nową właściwość: CountLarge. Właściwość ta używa danych typu Double, które pozwalają na przechowywanie liczb z zakresu do 1,79+E308.

Wnioski? W przytaczającej większości przypadków właściwość Count będzie działała najzupełniej poprawnie. Jeżeli jednak zakładasz, że będziesz liczyć arkusze zawierające naprawdę duże ilości komórek (na przykład wszystkie komórki arkusza), to zamiast właściwości Count powinieneś użyć właściwości CountLarge.

Jeżeli aktywny arkusz zawiera zakres o nazwie dane, polecenie przedstawione poniżej przepisze liczbę jego komórek zmiennej CellCount:

```
CellCount = Range("dane").Count
```

Możesz również określić liczbę wierszy lub kolumn w zakresie. Poniższe wyrażenie wyznacza liczbę kolumn znajdujących się w aktualnie zaznaczonym zakresie:

```
Selection.Columns.Count
```

Oczywiście liczbę wierszy zakresu można również określić przy użyciu właściwości Rows. Poniższa instrukcja określa liczbę wierszy zakresu o nazwie dane i przypisuje wartość zmiennej RowCount:

```
RowCount = Range("dane").Rows.Count
```

Określanie typu zaznaczonego zakresu

Excel obsługuje kilka typów zaznaczeń zakresów. Oto one:

- pojedyncza komórka,
- ciągły zakres komórek,
- jedna lub więcej kolumn,
- jeden lub więcej wierszy,
- cały arkusz,
- dowolna kombinacja wyżej wymienionych typów, czyli zaznaczenie wielokrotne.

Ponieważ istnieje kilka typów zaznaczeń, w trakcie przetwarzania zakresu procedura języka VBA nie może z góry przewidzieć, jakiego rodzaju jest dany typ zaznaczenia. Na przykład zaznaczony obszar może składać się z dwóch zakresów komórek, A1:A10 i C1:C10 (aby utworzyć zaznaczenie wielokrotne, podczas zaznaczania kolejnych zakresów trzymaj wciśnięty klawisz *Ctrl*).

Jeżeli zakres został zdefiniowany przez wiele zaznaczeń, obiekt Range będzie się składać z oddzielnych obszarów. Aby stwierdzić, czy zaznaczenie jest zaznaczeniem wielokrotnym, należy użyć metody Areas zwracającej kolekcję Areas. Kolekcja reprezentuje wszystkie obszary wchodzące w skład zakresu stworzonego poprzez zaznaczenie wielokrotne.

W celu stwierdzenia, czy wybrany zakres posiada wiele obszarów, należy zastosować wyrażenie podobne do poniższego:

```
NumAreas = Selection.Areas.Count
```

Jeżeli zmienna NumAreas zawiera wartość większą od 1, zaznaczenie jest zaznaczeniem wielokrotnym.

Poniżej zamieszczono kod funkcji o nazwie AreaType, która zwraca łańcuch tekstu opisujący rodzaj zaznaczenia.

```
Function AreaType(RangeArea As Range) As String
    ' Funkcja określa rodzaj zaznaczonego obszaru
    Select Case True
        Case RangeArea.Cells.CountLarge = 1
            AreaType = "Komórka"
        Case RangeArea.CountLarge = Cells.CountLarge
            AreaType = "Arkusz"
        Case RangeArea.Rows.Count = Cells.Rows.Count
            AreaType = "Kolumna"
        Case RangeArea.Columns.Count = Cells.Columns.Count
            AreaType = "Wiersz"
        Case Else
            AreaType = "Blok"
    End Select
End Function
```

Funkcja jako argument pobiera obiekt Range i zwraca jeden z pięciu łańcuchów opisujących obszar — Komórka, Arkusz, Kolumna, Wiersz lub Blok. W celu określenia, które z pięciu wyrażeń porównujących ma wartość True, funkcja korzysta z konstrukcji Select Case. Na przykład: jeżeli zakres składa się z jednej komórki, funkcja zwróci łańcuch Komórka. Jeżeli liczba komórek zakresu jest równa liczbie komórek arkusza, funkcja zwróci łańcuch Arkusz. Jeżeli liczba wierszy zakresu jest równa liczbie wierszy arkusza, funkcja zwróci łańcuch Kolumna. Jeżeli liczba kolumn zakresu jest równa liczbie kolumn arkusza, funkcja zwróci łańcuch Wiersz. Jeżeli żadne wyrażenie instrukcji Case nie będzie miało wartości True, funkcja zwróci łańcuch Blok.

Zauważ, że do liczenia komórek użyta została właściwość CountLarge, ponieważ — jak już wspominaliśmy wcześniej — całkowita liczba zaznaczonych komórek w arkuszu może teoretycznie przekroczyć limit typu danych właściwości Count.



W sieci

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt *Zaznaczanie.xlsxm* zawierający procedurę RangeDescription, która posługuje się funkcją AreaType w celu wyświetlenia okna komunikatu opisującego typ zaznaczenia aktualnego zakresu. Na rysunku 9.8 pokazano przykład jej działania. Zrozumienie zasad działania funkcji będzie stanowiło dobre przygotowanie do wykonywania kolejnych operacji na obiektach klasy Range.

Rysunek 9.8. Procedura AboutRangeSelection analizuje aktualnie zaznaczony zakres



Excel pozwala na tworzenie wielu identycznych zaznaczeń. Na przykład, jeżeli trzymając wciśnięty klawisz **Ctrl**, pięciokrotnie klikniesz komórkę A1, zaznaczenie będzie złożone z pięciu identycznych obszarów. Procedura RangeDescription uwzględnia taką sytuację i nie zlicza wielokrotnie tych samych komórek. Zwróć również uwagę na jedną z nowych funkcji Excela 2013, który używa progresywnych odcieni szarości do wyróżniania nakładających się na siebie zaznaczeń zakresów komórek.

Wydajne przetwarzanie komórek zaznaczonego zakresu przy użyciu pętli

Jednym z częściej wykonywanych przez makra zadań jest sprawdzanie poszczególnych komórek zakresu i wykonywanie określonych operacji, jeżeli komórka spełnia zadane kryterium. Kolejna procedura, której kod przedstawiamy poniżej, jest właśnie przykładem takiego makra. Procedura `ColorNegative` ustawia czerwony kolor tła dla wszystkich komórek zaznaczenia, które przechowują wartość ujemną. Kolor tła pozostałych komórek jest zerowany.



Poniższy przykład został opracowany tylko i wyłącznie w celach edukacyjnych. W zastosowaniach praktycznych o wiele lepszym rozwiązaniem będzie po prostu użycie mechanizmu formatowania warunkowego.

```
Sub ColorNegative()
    ' Jeżeli wartość jest ujemna, zmienia kolor tła komórki na czerwony
    Dim cell As Range
    If TypeName(Selection) <> "Range" Then Exit Sub
    Application.ScreenUpdating = False
    For Each cell In Selection
        If cell.Value < 0 Then
            cell.Interior.ColorIndex = RGB(255, 0, 0)
        Else
            cell.Interior.ColorIndex = xlNone
        End If
    Next cell
End Sub
```

Z pewnością procedura `ColorNegative` zadziała, ale zawiera poważny błąd. Dla przykładu: co się stanie, jeżeli obszar danych na arkuszu jest bardzo mały, a użytkownik zaznaczy na przykład całą kolumnę? Albo 10 kolumn? Albo może nawet cały arkusz? Jak się zapewne sam domyślasz, nie ma żadnej potrzeby sprawdzania wszystkich pustych, nie używanych komórek arkusza, nie mówiąc już o tym, że przy dużych zaznaczonych obszarach użytkownik z pewnością poddałby się, zanim cała procedura dobiegłaby do końca.

Lepszym rozwiązaniem jest procedura `ColorNegative2`, której kod przedstawiamy poniżej. W tej poprawionej wersji utworzyliśmy zmienną obiektową `WorkRange`, której zawartość odpowiada części wspólnej zaznaczonego obszaru i użytego obszaru arkusza.

```
Sub ColorNegative2()
    ' Jeżeli wartość jest ujemna, zmienia kolor tła komórki na czerwony
    Dim WorkRange As Range
    Dim cell As Range
    If TypeName(Selection) <> "Range" Then Exit Sub
    Application.ScreenUpdating = False
    Set WorkRange = Application.Intersect(Selection, ActiveSheet.UsedRange)
    For Each cell In WorkRange
        If cell.Value < 0 Then
            cell.Interior.Color = RGB(255, 0, 0)
        Else
            cell.Interior.Color = xlNone
        End If
    Next cell
End Sub
```

Przykładową sytuację ilustruje rysunek 9.9. Jak widać, zaznaczona jest cała kolumna D (czyli 1 048 576 komórek), ale użyty zakres komórek sprowadza się już tylko do obszaru B2:I18. Zatem przecięciem tych dwóch obszarów jest zakres D2:D18, co jest zdecydowanie mniejszym zakresem niż początkowe zaznaczenie. Nie trzeba chyba nikogo dłucho przekonywać, że różnica pomiędzy czasem przetwarzania 15 komórek a 1 048 576 komórek będzie naprawdę znacząca.

Rysunek 9.9.

Zastosowanie przecięcia zaznaczonego zakresu i użytego obszaru arkusza skutkuje znaczącym zmniejszeniem liczby komórek, które będziemy musieli przetwarzać

A	B	C	D	E	F	G	H	I	J
1			-7	3	-3	7	-6	-9	
2	-5	0	-6	-10	-1	10	9	-10	
3	-5	-6	1	4	-3	3	-8	-3	
4	-2	5	-3	-8	1	8	8	6	
5	1	8	1	3	-1	7	5	2	
6	0	-4	-3	8	1	-8	7	9	
7	-10	4	1	7	10	-1	8	-3	
8	5	-4	-1	-8	-2	-1	-6	8	
9	1	4	1	-1	7	6	7	9	
10	-8	-3	10	-10	9	-9	2	-4	
11	0	-2	-2	-1	9	7	7	7	
12	10	4	7	6	10	-10	10	4	
13	-5	-1	9	7	0	8	6	9	
14	3	-4	10	-10	9	-9	2	-4	
15	4	9	0	8	4	7	-1	-4	
16	0	1	9	-9	2	7	-7	0	
17									
18									
19									
20									

Procedura ColorNegative2 jest znacznie lepsza, ale mimo to nadal nie jest tak efektywna, jak być powinna, a to z prostego powodu — nadal niepotrzebnie przetwarza puste komórki. Trzecia wersja naszej procedury, ColorNegative3, jest nieco dłuższa, ale jednocześnie o wiele bardziej wydajna. W celu wygenerowania dwóch podzbiorów zaznaczenia użyłem metody SpecialCells. Pierwszy podzbiór obejmuje jedynie komórki zawierające stałe numeryczne (ConstantCells), natomiast drugi — komórki przechowujące formuły numeryczne (FormulaCells). Komórki obu podzbiorów są następnie przetwarzane za pomocą dwóch konstrukcji For Each ... Next. W efekcie przetwarzane są tylko niepuste komórki zawierające wartości numeryczne, dzięki czemu uzyskujemy znaczące zwiększenie szybkości działania makra.

```

Sub ColorNegative3()
    ' Jeżeli wartość jest ujemna, zmienia kolor tła komórki na czerwony
    Dim FormulaCells As Range, ConstantCells As Range
    Dim cell As Range
    If TypeName(Selection) <> "Range" Then Exit Sub
    Application.ScreenUpdating = False

    ' Tworzy podzbiory oryginalnego obszaru zaznaczenia
    On Error Resume Next
    Set FormulaCells = Selection.SpecialCells(xlFormulas, xlNumbers)
    Set ConstantCells = Selection.SpecialCells(xlConstants, xlNumbers)
    On Error GoTo 0

    ' Przetwarzanie komórek zawierających formuły
    If Not FormulaCells Is Nothing Then
        For Each cell In FormulaCells

```

```

        If cell.Value < 0 Then
            cell.Interior.Color = RGB(255, 0, 0)
        Else
            cell.Interior.Color = xlNone
        End If
    Next cell
End If
' Przetwarzanie komórek zawierających stałe wartości numeryczne
If Not ConstantCells Is Nothing Then
    For Each cell In ConstantCells
        If cell.Value < 0 Then
            cell.Interior.Color = RGB(255, 0, 0)
        Else
            cell.Interior.Color = xlNone
        End If
    Next cell
End If
End Sub

```



Zastosowanie instrukcji On Error jest konieczne, ponieważ metoda SpecialCells generuje błąd, gdy żadna komórka nie spełnia kryterium.



Skoroszyt z tymi przykładami (*Tworzenie wydajnych pętli.xlsm*), zawierający trzy wersje procedury ColorNegative, znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Usuwanie wszystkich pustych wierszy

Poniższa procedura usuwa puste wiersze aktywnego arkusza. Procedura jest szybka i wydajna, ponieważ nie sprawdza wszystkich wierszy, a jedynie wiersze używane przez zakres, który jest identyfikowany za pomocą właściwości UsedRange obiektu Worksheet.

```

Sub DeleteEmptyRows()
    Dim LastRow As Long
    Dim r As Long
    Dim Counter As Long
    Application.ScreenUpdating = False
    LastRow = ActiveSheet.UsedRange.Rows.Count + ActiveSheet.UsedRange.Rows(1).Row - 1
    For r = LastRow To 1 Step -1
        If Application.WorksheetFunction.CountA(Rows(r)) = 0 Then
            Rows(r).Delete
            Counter = Counter + 1
        End If
    Next r
    Application.ScreenUpdating = True
    MsgBox Counter & " pustych wierszy zostało usuniętych."
End Sub

```

Pierwszym krokiem jest określenie ostatnio używanego wiersza, a następnie przypisanie jego numeru zmiennej LastRow. Nie jest to takie proste, jak mogłoby się wydawać, ponieważ używany zakres może, ale nie musi rozpoczynać się od wiersza 1. A zatem wartość zmiennej LastRow jest obliczana poprzez określenie liczby wierszy używanego zakresu, dodanie numeru pierwszego wiersza zakresu i odjęcie jedynki.

W celu stwierdzenia, czy wiersz jest pusty, procedura korzysta z funkcji arkuszowej COUNTA (ILE.NIEPUSTYCH) Excela. Jeżeli dla określonego wiersza funkcja zwróci wartość 0, oznacza to, że jest pusty. Procedura przetwarza wiersze od dołu do góry, a ponadto w pętli For ... Next używa ujemnej wartości skoku (Step). Jest to niezbędne, ponieważ operacja usuwania wierszy powoduje, że wszystkie kolejne wiersze są przesuwane w górę arkusza. Jeżeli pętla przetwarzałaby wiersze od góry do dołu, jej licznik po usunięciu wiersza nie miałby właściwej wartości.

Makro wykorzystuje również inną zmienną, Counter, do śledzenia liczby usuniętych wierszy. Liczba ta jest wyświetlana w oknie dialogowym, kiedy procedura kończy działanie.



Skoroszyt z tym przykładem (*Usuń puste wiersze.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

W sieci

Powielanie wierszy

Przykład, który omówimy w tym podrozdziale, ilustruje sposób wykorzystania VBA do tworzenia duplikatów istniejących wierszy. Na rysunku 9.10 przedstawiono wygląd skoroszytu zawierającego informacje o loterii biurowej. Kolumna A przechowuje imiona graczy, kolumna B liczbę losów zakupionych przez poszczególne osoby, a w kolumnie C znajduje się liczba losowa (wygenerowana przy użyciu funkcji RAND). Zwycięzca zostanie wyłoniony przez sortowanie danych w kolumnie C (wygrywa osoba, do której została przypisana największa liczba losowa).

Rysunek 9.10.
Zadanie polega
na powielaniu
istniejących wierszy
w oparciu o wartości
z kolumny B

	A	B	C	D
1	Imię	Liczba biletów	Liczba losowa	
2	Adam	1	0,084470747	
3	Barbara	2	0,729463003	
4	Czesław	1	0,205483062	
5	Dariusz	5	0,897806312	
6	Franek	3	0,369899976	
7	Grzegorz	1	0,291293755	
8	Hubert	1	0,644377538	
9	Inka	2	0,918673327	
10	Marek	1	0,655386989	
11	Norbert	10	0,116601653	
12	Paweł	2	0,707836524	
13	Rafał	1	0,264928433	
14	Wiesław	2	0,908818181	
15				
16				
17				

◀ ▶ Arkusz1 +

Nasze zadanie polega na powielaniu istniejących wierszy dla poszczególnych osób, tak aby każda z nich miała tyle osobnych wierszy, ile zakupiła losów. Na przykład Barbara kupiła 2 losy, a zatem powinniśmy dla niej utworzyć 2 osobne wiersze. Kod procedury realizującej takie zadanie został zamieszczony poniżej:

```
Sub DupeRows()
    Dim cell As Range
    ' Pierwsza komórka z liczbą losów
```

```

Set cell = Range("B2")
Do While Not IsEmpty(cell)
    If cell > 1 Then
        Range(cell.Offset(1, 0), cell.Offset(cell.Value - 1, _
            0)).EntireRow.Insert
        Range(cell, cell.Offset(cell.Value - 1, 1)).EntireRow.FillDown
    End If
    Set cell = cell.Offset(cell.Value, 0)
Loop
End Sub

```

Zmiennej obiektowej `cell` zostaje przypisana wartość reprezentująca komórkę B2, czyli pierwszą komórkę, w której przechowywana jest liczba zakupionych losów. Pętla wstawia nowy wiersz i następnie kopiuje go odpowiednią ilość razy przy użyciu metody `FillDown`. Zmienna `cell` jest inkrementowana tak, aby wskazywała na liczbę losów zakupionych przez kolejną osobę i procedura kontynuuje działanie aż do momentu napotkania pierwszej pustej komórki. Na rysunku 9.11 przedstawiono wygląd arkusza po zakończeniu działania procedury.

Rysunek 9.11.
Procedura dodała do arkusza nowe wiersze w oparciu o wartości w kolumnie B

	A	B	C	D
1	Imię	Liczba biletów	Liczba losowa	
2	Adam	1	0,835736322	
3	Barbara	2	0,919536022	
4	Barbara	2	0,292479801	
5	Czesław	1	0,605373498	
6	Dariusz	5	0,800223768	
7	Dariusz	5	0,745315082	
8	Dariusz	5	0,661660522	
9	Dariusz	5	0,112026034	
10	Dariusz	5	0,415874182	
11	Franek	3	0,4371568	
12	Franek	3	0,066350504	
13	Franek	3	0,38628981	
14	Grzegorz	1	0,017185406	
15	Hubert	1	0,814217192	
16	Inka	2	0,829365312	
17	Inka	2	0,957102055	
18	Marek	1	0,982584205	
19	Norbert	10	0,870535963	
20	Norbert	10	0,774169106	
21	Norbert	10	0,101379233	
22	Norbert	10	0,900583862	
23	Norbert	10	0,795690139	
24	Norbert	10	0,335875045	

← → Arkusz1 +

Skoroszyt z tym przykładem (*Powielanie wierszy.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).



W sieci

Określanie, czy zakres zawiera się w innym zakresie

Poniższa funkcja InRange pobiera dwa argumenty (obiekty klasy Range) i zwraca wartość True, jeżeli pierwszy zakres zawiera się w drugim. Przedstawiona funkcja może być wykorzystywana w formułach arkuszowych, ale będzie znacznie bardziej efektywna, jeżeli zostanie wywołana z poziomu innej procedury VBA.

```
Function InRange(rng1, rng2) As Boolean
    ' Zwraca wartość True, jeżeli rng1 jest podzbiorem rng2
    On Error GoTo ErrHandler
    If Union(rng1, rng2).Address = rng2.Address Then
        InRange = True
        Exit Function
    End If
ErrHandler:
    InRange = False
End Function
```

Metoda Union obiektu Application zwraca obiekt Range reprezentujący unię dwóch obiektów Range będących jej argumentami. Unia będąca wynikiem działania tej metody zawiera wszystkie komórki z obu zakresów. Jeżeli adres unii obu zakresów jest taki sam jak adres drugiego zakresu (drugiego argumentu wywołania metody), oznacza to, że pierwszy zakres komórek w całości zawiera się w obrębie drugiego zakresu komórek.

Jeżeli zakresy będące argumentami metody Union znajdują się w różnych arkuszach, wywołanie tej metody spowoduje wygenerowanie błędu, który jest następnie obsługiwany za pomocą polecenia On Error.



Skoroszyt z tym przykładem (*Funkcja InRange.xls*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Określanie typu danych zawartych w komórce

Excel oferuje kilka wbudowanych funkcji arkuszowych, które pomagają w określaniu typu danych zawartych w komórce. Należy do nich zaliczyć funkcje CZY.TEKST, CZY.LOGICZNA i CZY.BŁĄD. Dodatkowo język VBA zawiera funkcje IsEmpty, IsDate i IsNumeric.

Zamieszczona poniżej funkcja CellType akceptuje argument Range i zwraca łańcuch (Pusta, Tekst, Logiczny, Błąd, Data, Czas lub Liczba) opisujący typ danych zawartych w górnej lewej komórce zakresu.

```
Function CellType(Rng) As String
    ' Zwraca typ górnej lewej komórki zakresu
    Dim TheCell As Range
    Set TheCell = Rng.Range("A1")
    Select Case True
        Case IsEmpty(TheCell)
            CELLTYPE = "Pusta"
        Case TheCell.NumberFormat = "@"
            CELLTYPE = "Tekst"
        Case Application.IsText(TheCell)
            CELLTYPE = "Logiczny"
        Case Else
            CELLTYPE = "Błąd"
    End Select
End Function
```

```

CELLTYPE = "Tekst"
Case Application.IsLogical(TheCell)
    CELLTYPE = "Wartość logiczna"
Case Application.IsErr(TheCell)
    CELLTYPE = "Błąd"
Case IsDate(TheCell)
    CELLTYPE = "Data"
Case InStr(1, TheCell.Text, ":") > 0
    CELLTYPE = "Czas"
Case IsNumeric(TheCell)
    CELLTYPE = "Liczba"
End Select
End Function

```

Funkcja CellType może zostać użyta w formule arkusza lub wywołana z innej procedury języka VBA. Na rysunku 9.12 przedstawiono sytuację, gdzie funkcja została użyta w formułach w kolumnie B, które jako argumenty wywołania wykorzystują dane znajdujące się w kolumnie A. Wartości znajdujące się w kolumnie C to po prostu opisy poszczególnych przykładów.

Rysunek 9.12.
Zastosowanie funkcji CellType do określania rodzaju danych przechowywanych w komórce

	A	B	C	D	E
1	145,4	Liczba	Wartość liczbową		
2	8,6	Liczba	Formuła zwracająca wartość liczbową		
3	Arkusz budżetu	Tekst	łańcuch tekstu		
4	FALSZ	Logiczna	Formuła zwracająca wartość logiczną		
5	PRAWDA	Logiczna	Wartość logiczna		
6	#DZIEL/0!	Błąd	Błąd formuły		
7	2013-03-01	Data	Formuła zwracająca datę		
8	4:00 PM	Czas	Czas		
9	1-13-10 5:25 AM	Data	Data i czas		
10	143	Tekst	Wartość poprzedzona apostrofem		
11	434	Liczba	Komórka sformatowana jako tekst		
12	A1:C4	Tekst	Tekst zawierający dwukropki		
13		Tekst	Komórka zawierająca pojedynczą spację		
14		Tekst	Komórka zawierająca pusty łańcuch tekstu (pojedynczy apostrof)		
15					
16					

Zwróć uwagę na użycie polecenia Set TheCell. Funkcja CellType akceptuje argument Range dowolnej wielkości, ale ta instrukcja powoduje, że funkcja przetwarza tylko górną lewą komórkę zakresu reprezentowanego przez zmienną TheCell.



Skoroszyt z tym przykładem (*Funkcja CellType.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Odczytywanie i zapisywanie zakresów

Wiele operacji wykonywanych przy użyciu języka VBA wymaga przenoszenia wartości z tablicy do zakresu lub z zakresu do tablicy. Excel odczytuje dane z zakresów komórek o wiele szybciej, niż je w nich zapisuje, co wynika zapewne z tego, że ta druga operacja odbywa się z wykorzystaniem mechanizmów przeliczania arkusza. Poniżej przedstawiamy kod procedury WriteReadRange, która demonstruje porównanie względnych szybkości wykonywania operacji zapisu i odczytu zakresu komórek.

Procedura tworzy tablicę, a następnie za pomocą pętli For ... Next zapisuje jej zawartość w zakresie i ponownie wczytuje ją do tablicy. Czas wymagany do wykonania każdej z tych operacji jest obliczany przy użyciu funkcji Timer języka VBA.

```
Sub WriteReadRange()
    Dim MyArray()
    Dim Time1 As Double
    Dim NumElements As Long, i As Long
    Dim WriteTime As String, ReadTime As String
    Dim Msg As String

    NumElements = 250000
    ReDim MyArray(1 To NumElements)

    ' Wypełnienie tablicy
    For i = 1 To NumElements
        MyArray(i) = i
    Next i

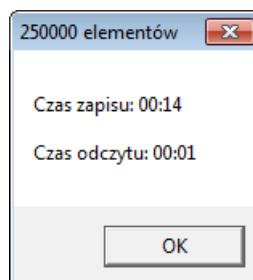
    ' Zapis danych z tablicy do zakresu
    Time1 = Timer
    For i = 1 To NumElements
        Cells(i, 1) = MyArray(i)
    Next i
    WriteTime = Format(Timer - Time1, "00:00")

    ' Odczytanie danych z zakresu i załadowanie do tablicy
    Time1 = Timer
    For i = 1 To NumElements
        MyArray(i) = Cells(i, 1)
    Next i
    ReadTime = Format(Timer - Time1, "00:00")

    ' Wyświetlenie wyników
    Msg = "Czas zapisu: " & WriteTime
    Msg = Msg & vbCrLf
    Msg = Msg & "Czas odczytu: " & ReadTime
    MsgBox Msg, vbOKOnly, NumElements & " elementów"
End Sub
```

Wyniki działania procedury zostały przedstawione na rysunku 9.13. Na moim komputerze przepisanie tablicy liczącej 250 000 elementów do zakresu komórek zajęło 14 sekund, natomiast wczytanie zakresu komórek do tablicy poniżej jednej sekundy.

Rysunek 9.13.
Porównanie czasów
zapisu i odczytu
danych z zakresu
komórek



Lepsza metoda zapisywania danych do zakresu komórek

W poprzednim przykładzie, aby przenieść zawartość tablicy do zakresu arkusza, użыта została pętla For ... Next. W tym podrozdziale zademonstrujemy dużo wydajniejszą metodę osiągnięcia tego samego celu.

Kod procedury przedstawiony poniżej ilustruje najbardziej oczywisty (ale niestety nie naj-wydajniejszy) sposób wypełniania zakresu danymi. Do umieszczenia danych w zakresie użytą została pętla For ... Next.

```
Sub LoopFillRange()
    ' Wypełnianie zakresu przy użyciu pętli przetwarzającej komórki
    Dim CellsDown As Long, CellsAcross As Integer
    Dim CurrRow As Long, CurrCol As Integer
    Dim StartTime As Double
    Dim CurrVal As Long

    ' Pobieranie wymiarów zakresu komórek
    CellsDown = InputBox("Ile komórek w pionie?")
    If CellsDown = 0 Then Exit Sub
    CellsAcross = InputBox("Ile komórek w poziomie?")
    If CellsAcross = 0 Then Exit Sub

    ' Zarejestrowanie czasu rozpoczęcia
    StartTime = Timer

    ' Przetwarzanie komórek w pętli i wstawianie wartości
    CurrVal = 1
    Application.ScreenUpdating = False
    For CurrRow = 1 To CellsDown
        For CurrCol = 1 To CellsAcross
            ActiveCell.Offset(CurrRow - 1, -
                CurrCol - 1).Value = CurrVal
            CurrVal = CurrVal + 1
        Next CurrCol
    Next CurrRow

    ' Wyświetlanie czasu trwania operacji
    Application.ScreenUpdating = True
    MsgBox Format(Timer - StartTime, "00.00") & " sekund"
End Sub
```

Kolejny przykład demonstruje o wiele szybszą metodę pozwalającą na uzyskanie tego samego efektu. Procedura wstawia do tablicy poszczególne wartości, a następnie za pomocą jednej instrukcji przenosi zawartość tablicy do zakresu.

```
Sub ArrayFillRange()
    ' Wypełnianie zakresu poprzez transfer tablicy
    Dim CellsDown As Long, CellsAcross As Integer
    Dim i As Long, j As Integer
    Dim StartTime As Double
    Dim TempArray() As Long
    Dim TheRange As Range
    Dim CurrVal As Long

    ' Pobieranie wymiarów
    CellsDown = InputBox("Ile komórek w pionie?")
```

```
If CellsDown = 0 Then Exit Sub
CellsAcross = InputBox("Ile komórek w poziomie?")
If CellsAcross = 0 Then Exit Sub

    ' Zarejestrowanie czasu rozpoczęcia
    StartTime = Timer

    ' Przeskalowanie wymiarów tablicy tymczasowej
    ReDim TempArray(1 To CellsDown, 1 To CellsAcross)

    ' Zdefiniowanie zakresu w arkuszu
    Set TheRange = ActiveCell.Range(Cells(1, 1), _
        Cells(CellsDown, CellsAcross))

    ' Wypełnianie tablicy tymczasowej
    CurrVal = 0
    Application.ScreenUpdating = False
    For i = 1 To CellsDown
        For j = 1 To CellsAcross
            TempArray(i, j) = CurrVal + 1
            CurrVal = CurrVal + 1
        Next j
    Next i
    ' Transfer tablicy tymczasowej do arkusza
    TheRange.Value = TempArray

    ' Wyświetlanie czasu trwania operacji
    Application.ScreenUpdating = True
    MsgBox Format(Timer - StartTime, "00.00") & " sekund"
End Sub
```

W moim systemie wypełnienie zakresu o wymiarach 1000×250 komórek (250 000 komórek) przy użyciu pętli zajęło 15,80 sekundy. Metoda oparta na transferze zawartości tablicy do uzyskania identycznego efektu potrzebowała zaledwie 0,15 sekundy, czyli działała prawie 100 razy szybciej! Jaki wniosek wynika z tego przykładu? Jeżeli chcesz przenieść do arkusza dużą ilość danych, wszędzie, gdzie tylko jest to możliwe, unikaj stosowania pętli.



Osiągnięte czasy w dużej mierze zależą od obecności w arkuszu innych formuł. W praktyce lepsze czasy otrzymasz w sytuacji, kiedy podczas testu nie będą otwarte inne skoroszyty zawierające makra lub kiedy przełączysz Excela w tryb ręcznego przeliczania arkusza.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt *Wypełnianie zakresu przy użyciu pętli i tablicy.xlsm*, zawierający procedury WriteReadRange, LoopFillRange oraz ArrayFillRange.

Przenoszenie zawartości tablic jednowymiarowych

W poprzednim przykładzie zastosowano tablicę dwuwymiarową, dobrze sprawdzającą się w arkuszach, w których dane przechowywane są w uporządkowanej strukturze wierszy i kolumn.

Aby przenieść zawartość tablicy jednowymiarowej, zakres musi mieć orientację poziomą — czyli inaczej mówiąc, posiadać jeden wiersz z wieloma *kolumnami*. Jeżeli jednak musisz

użyć zakresu pionowego, najpierw będziesz musiał dokonać transponowania tablicy z poziomej na pionową. Aby to zrobić, możesz użyć funkcji arkuszowej TRANSPOSE (TRANSPONUJ) Excela. Poniższe polecenie przenosi tablicę liczącą 100 elementów do pionowego zakresu arkusza (A1:A100):

```
Range("A1:A100").Value = Application.WorksheetFunction.Transpose(MyArray)
```

Przenoszenie zawartości zakresu do tablicy typu Variant

W tym podrozdziale omówimy kolejną metodę przetwarzania zawartości arkusza przy użyciu języka VBA. W poniższym przykładzie zawartość zakresu komórek jest przenoszona do dwuwymiarowej tablicy typu Variant. Następnie w oknach komunikatów są wyświetlane górne granice każdego wymiaru tablicy.

```
Sub RangeToVariant()
    Dim x As Variant
    x = Range("A1:L600").Value
    MsgBox UBound(x, 1)
    MsgBox UBound(x, 2)
End Sub
```

W pierwszym oknie komunikatu jest wyświetlana wartość 600 (liczba wierszy oryginalnego zakresu), natomiast w drugim — 12 (liczba kolumn). Jak sam się przekonasz, przeniesienie zawartości zakresu do tablicy typu Variant odbywa się prawie natychmiast.

Poniższa procedura wczytuje zawartość zakresu o nazwie data do tablicy typu Variant, wykonuje na poszczególnych elementach tablicy prostą operację mnożenia, a następnie ponownie przenosi dane zapisane w tablicy do zakresu.

```
Sub RangeToVariant2()
    Dim x As Variant
    Dim r As Long, c As Integer

    ' Wczytanie danych do tablicy typu Variant
    x = Range("data").Value

    ' Wykonanie pętli dla tablicy typu Variant
    For r = 1 To UBound(x, 1)
        For c = 1 To UBound(x, 2)
            ' Mnożenie kolejnych elementów tablicy przez 2
            x(r, c) = x(r, c) * 2
        Next c
    Next r

    ' Ponowne przeniesienie zawartości tablicy typu Variant do arkusza
    Range("data") = x
End Sub
```

Jak sam się możesz przekonać, cała procedura działa naprawdę szybko. Przetwarzanie 30 000 komórek na moim komputerze trwało poniżej jednej sekundy.



Skoroszyt z tym przykładem (*Przenoszenie tablicy typu Variant.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zaznaczanie komórek na podstawie wartości

Nasz kolejny przykład ilustruje, w jaki sposób można zaznaczać wybrane komórki w oparciu o ich wartości. Co ciekawe, Excel nie posiada swojego własnego mechanizmu, który umożliwiałby bezpośrednie wykonanie takiej operacji. Poniżej przedstawiamy kod procedury `SelectByValue`, której zadaniem jest zaznaczenie komórek zakresu zawierających wartości ujemne, aczkolwiek można to w łatwy sposób zmodyfikować i dopasować do własnych potrzeb.

```
Sub SelectByValue()
    Dim Cell As Object
    Dim FoundCells As Range
    Dim WorkRange As Range

    If TypeName(Selection) <> "Range" Then Exit Sub

    ' Sprawdzamy wszystko czy tylko zaznaczenie
    If Selection.CountLarge = 1 Then
        Set WorkRange = ActiveSheet.UsedRange
    Else
        Set WorkRange = Application.Intersect(Selection, ActiveSheet.UsedRange)
    End If

    ' Zredukuj liczbę przetwarzanych komórek do komórek zawierających wartości numeryczne
    On Error Resume Next
    Set WorkRange = WorkRange.SpecialCells(xlConstants, xlNumbers)
    If WorkRange Is Nothing Then Exit Sub
    On Error GoTo 0

    ' Sprawdzaj w pętli kolejne komórki i dodawaj do zakresu FoundCells, jeżeli spełniają kryterium
    For Each Cell In WorkRange
        If Cell.Value < 0 Then
            If FoundCells Is Nothing Then
                Set FoundCells = Cell
            Else
                Set FoundCells = Union(FoundCells, Cell)
            End If
        End If
    Next Cell

    ' Pokaż komunikat lub zaznacz komórki
    If FoundCells Is Nothing Then
        MsgBox "Nie znaleziono komórek spełniających kryterium."
    Else
        FoundCells.Select
        MsgBox "Zaznaczono " & FoundCells.Count & " komórek."
    End If
End Sub
```

Procedura rozpoczyna działanie od sprawdzenia zaznaczonego zakresu. Jeżeli jest to pojedyncza komórka, przeszukiwany jest cały arkusz. Jeżeli zaznaczone zostały co najmniej 2 komórki, wtedy przeszukiwany jest tylko zaznaczony zakres. Zakres, który będzie przeszukiwany, jest następnie redefiniowany poprzez użycie metody `SpecialCells` do utworzenia obiektu klasy `Range`, który składa się tylko z komórek zawierających wartości numeryczne.

Kod w pętli For ... Next sprawdza wartości kolejnych komórek. Jeżeli komórka spełnia zadane kryterium (czyli jej zawartość jest mniejsza od 0), komórka jest dodawana za pomocą metody Union do obiektu FoundCells klasy Range. Zwróć uwagę na fakt, że nie możesz użyć metody Union dla pierwszej komórki — jeżeli zakres FoundCells nie zawiera żadnych komórek, próba użycia metody Union spowoduje wygenerowanie błędu. Właśnie dlatego w naszym programie zamieściliśmy kod sprawdzający, czy zawartość zakresu FoundCells to Nothing.

Kiedy pętla kończy swoje działanie, obiekt FoundCells składa się z komórek, które spełniają kryterium wyszukiwania (jeżeli nie znaleziono żadnych komórek, jego zawartością będzie Nothing). Jeżeli żadna komórka nie spełni kryteriów, na ekranie zostanie wyświetcone okno dialogowe z odpowiednim komunikatem. W przeciwnym razie komórki spełniające kryteria zostaną zaznaczone.



Skoroszyt z tym przykładem (*Zaznaczanie według wartości.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Kopiowanie nieciągłego zakresu komórek

Jeżeli kiedykolwiek próbowałeś kopować nieciągły zakres komórek, z pewnością przekonałeś się, że Excel nie obsługuje takiej operacji. Próba jej wykonania kończy się wyświetleniem komunikatu *Wykonanie tego polecenia dla kilku zakresów nie jest możliwe*.

Istnieją jednak dwa wyjątki. Pierwszy z nich dotyczy sytuacji, kiedy próbujesz kopować zaznaczenie wielokrotne, składające się z całych wierszy lub kolumn, a drugi, kiedy zaznaczenie wielokrotne jest zlokalizowane w tych samych wierszach lub kolumnach. W takich sytuacjach Excel pozwala na wykonanie takiej operacji.

Kiedy napotykaš takie ograniczenia w programie Excel, zazwyczaj możesz je obejść za pomocą odpowiedniego makra. Procedura, którą omówimy poniżej, jest przykładem takiego makra, które pozwala na kopowanie wielu zaznaczonych zakresów komórek w inne miejsce arkusza.

```
Sub CopyMultipleSelection()
    Dim SelAreas() As Range
    Dim PasteRange As Range
    Dim UpperLeft As Range
    Dim NumAreas As Long, i As Long
    Dim TopRow As Long, LeftCol As Long
    Dim RowOffset As Long, ColOffset As Long

    If TypeName(Selection) <> "Range" Then Exit Sub

    ' Zapisuje poszczególne zakresy jako osobne obiekty klasy Range
    NumAreas = Selection.Areas.Count
    ReDim SelAreas(1 To NumAreas)
    For i = 1 To NumAreas
        Set SelAreas(i) = Selection.Areas(i)
    Next

    ' Określa górną, lewą komórkę zaznaczonych obszarów
    TopRow = ActiveSheet.Rows.Count
```

```

LeftCol = ActiveSheet.Columns.Count
For i = 1 To NumAreas
    If SelAreas(i).Row < TopRow Then TopRow = SelAreas(i).Row
    If SelAreas(i).Column < LeftCol Then LeftCol = SelAreas(i).Column
Next
Set UpperLeft = Cells(TopRow, LeftCol)

Pobiera adres obszaru docelowego
On Error Resume Next
Set PasteRange = Application.InputBox(
    (Prompt:="Podaj adres lewej, górnej komórki obszaru docelowego: ", _
    Title:="Kopiowanie wielu zakresów", _
    Type:=8)
On Error GoTo 0
Jeżeli operacja została anulowana, zakończ działanie
If TypeName(PasteRange) <> "Range" Then Exit Sub

Upewnij się, że używamy tylko lewej, górnej komórki
Set PasteRange = PasteRange.Range("A1")

Kopiowanie i wklejanie poszczególnych zakresów
For i = 1 To NumAreas
    RowOffset = SelAreas(i).Row - TopRow
    ColOffset = SelAreas(i).Column - LeftCol
    SelAreas(i).Copy PasteRange.Offset(RowOffset, ColOffset)
Next i
End Sub

```

Na rysunku 9.14 przedstawiono okno dialogowe, w którym użytkownik powinien zdefiniować adres obszaru docelowego.

The screenshot shows a portion of an Excel spreadsheet with data from row 4 to 19. A range of cells from B4 to D6 is currently selected. A 'Copy Special' dialog box is open over the spreadsheet, prompting the user to 'Wybierz lewą, górną komórkę obszaru docelowego:' (Select the top-left cell of the target range). The dialog has 'OK' and 'Anuluj' (Cancel) buttons.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Zaznacz nieciągły zakres komórek i naciśnij przycisk												
2													
3													
4	18	47	94	12	41	87	72	90					
5	100	89	42	65	84	54	1	63					
6	82	77	47	99	68	76	8	49					
7	23	51	17	97	26	8	88	30					
8	44	76	90	21	59	67	99	84					
9	14	44	89	55	98	50	10	6					
10	63	77	45	36	92	13	50	51					
11	9	91	68	19	91	67	19	85					
12	29	34	40	4	7	60	51	94					
13	9	30	19	89	98	13	79	74					
14	38	82	90	29	70	52							
15	81	22	33	98	28	40							
16	29	37	62	15	34	33							
17	86	64	3	88	12	20							
18	41	100	15	36	13	34							
19	88	74	6	68	67	19							
20	31	100	26	36	23	5	28	65					
21	2	5	49	54	26	47	75	22					
22	19	2	20	48	52	71	93	43					
23	5	88	86	12	100	23	56	40					
24	34	40	41	32	77	46	85	70					
25	845	1230	982	1013	1166	885	1147	1271					
26													

Rysunek 9.14. Zastosowanie metody `InputBox` programu Excel do pobierania lokalizacji komórki



Skoroszyt z tym przykładem (*Kopiowanie wielu zakresów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Dodatkowo w skoroszycie tym znajduje się inna wersja tej procedury, która ostrzega użytkownika, jeżeli w wyniku kopiowania zostaną nadpisane dane istniejące w obszarze docelowym.

Przetwarzanie skoroszytów i arkuszy

Kolejne przykłady omawiane w tym podrozdziale demonstrują metody przetwarzania skoroszytów i arkuszy przy użyciu języka VBA.

Zapisywanie wszystkich skoroszytów

Poniższa procedura przy użyciu pętli przetwarza wszystkie skoroszyty w kolekcji Workbooks i ponownie zapisuje każdy plik, który był już wcześniej zapisany:

```
Public Sub SaveAllWorkbooks()
    Dim Book As Workbook
    For Each Book In Workbooks
        If Book.Path <> "" Then Book.Save
    Next Book
End Sub
```

Zwróć uwagę na właściwość Path. Jeżeli właściwość Path danego skoroszytu jest pusta, oznacza to, że jego plik nigdy nie został zapisany (jest to nowy skoroszyt). Procedura ignoruje tego typu skoroszyty i zapisuje tylko te, których właściwość Path ma ustawioną dowolną wartość.

Bardziej efektywna wersja procedury powinna sprawdzać wartość właściwości Saved. Wspomniana właściwość ma wartość True, jeżeli skoroszyt nie został zmodyfikowany od czasu ostatniego zapisania na dysku. Procedura SaveAllWorkbooks2 — jej kod zamieszczamy poniżej — nie zapisuje plików, które nie muszą być zapisywane.

```
Public Sub SaveAllWorkbooks2()
    Dim Book As Workbook
    For Each Book In Workbooks
        If Book.Path <> "" Then
            If Book.Saved <> True Then
                Book.Save
            End If
        End If
    Next Book
End Sub
```

Zapisywanie i zamknięcie wszystkich skoroszytów

Poniższa procedura przy użyciu pętli przetwarza kolekcję Workbooks, zapisując i zamkając wszystkie skoroszyty:

```
Sub CloseAllWorkbooks()
    Dim Book As Workbook
```

```

For Each Book In Workbooks
    If Book.Name <> ThisWorkbook.Name Then
        Book.Close savechanges:=True
    End If
Next Book
ThisWorkbook.Close savechanges:=True
End Sub

```

Aby określić, czy dany skoroszyt to skoroszyt zawierający kod aktualnie wykonywanej procedury, nasza procedura używa instrukcji If umieszczonej w pętli For-Next. Jest to konieczne, ponieważ zamknięcie takiego skoroszytu spowoduje przerwanie wykonywania kodu, na skutek czego inne skoroszyty nie zostaną zapisane. Po zamknięciu wszystkich innych skoroszytów procedura zamyka również swój macierzysty skoroszyt.

Ukrywanie wszystkich komórek arkusza poza zaznaczonym zakresem

Procedura, którą omówimy w tym podrozdziale, ukrywa wszystkie komórki arkusza poza tymi, które znajdują się w aktualnie zaznaczonym zakresie.

```

Sub HideRowsAndColumns()
    Dim row1 As Long, row2 As Long
    Dim col1 As Long, col2 As Long

    If TypeName(Selection) <> "Range" Then Exit Sub

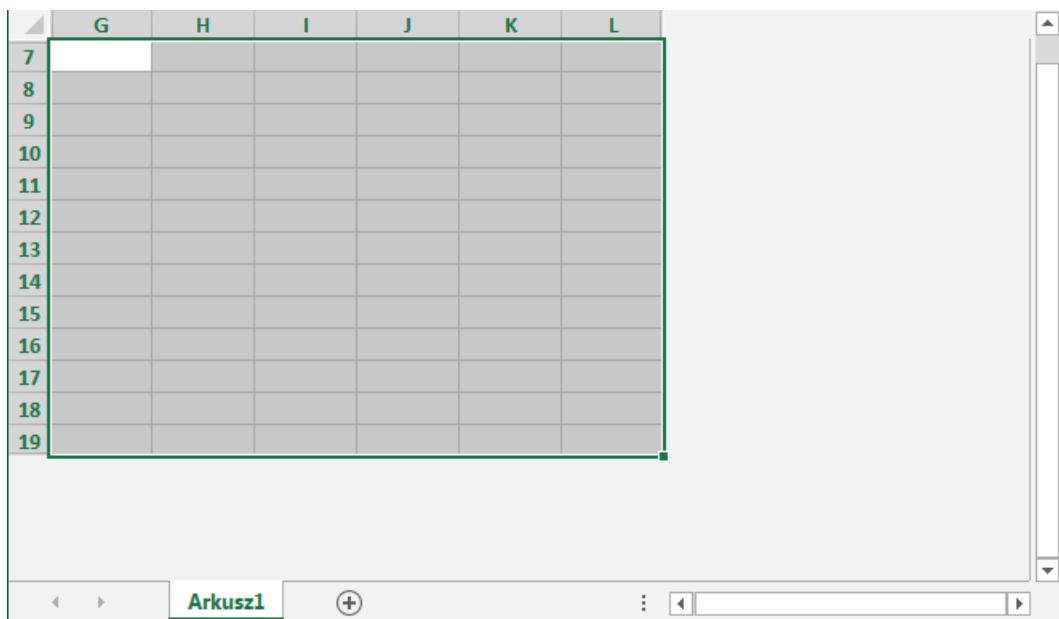
    ' Jeżeli ostatni wiersz lub kolumna są ukryte, odkryj wszystko i zakończ działanie
    If Rows(Rows.Count).EntireRow.Hidden Or
        Columns(Columns.Count).EntireColumn.Hidden Then
        Cells.EntireColumn.Hidden = False
        Cells.EntireRow.Hidden = False
        Exit Sub
    End If

    row1 = Selection.Rows(1).Row
    row2 = row1 + Selection.Rows.Count - 1
    col1 = Selection.Columns(1).Column
    col2 = col1 + Selection.Columns.Count - 1

    Application.ScreenUpdating = False
    On Error Resume Next
    ' Ukryj wiersze
    Range(Cells(1, 1).Cells(row1 - 1, 1)).EntireRow.Hidden = True
    Range(Cells(row2 + 1, 1), Cells(Rows.Count, 1)).EntireRow.Hidden = True
    ' Ukryj kolumny
    Range(Cells(1, 1).Cells(1, col1 - 1)).EntireColumn.Hidden = True
    Range(Cells(1, col2 + 1), Cells(1, Columns.Count)).EntireColumn.Hidden = True
End Sub

```

Przykład takiej sytuacji przedstawiono na rysunku 9.15. Jeżeli zaznaczony obszar arkusza składa się z kilku nieciągłych zakresów komórek, bazą do ukrywania wierszy i kolumn jest pierwszy zakres. Zwróć uwagę na fakt, że procedura działa jak przełącznik — ponowne wykonanie procedury w sytuacji, kiedy ostatni wiersz lub ostatnia kolumna są ukryte, powoduje odkrycie wszystkich wierszy i kolumn.



Rysunek 9.15. Wszystkie komórki arkusza poza zaznaczonym zakresem (G7:L19) zostały ukryte



Skoroszyt z tym przykładem (*Ukrywanie wierszy i kolumn.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Tworzenie spisu treści zawierającego hiperłącza

Procedura CreateTOC wstawia na początku aktywnego skoroszytu nowy arkusz, a następnie tworzy swego rodzaju spis treści, zawierający listę hiperłączy umożliwiających szybkie przechodzenie do poszczególnych arkuszy.

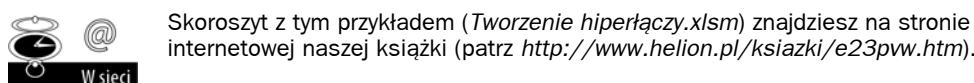
```
Sub CreateTOC()
    Dim i As Integer
    Sheets.Add Before:=Sheets(1)
    For i = 2 To Worksheets.Count
        ActiveSheet.Hyperlinks.Add _
            Anchor:=Cells(i, 1), _
            Address:="", _
            SubAddress:="'" & Worksheets(i).Name & "'!A1", _
            TextToDisplay:=Worksheets(i).Name
    Next i
End Sub
```

Utworzenie hiperłącza prowadzącego do arkusza wykresu nie jest możliwe, stąd kod procedury wykorzystuje kolekcję Worksheet zamiast kolekcji Sheets.

Na rysunku 9.16 przedstawiono przykładowy spis treści, zawierający hiperłącza prowadzące do arkuszy, których nazwy reprezentują nazwy kolejnych miesięcy.

	A	B	C	D	E	F	G	H	I
1									
2	Styczeń								
3	Luty								
4	Marzec								
5	Kwiecień								
6	Maj								
7	Czerwiec								
8	Lipiec								
9	Sierpień								
10	Wrzesień								
11	Październik								
12	Listopad								
13	Grudzień								
14									

Rysunek 9.16. Hiperłącza prowadzące do poszczególnych arkuszy, utworzone przez makro

 Skoroszyt z tym przykładem (*Tworzenie hiperłączny.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/książki/e23pvw.htm>).

Synchronizowanie arkuszy

Jeżeli korzystałeś kiedykolwiek ze skoroszytów wieloarkuszowych, to wiesz zapewne, że Excel nie potrafi synchronizować danych z poszczególnych arkuszy skoroszytu. Innymi słowy, nie ma możliwości automatycznego zaznaczenia tego samego zakresu i ustawienia górnej, lewej komórki takiego multizakresu. Poniższe makro języka VBA jako bazy używa aktywnego arkusza, a na pozostałych arkuszach skoroszytu wykonuje następujące operacje:

- zaznacza taki sam zakres, jak w przypadku aktywnego arkusza;
- ustawia taką samą górną lewą komórkę okna, jak w aktywnym arkuszu.

Oto kod źródłowy naszej procedury:

```
Sub SynchSheets()
    ' Ustawia we wszystkich arkuszach tę samą aktywną komórkę i lewą, górną komórkę zakresu
    If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
    Dim UserSheet As Worksheet, sht As Worksheet
    Dim TopRow As Long, LeftCol As Integer
    Dim UserSel As String

    Application.ScreenUpdating = False

    ' Zapamiętuje aktualny arkusz
    Set UserSheet = ActiveSheet

    ' Zapisuje informacje z aktywnego arkusza
    TopRow = ActiveWindow.ScrollRow
    LeftCol = ActiveWindow.ScrollColumn
```

```
UserSel = ActiveWindow.RangeSelection.Address

' Przechodzi w pętli przez poszczególne arkusze
For Each sht In ActiveWorkbook.Worksheets
    If sht.Visible Then ' pomija ukryte arkusze
        sht.Activate
        Range(UserSel).Select
        ActiveWindow.ScrollRow = TopRow
        ActiveWindow.ScrollColumn = LeftCol
    End If
Next sht

' Przywraca oryginalne ustawienie aktywnego arkusza
UserSheetActivate
Application.ScreenUpdating = True
End Sub
```



Skoroszyt z tym przykładem (*Synchronizacja arkuszy.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Techniki programowania w języku VBA

Następne przykłady ilustrują często używane techniki programowania w języku VBA, które możesz bardzo łatwo zaadaptować do użycia we własnych projektach.

Przełączanie wartości właściwości typu logicznego

Właściwość typu logicznego posiada wartość True lub False. Najprostsza metoda przełączania wartości takiej właściwości polega na zastosowaniu operatora Not. Zostało to pokazane w poniższym przykładzie, w którym przełączana jest wartość właściwości WrapText obiektu Selection:

```
Sub ToggleWrapText()
    ' Włącza lub wyłącza zawijanie tekstu dla zaznaczonych komórek
    If TypeName(Selection) = "Range" Then
        Selection.WrapText = Not ActiveCell.WrapText
    End If
End Sub
```

Oczywiście w razie potrzeby możesz dowolnie zmodyfikować tę procedurę, tak aby przełączała inne właściwości typu logicznego.

Zauważ, że przełączanie jest wykonywane w oparciu o aktywną komórkę. Jeżeli po zaznaczeniu zakresu właściwości komórek mają różne wartości (np. zawartość niektórych komórek jest pogrubiona, a innych nie) Excel do określenia sposobu przełączania wartości posługuje się aktywną komórką. Jeżeli na przykład zawartość aktywnej komórki jest pogrubiona, po naciśnięciu przycisku na Wstążce pogrubienie zostanie usunięte z wszystkich komórek zaznaczenia. Ta prosta procedura naśladowuje zachowanie Excela, co zazwyczaj jest najlepszym rozwiązaniem.

Zauważ również, że do sprawdzenia, czy zaznaczono zakres, procedura wykorzystuje funkcję TypeName. Jeżeli zaznaczony obszar nie jest zakresem, nie zostanie wykonana żadna operacja.

Operator Not umożliwia przełączanie wartości wielu innych właściwości. Aby na przykład w arkuszu wyświetlić lub ukryć nagłówki wierszy i kolumn, należy użyć następującej instrukcji:

```
ActiveWindow.DisplayHeadings = Not ActiveWindow.DisplayHeadings
```

Aby w aktywnym arkuszu wyświetlić lub ukryć linie siatki, należy użyć następującej instrukcji:

```
ActiveWindow.DisplayGridlines = Not ActiveWindow.DisplayGridlines
```

Wyświetlanie daty i czasu

Jeżeli poznaleś już system liczb seryjnych używany w Excelu do przechowywania dat i godzin, nie będziesz miał żadnych problemów z zastosowaniem dat i czasu w procedurach języka VBA.

Procedura DateAndTime wyświetla okno komunikatu z aktualną datą i czasem, tak jak to zostało przedstawione na rysunku 9.17. W pasku tytułowym okna wyświetlany jest komunikat odpowiedni dla danej pory dnia.

Rysunek 9.17.
Okno komunikatu
wyświetlające
datę i czas



Procedura jako argument funkcji Format wykorzystuje funkcję Date. Wynikiem działania procedury jest łańcuch zawierający sformatowaną datę. Podobnej metody użyliśmy do formatowania czasu.

```
Sub DateAndTime()
    Dim TheDate As String, TheTime As String
    Dim Greeting As String
    Dim FullName As String, FirstName As String
    Dim SpaceInName As Long

    TheDate = Format(Date, "Long date")
    TheTime = Format(Time, "Long time")

    ' Określenie powitania w oparciu o czas
    Select Case Time
        Case Is < TimeValue("12:00"): Greeting = "Dzień dobry "
        Case Is >= TimeValue("17:00"): Greeting = "Dobry wieczór "
        Case Else: Greeting = "Dzień dobry "
    End Select

```

```

    ' Dodanie do powitania imienia użytkownika
    FullName = Application.UserName
    SpaceInName = InStr(1, FullName, " ", 1)

    ' Obsługa przypadku, gdy nazwa nie zawiera spacji
    If SpaceInName = 0 Then SpaceInName = Len(FullName)
    FirstName = Left(FullName, SpaceInName)
    Greeting = Greeting & FirstName

    ' Wyświetlenie komunikatu
    MsgBox TheDate & vbCrLf & vbCrLf & "godzina: " & TheTime, vbOKOnly, Greeting
End Sub

```

W celu zagwarantowania, że makro będzie poprawnie działało niezależnie od ustawień regionalnych systemu użytkownika, w powyższym przykładzie użyliśmy nazw formatów daty i czasu (Long Date i Long Time). Oczywiście w razie potrzeby możesz posłużyć się innymi formatami. Aby na przykład wyświetlić datę w formacie *mm/dd/rr*, możesz użyć następującego polecenia:

```
TheDate = Format(Date, "mm/dd/yy")
```

Aby uzależnić treść komunikatu wyświetlanego na pasku tytułowym okna od pory dnia, zastosowaliśmy konstrukcję Select Case. Wartości związane z czasem stosowane w języku VBA funkcjonują podobnie jak w Excelu. Jeżeli wartość czasu jest mniejsza od liczby 0,5 (południe), oznacza to, że jest przedpołudnie. Jeżeli z kolei wartość ta jest większa od liczby 0,7083 (godzina 17:00), oznacza to, że jest wieczór. W innych przypadkach jest południe. Wybraliśmy proste rozwiązanie polegające na zastosowaniu funkcji *TimeValue* języka VBA, która pobierając łańcuch, zwraca wartość czasu.

Kolejna grupa instrukcji identyfikuje imię użytkownika znajdujące się w zakładce *Ogólne* okna dialogowego *Opcje*. W celu zlokalizowania pierwszej spacji zawartej w personaliach użytkownika użyłem funkcji *InStr* języka VBA. Po napisaniu procedury stwierdziłem, że nie uwzględniałem identyfikatora użytkownika, w którym nie występuje spacja. Gdy więc uruchomiłem ją w systemie używanym przez użytkownika *Nobody*, nie zadziałała prawidłowo. Jest to potwierdzeniem tezy, że nie można przewidzieć wszystkiego, i nawet najprostsze procedury mogą nie zadziałać. Nawiąsem mówiąc, jeżeli nie zostaną podane personalia użytkownika, Excel zawsze użycie nazwy aktualnie zalogowanego użytkownika. W procedurze problem ten został rozwiązany poprzez przypisanie zmiennej *SpaceInName* długości pełnej nazwy użytkownika, tak aby funkcja *Left* zwróciła odpowiednią nazwę.

Funkcja *MsgBox* łączy datę i czas, a ponadto w celu wstawienia pomiędzy nimi znaku podziału stosuje wbudowaną stałą *vbCrLf*. Stała *vbOKOnly* jest predefiniowaną stałą zwracającą zero i powodującą, że w oknie komunikatu zostanie wyświetlony jedynie przycisk *OK*. Ostatni argument o nazwie *Greeting* został wcześniej zdefiniowany w procedurze.



Skoroszyt z tym przykładem (*Data i czas.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wyświetlanie czasu w formie przyjaznej dla użytkownika

Jeżeli nie zależy Ci na 100-procentowej dokładności w określaniu czasu, może spodobać Ci się funkcja FT, której kod został przedstawiony poniżej. Funkcja FT, której nazwa pochodzi od angielskiego określenia *friendly time*, wyświetla różnicę w czasie opisowo, przy użyciu słów.

```
Function FT(t1, t2)
    Dim SDif As Double, DDif As Double

    If Not (IsDate(t1) And IsDate(t2)) Then
        FT = CVErr(xlErrValue)
        Exit Function
    End If

    DDif = Abs(t2 - t1)
    SDif = DDif * 24 * 60 * 60

    If DDif < 1 Then
        If SDif < 10 Then FT = "Teraz": Exit Function
        If SDif < 60 Then FT = SDif & " sekund temu": Exit Function
        If SDif < 120 Then FT = "minutę temu": Exit Function
        If SDif < 3600 Then FT = Round(SDif / 60, 0) & " minut temu": Exit Function
        If SDif < 7200 Then FT = "Godzinę temu": Exit Function
        If SDif < 86400 Then FT = Round(SDif / 3600, 0) & " godzin temu": Exit Function
    End If

    If DDif = 1 Then FT = "Wczoraj": Exit Function
    If DDif < 7 Then FT = Round(DDif, 0) & " dni temu": Exit Function
    If DDif < 31 Then FT = Round(DDif / 7, 0) & " tygodnie temu": Exit Function
    If DDif < 365 Then FT = Round(DDif / 30, 0) & " miesięcy temu": Exit Function
    FT = Round(DDif / 365, 0) & " lat temu"
End Function
```

Na rysunku 9.18 przedstawiono przykład wykorzystania funkcji FT w formułach arkuszowych. Jeżeli rzeczywiście będzie Ci potrzebna taka funkcjonalność Excela, to kod funkcji FT jest z pewnością bardzo wdzięcznym miejscem na wprowadzanie poprawek i ulepszeń. Na przykład możesz spróbować dopisać w nim polecenia, które będą zapobiegały wyświetlaniu takich dziwolągów językowych jak *1 miesięcy temu* czy *1 lat temu*.

Rysunek 9.18.

Zastosowanie funkcji FT do wyświetlania różnicy czasu w sposób przyjazny dla użytkownika

	A	B	C	D	E	F
1	Data 1	Data 2	Różnica w czasie			
2	2013-03-30 08:45:00	2013-03-30 08:46:26	minutę temu			
3	2013-03-30 08:45:00	2013-04-01 01:33:00	2 dni temu			
4	2013-03-30 08:45:00	2013-04-13 01:47:24	2 tygodnie temu			
5	2013-03-30 08:45:00	2013-05-01 14:20:31	1 miesięcy temu			
6	2013-03-30 08:45:00	2013-06-28 14:04:41	3 miesięcy temu			
7	2013-03-30 08:45:00	2014-01-24 11:37:48	10 miesięcy temu			
8	2013-03-30 08:45:00	2014-04-21 23:09:00	1 lat temu			
9	2013-03-30 08:45:00	2021-06-16 16:25:48	8 lat temu			
10						

← → Arkusz1 ⊕

Skoroszyt z tym przykładem (*Przyjazny czas.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Pobieranie listy czcionek

Jeżeli będziesz chciał pobrać listę wszystkich zainstalowanych czcionek, przekonasz się, że Excel nie oferuje bezpośrednio metody uzyskania takiej listy. Technika opisywana tutaj wykorzystuje fakt, że ze względu na konieczność zachowania kompatybilności z poprzednimi wersjami Excel 2013 nadal obsługuje stare metody i właściwości obiektów CommandBar, które w wersjach wcześniejszych niż 2007 były wykorzystywane do pracy z paskami narzędzi i menu.

Procedura ShowInstalledFonts wyświetla w kolumnie A aktywnego arkusza listę zainstalowanych czcionek. Aby to osiągnąć, procedura tworzy tymczasowy pasek narzędzi (obiekt klasy CommandBar), dodaje do niego formant Font i odczytuje listę czcionek z właściwością formantu. Po zakończeniu tymczasowy pasek narzędzi jest usuwany.

```
Sub ShowInstalledFonts()
    Dim FontList As CommandBarControl
    Dim TempBar As CommandBar
    Dim i As Long

    ' Tworzy tymczasowy pasek narzędzi (obiekt klasy CommandBar)
    Set TempBar = Application.CommandBars.Add
    Set FontList = TempBar.Controls.Add(ID:=1728)

    ' Umieszcza listę czcionek w kolumnie A
    Range("A:A").ClearContents
    For i = 0 To FontList.ListCount - 1
        Cells(i + 1, 1) = FontList.List(i + 1)
    Next i

    ' Usuwa tymczasowy pasek narzędzi (obiekt klasy CommandBar)
    TempBar.Delete
End Sub
```



Opcjonalnie można też wyświetlić nazwę czcionki przy użyciu tej czcionki, tak jak to zostało zaprezentowane na rysunku 9.19. Aby to zrobić, wewnątrz pętli For ... Next należy umieścić następującą instrukcję:

```
Cells(i+1, 1).Font.Name = FontList.List(i + 1)
```

Trzeba jednak mieć świadomość, że zastosowanie w skoroszycie wielu czcionek spowoduje zużycie znacznej ilości zasobów systemowych, a nawet może doprowadzić do zawieszenia komputera.



Skoroszyt z tym przykładem (*Lista czcionek.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Sortowanie tablicy

Co prawda sam Excel posiada wbudowane polecenie sortujące zakresy arkusza, ale język VBA nie dysponuje metodą sortowania tablic. Skuteczne, ale niewygodne rozwiązanie tego problemu polega na przeniesieniu zawartości tablicy do zakresu arkusza, posortowaniu jej przy użyciu polecenia Excela, a następnie wczytaniu wyniku do tablicy. Jeżeli jednak szybkość odgrywa dużą rolę, lepiej stworzyć w języku VBA procedurę sortującą.

Rysunek 9.19.

Wyświetlanie listy czcionek przy użyciu odpowiedniego kroju czcionek

A	B
101 Gill Sans MT	
102 Gill Sans MT Condensed	
103 Gill Sans MT Ext Condensed Bold	
104 Gill Sans Ultra Bold	
105 Gill Sans Ultra Bold Condensed	
106 Gisha	
107 Gloucester MT Extra Condensed	
108 Goudy Old Style	
109 GOUZY STOUT	
110 Gulim	
111 Gui I mChe	
112 Gungsuh	
113 GungsuhChe	
114 Haettenschweiler	
115 Harlow Solid Italic	
116 Harrington	
117 High Tower Text	
118 Impact	
119 Imprint MT Shadow	

Arkusz1



W tym punkcie omówimy cztery różne metody sortowania:

- *Sortowanie arkuszowe* polega na przeniesieniu zawartości tablicy do zakresu arkusza, posortowaniu jej, a następnie ponownym umieszczeniu w tablicy. Jedynym argumentem procedury opartej na tej metodzie jest tablica.
- *Sortowanie bąbelkowe* jest prostą metodą sortowania (zastosowano ją też w przykładzie demonstrującym sortowanie arkusza w rozdziale 7.). Co prawda metoda sortowania bąbelkowego jest łatwa w kodowaniu, ale ma raczej powolny algorytm, zwłaszcza gdy przetwarzaniu podlega duża liczba elementów.
- *Sortowanie metodą quick-sort* (sortowanie szybkie) w porównaniu z bąbelkowym jest o wiele szybszą metodą sortowania, ale też znacznie trudniejszą do zrozumienia. Metoda może zostać wykorzystana tylko w przypadku takich typów danych, jak Integer lub Long.
- *Sortowanie zliczające* jest bardzo szybkie, ale również trudne do zrozumienia. Technika ta, podobnie jak sortowanie szybkie, działa tylko w przypadku takich typów danych, jak Integer lub Long.

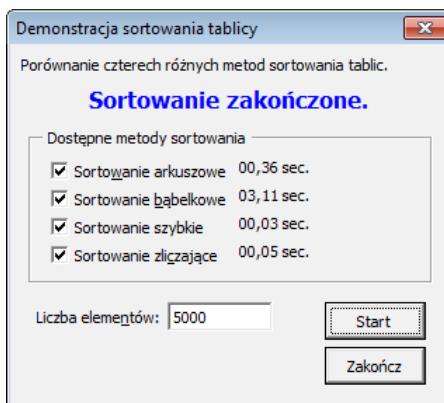


W sieci

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt o nazwie *Sortowanie.xlsxm*, który porównuje wyżej wymienione metody sortowania. Skoroszyt przydaje się w przypadku porównywania metod sortowania tablic o różnych rozmiarach. Oczywiście w razie potrzeby możesz skopiować z niego odpowiednie procedury i użyć ich w swoich programach.

Na rysunku 9.20 pokazano okno dialogowe naszego programu. Procedury sortujące zostały przetestowane przy użyciu tablic o siedmiu różnych rozmiarach liczących od 500 do 100 000 elementów. W tablicach zawarte były wartości losowe typu Long.

Rysunek 9.20.
Porównanie czasu potrzebnego do wykonania operacji sortowania tablic o różnych rozmiarach



W tabeli 9.1 zawarłem wyniki testów. Wartość 0,00 oznacza, że sortowanie zostało zakończone prawie natychmiast (w czasie krótszym niż 0,01 sekundy).

Tabela 9.1. Czas trwania (wyrażony w sekundach) operacji sortowania tablic wypełnionych losowymi wartościami przy użyciu czterech algorytmów sortujących

Liczba elementów tablicy	Sortowanie arkuszowe Excela	Sortowanie bąbelkowe przy użyciu języka VBA	Sortowanie szybkie przy użyciu języka VBA	Sortowanie zliczające przy użyciu języka VBA
500	0,01	0,01	0,00	0,00
1000	0,02	0,05	0,00	0,01
5000	0,05	1,20	0,00	0,01
10 000	0,07	4,82	0,01	0,01
25 000	0,16	29,19	0,04	0,01
50 000	0,31	112,05	0,07	0,02
100 000	0,63	418,38	0,17	0,04

Algorytm sortowania arkuszowego jest wyjątkowo szybki, zwłaszcza że operacja uwzględnia przeniesienie zawartości tablicy do arkusza, sortowanie jej i ponowne wczytanie danych do tablicy. Jeżeli tablica jest już prawie posortowana, sortowanie arkuszowe będzie jeszcze szybsze.

Algorytm sortowania bąbelkowego jest dość szybki w przypadku niewielkich tablic, ale przy większych (liczących ponad 5000 elementów) powinieneś po prostu o nim zapomnieć. Sortowanie szybkie oraz sortowanie zliczające są bardzo szybkie, ale ich funkcjonalność jest ograniczona jedynie do danych typu Integer oraz Long.

Przetwarzanie grupy plików

Jednym z częstych zastosowań makr jest oczywiście kilkakrotne powtarzanie określonej operacji. Przykład przedstawiony poniżej ilustruje, jak przy użyciu makra przetworzyć kilka różnych plików zapisanych na dysku. Procedura, która może pomóc w napisaniu

własnego makra realizującego tego typu zadanie, prosi użytkownika o podanie wzorca nazw plików, a następnie przetwarza wszystkie pliki, których nazwy są z nim zgodne. W tym przypadku operacja przetwarzania polega na zimportowaniu pliku i wprowadzeniu grupy formuł sumujących, które opisują zawarte w nim dane.

```
Sub BatchProcess()
    Dim FileSpec As String
    Dim i As Integer
    Dim FileName As String
    Dim FileList() As String
    Dim FoundFiles As Integer

    ' Określenie ścieżki i wzorca nazwy
    FileSpec = ThisWorkbook.Path & "\" & "text???.txt"
    FileName = Dir(FileSpec)

    ' Czy plik został znaleziony?
    If FileName <> "" Then
        FoundFiles = 1
        ReDim Preserve FileList(1 To FoundFiles)
        FileList(FoundFiles) = FileName
    Else
        MsgBox "Nie znaleziono żadnych plików pasujących do wzorca " & FileSpec
        Exit Sub
    End If

    ' Pobierz nazwy pozostałych plików
    Do
        FileName = Dir
        If FileName = "" Then Exit Do
        FoundFiles = FoundFiles + 1
        ReDim Preserve FileList(1 To FoundFiles)
        FileList(FoundFiles) = FileName & "*"
    Loop

    ' Przetwarzanie kolejnych plików w pętli
    For i = 1 To FoundFiles
        Call ProcessFiles(FileList(i))
    Next i
End Sub
```



Skoroszyt z tym przykładem (*Przetwarzanie wsadowe.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Przykład korzysta z trzech dodatkowych plików, również znajdujących się na dysku CD-ROM. Są to: *text01.txt*, *text02.txt* i *text03.txt*. Aby zaimportować inne pliki tekstowe, będziesz musiał odpowiednio zmodyfikować kod procedury.

Nazwy plików pasujące do wzorca są przechowywane w tablicy o nazwie *FoundFiles*. Pliki są przetwarzane przy użyciu pętli *For ... Next*. W trakcie przetwarzania wewnętrz pętli jest wywoływana prosta procedura *ProcessFiles*. W celu zaimportowania pliku korzysta ona z metody *OpenText*, a następnie wstawia pięć formuł. Oczywiście zamiast poniższej można zastosować własną procedurę.

```
Sub ProcessFiles(FileName As String)
    ' Importowanie pliku
    Workbooks.OpenText FileName:=FileName, _
        Origin:=xlWindows, _
```

```

    StartRow:=1,
    DataType:=xlFixedWidth,
    FieldInfo:=
        Array(Array(0, 1), Array(3, 1), Array(12, 1))
    ' Wprowadzanie formuł podsumowujących
    Range("D1").Value = "A"
    Range("D2").Value = "B"
    Range("D3").Value = "C"
    Range("E1:E3").Formula = "=COUNTIF(B:B,D1)"
    Range("F1:F3").Formula = "=SUMIF(B:B,D1,C:C)"
End Sub

```



Więcej szczegółowych informacji na temat pracy z plikami z poziomu języka VBA znajdziesz w rozdziale 25.

Ciekawe funkcje, których możesz użyć w swoich projektach

W tym podrozdziale zaprezentujemy kilka niestandardowych funkcji, które można albo bezpośrednio stosować w aplikacjach użytkowych, albo modyfikować, traktując je jako twórczą inspirację. Najprzydatniejsze są wtedy, gdy wywołuje się je z innej procedury języka VBA. Procedury zostały zadeklarowane przy użyciu słowa kluczowego `Private`, dzięki czemu nie będą widoczne w oknie dialogowym *Wstawianie funkcji* Excela.



Skoroszyt z tym przykładem (*Funkcje użytkowe VBA.xls*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Funkcja FileExists

Funkcja pobiera jeden argument (ścieżka pliku wraz z jego nazwą) i zwraca wartość `True`, jeżeli plik istnieje:

```

Private Function FileExists(fname) As Boolean
    ' Zwraca wartość TRUE, jeżeli plik istnieje
    FileExists = (Dir(fname) <> "")
End Function

```

Funkcja FileNameOnly

Funkcja pobiera jeden argument (ścieżka pliku wraz z jego nazwą) i zwraca tylko nazwę pliku (innymi słowy — usuwa ścieżkę pliku):

```

Private Function FileNameOnly(pname) As String
    ' Zwraca nazwę pliku pobraną z łańcucha złożonego ze ścieżki i nazwy pliku
    Dim temp As Variant
    length = Len(pname)
    temp = Split(pname, Application.PathSeparator)
    FileNameOnly = temp(UBound(temp))
End Function

```

Funkcja `FileNameOnly` wykorzystuje funkcję `Split` VBA, która pobiera łańcuch tekstu (oraz separator) i zwraca tabelę typu `Variant`, zawierającą elementy łańcucha znajdujące się pomiędzy znakami separatora. W tym przypadku zmienna `temp` zawiera tablicę z łańcuchami tekstu znajdującymi się pomiędzy separatorami definiowanymi przez `Application.PathSeparator` (zaznaczaj są to znaki lewego ukośnika). Inny przykład zastosowania funkcji `Split` znajdziesz w podrozdziale „Wyznaczanie n-tego elementu łańcucha” w dalszej części tego rozdziału.

Jeżeli argumentem wywołania funkcji jest ścieżka `c:\excel files\2013\backup\budżet.xlsxm`, funkcja zwróci łańcuch `budżet.xlsxm`.

Funkcja `FileNameOnly` przetwarza dowolną ścieżkę i nazwę pliku (nawet jeżeli plik nie istnieje). Jeżeli plik istnieje, poniższa funkcja oferuje prostszą metodę usuwania ścieżki i zwracania tylko nazwy pliku:

```
Private Function FileNameOnly2(pname) As String
    FileNameOnly2 = Dir(pname)
End Function
```

Funkcja `PathExists`

Funkcja pobiera jeden argument (ścieżka pliku) i zwraca wartość `True`, jeżeli ścieżka istnieje:

```
Private Function PathExists(pname) As Boolean
    ' Zwraca wartość True, jeżeli istnieje ścieżka
    If Dir(pname, vbDirectory) = "" Then
        PathExists = False
    Else
        PathExists = (GetAttr(pname) And vbDirectory) = vbDirectory
    End If
End Function
```

Funkcja `RangeNameExists`

Funkcja pobiera jeden argument (nazwa zakresu) i zwraca wartość `True`, jeżeli w aktywnym skoroszycie istnieje zakres o takiej nazwie:

```
Private Function RangeNameExists(nname) As Boolean
    ' Zwraca wartość True, jeżeli istnieje nazwa zakresu
    Dim n As Name
    RangeNameExists = False
    For Each n In ActiveWorkbook.Names
        If UCASE(n.Name) = UCASE(nname) Then
            RangeNameExists = True
            Exit Function
        End If
    Next n
End Function
```

Inny sposób napisania takiej funkcji przedstawiono poniżej. Funkcja w tej wersji próbuje utworzyć zmienną obiektową przy użyciu nazwy zakresu. Jeżeli taka próba zakończy się wygenerowaniem błędu, oznacza to, że dana nazwa zakresu nie istnieje.

```

Private Function RangeNameExists2(nname) As Boolean
    ' Zwraca wartość TRUE, jeżeli istnieje zakres o takiej nazwie
    Dim n As Range
    On Error Resume Next
    Set n = Range(nname)
    If Err.Number = 0 Then RangeNameExists2 = True _
        Else RangeNameExists2 = False
End Function

```

Funkcja SheetExists

Funkcja pobiera jeden argument (nazwa arkusza) i zwraca wartość True, jeżeli w aktywnym skoroszycie istnieje arkusz o takiej nazwie:

```

Private Function SheetExists(sname) As Boolean
    ' Zwraca wartość True, jeżeli w aktywnym skoroszycie istnieje arkusz o takiej nazwie
    Dim x As Object
    On Error Resume Next
    Set x = ActiveWorkbook.Sheets(sname)
    If Err = 0 Then SheetExists = True Else SheetExists = False
End Function

```

Sprawdzanie, czy dany obiekt należy do kolekcji

Poniższa procedura Function jest prostą funkcją sprawdzającą, czy dany obiekt należy do kolekcji:

```

Private Function IsInCollection
    (Coln As Object, Item As String) As Boolean
    Dim Obj As Object
    On Error Resume Next
    Set Obj = Coln(Item)
    IsInCollection = Not Obj Is Nothing
End Function

```

Funkcja pobiera dwa argumenty — kolekcję (obiekt) i element (łańcuch), który może, ale nie musi należeć do kolekcji. Funkcja próbuje utworzyć zmienną obiektową reprezentującą element kolekcji. Jeżeli próba się powiedzie, funkcja zwraca wartość True. W przeciwnym razie funkcja zwraca wartość False.

Funkcji IsInCollection można użyć zamiast trzech innych funkcji wymienionych w rozdziale (RangeNameExists, SheetExists i WorkbookIsOpen). Aby na przykład stwierdzić, czy w aktywnym skoroszycie istnieje zakres o nazwie Data, należy wywołać funkcję IsInCollection przy użyciu poniższej instrukcji:

```
MsgBox IsInCollection(ActiveWorkbook.Names, "Data")
```

Aby stwierdzić, czy otwarto skoroszyt o nazwie *budżet.xlsx*, należy użyć następującej instrukcji:

```
MsgBox IsInCollection(Workbooks, "budżet.xlsx")
```

Aby stwierdzić, czy aktywny skoroszyt zawiera arkusz o nazwie Arkusz1, należy użyć następującej instrukcji:

```
MsgBox IsInCollection(ActiveWorkbook.Worksheets, "Arkusz1")
```

Funkcja WorkbookIsOpen

Funkcja pobiera jeden argument (nazwa skoroszytu) i zwraca wartość True, jeżeli skoroszyt jest otwarty:

```
Private Function WorkbookIsOpen(wbname) As Boolean
    ' Zwraca wartość TRUE, jeżeli skoroszyt jest otwarty
    Dim x As Workbook
    On Error Resume Next
    Set x = Workbooks(wbname)
    If Err = 0 Then WorkbookIsOpen = True _
        Else WorkbookIsOpen = False
End Function
```

Pobieranie wartości z zamkniętego skoroszytu

Język VBA nie posiada metody umożliwiającej pobranie wartości z zamkniętego skoroszytu. W razie potrzeby możemy jednak skorzystać z faktu, że Excel obsługuje łącza do plików. Zamieszczona poniżej funkcja GetValue języka VBA pobiera wartość z zamkniętego skoroszytu. Zadanie to jest realizowane poprzez wywołanie starszego typu *makra XLM*, które było stosowane w wersjach Excela sprzed wersji 5. Na szczęście, jak widać, Excel nadal obsługuje makra tego starego typu.

```
Private Function GetValue(path, file, sheet, ref)
    ' Pobiera wartość z zamkniętego skoroszytu
    Dim arg As String

    ' Sprawdza, czy istnieje plik
    If Right(path, 1) <> "\" Then path = path & "\"
    If Dir(path & file) = "" Then
        GetValue = "Plik nie został znaleziony."
        Exit Function
    End If

    ' Tworzenie argumentu
    arg = "" & path & "[" & file & "]" & sheet & "!" & _
        Range(ref).Range("A1").Address(, , xlR1C1)

    ' Wykonanie makra XLM
    GetValue = ExecuteExcel4Macro(arg)
End Function
```

Funkcja GetValue pobiera cztery argumenty:

- path — ścieżka zamkniętego pliku (np. "d:\pliki");
- file — nazwa pliku skoroszytu (np. "budżet.xlsx");
- sheet — nazwa arkusza (np. "Arkusz1");
- ref — odwołanie do komórki (np. "C4").

Poniższa procedura Sub demonstruje, w jaki sposób użyć funkcji GetValue. Procedura wyświetla po prostu wartość w komórce A1 arkusza Arkusz1 pliku o nazwie 2013Budżet.xlsx znajdującego się w katalogu XLPliki\Budżet na dysku C.

```
Sub TestGetValue()
    Dim p As String, f As String
    Dim s As String, a As String

    p = "c:\XLPliki\Budżet"
    f = "2013Budżet.xlsx"
    s = "Arkusz1"
    a = "A1"
    MsgBox GetValue(p, f, s, a)
End Sub
```

Kolejna procedura odczytuje z zamkniętego pliku 1200 wartości (zajmujących obszar 100 wierszy×12 kolumn), a następnie umieszcza je w aktywnym arkuszu:

```
Sub TestGetValue2()
    Dim p As String, f As String
    Dim s As String, a As String
    Dim r As Long, c As Long

    p = "c:\XLPliki\Budżet"
    f = "2013Budżet.xlsx"
    s = "Arkusz1"
    Application.ScreenUpdating = False
    For r = 1 To 100
        For c = 1 To 12
            a = Cells(r, c).Address
            Cells(r, c) = GetValue(p, f, s, a)
        Next c
    Next r
End Sub
```

Alternatywnym rozwiązaniem może być napisanie kodu, który będzie wyłączał aktualizację ekranu, otwierał plik, pobierał odpowiednią wartość i zamykał plik. Jeżeli taki plik nie będzie zbyt duży, to użytkownik nie powinien nawet zauważyc tej operacji.



Funkcja GetValue nie zadziała po umieszczeniu jej w formule arkuszowej. W praktyce nie ma jednak żadnej potrzeby umieszczania tej funkcji w jakiejkolwiek formule. W celu pobrania wartości z zamkniętego pliku wystarczy stworzyć łączce do komórki znajdującej się w takim pliku.



Skoroszyt z tym przykładem (*Pobieranie wartości z zamkniętego pliku.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Procedury zawarte w tym skoroszycie pobierają dane ze skoroszytu o nazwie *Mój skoroszyt.xlsx*.

Użyteczne, niestandardowe funkcje arkuszowe

W tym podrozdziale zamieściliśmy przykłady niestandardowych funkcji, które możesz zastosować w formułach arkusza. Pamiętaj, że takie funkcje (procedury typu Function), muszą zostać zdefiniowane w module VBA, a nie w modułach powiązanych z obiektami takimi jak *ThisWorkbook*, *Arkusz1* lub *UserForm1*.



Skoroszyt z przykładami omawianymi w tym podrozdziale (*Funkcje arkuszowe.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Funkcje zwracające informacje o formatowaniu komórki

W tym podrozdziale znajdziesz szereg niestandardowych funkcji zwracających różnorodne informacje o sposobie formatowania komórki. Funkcje takie są przydatne na przykład w sytuacji, kiedy trzeba posortować dane w oparciu o formatowanie (np. gdy szukasz wszystkich komórek, wobec których użyto pogrubienia).



Takie funkcje nie zawsze są automatycznie przeliczane, ponieważ zmiana formatowania nie uaktywnia mechanizmu Excela wykonującego ponowne obliczenia. Aby wymusić ponowne obliczenie wszystkich skoroszytów wraz z aktualizacją wartości funkcji niestandardowych, należy nacisnąć kombinację klawiszy **Ctrl+Alt+F9**.

Innym rozwiązaniem jest umieszczenie w kodzie źródłowym funkcji następującego polecenia:

```
Application.Volatile
```

Po zastosowaniu tego polecenia naciśnięcie klawisza **F9** spowoduje ponowne przeliczenie arkusza wraz z funkcjami niestandardowymi.

Poniższa funkcja zwraca wartość **TRUE**, jeżeli w jednokomórkowym zakresie będącym jej argumentem zastosowano pogrubienie. Jeżeli jako argument wywołania funkcji zostanie przekazany zakres, funkcja użyje lewej, górnej komórki tego zakresu:

```
Function IsBold(cell) As Boolean
    ' Zwraca wartość TRUE, jeżeli zawartość komórki została pogrubiona
    IsBold = cell.Range("A1").Font.Bold
End Function
```

Pamiętaj, że takie funkcje działają poprawnie tylko z komórkami, którym formatowanie zostało nadane bezpośrednio za pomocą polecen i przycisków formatujących i nie będą działały w przypadku formatowania nadanego za pomocą mechanizmu formatowania warunkowego. W Excelu 2010 wprowadzony został nowy obiekt, **DisplayFormat**, który potrafi obsługiwać elementy formatowania warunkowego. Poniżej zamieszczamy nową wersję funkcji **ISBOLD** — zwraca ona poprawne wyniki również dla komórek, których zawartość została pogrubiona za pośrednictwem formatowania warunkowego:

```
Function ISBOLD(cell) As Boolean
    ' Zwraca wartość TRUE, jeżeli zawartość komórki została pogrubiona (z uwzględnieniem formatowania warunkowego)
    ISBOLD = cell.Range("A1").DisplayFormat.Font.Bold
End Function
```

Poniższa funkcja zwraca wartość **True**, jeżeli w komórce będącej jej argumentem zastosowano kursywę:

```
Function ISITALIC(cell) As Boolean
    ' Zwraca wartość True, jeżeli w komórce użyto kursyw
    ISITALIC = cell.Range("A1").Font.Italic
End Function
```

Jeżeli w komórce zostanie zastosowane mieszane formatowanie (np. tylko *niektóre* znaki zostaną pogrubione), obie powyższe funkcje zwrócią błąd. Kolejna funkcja zwróci wartość **TRUE** tylko wtedy, gdy wszystkie znaki z komórki będą pogrubione:

```

Function ALLBOLD(cell) As Boolean
    ' Zwraca wartość True, jeżeli wszystkie znaki z komórki są pogrubione
    If IsNull(cell.Font.Bold) Then
        ALLBOLD = False
    Else
        ALLBOLD = cell.Font.Bold
    End If
End Function

```

Funkcja ALLBOLD może zostać uproszczona do następującej postaci:

```

Function ALLBOLD (cell) As Boolean
    ' Zwraca wartość TRUE, jeżeli wszystkie znaki z komórki są pogrubione
    ALLBOLD = Not IsNull(cell.Font.Bold)
End Function

```

Funkcja FILLCOLOR zwraca liczbę całkowitą odpowiadającą indeksowi koloru tła komórki (czyli inaczej mówiąc, indeksowi koloru jej wypełnienia). Kolor tła komórki zwykle zależy od wybranego stylu formatowania arkusza. Jeżeli tło komórki nie zostanie wypełnione, funkcja zwraca wartość -4142. Ta funkcja nie działa poprawnie z kolorami tła tabel (utworzonych za pomocą polecenia *WSTAWIANIE/Tabele/Tabela*) ani z tabelami przestawnymi. Aby uwzględnić kolory tła tabel, powinieneś użyć obiektu *DisplayFormat*, o którym wspominaliśmy wcześniej.

```

Function FILLCOLOR(cell) As Integer
    ' Zwraca liczbę całkowitą odpowiadającą kolorowi tła komórki
    FILLCOLOR = cell.Range("A1").Interior.ColorIndex
End Function

```

Gadający arkusz?

Funkcja SAYIT wykorzystuje wbudowany w Excela generator mowy do odczytywania „na głos” przekazanego jej argumentu (którym może być łańcuch tekstu lub odwołanie do wybranej komórki).

```

Function SAYIT(txt)
    Application.Speech.Speak (txt)
    SAYIT = txt
End Function

```

Opisywana funkcja daje całkiem interesujące efekty i może być naprawdę użytkownicza. Spróbuj użyć tej funkcji na przykład w następującej formule:

```
=JEŻELI(SUMA(A:A)>25000,SAYIT("Cel został osiągnięty!"))
```

Jeżeli suma wartości w kolumnie A przekroczy wartość 25000, usłyszysz zsyntyzowany głos radośnie oznajmujący, że cel został osiągnięty. Metody Speak możesz również użyć do powiadomienia o zakończeniu dugo działającej procedury. W ten sposób po uruchomieniu procedury będziesz mógł zająć się czymś innym, a Excel sam powiadomi Cię, kiedy procedura wreszcie zakończy działanie.

Wyświetlanie daty zapisania lub wydrukowania pliku

Skoroszyt Excela posiada szereg wbudowanych właściwości dokumentu, które są dostępne dla programów VBA za pośrednictwem właściwości `BuiltinDocumentProperties` obiektu `Workbook`. Poniższa funkcja zwraca datę i czas wykonania ostatniej operacji zapisu skoroszytu:

```
Function LASTSAVED()
    Application.Volatile
    LASTSAVED = ThisWorkbook.  
        BuiltinDocumentProperties("Last Save Time")
End Function
```

Data i czas ostatniego zapisu, zwracane przez tę funkcję, to dokładnie ta sama informacja, jaką znajdziesz w sekcji *Powiązane daty* widoku *Backstage*, dostępnego po przejściu na kartę *PLIK* i wybraniu polecenia *Informacje*. Pamiętaj, że mechanizm automatycznego zapisu również modyfikuje te dane, stąd znacznik czasu ostatniego zapisu dokumentu niekiedy musi być tożsamy z tym, kiedy dany dokument został zapisany *przez użytkownika*.

Poniższa funkcja jest podobna do poprzedniej, z tym że zwraca datę i czas wykonania ostatniej operacji drukowania lub podglądu wydruku dla skoroszytu. Jeżeli skoroszyt nie był jeszcze nigdy drukowany ani użytkownik nigdy nie korzystał z opcji podglądu wydruku, funkcja zwraca błąd #ARG!:

```
Function LASTPRINTED()
    Application.Volatile
    LASTPRINTED = ThisWorkbook.  
        BuiltinDocumentProperties("Last Print Date")
End Function
```

Jeżeli użyjesz tych funkcji w formule, to w celu uzyskania aktualnych wartości właściwości skoroszytu może być konieczne wymuszenie wykonania ponownych obliczeń (poprzez wcisnięcie klawisza *F9*).



Istnieje całkiem sporo dodatkowych wbudowanych właściwości, ale Excel nie korzysta ze wszystkich. Na przykład próba użycia właściwości `Number of Bytes` spowoduje wygenerowanie błędu. Pełną listę wbudowanych właściwości skoroszytów znajdziesz w pomocy systemowej programu Excel.

Funkcje `LASTSAVED` i `LASTPRINTED` zostały zaprojektowane z myślą o przechowywaniu ich w skoroszycie, w którym są używane. W niektórych przypadkach funkcja może zostać umieszczone w innym skoroszycie (np. *personal.xlsb*) lub dodatku. Ponieważ obie powyższe funkcje odwołują się do właściwości `ThisWorkbook`, po zapisaniu ich w innym skoroszycie nie będą działały poprawnie. Poniżej zawarto wersje tych funkcji o bardziej uniwersalnym przeznaczeniu. Funkcje używają właściwości `Application.Caller`, która zwraca obiekt klasy `Range` reprezentujący komórkę wywołującą funkcję. Właściwość `Parent.Parent` zwraca skoroszyt (obiekt `Workbook`), czyli obiekt nadrzędny przodka obiektu `Range`. Zagadnienie to zostanie omówione dokładniej w następnym podrozdziale.

```
Function LASTSAVED2()
    Application.Volatile
    LASTSAVED2 = Application.Caller.Parent.Parent.
        BuiltInDocumentProperties("Last Save Time") -
    End Function
```

Obiekty nadrzędne

Jak pamiętasz, model obiektowy Excela ma postać hierarchiczną — poszczególne obiekty są zawarte w innych obiektach. Na szczytce hierarchii Excela znajduje się obiekt klasy Application. Excel zawiera inne obiekty, które są kontenerami dla kolejnych obiektów, itd. Poniższa hierarchia ilustruje miejsce, jakie w tym schemacie zajmuje obiekt Range:

```
obiekt Application
    obiekt Workbook
        obiekt Worksheet
            obiekt Range
```

W języku programowania obiektowym mówimy, że przodkiem (obiektem nadzewnętrznym) obiektu Range jest obiekt Worksheet, będący jego kontenerem. Obiektem nadzewnętrznym (przodkiem) obiektu Worksheet jest obiekt Workbook przechowujący arkusz. Z kolei obiektem nadzewnętrznym obiektu Workbook jest obiekt Application.

W jaki sposób wykorzystać takie informację w praktyce? Aby się o tym przekonać, przeanalizujmy funkcję SHEETNAME języka VBA. Funkcja pobiera jeden argument (zakres) i zwraca nazwę arkusza zawierającego zakres. Funkcja używa właściwości Parent obiektu Range. Właściwość Parent zwraca obiekt przechowujący obiekt Range.

```
Function SHEETNAME(ref) As String
    SHEETNAME = ref.Parent.Name
End Function
```

Kolejna funkcja WORKBOOKNAME zwraca nazwę skoroszytu zawierającego określoną komórkę. Zauważ, że funkcja dwukrotnie używa właściwości Parent. Pierwsza właściwość Parent zwraca obiekt Worksheet, a druga — obiekt Workbook.

```
Function WORKBOOKNAME(ref) As String
    WORKBOOKNAME = ref.Parent.Parent.Name
End Function
```

Poniższa funkcja APPNAME przenosi nas na kolejny poziom w hierarchii, trzykrotnie korzystając z właściwości Parent. Funkcja zwraca nazwę obiektu Application powiązanego z określoną komórką. Oczywiście w naszym przypadku funkcja zawsze będzie zwracała wartość Microsoft Excel.

```
Function APPNAME(ref) As String
    APPNAME = ref.Parent.Parent.Parent.Name
End Function
```

Zliczanie komórek, których wartości zawierają się pomiędzy dwoma wartościami

Poniższa funkcja o nazwie COUNTBETWEEN zwraca liczbę wartości w zakresie (pierwszy argument), mieszczących się pomiędzy dwoma wartościami reprezentowanymi przez drugi i trzeci argument:

```
Function COUNTBETWEEN(InRange, num1, num2) As Long
    ' Zlicza wartości z przedziału od num1 do num2
    With Application.WorksheetFunction
        If num1 <= num2 Then
            COUNTBETWEEN = .CountIfs(InRange, ">=" & num1, _
                InRange, "<=" & num2)
        Else
            COUNTBETWEEN = .CountIfs(InRange, ">=" & num2, _
                InRange, "<=" & num1)
        End If
    End With
End Function
```

Funkcja korzysta z funkcji arkuszowej COUNTIFS (LICZ.WARUNKI) Excela i w praktyce spełnia rolę funkcji osłonowej upraszczającej tworzone formuły.



Funkcja LICZ.WARUNKI została wprowadzona w Excelu 2007, stąd taka funkcja nie będzie działała z wcześniejszymi wersjami Excela.

Poniżej zamieszczono przykład formuły korzystającej z funkcji COUNTBETWEEN, zwracającej liczbę komórek zakresu A1:A100, których wartości są większe lub równe 10 i mniejsze lub równe 20:

=COUNTBETWEEN(A1:A100, 10, 20)

Funkcja pobiera dwa argumenty numeryczne w dowolnej kolejności, stąd formuła przedstawiona poniżej działa dokładnie tak samo, jak jej poprzedniczka:

=COUNTBETWEEN(A1:A100, 20, 10)

Zastosowanie tej funkcji języka VBA jest zdecydowanie prostsze niż wprowadzenie następującej długiej (i jak widać — dosyć złożonej) formuły:

=LICZ.WARUNKI(A1:A100, ">=10", A1:A100"<=20")

Warto tutaj jednak zauważyć, że formuła arkuszowa działa znacznie szybciej.

Wyznaczanie ostatniej niepustej komórki kolumny lub wiersza

W tym podrozdziale zaprezentowano dwie bardzo przydatne funkcje. Pierwsza z nich, o nazwie LASTINCOLUMN, zwraca zawartość ostatniej niepustej komórki danej kolumny. Druga funkcja, LASTINROW, zwraca zawartość ostatniej niepustej komórki danego wiersza. Każda z funkcji pobiera pojedynczy argument, którym jest zakres. Zakresem może być cała kolumna (funkcja LASTINCOLUMN) lub cały wiersz (funkcja LASTINROW). Jeżeli przekazany argument nie jest całą kolumną lub wierszem, funkcja użyje wiersza lub kolumny

określonej przez górną lewą komórkę zakresu. Poniższa przykładowa formuła zwraca wartość ostatniej, niepustej komórki kolumny B:

```
=LASTINCOLUMN(B5)
```

Kolejna formuła zwraca wartość ostatniej niepustej komórki z wiersza 7:

```
=LASTINROW(C7:D9)
```

Oto kod źródłowy funkcji LASTINCOLUMN:

```
Function LASTINCOLUMN(rng As Range)
    ' Zwraca zawartość ostatniej niepustej komórki kolumny
    Dim LastCell As Range
    Application.Volatile
    With rng.Parent
        With .Cells(.Rows.Count, rng.Column)
            If Not IsEmpty(.Value) Then
                LASTINCOLUMN = .Value
            ElseIf IsEmpty(.End(xlUp)) Then
                LASTINCOLUMN = ""
            Else
                LASTINCOLUMN = .End(xlUp).Value
            End If
        End With
    End With
End Function
```

Funkcja jest dosyć złożona, dlatego poniżej przedstawiamy kilka punktów, które mogą pomóc Ci zrozumieć jej sposób działania:

- Metoda Application.Volatile powoduje, że funkcja zostanie wykonana każdorazowo przy obliczaniu arkusza.
- Właściwość Rows.Count zwraca liczbę wierszy arkusza. Zamiast na sztywno wprowadzać w kodzie źródłowym liczbę wierszy arkusza, użyłem właściwości Count, ponieważ kolejna wersja Excela może obsługiwać jeszcze większą niż dotychczas liczbę wierszy.
- Właściwość rng.Column zwraca numer kolumny górnej lewej komórki zakresu będącego wartością argumentu rng.
- Zastosowanie właściwości rng.Parent powoduje, że funkcja będzie działać poprawnie nawet wtedy, gdy argument rng odwołuje się do innego arkusza lub skoroszytu.
- Użycie metody End z argumentem xlUp jest równoznaczne z uaktywnieniem ostatniej komórki kolumny, wcisnięciem klawisza *End*, a następnie klawisza ↑.
- Funkcja IsEmpty sprawdza, czy komórka jest pusta. Jeżeli tak jest, zwraca pusty łańcuch. Gdyby funkcja IsEmpty nie została zastosowana, po napotkaniu pustej komórki nasza funkcja zwróciłaby wartość 0.

Poniżej przedstawiamy kod źródłowy funkcji LASTINROW, która jest bardzo podobna do funkcji LASTINCOLUMN:

```

Function LASTINROW(rng As Range)
    ' Zwraca zawartość ostatniej niepustej komórki wiersza
    Application.Volatile
    With rng.Parent
        With .Cells(rng.Row, .Columns.Count)
            If Not IsEmpty(.Value) Then
                LASTINROW = .Value
            ElseIf IsEmpty(.End(xlToLeft)) Then
                LASTINROW = ""
            Else
                LASTINROW = .End(xlToLeft).Value
            End If
        End With
    End With
End Function

```

Czy dany łańcuch tekstu jest zgodny ze wzorcem?

Funkcja ISLIKE jest bardzo prosta i jednocześnie bardzo użyteczna. Zwraca wartość TRUE, jeżeli łańcuch tekstowy jest zgodny ze zdefiniowanym wzorcem.

```

Function ISLIKE(text As String, pattern As String) As Boolean
    ' Zwraca wartość TRUE, jeżeli pierwszy argument jest podobny do drugiego
    ISLIKE = text Like pattern
End Function

```

Jak widać poniżej, kod funkcji jest bardzo prosty. Funkcja właściwie odgrywa rolę funkcji osłonowej, pozwalającej na wygodne użycie w tworzonych formułach wszechstronnego operatora Like języka VBA:

Funkcja ISLIKE pobiera dwa argumenty:

- text — łańcuch tekstowy lub odwołanie do komórki, która go zawiera;
- pattern — łańcuch zawierający znaki wieloznaczne, które wymieniono w poniższej tabeli.

Znaki zawarte we wzorcu	Zawartość łańcucha text zgodna ze wzorcem
?	Dowolny pojedynczy znak
*	0 lub więcej dowolnych znaków
#	Dowolna pojedyncza cyfra (0 – 9)
[lista_znaków]	Dowolny pojedynczy znak znajdujący się na liście_znaków
[!lista_znaków]	Dowolny pojedynczy znak, który nie znajduje się na liście_znaków

Poniższa formuła zwraca wartość TRUE, ponieważ wzorzec * jest zgodny z dowolną liczbą znaków. Formuła zwraca wartość TRUE, jeżeli pierwszy argument jest dowolnym łańcuchem tekstowym rozpoczynającym się od litery g:

```
=ISLIKE("gitara", "g*")
```

Kolejna formuła zwraca wartość TRUE, ponieważ wzorzec ? jest zgodny z dowolnym pojedynczym znakiem. Formuła zwróci wartość FALSE, jeżeli wartością pierwszego argumentu będzie łańcuch Jednostka12:

```
= ISLIKE("Jednostka1", "Jednostka?")
```

Następna formuła zwraca wartość TRUE, ponieważ jej pierwszy argument jest pojedynczym znakiem zawartym w drugim argumencie:

```
= ISLIKE("a", "[aeiou]")
```

Poniższa formuła zwraca wartość TRUE, jeżeli komórka A1 zawiera literę *a*, *e*, *i*, *o*, *u*, *A*, *E*, *I*, *O* lub *U*. Przetworzenie argumentów przy użyciu funkcji UPPER spowoduje, że formuła nie będzie rozróżniała wielkości znaków:

```
= ISLIKE(UPPER(A1), UPPER("[aeiou"]))
```

Poniższa formuła zwraca wartość TRUE, jeżeli komórka A1 zawiera wartość rozpoczynającą się cyfrą 1 i składającą się dokładnie z trzech cyfr (czyli dowolną liczbę całkowitą z przedziału od 100 do 199):

```
= ISLIKE(A1, "1##")
```

Wyznaczanie n-tego elementu łańcucha

Funkcja EXTRACTELEMENT jest niestandardową funkcją arkusza (może być też wywoływana z procedury języka VBA) wyznaczającą *n*-ty element łańcucha tekstowego. Jeżeli na przykład komórka zawiera poniższy tekst, w celu wydzielenia dowolnego podłańcucha zawartego pomiędzy łącznikami można użyć funkcji EXTRACTELEMENT:

```
123-456-789-0133-8844
```

Kolejna formuła zwraca podłańcuch 0133 będący czwartym elementem łańcucha (w roli separatora w łańcuchu jest używany łącznik):

```
=EXTRACTELEMENT("123-456-789-0133-8844", 4, "-")
```

Funkcja EXTRACTELEMENT pobiera trzy argumenty:

- *Txt* — łańcuch tekstowy, z którego są wydzielane podłańcuchy (może to być literał lub odwołanie do komórki);
- *n* — liczba całkowita reprezentująca wydzielany element;
- *Separator* — pojedynczy znak spełniający funkcję separatora.



Jeżeli wartością argumentu będzie spacja, ciągi kilku spacji zostaną potraktowane jak jedna spacja, co prawie zawsze będzie zgodne z Twoimi zamierzeniami. Jeżeli wartość argumentu *n* przekroczy liczbę elementów łańcucha, funkcja zwróci pusty łańcuch.

Oto kod źródłowy funkcji EXTRACTELEMENT języka VBA:

```
Function EXTRACTELEMENT(Txt, n, Separator) As String
    ' Zwraca n-ty element łańcucha tekstowego, w którym poszczególne elementy oddziela określony znak
    Dim AllElements As Variant
```

```

AllElements = Split(Txt, Separator)
EXTRACTELEMENT = AllElements(n - 1)
End Function

```

Funkcja korzysta z funkcji `Split` języka VBA zwracającej tablicę typu Variant, która zawiera poszczególne elementy łańcucha tekstu. Indeks tablicy rozpoczyna się od wartości 0, a nie 1, dlatego odwołania do kolejnych elementów tablicy są realizowane poprzez wyrażenie n-1.

Zamiana wartości na słowa¹

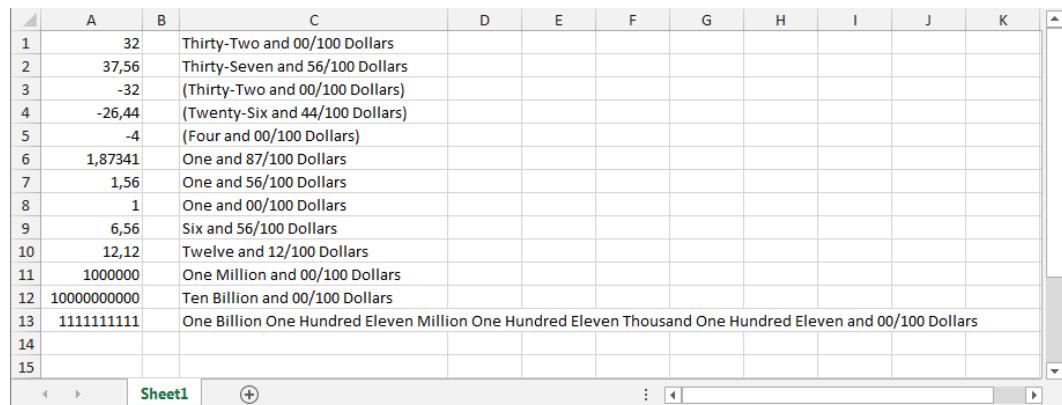
Funkcja `SPELLDOLLARS` zwraca „słowną” wersję wartości numerycznych podanych jako argument wywołania funkcji (tak jak w dobrze każdemu znanej formułce *SŁOWNIE*: spotykanej na blankietach przelewów i wpłat czy fakturach). Na przykład formuła przedstawiona poniżej zwraca następujący łańcuch tekstu: *One hundred twenty-three and 45/100 dollars* (sto dwadzieścia trzy dolary i 45 centów):

```
=SPELLDOLLARS(123.45)
```

Na rysunku 9.21 przedstawiono kilka przykładów zastosowania funkcji `SPELLDOLLARS`. Formuły zostały umieszczone w kolumnie C. Przykładowo formuła umieszczona w komórce C1 ma następującą postać:

```
=SPELLDOLLARS(A1)
```

Zwróć uwagę na fakt, że wartości ujemne są podawane w nawiasach.



	A	B	C	D	E	F	G	H	I	J	K
1	32		Thirty-Two and 00/100 Dollars								
2	37,56		Thirty-Seven and 56/100 Dollars								
3	-32		(Thirty-Two and 00/100 Dollars)								
4	-26,44		(Twenty-Six and 44/100 Dollars)								
5	-4		(Four and 00/100 Dollars)								
6	1,87341		One and 87/100 Dollars								
7	1,56		One and 56/100 Dollars								
8	1		One and 00/100 Dollars								
9	6,56		Six and 56/100 Dollars								
10	12,12		Twelve and 12/100 Dollars								
11	1000000		One Million and 00/100 Dollars								
12	10000000000		Ten Billion and 00/100 Dollars								
13	1111111111		One Billion One Hundred Eleven Million One Hundred Eleven Thousand One Hundred Eleven and 00/100 Dollars								
14											
15											

Rysunek 9.21. Przykłady zastosowania funkcji `SPELLDOLLARS`



Funkcja `SPELLDOLLARS` jest zbyt złożona, aby ją tutaj zaprezentować w całości, ale skoroszyt zawierający pełny kod tej funkcji (*Funkcja SPELLDOLLARS.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

¹ Uwaga: funkcja opisana w tym podrozdziale zwraca wartości w języku angielskim — *przyp. tłum.*

Funkcja wielofunkcyjna

Następny przykład prezentuje technikę, która może okazać się przydatna w niektórych sytuacjach. Sprawia ona, że pojedyncza funkcja arkusza zachowuje się jak wiele funkcji. Poniżej zamieszczamy kod źródłowy niestandardowej funkcji o nazwie STATFUNCTION, która pobiera dwa argumenty — zakres (rng) i operację (op). W zależności od wartości argumentu op funkcja zwraca wartość obliczoną przy użyciu dowolnej z następujących funkcji arkuszowych: AVERAGE (ŚREDNIA), COUNT (ILE.LICZB), MAX, MEDIAN (MEDIANA), MIN, MODE (WYST.NAJCZĘŚCIEJ), STDEV (ODCH.STANDARDOWE), SUM (SUMA) lub VAR (WARIANCJA).

Funkcji STATFUNCTION możesz użyć w arkuszu w następujący sposób:

```
=STATFUNCTION(B1:B24, A24)
```

Wynik formuły zależy od zawartości komórki A24, która powinna być takim łańcuchem, jak Average, Count, Max itd. Podobną technikę kodowania możesz zastosować w przypadku innych funkcji.

```
Function STATFUNCTION(rng, op)
    Select Case UCase(op)
        Case "SUM"
            STATFUNCTION = WorksheetFunction.Sum(rng)
        Case "AVERAGE"
            STATFUNCTION = WorksheetFunction.Average(rng)
        Case "MEDIAN"
            STATFUNCTION = WorksheetFunction.Median(rng)
        Case "MODE"
            STATFUNCTION = WorksheetFunction.Mode(rng)
        Case "COUNT"
            STATFUNCTION = WorksheetFunction.Count(rng)
        Case "MAX"
            STATFUNCTION = WorksheetFunction.Max(rng)
        Case "MIN"
            STATFUNCTION = WorksheetFunction.Min(rng)
        Case "VAR"
            STATFUNCTION = WorksheetFunction.Var(rng)
        Case "STDEV"
            STATFUNCTION = WorksheetFunction.StDev(rng)
        Case Else
            STATFUNCTION = CVErr(xlErrNA)
    End Select
End Function
```

Funkcja SHEETOFFSET

Excel oferuje ograniczoną obsługę trójwymiarowych skorosztów. Jeżeli na przykład konieczne jest odwołanie do innego arkusza skoroszytu, w formule trzeba uwzględnić nazwę arkusza. Nie stanowi to jednak dużego problemu... do momentu próby skopiowania formuły do innych arkuszy. Skopiowane formuły w dalszym ciągu odwołują się do nazwy oryginalnego arkusza, a odwołania do arkuszy nie są modyfikowane tak, jak miały to miejsce w prawdziwym trójwymiarowym arkuszu.

W tym podrozdziale został omówiony przykład funkcji języka VBA o nazwie SHEETOFFSET, umożliwiającej stosowanie względnych odwołań do arkuszy. Na przykład w celu odwołania się do komórki A1 poprzedniego arkusza należy użyć formuły:

```
=SHEETOFFSET(-1, A1)
```

Pierwszy argument funkcji, który może być wartością dodatnią, ujemną lub zerem, identyfikuje względne odwołanie do arkusza. Drugi argument musi być odwołaniem do pojętej komórki. Po skopiowaniu formuły do innych arkuszy odwołanie względne będzie obowiązywało we wszystkich jej kopiach.

Oto kod źródłowy funkcji SHEETOFFSET języka VBA:

```
Function SHEETOFFSET(Offset As Long, Optional Cell As Variant)
    ' Zwraca zawartość komórki względnie adresowanego arkusza, do której zdefiniowano odwołanie
    Dim WksIndex As Long, WksNum As Long
    Dim wks As Worksheet
    Application.Volatile
    If IsMissing(Cell) Then Set Cell = Application.Caller
    WksNum = 1
    For Each wks In Application.Caller.Parent.Parent.Worksheets
        If Application.Caller.Parent.Name = wks.Name Then
            SHEETOFFSET = Worksheets(WksNum + Offset).Range(Cell(1).Address)
            Exit Function
        Else
            WksNum = WksNum + 1
        End If
    Next wks
End Function
```

Zwracanie maksymalnej wartości ze wszystkich arkuszy

Aby określić maksymalną wartość komórki B1 z kilku arkuszy, można zastosować formułę podobną do poniższej:

```
=MAX(Arkusz1:Arkusz4!B1)
```

Formuła zwraca maksymalną wartość komórki B1 z arkuszy Arkusz1, Arkusz4 i wszystkich, które znajdują się pomiędzy nimi.

Co się jednak stanie, gdy za arkuszem Arkusz4 zostanie wstawiony nowy arkusz Arkusz5? Formuła nie uwzględnia tego automatycznie, dlatego konieczne będzie jej zmodyfikowanie w celu dodania nowego odwołania do arkusza:

```
=MAX(Arkusz1:Arkusz5!B1)
```

Funkcja MAXALLSHEETS pobiera jeden argument i zwraca maksymalną wartość określonej komórki ze wszystkich arkuszy skoroszytu. Przykładowo poniższa formuła zwraca maksymalną wartość komórki B1 z uwzględnieniem wszystkich arkuszy skoroszytu:

```
=MAXALLSHEETS(B1)
```

Po dodaniu nowego arkusza nie będzie już potrzeby edytowania formuły.

```

Function MAXALLSHEETS(cell)
    Dim MaxVal As Double
    Dim Addr As String
    Dim Wksht As Object
    Application.Volatile
    Addr = cell.Range("A1").Address
    MaxVal = -9.9E+307
    For Each Wksht In cell.Parent.Parent.Worksheets
        If Wksht.Name = cell.Parent.Name And
            Addr = Application.Caller.Address Then
            ' Uniknięcie odwołania cyklicznego
        Else
            If WorksheetFunction.IsNumber(Wksht.Range(Addr)) Then
                If Wksht.Range(Addr) > MaxVal Then
                    MaxVal = Wksht.Range(Addr).Value
                End If
            End If
        Next Wksht
        If MaxVal = -9.9E+307 Then MaxVal = 0
        MAXALLSHEETS = MaxVal
    End Function

```

Aby uzyskać dostęp do skoroszytu, pętla For Each używa następującego wyrażenia:

```
cell.Parent.Parent.Worksheets
```

Obiektem nadrzędnym komórki jest arkusz, natomiast przedkiem arkusza jest skoroszyt. Wynika z tego, że pętla For Each ... Next przetwarza wszystkie arkusze skoroszytu. Pierwsza instrukcja If z pętli sprawdza, czy przetwarzana komórka zawiera funkcję. Jeżeli tak jest, to w celu uniknięcia błędu odwołania cyklicznego komórka zostanie zignorowana.



Opisana funkcja z łatwością może zostać zmodyfikowana tak, aby wykonywała inne obliczenia międzyarkuszowe oparte na takich funkcjach, jak MIN, ŚREDNIA, SUMA itd.

Zwracanie tablicy zawierającej unikatowe, losowo uporządkowane liczby całkowite

Funkcja RANDOMINTEGERS zamieszczona w tym punkcie zwraca tablicę unikatowych liczb całkowitych. Stosowana jest w wielokomórkowych formułach tablicowych.

```
{=RANDOMINTEGERS()}
```

Zaznacz zakres, a następnie wprowadź formułę (bez nawiasów klamrowych) i zatwierdź ją poprzez naciśnięcie kombinacji klawiszy *Ctrl+Shift+Enter*. Formuła zwraca tablicę zawierającą unikatowe, losowo uporządkowane liczby całkowite. Jeżeli na przykład formula zostanie wprowadzona do zakresu złożonego z 50 komórek, jej kopie zwrócią unikatowe liczby całkowite z przedziału od 1 do 50.

Oto kod źródłowy funkcji RANDOMINTEGERS:

```

Function RANDOMINTEGERS()
    Dim FuncRange As Range
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double

```

```
Dim i As Integer, j As Integer
Dim r As Integer, c As Integer
Dim Temp1 As Variant, Temp2 As Variant
Dim RCount As Integer, CCount As Integer

' Tworzy obiekt klasy Range
Set FuncRange = Application.Caller

' Zwraca błąd, jeżeli wartość obiektu FuncRange jest zbyt duża
CellCount = FuncRange.Count
If CellCount > 1000 Then
    RANDOMINTEGERS = CVErr(xlErrNA)
    Exit Function
End If

' Przypisanie zmiennych
RCount = FuncRange.Rows.Count
CCount = FuncRange.Columns.Count
ReDim V(1 To RCount, 1 To CCount)
ReDim ValArray(1 To 2, 1 To CellCount)

' Wypełnienie tablicy losowymi wartościami i liczbami całkowitymi zakresu rng
For i = 1 To CellCount
    ValArray(1, i) = Rnd
    ValArray(2, i) = i
Next i

' Sortowanie tablicy ValArray według wymiaru o losowej wartości
For i = 1 To CellCount
    For j = i + 1 To CellCount
        If ValArray(1, i) > ValArray(1, j) Then
            Temp1 = ValArray(1, j)
            Temp2 = ValArray(2, j)
            ValArray(1, j) = ValArray(1, i)
            ValArray(2, j) = ValArray(2, i)
            ValArray(1, i) = Temp1
            ValArray(2, i) = Temp2
        End If
    Next j
Next i

' Wstawienie losowo uporządkowanych wartości do tablicy V
i = 0
For r = 1 To RCount
    For c = 1 To CCount
        i = i + 1
        V(r, c) = ValArray(2, i)
    Next c
Next r
RANDOMINTEGERS = V
End Function
```

Porządkowanie zakresu w losowy sposób

Funkcja RANGERANDOMIZE pobiera jeden argument będący zakresem i zwraca tablicę złożoną z losowo uporządkowanych wartości tego zakresu.

```

Function RANGERANDOMIZE(rng)
    Dim V() As Variant, ValArray() As Variant
    Dim CellCount As Double
    Dim i As Integer, j As Integer
    Dim r As Integer, c As Integer
    Dim Temp1 As Variant, Temp2 As Variant
    Dim RCount As Integer, CCount As Integer

    ' Zwraca błąd, jeżeli wartość obiektu rng jest zbyt duża
    CellCount = rng.Count
    If CellCount > 1000 Then
        RangeRandomize = CVErr(xlErrNA)
        Exit Function
    End If

    ' Przypisanie zmiennych
    RCount = rng.Rows.Count
    CCount = rng.Columns.Count
    ReDim V(1 To RCount, 1 To CCount)
    ReDim ValArray(1 To 2, 1 To CellCount)

    ' Wypełnienie tablicy ValArray losowymi wartościami i wartościami obiektu rng
    For i = 1 To CellCount
        ValArray(1, i) = Rnd
        ValArray(2, i) = rng(i)
    Next i

    ' Sortowanie tablicy ValArray według wymiaru o losowej wartości
    For i = 1 To CellCount
        For j = i + 1 To CellCount
            If ValArray(1, i) > ValArray(1, j) Then
                Temp1 = ValArray(1, j)
                Temp2 = ValArray(2, j)
                ValArray(1, j) = ValArray(1, i)
                ValArray(2, j) = ValArray(2, i)
                ValArray(1, i) = Temp1
                ValArray(2, i) = Temp2
            End If
        Next j
    Next i

    ' Wstawienie losowo uporządkowanych wartości do tablicy V
    i = 0
    For r = 1 To RCount
        For c = 1 To CCount
            i = i + 1
            V(r, c) = ValArray(2, i)
        Next c
    Next r
    RANGERANDOMIZE = V
End Function

```

Jak łatwo zauważyc, kod źródłowy tej funkcji jest bardzo podobny do kodu omawianej przed chwilą funkcji RANDOMINTEGERS.

Na rysunku 9.22 pokazano wynik działania funkcji. Formuła tablicowa zawarta w zakresie B2:B11 ma następującą postać:

{=RANGERANDOMIZE(A2:A11)}

Rysunek 9.22.

*Funkcja
RANGERANDOMIZE
zwraca zawartość
komórek zakresu
w przypadkowej
 kolejności*

A	B
1 Oryginal	Losowo
2 Antylopa	Gekon
3 Bażant	Drozd
4 Czapla	Emu
5 Drozd	Bażant
6 Emu	Czapla
7 Fretka	Antylopa
8 Gekon	Jastrząb
9 Hipopotam	Hipopotam
10 Iguana	Fretka
11 Jastrząb	Iguana
12	
13	
14	

Formuła zwraca zawartość komórek zakresu A2:A11, ale uporządkowanego w losowy sposób.

Sortowanie zakresów

Funkcja SORTED pobiera zakres komórek jako argument wywołania, sortuje go i zwraca w postaci posortowanej:

```
Function SORTED(Rng)
    Dim SortedData() As Variant
    Dim Cell1 As Range
    Dim Temp As Variant, i As Long, j As Long
    Dim NonEmpty As Long

    ' Przenoszenie danych do tablicy SortedData
    For Each Cell In Rng
        If Not IsEmpty(Cell) Then
            NonEmpty = NonEmpty + 1
            ReDim Preserve SortedData(1 To NonEmpty)
            SortedData(NonEmpty) = Cell.Value
        End If
    Next Cell

    ' Sortowanie tablicy
    For i = 1 To NonEmpty
        For j = i + 1 To NonEmpty
            If SortedData(i) > SortedData(j) Then
                Temp = SortedData(j)
                SortedData(j) = SortedData(i)
                SortedData(i) = Temp
            End If
        Next j
    Next i
End Function
```

```

    Next j
Next i

' Transpozycja i zwracanie tablicy
SORTED = Application.Transpose(SortedData)
End Function

```

Na rysunku 9.23 przedstawiono wyniki działania funkcji SORTED, która została wprowadzona jako formula tablicowa do zakresu komórek.

Rysunek 9.23.
Funkcja SORTED
zwraca posortowaną
zawartość komórek
zakresu

	A	B	C	D	E	F	G	H
1	Losowe	Posortowane						
2	Kamil	Aneta						
3	Franek	Anna						
4	Jacek	Darek						
5	Grzegorz	Franek						
6	Anna	Grzegorz						
7	Lalka	Jacek						
8	Zosia	Kamil						
9	Ola	Lalka						
10	Rafal	Maria						
11	Maria	Ola						
12	Aneta	Rafal						
13	Darek	Zosia						
14								
15								
16								

Funkcja SORTED rozpoczyna działanie od utworzenia tablicy o nazwie SortedData, która zawiera wszystkie niezerowe wartości z zakresu komórek będącego argumentem wywołania funkcji. Następnie zawartość tej tablicy jest sortowana przy użyciu algorytmu sortowania bąbelkowego. Ponieważ tablica SortedData jest pozioma, przed zwróceniem jako wynik działania funkcji zostaje transponowana do postaci pionowej.

Funkcja Sorted może działać z zakresami o dowolnym rozmiarze dopóty, dopóki mają postać pojedynczego wiersza lub pojedynczej kolumny. Jeżeli nieposortowane dane znajdują się w wierszu, to do wyświetlania posortowanych wyników w postaci wiersza powinieneś użyć funkcji arkuszowej TRANSPOSE, tak jak to zostało przedstawione w przykładzie poniżej:

```
=TRANSPOSE(SORTED(A16:L16))
```

Wywołania funkcji interfejsu Windows API

Jedną z najważniejszych cech języka VBA jest możliwość wywoływanego funkcji przechowywanych w bibliotekach DLL (ang. *Dynamic Link Library*). W przykładach zaprezentowanych w tym podrozdziale będziemy korzystali z często używanych funkcji interfejsu API systemu Windows.



Dla uproszczenia deklaracje funkcji API przedstawiane w tym podrozdziale działają tylko w Excelu 2010 i Excelu 2013 (zarówno w wersjach 32-bitowych, jak i 64-bitowych), natomiast przykłady zamieszczone na dysku CD-ROM dołączonym do książki zawierają odpowiednie dyrektywy kompilatora, dzięki czemu będą poprawnie działać również we wcześniejszych wersjach Excela.

Określanie skojarzeń plików

W systemie Windows wiele typów plików jest kojarzonych z określona aplikacją. Po wykonaniu takiego skojarzenia plik można otworzyć w powiązanej z nim aplikacji poprzez dwukrotne kliknięcie pliku.

Funkcja GetExecutable w celu uzyskania pełnej ścieżki aplikacji skojarzonej z określonym plikiem korzysta z funkcji interfejsu API systemu Windows. Przyjmijmy, że w Twoim systemie znajduje się wiele plików o rozszerzeniu .txt. Jeden z nich, o nazwie *Readme.txt*, prawdopodobnie znajduje się w katalogu systemu Windows. Aby określić pełną ścieżkę aplikacji otwierającej plik po jego dwukrotnym kliknięciu, można użyć funkcji GetExecutable.



Deklaracje funkcji interfejsu API systemu Windows muszą zostać umieszczone na początku modułu kodu VBA.

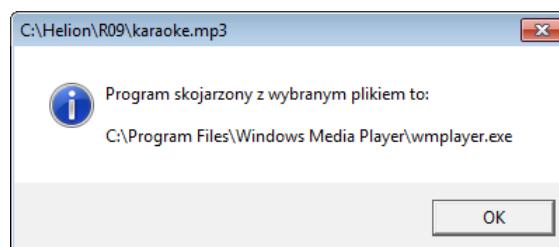
```
Private Declare Function FindExecutableA Lib "shell32.dll" _
    (ByVal lpFile As String, ByVal lpDirectory As String, _
    ByVal lpResult As String) As Long

Function GetExecutable(strFile As String) As String
    Dim strPath As String
    Dim intLen As Integer
    strPath = Space(255)
    intLen = FindExecutableA(strFile, "\", strPath)
    GetExecutable = Trim(strPath)
End Function
```

Na rysunku 9.24 pokazano wynik wywołania funkcji GetExecutable, która jako argument pobrała nazwę pliku muzycznego w formacie MP3. Funkcja zwraca pełną ścieżkę aplikacji powiązanej z plikiem.

Rysunek 9.24.

Określanie ścieżki aplikacji powiązanej z określonym plikiem

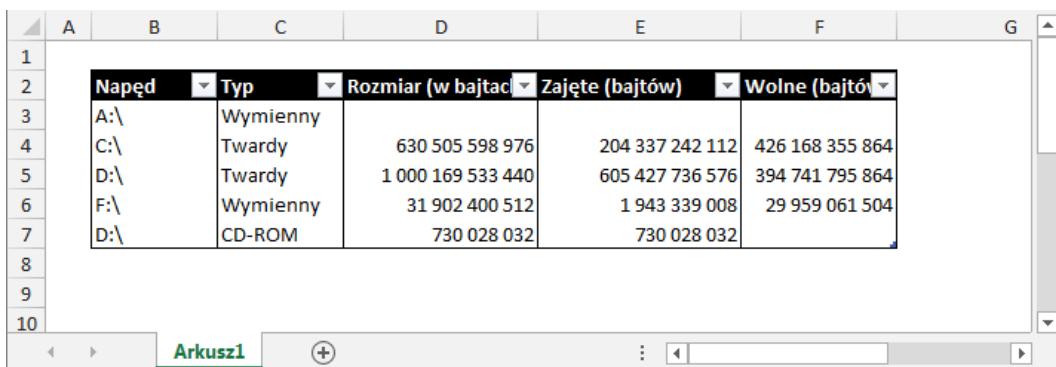


Skoroszyt z tym przykładem (*Skojarzenia plików.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Pobieranie informacji o napędach dyskowych

VBA nie posiada metody pozwalającej na bezpośrednie pobieranie informacji o zainstalowanych w systemie napędach dyskowych. Jednak dzięki zastosowaniu trzech funkcji API możesz uzyskać niezbędne informacje.

Na rysunku 9.25 przedstawiono wynik działania procedury VBA, która identyfikuje wszystkie podłączone do systemu dyski, określa ich typ, sprawdza całkowitą pojemność, rozmiar użytego miejsca oraz rozmiar wolnego miejsca.



The screenshot shows a Microsoft Excel spreadsheet titled 'Arkusz1'. The table has columns labeled 'Napęd', 'Typ', 'Rozmiar (w bajtach)', 'Zajęte (abajtów)', and 'Wolne (abajtów)'. The data is as follows:

Napęd	Typ	Rozmiar (w bajtach)	Zajęte (abajtów)	Wolne (abajtów)
A:\	Wymienny			
C:\	Twardy	630 505 598 976	204 337 242 112	426 168 355 864
D:\	Twardy	1 000 169 533 440	605 427 736 576	394 741 795 864
F:\	Wymienny	31 902 400 512	1 943 339 008	29 959 061 504
D:\	CD-ROM	730 028 032	730 028 032	

Rysunek 9.25. Zastosowanie funkcji Windows API do pobierania informacji o dyskach



Kod procedury jest dosyć długi i złożony, dlatego nie umieszczono go tutaj, ale jeżeli jesteś ciekawy, jak to działa, możesz zajrzeć na stronę internetową naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>), gdzie znajdziesz skoroszyt z tym przykładem (*Informacja o dyskach.xlsxm*).

Pobieranie informacji dotyczących drukarki domyślnej

W kolejnym przykładzie użyjemy funkcji interfejsu Windows API zwracającej informacje na temat domyślnej drukarki. Dane znajdują się w pojedynczym łańcuchu tekstowym. Poniższa procedura analizuje łańcuch i wyświetla informacje przy użyciu bardziej czytelnego dla użytkownika formatu:

```
Private Declare Function GetProfileStringA Lib "kernel32" _
    (ByVal lpAppName As String, ByVal lpKeyName As String, _
    ByVal lpDefault As String, ByVal lpReturnedString As String, _
    ByVal nSize As Long) As Long

Sub DefaultPrinterInfo()
    Dim strLPT As String * 255
    Dim Result As String
    Call GetProfileStringA("Windows", "Device", "", strLPT, 254)

    Result = Application.Trim(strLPT)
    ResultLength = Len(Result)
```

```

Comma1 = InStr(1, Result, ".", 1)
Comma2 = InStr(Comma1 + 1, Result, ".", 1)

    ' Pobiera nazwę drukarki
    Printer = Left(Result, Comma1 - 1)

    ' Pobiera informacje na temat sterownika
    Driver = Mid(Result, Comma1 + 1, Comma2 - Comma1 - 1)

    ' Pobiera ostatnią część informacji na temat urządzenia
    Port = Right(Result, ResultLength - Comma2)

    ' Tworzy komunikat
    Msg = "Drukarka:" & Chr(9) & Printer & Chr(13)
    Msg = Msg & "Sterownik:" & Chr(9) & Driver & Chr(13)
    Msg = Msg & "Port:" & Chr(9) & Port

    ' Wyświetla komunikat
    MsgBox Msg, vbInformation, "Informacje o drukarce domyślnej"
End Sub

```

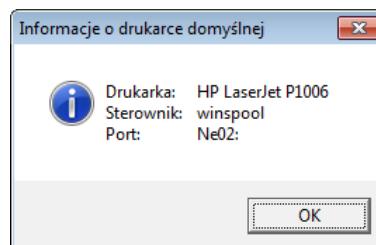


Co prawda właściwość ActivePrinter obiektu Application zwraca nazwę domyślnej drukarki i umożliwia jej zmianę, ale nie istnieje bezpośrednia metoda określenia, jaki sterownik i port urządzenia jest używany. Z tego właśnie powodu czasami przydatna może być nasza funkcja GetProfileStringA.

Na rysunku 9.26 pokazano przykładowe okno komunikatu wyświetcone przez tę procedurę.

Rysunek 9.26.

Informacja o drukarce domyślnej wyświetlona przy użyciu funkcji interfejsu API systemu Windows



Skoroszyt z tym przykładem (*Informacja o drukarce.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Pobieranie informacji o aktualnej rozdzielczości karty graficznej

Zamieszczony w tym punkcie kod korzysta z funkcji interfejsu API w celu określenia aktualnej rozdzielczości karty graficznej używanej w systemie. Jeżeli używana aplikacja musi wyświetlić określoną ilość informacji na jednym ekranie, znajomość jego rozdzielczości może pomóc we właściwym przeskalowaniu tekstu. Oprócz tego kod procedury sprawdza liczbę monitorów podłączonych do komputera. Jeżeli podłączonych jest więcej monitorów niż jeden, procedura wyświetla pulpit wirtualnego.

```

Declare PtrSafe Function GetSystemMetrics Lib "user32" _
    (ByVal nIndex As Long) As Long

```

```
Public Const SM_CMONITORS = 80
Public Const SM_CXSCREEN = 0
Public Const SM_CYSCREEN = 1
Public Const SM_CXVIRTUALSCREEN = 78
Public Const SM_CYVIRTUALSCREEN = 79

Sub DisplayVideoInfo()
    Dim numMonitors As Long
    Dim vidWidth As Long, vidHeight As Long
    Dim virtWidth As Long, virtHeight As Long
    Dim Msg As String

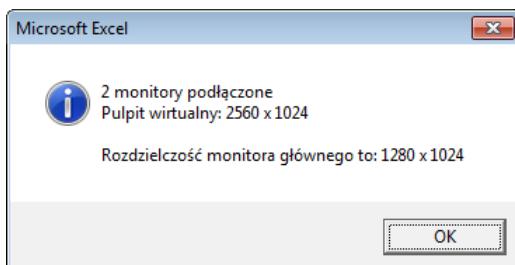
    numMonitors = GetSystemMetrics(SM_CMONITORS)
    vidWidth = GetSystemMetrics(SM_CXSCREEN)
    vidHeight = GetSystemMetrics(SM_CYSCREEN)
    virtWidth = GetSystemMetrics(SM_CXVIRTUALSCREEN)
    virtHeight = GetSystemMetrics(SM_CYVIRTUALSCREEN)

    If numMonitors > 1 Then
        Msg = numMonitors & " monitory podłączone" & vbCrLf
        Msg = Msg & "Pulpit wirtualny: " & virtWidth & " x "
        Msg = Msg & virtHeight & vbCrLf & vbCrLf
        Msg = Msg & "Rozdzielcość monitora głównego to: "
        Msg = Msg & vidWidth & " x " & vidHeight
    Else
        Msg = Msg & "Aktualny tryb graficzny: "
        Msg = Msg & vidWidth & " x " & vidHeight
    End If
    MsgBox Msg
End Sub
```

Na rysunku 9.27 pokazano okno komunikatu zwrócone przez powyższą procedurę uruchomioną w systemie używającym dwóch monitorów.

Rysunek 9.27.

Zastosowanie funkcji
interfejsu Windows API
do określenia
rozdzielcości karty
graficznej



Skoroszyt z tym przykładem (*Informacja o rozdzielcości karty graficznej.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Odczytywanie zawartości rejestru systemu Windows i zapisywanie w nim danych

Większość aplikacji Windows potrzebne informacje przechowuje w rejestrze systemu będącym bazą danych. Procedury języka VBA są w stanie odczytywać dane z rejestru i zapisywać w nim nowe wartości. Aby to było możliwe, konieczne jest zastosowanie następujących deklaracji funkcji interfejsu API systemu Windows:

```
Private Declare PtrSafe Function RegOpenKeyA Lib "ADVAPI32.DLL" _  
    (ByVal hKey As Long, ByVal sSubKey As String, _  
     ByRef hkeyResult As Long) As Long  
Private Declare PtrSafe Function RegCloseKey Lib "ADVAPI32.DLL" _  
    (ByVal hKey As Long) As Long  
Private Declare PtrSafe Function RegSetValueExA Lib "ADVAPI32.DLL" _  
    (ByVal hKey As Long, ByVal sValueName As String, _  
     ByVal dwReserved As Long, ByVal dwType As Long, _  
     ByVal sValue As String, ByVal dwSize As Long) As Long  
  
Private Declare PtrSafe Function RegCreateKeyA Lib "ADVAPI32.DLL" _  
    (ByVal hKey As Long, ByVal sSubKey As String, _  
     ByRef hkeyResult As Long) As Long  
Private Declare PtrSafe Function RegQueryValueExA Lib "ADVAPI32.DLL" _  
    (ByVal hKey As Long, ByVal sValueName As String, _  
     ByVal dwReserved As Long, ByRef lValueType As Long, _  
     ByVal sValue As String, ByRef lResultLen As Long) As Long
```



Utworzyłem dwie funkcje osłonowe ułatwiające korzystanie z rejestru. Są to: GetRegistry i WriteRegistry. Obie znajdują się w skoroszycie o nazwie *Rejestr systemu Windows.xlsm*, który został zamieszczony na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Przykładowy skoroszyt zawiera procedurę demonstrującą odczyt i zapis danych w rejestrze.

Odczyt danych z rejestru

Funkcja GetRegistry zwraca ustawienia znajdujące się w określonej lokalizacji rejestru. Funkcja pobiera trzy argumenty:

- RootKey — łańcuch reprezentujący główny klucz rejestru, który zostanie użyty.
Oto możliwe łańcuchy:
 - HKEY_CLASSES_ROOT
 - HKEY_CURRENT_USER
 - HKEY_LOCAL_MACHINE
 - HKEY_USERS
 - HKEY_CURRENT_CONFIG
- Path — pełna ścieżka kategorii rejestru, która zostanie użyta.
- RegEntry — nazwa ustawienia, które zostanie odczytane.

Aby na przykład odnaleźć w rejestrze aktualne ustawienie powiązane z aktywnym paskiem tytułu okna, należy w sposób pokazany poniżej wywołać funkcję GetRegistry (wielkość znaków nazw argumentów nie jest rozróżniana):

```
RootKey = "hkey_current_user"
Path = "Control Panel\Desktop"
RegEntry = "WallPaper"
MsgBox GetRegistry(RootKey, Path, RegEntry), _
vbInformation, Path & "\RegEntry"
```

Okno komunikatu wyświetli ścieżkę i nazwę pliku graficznego użytego w roli tapety pulpitu (jeżeli tapeta nie jest używana, funkcja zwróci pusty łańcuch).

Zapis danych w rejestrze

Funkcja WriteRegistry zapisuje wartość w określonej lokalizacji rejestru. Jeżeli operacja zakończy się powodzeniem, funkcja zwróci wartość True. W przeciwnym razie zwróci wartość False. Funkcja WriteRegistry pobiera następujące argumenty (wszystkie są łańcuchami tekstu):

- RootKey — łańcuch reprezentujący klucz rejestru, który zostanie użyty.
Oto możliwe łańcuchy:
 - HKEY_CLASSES_ROOT
 - HKEY_CURRENT_USER
 - HKEY_LOCAL_MACHINE
 - HKEY_USERS
 - HKEY_CURRENT_CONFIG
- Path — pełna ścieżka kategorii rejestru (jeżeli ścieżka nie istnieje, zostanie utworzona).
- RegEntry — nazwa kategorii rejestru, w której zostanie zapisana wartość (jeżeli kategoria nie istnieje, zostanie dodana).
- RegVal — zapisywana wartość.

Poniżej zamieszczono przykład procedury zapisującej w rejestrze wartość reprezentującą datę i czas uruchomienia Excela. Informacja jest zapisywana w miejscu, w którym są przechowywane ustawienia dotyczące Excela.

```
Sub Workbook_Open()
    RootKey = "hkey_current_user"
    Path = "software\microsoft\office\15.0\Excel\LastStarted"
    RegEntry = "DateTime"
    RegVal = Now()
    If WriteRegistry(RootKey, Path, RegEntry, RegVal) Then
        msg = RegVal & " została zapisana w rejestrze."
    Else
        msg = "Wystąpił błąd."
    End If
    MsgBox msg
End Sub
```

Jeżeli zapiszesz tę procedurę w module ThisWorkbook skoroszytu makr osobistych, ustawienia będą automatycznie aktualizowane przy każdym uruchomieniu programu Excel.

Łatwiejszy sposób uzyskania dostępu do rejestru

Jeżeli w celu zapisania i odczytania danych dostęp do rejestru systemu Windows chcesz uzyskać z poziomu aplikacji Excela, nie musisz stosować funkcji interfejsu API. Zamiast nich można użyć funkcji GetSetting i SaveSetting języka VBA. Zastosowanie tych funkcji jest znacznie łatwiejsze niż wykorzystywanie wywołań funkcji API.

Obie funkcje zostały objaśnione w systemie pomocy, dlatego nie będę ich tutaj szczegółowo omawiał. Jednak należy wiedzieć, że funkcje te działają tylko z kluczem o następującej nazwie:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings

Innymi słowy, funkcje nie mogą zostać zastosowane w celu uzyskania dostępu do dowolnego klucza rejestru. Funkcje te są najbardziej przydatne do zapisywania informacji o własnych aplikacjach Excela, które chcesz przechować pomiędzy kolejnymi sesjami.

Część III

Praca z formularzami

UserForm

W tej części:

- Rozdział 10. „Tworzenie własnych okien dialogowych”
- Rozdział 11. „Wprowadzenie do formularzy UserForm”
- Rozdział 12. „Przykłady formularzy UserForm”
- Rozdział 13. „Zaawansowane techniki korzystania z formularzy UserForm”

Rozdział 10.

Tworzenie własnych

okien dialogowych

W tym rozdziale:

- Zastosowanie okna wprowadzania danych — funkcje `InputBox` w języku VBA i Excelu
- Zastosowanie okna komunikatów — funkcja `MsgBox` języka VBA
- Wybieranie pliku z listy w oknie dialogowym
- Wybieranie katalogu z listy w oknie dialogowym
- Wyświetlanie wbudowanych okien dialogowych Excela

Zanim rozpoczęnesz tworzenie formularza `UserForm`

Okna dialogowe są jednym z najważniejszych elementów interfejsu użytkownika aplikacji Windows. Korzysta z nich prawie każda aplikacja Windows i większość użytkowników dobrze rozumie zasady ich funkcjonowania. Projektanci aplikacji Excela mogą tworzyć własne okna dialogowe przy użyciu formularzy `UserForm`, jak również mogą przy użyciu odpowiednich poleceń VBA wyświetlać standardowe okna dialogowe programu Excel i to przy minimalnym nakładzie programowania.

Zanim przejdziemy do szczegółowych zagadnień związanych z tworzeniem formularzy `UserForm` (o czym będziemy mówili w rozdziale 11.), warto zapoznać się z kilkoma narzędziami Excela pozwalającymi na wyświetlanie wbudowanych okien dialogowych, których zastosowanie pozwala w niektórych sytuacjach na wyeliminowanie konieczności tworzenia niestandardowego okna dialogowego.

Okno wprowadzania danych

Okno wprowadzania danych jest prostym oknem dialogowym umożliwiającym użytkownikowi wprowadzenie pojedynczego wiersza danych. Może posłużyć do wprowadzania tekstu, liczb, a nawet wybierania zakresu. Istnieją dwa sposoby utworzenia takiego okna:

pierwszy z nich polega na zastosowaniu odpowiedniej funkcji języka VBA, a drugi na użyciu wybranych metod obiektu Application.

Funkcja InputBox języka VBA

Składnia funkcji InputBox języka VBA jest następująca:

`InputBox(komunikat[, tytuł][, wartość_domyślna][, wsp_x][, wsp_y][, plik_pomocy, kontekst])`

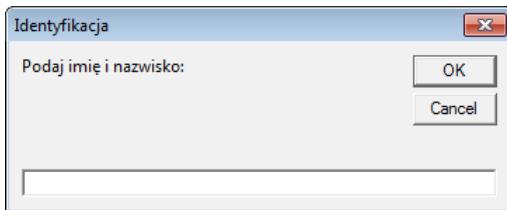
- *komunikat* (wymagany) — tekst wyświetlany w oknie wprowadzania danych;
- *tytuł* (opcjonalny) — tytuł okna wprowadzania danych;
- *wartość_domyślna* (opcjonalna) — domyślana wartość wyświetlna w oknie dialogowym;
- *wsp_x, wsp_y* (opcjonalne) — współrzędne pozycji górnego lewego narożnika okna;
- *plik_pomocy, kontekst* (opcjonalne) — plik i temat pomocy.

Funkcja InputBox prosi użytkownika o wprowadzenie pojedynczego wiersza informacji. Zawsze zwraca łańcuch, dlatego może być konieczne wykonanie konwersji wyniku na wartość liczbową.

Komunikat może liczyć do około 1024 znaków (mniej więcej, w zależności od ich szerokości). Dodatkowo można zdefiniować tytuł okna dialogowego i wartość domyślną oraz określić położenie okna na ekranie. Poza tym można określić niestandardowy temat pomocy. W tym przypadku w oknie wprowadzania danych pojawi się przycisk *Pomoc*.

W poniższej przykładowej procedurze, której efekt działania pokazano na rysunku 10.1, zastosowano funkcję InputBox języka VBA, która prosi użytkownika o wprowadzenie imienia i nazwiska. Później procedura pobiera imię i przy użyciu okna komunikatu wyświetla powitanie.

Rysunek 10.1.
Efekt wywołania funkcji
InputBox języka VBA

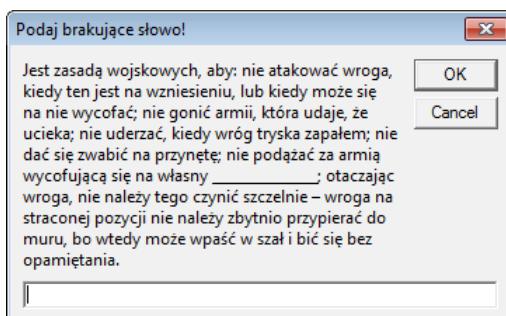


```
Sub GetName()
    Dim UserName As String
    Dim FirstSpace As Integer
    Do Until UserName <> ""
        UserName = InputBox("Podaj imię i nazwisko: ", "Identyfikacja")
    Loop
    FirstSpace = InStr(UserName, " ")
    If FirstSpace <> 0 Then
        UserName = Left(UserName, FirstSpace - 1)
    End If
    MsgBox "Witaj " & UserName
End Sub
```

Aby zagwarantować, że po pojawienniu się okna zostaną wprowadzone dane, funkcja InputBox została umieszczona wewnątrz pętli Do Until. Jeżeli użytkownik naciśnie przycisk Cancel lub nie wprowadzi żadnego tekstu, zmiennej UserName zostanie przypisany pusty łańcuch, po czym okno wprowadzania danych pojawi się ponownie. Po wprowadzeniu danych procedura szuka pierwszej spacji (przy użyciu funkcji InStr), aby pobrać imię, a następnie przy wykorzystaniu funkcji Left pobiera wszystkie znaki przed spacją. Jeżeli nie znajdzie spacji, pobierze cały wprowadzony łańcuch.

Na rysunku 10.2 przedstawiono kolejny przykład zastosowania funkcji InputBox języka VBA. Użytkownik jest proszony o wprowadzenie brakującego słowa. Przykład ten ilustruje również sposób użycia argumentów posiadających nazwy. Tekst zapowiedzi jest pobierany z komórki arkusza i przypisany do zmiennej p.

Rysunek 10.2.
Użycie funkcji InputBox
języka VBA
do wyświetlenia
długiej zapowiedzi



```
Sub GetWord()
    Dim TheWord As String
    Dim p As String
    Dim t As String
    p = Range("A1")
    t = "Podaj brakujący fragment tekstu:"
    TheWord = InputBox(prompt:=p, Title:=t)
    If UCASE(TheWord) = "TEREN" Then
        MsgBox "Zgadza się!"
    Else
        MsgBox "Wprowadzone słowo nie jest poprawne!"
    End If
End Sub
```

Jak już wspominałem, funkcja InputBox zawsze zwraca łańcuch tekstu. Jeżeli łańcuch tekstu zwrócony przez funkcję InputBox wygląda jak liczba, możesz go zamienić na wartość liczbową przy użyciu funkcji Val języka VBA lub poprzez wykonanie na takim ciągu znaków dowolnej operacji matematycznej.

Procedura, której kod został przedstawiony poniżej, wykorzystuje funkcję InputBox do poproszenia użytkownika o wprowadzenie wartości liczbowej. Następnie procedura używa funkcji IsNumeric do sprawdzenia, czy wprowadzony ciąg znaków może być zinterpretowany jako liczba. Jeżeli tak, procedura mnoży wartość wprowadzoną przez użytkownika przez 12 i wyświetla wynik na ekranie.

```
Sub GetValue()
    Dim Monthly As String
    Dim Annual As Double
    Monthly = InputBox("Podaj swoją miesięczną pensję:")
```

```

If Monthly = "" Then Exit Sub
On Error Resume Next
If IsNumeric(Monthly) Then
    MsgBox "Pensja rocznie: " & Monthly * 12
Else
    MsgBox "Wprowadziłeś niepoprawną wartość..."
End If
End Sub

```



Skoroszyt zawierający wszystkie trzy przykłady (*Funkcja InputBox.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Metoda InputBox Excela

Użycie metody InputBox Excela zamiast funkcji InputBox języka VBA przynosi następujące korzyści:

- możesz określić typ danych zwracanej wartości;
- możesz zdefiniować zakres poprzez przeciągnięcie myszą w obrębie arkusza;
- sprawdzanie poprawności wprowadzonych danych jest wykonywane automatycznie.

Składnia metody InputBox Excela jest następująca:

```
InputBox(komunikat [,tytuł] [,wart_domyślna] [,wsp_x] [,wsp_y] [,plik_pomocy, kontekst]
[,typ])
```

- *komunikat* (wymagany) — tekst wyświetlany w oknie wprowadzania danych;
- *tytuł* (opcjonalny) — tytuł okna wprowadzania danych;
- *wart_domyślna* (opcjonalna) — domyślana wartość zwracana przez funkcję, gdy użytkownik nie wprowadzi danych;
- *wsp_x, wsp_y* (opcjonalne) — współrzędne pozycji górnego lewego narożnika okna;
- *plik_pomocy, kontekst* (opcjonalne) — plik i temat pomocy;
- *typ* (opcjonalny) — kod identyfikujący typ danych zwracanej wartości (tabela 10.1).

Tabela 10.1. Kod identyfikujący typ danych zwracanych przez metodę InputBox Excela

Kod	Znaczenie
0	Formuła
1	Liczba
2	Łańcuch tekstowy
4	Wartość logiczna (True lub False)
8	Odwołanie do komórki będące obiektem Range
16	Wartość błędu, na przykład #N/D!
64	Tablica wartości



Wygląda na to, że argumenty `wsp_x`, `wsp_y`, `plik_pomocy` oraz `kontekst` nie są już w nowym Excelu obsługiwane. Co prawda przy wywołaniu metody możesz je podawać, ale nie mają one żadnego wpływu na jej działanie.

Metoda `InputBox` Excela jest dość wszechstronna. Aby funkcja zwracała wartość więcej niż jednego typu danych, należy użyć sumy odpowiednich kodów. Przykładowo: aby wyświetlić okno umożliwiające wprowadzanie tekstu i liczb, jako wartość `typ` należy podać 3 (1+2 lub `liczba+lańcuch tekstowy`). Jeżeli wartością argumentu `typ` będzie 8, użytkownik może ręcznie wprowadzić adres komórki lub zakresu albo wskazać go w arkuszu.

Procedura `EraseRange`, której kod zamieszczamy poniżej, używa metody `InputBox` do umożliwienia użytkownikowi wybrania zakresu, który zostanie wyczyszczony (patrz rysunek 10.3). Użytkownik może ręcznie wprowadzić adres zakresu lub zaznaczyć go w arkuszu za pomocą myszy.

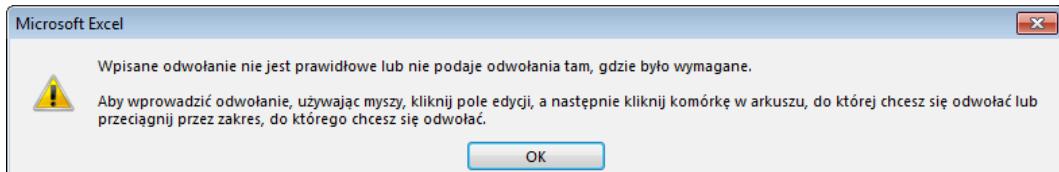
Rysunek 10.3. Użycie okna wprowadzania danych do zdefiniowania zakresu

Metoda `InputBox` z argumentem `Type` o wartości 8 zwraca obiekt `Range` (zwróć uwagę na słowo kluczowe `Set`). Zakres jest następnie czyszczony przy użyciu metody `Clear`. Domyślna wartość wyświetlana w oknie wprowadzania danych jest adresem aktualnego zaznaczenia. Jeżeli w oknie zostanie kliknięty przycisk `Anuluj`, instrukcja `On Error` zakończy wykonywanie procedury.

```
Sub EraseRange()
    Dim UserRange As Range
    On Error GoTo Canceled
    Set UserRange = Application.InputBox _
        (Prompt:="Podaj zakres, który zostanie wyczyszczony:", _
        Title:="Czyszczenie zakresu", _
        Default:=DefaultRange, _
        Type:=8)
    UserRange.Clear
    Canceled:
End Sub
```

```
UserRange.Select
Cancelled:
End Sub
```

Kolejną korzyścią wynikającą z zastosowania metody InputBox Excela jest automatyczne sprawdzenie wprowadzonych danych. Jeżeli w przykładowej procedurze EraseRange zostanie wprowadzone coś innego niż adres zakresu, Excel wyświetli komunikat i umożliwi użytkownikowi ponowne wykonanie operacji (patrz rysunek 10.4).

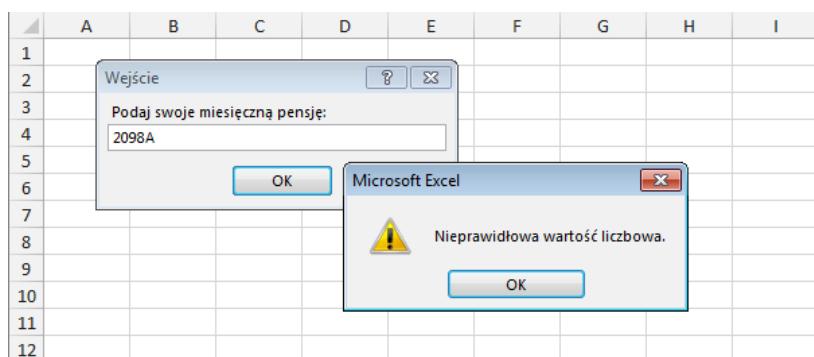


Rysunek 10.4. Metoda InputBox Excela automatycznie sprawdza poprawność wprowadzonych danych

Kod procedury przedstawiony poniżej jest podobny do kodu procedury GetValue, o której mówiliśmy w poprzednim podrozdziale, ale różni się tym, że wykorzystuje metodę InputBox Excela. Mimo że typ argumentu został w kodzie zdefiniowany jako 1 (czyli wartość numeryczna), zmienna Monthly została zadeklarowana jako Variant. Musimy tak zrobić, ponieważ jeżeli użytkownik naciśnie przycisk *Cancel*, metoda InputBox zwróci wartość logiczną False (Fałsz). Jeżeli użytkownik wprowadzi wartość, która nie jest numeryczna, Excel wyświetli na ekranie odpowiedni komunikat i poprosi o ponowne wprowadzenie prawidłowej wartości (patrz rysunek 10.5).

```
Sub GetValue2()
    Dim Monthly As Variant
    Monthly = Application.InputBox _
        (Prompt:=" Podaj swoją miesięczną pensję: ", _
        Type:=1)
    If Monthly = False Then Exit Sub
    MsgBox "Pensja rocznie: " & Monthly * 12
End Sub
```

Rysunek 10.5.
Kolejny przykład sprawdzania poprawności wprowadzanych wartości przy użyciu metody InputBox



Skoroszyt zawierający oba przykłady (*Metoda InputBox.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci

Funkcja MsgBox języka VBA

Funkcja MsgBox języka VBA umożliwia wyświetlenie komunikatu użytkownikowi lub odebranie od niego prostej odpowiedzi, takiej jak wcisnięcie przycisków *OK* lub *Anuluj* (*Cancel*). Funkcja MsgBox występuje w wielu omawianych przykładach, gdzie używa się jej do wyświetlania wartości zmiennej.

Pamiętaj, że MsgBox jest funkcją, stąd po jej wywołaniu realizacja procedury zostaje wstrzymana aż do momentu, kiedy użytkownik zamknie okno dialogowe.



Kiedy w oknie dialogowym zostanie wyświetlony komunikat, możesz skopiować jego treść do schowka systemowego, naciskając po prostu kombinację klawiszy *Ctrl+C*.

Oficjalna składnia funkcji MsgBox jest następująca:

```
MsgBox(komunikat[, przyciski][, tytuł][, plik_pomocy, kontekst])
```

- **komunikat** (wymagany) — tekst wyświetlany w oknie komunikatu;
- **przyciski** (opcjonalne) — wyrażenia numeryczne decydujące o tym, które przyciski i ikony pojawią się w oknie komunikatu (patrz tabela 10.2);

Tabela 10.2. Stałe odpowiadające przyciskom obsługiwany przez funkcję MsgBox

Stała	Wartość	Opis
vbOKOnly	0	Wyświetla jedynie przycisk <i>OK</i> .
vbOKCancel	1	Wyświetla przyciski <i>OK</i> i <i>Anuluj</i> .
vbAbortRetryIgnore	2	Wyświetla przyciski <i>Przerwij</i> , <i>Ponów próbę</i> i <i>Ignoruj</i> .
vbYesNoCancel	3	Wyświetla przyciski <i>Tak</i> , <i>Nie</i> i <i>Anuluj</i> .
vbYesNo	4	Wyświetla przyciski <i>Tak</i> i <i>Nie</i> .
vbRetryCancel	5	Wyświetla przyciski <i>Ponów próbę</i> i <i>Anuluj</i> .
vbCritical	16	Wyświetla ikonę komunikatu krytycznego.
vbQuestion	32	Wyświetla ikonę pytania.
vbExclamation	48	Wyświetla ikonę komunikatu ostrzegawczego.
vbInformation	64	Wyświetla ikonę komunikatu informacyjnego.
vbDefaultButton1	0	Domyślny jest pierwszy przycisk.
vbDefaultButton2	256	Domyślny jest drugi przycisk.
vbDefaultButton3	512	Domyślny jest trzeci przycisk.
vbDefaultButton4	768	Domyślny jest czwarty przycisk.
vbSystemModal	4096	Wszystkie aplikacje są wstrzymywane do momentu uzyskania od użytkownika odpowiedzi na wyświetcone okno komunikatu (może nie działać w każdej sytuacji).
vbMsgBoxHelpButton	16384	Wyświetla przycisk <i>Pomoc</i> (aczkolwiek nie ma żadnego sposobu na wyświetlenie pomocy po naciśnięciu tego przycisku (!)).

- *tytuł* (opcjonalny) — tytuł okna komunikatu;
- *plik pomocy, kontekst* (opcjonalne) — plik i temat pomocy.

Elastyczność argumentu *przyciski* pozwala z łatwością dostosowywać okna komunikatów. Tabela 10.2 zawiera wiele stałych, które mogą być użyte w przypadku tego argumentu. Możesz określić, które przyciski się pojawią, czy będzie widoczna ikona i jaki przycisk będzie domyślny.

Funkcja MsgBox może zostać użyta samodzielnie (w celu wyświetlenia jedynie komunikatu) lub jej wynik może być przypisany zmiennej. Gdy funkcja MsgBox zwróci wynik, reprezentuje on przycisk kliknięty przez użytkownika. Poniższa przykładowa procedura wyświetla komunikat oraz przycisk *OK* i nie zwraca żadnego wyniku:

```
Sub MsgBoxDemo()
    MsgBox "Makro zakończyło działanie bez błędów."
End Sub
```

Zwróć uwagę, że argument wywołania funkcji MsgBox w powyższym przykładzie nie został ujęty w nawiasy. Dzieje się tak, ponieważ w tym przypadku funkcja MsgBox nie jest przypisana do żadnej zmiennej.

Aby pobrać odpowiedź użytkownika udzieloną w oknie komunikatu, należy wyniki działania funkcji MsgBox przypisać zmiennej. W poniższym kodzie źródłowym zastosowano dwie z wbudowanych stałych omówionych w tabeli 10.3, które ułatwiają przetwarzanie wartości zwracanych przez funkcję MsgBox:

```
Sub GetAnswer()
    Dim Ans As Integer
    Ans = MsgBox("Czy kontynuować?", vbYesNo)
    Select Case Ans
        Case vbYes
            ' [...] kod wykonywany, gdy zostanie naciśnięty przycisk Tak...
        Case vbNo
            ' [...] kod wykonywany, gdy zostanie naciśnięty przycisk Nie...
    End Select
End Sub
```

Tabela 10.3. Stałe stosowane jako wartości zwracane przez funkcję MsgBox

Stała	Wartość	Kliknięty przycisk
vbOK	1	<i>OK</i>
vbCancel	2	<i>Anuluj</i>
vbAbort	3	<i>Przerwij</i>
vbRetry	4	<i>Ponów próbę</i>
vbIgnore	5	<i>Ignoruj</i>
vbYes	6	<i>Tak</i>
vbNo	7	<i>Nie</i>

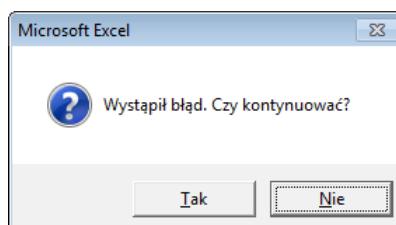
Zmienna zwracana przez funkcję MsgBox jest typu Integer. Aby dalej przetwarzarć wynik działania funkcji MsgBox, nie musisz koniecznie stosować zmiennej. Procedura przedstawiona poniżej jest zmodyfikowaną wersją procedury GetAnswer.

```
Sub GetAnswer2()
    If MsgBox("Czy chcesz kontynuować?", vbYesNo) = vbYes Then
        ' ...kod wykonywany, gdy zostanie naciśnięty przycisk Tak...
    Else
        ' ...kod wykonywany, gdy zostanie naciśnięty przycisk Nie...
    End If
End Sub
```

Aby w oknie komunikatu zostały wyświetlane przyciski *Tak* i *Nie* oraz ikona pytajnika, w poniższym przykładzie zastosowałem kombinację stałych. Drugi przycisk spełnia funkcję przycisku domyślnego (patrz rysunek 10.6). Dla uproszczenia stałe przypisane zostały do zmiennej Config.

```
Private Function ContinueProcedure() As Boolean
    Dim Config As Integer
    Dim Ans As Integer
    Config = vbYesNo + vbQuestion + vbDefaultButton2
    Ans = MsgBox("Wystąpił błąd. Czy kontynuować?", Config)
    If Ans = vbYes Then ContinueProcedure = True _
    Else ContinueProcedure = False
End Function
```

Rysunek 10.6.
Argument przyciski
funkcji MsgBox
określa, jakie przyciski
pojawią się w oknie



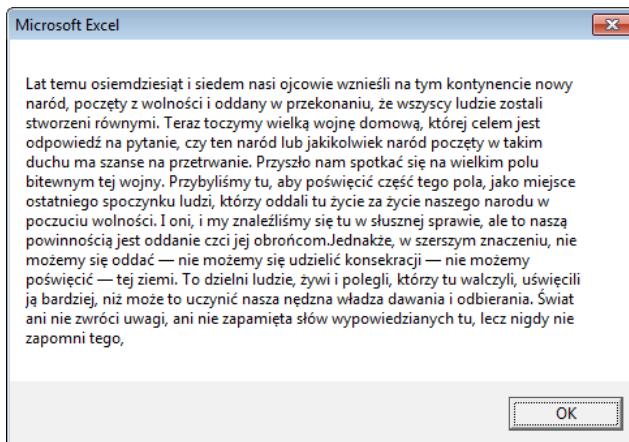
Funkcja ContinueProcedure może zostać wywołana z innej procedury. Przykładowo poniższa instrukcja wywołuje funkcję ContinueProcedure (która wyświetla okno komunikatu). Jeżeli funkcja zwróci wartość False (użytkownik naciągnął przycisk *Nie*), procedura zakończy działanie. W przeciwnym razie zostanie wykonana następna instrukcja.

```
If Not ContinueProcedure() Then Exit Sub
```

Szerokość okna dialogowego zależy od aktualnie używanej rozdzielczości karty graficznej. Na rysunku 10.7 przedstawiono wygląd okna dialogowego wyświetlającego długi tekst bez wymuszonych podziałów wierszy¹.

¹ Tekst wyświetlony w oknie dialogowym to fragment słynnej przemowy prezydenta USA Abrahama Lincoln, wygłoszonej 19 listopada 1863 roku podczas uroczystości na Narodowym Cmentarzu Gettysburgskim — przyp. tłum.

Rysunek 10.7.
Wyświetlanie długiego komunikatu w oknie dialogowym



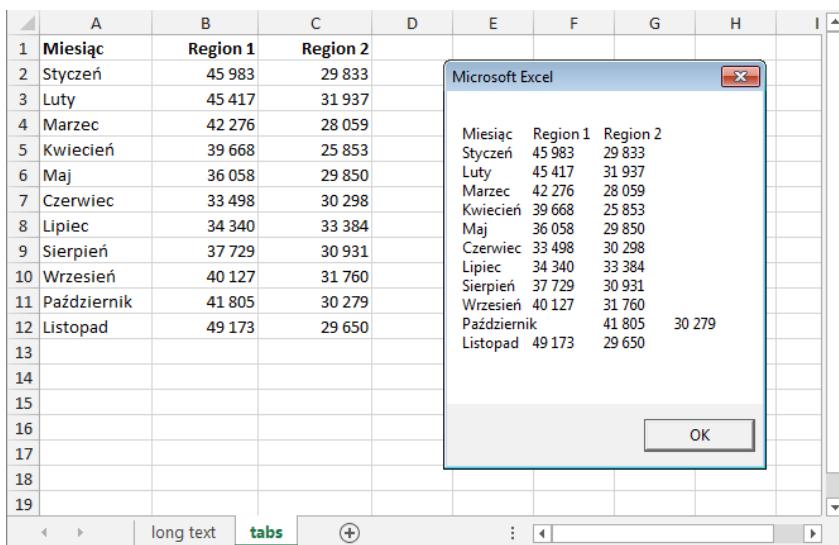
Jeżeli chcesz wstawić w komunikacie znak podziału wiersza, powinieneś użyć w tekście stałej vbCrLf lub vbCrLf. Poniższy przykład wyświetla komunikat umieszczony w trzech wierszach.

```
Sub Multiline()
    Dim Msg As String
    Msg = "To jest pierwszy wiersz." & vbCrLf & vbCrLf
    Msg = Msg & "To jest drugi wiersz." & vbCrLf
    Msg = Msg & "To jest ostatni wiersz."
    MsgBox Msg
End Sub
```

Przy użyciu stałej vbTab można też wstawić znak tabulacji. Procedura przedstawiona poniżej używa okna komunikatu do wyświetlenia wartości zakresu A1:C13, złożonego z 13×3 komórek (patrz rysunek 10.8). Procedura oddziela kolumny przy użyciu stałej vbTab i wstawia nowy wiersz, korzystając ze stałej vbCrLf. Funkcja MsgBox akceptuje łańcuch o maksymalnej długości wynoszącej 1023 znaki, co stanowi ograniczenie dla liczby komórek, których zawartość może zostać wyświetlona. Zauważ również, że poszczególne tabulatory są wyrównane, stąd jeżeli w komórce znajduje się więcej niż 11 znaków, kolumny nie będą wyrównane (patrz wiersz Październik).

```
Sub ShowRange()
    Dim Msg As String
    Dim r As Integer, c As Integer
    Msg = ""
    For r = 1 To 12
        For c = 1 To 3
            Msg = Msg & Cells(r, c).Text
            If c >< 3 Then Msg = Msg & vbCrLf
        Next c
        Msg = Msg & vbCrLf
    Next r
    MsgBox Msg
End Sub
```

Przykłady omawiane w tej sekcji znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>), w skoroszytce o nazwie *Przykłady MsgBox.xlsx*.



The screenshot shows a Microsoft Excel spreadsheet with data in columns A, B, and C. A custom dialog box titled "Microsoft Excel" is overlaid on the spreadsheet. The dialog box contains the same data as the spreadsheet, with columns labeled "Miesiąc", "Region 1", and "Region 2". An "OK" button is at the bottom of the dialog box.

	A	B	C	D	E	F	G	H	I
1	Miesiąc	Region 1	Region 2						
2	Styczeń	45 983	29 833						
3	Luty	45 417	31 937						
4	Marzec	42 276	28 059						
5	Kwiecień	39 668	25 853						
6	Maj	36 058	29 850						
7	Czerwiec	33 498	30 298						
8	Lipiec	34 340	33 384						
9	Sierpień	37 729	30 931						
10	Wrzesień	40 127	31 760						
11	Październik	41 805	30 279						
12	Listopad	49 173	29 650						
13									
14									
15									
16									
17									
18									
19									

Rysunek 10.8. Okno komunikatu wyświetlające tekst sformatowany przy użyciu znaków tabulacji i znaków podziału wiersza



W rozdziale 13. znajdziesz przykład formularza *UserForm* emulującego działanie funkcji *MsgBox*.

Metoda GetOpenFilename programu Excel

Jeżeli aplikacja musi poprosić użytkownika o podanie nazwy pliku, można w tym celu użyć funkcji *InputBox*. Niestety rozwiązanie takie jest niezbyt eleganckie i podatne na błędy. Lepszą propozycją jest zastosowanie metody *GetOpenFilename* obiektu *Application*, która gwarantuje, że aplikacja pobierze prawidłową nazwę pliku, a także jego pełną ścieżkę.

Metoda wyświetla standardowe okno dialogowe *Otwieranie*, ale *nie* otwiera pliku, którego nazwę wybierze użytkownik. Zamiast tego zwraca łańcuch zawierający ścieżkę i nazwę wybranego pliku. Teraz możesz z łatwością napisać procedurę, która będzie w dowolny sposób przetwarzała nazwę pliku.

Składnia metody *GetOpenFilename* jest następująca (wszystkie argumenty są opcjonalne):

`Application.GetOpenFilename(FileFilter, FilterIndex, Title, ButtonText, MultiSelect)`

- **FileFilter** — łańcuch określający kryterium filtrowania plików;
- **FilterIndex** — numery indeksów domyślnego kryterium filtrowania plików;
- **Title** — tytuł okna dialogowego (jeżeli zostanie pominięty, tytułem będzie łańcuch *Otwieranie*);
- **ButtonText** — argument stosowany tylko dla systemów Macintosh;
- **MultiSelect** — jeżeli argument ma wartość *True*, wybranych może być wiele plików; wartością domyślną jest *False*.

Argument `FileFilter` określa, co pojawi się na liście rozwijanej *Pliki typu* okna dialogowego. Argument składa się z łańcuchów identyfikujących filtr plików, za którymi znajdują się symbole filtrujących znaków wieloznacznych, w przypadku których każda część i każda para jest oddzielona przecinkami. Jeżeli wartość argumentu nie zostanie podana, będzie użyta wartość domyślna o następującej postaci:

"Wszystkie pliki (*.*),*.*"

Pierwsza część powyższego łańcucha (*Wszystkie pliki (*.*)*) jest tekstem wyświetlonym na liście rozwijanej *Pliki typu*. Z kolei jego druga część (**.**) określa, jakie pliki zostaną wyświetcone w oknie.

Poniższa instrukcja przypisuje łańcuch tekstu do zmiennej o nazwie `Filt`. Ten łańcuch tekstu może następnie zostać użyty jako argument `FileFilter` metody `GetOpenFilename`. W tym przypadku okno dialogowe umożliwi użytkownikowi wybranie jednego z czterech różnych typów plików (dodatkowo dostępna jest pozycja *Wszystkie pliki*). Przy definiowaniu zmiennej `Filt` posłużyłem się znakami kontynuacji polecenia VBA w następnym wierszu. Dzięki temu łatwiejsze będzie modyfikowanie tego raczej złożonego argumentu.

```
Filt = "Pliki tekstowe (*.txt),*.txt," & _
      "Pliki arkusza kalkulacyjnego firmy Lotus (*.prn),*.prn," & _
      "Pliki używające przecinka jako separatora (*.csv),*.csv," & _
      "Pliki ASCII (*.asc),*.asc," & _
      "Wszystkie pliki (*.*),*.*"
```

Argument `FilterIndex` określa, który filtr plików będzie domyślny (przypisany zmiennej `FileFilter`), natomiast argument `Title` zawiera tekst wyświetlany na pasku tytułu. Jeżeli wartością argumentu `MultiSelect` jest `True`, użytkownik może wybrać wiele plików, które zostaną zwrócone w tablicy.

W poniższym przykładzie użytkownik jest proszony o podanie nazwy pliku. W procedurze zdefiniowano pięć filtrów plików.

```
Sub GetImportFileName()
    Dim Filt As String
    Dim FilterIndex As Integer
    Dim Title As String
    Dim FileName As Variant

    ' Tworzenie listy filtrów plików
    Filt = "Pliki tekstowe (*.txt),*.txt," & _
           "Pliki arkusza kalkulacyjnego firmy Lotus (*.prn),*.prn," & _
           "Pliki używające przecinka jako separatora (*.csv),*.csv," & _
           "Pliki ASCII (*.asc),*.asc," & _
           "Wszystkie pliki (*.*),*.*"

    ' Domyślnie wyświetlany jest filtr *.*
    FilterIndex = 5

    ' Tworzenie tytułu okna dialogowego
    Title = "Wybierz plik, który chcesz importować"

    ' Pobranie nazwy pliku
    FileName = Application.GetOpenFilename( _
               FileFilter:=Filt, _
```

```

        FilterIndex:=FilterIndex, _
        Title:=Title)

    ' Zakończenie pracy, jeżeli w oknie dialogowym zostanie naciśnięty przycisk Anuluj
    If FileName = False Then
        MsgBox "Nie wybrano żadnego pliku."
        Exit Sub
    End If

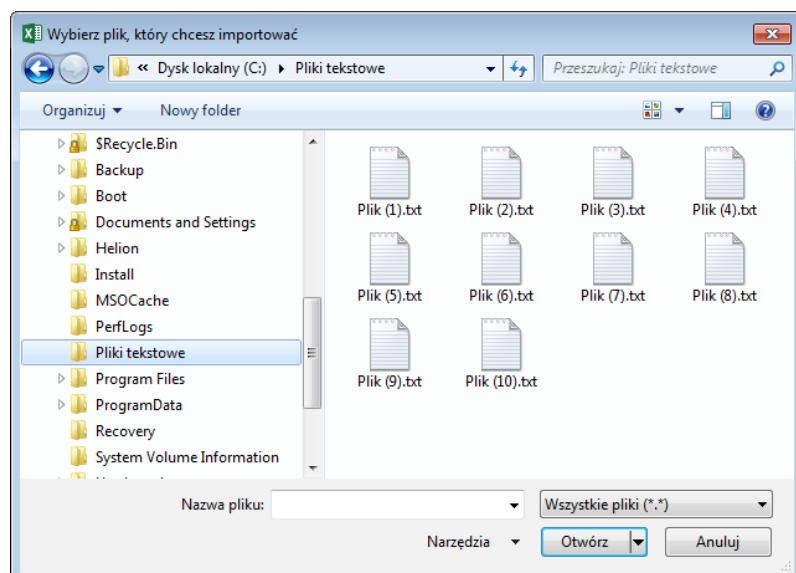
    ' Wyświetlenie pełnej ścieżki i nazwy wskazanego pliku
    MsgBox "Wybrałeś plik: " & FileName
End Sub

```

Na rysunku 10.9 pokazano okno dialogowe pojawiające się po wywołaniu procedury, kiedy użytkownik jako filtr wybierze pliki tekstowe.

Rysunek 10.9.

*Metoda
GetOpenFilename
wyświetla okno
dialogowe
dostosowane
do własnych
wymagań użytkownika*



Kolejny przykład jest podobny do poprzedniego. Różnica polega na tym, że użytkownik może wcisnąć klawisz *Ctrl* lub *Shift* i po otwarciu okna dialogowego wybrać wiele plików. Poprzez określenie, czy zmienna *FileName* jest tablicą, sprawdzamy, czy został naciśnięty przycisk *Anuluj*. Jeżeli użytkownik nie naciśnie przycisku *Anuluj*, wynikiem będzie tablica z przynajmniej jednym elementem. W naszym przykładzie lista wybranych plików jest wyświetlana w oknie komunikatu.

```

Sub GetImportFileName2()
    Dim Filt As String
    Dim FilterIndex As Integer
    Dim FileName As Variant
    Dim Title As String
    Dim i As Integer
    Dim Msg As String

    ' Tworzenie listy filtrów plików
    Filt = "Pliki tekstowe (*.txt),*.txt," & _

```

```

    "Pliki arkusza kalkulacyjnego firmy Lotus (*.prn),*.prn," &
    "Pliki używające przecinka jako separatora (*.csv),*.csv," &
    "Pliki ASCII (*.asc),*.asc," &
    "Wszystkie pliki (*.*),*.*"
    ' Domyślnie wyświetlany jest filtr *.*
    FilterIndex = 5

    ' Tworzenie tytułu okna dialogowego
    Title = "Wybierz plik do zaimportowania"

    ' Pobranie nazwy pliku
    FileName = Application.GetOpenFilename _
        (FileFilter:=Filt,
        FilterIndex:=FilterIndex, _
        Title:=Title, _
        MultiSelect:=True)

    ' Zakończenie pracy, jeżeli w oknie dialogowym zostanie naciśnięty przycisk Anuluj
    If Not IsArray(FileName) Then
        MsgBox "Nie wybrano żadnego pliku."
        Exit Sub
    End If

    ' Wyświetlenie pełnych ścieżek i nazw plików
    For i = LBound(FileName) To UBound(FileName)
        Msg = Msg & FileName(i) & vbCrLf
    Next i
    MsgBox "Wybrałeś następujące pliki:" & vbCrLf & Msg
End Sub

```

Zmienna `FileName` została zdefiniowana przy użyciu typu `Variant`, a nie `String`, jak w poprzednich przykładach. Wynika to stąd, że zmienna `FileName` zamiast pojedynczej nazwy pliku może przechowywać tablicę nazw.



Skoroszyt z tymi przykładami (*Pobieranie nazwy pliku.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Metoda GetSaveAsFilename programu Excel

Metoda `GetSaveAsFilename` jest bardzo podobna do metody `GetOpenFilename`. Wyświetla okno dialogowe *Zapisywanie jako* i umożliwia użytkownikowi wybranie pliku lub podanie jego nazwy. Funkcja zwraca nazwę pliku i jego ścieżkę, ale nie podejmuje żadnych działań. Podobnie jak w przypadku metody `GetOpenFilename` wszystkie argumenty metody `GetSaveAsFilename` są opcjonalne.

Składnia metody `GetSaveAsFilename` jest następująca:

```
Application.GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex, Title,
    ↳ButtonText)
```

Oto jej argumenty:

- `InitialFilename` (opcjonalny) — pozwala określić sugerowaną nazwę pliku (wraz ze ścieżką do pliku);

- **FileFilter** (opcjonalny) — łańcuch określający kryterium filtrowania plików;
- **FilterIndex** (opcjonalny) — numery indeksu domyślnego kryterium filtrowania plików;
- **Title** (opcjonalny) — tytuł okna dialogowego;
- **ButtonText** — argument używany tylko dla systemów Macintosh.

Okno wybierania katalogu

Jeżeli chcesz pobrać nazwę pliku, to najprostszym rozwiązaniem jest zastosowanie metody `GetOpenFilename`, co już zostało omówione w poprzednim podrozdziale. Jeżeli jednak musisz pobrać tylko nazwę wybranego katalogu (a nie nazwę pliku), możesz użyć funkcji `FileDialog` programu Excel.

Procedura przedstawiona poniżej wyświetla okno dialogowe, które pozwala użytkownikowi na wybranie nazwy katalogu. Po naciśnięciu przycisku *OK* procedura wyświetla przy użyciu funkcji `MsgBox` pełną ścieżkę wybranego katalogu. Jeżeli użytkownik naciśnie przycisk *Anuluj*, w oknie komunikatu pojawi się łańcuch *Anulowano*.

```
Sub GetAFolder ()  
    With Application.FileDialog(msoFileDialogFolderPicker)  
        .InitialFileName = Application.DefaultFilePath & "\"  
        .Title = "Wybierz lokalizację kopii zapasowej"  
        .Show  
        If .SelectedItems.Count = 0 Then  
            MsgBox "Anulowano"  
        Else  
            MsgBox .SelectedItems(1)  
        End If  
    End With  
End Sub
```

Obiekt `FileDialog` umożliwia określenie katalogu startowego poprzez ustawienie odpowiedniej wartości właściwości `InitialFilename`. W tym przypadku kod procedury jako katalogu startowego używa domyślnej ścieżki programu Excel.

Wyświetlanie wbudowanych okien dialogowych Excela

Za pomocą kodu programu napisanego w języku VBA możesz wykonywać wiele poleceń ze Wstążki Excela. Jeżeli dane polecenie otwiera okno dialogowe, kod źródłowy umożliwia „wybieranie” lub „modyfikowanie” zawartych w nim opcji (pomimo iż samo okno nie jest wyświetlane). Na przykład wykonanie poniższego polecenia języka VBA odpowiada przejściu na kartę *NARZĘDZIA GŁÓWNE*, wybraniu polecenia *Znajdź i zaznacz*, znajdującego się w grupie opcji *Edytowanie*, następnie wybraniu z menu podręcznego polecenia *Przejdz do*, wpisaniu w oknie dialogowym *Przechodzenie do zakresu A1:C3* i naciśnięciu przycisku *OK*.

```
Application.Goto Reference:="Range("A1:C3")"
```

Zauważ, że samo okno dialogowe *Przechodzenie do* nie pojawi się na ekranie (i właśnie o to nam chodziło).

W niektórych sytuacjach może Ci zależeć na otwarciu jednego z wbudowanych okien dialogowych Excela, tak aby końcowy użytkownik mógł dokonać wyboru. Możesz tego dokonać, tworząc kod wykonujący odpowiednie polecenia ze Wstążki.



Uwaga

Innym znanym sposobem na wyświetlanie wbudowanych okien dialogowych Excela jest zastosowanie kolekcji **Dialogs** obiektu **Application**. Niestety Microsoft nie zadbał o aktualizację tego mechanizmu w nowej wersji Excela, stąd nie będziemy omawiać tego rozwiązania. Metoda, którą opisujemy poniżej, jest znacznie lepszym rozwiązaniem.

W poprzednich wersjach programu Excel programiści mogli przy użyciu obiektu **CommandBar** tworzyć własne menu i paski narzędzi. W Excelu 2007 i nowszych obiekt **CommandBar** jest co prawda nadal dostępny, ale niestety nie działa już tak samo jak w poprzednich wersjach.

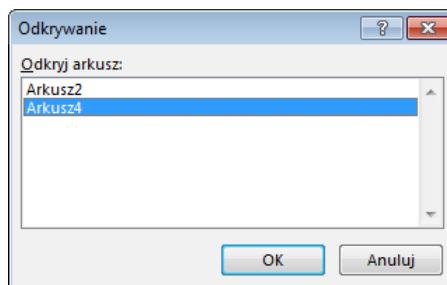


Więcej szczegółowych informacji na temat obiektu **CommandBar** znajdziesz w rozdziałach 20. i 21.

Począwszy od Excela 2007, funkcjonalność obiektu **CommandBar** została rozszerzona o możliwość wykonywania z poziomu języka VBA polecen znajdujących się na Wstążce. Wiele poleceń Wstążki wyświetla na ekranie różne okna dialogowe. Na przykład polecenie przedstawione poniżej wyświetla na ekranie okno dialogowe *Odkrywanie* (patrz rysunek 10.10):

```
Application.CommandBars.ExecuteMso("SheetUnhide")
```

Rysunek 10.10.
To okno dialogowe zostało wyświetlone za pomocą polecenia języka VBA



Pamiętaj, że Twój program w taki sposób nie jest w stanie pozyskać informacji o akcjach podejmowanych przez użytkownika. Na przykład kiedy wykonywane jest powyższe polecenie, nie będziesz mógł w żaden sposób sprawdzić, który arkusz został zaznaczony lub czy użytkownik naciągnął przycisk *Anuluj*. Nie trzeba chyba dodawać, że kod, który wykonuje polecenia związane ze Wstążką, nie jest kompatybilny z wersjami Excela starszymi niż 2007.

Metoda **ExecuteMso** pobiera tylko jeden argument, **idMso**, który reprezentuje wybrany formant Wstążki. Niestety w pomocy systemowej programu Excel nie znajdziemy listy parametrów tej metody.

Bezpośrednie wykonywanie poleceń ze starego menu programu Excel

Istnieje jeszcze inna metoda wyświetlania wbudowanego okna dialogowego, która wymaga pewnej wiedzy na temat pasków narzędzi używanych w poprzednich wersjach programu Excel (przed wersją 2007), oficjalnie nazywanych obiektem klasy CommandBar. Pomimo iż w Excelu obiekty CommandBar nie są już używane, to jednak są nadal obsługiwane ze względu na konieczność zachowania kompatybilności z poprzednimi wersjami programu.

Przykładowo, wykonanie poniższej instrukcji odpowiada wybraniu w wersji Excel 2003 polecenia *Format/Arkusz/Odkryj*:

```
Application.CommandBars("Worksheet Menu Bar").  
    Controls("Format").Controls("Arkusz").  
        Controls("Odkryj...").Execute
```

Po wykonaniu tego polecenia na ekranie pojawi się okno dialogowe *Odkrywanie*. Zwróć uwagę, że nazwy poszczególnych poleceń menu muszą być podawane dokładnie, w pełnym, oryginalnym brzmieniu (na przykład z uwzględnieniem trzech kropek po nazwie polecenia *Unhide*).

A oto kolejny przykład. Polecenie przedstawione poniżej wyświetla na ekranie okno dialogowe *Formatowanie komórek*:

```
Application.CommandBars("Worksheet Menu Bar").  
    Controls("Format").Controls("Komórki...").Execute
```

Mimo pozornej wygody poleganie tylko i wyłącznie na obiektach CommandBar nie wydaje się być zbyt dobrym rozwiązaniem, ponieważ w przyszłości obiekty takie mogą zostać całkowicie usunięte z kolejnych wersji programu Excel.

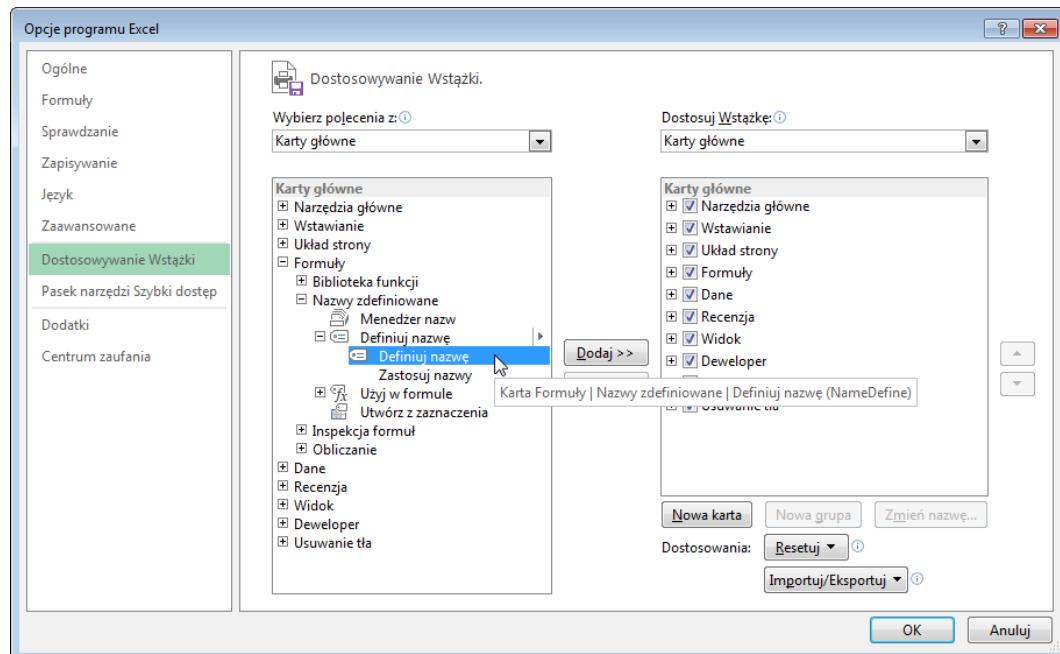
Jeżeli spróbujesz wywołać wbudowane okno dialogowe w niepoprawnym kontekście, Excel wyświetli na ekranie odpowiedni komunikat o wystąpieniu błędu. Na przykład polecenie przedstawione poniżej przywołuje na ekran okno dialogowe *Formatowanie komórek/Liczby*:

```
Application.CommandBars.ExecuteMso ("NumberFormatsDialog")
```

Jeżeli spróbujesz wykonać polecenie przedstawione powyżej w nieodpowiednim momencie (na przykład kiedy aktualnie zaznaczonym elementem nie jest komórka, ale kształt), Excel wyświetli na ekranie komunikat o wystąpieniu błędu. Dzieje się tak, ponieważ to okno dialogowe działa poprawnie tylko i wyłącznie w kontekście komórek arkusza.

Excel ma tysiące różnych poleceń. Jak zatem możesz odnaleźć nazwę polecenia, które jest Ci potrzebne? Jednym ze sposobów jest użycie karty *Dostosowywanie Wstążki* okna dialogowego *Opcje programu Excel* (aby je przywołać na ekran, kliknij dowolne miejsce Wstążki prawym przyciskiem myszy i z menu podręcznego, które pojawi się na ekranie, wybierz polecenie *Dostosuj Wstążkę*). Praktycznie każde polecenie dostępne w Excelu jest wymienione na liście znajdującej się w lewym panelu okna. Aby znaleźć nazwę wybranego polecenia, odszukaj je na liście i ustawi nad nim wskaźnik myszy — po chwili pojawi się podpowiedź ekranowa zawierająca nazwę polecenia. Przykład został przedstawiony na rysunku 10.11 — w tym przypadku sprawdzaliśmy nazwę polecenia, które przywołuje na ekran okno dialogowe *Nowa nazwa*:

```
Application.CommandBars.ExecuteMso ("NameDefine")
```



Rysunek 10.11. Zastosowanie karty Dostosowywanie Wstążki do identyfikacji nazwy polecenia

Wyświetlanie formularza danych

Wielu ludzi używa Excela do zarządzania listami danych, w których informacje są poukładowane w kolejnych wierszach. Excel ułatwia pracę z tego rodzaju danymi poprzez zastosowanie wbudowanego, prostego i tworzonego automatycznie formularza, pozwalającego zarówno na wprowadzanie nowych danych, jak i przeglądanie i modyfikację danych już istniejących. Formularz ten może działać zarówno z normalnymi zakresami danych, jak i zakresami, które zostały przekształcone do postaci tabeli danych (aby to zrobić, przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Tabela*, znajdujący się w grupie opcji *Tabele*). Na rysunku 10.12 przedstawiono przykład takiego formularza w działaniu.

Wyświetlanie formularza wprowadzania danych

Z niewiadomego powodu polecenie wyświetlające formularz wprowadzania danych nie jest bezpośrednio dostępne na Wstążce programu Excel. Aby z niego skorzystać, musisz najpierw samodzielnie umieścić ikonę tego polecenia na pasku narzędzi *Szybki dostęp* lub na Wstążce. Poniżej przedstawiamy sposób postępowania przy dodawaniu ikony tego do paska narzędzi *Szybki dostęp*.

1. Kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Dostosuj pasek narzędzi Szybki dostęp*.
2. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*.

1	Agent	Data	Obszar	Cena	Liczba sypialni	Lazienki	Powierzchnia m ²	Typ	Basen	Sprzedany?
2	Adamek	2007-10-09	Śródmieście	199 000,00 zł	3	2,5	140	Mieszkanie	FAŁSZ	FAŁSZ
3	Adamek	2007-08-19	Śródmieście	214 500,00 zł	4	2,5	173	Dom	PRAWDA	FAŁSZ
4	Adamek	2007-04-28	Śródmieście					Dom	FAŁSZ	FAŁSZ
5	Adamek	2007-07-19	Śródmieście					Dom	PRAWDA	PRAWDA
6	Adamek	2007-02-06	Śródmieście					Dom	PRAWDA	FAŁSZ
7	Adamek	2007-08-01	Śródmieście					Dom	PRAWDA	PRAWDA
8	Adamek	2007-01-15	Śródmieście					Dom	FAŁSZ	PRAWDA
9	Jelonek	2007-01-29	Trynek					Dom	PRAWDA	FAŁSZ
10	Robaczek	2007-04-04	Trynek					Dom	FAŁSZ	FAŁSZ
11	Hirek	2007-02-24	Trynek					Dom	PRAWDA	FAŁSZ
12	Rafalski	2007-04-24	Trynek					Dom	PRAWDA	PRAWDA
13	Adamek	2007-04-21	Sikornik					Dom	FAŁSZ	FAŁSZ
14	Szostak	2007-03-24	Trynek					Dom	FALSZ	FALSZ
15	Krzaczek	2007-06-09	Trynek					Dom	FALSZ	FALSZ
16	Szostak	2007-08-17	Trynek					Dom	FALSZ	FALSZ
17	Adamek	2007-06-06	Trynek					Mieszkanie	FALSZ	FALSZ
18	Adamek	2007-02-08	Trynek					Mieszkanie	FALSZ	PRAWDA
19	Raczek	2007-03-30	Trynek					Dom	FALSZ	PRAWDA
20	Barnes	2007-06-26	Sikornik					Dom	FALSZ	FALSZ
21	Benek	2007-05-12	Śródmieście	229 500,00 zł	4	3	190	Dom	FAŁSZ	PRAWDA
22	Benek	2007-05-09	Śródmieście	549 000,00 zł	4	3	180	Dom	PRAWDA	FAŁSZ
23	Szostak	2007-07-15	Trynek	374 900,00 zł	4	3	365	Dom	FAŁSZ	FAŁSZ
24	Lang	2007-05-03	Trynek	369 900,00 zł	3	2,5	189	Mieszkanie	PRAWDA	FALSZ
25	Robaczek	2007-01-28	Trynek	369 900,00 zł	4	3	185	Mieszkanie	FAŁSZ	PRAWDA
26	Benek	2007-06-26	Sikornik	229 900,00 zł	3	2,5	147	Dom	PRAWDA	FAŁSZ

Arkusz1

Agent: Benek
 Data: 5/12/2007
 Obszar: Śródmieście
 Cena: 229500
 Liczba sypialni: 4
 Lazienki: 3
 Powierzchnia m²: 190
 Typ: Dom
 Basen: FALSE
 Sprzedany?: TRUE
 Cgna za m²: 1,210,35 zł

20 z 125 Nowy Usuń Przywróć Zamknij Kryteria Znajdź poprzedni Znajdź następny

Rysunek 10.12. Niektórzy użytkownicy do wprowadzania danych wolą używać wbudowanych formularzy Excela

3. Z listy rozwijanej *Wybierz polecenia* z wybierz opcję *Polecenia, których nie ma na Wstążce*.
4. Na liście poniżej (w lewej części okna) odszukaj i zaznacz polecenie *Formularz*.
5. Naciśnij przycisk *Dodaj*. Przycisk wybranego polecenia zostanie dodany do paska narzędzi *Szybki dostęp*.
6. Naciśnij przycisk *OK*, aby zamknąć okno dialogowe *Opcje programu Excel*.

Po wykonaniu tych poleceń na pasku narzędzi *Szybki dostęp* pojawi się ikona nowego polecenia.

Aby skorzystać z formularza wprowadzania danych, musisz tak zorganizować dane w arkuszu, aby Excel mógł je rozpoznać jako tabelę. Rozpocznij od utworzenia w pierwszym wierszu zakresu danych nagłówków opisujących zawartość poszczególnych kolumn. Następnie zaznacz dowolną komórkę tabeli i naciśnij przycisk *Formularz* znajdujący się na pasku narzędzi *Szybki dostęp*. Na ekranie pojawi się okno dialogowe zawierające formularz dostosowany do struktury danych w tabeli. Do przechodzenia pomiędzy poszczególnymi polami możesz użyć klawisza *Tab*. Jeżeli dana komórka zawiera formułę, wynik jej działania pojawia się na formularzu jako tekst, a nie jako pole edytowalne (inaczej mówiąc, za pomocą formularza wprowadzania danych nie możesz modyfikować formuł).

Po zakończeniu wprowadzania danych do bieżącego rekordu naciśnij przycisk *Nowy*. Excel zapisze bieżące dane w arkuszu i wyczyści wszystkie pola formularza, umożliwiając Ci w ten sposób wprowadzenie nowego rekordu danych.

Wyświetlanie formularza wprowadzania danych za pomocą VBA

Do wyświetlenia formularza wprowadzania danych na ekranie możesz użyć metody ShowDataForm, która wymaga jedynie, aby w momencie wywołania metody komórka aktywna znajdowała się w obrębie tabeli danych.

Procedura przedstawiona poniżej wyświetla na ekranie formularz wprowadzania danych.

```
Sub DisplayDataForm()
    ActiveSheet.ShowDataForm
End Sub
```

Takie makro będzie działać poprawnie nawet wtedy, kiedy polecenie *Formularz* nie zostało dodane do Wstążki ani paska narzędzi *Szybki dostęp*.



Skoroszyt z tym przykładem (*Formularz wprowadzania danych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Jeżeli podoba Ci się idea utworzenia uniwersalnego formularza ogólnego przeznaczenia, służącego do wprowadzania danych, możesz wypróbować dodatek *Enhanced Data Form*, który napisałem w języku VBA. Dodatek możesz pobrać bezpośrednio z mojej strony internetowej <http://spreadsheetpage.com/index.php/dataform/>.

Rozdział 11.

Wprowadzenie do formularzy UserForm

W tym rozdziale:

- Tworzenie, wyświetlanie i zamykanie formularzy *UserForm*
- Przegląd formantów formularzy *UserForm*
- Ustawianie właściwości formantów formularzy *UserForm*
- Sterowanie formularzami *UserForm* z poziomu procedur VBA
- Przykład tworzenia formularzy *UserForm*
- Wprowadzenie do zdarzeń związanych z formularzami *UserForm* i formantami
- Dostosowywanie okna *Toolbox* do własnych wymagań
- Lista kontrolna tworzenia i testowania formularzy *UserForm*

Jak Excel obsługuje niestandardowe okna dialogowe

Excel pozwala w stosunkowo prosty sposób tworzyć w aplikacjach niestandardowe okna dialogowe. Tak naprawdę możesz powieść wygląd i sposób obsługi prawie wszystkich okien dialogowych tego arkusza kalkulacyjnego. Niestandardowe okna dialogowe twozone są przy użyciu formularza *UserForm* z wykorzystaniem edytora VBE.

Typowy proces tworzenia formularza *UserForm* składa się z następujących kroków:

1. Wstawienie do projektu nowego formularza *UserForm*.
2. Umieszczenie formantów na formularzu *UserForm*.
3. Ustawienie właściwości formantów formularza.
4. Utworzenie procedur obsługi zdarzeń dla formantów znajdujących się na formularzu.

Procedury obsługi zdarzeń znajdują się w module kodu formularza *UserForm* i są wywoływanne po zaistnieniu wybranych zdarzeń związanych z formularzem (na przykład po naciśnięciu przycisku przez użytkownika).

5. Utworzenie procedury wyświetlającej formularz *UserForm*.

Procedura taka powinna znajdować się w standardowym module VBA (nie w kodzie modułu związanego z formularzem).

6. Umożliwienie użytkownikowi łatwego uruchomienia procedury utworzonej w punkcie 5.

Możesz tego dokonać poprzez utworzenie przycisku na arkuszu, polecenia na Wstążce itp.

Wstawianie nowego formularza *UserForm*

Aby wstawić nowy formularz *UserForm*, powinieneś uruchomić edytor Visual Basic (możesz to zrobić, naciskając kombinację klawiszy *Alt+F11*), a następnie w oknie *Project* zaznaczyć projekt skoroszytu i z menu *Insert* wybrać polecenie *UserForm*. Kolejnym utworzonym formularzom *UserForm* są nadawane domyślne nazwy *UserForm1*, *UserForm2* itd.



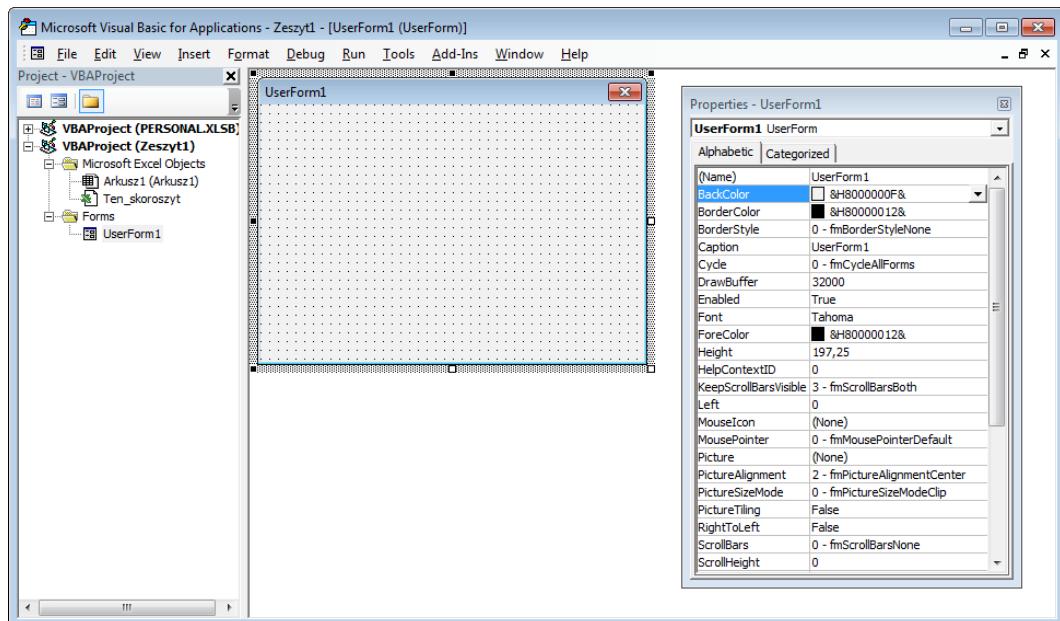
Aby ułatwić identyfikację formularza *UserForm*, możesz zmienić jego domyślną nazwę. W tym celu należy wybrać formularz i przy użyciu okna *Properties* zmienić wartość właściwości *Name* (jeżeli okno *Properties* nie jest widoczne, naciśnij klawisz *F4*). Na rysunku 11.1 pokazano wygląd okna *Properties* po wybraniu pustego formularza *UserForm*.

Skoroszyt może zawierać dowolną liczbę formularzy *UserForm*, z których każdy będzie przechowywać pojedyncze niestandardowe okno dialogowe.

Dodawanie formantów do formularza *UserForm*

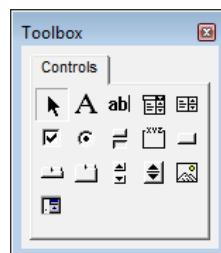
Aby dodać formanty do formularza *UserForm*, należy użyć okna *Toolbox*, które zostało przedstawione na rysunku 11.2 (edytor VBA nie ma menu z poleceniami pozwalającymi na wstawianie formantów). Jeżeli okno *Toolbox* nie jest widoczne, z menu *View* wybierz polecenie *Toolbox*. Okno *Toolbox* jest „pływające”, stąd możesz je przenosić w dowolne miejsce ekranu (w zależności od potrzeb).

Aby dodać wybrany formant o domyślnym rozmiarze, wystarczy kliknąć w oknie *Toolbox* przycisk reprezentujący żądaną formant, a następnie kliknąć lewym przyciskiem myszy w obszarze tworzonego okna dialogowego. Zamiast tego możesz kliknąć wybrany formant, a następnie przeciągnąć kurSOR myszy w obszarze okna dialogowego i „narysować” formant o żądanym rozmiarze.



Rysunek 11.1. Okno Properties pustego formularza UserForm

Rysunek 11.2.
Okno Toolbox
umożliwia dodawanie
formantów
do formularza
UserForm



Nowemu formantowi zostanie nadana domyślna nazwa złożona z identyfikatora typu formantu i kolejnej wartości liczbowej. Jeżeli na przykład do pustego formularza *UserForm* zostanie dodany formant typu *CommandButton*, jego domyślną nazwą będzie *CommandButton1*. Jeżeli następnie zostanie dodany drugi tego typu formant, otrzyma nazwę *CommandButton2*.



Zmiana domyślnych nazw wszystkich formantów, które będą przetwarzane przy użyciu instrukcji kodu źródłowego języka VBA, jest bardzo dobrym rozwiązaniem. Dzięki temu zamiast używać ogólnych nazw w stylu *ListBox1*, będzie można odwoływać się do bardziej zrozumiałych nazw, takich jak na przykład *ListaProduktów*. Aby zmienić nazwę formantu, powinieneś posłużyć się oknem *Properties* edytora VBE. Aby to zrobić, po prostu zaznacz wybrany obiekt i wprowadź nową nazwę.

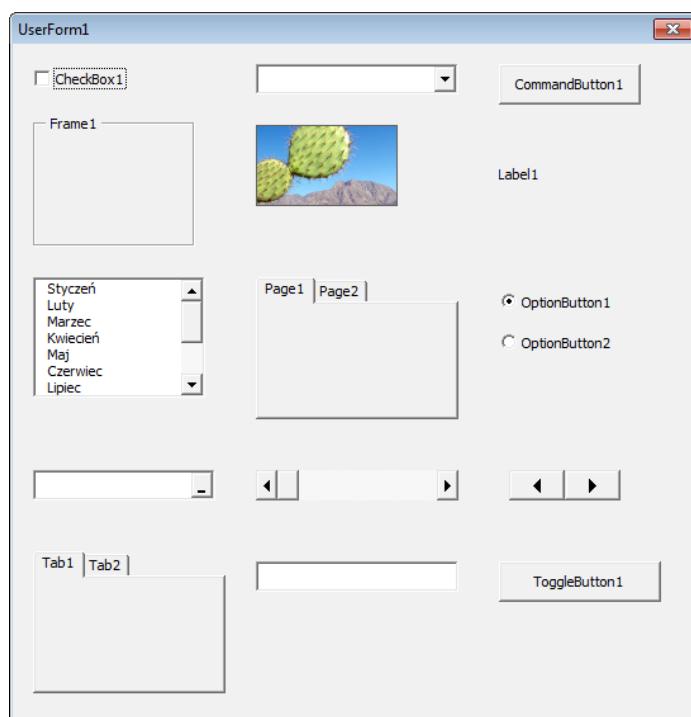
Formanty okna Toolbox

W kolejnych podrozdziałach omówimy formanty dostępne w oknie *Toolbox*.



W sieci

Rysunek 11.3.
Na tym formularzu
UserForm umieszczono
po jednym
z 15 dostępnych
rodzajów formantów



W formularzach *UserForm* możesz również używać formantów ActiveX, które nie są częścią Excela. Więcej szczegółowych informacji na ten temat znajdziesz w podrozdziale „Dostosowywanie okna Toolbox do własnych wymagań” w dalszej części tego rozdziału.

Formant CheckBox

Formant *CheckBox* (pole wyboru) jest używany w przypadku pobierania wartości dwustanowych typu *tak* lub *nie*, *prawda* lub *fałsz*, *włączony* lub *wyłączony* itd. Zaznaczony formant *CheckBox* ma wartość *True*, natomiast wyłączony — wartość *False*.

Formant ComboBox

Formant *ComboBox* (pole kombi) to połączenie pola tekstowego z rozwijaną listą elementów. W danej chwili wyświetlana jest tylko jedna pozycja listy. W przeciwnieństwie do formantu *ListBox* pole kombi pozwala użytkownikowi na wprowadzenie wartości, której nie ma na liście elementów.

Formant CommandButton

W każdym oknie dialogowym zazwyczaj znajduje się przynajmniej jeden formant typu CommandButton (przycisk polecenia). Najczęściej używane formanty typu CommandButton to przyciski o nazwie *OK* oraz *Anuluj*.

Formant Frame

Formant Frame (pole grupy) to kontener dla innych formantów. Formanty są zwykle grupowane ze względów estetycznych lub w celu pogrupowania zbioru logicznie ze sobą powiązanych formantów. Formant Frame jest szczególnie przydatny, gdy okno dialogowe zawiera więcej niż jeden zbiór formantów OptionButton.

Formant Image

Formant Image (obraz) służy do wyświetlanego obrazu pobieranego z pliku lub wklejanego ze schowka. Formant taki może zostać użyty w oknie dialogowym na przykład do wyświetlenia logo firmy. Sam obraz jest przechowywany w skoroszycie, dzięki czemu w sytuacji, gdy skoroszyt zostanie przesłany komuś innemu, nie będziesz musiał dodać kopii pliku graficznego.



Niektóre pliki graficzne są bardzo duże, dlatego ich użycie może znacznie zwiększyć rozmiar skoroszytu. Dla uzyskania najlepszych wyników użycie obrazów powinno być starannie przemyślane; najlepiej używać obrazów o niewielkich rozmiarach.

Formant Label

Formant Label (etykieta) po prostu wyświetla tekst w oknie dialogowym.

Formant ListBox

Formant ListBox (pole listy) prezentuje listę pozycji, które użytkownik może wybrać (jedna lub wiele). Formanty typu ListBox są bardzo elastyczne w zastosowaniu. Na przykład można określić zakres zawierający poszczególne elementy listy, który może składać się z wielu kolumn. Elementy listy formantu ListBox mogą też zostać zdefiniowane przy użyciu instrukcji języka VBA.

Formant MultiPage

Formant MultiPage umożliwia tworzenie okien dialogowych wyposażonych w karty, takich jak na przykład okno *Formatowanie komórek*. Domyslnie formant MultiPage posiada dwie karty, ale w razie potrzeby możesz dodać dowolną liczbę dodatkowych kart.

Formant OptionButton

Formant OptionButton (przycisk opcji) przydaje się, gdy użytkownikowi zależy na wybraniu jednej opcji spośród kilku dostępnych. Formant ten zawsze ma postać grupy złożonej z przynajmniej dwóch opcji. Po zaznaczeniu jednego formantu OptionButton w danej grupie z pozostałych formantów zaznaczenie jest automatycznie usuwane.

Jeżeli formularz *UserForm* zawiera więcej niż jedną grupę formantów OptionButton, formanty każdej grupy muszą współdzielić unikatową wartość właściwości *GroupName*, w przeciwnym razie wszystkie formanty OptionButton staną się częścią tej samej grupy. Inne rozwiązanie polega na umieszczeniu formantów OptionButton w grupie Frame, która automatycznie przydzieli poszczególne formanty do tej grupy.

Formant RefEdit

Formant RefEdit jest używany w sytuacji, kiedy użytkownik musi zaznaczyć zakres komórek arkusza. Formant ten pozwala na wprowadzenie adresu komórki bądź wskazanie przy użyciu myszy zakresu komórek arkusza.

Formant ScrollBar

Formant ScrollBar (pasek przewijania) jest podobny do formantu SpinButton. Różnica polega na tym, że w celu zmiany wartości formantu ScrollBar w większym zakresie użytkownik może przeciągać jego suwak. Formant ScrollBar jest najbardziej użyteczny w przypadku wybierania wartości, która zawiera się w szerokim zakresie możliwych wartości.

Formant SpinButton

Formant SpinButton (pokrętło) umożliwia użytkownikowi wybranie wartości poprzez kliknięcie dwóch strzałek. Pierwsza z nich zwiększa wartość, natomiast druga zmniejsza. Formant SpinButton często jest stosowany w połączeniu z formantami TextBox lub Label, które wyświetlają aktualną wartość związaną z tym formantem. Formanty SpinButton mogą mieć orientację poziomą lub pionową.

Formant TabStrip

Formant TabStrip jest podobny do formantu MultiPage, ale w przeciwieństwie do niego nie spełnia funkcji kontenera dla innych obiektów. Jest również bardziej skomplikowany i znacznie mniej uniwersalny.

Formant TextBox

Formant TextBox (pole tekstowe) umożliwia użytkownikowi wprowadzanie tekstu.

Formant ToggleButton

Formant ToggleButton (przycisk przełącznika) posiada dwa stany — włączony lub wyłączony. Kliknięcie przycisku powoduje przełączanie pomiędzy dwoma stanami. Dodatkowo zmienia się wygląd przycisku. Wartością formantu jest True (przycisk wciśnięty) lub False (przycisk nie jest wciśnięty). Osobiście staram się nie korzystać z tego formantu — po prostu moim skromnym zdaniem zastosowanie formantów typu CheckBox jest znacznie lepszym i bardziej przejrzystym rozwiązaniem.

Używanie formantów w arkuszu

Wiele formantów formularzy *UserForm* może być osadzanych bezpośrednio na arkuszu. Listę takich formantów znajdziesz po przejściu na kartę *Deweloper* i wybraniu polecenia *Wstaw* znajdującego się w grupie opcji *Formanty*. Osadzanie formantów bezpośrednio na arkuszu wymaga znacznie mniej pracy niż tworzenie formularzy *UserForm*. Dodatkowo nie jest konieczne tworzenie makr, ponieważ formanty mogą zostać połączone bezpośrednio z komórką arkusza. Jeżeli na przykład w arkuszu zostanie umieszczony formant CheckBox, w celu przypisania go do określonej komórki arkusza powinieneś ustawić wartość właściwości *LinkedCell*. Po zaznaczeniu formantu CheckBox w połączonej z nim komórce pojawi się wartość PRAWDA. Po usunięciu zaznaczenia formantu CheckBox, w komórce pojawi się wartość FAŁSZ.

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt *Formanty ActiveX arkusza.xlsx*, z przykładami formantów osadzonych na arkuszu. Formanty osadzone na arkuszu są połączone bezpośrednio z komórkami arkusza, stąd skoroszyt nie zawiera żadnych makr.

The screenshot shows an Excel spreadsheet with a UserForm calculator on the left and its corresponding linked cells on the right.

Kalkulator kredytowy (UserForm):

- Wartość zakupu: 345 000,00 zł
- Oplata za udzielenie kredytu (5000,-zł)
- Wkład własny:
 - 10%
 - 15%
 - 20%
- Rodzaj kredytu:
 - 30 lat, stała rata
 - 15 lat, stała rata
- Kwota kredytu: 315 500,00 zł
- Miesięczna rata: 3 399,06 zł
- Harmonogram spłaty

Powiązane komórki:

1263 Oprocentowanie (dane z suwaka)	12,63 Oprocentowanie (dane z suwaka)
PRAWDA	Oplata za udzielenie kredytu
315 500,00 zł	Kwota kredytu
PRAWDA	30 lat, stała rata
FAŁSZ	15 lat, stała rata
30 Wybrany okres spłaty	
PRAWDA	10% wkład własny
FAŁSZ	15% wkład własny
FAŁSZ	20% wkład własny
10% wkład własny	

Osadzanie formantów bezpośrednio na arkuszu może okazać się nieco zagmatwane, ponieważ takie formanty mogą pochodzić z dwóch źródeł:

- **Formanty formularza** — to obiekty, które możesz wstawić bezpośrednio na arkusz.
- **Formanty ActiveX** — to podzbior formantów, które są dostępne podczas tworzenia formularzy *UserForm*.

Możesz używać formantów z obu źródeł, ale powinieneś wiedzieć, jakie są między nimi różnice. Formanty formularza działają w nieco inny sposób niż formanty ActiveX.

Gdy do arkusza jest dodawany formant ActiveX, Excel przełącza się w *tryb projektowania*. W przypadku tego trybu możesz modyfikować właściwości dowolnego formantu umieszczonego w arkuszu, dodawać lub edytować procedury obsługi zdarzeń powiązanych z formantami bądź zmieniać wielkość lub położenie formantów. Aby wyświetlić okno właściwości formantu ActiveX, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Właściwości* znajdujący się w grupie opcji *Formanty*.

W przypadku tworzenia prostych przycisków często używam formantu formularza *Przycisk* (Button), ponieważ można do niego przypisać dowolne makro. Jeżeli zamiast tego zastosujemy formant ActiveX *Przycisk polecenia* (CommandButton), po jego naciśnięciu zostanie wykonana powiązana z nim procedura obsługi zdarzenia (mająca np. nazwę CommandButton1_Click), której kod źródłowy znajduje się w module obiektu Arkusz — do takiego formantu nie możesz bezpośrednio przypisać dowolnego makra.

Kiedy Excel znajduje się w trybie projektowania, nie możesz testować działania formantów. Aby wyłączyć tryb projektowania, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Tryb projektowania* znajdujący się w grupie opcji *Formanty* (przycisk ten działa jak przełącznik).

Modyfikowanie formantów formularza UserForm

Po osadzeniu na formularzu *UserForm* wybranego formantu możesz zmieniać jego wielkość i położenie za pomocą standardowych technik myszy.



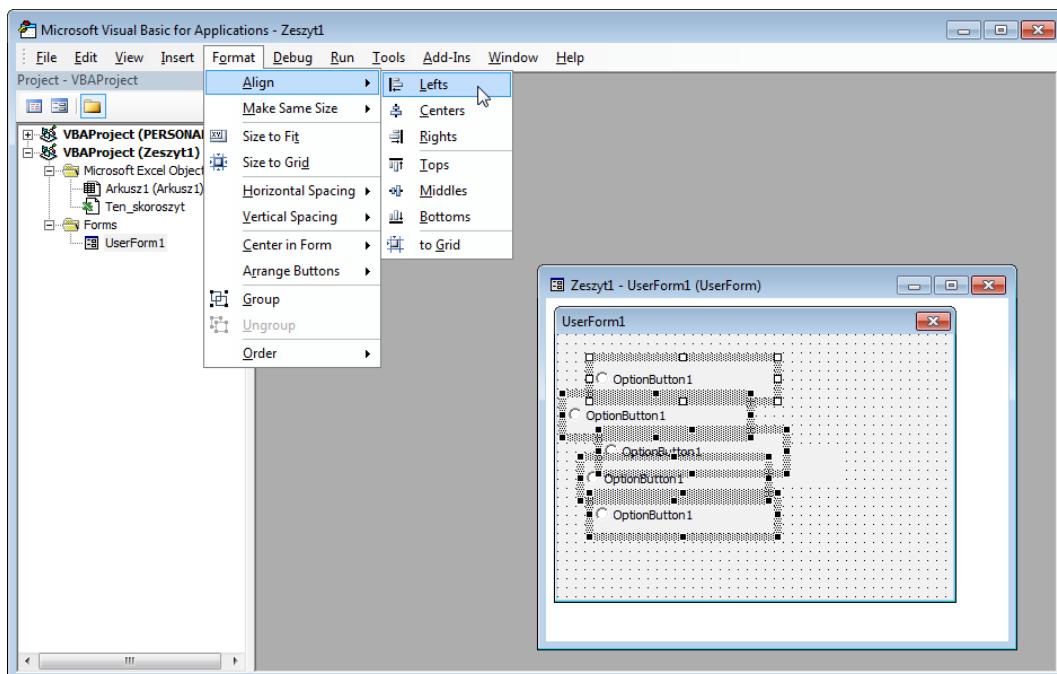
W razie potrzeby możesz zaznaczyć wiele formantów jednocześnie. W tym celu powinieneś wcisnąć klawisz *Shift* i kliknąć wybrane formanty lewym przyciskiem myszy lub naciągnąć i przytrzymać przycisk myszy, a następnie „obrysować” grupę formantów, które chcesz zaznaczyć.

Formularz *UserForm* może zawierać poziome i pionowe linie siatki złożone z kropek, które pomagają przy wyrównywaniu osadzonych na nim formantów. Podczas dodawania lub przemieszczania formantów *przylegają* do linii siatki, co ułatwia wyrównanie ich w stosunku do innych formantów. Linie siatki można wyłączyć, wybierając z menu *Tools* edytora VBE polecenie *Options*. W oknie dialogowym *Options* przejdź na kartę *General* i w grupie *Form Grid Settings* wybierz żądane opcje.

Menu *Format* edytora Visual Basic oferuje kilka poleceń pomocnych w precyzyjnym wyrównaniu formantów w oknie dialogowym i ustawnieniu odległości pomiędzy nimi. Przed użyciem tych poleceń należy zaznaczyć formanty, które będą modyfikowane. Poszczególne polecenia działają zgodnie z oczekiwaniemi, dlatego nie będziemy ich tutaj objaśniać. Na rysunku 11.4 przedstawiono okno dialogowe zawierające kilka formantów Option Button, które zostaną wyrównane. Rysunek 11.5 przedstawia wygląd tego samego okna dialogowego po wyrównaniu położenia formantów.



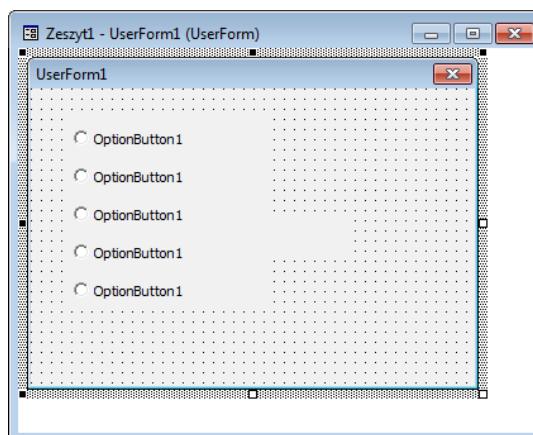
Gdy jednocześnie zaznaczasz wiele formantów, wokół ostatniego z nich zamiast standardowych czarnych uchwytów pojawiają się białe. Formant z białymi uchwytami spełnia rolę wzorca dla położenia i zmiany rozmiaru zaznaczonych formantów.



Rysunek 11.4. Polecenie Format/Align umożliwia zmianę sposobu wyrównania formantów

Rysunek 11.5.

Formanty po wyrównaniu do lewej i wyrównaniu odstępów w pionie



Modyfikowanie właściwości formantów

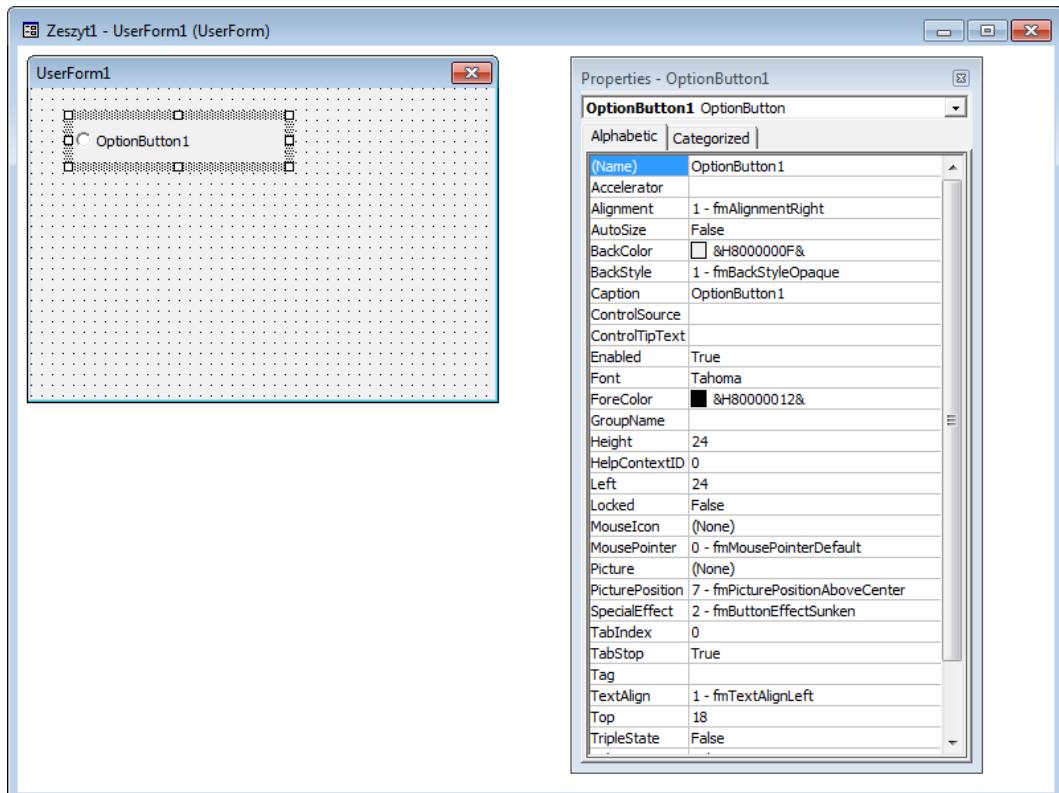
Każdy formant posiada kilka właściwości określających jego wygląd i zachowanie. Właściwości formantów możesz modyfikować na dwa sposoby:

- **W czasie projektowania**, czyli w trakcie projektowania formularza *UserForm*. Użyj do tego celu odpowiednich opcji okna *Properties*.

- W czasie działania programu, po wyświetleniu formularza *UserForm* przez użytkownika. Aby zmienić właściwości formantów, powinieneś użyć odpowiednich polecień języka VBA.

Zastosowanie okna Properties

Zawartość okna *Properties* edytora VBE zmienia się w zależności od właściwości aktualnie wybranego obiektu (którym może być wybrany formant lub sam formularz *UserForm*). Oprócz tego żądana formant można wybrać z listy rozwijanej znajdującej się w górnej części okna *Properties*. Na rysunku 11.6 przedstawiono okno właściwości formantu *OptionButton*.



Rysunek 11.6. Okno właściwości formantu *OptionButton*



Okno *Properties* ma dwie karty. Karta *Alphabetic* wyświetla w kolejności alfabetycznej właściwości zaznaczonego obiektu. Karta *Categorized* wyświetla właściwości pogrupowane według logicznych kategorii. Obie karty zawierają taki sam zestaw właściwości, ale uporządkowanych w różny sposób.

Aby zmienić wybraną właściwość, wystarczy ją kliknąć i podać jej nową wartość. Niektóre właściwości mogą przyjmować skońzoną liczbę wartości wybieranych z listy. Jeżeli tak jest, w oknie *Properties* pojawia się przycisk ze strzałką skierowaną w dół. Po kliknięciu przycisku możliwe będzie wybranie z listy wartości właściwości. Przykładowo

właściwość TextAlign może posiadać takie wartości, jak 1 - fmTextAlignLeft, 2 - fmTextAlignCenter lub 3 - fmTextAlignRight.

Po wybraniu niektórych właściwości (np. Font i Picture) pojawia się niewielki przycisk z trzema kropkami, który trzeba kliknąć, aby wyświetlić okno dialogowe powiązane z właściwością.

Właściwość Picture formantu Image jest godna osobnej wzmianki, ponieważ możesz dla niej albo wybrać plik graficzny znajdujący się na dysku, albo wkleić obraz ze schowka systemowego. Aby wstawić obraz ze schowka, najpierw powinieneś go skopiować do schowka, a następnie wybrać właściwość Picture formantu Image i nacisnąć kombinację klawiszy *Ctrl+V*.



Jeżeli zaznaczysz jednocześnie dwa lub więcej formantów, w oknie *Properties* pojawią się tylko te właściwości, które są wspólne dla wszystkich zaznaczonych formantów.



Wiele właściwości formularza *UserForm* można modyfikować. Wartości tych właściwości pełnią później funkcję domyślnych wartości dla wielu właściwości formantów dodawanych do formularza *UserForm*. Jeżeli na przykład zmienisz wartość właściwości Font formularza *UserForm*, wszystkie dodawane formanty będą używały nowej, wybranej przez Ciebie czcionki. Warto jednak zauważyć, że taka zmiana nie będzie miała wpływu na formanty, które zostały już wcześniej umieszczone na formularzu.

Wspólne właściwości

Pomimo iż każdy formant posiada własny zestaw unikatowych właściwości, niektóre właściwości są dla wielu formantów wspólne. Na przykład każdy formant posiada właściwość Name i właściwości określające jego wielkość i położenie (Height, Width, Left i Right).

Jeżeli danym formantem będziesz operował przy użyciu poleceń języka VBA, warto nadać mu bardziej znaczącą nazwę. Przykładowo pierwszy formant OptionButton dodany do formularza *UserForm* posiada domyślną nazwę OptionButton1. W celu odwołania się w kodzie programu do tego obiektu powinieneś użyć następującej instrukcji:

```
OptionButton1.Value = True
```

Jeżeli jednak nadasz formantowi OptionButton bardziej znaczącą nazwę, taką jak na przykład obOrientacjaPozioma, możesz użyć następującej instrukcji:

```
obOrientacjaPozioma.Value = True
```



Wielu programistów uważa, że wielce pomocne jest stosowanie nazw identyfikujących również typ obiektu. W poprzednim przykładzie użyłem prefiksu ob, aby wskazać, że nasz formant to OptionButton. W zasadzie nie istnieje żaden standardowy sposób tworzenia nazw prefiksów, więc możesz swobodnie popuścić wodze fantazji.

W razie potrzeby możesz modyfikować właściwości wielu formantów jednocześnie. Na przykład kilka formantów OptionButton możesz wyrównać względem lewej strony. W tym celu powinieneś zaznaczyć wszystkie formanty, których położenie chcesz wyrównać, a następnie w oknie *Properties* zmodyfikować wartość właściwości Left. Wszystkie zaznaczone formanty użyją nowej wartości właściwości Left.

Najlepszym źródłem wiedzy o właściwościach poszczególnych formantów jest po prostu pomoc systemowa. Aby z niej skorzystać, kliknij wybraną właściwość w oknie *Properties* i następnie naciśnij klawisz *F1*.

Uwzględnienie wymagań użytkowników preferujących korzystanie z klawiatury

Wielu użytkowników preferuje poruszanie się w oknie dialogowym za pomocą klawiatury. Naciskając klawisz *Tab* i kombinację klawiszy *Shift+Tab*, można przechodzić pomiędzy kolejnymi formantami okna dialogowego. Naciśnięcie wybranego klawisza skrótu powoduje uaktywnienie powiązanego z nim formantu. Aby mieć pewność, że okno dialogowe uwzględnia wymagania użytkowników preferujących klawiaturę, musisz pamiętać o dwóch sprawach: klawiszach skrótu i kolejności tabulacji.

Zmiana kolejności tabulacji formantów

Kolejność tabulacji to inaczej kolejność uaktywniania formantów, kiedy użytkownik wciska klawisz *Tab* lub kombinację klawiszy *Shift+Tab*. Kolejność tabulacji określa również, który formant będzie aktywny jako pierwszy. Jeżeli na przykład użytkownik wprowadza tekst w formancie *TextBox*, formant ten staje się aktywny. Jeżeli klikniesz formant *OptionButton*, to on będzie aktywnym obiektem. Gdy okno dialogowe jest wyświetlane po raz pierwszy, domyślnie aktywny będzie pierwszy formant na liście kolejności tabulacji.

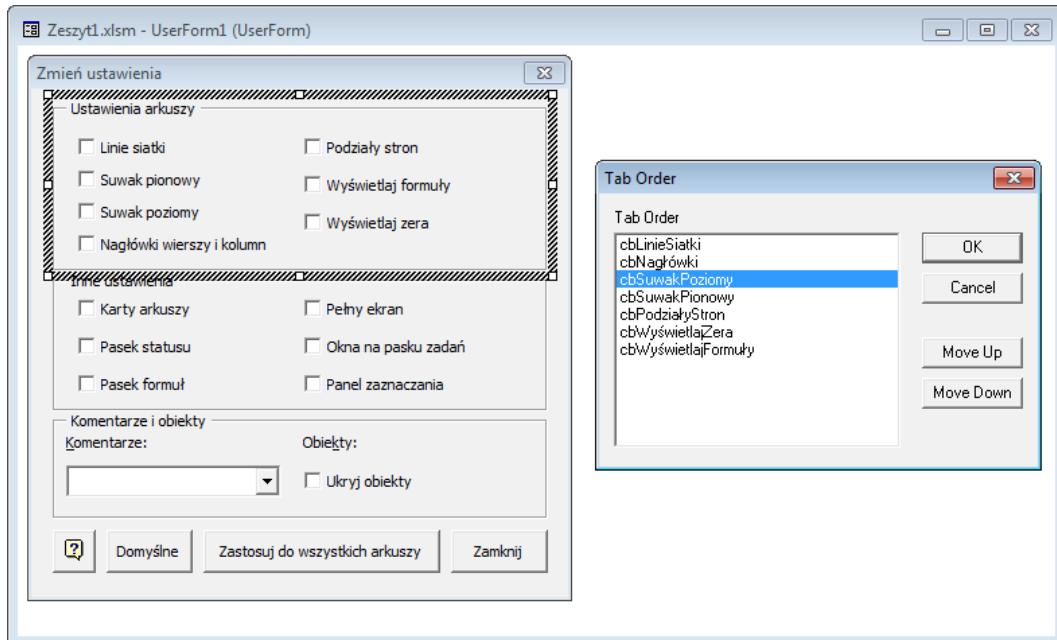
Aby dla formantów zdefiniować kolejność tabulacji, z menu *View* powinieneś wybrać polecenie *Tab Order*. Możesz również prawym przyciskiem myszy kliknąć okno dialogowe i z menu podręcznego wybrać polecenie *Tab Order*. W obu przypadkach Excel wyświetli okno dialogowe *Tab Order*, zawierające listę wszystkich formantów w kolejności zgodnej z kolejnością uaktywniania ich w formularzu *UserForm*.

Aby zmienić pozycję formantu, powinieneś go zaznaczyć i nacisnąć przycisk *Move Up* lub *Move Down*. Jednocześnie możesz zaznaczyć więcej niż jeden formant (należy je kliknąć, trzymając wciśnięty klawisz *Shift* lub *Ctrl*) i zmienić pozycję ich wszystkich.

Przy użyciu okna *Properties* można też określić położenie pojedynczego formantu znajdującego się na liście kolejności tabulacji. Pierwszy formant z tej listy ma wartość właściwości *TabIndex* równą zero. Zmiana wartości właściwości *TabIndex* formantu może mieć wpływ na właściwość *TabIndex* pozostałych formantów. Operacja odpowiedniego ustawienia wartości właściwości *TabIndex* jest wykonywana automatycznie, co ma na celu zagwarantowanie, że w przypadku ani jednego formantu wartość tej właściwości nie przekroczy liczby wszystkich użytych formantów. W celu usunięcia formantu z listy kolejności tabulacji dla jego właściwości *TabStop* powinieneś ustawić wartość *False*.



Uwaga Niektóre formanty, takie jak *Frame* i *MultiPage*, pełnią funkcję kontenerów innych formantów. Formanty znajdujące się wewnątrz kontenera mają własną kolejność tabulacji. Aby stworzyć kolejność tabulacji dla grupy formantów *OptionButton* znajdujących się wewnątrz formantu *Frame*, przed wybraniem z menu *View* pozycji *Tab Order* powinieneś zaznaczyć formant *Frame*. Na rysunku 11.7 przedstawiono wygląd okna dialogowego *Tab Order* po zaznaczeniu formantu *Frame*.



Rysunek 11.7. Okno dialogowe Tab Order umożliwia określenie kolejności tabulacji formantów

Definiowanie klawiszy skrótu

Klawisze skrótu mogą być powiązane z większością formantów w oknach dialogowych. Dzięki temu użytkownik może ich używać, naciskając kombinację klawisza *Alt* i klawisza skrótu. Aby zdefiniować klawisz skrótu, powinieneś posłużyć się właściwością *Accelerator* w oknie *Properties*.



Niektóre formanty (np. *TextBox*) nie posiadają właściwości *Accelerator*, ponieważ nie wyświetlają opisu. Jednak przy użyciu formantu *Label* możesz im przypisać wybrany klawisz skrótu. Aby to zrobić, powinieneś dla formantu *Label* zdefiniować klawisz skrótu, a następnie na liście kolejności tabulacji umieścić go przed formantem *TextBox*.

Testowanie formularza UserForm

Zazwyczaj testowanie nowego formularza *UserForm* będziesz chciał przeprowadzić już na etapie jego tworzenia. Formularz *UserForm* możesz przetestować na trzy sposoby, bez konieczności wywoływania go z poziomu kodu źródłowego procedury języka VBA:

- Wybierz z menu *Run* polecenie *Run Sub/UserForm*.
- Naciśnij klawisz *F5*.
- Naciśnij przycisk *Run Sub/UserForm* znajdujący się na pasku narzędzi *Standard*.

Wszystkie trzy sposoby generują zdarzenie *Initialize* powiązane z formularzem *UserForm*. Po wyświetleniu okna dialogowego w trybie testowym możesz sprawdzić kolejność tabulacji i klawisze skrótu.

Wyświetlanie formularza UserForm

Aby z poziomu kodu źródłowego wyświetlić formularz *UserForm*, powinieneś utworzyć procedurę korzystającą z metody *Show* obiektu *UserForm*. Jeżeli formularz *UserForm* nosi nazwę *UserForm1*, poniższa procedura wyświetli okno dialogowe tego formularza:

```
Sub ShowForm()
    UserForm1.Show
End Sub
```

Taka procedura musi znaleźć się w standardowym module VBA, a nie w module kodu źródłowego formularza *UserForm*.

Formularz *UserForm* pozostanie widoczny na ekranie do momentu jego zamknięcia (przy użyciu polecenia *Unload*) lub ukrycia (przy użyciu metody *Hide* obiektu *UserForm*). Zazwyczaj na formularzu *UserForm* jest umieszczany przycisk *CommandButton* wykonujący procedurę zamkującą formularz. Zagadnienie to stanie się bardziej zrozumiałe po zapoznaniu się z różnymi przykładami z tego i kolejnych rozdziałów.

Zmiana położenia formularza na ekranie

Miejsce na ekranie, w którym będzie wyświetlany formularz *UserForm*, zależy od wartości właściwości o nazwie *StartPosition*, którą możesz zdefiniować na etapie projektowania, za pomocą okna dialogowego *Properties*, bądź dynamicznie, z poziomu kodu programu. Domyślnie wartością tej właściwości jest 1, co powoduje, że okna dialogowe (formularze) są wyświetlane na środku okna programu Excel.

Jeżeli jednak korzystasz z systemu wyposażonego w dwa monitory, przekonasz się, że czasami właściwość *StartPosition* jest po prostu ignorowana. Dzieje się tak zwłaszcza wtedy, kiedy okno Excela zostaje przeniesione na drugi monitor — w takiej sytuacji formularze *UserForm* mogą się pojawiać przy lewej krawędzi monitora podstawowego.

Fragment kodu przedstawiony poniżej powoduje, że okno formularza *UserForm* będzie zawsze wyświetlane na środku okna programu Excel:

```
With UserForm1
    .StartUpPosition = 0
    .Left = Application.Left + (0.5 * Application.Width) - (0.5 * .Width)
    .Top = Application.Top + (0.5 * Application.Height) - (0.5 * .Height)
    .Show
End With
```

Wyświetlanie niemodalnych okien formularzy UserForm

Domyślnie wszystkie formularze *UserForm* są wyświetlane jako okna modalne. Oznacza to, że w celu wykonania jakiejkolwiek operacji związanej z arkuszem Excela konieczne jest zamknięcie formularza. W razie potrzeby jednak istnieje możliwość wyświetlania niemodalnych formularzy *UserForm*. Po wyświetleniu formularza w trybie niemodalnym

użytkownik może w dalszym ciągu używać Excela, a okno formularza pozostaje ciągle widoczne. Aby wyświetlić niemodalny formularz *UserForm*, powinieneś użyć następującego polecenia:

```
UserForm1.Show vbModeless
```



Teraz, jednodokumentowy interfejs wprowadzony w Excelu 2013 ma dosyć istotny wpływ na funkcjonowanie niemodalnych okien formularzy. W poprzednich wersjach Excela niemodalne okna formularzy były widoczne na ekranie niezależnie od tego, który arkusz był w danej chwili aktywny. W Excelu 2013 niemodalne okna formularza są powiązane ze skoroszytem, który był aktywny podczas pojawienia się formularza. Jeżeli później przejdziesz do innego skoroszytu, to taki formularz może nie być widoczny. W rozdziale 13. znajdziesz przykład kodu pokazujący, w jaki sposób można tworzyć niemodalne formularze *UserForm*, które będą widoczne we wszystkich oknach arkuszy.

Wyświetlanie formularza UserForm na podstawie zmiennej

W niektórych przypadkach możesz korzystać z kilku formularzy *UserForm*, a program podejmuje decyzję o tym, który z nich ma zostać wyświetlony. Jeżeli nazwa formularza *UserForm* jest przechowywana jako zmienna typu *String*, w celu dodania formularza do kolekcji *UserForms* można użyć metody *Add*, a następnie metody *Show* kolekcji *UserForms*. Poniżej zamieszczono przykład, w którym przypisano nazwę formularza *UserForm* zmiennej *MyForm*, a następnie go wyświetlono:

```
MyForm = "UserForm1"  
UserForms.Add(MyForm).Show
```

Ładowanie formularza UserForm

Język VBA oferuje również polecenie *Load*, które ładuje formularz do pamięci. Formularz pozostaje niewidoczny do momentu użycia metody *Show*. Aby załadować formularz *UserForm*, należy użyć następującej instrukcji:

```
Load UserForm1
```

Jeżeli korzystasz z bardzo rozbudowanego, złożonego formularza *UserForm*, możesz go załadować do pamięci, zanim będzie potrzebny, dzięki czemu po zastosowaniu metody *Show* pojawi się znacznie szybciej. Jednak w większości przypadków stosowanie instrukcji *Load* nie jest konieczne.

Procedury obsługi zdarzeń

Po wyświetleniu formularza *UserForm*, użytkownik może z niego korzystać, wybierając elementy z listy formantu *ListBox*, klikając formanty *CommandButton* itd. Aby posłużyć się oficjalną terminologią, należy powiedzieć, że użytkownik generuje różnorodne *zdarzenia*. Na przykład kliknięcie formantu *CommandButton* wywołuje powiązane z nim zdarzenie *Click*, stąd będziesz musiał napisać odpowiednie procedury wykonywane po wygenerowaniu takich zdarzeń, nazywanych czasami procedurami *obsługi zdarzeń*.



Kod źródłowy procedur obsługi zdarzeń musi zostać umieszczony w oknie *Code* formularza *UserForm*, jednak procedury obsługi zdarzeń mogą wywołać inne procedury znajdujące się w standardowym module kodu VBA.

Instrukcje kodu źródłowego języka VBA mogą modyfikować wartości właściwości formantów wyświetlanego formularza *UserForm*. Na przykład do formantu *ListBox* można przypisać procedurę, która po wybraniu z listy danego elementu zmienia tekst formantu *Label*. Więcej informacji na temat takich operacji znajdziesz w dalszej części tego rozdziału.

Zamykanie formularza UserForm

Aby zamknąć formularz *UserForm*, należy posłużyć się instrukcją `Unload`. Oto przykład:

```
Unload UserForm1
```

Jeżeli kod źródłowy znajduje się w module formularza *UserForm*, można użyć następującej instrukcji:

```
Unload Me
```

W tym przypadku słowo kluczowe `Me` odnosi się do formularza *UserForm*. Użycie słowa kluczowego `Me` zamiast nazwy formularza eliminuje konieczność modyfikacji kodu programu w sytuacji, kiedy nazwa formularza zostanie zmieniona.

Zwykle w kodzie źródłowym języka VBA za instrukcjami wykonującymi operacje związane z formularzem *UserForm* powinna się znaleźć instrukcja `Unload`. Na przykład formularz *UserForm* może zawierać formant *CommandButton* spełniający funkcję przycisku *OK*. Kliknięcie przycisku spowoduje wykonanie makra. Jedna z instrukcji makra usunie z pamięci formularz *UserForm*, ale pozostanie on widoczny na ekranie do momentu zakończenia wykonywania tego makra.

Gdy formularz *UserForm* jest usuwany z pamięci, właściwości jego formantów przyjmują domyślne wartości. Innymi słowy, po zamknięciu formularza *UserForm* instrukcje kodu źródłowego nie będą dysponowały dostępem do wartości wprowadzanych przez użytkownika. Jeżeli wartość podana przez użytkownika musi zostać później wykorzystana (po usunięciu z pamięci formularza *UserForm*), konieczne będzie zapisanie jej w zmiennej publicznej zadeklarowanej (z użyciem słowa kluczowego `Public`) w standardowym module języka VBA. Wartość można też umieścić w komórce arkusza lub nawet w odpowiednim wpisie w rejestrze systemowym.



Formularz *UserForm* jest automatycznie usuwany z pamięci, gdy użytkownik kliknie przycisk *Zamknij* (ikona X znajdująca się na pasku tytułu formularza). Operacja ta powoduje wygenerowanie zdarzenia `QueryClose` powiązanego z formularzem *UserForm*, a po nim zdarzenia `Terminate`.

Formularze *UserForms* posiadają również metodę `Hide`. Po jej wywołaniu formularz *UserForm* zostanie ukryty, ale pozostanie w pamięci, dzięki czemu kod programu cały czas będzie dysponował dostępem do różnych właściwości formantów. Oto przykład instrukcji ukrywającej formularz *UserForm*:

```
UserForm1.Hide
```

Jeżeli kod źródłowy znajduje się w module formularza *UserForm*, można użyć następującej instrukcji:

```
Me.Hide
```

Jeżeli z jakiegoś powodu formularz *UserForm* ma zostać ukryty natychmiast po uruchomieniu procedury, na jej początku należy umieścić metodę *Hide*. Na przykład poniższa procedura ukrywa formularz *UserForm* bezpośrednio po kliknięciu formantu Command \rightarrow Button1, a ostatnia instrukcja usuwa formularz *UserForm* z pamięci:

```
Private Sub CommandButton1_Click()
    Me.Hide
    Application.ScreenUpdating = True
    For r = 1 To 10000
        Cells(r, 1) = r
    Next r
    Unload Me
End Sub
```

W tym przykładzie nadajemy właściwości *ScreenUpdating* wartość *True*, co wymusza całkowite ukrycie okna formularza. Jeżeli nie użylibyśmy tego polecenia, w pewnych sytuacjach formularz *UserForm* mógłby pozostać widoczny na ekranie.



W rozdziale 13. wyjaśniono, w jaki sposób wyświetlić wskaźnik postępu, który wykorzystuje to, że formularz *UserForm* pozostaje widoczny w trakcie wykonywania makra.

Przykład tworzenia formularza *UserForm*

Jeżeli jeszcze nigdy nie tworzyłeś formularza *UserForm*, powinieneś dokładnie zapoznać się z przykładem opisany w tym podrozdziale. Zaprezentujemy tutaj krok po kroku sposób tworzenia prostego okna dialogowego oraz obsługującej go procedury języka VBA.

W przykładzie użyto formularza *UserForm* pobierającego dwie informacje — imię i płeć. W celu pobrania imienia użyto formantu *TextBox*, a do pobrania informacji o płci (kobieta, mężczyzna, nieznana) zostały użyte trzy formanty *OptionButton*. Informacja wprowadzona w oknie dialogowym jest przesyłana do następnego pustego wiersza arkusza.

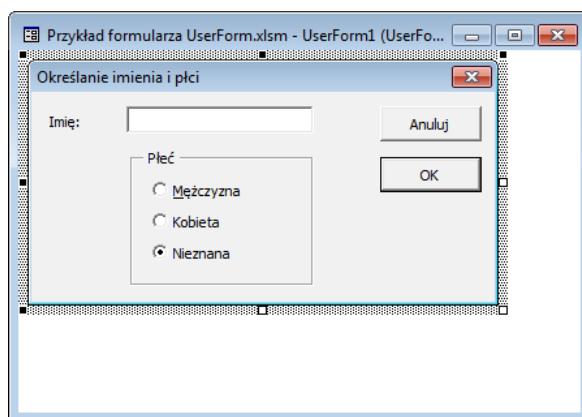
Tworzenie formularza *UserForm*

Na rysunku 11.8 pokazano gotowy formularz *UserForm* zastosowany w przykładzie.

Aby uzyskać jak najlepsze efekty, otwórz nowy skoroszyt zawierający tylko jeden arkusz i wykonaj następujące polecenia:

1. Uruchom edytor VBE, naciskając kombinację klawiszy *Alt+F11*.
 2. W oknie *Project Explorer* wybierz projekt skoroszytu i dodaj pusty formularz *UserForm*. Aby to zrobić, z menu *Insert* wybierz polecenie *UserForm*.
- Domyślną wartością właściwości *Caption* formularza *UserForm* jest *UserForm1*.
3. Użyj okna *Properties* do zmiany nazwy formularza na *Określanie imienia i płci*.

Rysunek 11.8.
Okno dialogowe
umożliwia
użytkownikowi
wprowadzenie imienia
i określenie płci



4. Jeżeli okno *Properties* nie jest widoczne, naciśnij klawisz *F4*.
5. Dodaj formant *Label* i ustaw wymienione niżej wartości dla następujących właściwości:

Właściwość	Wartość
Accelerator	I
Caption	Imię:
TabIndex	0

6. Dodaj formant *TextBox* i ustaw wymienione niżej wartości dla następujących właściwości:

Właściwość	Wartość
Name	TextName
TabIndex	1

7. Dodaj formant *Frame* i ustaw wymienione niżej wartości dla następujących właściwości:

Właściwość	Wartość
Caption	Płeć
TabIndex	2

8. Wewnątrz kontenera *Frame* umieść formant *OptionButton* i ustaw wartości dla następujących właściwości:

Właściwość	Wartość
Accelerator	M
Caption	Mężczyzna
Name	OptionMale
TabIndex	0

9. Wewnątrz kontenera Frame umieść kolejny formant OptionButton i ustaw wartości dla następujących właściwości:

Właściwość	Wartość
Accelerator	K
Caption	Kobieta
Name	OptionFemale
TabIndex	1

10. Wewnątrz kontenera Frame umieść jeszcze jeden formant OptionButton i ustaw wartości dla następujących właściwości:

Właściwość	Wartość
Accelerator	N
Caption	Nieznana
Name	OptionUnknown
TabIndex	2
Value	True

11. Poza obszarem kontenera Frame umieść formant CommandButton i ustaw wartości dla następujących właściwości:

Właściwość	Wartość
Caption	OK
Default	True
Name	OKButton
TabIndex	3

12. Umieść kolejny formant CommandButton i ustaw wartości dla następujących właściwości:

Właściwość	Wartość
Caption	Anuluj
Cancel	True
Name	CancelButton
TabIndex	4



W przypadku tworzenia kilku podobnych formantów od dodawania nowego formantu łatwiejsze może okazać się skopiowanie formantu już istniejącego. Aby skopiować formant, w trakcie przeciągania należy trzymać wcisnięty klawisz *Ctrl*. Po wykonaniu tej operacji można zmodyfikować wartości właściwości skopiowanego formantu.

Tworzenie kodu procedury wyświetlającej okno dialogowe

Pora dodać do arkusza formant ActiveX o nazwie *Przycisk polecenia* (CommandButton). Naciśnięcie tego przycisku spowoduje wykonanie procedury wyświetlającej formularz *UserForm*. Aby to było możliwe, wykonaj następujące kroki:

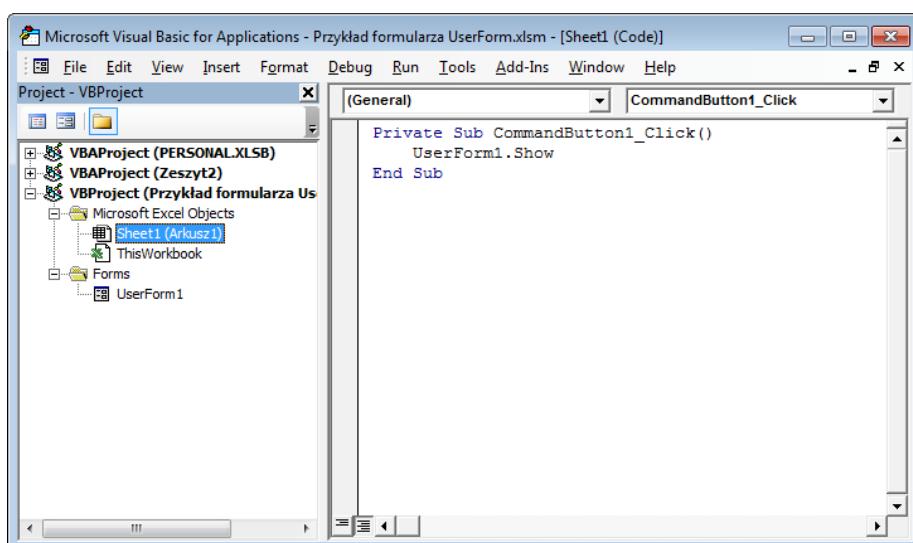
1. Uaktywnij okno Excela (np. za pomocą kombinacji klawiszy *Alt+F11*).
2. Przejdź na kartę *DEVELOPER*, naciśnij przycisk *Wstaw* znajdujący się w grupie opcji *Formanty* i z menu podręcznego wybierz *Przycisk polecenia (formant ActiveX)*.
3. Przeciągnij formant myszą tak, aby dobrać jego rozmiary i położenie na arkuszu.

W razie potrzeby możesz zmienić etykietę formantu CommandButton umieszczonego w arkuszu. Aby to zrobić, prawym przyciskiem myszy kliknij przycisk i z menu podręcznego wybierz polecenie *Obiekt CommandButton/Edit*. Po wykonaniu tej operacji będziesz mógł zmienić etykietę przycisku CommandButton. Aby zmienić inne właściwości tego obiektu, kliknij go prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Właściwości*. Na ekranie pojawi się okno dialogowe *Properties*, w którym będziesz mógł zmienić ustawienia właściwości.

4. Dwukrotnie kliknij formant CommandButton.

Na ekranie pojawi się okno edytora VBE, a dokładniej mówiąc, zostanie wyświetlony moduł kodu programu arkusza zawierający pustą procedurę obsługi zdarzenia powiązanego z formantem CommandButton.

5. Wewnątrz procedury *CommandButton1_Click* wprowadź pojedynczą instrukcję (patrz rysunek 11.9).



Rysunek 11.9. Procedura *CommandButton1_Click* jest wykonywana po kliknięciu przycisku znajdującego się w arkuszu

6. Ta krótka procedura używa metody Show obiektu UserForm1 do wyświetlenia formularza *UserForm* na ekranie.

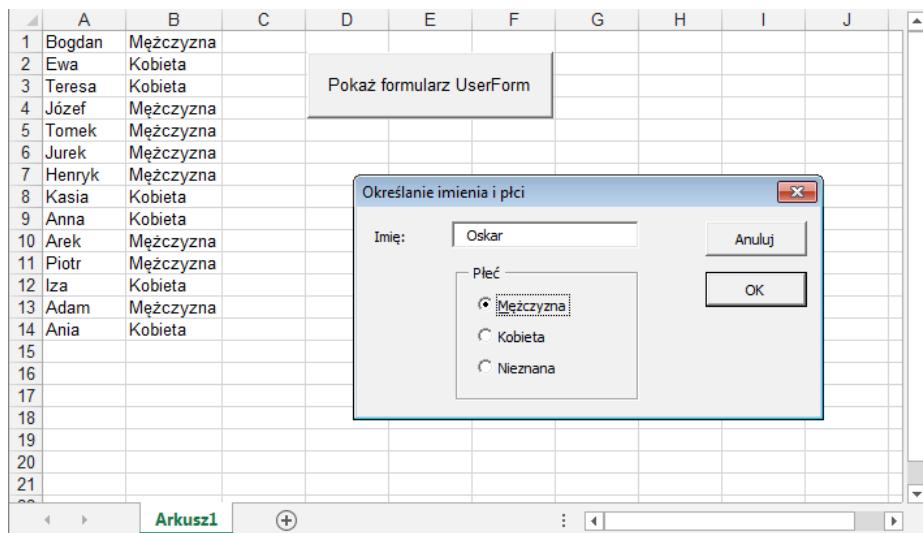
Testowanie okna dialogowego

Następnym krokiem jest przetestowanie procedury wyświetlającej okno dialogowe.



Po kliknięciu formantu CommandButton znajdującego się w arkuszu przekonasz się, że nic się nie dzieje (poza tym, że przycisk zostaje zaznaczony). Dzieje się tak, ponieważ Excel nadal znajduje się w trybie projektowania, który uaktywnia się automatycznie, gdy dodawany jest formant ActiveX. Aby wyjść z trybu projektowania, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Tryb projektowania*, znajdujący się w grupie opcji *Formanty*. Aby można było dokonać jakichkolwiek modyfikacji kontrolki CommandButton, konieczne będzie ponowne przełączenie Excela w tryb projektowania.

Po zakończeniu trybu projektowania kliknięcie przycisku spowoduje wyświetlenie formularza *UserForm* (patrz rysunek 11.10).



Rysunek 11.10. Procedura obsługi zdarzenia Click formantu CommandButton wyświetla formularz *UserForm*

Po wyświetleniu okna dialogowego w polu *Imię* można wprowadzić tekst i nacisnąć przycisk *OK*. Nic się jednak nie wydarzy, ponieważ dla formularza *UserForm* nie utworzyliśmy jeszcze żadnej procedury obsługi zdarzenia.



Kliknij przycisk *Zamknij* znajdujący się na pasku tytułu formularza *UserForm*, co spowoduje zamknięcie okna dialogowego.

Dodawanie procedur obsługi zdarzeń

W tym podrozdziale wyjaśnimy, jak napisać procedury obsługujące zdarzenia generowane po wyświetleniu formularza *UserForm*. Aby kontynuować nasz przykład, wykonaj następujące kroki:

1. W celu uaktywnienia edytora Visual Basic wciśnij kombinację klawiszy *Alt+F11*.
2. Po upewnieniu się, że formularz *UserForm* został wyświetlony, dwukrotnie kliknij przycisk *Anuluj*. Wykonanie takiej operacji spowoduje uaktywnienie okna *Code* formularza *UserForm* i wstawienie w nim pustej procedury o nazwie *CancelButton_Click*. Zwróć uwagę, że nazwa procedury jest złożona z nazwy obiektu, znaku podkreślenia i nazwy obsługiwanej zdarzenia.
3. Zmodyfikuj procedurę zgodnie ze wzorem przedstawionym poniżej (jest to procedura obsługi zdarzenia *Click* powiązanego z formantem *CancelButton*).

```
Private Sub CancelButton_Click()
    Unload UserForm1
End Sub
```

Procedura wykonywana po kliknięciu przez użytkownika przycisku *Anuluj* zamknie i usuwa z pamięci formularz *UserForm*.

4. Aby ponownie wyświetlić formularz *UserForm1*, wciśnij kombinację klawiszy *Shift+F7* lub kliknij ikonę *View Object* widoczną w górnej części okna *Project Explorer*.
5. Dwukrotnie kliknij przycisk *OK* i wprowadź poniższą procedurę obsługi zdarzenia *Click* formantu *OKButton*:

```
Private Sub OKButton_Click()
    Dim NextRow As Long
    ' Sprawdza, czy uaktywniono Arkusz1
    Sheets("Arkusz1").Activate

    ' Określa następny pusty wiersz
    NextRow =
        Application.WorksheetFunction.CountA(Range("A:A")) + 1
    ' Przenosi imię
    Cells(NextRow, 1) = TextName.Text

    ' Przenosi plec
    If OptionMale Then Cells(NextRow, 2) = "Mężczyzna"
    If OptionFemale Then Cells(NextRow, 2) = "Kobieta"
    If OptionUnknown Then Cells(NextRow, 2) = "Nieznanego"

    ' Czyści formanty przed wprowadzeniem kolejnych danych
    TextName.Text = ""
    OptionUnknown = True
    TextName.SetFocus
End Sub
```

6. Uaktywnij okno Excela i ponownie naciśnij przycisk *CommandButton*, aby wyświetlić formularz *UserForm*. Uruchom procedurę jeszcze raz.

Przekonasz się, że teraz formanty formularza *UserForm* będą działać poprawnie. Od tej chwili możesz używać formularza do dodawania nowych imion do listy.

Wyjaśnij teraz sposób działania procedury `OKButton_Click`. Najpierw procedura sprawdza, czy został uaktywniony właściwy arkusz (Arkusz1). Następnie jest używana funkcja COUNTA (ILE_NIEPUSTYCH) Excela, która identyfikuje kolejną pustą komórkę kolumny A. W dalszej kolejności procedura przenosi tekst z formantu TextBox do kolumny A, po czym przy użyciu kilku instrukcji If określa, który formant OptionButton zaznaczono, i w kolumnie B zapisuje odpowiedni łańcuch (Kobieta, Mężczyzna lub Nieznana). W celu przygotowania okna dialogowego do wprowadzenia następnych danych na końcu są ustawiane dla niego wartości początkowe. Zwróć uwagę, że kliknięcie *OK* nie powoduje zamknięcia okna dialogowego. Aby zakończyć wprowadzanie danych i usunąć z pamięci formularz *UserForm*, należy kliknąć przycisk *Anuluj*.

Sprawdzanie poprawności danych

Kiedy poeksperymentujesz trochę z naszym formularzem przekonasz się, że mamy tutaj drobny problem. Otóż nasza procedura nie gwarantuje, że użytkownik faktycznie wprowadzi w polu tekstowym dane. Zamieszczony poniżej kod źródłowy został umieszczony w procedurze `OKButton_Click` przed instrukcją przenoszącą tekst do arkusza. Dzięki temu uzyskujemy pewność, że użytkownik poda imię (lub wpisze jakikolwiek tekst) w polu TextBox. Jeżeli pole TextBox jest puste, pojawi się odpowiedni komunikat, po czym zostanie ponownie uaktywniony formant TextBox. Instrukcja Exit Sub kończy procedurę bez wykonywania dodatkowych operacji.

```
' Sprawdza, czy wprowadzono imię
If TextName.Text = "" Then
    MsgBox "Musisz podać imię."
    TextName.SetFocus
    Exit Sub
End If
```

Zakończenie tworzenia okna dialogowego

Po wykonaniu wszystkich modyfikacji przekonasz się, że okno dialogowe działa bezbłędnie (nie zapomnij o przetestowaniu skrótów klawiszowych). W rzeczywistej aplikacji prawdopodobnie będziesz musiał pobierać większą liczbę informacji, a nie tylko imię i płeć. Niezależnie jednak od ilości wprowadzanych czy jak kto woli, pobieranych informacji, cały czas obowiązują takie same zasady, a zmienia się jedynie liczba formularzy oraz formantów, które będziesz musiał obsługiwać.



Skoroszyt z tym przykładem (*Przykład formularza UserForm.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci

Zdarzenia powiązane z formularzem UserForm

Każdy formant formularza *UserForm*, a także sam formularz odpowiada na zdarzenia określonego typu. Zdarzenia te mogą generować użytkownicy lub Excel. Na przykład kliknięcie formantu CommandButton powoduje wygenerowanie powiązanego z nim zdarzenia Click, a Ty możesz napisać odpowiednią procedurę, która będzie wykonywana, gdy wystąpi takie zdarzenie.

Niektóre operacje generują wiele zdarzeń. Na przykład kliknięcie przycisku strzałki w góre formantu SpinButton wywołuje zdarzenia SpinUp oraz Change. Po wyświetleniu formularza *UserForm* Excel generuje przy użyciu metody Show powiązane z formularzem zdarzenia Initialize i Activate (tak naprawdę zdarzenie Initialize występuje w chwili ładowania do pamięci formularza *UserForm*, jeszcze przed jego wyświetleniem).



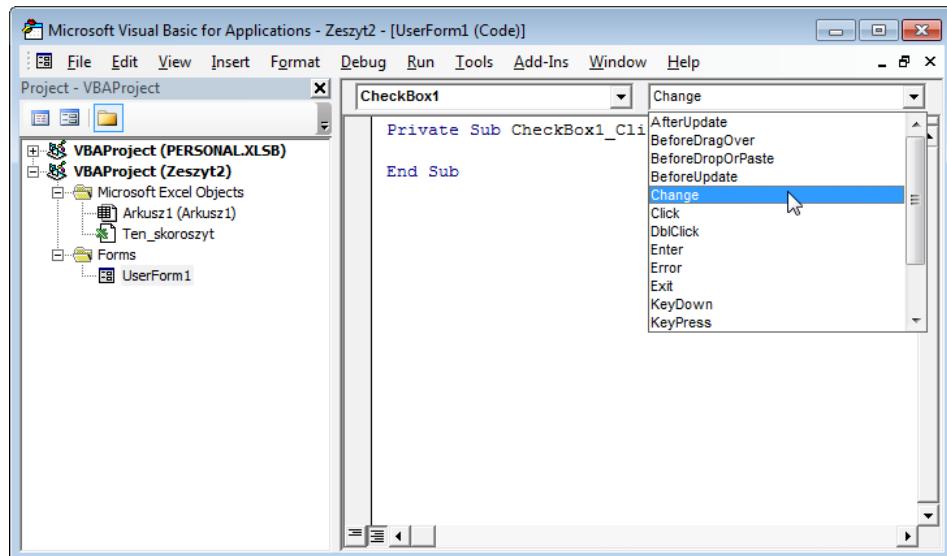
Excel obsługuje też zdarzenia powiązane z obiektami Sheet, Chart i ThisWorkbook. Zostaną one omówione w rozdziale 17.

Zdobywanie informacji na temat zdarzeń

Aby dowiedzieć się, które zdarzenia obsługuje określony formant, wykonaj następujące kroki:

1. Do formularza *UserForm* dodaj wybrany formant.
2. W celu uaktywnienia modułu kodu źródłowego formularza *UserForm* dwukrotnie kliknij formant lewym przyciskiem myszy.
3. Edytor VBE automatycznie wstawi pustą procedurę obsługi domyślnego zdarzenia powiązanego z tym formantem.
4. Po kliknięciu listy rozwijanej znajdującej się w prawym górnym narożniku okna modułu pojawi się pełna lista zdarzeń powiązanych z tym formantem.

Na rysunku 11.11 przedstawiono listę zdarzeń formantu CheckBox.



Rysunek 11.11. Lista zdarzeń powiązanych z formantem CheckBox

5. Po wybraniu zdarzenia z listy edytor Visual Basic automatycznie utworzy pustą procedurę obsługi zdarzenia.

6. Aby zapoznać się z dokładnymi informacjami o danym zdarzeniu, powinieneś skorzystać z systemu pomocy. W systemie pomocy znajdziesz również listę zdarzeń dostępnych dla każdego formantu.



Nazwa procedury obsługi zdarzenia zawiera nazwę powiązanego z nią obiektu. A zatem jeżeli zostanie zmieniona nazwa formantu, konieczne będzie też dokonanie odpowiednich modyfikacji w powiązanej z nią procedurze lub procedurach obsługi zdarzeń. Nazwy nie są zmieniane automatycznie! Aby uprościć sobie zadanie, można nadać nazwy formatom, zanim rozpoczęcie się tworzenie procedur obsługi zdarzeń.

Zdarzenia formularza UserForm

Z wyświetlaniem formularza *UserForm* i usuwaniem go z pamięci związań jest kilka zdarzeń. Oto one:

- Initialize — występuje przed załadowaniem lub wyświetleniem formularza *UserForm*, ale nie jest generowane, gdy formularz został wcześniej ukryty;
- Activate — występuje po wyświetleniu formularza *UserForm*;
- Deactivate — występuje po dezaktywacji formularza *UserForm*, ale nie jest generowane w przypadku ukrywania formularza;
- QueryClose — występuje przed rozpoczęciem usuwania z pamięci formularza *UserForm*;
- Terminate — występuje po usunięciu z pamięci formularza *UserForm*.



Podczas pisania programy krytyczne znaczenie ma wybór odpowiedniego zdarzenia, z którymi będzie powiązana procedura obsługi zdarzenia, oraz dobra znajomość kolejności występowania zdarzeń. Zastosowanie metody *Show* spowoduje wygenerowanie zdarzeń *Initialize* i *Activate* (w takiej kolejności). Instrukcja *Load* generuje tylko zdarzenie *Initialize*. Z kolei instrukcja *Unload* wywołuje zdarzenia *QueryClose* i *Terminate* (właśnie w takiej kolejności). Metoda *Hide* nie generuje żadnego z wymienionych zdarzeń.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt o nazwie *Zdarzenia formularza UserForm.xlsm*, monitorujący wszystkie zdarzenia i po ich wystąpieniu wyświetlający okno komunikatu. Jeżeli zdarzenia związane z formularzem *UserForm* wydają Ci się zagmatwane, przeanalizowanie kodu źródłowego zamieszczonego w tym przykładzie powinno rozwiązać wiele wątpliwości.

Zdarzenia związane z formantem SpinButton

Aby ułatwić zrozumienie zdarzeń, w tym punkcie omówimy bliżej zdarzenia powiązane z formantem *SpinButton*. Niektóre z tych zdarzeń związane są z innymi formantami, a inne odnoszą się tylko i wyłącznie do formantu *SpinButton*.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt o nazwie *Zdarzenia formantu SpinButton.xlsm*, demonstrujący sekwencję zdarzeń generowanych przez formant *SpinButton* i zawierający go formularz *UserForm*. Skoroszyt przechowuje kilka procedur obsługi zdarzeń, po jednej dla każdego zdarzenia powiązanego z formantem *SpinButton* i formularzem *UserForm*. Każda z tych procedur wyświetla okno komunikatu informujące o właśnie wygenerowanym zdarzeniu.

W tabeli 11.1 zamieszczone zostały zdarzenia powiązane z formantem SpinButton.

Tabela 11.1. Zdarzenia powiązane z formantem SpinButton

Zdarzenie	Opis
AfterUpdate	Występuje po zmodyfikowaniu formantu za pośrednictwem interfejsu użytkownika.
BeforeDragOver	Występuje w trakcie wykonywania operacji przeciągania i upuszczania.
BeforeDropOrPaste	Występuje, gdy użytkownik zamierza na obszarze formantu wstawić lub wkleić dane.
BeforeUpdate	Występuje przed zmodyfikowaniem formantu.
Change	Występuje po zmianie wartości właściwości Value.
Enter	Występuje przed faktycznym uaktywnieniem formantu przez poprzedni obiekt znajdujący się na tym samym formularzu <i>UserForm</i> .
Error	Występuje, gdy formant wykryje błąd i nie jest w stanie zwrócić informacji o nim procedurze wywołującej.
Exit	Występuje bezpośrednio przed przekazaniem przez formant stanu aktywności innego formantu znajdującego się na tym samym formularzu.
KeyDown	Występuje, gdy obiekt jest uaktywniony i użytkownik wcisnie dowolny klawisz.
KeyPress	Występuje, gdy użytkownik wcisnie dowolny klawisz, który spowoduje wprowadzenie znaku.
KeyUp	Występuje, gdy obiekt jest uaktywniony a użytkownik zwolni wcisnięty klawisz.
SpinDown	Występuje, gdy użytkownik kliknie przycisk strzałki formantu SpinButton skierowanej w dół lub w lewo.
SpinUp	Występuje, gdy użytkownik kliknie przycisk strzałki formantu SpinButton skierowanej w górę lub w prawo.

Użytkownik może skorzystać z formantu SpinButton, klikając go myszą lub wciskając klawisz strzałki w góre bądź w dół (jeżeli formant jest aktywny).

Zdarzenia generowane przez mysz

Gdy użytkownik kliknie przycisk strzałki w góre formantu SpinButton, zdarzenia są generowane w następującej kolejności:

1. Enter (wywoływane tylko wtedy, gdy formant SpinButton nie był wcześniej aktywny)
2. Change
3. SpinUp

Zdarzenia generowane przez klawiaturę

W celu uaktywnienia formantu SpinButton użytkownik może też wcisnąć klawisz *Tab*, a następnie zastosować klawisze strzałek zwiększające lub zmniejszające wartość formantu. W tym przypadku wystąpią następujące zdarzenia (w podanej kolejności):

1. Enter
2. KeyDown
3. Change
4. SpinUp (lub SpinDown)
5. KeyUp

Dokonywanie zmian za pośrednictwem instrukcji języka VBA

Formant SpinButton może być też modyfikowany przy użyciu instrukcji języka VBA generujących odpowiednie zdarzenia. Poniższa instrukcja nadaje właściwości Value obiektu SpinButton1 wartość 0 i generuje zdarzenie Change powiązane z formantem SpinButton (oczywiście tylko wtedy, gdy wcześniej wartością właściwości Value nie było zero):

```
SpinButton1.Value = 0
```

Można by przypuszczać, że poprzez ustawienie dla właściwości EnableEvents obiektu Application wartości False możliwe jest wyłączenie zdarzeń. Niestety właściwość ta dotyczy tylko zdarzeń powiązanych z typowymi obiektami Excela, takimi jak Workbooks, Worksheets i Charts.

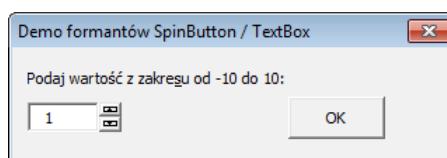
Współpraca formantu SpinButton z formantem TextBox

Formant SpinButton posiada właściwość Value, ale nie dysponuje etykietą umożliwiającą wyświetlenie jej wartości. Jednak w wielu przypadkach użytkownik powinien zobaczyć wartość właściwości Value formantu SpinButton. Czasem będzie też wskazane, aby użytkownik mógł zmienić tę wartość bezpośrednio, a nie poprzez kilkakrotne kliknięcie formantu SpinButton.

Rozwiążanie polega na wspólnym użyciu formantów SpinButton i TextBox. Umożliwia ono użytkownikowi określenie wartości poprzez wprowadzenie jej bezpośrednio w kontrolce TextBox lub poprzez kliknięcie formantu SpinButton, co powoduje zwiększenie lub zmniejszenie wartości wyświetlonej przez formant TextBox.

Na rysunku 11.12 pokazano prosty przykład. Wartością właściwości Min formantu Spin→Button jest -10, natomiast właściwości Max — liczba 10. W związku z tym kliknięcie strzałek formantu SpinButton spowoduje zmianę jej wartości całkowitej zawartej w przedziale od -10 do 10.

Rysunek 11.12.
Formant SpinButton zastosowany razem z formantem TextBox



Skoroszyt z tym przykładem (*Formanty SpinButton i TextBox.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Kod źródłowy łączący formant SpinButton z formantem TextBox jest stosunkowo prosty. Aby zsynchronizować właściwość Value formantu SpinButton z właściwością Text formantu TextBox, wystarczy napisać odpowiednie procedury obsługi zdarzeń.

Poniższa procedura jest wykonywana każdorazowo po wygenerowaniu zdarzenia Change powiązanego z formantem SpinButton. Oznacza to, że procedura jest uruchamiana, gdy użytkownik kliknie formant SpinButton lub gdy zmieni wartość jego właściwości Value po wcięnięciu klawisza strzałki w góre bądź w dół.

```
Private Sub SpinButton1_Change()
    TextBox1.Text = SpinButton1.Value
End Sub
```

Procedura przypisuje jedynie wartość właściwości Value formantu SpinButton do właściwości Text formantu TextBox. Jeżeli użytkownik bezpośrednio wprowadzi wartość w formancie TextBox, zostanie wygenerowane związane z nią zdarzenie Change i wykonana następująca procedura:

```
Private Sub TextBox1_Change()
    Dim NewVal As Integer
    If IsNumeric(TextBox1.Text) Then
        NewVal = Val(TextBox1.Text)
        If NewVal >= SpinButton1.Min And _
            NewVal <= SpinButton1.Max Then _
            SpinButton1.Value = NewVal
    End If
End Sub
```

Procedura rozpoczyna działanie od sprawdzenia, czy wartość przypisana do formantu TextBox jest liczbą. Jeżeli tak, procedura kontynuuje działanie i przypisuje tę wartość do zmiennej NewVal (dokonując jednocześnie konwersji tekstu na postać numeryczną). Kolejna instrukcja określa, czy wartość jest zawarta w przedziale obsługiwany przez formant SpinButton. Jeżeli tak, właściwości Value formantu SpinButton zostaje przypisana wartość wprowadzona w formancie TextBox. Jeżeli wprowadzona wartość nie jest numeryczna lub wykracza poza dopuszczalny zakres, nic się nie dzieje.

W omawianym przykładzie kliknięcie przycisku *OK* (formant o nazwie OKButton) powoduje przeniesienie wartości formantu SpinButton do aktywnej komórki. Procedura obsługi zdarzenia Click powiązanego z formantem CommandButton wygląda następująco:

```
Private Sub OKButton_Click()
    ' Wprowadzenie wartości do aktywnej komórki
    If CStr(SpinButton1.Value) = TextBox1.Text Then
        ActiveCell = SpinButton1.Value
        Unload Me
    Else
        MsgBox "Nieprawidłowa wartość.", vbCritical
        TextBox1.SetFocus
        TextBox1.SelStart = 0
        TextBox1.SelLength = Len(TextBox1.Text)
    End If
End Sub
```

Właściwość Tag

Wszystkie formularze *UserForm* i formanty posiadają właściwość o nazwie Tag. Właściwość ta nie reprezentuje niczego szczególnego i domyślnie jest pusta. Dzięki temu jednak możesz użyć tej właściwości do przechowywania różnych dodatkowych informacji.

Załóżmy, że w formularzu *UserForm* znajduje się kilka formantów *TextBox*. Od użytkownika może być wymagane wprowadzenie tekstu tylko do niektórych z nich. Do identyfikacji wymaganych pól możesz w takiej sytuacji użyć właściwości Tag wybranych formantów, na przykład przypisując jej ciąg znaków *Wymagane*. Później w trakcie pisania kodu źródłowego sprawdzającego poprawność danych wprowadzonych przez użytkownika można odwołać się do właściwości Tag.

Poniżej zamieszczono przykład funkcji sprawdzającej wszystkie formanty *TextBox* formularza *UserForm1* i zwracającej liczbę wymaganych formantów *TextBox*, które są puste. Jeżeli funkcja zwraca wartość większą niż 0, oznacza to, że nie wszystkie wymagane pola zostały wypełnione.

```
Function EmptyCount()
    Dim ctl As Control
    EmptyCount = 0
    For Each ctl In UserForm1.Controls
        If TypeName(ctl) = "TextBox" Then
            If ctl.Tag = "Wymagane" Then
                If ctl.Text = "" Then
                    EmptyCount = EmptyCount + 1
                End If
            End If
        End If
    Next ctl
End Function
```

W trakcie korzystania z formularzy *UserForm* prawdopodobnie przyjdą Ci na myśl inne zastosowania właściwości Tag.

Na koniec procedura sprawdza, czy tekst wprowadzony w formancie *TextBox* odpowiada wartości właściwości *Value* formantu *SpinButton*. Operacja taka jest konieczna na wypadek wprowadzenia niepoprawnych danych. Jeżeli na przykład użytkownik wprowadziłby w formancie *TextBox* łańcuch 3r, wartość właściwości *Value* formantu *SpinButton* nie zostałaby zmodyfikowana i w efekcie wynik umieszczony w aktywnej komórce nie byłby zgodny z oczekiwaniami. Wartość właściwości *Value* formantu *SpinButton* jest zamieniana na łańcuch przez funkcję *CStr*. Dzięki temu uzyskuje się gwarancję, że w przypadku porównywania wartości z łańcuchem nie zostanie wygenerowany błąd. Jeżeli wartość formantu *SpinButton* nie odpowiada zawartości formantu *TextBox*, zostanie wyświetlony komunikat błędu. Należy zwrócić uwagę, że uaktywniany jest obiekt *TextBox* i przy użyciu właściwości *SelStart* i *SelLength* zaznaczana jego zawartość. Dzięki temu użytkownik w bardzo prosty sposób może poprawić wprowadzone dane.

Odwoływanie się do formantów formularza *UserForm*

Przy korzystaniu z formantów formularza *UserForm* powiązany z nimi kod VBA procedur obsługi zdarzeń zazwyczaj znajduje się w oknie *Code* formularza. W takiej sytuacji, odwołując się do poszczególnych formantów, nie musisz posługiwać się ich pełnymi

nazwami kwalifikowanymi, ponieważ VBA domyślnie przyjmuje, że należą one do danego formularza.

Do formantów formularza *UserForm* możesz odwoływać się też z poziomu kodu źródłowego zawartego w podstawowym module VBA. W tym celu konieczne jest zdefiniowanie odwołania do formantu poprzez zawarcie w nim nazwy formularza *UserForm*. Dla przykładu przyjrzyjmy się poniższej procedurze znajdującej się w module VBA, która wyświetla jedynie formularz *UserForm* o nazwie *UserForm1*:

```
Sub GetData()
    UserForm1.Show
End Sub
```

Załóżmy, że formularz *UserForm1* zawiera formant *TextBox* o nazwie *TextBox1*, któremu chcesz przypisać domyślną wartość. Aby to zrobić, możesz zmodyfikować tę procedurę w następujący sposób:

```
Sub GetData()
    UserForm1.TextBox1.Value = "Jan Nowak"
    UserForm1.Show
End Sub
```

Innym sposobem na ustawienie wartości domyślnej jest użycie zdarzenia *Initialize* formularza *UserForm*. Instrukcje kodu źródłowego mogą zostać umieszczone wewnątrz procedury *UserForm_Initialize* znajdującej się w module formularza *UserForm*. Oto przykład takiego rozwiązania:

```
Private Sub UserForm_Initialize()
    TextBox1.Value = "Jan Nowak"
End Sub
```

Kolekcja Controls

Formanty formularza *UserForm* tworzą kolekcję. Na przykład poniższa instrukcja wyświetla liczbę formantów formularza *UserForm1*:

```
MsgBox UserForm1.Controls.Count
```

VBA nie przechowuje osobnych kolekcji dla poszczególnych typów formantów. Na przykład nie istnieje kolekcja formantów *CommandButton*. W razie potrzeby jednak, za pomocą funkcji *TypeName* można określić typ formantu. Poniższa procedura przy użyciu pętli *For Each ... Next* przetwarza kolekcję *Controls*, a następnie wyświetla liczbę formantów *CommandButton* formularza *UserForm1*:

```
Sub CountButtons()
    Dim cbCount As Integer
    Dim ctl As Control
    cbCount = 0
    For Each ctl In UserForm1.Controls
        If TypeName(ctl) = "CommandButton" Then _
            cbCount = cbCount + 1
    Next ctl
    MsgBox cbCount
End Sub
```

Gdy odwołanie do formantu znajduje się w module kodu źródłowego formularza *UserForm*, nie musisz używać nazwy formularza w odwołaniu. Z drugiej jednak strony używanie pełnych odwołań kwalifikowanych niesie ze sobą pewne korzyści — dzięki temu na przykład możesz korzystać z opcji *Auto List Members*, która pozwala na wybieranie nazw formantów z listy rozwijanej.



Zamiast stosowania rzeczywistej nazwy formularza *UserForm* preferowanym rozwiązaniem jest używanie słowa kluczowego *Me*. Dzięki temu po zmianie nazwy formularza *UserForm* nie będzie konieczna modyfikacja odwołań do niego, zawartych w kodzie źródłowym.

Dostosowywanie okna Toolbox do własnych wymagań

Po uaktywnieniu w edytorze VBE formularza *UserForm* jest wyświetlane okno *Toolbox* zawierające formanty, które można umieścić na formularzu. Jeżeli okno *ToolBox* nie jest widoczne, powinieneś wybrać z menu głównego edytora VBE polecenie *View/Toolbox*. W niniejszym podrozdziale zostaną omówione metody dostosowywania okna *Toolbox*.

Dodawanie nowych kart

Początkowo okno *Toolbox* zawiera jedną kartę. Aby do okna *Toolbox* dodać nową kartę, należy prawym przyciskiem myszy kliknąć już istniejącą i z menu podręcznego wybrać polecenie *New Page*. Po wybraniu z menu podręcznego polecenia *Rename* można też zmienić domyślną nazwę karty.

Dostosowywanie lub łączenie formantów

Bardzo przydatną opcją jest możliwość dostosowywania wybranych formantów do własnych potrzeb, a następnie zapisywanie ich do późniejszego użytku. Dla przykładu możesz utworzyć formant *CommandButton*, który będzie spełniał funkcję przycisku *OK*, następnie ustawić wartości dla takich właściwości jak *Width*, *Height*, *Caption*, *Default* i *Name* i wreszcie tak zmodyfikowany formant *CommandButton* przeciągnąć do okna *Toolbox*. W efekcie zostanie utworzony nowy formant, dostosowany do Twoich potrzeb. Aby zmienić domyślną nazwę nowego formantu lub jego ikonę, kliknij formant prawym przyciskiem myszy i wybierz odpowiednie polecenie z menu podręcznego.

W oknie *Toolbox* możesz również utworzyć nowy element składający się z wielu formantów, na przykład z dwóch formantów *CommandButton* reprezentujących przyciski *OK* i *Anuluj* formularza *UserForm*. Po dostosowaniu obu formantów do własnych wymagań należy je przeciągnąć do okna *Toolbox*. Przy użyciu tego nowego formantu możesz za jednym razem dodać dwa odpowiednio zmodyfikowane przyciski.

To samo dotyczy formantów pełniących funkcję kontenerów. Przykładowo możesz zdefiniować formant *Frame* i umieścić w nim cztery wyrównane i jednakowo od siebie oddalone formanty *OptionButton*. W celu stworzenia własnej wersji formantu *Frame* możesz następnie do okna *Toolbox* przeciągnąć obiekt *Frame*.

Aby ułatwić identyfikację formantów, kliknij wybrany formant prawym przyciskiem myszy i z menu podręcznego, które pojawi się na ekranie wybierz polecenie *Customize xxx* (gdzie *xxx* to nazwa formantu). Na ekranie pojawi się okno dialogowe, które pozwoli Ci zmienić tekst podpowiedzi ekranowej formantu, oraz edytować a nawet zmienić jego ikonę.

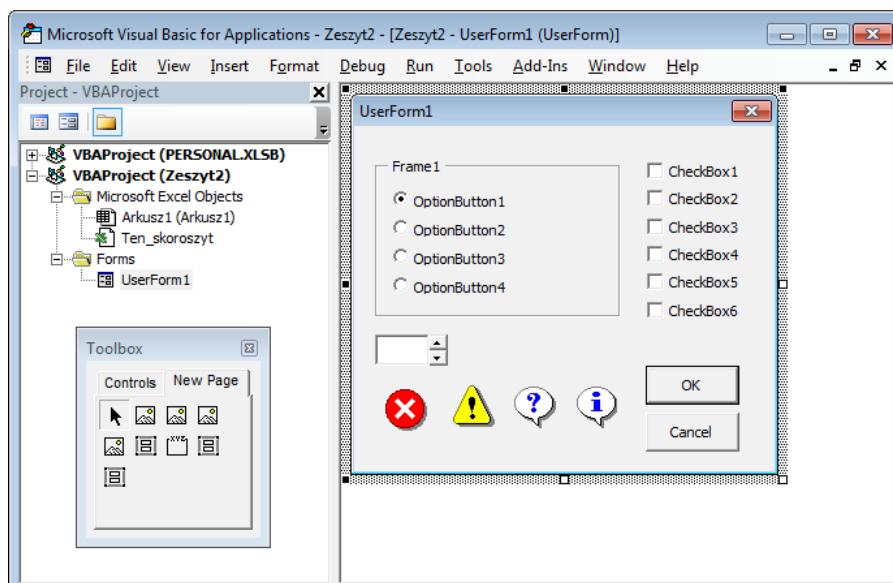


Istnieje możliwość umieszczenia dostosowanych kontrolek na oddzielnej karcie okna *Toolbox*. Dzięki temu można wyeksportować całą kartę i udostępnić ją innym użytkownikom Excela. Aby wyeksportować kartę okna *Toolbox*, należy ją kliknąć prawym przyciskiem myszy i z menu podręcznego wybrać polecenie *Export Page*.

Na rysunku 11.13 przedstawiono wygląd nowej karty okna *Toolbox*, na której znajduje się 8 niestandardowych formantów, utworzonych i dostosowanych przez użytkownika. Są to m.in.:

- Kontener Frame z czterema przyciskami *OptionButton*
- Pole tekstowe *TextBox* wraz z pokrętłem *Spinner*
- Grupa sześciu formantów *CheckBox*
- Ikona błędu krytycznego (X na czerwonym tle)
- Ikona wykrzyknika
- Ikona znaku zapytania
- Ikona informacji

Wymienione cztery ikony to te same obrazy, które są wyświetlane w różnych kontekstach przez funkcję *MsgBox*.



Rysunek 11.13. Okno Toolbox zawierające nową kartę z formantami

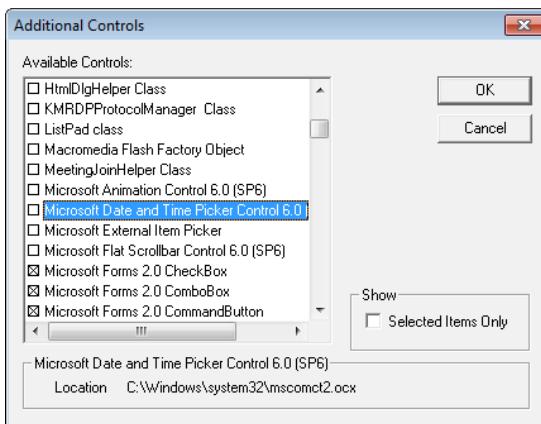


Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz plik *NoweFormanty.pag* zawierający kilka dostosowanych formantów, który możesz zaimportować do okna *Toolbox* jako nową kartę. Aby to zrobić, kliknij prawym przyciskiem myszy istniejącą kartę okna *Toolbox*, z menu podręcznego wybierz polecenie *Import Page* i następnie zlokalizuj i zaznacz odpowiedni plik o rozszerzeniu *.pag*.

Dodawanie nowych formantów ActiveX

Formularze *UserForm* mogą zawierać dodatkowe formanty ActiveX utworzone przez Microsoft lub innych producentów. Aby dodać do okna *Toolbox* dodatkowe formanty ActiveX, należy kliknąć okno prawym przyciskiem myszy i z menu wybrać polecenie *Additional Controls*. Na ekranie pojawi się okno dialogowe przedstawione na rysunku 11.14.

Rysunek 11.14.
Okno dialogowe
Additional Controls
umożliwia dodawanie
innych formantów
ActiveX



Okno dialogowe *Additional Controls* zawiera wszystkie formanty ActiveX zainstalowane w systemie. Aby dołączyć ikony wybranych formantów, powinieneś zaznaczyć wybrane formanty i nacisnąć przycisk *OK*.



Nie wszystkie formanty ActiveX zainstalowane w systemie będą poprawnie działać na formularzu *UserForm* Excela. Tak naprawdę większość z nich prawdopodobnie nie zadziała. Poza tym użycie niektórych formantów w aplikacji wymaga zakupu odpowiedniej licencji. Jeżeli użytkownik aplikacji nie dysponuje licencją dla takiego formantu, przy próbie jej użycia zostanie wygenerowany błąd.

Tworzenie szablonów formularzy UserForm

Bardzo często tworząc kolejne formularze *UserForm*, będziesz używał takich samych lub bardzo podobnych zestawów formantów. Na przykład każdy formularz *UserForm* może zawierać dwa formanty *CommandButton* spełniające funkcję przycisków *OK* i *Anuluj*. W ostatnim punkcie wyjaśniłem, w jaki sposób stworzyć nowy formant łączący oba dostosowane przyciski. Alternatywna metoda polega na stworzeniu szablonu formularza *UserForm* i wyeksportowaniu go, dzięki czemu możliwe będzie zimportowanie go w innych projektach. Kod źródłowy procedur obsługi zdarzeń powiązanych z formantami jest umieszczany w szablonie.

Emulowanie okien dialogowych Excela

Wgląd i zachowanie okien dialogowych systemu Windows różni się w zależności od programu. Projektując aplikację Excela, wszędzie, gdzie jest to możliwe, należy imitować wygląd okien dialogowych arkusza kalkulacyjnego.

Dobrą metodą nauczenia się tworzenia efektywnych okien dialogowych jest dokonanie próby skopiowania okna dialogowego Excela z uwzględnieniem jego najdrobniejszych szczegółów. Na przykład, powinieneś zadbać o to, aby zdefiniować wszystkie skróty klawiaturowe i zastosować identyczną kolejność tabulacji. Konieczne jest również przetestowanie powielonego okna i sprawdzenie, jak się zachowuje w różnych sytuacjach. Gwarantuję, że przeanalizowanie okien dialogowych Excela przyczyni się do zwiększenia Twoich umiejętności tworzenia okien.

Pamiętaj jednak, że funkcjonalności niektórych okien dialogowych Excela nie da się powięlić za pomocą formularzy *UserForm*. Na przykład nie będziesz w stanie utworzyć duplikatu okna dialogowego kreatora konwersji tekstu na kolumny, które normalnie możesz przywołać na ekran, przechodząc na kartę *DANE* i wybierając polecenie *Tekst jako kolumny*, znajdujące się w grupie opcji *Narzędzia danych*. Dzieje się tak, ponieważ w oknie tego kreatora są wykorzystywane formanty, które nie są dostępne dla użytkownika z poziomu języka VBA.

Na początku powinieneś utworzyć formularz *UserForm* zawierający wszystkie formanty i modyfikacje, które będą wymagane w innych projektach. Po sprawdzeniu, czy formularz *UserForm* jest zaznaczony, z menu *File* wybierz polecenie *Export File* lub naciśnij kombinację klawiszy *Ctrl+E*. Na ekranie pojawi się prośba o podanie nazwy pliku.

W celu załadowania zapisanego formularza *UserForm* w nowym projekcie z menu *File* wybierz opcję *Import File*.

Lista kontrolna tworzenia i testowania formularzy UserForm

Zanim formularz *UserForm* zostanie udostępniony końcowym użytkownikom, należy upewnić się, czy wszystko poprawnie działa. Poniższa lista kontrolna powinna pomóc w identyfikacji potencjalnych problemów.

- Czy podobne formanty są jednakowej wielkości?
- Czy odległość pomiędzy formantami jest identyczna?
- Czy zawartość okna dialogowego nie jest zbyt przytłaczająca? Jeżeli tak, przy użyciu formantu *MultiPage* można pogrupować formanty na poszczególnych kartach.
- Czy każdy formant może zostać uaktywniony przy użyciu skrótu klawiszowego?
- Czy nie doszło do powielenia skrótów klawiszowych?
- Czy poprawnie zdefiniowano kolejność tabulacji?
- Czy po wcisnięciu przez użytkownika klawisza *Esc* lub kliknięciu przycisku *Zamknij* formularza *UserForm* kod źródłowy języka VBA wykonuje właściwą operację?
- Czy w etykietach nie występują literówki?

- Czy okno dialogowe posiada odpowiedni tytuł?
- Czy okno dialogowe zostanie poprawnie wyświetcone w przypadku każdej rozdzielczości ekranu?
- Czy formanty są logicznie pogrupowane (według funkcji)?
- Czy formanty ScrollBar i SpinButton umożliwiają ustawienie tylko poprawnych wartości?
- Czy formularz *UserForm* korzysta z formantów, które mogą nie być zainstalowane na innych komputerach?
- Czy poprawnie zdefiniowano formanty ListBox (Single, Multi lub Extended)? Więcej szczegółowych informacji na temat formantów ListBox znajdziesz w rozdziale 12.

Rozdział 12.

Przykłady formularzy

UserForm

W tym rozdziale:

- Tworzenie formularza *UserForm* pełniącego funkcję prostego menu
- Zaznaczanie zakresów przy użyciu formularza *UserForm*
- Tworzenie okna powitalnego
- Zmiana rozmiaru formularza *UserForm* podczas wyświetlania
- Powiększanie i przewijanie arkusza przy użyciu formularza *UserForm*
- Różne techniki pracy z formantami *ListBox*
- Zastosowanie formantów zewnętrznych
- Zastosowanie formantów *MultiPage*
- Animowanie formantów *Label*

Tworzenie formularza *UserForm* pełniącego funkcję menu

W niektórych sytuacjach formularz *UserForm* może zostać użyty w roli menu. Inaczej mówiąc, formularz *UserForm* zostanie użyty do wyświetlania opcji, spośród których użytkownik menu będzie mógł dokonać wyboru. W tym podrozdziale zaprezentujemy dwie metody tworzenia menu — przy użyciu formantów *CommandButton* lub *ListBox*.



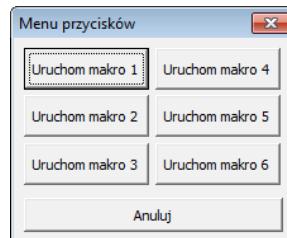
W rozdziale 13. zamieszczono dodatkowe przykłady bardziej zaawansowanych metod wykorzystania formularzy *UserForm*.

Zastosowanie w formularzu UserForm formantów CommandButton

Na rysunku 12.1 pokazano formularz *UserForm* zawierający formanty *CommandButton* i pełniący funkcję prostego menu.

Rysunek 12.1.

Okno dialogowe z formantami *CommandButton* spełniającymi funkcję menu



Utworzenie takiego formularza to bardzo łatwe zadanie, a powiązany z nim kod źródłowy jest dość prosty. Każdy formant *CommandButton* posiada własną procedurę obsługi zdarzenia. Na przykład procedura przedstawiona poniżej jest wykonywana po kliknięciu formantu *CommandButton1*:

```
Private Sub CommandButton1_Click()
    Me.Hide
    Call Makrol1
    Unload Me
End Sub
```

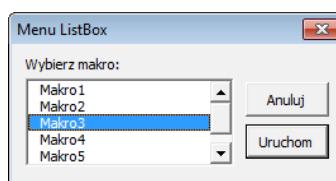
Procedura ukrywa formularz, wywołuje makro o nazwie *Makrol1* i następnie zamknie formularz *UserForm*. Z innymi przyciskami są powiązane podobne procedury obsługi zdarzeń.

Zastosowanie w formularzu UserForm formantu ListBox

Na rysunku 12.2 pokazano kolejny przykład, w którym w roli menu zastosowano formant *ListBox*.

Rysunek 12.2.

Okno dialogowe używające formantu *ListBox* jako prostego menu



Taki rodzaj menu jest nieco bardziej praktyczny, ponieważ zdecydowanie łatwiej można dodawać do menu nowe polecenia i nie wymaga to modyfikacji rozmiaru samego formularza *UserForm*. Przed wyświetleniem formularza *UserForm* wywoływana jest procedura obsługi zdarzenia *Initialize*. Procedura, której kod źródłowy zamieszczono poniżej, używa metody *AddItem* do utworzenia dodaje sześciu elementów na liście formantu *ListBox*.

```
Private Sub UserForm_Initialize()
    With ListBox1
        .AddItem "Makrol1"
```

```
.AddItem "Makro2"  
.AddItem "Makro3"  
.AddItem "Makro4"  
.AddItem "Makro5"  
.AddItem "Makro6"  
End With  
End Sub
```

Z przyciskiem *Wykonaj* powiązana jest procedura obsługi zdarzenia Click:

```
Private Sub ExecuteButton_Click()  
Select Case ListBox1.ListIndex  
Case -1  
    MsgBox "Wybierz makro z listy."  
    Exit Sub  
Case 0: Call Macro1  
Case 1: Call Macro2  
Case 2: Call Macro3  
Case 3: Call Macro4  
Case 4: Call Macro5  
Case 5: Call Macro6  
End Select  
Unload Me  
End Sub
```

Aby zidentyfikować wybrany przez użytkownika element, procedura używa właściwości ListIndex formantu ListBox. Jeżeli nie zostanie zaznaczona żaden element, wartością właściwości ListIndex będzie -1 i na ekranie zostanie wyświetlony odpowiedni komunikat.

Oprócz tego, ten formularz posiada procedurę, która obsługuje podwójne kliknięcie na dowolnym elemencie listy formantu ListBox. Dwukrotne kliknięcie lewym przyciskiem myszy wybranego elementu listy powoduje uruchomienie powiązanego z nim makra.



Skoroszyt z tymi przykładami (*Menu — formularze UserForm.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



W rozdziale 13. znajdziesz podobny przykład, w którym formularza *UserForm* użyjemy do symulowania paska narzędzi.

Zaznaczanie zakresów przy użyciu formularza *UserForm*

Wiele spośród wbudowanych okien dialogowych Excela umożliwia użytkownikowi zdefiniowanie zakresu. Na przykład okno dialogowe *Szukanie wyniku* prosi użytkownika o zaznaczenie dwóch zakresów (aby przywołać to okno na ekran, przejdź na kartę *DANE*, naciśnij przycisk *Analiza warunkowa*, znajdujący się w grupie opcji *Narzędzia danych*, i następnie z menu podręcznego wybierz polecenie *Szukaj wyniku*). Użytkownik może wprowadzić odwołanie do zakresu bezpośrednio w polu tekstowym bądź użyć myszy do zaznaczenia zakresu bezpośrednio na arkuszu.

Jest to bardzo wygodne rozwiązanie, a dzięki formantowi RefEdit formularze *UserForm* również posiadają taką możliwość. Formant RefEdit nie wygląda dokładnie tak, jak formant umożliwiający określenie zakresu, używany powszechnie w oknach dialogowych Excela, ale działa w bardzo podobny sposób. Jeżeli użytkownik kliknie mały przycisk z prawej strony formantu, okno dialogowe na chwilę zniknie i pojawi się niewielkie okno umożliwiające określenie zakresu. Podobnie dzieje się w przypadku wbudowanych okien dialogowych Excela.



Uwaga

Niestety z używaniem formantu RefEdit związanym jest kilka drobnych błędów, które do tej pory nie zostały naprawione. Przekonasz się, że ten formant nie pozwala na zaznaczanie zakresów przy użyciu skrótów klawiszowych (na przykład naciśnięcie klawisza *End* i następnie kombinacji klawiszy *Shift+↓* nie spowoduje zaznaczenia wszystkich komórek do końca kolumny). Po naciśnięciu małego przycisku po prawej stronie formantu (aby ukryć okno dialogowe na czas zaznaczania zakresu) jesteś skazany na korzystanie z myszy — do zaznaczania zakresu nie możesz użyć klawiatury.

Na rysunku 12.3 pokazano formularz *UserForm* zawierający formant RefEdit. Okno dialogowe umożliwia wykonanie prostych operacji matematycznych na wszystkich komórkach zaznaczonego zakresu, w których nie umieszczono formuł (i które nie są puste). Rodzaj wykonywanych obliczeń zależy od wybranej opcji (formanty *OptionButton*).

A	B	C	D	E	F	G	H	I	J	K	L
1	-17	-15	-78	9	-53						
2	96	-34	-73	4	48	-75					
3	65	42	-38	-90	37	79					
4	-16	4	-72	32	-77	48					
5	68	77	89	-79	-89	-42					
6	70	37	-45	-90	-31	-32					
7	-13	-8	40	-26	57	17					
8	94	90	57	56	-38	33					
9	-75	79	-74	-31	-91	-32					
10	90	18	-55	21	49	-66					
11	10	91	-12	-33	65	5					
12	-67	-32	22	53	-27	-23					
13	-64	-80	42	95	-70	-74					
14	70	20	-44	-2	-71	29					
15	69	5	17	-91	89	-13					
16	78	-62	94	-52	48	-62					
17											
18											
19											

Rysunek 12.3. Formant RefEdit umożliwia użytkownikowi zaznaczenie zakresu



Skoroszyt z tym przykładem (*Zaznaczanie zakresu.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Oto kilka zagadnień, o których należy pamiętać przy stosowaniu formantu RefEdit:

- Formant RefEdit zwraca łańcuch tekstowy reprezentujący adres zakresu. W razie potrzeby możesz zamienić taki łańcuch tekstu na obiekt klasy Range. Aby to zrobić, użyj następującego polecenia:

```
Set UserRange = Range(RefEdit1.Text)
```

- Dobrym nawykiem jest wcześniejsze zainicjowanie formantu RefEdit, tak aby domyślnie wyświetlał aktualnie zaznaczony zakres. Możesz to zrobić poprzez umieszczenie w procedurze UserForm_Initialize następującego polecenia:

```
RefEdit1.Text = ActiveWindow.RangeSelection.Address
```

- Aby uniknąć problemów, nie powinieneś umieszczać formantu RefEdit wewnętrz kontenerów Frame lub MultiPage. Umieszczenie formantu RefEdit w jednym z takich kontenerów może spowodować zawieszenie się Excela.
- Nigdy nie zakładaj, że formant RefEdit zawsze zwróci poprawny adres zakresu. Zaznaczenie myszą zakresu nie jest jedną metodą umieszczania adresu w formancie. Użytkownik może wprowadzić dowolny tekst, jak również zmodyfikować lub usunąć łańcuch zawarty w formancie. W związku z tym konieczne jest sprawdzenie, czy adres zakresu jest poprawny. Poniżej zamieszczamy kod źródłowy sprawdzający poprawność zakresu. Jeżeli wprowadzony zakres jest niepoprawny, wyświetlany jest odpowiedni komunikat i użytkownik może ponownie wprowadzić zakres (fokus jest ponownie ustawiany na formant RefEdit).

```
On Error Resume Next
Set UserRange = Range(RefEdit1.Text)
If Err.Number <> 0 Then
    MsgBox "Zaznaczyłeś nieprawidłowy zakres."
    RefEdit1.SetFocus
    Exit Sub
End If
On Error GoTo 0
```

- Podczas zaznaczania zakresu przy użyciu formantu RefEdit, użytkownik może kliknąć karty arkuszy. W związku z tym nie możesz z góry przyjąć, że zaznaczony zakres znajduje się w aktywnym arkuszu. Jeżeli użytkownik wybierze zakres znajdujący się na innym arkuszu, adres takiego zakresu zostanie poprzedzony nazwą arkusza. Oto przykład:

```
Arkusz2!$A$1:$C$4
```

- Aby pobrać adres jednej komórki zakresu zazначенego przez użytkownika, powinieneś wybrać górną lewą komórkę zakresu, używając poniższego polecenia:

```
Set OneCell = Range(RefEdit1.Text).Range("A1")
```

W rozdziale 10. dowiesz się, jak przy użyciu funkcji InputBox języka VBA umożliwić użytkownikowi zaznaczenie zakresu.



Tworzenie okna powitalnego

Niektóre aplikacje podczas uruchamiania wyświetlają okno zawierające różne informacje. Takie okno jest często nazywane *oknem powitalnym* (ang. *splash screen*). Z pewnością znasz okno powitalne Excela pojawiające się na kilka sekund podczas uruchamiania programu.

Okno powitalne własnej aplikacji Excela można utworzyć przy użyciu formularza *UserForm*. W poniższym przykładzie zaprezentowano formularz *UserForm*, który jest wyświetlane automatycznie i zamknięty po pięciu sekundach.



Skoroszyt z tym przykładem (*Okno powitalne.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Aby utworzyć takie okno powitalne, wykonaj następujące polecenia:

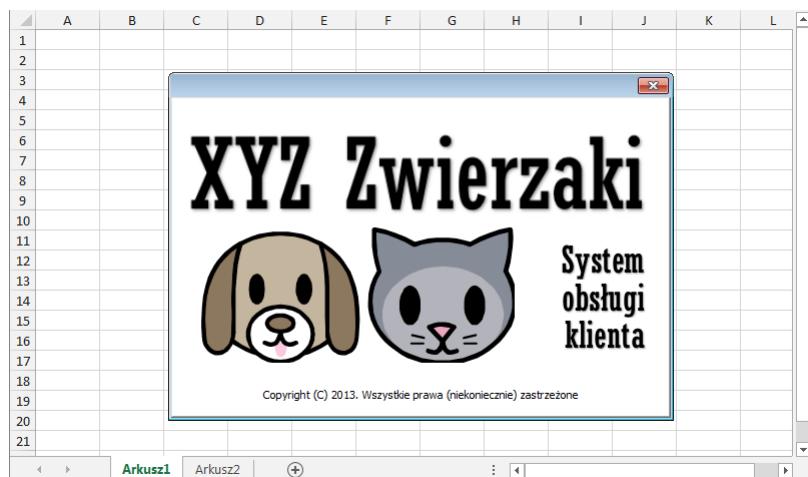
1. Utwórz nowy skoroszyt.
2. Uaktywnij okno edytora Visual Basic i w projekcie umieść nowy formularz *UserForm*.
3. W naszym przykładzie założyliśmy, że formularz nosi nazwę po prostu *UserForm1*.
4. Umieść na formularzu *UserForm1* żądane formanty.

Na przykład, możesz użyć formantu *Image* zawierającego logo Twojej firmy.

Na rysunku 12.4 przedstawiono właśnie takie okno powitalne.

Rysunek 12.4.

Okno powitalne jest przez chwilę wyświetlane w trakcie otwierania skoroszytu



5. Umieść poniższą procedurę w module kodu źródłowego obiektu *ThisWorkbook*:

```
Private Sub Workbook_Open()
    UserForm1.Show
End Sub
```

6. Wstaw poniższą procedurę w module kodu źródłowego formularza *UserForm1*.

7. Aby zmienić domyślne, pięciosekundowe opóźnienie, odpowiednio zmodyfikuj argument wywołania funkcji *TimeValue*:

```
Private Sub UserForm_Activate()
    Application.OnTime Now +
        TimeValue("00:00:05"), "KillTheForm"
End Sub
```

8. Wstaw poniższą procedurę w podstawowym module kodu VBA:

```
Private Sub KillTheForm()
    Unload UserForm1
End Sub
```

Po otwarciu skoroszytu jest wykonywana procedura Workbook_Open, która wyświetla formularz *UserForm* (punkt 4.). W trakcie tej operacji powiązane z formularzem zdarzenie Activate wywołuje procedurę UserForm_Activate (punkt 5.). W celu wykonania w określonym czasie procedury KillTheForm procedura Workbook_Open korzysta z metody OnTime. W naszym przypadku opóźnienie wynosi pięć sekund od wygenerowania zdarzenia Activate. Procedura KillTheForm zamyka i usuwa z pamięci formularz *UserForm*.

9. Opcjonalnie możesz dodać niewielki formant CommandButton o nazwie CancelButton, ustawić dla jego właściwości Cancel wartość True, a następnie w module kodu źródłowego formularza umieścić poniższą procedurę obsługi zdarzenia:

```
Private Sub CancelButton_Click()
    KillTheForm
End Sub
```

Dzięki takiemu rozwiązaniu użytkownik przez wcisnięcie klawisza *Esc* będzie mógł zamknąć okno powitalne jeszcze przed upływem przyjętego czasu opóźnienia. Taki przycisk możesz nawet umieścić za innym obiektem formularza, tak że nie będzie widoczny.



Pamiętaj, że okno powitalne nie jest wyświetlane do momentu całkowitego załadowania skoroszytu. Innymi słowy, jeżeli chcesz, aby użytkownik miał na co popatrzeć podczas ładowania skoroszytu, takie rozwiązanie nie spełni pokładanych w nim nadziei.



Jeżeli w trakcie uruchamiania aplikacji musi wykonać kilka procedur języka VBA, możesz wyświetlić niemodalny formularz *UserForm*, dzięki czemu podczas wyświetlania okna dialogowego działanie programu nie zostanie wstrzymane. Aby to zrobić, powinieneś zmodyfikować procedurę Workbook_Open w następujący sposób:

```
Private Sub Workbook_Open()
    UserForm1.Show vbModeless
    ' W tym miejscu znajduje się reszta kodu
End Sub
```

Wyłączanie przycisku Zamknij formularza UserForm

Kliknięcie przycisku *Zamknij* (ikona *X* w prawym górnym narożniku) powoduje zamknięcie formularza *UserForm*. Zdarzają się jednak sytuacje, kiedy taka możliwość jest niewskazana, na przykład wówczas, gdy formularz *UserForm* ma być zamykany tylko po kliknięciu określonego formantu *CommandButton*.

Co prawda, w rzeczywistości nie możesz całkowicie wyłączyć przycisku *Zamknij*, ale monitorując zdarzenie QueryClose formularza *UserForm*, możesz uniemożliwić użytkownikowi zamknięcie okna przez kliknięcie ikony *X*.

Poniższa procedura znajdująca się w module kodu źródłowego formularza *UserForm* jest wykonywana przed jego zamknięciem, czyli po wystąpieniu zdarzenia *QueryClose*:

```
Private Sub UserForm_QueryClose
    (Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
        MsgBox "Aby zamknąć okno, naciśnij przycisk OK."
        Cancel = True
    End If
End Sub
```

Procedura *UserForm_QueryClose* używa dwóch argumentów. Argument *CloseMode* zawiera wartość identyfikującą przyczynę wystąpienia zdarzenia *QueryClose*. Jeżeli wartością argumentu *CloseMode* jest *vbFormControlMenu* (wbudowana stała), oznacza to, że użytkownik naciągnął przycisk *Zamknij*. W takim przypadku zostanie wyświetlony komunikat, a argument *Cancel* jest ustawiany na *True* i formularz nie jest zamknięty.



Skoroszyt z tym przykładem (*Zdarzenie QueryClose.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

W sieci



Pamiętaj, że w celu przerwania wykonywania makra użytkownik może wcisnąć kombinację klawiszy *Ctrl+Break*. W przypadku powyższej procedury wciśnięcie tej kombinacji klawiszy w trakcie wyświetlania formularza *UserForm* spowoduje jego zamknięcie. Aby temu zapobiec, przed wyświetleniem formularza *UserForm* należy wykonać następującą instrukcję:

```
Application.EnableCancelKey = xlDisabled
```

Wcześniej jednak należy upewnić się, że aplikacja nie ma błędów. W przeciwnym razie, jeżeli w kodzie źródłowym wystąpi przypadkowa nieskończona pętla, nie będzie możliwe przerwanie jej działania.

Zmiana wielkości formularza UserForm

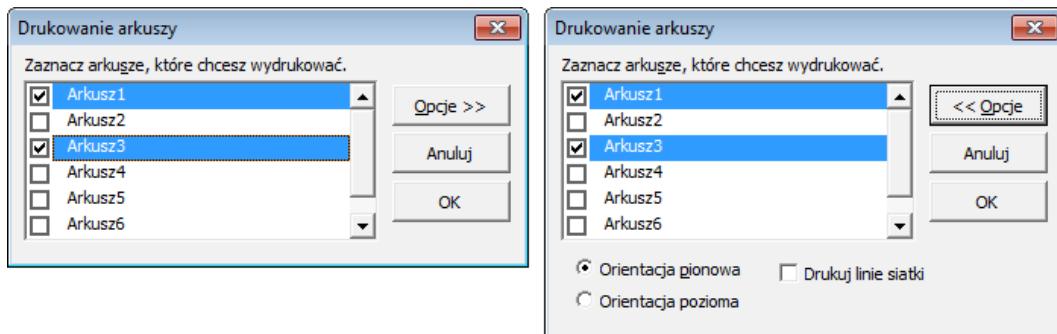
Wiele aplikacji korzysta z okien dialogowych, które zmieniają swoją wielkość. Przykładem takiego okna dialogowego jest okno *Znajdowanie i zamianianie*, które po kliknięciu przycisku *Opcje* zmienia wysokość (aby je przywołać na ekran, przejdź na kartę *NARZĘDZIA GŁÓWNE*, naciśnij przycisk *Znajdź i zaznacz*, znajdujący się w grupie opcji *Edytowanie*, i wybierz z menu podręcznego polecenie *Zamień*).

Poniższy przykład demonstruje, w jaki sposób dynamicznie modyfikować wielkość formularza *UserForm*. Zmiana wielkości okna dialogowego odbywa się poprzez modyfikację wartości właściwości *Width* lub *Height* obiektu *UserForm*. Nasz przykład wyświetla listę arkuszy w aktywnym skoroszycie i pozwala użytkownikowi wybrać, które arkusze będą drukowane.



W rozdziale 13. znajdziesz przykład, w którym użytkownik może zmienić rozmiary okna formularza *UserForm* poprzez przeciągnięcie przy użyciu myszy prawego dolnego narożnika okna.

Na rysunku 12.5 pokazano dwa stany okna dialogowego — pierwszy, który ilustruje początkowy wygląd okna, i drugi, gdzie zaprezentowano wygląd tego samego okna po kliknięciu przez użytkownika przycisku *Opcje*. Zauważ, jak w zależności od wielkości okna zmienia się etykieta tego przycisku.



Rysunek 12.5. Proste okno dialogowe przed i po wyświetleniu dodatkowych opcji

W trakcie tworzenia formularza *UserForm* należy ustawić dla niego jak największy rozmiar, tak aby wszystkie formanty swobodnie się na nim mieściły. Następnie w celu ustawienia domyślnej wielkości (mniejsze okno) formularza należy użyć procedury *UserForm_Initialize*.

W procedurze zostały użyte dwie stałe, zdefiniowane na początku modułu kodu:

```
Const SmallSize As Integer = 124
Const LargeSize As Integer = 164
```

Poniżej przedstawiamy procedurę obsługi zdarzenia, która jest wykonywana po kliknięciu formantu CommandButton o nazwie *OptionsButton*:

```
Private Sub OptionsButton_Click()
    If OptionsButton.Caption = "Opcje >>" Then
        Me.Height = LargeSize
        OptionsButton.Caption = "<< Opcje"
    Else
        Me.Height = SmallSize
        OptionsButton.Caption = "Opcje >>"
    End If
End Sub
```

Procedura sprawdza wartość właściwości *Caption* formantu *CommandButton* i ustawia odpowiednią wartość dla właściwości *Height* obiektu *UserForm*.



Gdy formanty nie są wyświetlane, ponieważ znajdują się poza widoczną częścią formularza *UserForm*, można je uaktywnić przy użyciu klawiszy skrótów, które zostały z nimi powiązane. W tym przykładzie użytkownik może wcisnąć kombinację klawiszy *Alt+L* (uaktywnia opcję *Orientacja pozioma*) nawet wtedy, gdy opcja nie jest widoczna. W celu zablokowania dostępu do opcji, które nie są wyświetlane, należy napisać kod źródłowy wyłączający formanty, które w danej chwili nie są widoczne.

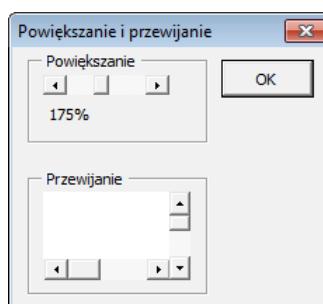


Skoroszyt z tym przykładem (*Zmiana rozmiaru formularza.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Powiększanie i przewijanie arkusza przy użyciu formularza UserForm

Przykład, który omawiamy w tym podrozdziale, ilustruje sposób użycia formantu ScrollBar do przewijania i powiększania zawartości arkusza po wyświetleniu okna dialogowego. Na rysunku 12.6 pokazano przykładowe okna dialogowe. Po wyświetleniu formularza *UserForm* użytkownik może modyfikować wartość wskaźnika powiększenia arkusza (w zakresie od 10 do 400% przy użyciu formantu ScrollBar w górnej części okna). Dwa formanty ScrollBar umieszczone w dolnej części okna dialogowego umożliwiają użytkownikowi przewijanie arkusza w pionie lub poziomie.

Rysunek 12.6.
Formanty ScrollBar
umożliwiają
powiększanie
i przewijanie arkusza



Skoroszyt z tym przykładem (*Powiększanie i przewijanie arkusza.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pzw.htm>).

Jak sam się przekonasz, kod źródłowy naszej procedury jest niezwykle prosty. Formanty są inicjowane przy użyciu następującej procedury *UserForm_Initialize*:

```
Private Sub UserForm_Initialize()
    LabelZoom.Caption = ActiveWindow.Zoom & "%"
    ' Powiększanie
    With ScrollBarZoom
        .Min = 10
        .Max = 400
        .SmallChange = 1
        .LargeChange = 10
        .Value = ActiveWindow.Zoom
    End With

    ' Przewijanie w poziomie
    With ScrollBarColumns
        .Min = 1
        .Max = ActiveSheet.UsedRange.Columns.Count
        .Value = ActiveWindow.ScrollColumn
        .LargeChange = 25
        .SmallChange = 1
    End With

    ' Przewijanie w pionie
    With ScrollBarRows
        .Min = 1
    End With
End Sub
```

```
.Max = ActiveSheet.UsedRange.Rows.Count  
.Value = ActiveWindow.ScrollRow  
.LargeChange = 25  
.SmallChange = 1  
End With  
End Sub
```

Procedura modyfikuje wybrane właściwości formantów ScrollBar, korzystając z wartości ustawianych przez użytkownika w aktywnym oknie formularza.

Po uaktywnieniu formantu ScrollBarZoom wykonywana jest procedura ScrollBarZoom_Change, która dla właściwości Value formantu ScrollBar ustawia wartość właściwości Zoom obiektu ActiveWindow (kod procedury przedstawiamy poniżej). Dodatkowo w celu wyświetlenia aktualnego wskaźnika powiększenia modyfikowany jest formant Label.

```
Private Sub ScrollBarZoom_Change()  
    With ActiveWindow  
        .Zoom = ScrollBarZoom.Value  
        LabelZoom = .Zoom & "%"  
    End With  
End Sub
```

Przewijaniem arkusza sterują dwie poniższe procedury, które właściwościom ScrollRow lub ScrollColumns obiektu ActiveWindow nadają wartość pobieraną z właściwości Value formantu ScrollBar:

```
Private Sub ScrollBarColumns_Change()  
    ActiveWindow.ScrollColumn = ScrollBarColumns.Value  
End Sub  
  
Private Sub ScrollBarRows_Change()  
    ActiveWindow.ScrollRow = ScrollBarRows.Value  
End Sub
```



Zamiast zdarzenia Change zastosowanego w poprzednich procedurach można użyć zdarzenia Scroll. Różnica polega na tym, że zdarzenie Scroll jest generowane po przeciągnięciu suwaka formantu ScrollBar. W efekcie operacje powiększania i przewijania arkusza są płynne i realizowane bez skokowych zmian wartości. Aby zastosować zdarzenie Scroll, wystarczy w nazwie procedury w miejscu słowa Change wstawić Scroll.

Zastosowania formantu ListBox

Formant ListBox jest wyjątkowo wszechstronny, ale korzystając z niego, należy zachować ostrożność. W tym punkcie omówimy kilka prostych przykładów ilustrujących najczęstsze techniki użycia formantu ListBox.



W większości przypadków techniki omówione w tym punkcie dotyczą też formantu ComboBox.

Poniżej zamieszczamy listę kilku zagadnień, o których powinieneś pamiętać podczas pracy z formantami `ListBox`. Wiele z tych zagadnień zostało zilustrowanych przykładami w dalszej części tego rozdziału.

- Elementy listy formantu `ListBox` mogą być pobierane z zakresu komórek (określonego przez właściwość `RowSource`) lub dodane przy użyciu kodu języka VBA (przy użyciu metody `AddItem`).
- Formant `ListBox` obsługuje pojedyncze i wielokrotne zaznaczenia. Do określenia rodzaju dozwolonych zaznaczeń powinieneś użyć właściwości `MultiSelect`.
- Jeżeli formant `ListBox` został tak zdefiniowany, aby nie obsługiwać wielokrotnych zaznaczeń, to jego właściwość `Value` może zostać połączona z komórką arkusza przy użyciu właściwości `ControlSource`.
- Możliwe jest wyświetlenie formantu `ListBox` bez zaznaczonych elementów (w takiej sytuacji wartością właściwości `ListIndex` będzie -1). Jednak po wybraniu danego elementu użytkownik nie będzie mógł usunąć zaznaczenia, będzie mógł tylko zmienić zaznaczony element. Wyjątkiem jest sytuacja, gdy wartością właściwości `MultiSelect` jest `True`.
- Formant `ListBox` może zawierać wiele kolumn (definiowanych przez właściwość `ColumnCount`), a nawet opisowe nagłówki kolumn (definiowanych przez właściwość `ColumnHeads`).
- Wysokość formantu `ListBox` wyświetlonego w tworzonym formularzu `UserForm` nie zawsze jest taka sama jak jego wysokość po otwarciu okna formularza uruchomionej aplikacji.
- Elementy listy formantu `ListBox` mogą mieć postać formantów `CheckBox` (w przypadku zaznaczenia wielokrotnego) lub `OptionButton` (w przypadku zaznaczenia pojedynczego). O wyglądzie elementów listy decyduje właściwość `ListStyle`.

Więcej szczegółowych informacji na temat właściwości i metod formantu `ListBox` znajdziesz w systemie pomocy programu Excel.

Tworzenie listy elementów formantu `ListBox`

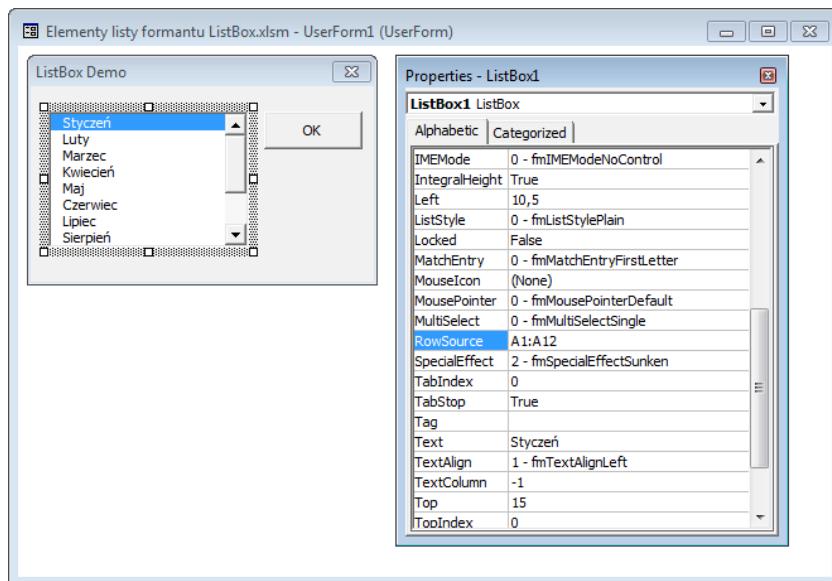
Zanim wyświetlisz formularz `UserForm` używający formantu `ListBox`, musisz wypełnić listę odpowiednimi elementami. Elementy listy mogą zostać umieszczone w formancie `ListBox` już podczas projektowania (za pomocą elementów zapisanych w komórkach zakresu arkusza) lub po uruchomieniu aplikacji (za pomocą instrukcji języka VBA).

W dwóch przykładach zamieszczonych w tym punkcie przyjęto następujące założenia:

- formularz `UserForm` nosi nazwę `UserForm1`;
- formularz `UserForm` zawiera formant `ListBox` o nazwie `ListBox1`;
- skoroszyt zawiera arkusz `Arkusz1` i zakres `A1:A12` przechowujący elementy listy, która zostanie wyświetlona w formancie `ListBox`.

Tworzenie listy elementów formantu ListBox podczas projektowania

Aby dodać elementy listy formantu ListBox podczas projektowania, kolejne elementy listy muszą być przechowywane w wybranym zakresie komórek arkusza. Aby określić zakres zawierający listę elementów formantu ListBox, powinieneś użyć właściwości RowSource. Na rysunku 12.7 pokazano okno *Properties* formantu ListBox. Wartością właściwości RowSource jest odwołanie Arkusz1!A1:A12. Po wyświetleniu formularza *UserForm* lista formantu ListBox będzie zawierała 12 elementów, pobieranych z tego zakresu. Podczas projektowania elementy listy pojawiają się na formancie ListBox bezpośrednio po zdefiniowaniu wartości (zakresu) właściwości RowSource.



Rysunek 12.7. Ustawianie wartości właściwości RowSource w trakcie projektowania



W zdecydowanej większości przypadków, ustawiając wartość właściwości RowSource, należy uwzględnić nazwę arkusza. W przeciwnym razie formant ListBox używa zdefiniowanego zakresu znajdującego się w aktywnym arkuszu. W niektórych przypadkach może być konieczne zastosowanie pełnego, kwalifikowanego odwołania do zakresu uwzględniającego nazwę skoroszytu. Oto przykład:

[Zeszyt1.xlsx]Arkusz1!A1:A12

Lepszym rozwiązaniem jest nadanie nazwy zakresowi i użycie jej w kodzie źródłowym. Dzięki temu na pewno zostanie użyty właściwy zakres nawet wtedy, gdy powyżej niego użytkownik usunie bądź doda nowe wiersze.

Tworzenie listy elementów formantu ListBox po uruchomieniu aplikacji

Aby po uruchomieniu aplikacji utworzyć listę elementów formantu ListBox, możesz wybrać jedno z dwóch rozwiązań:

- przy użyciu odpowiedniego polecenia VBA określić wartość (zakres) właściwości RowSource;
- utworzyć procedurę, który za pomocą metody AddItem utworzy poszczególne elementy listy formantu ListBox.

Jak możesz tego oczekwać, wartość właściwości RowSource zamiast przy użyciu okna *Properties* może być ustawiana za pomocą odpowiednich poleceń VBA. Na przykład poniższa procedura ustawia wartość właściwości RowSource formantu ListBox jeszcze przed wyświetleniem formularza *UserForm*. W tym przypadku elementy listy formantu są przechowywane w komórkach zakresu Kategorie znajdującego się w arkuszu Budżet:

```
UserForm1.ListBox1.RowSource = "Budżet!Kategorie"
UserForm1.Show
```

Jeżeli elementy listy formantu ListBox nie są przechowywane w zakresie arkusza, w celu ich dodania przed wyświetleniem okna dialogowego można napisać odpowiednią procedurę języka VBA. Poniższa procedura przy użyciu metody AddItem tworzy listę elementów formantu ListBox (lista miesięcy).

```
Sub ShowUserForm2()
    ' Wypełnienie formantu ListBox
    With UserForm1.ListBox1
        .RowSource = ""
        .AddItem "Styczeń"
        .AddItem "Luty"
        .AddItem "Marzec"
        .AddItem "Kwiecień"
        .AddItem "Maj"
        .AddItem "Czerwiec"
        .AddItem "Lipiec"
        .AddItem "Sierpień"
        .AddItem "Wrzesień"
        .AddItem "Październik"
        .AddItem "Listopad"
        .AddItem "Grudzień"
    End With
    UserForm1.Show
End Sub
```



Zwróć uwagę, że w naszej procedurze właściwości RowSource jest przypisywany pusty ciąg znaków. Ma to na celu uniknięcie możliwości wystąpienia błędu, który mógłby zaistnieć, jeżeli wcześniej w oknie *Properties* dla właściwości RowSource zostanie ustawiona jakaś wartość. Przy próbie utworzenia nowych elementów listy formantu ListBox, w przypadku którego wartością właściwości RowSource nie jest pusty łańcuch, zostanie wygenerowany błąd informujący o braku dostępu.

Metody AddItem możesz również użyć do utworzenia listy elementów formantu ListBox na podstawie zakresu komórek. Oto przykład kodu wypełniającego listę elementów formantu ListBox zawartością komórek zakresu A1:A12 arkusza Arkusz1:

```
For Row = 1 To 12
    UserForm1.ListBox1.AddItem Sheets("Arkusz1").Cells(Row, 1)
Next Row
```

Użycie właściwości List jest jeszcze łatwiejsze. Poniżej przedstawiamy polecenie, którego efekt działania jest identyczny jak w przypadku poprzedniej pętli:

```
UserForm1.ListBox1.List = Application.Transpose(Sheets("Arkusz1").  
Range("A1:A12"))
```

Zwróć uwagę na fakt, że użyliśmy tutaj funkcji Transpose, ponieważ właściwość List oczekuje, że dane będą zapisane w tablicy poziomej, a nasz zakres komórek to kolumna, a nie wiersz.

Jeżeli dane są przechowywane w tablicy jednowymiarowej, również możesz użyć właściwości List. Założmy, że istnieje tablica MojaLista zawierająca 50 elementów. Poniższe polecenie utworzy dla formantu ListBox listę złożoną z 50 elementów:

```
UserForm1.ListBox1.List = MojaLista
```



Skoroszyt z tym przykładem (Elementy listy formantu ListBox.xls) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Tworzenie unikatowych elementów listy formantu ListBox

W niektórych przypadkach może być konieczne wypełnienie listy formantu ListBox elementami *unikatowymi* (czyli takimi elementami, które się nie powtarzają). Założymy, że w arkuszu są przechowywane dane o klientach. W jednej z kolumn mogą być przechowywane identyfikatory stanów USA (patrz rysunek 12.8). Naszym zadaniem jest wypełnienie listy elementów formantu ListBox identyfikatorami nazw stanów powiązanych z klientami, ale bez ich powtarzania.

Rysunek 12.8.
Do wypełnienia listy formantu ListBox elementami unikatowymi używamy obiektu klasy Collection

	A	B	C	D	E	F	G	H
1	Nr klienta	Stan						
2	1001	CA						
3	1002	OH						
4	1003	FL						
5	1004	NY						
6	1005	MA						
7	1006	PA						
8	1007	CA						
9	1008	IL						
10	1009	AZ						
11	1010	IN						
12	1011	CA						
13	1012	FL						
14	1013	NV						
15	1014	NJ						
16	1015	MN						
17	1016	CA						
18	1017	NY						
19	1018	IL						
20	1019	MD						
21	1020	PA						
22	1021	FL						
23	1022	FL						

Wybierz element
Liczba elementów unikatowych: 41

CA
OH
FL
NY
MA
PA
IL
AZ

Jedna z szybkich i efektywnych technik polega na zastosowaniu obiektu klasy Collection. Aby do kolekcji dodać nowe elementy, należy użyć polecenia o następującej składni:

obiekt.Add pozycja, klucz, przed, po

Jeżeli zostanie użyty argument *klucz*, jego wartość musi być unikatowym łańcuchem tekstowym pełniącym funkcję niezależnego klucza stosowanego przy uzyskiwaniu dostępu do elementu zbioru. Istotne jest tutaj słowo *unikatowym*. Przy próbie umieszczenia w zbiorze klucza, który nie jest unikatowy, pojawi się błąd i pozycja nie zostanie dodana. Zdarzenie to może zostać wykorzystane do utworzenia kolekcji składającej się wyłącznie z unikatowych elementów.

Procedura przedstawiona poniżej rozpoczyna działanie od zadeklarowania nowego obiektu NoDuplicates klasy Collection. Zakładamy, że zakres o nazwie Data zawiera listę elementów, z których niektóre mogą się powtarzać.

Procedura przetwarza w pętli poszczególne komórki zakresu i próbuje dodać do kolekcji NoDuplicates przechowywane w komórkach wartości. Wartość komórki jest też używana przez argument *klucz* (po zamianie na łańcuch). Instrukcja On Error Resume Next powoduje, że interpreter języka VBA ignoriuje błąd pojawiający się w sytuacji, gdy klucz nie jest unikatowy. Po wystąpieniu błędu, zgodnie z oczekiwaniem, element nie zostanie umieszczony w kolekcji. Następnie procedura przenosi elementy z kolekcji NoDuplicates do listy formantu ListBox. Formularz UserForm zawiera też formant Label wyświetlający liczbę elementów unikatowych.

```
Sub RemoveDuplicates()
    Dim AllCells As Range, Cell As Range
    Dim NoDuplicates As New Collection

    On Error Resume Next
    For Each Cell In Range("Stany")
        NoDuplicates.Add Cell.Value, CStr(Cell.Value)
    Next Cell
    On Error GoTo 0

    ' Dodawanie unikatowych elementów do listy formantu ListBox
    For Each Item In NoDuplicates
        UserForm1.ListBox1.AddItem Item
    Next Item

    ' Wyświetlanie liczby pozycji
    UserForm1.Label1.Caption =
        "Elementy unikatowe: " & NoDuplicates.Count

    ' Wyświetlanie formularza UserForm
    UserForm1.Show
End Sub
```

Skoroszyt z tym przykładem (*Elementy unikatowe1.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Znajdziesz tam również skoroszyt o nazwie *Elementy unikatowe2.xlsxm*, zawierający bardziej rozbudowaną wersję tej procedury (między innymi wyposażoną w sortowanie odszukanych unikatowych elementów listy).



@

W sieci

Identyfikowanie zaznaczonego elementu listy formantu ListBox

Po wykonaniu procedur z poprzednich punktów wyświetlany jest formularz *UserForm* zawierający formant *ListBox* wypełniony różnymi elementami. W omawianych dotąd procedurach pominięto jednak kluczową kwestię: jak sprawdzić, czy użytkownik wybrał jakiś element, i jeżeli tak, to który.



W naszym przykładzie zakładamy, że obiekt *ListBox* pozwala na wybranie i zaznaczenie tylko pojedynczych elementów (czyli że właściwość *MultiSelect* formantu *ListBox* ma wartość 0).

Aby sprawdzić, który element listy został zaznaczony, powinieneś użyć właściwości *Value* formantu *ListBox*. Przykładowo poniższa instrukcja wyświetla wartość zaznaczonego elementu listy formantu *ListBox*:

```
MsgBox ListBox1.Value
```

Jeżeli użytkownik nie zaznaczył żadnego elementu listy, próba wykonania tego polecenia spowoduje wystąpienie błędu.

Aby zamiast wartości zaznaczonego elementu określić jego położenie na liście, możesz użyć właściwości *ListBox1.ListIndex*. Poniższe polecenie wyświetla okno komunikatu zawierające numer zaznaczonego elementu listy formantu *ListBox*:

```
MsgBox "Zaznaczono element numer " & ListBox1.ListIndex
```

Jeżeli nie zaznaczono żadnego elementu, wartością właściwości *ListIndex* będzie -1.



Numerowanie elementów listy formantu *ListBox* rozpoczyna się od 0, a nie od 1. Z tego względu właściwość *ListIndex* dla pierwszego elementu listy formantu *ListBox* ma wartość 0, a dla ostatniego elementu tej listy jest równa wartości właściwości *ListCount* pomniejszonej o 1.

Identyfikowanie wielu zaznaczonych elementów listy formantu ListBox

Właściwość *MultiSelect* formantu *ListBox* może przyjmować jedną z trzech wartości:

- 0 (*fmMultiSelectSingle*). Można wybrać i zaznaczyć tylko pojedynczy element listy. Jest to ustawienie domyślne.
- 1 (*fmMultiSelectMulti*). Naciśnięcie klawisza spacji lub klikanie wybranych elementów powoduje ich zaznaczanie lub usuwanie zaznaczenia.
- 2 (*fmMultiSelectExtended*). Kliknięcie wybranego elementu wraz z przytrzymaniem wciśniętego klawisza *Shift* powoduje zaznaczenie wszystkich elementów listy od poprzednio zaznaczonego elementu do elementu klikniętego. Do zaznaczania elementów takiej listy możesz również użyć klawisza *Shift* i klawiszy kurSORA.

Jeżeli formant *ListBox* pozwala na zaznaczanie wielu elementów listy (czyli kiedy właściwość *MultiSelect* ma wartość 1 lub 2), próba użycia właściwości *ListIndex* lub *Value* spowoduje wystąpienie błędu. Zamiast tego powinieneś użyć właściwości *Selected* zwracającej tablicę wartości logicznych odpowiadających zaznaczonym pozycjom w liście,

przy czym pierwszemu elementowi listy odpowiada indeks o wartości 0. Przykładowo poniższe polecenie wyświetli wartość True, jeżeli na liście formantu ListBox zostanie zaznaczony pierwszy element:

```
MsgBox ListBox1.Selected(0)
```



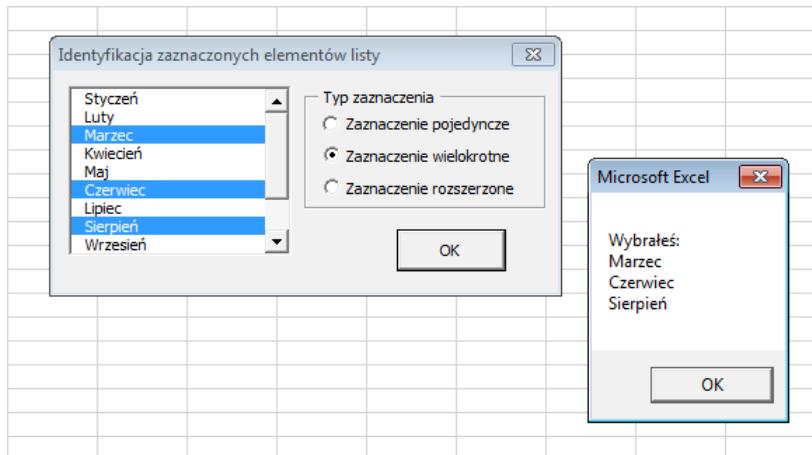
Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz przykładowy skoroszyt o nazwie *Zaznaczanie elementów listy.xlsxm*, demonstrujący, w jaki sposób można identyfikować zaznaczone elementy listy formantów ListBox, obsługujących zarówno pojedyncze, jak i wielokrotne zaznaczenia.

Poniższa procedura, zamieszczona w przykładowym skoroszycie znajdującym się na dysku CD-ROM, przy użyciu pętli przetwarzającej każdy element listy formantu ListBox. Po zaznaczeniu danego elementu procedura dołącza jego wartość do zmiennej Msg. Na końcu nazwy wszystkich zaznaczonych elementów są wyświetlane w oknie komunikatu.

```
Private Sub OKButton_Click()
    Msg = ""
    For i = 0 To ListBox1.ListCount - 1
        If ListBox1.Selected(i) Then
            Msg = Msg & ListBox1.List(i) & vbCrLf
        Next i
    MsgBox "Wybrałeś: " & vbCrLf & Msg
    Unload Me
End Sub
```

Na rysunku 12.9 pokazano wynik działania procedury w sytuacji, gdy zaznaczonych zostało wiele elementów listy formantu ListBox.

Rysunek 12.9.
Okno komunikatu wyświetla listę zaznaczonych elementów formantu ListBox



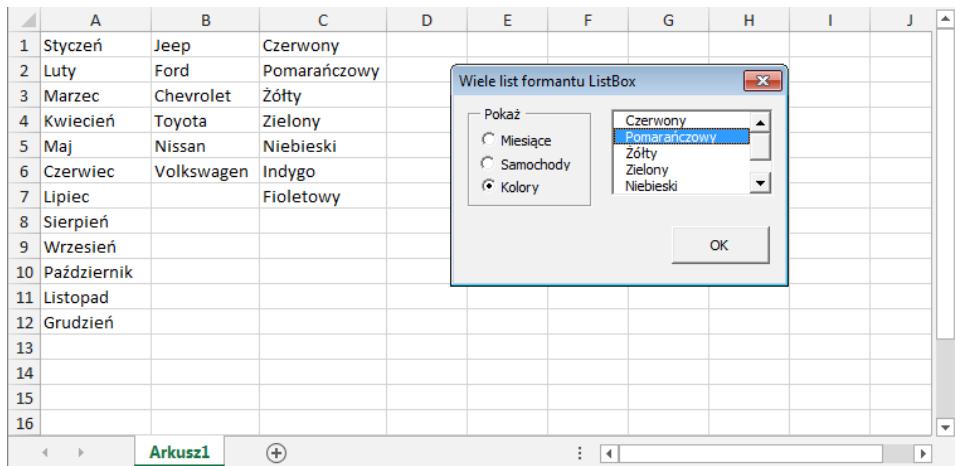
Wiele list w jednym formancie ListBox

Kolejny przykład pokazuje, jak utworzyć formant ListBox, którego zawartość będzie się zmieniała w zależności od opcji wybranej przez użytkownika spośród grupy obiektów OptionButton.

Elementy listy formantu **ListBox** są pobierane z zakresu komórek arkusza. Procedury obsługi zdarzenia **Click** formantów **OptionButton** po prostu ustawiają różne zakresy właściwości **RowSource** formantu **ListBox**. Oto kod jednej z tych procedur:

```
Private Sub obMonths_Click()
    ListBox1.RowSource = "Arkusz1!Miesiące"
End Sub
```

Wygląd formularza *UserForm* został przedstawiony na rysunku 12.10.



Rysunek 12.10. Zawartość listy formantu **ListBox** zależy od wybranej opcji (formanty **OptionButton**)

Kliknięcie formantu **OptionButton** o nazwie **obMonths** zmienia wartość właściwości **RowSource** formantu **ListBox** na zakres **Miesiące** arkusza **Arkusz1**.



Skoroszyt z tym przykładem (*Wiele list formantu ListBox.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

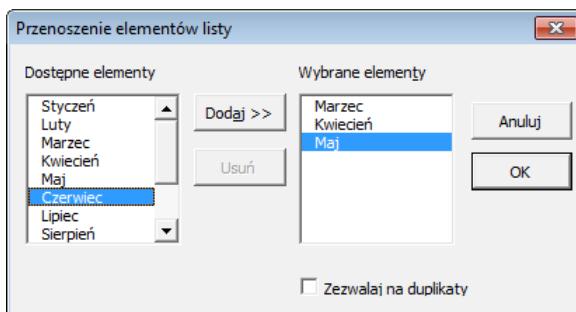
Przenoszenie elementów listy formantu **ListBox**

Niektóre aplikacje wymagają od użytkownika wybrania z listy kilku pozycji. Jest to często stosowane do tworzenia nowych list, składających się z zaznaczonych elementów, i wyświetlania takiej nowo utworzonej listy w kolejnym formancie **ListBox**. Dobrym przykładem takiego rozwiązania jest karta *Pasek narzędzi Szybki dostęp* okna dialogowego *Opcje programu Excel*.

Na rysunku 12.11 pokazano okno dialogowe z dwoma formantami **ListBox**. Przycisk *Dodaj* umieszcza na liście prawego formantu **ListBox** elementy zaznaczone na liście formantu **ListBox** znajdującego się po lewej. Przycisk *Usuń* usuwa element zaznaczony na liście znajdującej się po prawej stronie. Opcja *Zezwalaj na duplikaty* określa zachowanie formularza, gdy do listy zostanie dodany duplikat elementu. Jeżeli opcja *Zezwalaj na duplikaty* nie jest zaznaczona, przy próbie dodania do listy elementu, który się już na niej znajduje, zostanie wyświetlone okno komunikatu.

Rysunek 12.11.

*Tworzenie listy
w oparciu o inną listę*



Kod programu w tym przykładzie jest dość prosty. Oto procedura wykonywana po kliknięciu przez użytkownika przycisku *Dodaj*:

```
Private Sub AddButton_Click()
    If ListBox1.ListIndex = -1 Then Exit Sub
    If Not cbDuplicates Then
        ' Sprawdza, czy element już istnieje
        For i = 0 To ListBox2.ListCount - 1
            If ListBox1.Value = ListBox2.List(i) Then Exit Sub
        Next i
    End If
    ListBox2.AddItem ListBox1.Value
End Sub
```

Kod źródłowy procedury dla przycisku *Usuń* jest jeszcze prostszy:

```
Private Sub RemoveButton_Click()
    If ListBox2.ListIndex <> -1 Then
        ListBox2.RemoveItem ListBox2.ListIndex
    End If
    If ListBox2.ListIndex = -1 Then RemoveButton.Enabled = False
End Sub
```

Zauważ, że obie procedury sprawdzają, czy dany element faktycznie został zaznaczony. Jeżeli wartością właściwości *ListIndex* formantu *ListBox* jest *-1*, oznacza to, że nie zaznaczono żadnego elementu i procedura kończy działanie.

W tym przykładzie znajdziesz jeszcze dwie dodatkowe procedury, które sterują tym, czy przycisk *Usuń* jest aktywny lub zablokowany. Zdarzenia są generowane, kiedy użytkownik aktywuje formant *ListBox* (przy użyciu klawiatury lub myszy). W efekcie działania tych procedur przycisk *Usuń* jest aktywny tylko wtedy, kiedy użytkownik operuje na liście znajdującej się z prawej strony (formant *ListBox2*).

```
Private Sub ListBox1_Enter()
    RemoveButton.Enabled = False
End Sub

Private Sub ListBox2_Enter()
    RemoveButton.Enabled = True
End Sub
```

Skoroszyt z tym przykładem (*Przenoszenie elementów listy formantu ListBox.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

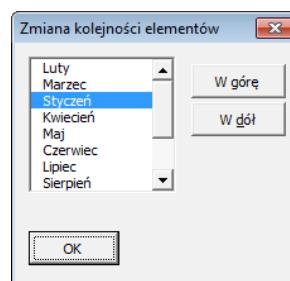
Zmiana kolejności elementów listy formantu **ListBox**

Bardzo często zdarzają się sytuacje, że kolejność elementów na liście jest bardzo istotna. Przykład omawiany w tym podrozdziale pokazuje, jak umożliwić użytkownikowi przenoszenie elementów w górę i w dół listy (zmianę kolejności elementów na liście). Taką technikę wykorzystuje edytor VBE do zmiany kolejności tabulacji formantów formularza *UserForm* (aby się o tym przekonać, kliknij formularz *UserForm* prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Tab Order*).

Na rysunku 12.12 pokazano okno dialogowe zawierające formant *ListBox* i dwa formanty *CommandButton*. Kliknięcie przycisku *Przesuń w górę* spowoduje przemieszczenie elementu w górę listy formantu *ListBox*, natomiast kliknięcie przycisku *Przesuń w dół* — przemieszczenie w dół listy.

Rysunek 12.12.

Przyciski umożliwiają użytkownikowi przemieszczanie elementów w górę i w dół listy



Skoroszyt z tym przykładem (*Zmiana kolejności elementów listy.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Procedury obsługi zdarzeń formantów *CommandButton* wyglądają następująco:

```

Private Sub MoveUpButton_Click()
    Dim NumItems As Integer, i As Integer, ItemNum As Integer
    Dim TempItem As String, TempList()
    If ListBox1.ListIndex <= 0 Then Exit Sub
    NumItems = ListBox1.ListCount
    Dim TempList()
    ReDim TempList(0 To NumItems - 1)
    ' Wypełnienie tablicy elementami listy formantu ListBox
    For i = 0 To NumItems - 1
        TempList(i) = ListBox1.List(i)
    Next i
    ' Wybrany element
    ItemNum = ListBox1.ListIndex
    ' Zmiana położenia elementu
    TempItem = TempList(ItemNum)
    TempList(ItemNum) = TempList(ItemNum - 1)
    TempList(ItemNum - 1) = TempItem
    ListBox1.List = TempList
    ' Zmiana indeksu listy elementów
    ListBox1.ListIndex = ItemNum - 1
End Sub

Private Sub MoveDownButton_Click()
    Dim NumItems As Integer, i As Integer, ItemNum As Integer

```

```

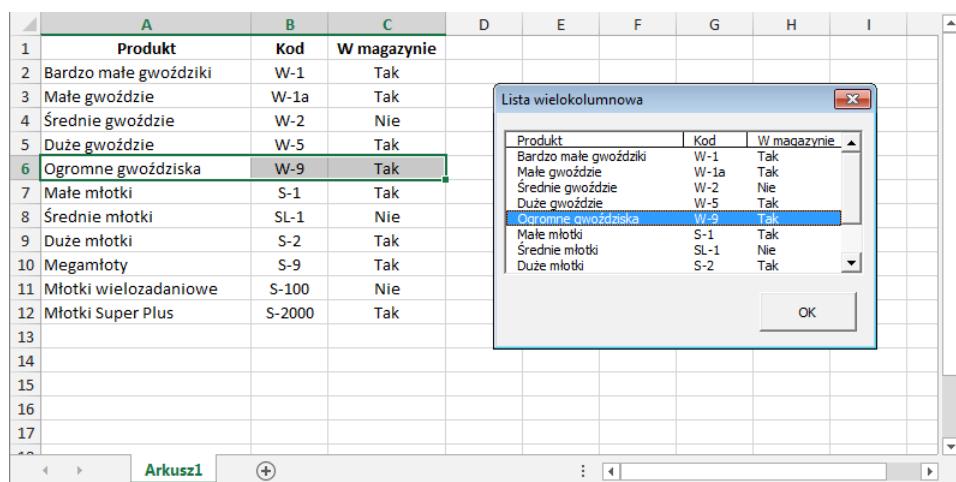
Dim TempItem As String, tempList()
If ListBox1.ListIndex = ListBox1.ListCount - 1 Then Exit Sub
NumItems = ListBox1.ListCount
Dim tempList()
ReDim tempList(0 To NumItems - 1)
' Wypełnienie tablicy elementami listy formantu ListBox
For i = 0 To NumItems - 1
    tempList(i) = ListBox1.List(i)
Next i
' Wybrany element
ItemNum = ListBox1.ListIndex
' Zmiana położenia elementu
TempItem = tempList(ItemNum)
tempList(ItemNum) = tempList(ItemNum + 1)
tempList(ItemNum + 1) = TempItem
ListBox1.List = tempList
' Zmiana indeksu listy elementów
ListBox1.ListIndex = ItemNum + 1
End Sub

```

Pracując z tym przykładem, zauważylem, że z jakiegoś powodu szybkie kolejne kliknięcia przycisków *W górę* lub *W dół* nie zawsze są rejestrowane jako wielokrotne naciśnięcia danego przycisku. Aby uporać się z tym problemem, dodałem do kodu dwie dodatkowe procedury, które obsługują zdarzenia związane z podwójnym kliknięciem każdego z przycisków. Każda z tych procedur po prostu wywołuje odpowiednią procedurę obsługi zdarzenia *Click* z omawianych powyżej.

Wielokolumnowe formanty ListBox

Zwykły formant *ListBox* korzysta z listy jednokolumnowej. Można jednak stworzyć formant *ListBox* wyświetlający listę wielu kolumn i (opcjonalnie) ich nagłówki. Na rysunku 12.13 pokazano przykład wielokolumnowego formantu *ListBox* pobierającego dane z zakresu arkusza.



Rysunek 12.13. Formant *ListBox* wyświetlający trójkolumnową listę wraz z etykietami kolumn



Skoroszyt z tym przykładem (*Wielokolumnowe formanty ListBox1.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Aby stworzyć wielokolumnowy formant *ListBox* korzystający z danych przechowywanych w zakresie komórek arkusza, wykonaj następujące polecenia:

1. Upewnij się, czy dla właściwości *ColumnCount* formantu *ListBox* ustawiono poprawną liczbę kolumn.
2. Zdefiniuj w arkuszu Excela wielokolumnowy zakres, który stanie się wartością właściwości *RowSource* formantu *ListBox*.
3. Aby wyświetlić nagłówki kolumn, dla właściwości *ColumnHeads* ustaw wartość *True*.

Przy ustawianiu zakresu dla właściwości *RowSource* nie uwzględniaj nagłówków kolumn. VBA automatycznie używa wiersza znajdującego się bezpośrednio nad pierwszym wierszem zakresu będącego wartością właściwości *RowSource*.

4. Zdefiniuj szerokość poszczególnych kolumn, wpisując szereg wartości wyrażanych w punktach (1 punkt = $\frac{1}{72}$ cala) i oddzielonych od siebie średnikami jako wartość właściwości *ColumnWidths*.

Przykładowo dla trójkolumnowego formantu *ListBox* wartością właściwości *ColumnWidths* może być następujący łańcuch tekstu:

100 pt;40 pt;30 pt

5. Wybierz kolumnę, którą wpiszesz jako wartość właściwości *BoundColumn*.

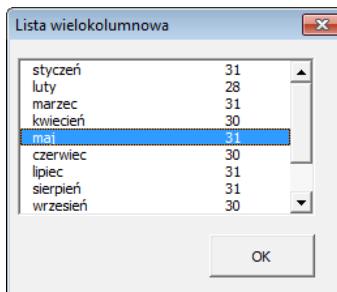
Właściwość *BoundColumn* identyfikuje kolumnę, która jest używana gdy program sprawdza wartość właściwości *Value* formantu *ListBox*.

Aby wypełnić danymi wielokolumnowy formant *ListBox* bez korzystania z zakresu komórek, powinieneś najpierw utworzyć tablicę dwuwymiarową, a następnie przypisać ją właściwości *List* formantu *ListBox*. Poniższa procedura demonstruje zastosowanie tablicy o nazwie *Data* złożonej z 12 wierszy i 2 kolumn. W pierwszej kolumnie dwukolumnowego formantu *ListBox* są wyświetlane nazwy miesięcy, natomiast w drugiej — liczby dni miesięcy (patrz rysunek 12.14). Zwróć uwagę, że właściwość *ColumnCount* jest ustawiana na wartość 2.

```
Private Sub UserForm_Initialize()
    ' Wypełnienie listy formantu ListBox
    Dim Data(1 To 12, 1 To 2)
    For i = 1 To 12
        Data(i, 1) = Format(DateSerial(2010, i, 1), "mmmm")
    Next i
    For i = 1 To 12
        Data(i, 2) = Day(DateSerial(2010, i + 1, 1)) - 1
    Next i
    ListBox1.ColumnCount = 2
    ListBox1.List = Data
End Sub
```

Rysunek 12.14.

Dwukolumnowy formant *ListBox* wypełniony danymi przechowywanymi w tablicy



Skoroszyt z tym przykładem (*Wielokolumnowe formanty ListBox2.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Jeżeli źródłem danych listy jest tablica języka VBA, zdefiniowanie nagłówków kolumn dla właściwości *ColumnHeads* nie jest możliwe.

Zastosowanie formantu *ListBox* do wybierania wierszy arkusza

Procedura zamieszczona w tym podrozdziale wyświetla formant *ListBox* zawierający cały używany zakres aktywnego arkusza (patrz rysunek 12.15). Użytkownik może zaznaczyć wiele elementów listy. Klikając przycisk *Wszystkie*, zaznaczy wszystkie elementy; klikając przycisk *Żaden*, spowoduje usunięcie wszystkich zaznaczeń. Kliknięcie *OK* spowoduje zaznaczenie w arkuszu wierszy odpowiadających zaznaczonym elementom listy. W razie potrzeby można oczywiście zaznaczać wiersze bezpośrednio w arkuszu, przytrzymując wciśnięty klawisz *Ctrl*, aczkolwiek w pewnych sytuacjach zaznaczanie wierszy może być łatwiejsze przy użyciu formantu *ListBox*.

The screenshot shows a spreadsheet with data in columns A through J. A 'Wybieranie wierszy' (Selecting rows) dialog box is open over the data. The dialog has a title bar 'Row 15' and a list of rows from 6 to 24. Each row has a checkbox next to it. Some checkboxes are checked (e.g., row 16, row 20), while others are empty. At the bottom of the dialog are buttons for 'Wszystkie' (All), 'Usuń zaznaczenie' (Delete selection), 'Anuluj' (Cancel), and 'OK'. The main spreadsheet area shows data for various properties like Agent, Data, Obszar, Cena, Liczba sypialni, Łazienki, Powierzchnia m², Typ, Basen, and Sprzedane.

Rysunek 12.15. Formant *ListBox* ułatwia zaznaczanie wierszy arkusza



Skoroszyt z tym przykładem (*Wybieranie wierszy.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pwv.htm>).

Zaznaczanie wielu elementów jest możliwe, ponieważ wartością właściwości MultiSelect formantu ListBox jest 1 (fmMultiSelectMulti). Pole wyboru obok każdej pozycji jest wyświetlane, ponieważ wartością właściwości ListStyle kontrolki ListBox jest 1 (fmListStyleOption).

Poniżej zamieszczono procedurę obsługi zdarzenia Initialize powiązanego z formularzem *UserForm*. Procedura tworzy obiekt Range o nazwie rng reprezentujący używany zakres aktywnego arkusza. Dodatkowy kod źródłowy ustawia wartości dla właściwości formantu ListBox o nazwach ColumnCount i RowSource, a także modyfikuje wartość właściwości ColumnWidths w taki sposób, że szerokość kolumn formantu ListBox jest proporcjonalna do szerokości kolumn arkusza.

```
Private Sub UserForm_Initialize()
    Dim ColCnt As Integer
    Dim rng As Range
    Dim cw As String
    Dim c As Integer
    ColCnt = ActiveSheet.UsedRange.Columns.Count
    Set rng = ActiveSheet.UsedRange
    With ListBox1
        .ColumnCount = ColCnt
        .RowSource = rng.Address
        cw = ""
        For c = 1 To .ColumnCount
            cw = cw & rng.Columns(c).Width & ";"
        Next c
        .ColumnWidths = cw
        .ListIndex = 0
    End With
End Sub
```

Z przyciskami *Wszystkie* i *Żadna*, reprezentowanymi przez formanty odpowiednio SelectAllButton i SelectNoneButton, są skojarzone proste procedury obsługi zdarzeń. Oto one:

```
Private Sub SelectAllButton_Click()
    Dim r As Integer
    For r = 0 To ListBox1.ListCount - 1
        ListBox1.Selected(r) = True
    Next r
End Sub

Private Sub SelectNoneButton_Click()
    Dim r As Integer
    For r = 0 To ListBox1.ListCount - 1
        ListBox1.Selected(r) = False
    Next r
End Sub
```

Poniżej zamieszczono kod źródłowy procedury `OKButton_Click`. Procedura tworzy obiekt Range o nazwie RowRange, złożony z wierszy odpowiadających zaznaczonym elementom formantu `ListBox`. Aby stwierdzić, czy zaznaczono wiersz, procedura sprawdza wartość właściwości `Selected` formantu `ListBox`. W celu dodania do obiektu RowRange dodatkowych zakresów procedura korzysta z funkcji `Union`.

```
Private Sub OKButton_Click()
    Dim RowRange As Range
    Dim RowCnt As Integer, r As Integer
    RowCnt = 0
    For r = 0 To ListBox1.ListCount - 1
        If ListBox1.Selected(r) Then
            RowCnt = RowCnt + 1
            If RowCnt = 1 Then
                Set RowRange = ActiveSheet.UsedRange.Rows(r + 1)
            Else
                Set RowRange =
                    Union(RowRange, ActiveSheet.UsedRange.Rows(r + 1))
            End If
        End If
    Next r
    If Not RowRange Is Nothing Then RowRange.Select
    Unload Me
End Sub
```



Skoroszyt z tym przykładem (*Wybieranie wierszy.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

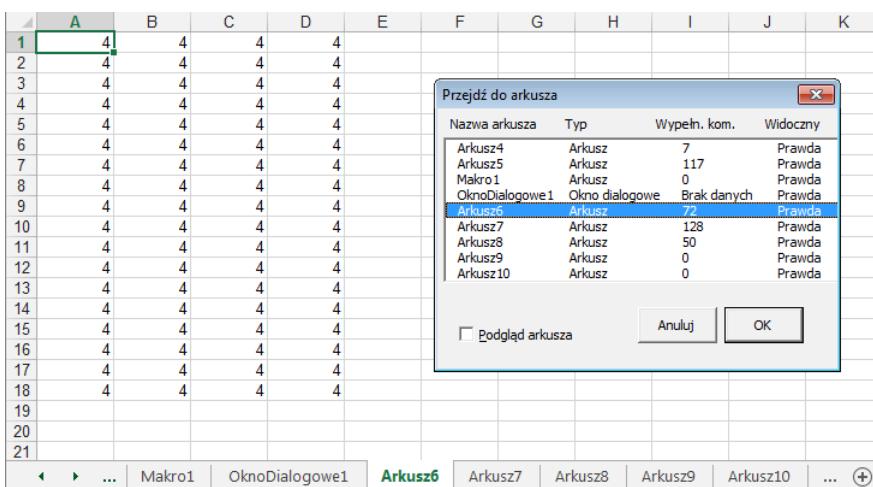
Uaktynianie arkusza za pomocą formantu `ListBox`

Przykład zaprezentowany w tym podrozdziale jest zarówno bardzo przydatny jak i pouczający. Zastosowano tu wielokolumnowy formant `ListBox` wyświetlający listę arkuszy aktywnego skoroszytu. W poszczególnych kolumnach wyświetlane są takie informacje jak:

- Nazwa arkusza.
- Typ arkusza (zwykły, wykresu lub okna dialogowego Excela w wersjach 5 i 95).
- Liczba niepustych komórek arkusza.
- Widoczność arkusza.

Na rysunku 12.16 pokazano przykładowe okno dialogowe.

Poniżej zamieszczono kod źródłowy procedury `UserForm_Initialize` tworzącej dwuwykwiarową tablicę i zbierającą informacje poprzez przetworzenie przy użyciu pętli każdego arkusza aktywnego skoroszytu. Następnie procedura przenosi zawartość tablicy do listy formantu `ListBox`.



Rysunek 12.16. Okno dialogowe umożliwia użytkownikowi uaktywnienie wybranego arkusza

```

Public OriginalSheet As Object

Private Sub UserForm_Initialize()
    Dim SheetData() As String, Sht As Object
    Dim ShtCnt As Integer, ShtNum As Integer, ListPos As Integer

    Set OriginalSheet = ActiveSheet
    ShtCnt = ActiveWorkbook.Sheets.Count
    ReDim SheetData(1 To ShtCnt, 1 To 4)
    ShtNum = 1
    For Each Sht In ActiveWorkbook.Sheets
        If Sht.Name = ActiveSheet.Name Then _
            ListPos = ShtNum - 1
        SheetData(ShtNum, 1) = Sht.Name
        Select Case TypeName(Sht)
            Case "Worksheet"
                SheetData(ShtNum, 2) = "Arkusz"
                SheetData(ShtNum, 3) = Application.CountA(Sht.Cells)
            Case "Chart"
                SheetData(ShtNum, 2) = "Wykres"
                SheetData(ShtNum, 3) = "Nie dotyczy"
            Case "DialogSheet"
                SheetData(ShtNum, 2) = "Okno dialogowe"
                SheetData(ShtNum, 3) = " Nie dotyczy "
        End Select
        If Sht.Visible Then
            SheetData(ShtNum, 4) = "Prawda"
        Else
            SheetData(ShtNum, 4) = "Fałsz"
        End If
        ShtNum = ShtNum + 1
    Next Sht
    With ListBox1
        .ColumnWidths = "100 pt;30 pt;40 pt;50 pt"
    End With
End Sub

```

```

    .List = SheetData
    .ListIndex = ListPos
End With
End Sub

```

Procedura `ListBox1_Click` wygląda następująco:

```

Private Sub ListBox1_Click()
    If cbPreview Then Sheets(ListBox1.Value).Activate
End Sub

```

Wartość właściwości `Value` formantu `CheckBox` o nazwie `cbPreview` decyduje, czy dla wybranego arkusza zostanie wyświetlony podgląd, jeżeli użytkownik zaznaczy w formancie `ListBox` powiązany z nim element.

Kliknięcie *OK* (formant `OKButton`) spowoduje wykonanie procedury `OKButton_Click`, której kod źródłowy wygląda następująco:

```

Private Sub OKButton_Click()
    Dim UserSheet As Object
    Set UserSheet = Sheets(ListBox1.Value)
    If UserSheet.Visible Then
        UserSheet.Activate
    Else
        If MsgBox("Czy pokazać arkusz?", _
            vbQuestion + vbYesNoCancel) = vbYes Then
            UserSheet.Visible = True
            UserSheet.Activate
        Else
            OriginalSheet.Activate
        End If
    End If
    Unload Me
End Sub

```

Procedura `OKButton_Click` tworzy zmienną obiektową reprezentującą wybrany arkusz. Jeżeli arkusz jest widoczny, zostanie uaktywniony. Jeżeli nie jest widoczny, wyświetli się okno komunikatu i użytkownik zostanie zapytany, czy arkusz powinien być pokazany i uaktywniony. W przeciwnym razie uaktywniony zostanie oryginalny arkusz (przecho-wywany przez publiczną zmienną obiektową `OriginalSheet`).

Dwukrotne kliknięcie danego elementu listy formantu `ListBox` daje takie same efekty, jak kliknięcie przycisku *OK*. Poniższa procedura `ListBox1_DblClick` wywołuje jedynie procedurę `OKButton_Click`:

```

Private Sub ListBox1_DblClick(ByVal Cancel As MSForms.ReturnBoolean)
    Call OKButton_Click
End Sub

```

Skoroszyt z tym przykładem (*Aktywacja arkusza.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

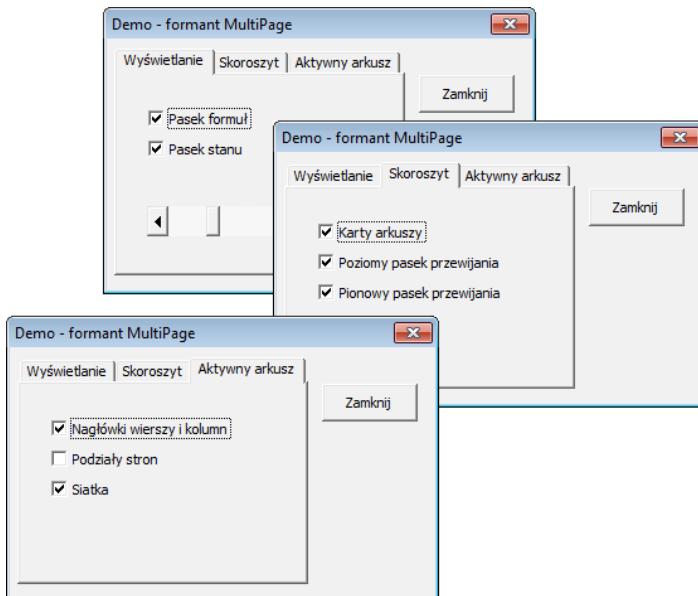


Zastosowanie formantu MultiPage na formularzach UserForm

Formant MultiPage jest bardzo użyteczny w przypadku formularzy *UserForm* zawierających wiele opcji, ponieważ umożliwia grupowanie wybranych obiektów i umieszczenie ich na oddzielnych kartach.

Na rysunku 12.17 pokazano przykładowy formularz *UserForm* z formantem MultiPage. W tym przypadku formant posiada trzy karty (strony).

Rysunek 12.17.
Formant MultiPage grupuje inne formanty na kolejnych kartach



Skoroszyt z tym przykładem (*Formant MultiPage.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Okno *Toolbox* zawiera również formant TabStrip przypominający nieco formant MultiPage. Jednak w przeciwieństwie do formantu MultiPage, formant TabStrip nie pełni funkcji kontenera dla innych obiektów. Formant MultiPage jest o wiele bardziej uniwersalny i szczerze mówiąc, z tego powodu nigdy nie korzystałem z formantu TabStrip.

Korzystając z formantu MultiPage, należy zachować ostrożność. Poniżej zamieszczamy listę kilka zagadnień, o których trzeba pamiętać.

- Karta (strona), która zostanie domyślnie wyświetlona, zależy od wartości właściwości Value formantu MultiPage. Wartość 0 spowoduje uaktywnienie pierwszej karty, 1 — drugiej itd.

- Domyślnie format MultiPage posiada dwie karty. Aby w edytorze Visual Basic dodać do formantu nową kartę, należy kliknąć ją prawym przyciskiem myszy i z menu podręcznego wybrać pozycję *New Page*.
- Aby w trakcie modyfikowania formantu MultiPage określić właściwości wybranej strony, wystarczy ją kliknąć. W oknie *Properties* zostaną wyświetlane właściwości, które można edytować.
- Zaznaczenie formantu MultiPage może okazać się trudne, ponieważ jego kliknięcie spowoduje wybranie jednej z kart. Aby zaznaczyć cały formant MultiPage, należy kliknąć jego obramowanie lub wcisnąć klawisz *Tab* do momentu uaktywnienia formantu. Kolejna metoda polega na zaznaczeniu formantu MultiPage poprzez wybranie go z listy rozwijanej okna *Properties*.
- Jeżeli utworzony formant MultiPage zawiera wiele kart, to w celu wyświetlania ich w więcej niż jednym rzędzie należy dla właściwości MultiRow ustawić wartość True.
- Zamiast kart w formancie MultiPage można też wyświetlić przyciski. W tym celu wystarczy dla właściwości Style ustawić wartość 1 — fmTabStyleButtons. Jeżeli wartością właściwości Style jest 2 — fmTabStyleNone, formant MultiPage nie wyświetli ani kart, ani przycisków.
- Właściwość TabOrientation określa położenie kart formantu MultiPage.

Korzystanie z formantów zewnętrznych

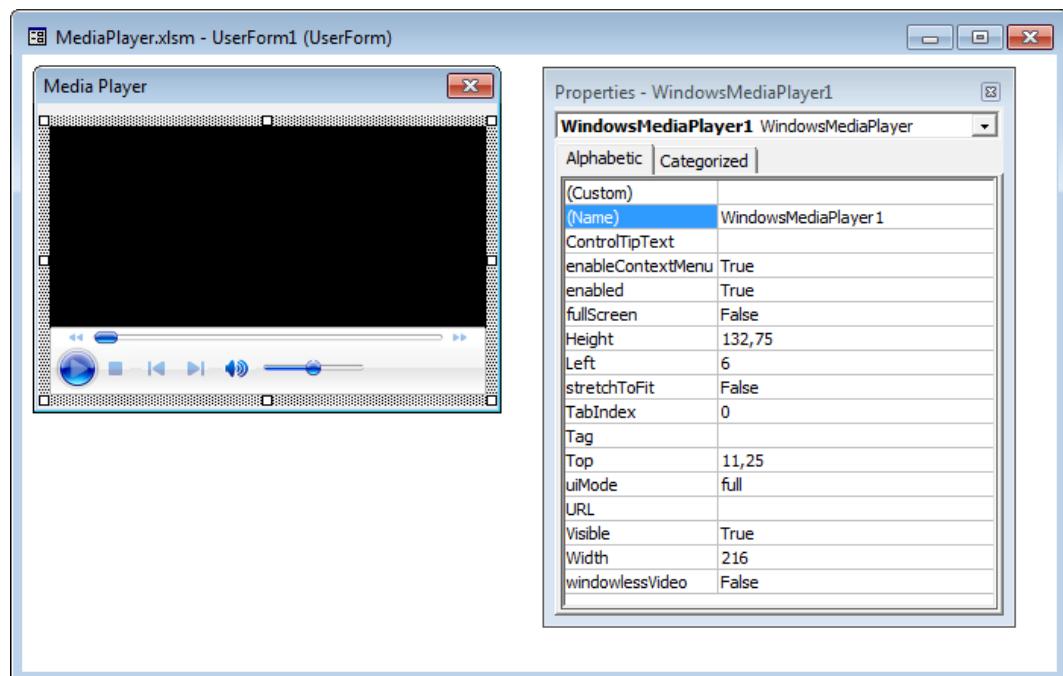
W przykładzie omawianym w tym podrozdziale użyjemy formantu ActiveX *Windows Media Player*. Pomimo iż nie jest to formant programu Excel (jest instalowany razem z systemem Windows), można go bez problemu używać na formularzach *UserForm*.

Aby skorzystać z tego formantu, dodaj do skoroszytu nowy formularz *UserForm* i wykonaj następujące polecenia:

1. Uruchom edytor VBE.
 2. Kliknij prawym przyciskiem myszy okno *Toolbox* i z menu podręcznego wybierz polecenie *Additional controls*.
- Jeżeli okno *Toolbox* nie jest widoczne na ekranie, wybierz z menu głównego edytora VBE polecenie *View/Toolbox*.
3. Na ekranie pojawi się okno dialogowe *Additional controls*. Odszukaj na liście i zaznacz opcję *Windows Media Player*.
 4. Naciśnij przycisk *OK*.

W oknie *Toolbox* pojawi się ikona nowego formantu.

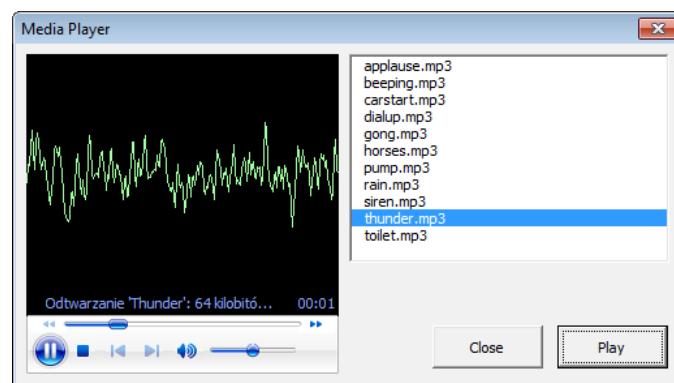
Na rysunku 12.18 przedstawiono wygląd formantu *Windows Media Player* osadzonego na formularzu *UserForm* oraz jego okna *Properties*. Właściwość URL reprezentuje aktualnie odtwarzany rodzaj klipu (muzyka lub wideo). Jeżeli klip znajduje się na dysku twardym komputera, wartością właściwości URL będzie pełna ścieżka wraz z nazwą odtwarzanego klipu.



Rysunek 12.18. Formant Windows Media Player osadzony na formularzu UserForm

Na rysunku 12.19 przedstawiono sposób korzystania z tego formantu. Obraz wideo pokazuje wizualizację odtwarzanego w danej chwili dźwięku. Ponadto na formularzu *UserForm* umieściliśmy dodatkowy formant *ListBox* wypełniony nazwami plików MP3. Naciśnięcie przycisku *Odtwórz* powoduje rozpoczęcie odtwarzania zaznaczonego pliku. Naciśnięcie przycisku *Zamknij* powoduje zatrzymanie odtwarzania i zamknięcie formularza *UserForm*. Okno dialogowe formularza jest wyświetlane w trybie niemodalnym (ang. *modeless*), dzięki czemu użytkownik może korzystać z arkusza bez konieczności zamykania tego okna.

Rysunek 12.19.
Przykład użycia
formantu Windows
Media Player





Skoroszyt z tym przykładem (*Media Player.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>). Skoroszyt znajduje się w osobnym folderze o nazwie *medioplayer*, w którym znajdziesz również kilkanaście przykładowych plików MP3, objętych licencją public domain.

Utworzenie tego przykładu było bardzo prostym zadaniem. Procedura *UserForm_Initialize* dodaje nazwy plików MP3 do listy formantu *ListBox*. Aby uprościć całe zadanie, procedura odczytuje nazwy plików znajdujących się w tym samym folderze, co skoroszyt. Oczywiście bardziej użytecznym rozwiążaniem byłoby zapewnienie użytkownikowi możliwości wybrania foldera z plikami MP3, które mają być odtwarzane.

```
Private Sub UserForm_Initialize()
    Dim FileName As String
    ' Wypełnia listę formantu ListBox nazwami plików MP3
    FileName = Dir(ThisWorkbook.Path & "\*.mp3", vbNormal)
    Do While Len(FileName) > 0
        ListBox1.AddItem FileName
        FileName = Dir()
    Loop
    ListBox1.ListIndex = 0
End Sub
```



Więcej szczegółowych informacji na temat polecenia *Dir* znajdziesz w rozdziale 25.

Procedura obsługi zdarzenia *PlayButton_Click* składa się z pojedynczego polecenia, które przypisuje nazwę wybranego pliku do właściwości *URL* obiektu *WindowsMediaPlayer1*.

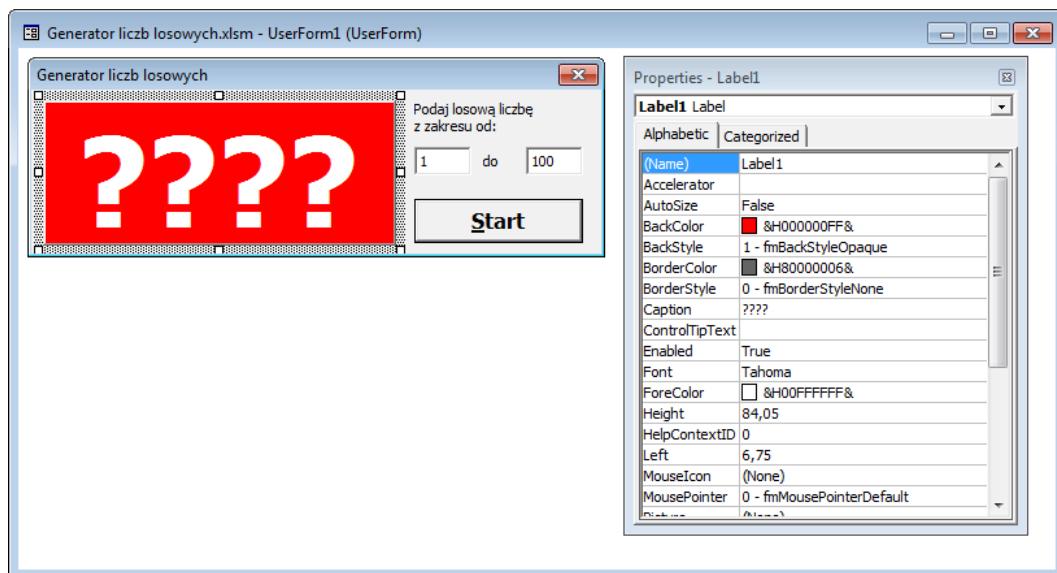
```
Private Sub PlayButton_Click()
    ' Właściwość URL odpowiada za załadowanie i rozpoczęcie odtwarzania pliku
    WindowsMediaPlayer1.URL =
        ThisWorkbook.Path & "\" & ListBox1.List(ListBox1.ListIndex)
End Sub
```

Z pewnością znajdziesz wiele możliwości rozbudowania tej prostej aplikacji. Zwróć uwagę na fakt, że ten formant potrafi odpowiadać na wiele różnych zdarzeń.

Animowanie etykiet

Ostatni przykład w tym rozdziale demonstruje sposób animowania formantów *Label*. Formularz *UserForm* przedstawiony na rysunku 12.20 spełnia rolę interaktywnego generatora liczb losowych.

Dwa formanty *TextBox* przechowują dolną i górną wartość zakresu generowanych liczb losowych. Formant *Label* początkowo wyświetla cztery znaki zapytania, ale po naciśnięciu przez użytkownika przycisku *Start* tekst jest animowany i wyświetla generowane liczby losowe. Przycisk *Start* zamienia się w przycisk *Stop*, a naciśnięcie tego przycisku zatrzymuje generator i wyświetla otrzymaną liczbę losową. Na rysunku 12.21 przedstawiono okno dialogowe wyświetlające losową liczbę z zakresu od -1000 do 1000.



Rysunek 12.20. Generowanie liczby losowej

Rysunek 12.21.
Liczba losowa została
wygenerowana



Kod procedury przypisanej do przycisku wygląda następująco:

```

Dim Stopped As Boolean

Private Sub StartStopButton_Click()
    Dim Low As Double, Hi As Double

    If StartStopButton.Caption = "Start" Then
        ' Sprawdź poprawność dolnej i górnej granicy zakresu generowanych liczb
        If Not IsNumeric(TextBox1.Text) Then
            MsgBox "Wymagana numeryczna wartość początkowa.", vbInformation
            With TextBox1
                .SelStart = 0
                .SelLength = Len(.Text)
                .SetFocus
            End With
            Exit Sub
        End If

        If Not IsNumeric(TextBox2.Text) Then
            MsgBox "Wymagana numeryczna wartość końcowa.", vbInformation
            With TextBox2
                .SelStart = 0
                .SelLength = Len(.Text)
            End If
        End If
    End If
End Sub

```

```
    .SetFocus
End With
Exit Sub
End If

' Upewnij się, że granice zakresu zostały podane w poprawnej kolejności
Low = Application.Min(Val(TextBox1.Text), Val(TextBox2.Text))
Hi = Application.Max(Val(TextBox1.Text), Val(TextBox2.Text))

' Dopasuj rozmiar czcionki (jeżeli to konieczne)
Select Case Application.Max(Len(TextBox1.Text), Len(TextBox2.Text))
    Case Is < 5: Label1.Font.Size = 72
    Case 5: Label1.Font.Size = 60
    Case 6: Label1.Font.Size = 48
    Case Else: Label1.Font.Size = 36
End Select

StartStopButton.Caption = "Stop"
Stopped = False
Randomize
Do Until Stopped
    Label1.Caption = Int((Hi - Low + 1) * Rnd + Low)
    DoEvents ' Wyświetlanie animacji
Loop
Else
    Stopped = True
    StartStopButton.Caption = "Start"
End If
End Sub
```

Ponieważ przycisk spełnia dwa zadania (uruchamianie i zatrzymywanie generatora liczb losowych), procedura używa zmiennej publicznej, `Stopped`, do śledzenia stanu generatora. Pierwsza część procedury składa się z dwóch struktur `If ... Then`, które sprawdzają poprawność danych wprowadzanych do formantów `TextBox`. Kolejne dwa polecenia sprawdzają, czy podana dolna granica zakresu nie jest wyższa niż górna granica zakresu. Kolejna grupa poleceń dopasowuje rozmiar czcionki formantu `Label` do maksymalnej wartości generowanej liczby. Pętla `Do Until` jest odpowiedzialna za generowanie i wyświetlanie liczb losowych.

Zwróc uwagę na polecenie `DoEvents`, które przekazuje sterowanie do systemu operacyjnego. Sterowanie jest zwracane, gdy system operacyjny skończy przetwarzanie zdarzeń w swojej kolejce. Bez tego polecenia formant `Label` nie mógłby wyświetlać kolejno generowanych liczb losowych. Inaczej mówiąc, to właśnie polecenie `DoEvents` powoduje, że animacja generowanych liczb jest możliwa.

Formularz `UserForm` zawiera również formant `CommandButton`, który spełnia rolę przycisku *Anuluj*. Formant ten jest zlokalizowany poza formularzem `UserForm` i z tego powodu nie jest widoczny. Formant ten ma właściwość `Cancel` ustawioną na wartość `True`, dzięki czemu naciśnięcie klawisza *Esc* spełnia tę samą rolę, co kliknięcie przycisku. Procedura obsługi zdarzenia `Click` tego przycisku po prostu ustawia zmienną `Stopped` na wartość `True` i usuwa formularz `UserForm` z pamięci.

```
Private Sub CancelButton_Click()
    Stopped = True
    Unload Me
End Sub
```



Skoroszyt z tym przykładem (*Generator liczb losowych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Rozdział 13.

Zaawansowane techniki korzystania z formularzy UserForm

W tym rozdziale:

- Zastosowanie niemodalnych okien dialogowych
- Wyświetlanie wskaźnika postępu zadania
- Tworzenie *kreatora* — interaktywnej sekwencji okien dialogowych
- Tworzenie funkcji emulującej funkcję MsgBox języka VBA
- Zezwalanie użytkownikowi na przenoszenie formantów formularza *UserForm*
- Wyświetlanie formularza *UserForm* bez paska tytułowego
- Symulacja paska narzędzi za pomocą formularza *UserForm*
- Emulacja panelu zadań za pomocą formularza *UserForm*
- Zezwolenie użytkownikowi na zmianę rozmiaru formularza *UserForm*
- Obsługa wielu formantów za pomocą jednej procedury obsługi zdarzeń
- Wybór koloru za pomocą formularza *UserForm*
- Wyświetlanie wykresów na formularzach *UserForm*
- Korzystanie z zaawansowanych formularzy danych
- Przykład tworzenia prostej gry puzzle

Niemodalne okna dialogowe

Większość okien dialogowych, z którymi spotykasz się na co dzień, to okna **modalne** (ang. *modal dialog boxes*), które muszą zostać zamknięte, zanim użytkownik będzie mógł kontynuować pracę z aplikacją. Niektóre okna dialogowe są jednak **niemodalne** (ang.

modeless dialog boxes), co oznacza, że w czasie, kiedy jest wyświetlane okno dialogowe, użytkownik może pracować z aplikacją.

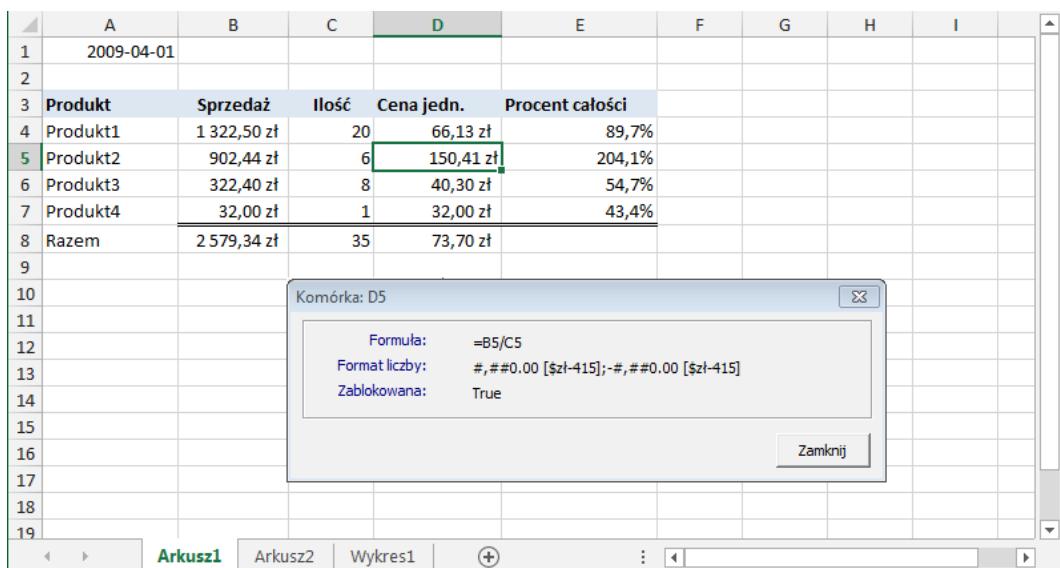
Aby wyświetlić niemodalne okno formularza *UserForm*, powinieneś użyć następującego polecenia:

```
UserForm1.Show vbModeless
```

Słowo kluczowe *vbModeless* jest wbudowaną stałą o wartości 0, stąd powyższa instrukcja działa tak, jak instrukcja:

```
UserForm1.Show 0
```

Na rysunku 13.1 pokazano niemodalne okno dialogowe wyświetlające informacje na temat aktywnej komórki. Podczas wyświetlania okna użytkownik może swobodnie przemieszczać kurSOR do innych komórek, uaktywniać inne arkusze i wykonywać inne operacje w programie Excel.



Rysunek 13.1. Niemodalne okno dialogowe umożliwia użytkownikowi pracę z innymi oknami



Skoroszyt z tym przykładem (*Niemodalny formularz UserForm1.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Kluczowym zagadnieniem jest określenie momentu uaktualnienia zawartości okna dialogowego. W tym celu są monitorowane dwa zdarzenia związane ze skoroszytem: Sheet \rightarrow SelectionChange oraz SheetActivate. Te procedury obsługi zdarzeń są umieszczone w module kodu obiektu ThisWorkbook.



Dodatkowe informacje na temat obsługi zdarzeń znajdziesz w rozdziale 17.

Procedury obsługi zdarzeń zaprezentowano poniżej:

```
Private Sub Workbook_SheetSelectionChange _  
    (ByVal Sh As Object, ByVal Target As Range)  
    Call UpdateBox  
End Sub  
  
Private Sub Workbook_SheetActivate(ByVal Sh As Object)  
    Call UpdateBox  
End Sub
```

Procedury obsługi zdarzeń wywołują procedurę `UpdateBox` zamieszczoną poniżej:

```
Sub UpdateBox()  
    With UserForm1  
        ' Upewnij się, czy arkusz jest aktywny  
        If TypeName(ActiveSheet) <> "Worksheet" Then  
            .lblFormula.Caption = "N/A"  
            .lblNumFormat.Caption = "N/A"  
            .lblLocked.Caption = "N/A"  
            Exit Sub  
        End If  
        .Caption = "Cell: " & ActiveCell.Address(False, False)  
        ' Formula  
        If ActiveCell.HasFormula Then  
            .lblFormula.Caption = ActiveCell.Formula  
        Else  
            .lblFormula.Caption = "(brak)"  
        End If  
        ' Format liczbowy  
        .lblNumFormat.Caption = ActiveCell.NumberFormat  
        ' Komórka zablokowana  
        .lblLocked.Caption = ActiveCell.Locked  
    End With  
End Sub
```

W celu wyświetlenia adresu aktywnej komórki procedura `UpdateBox` modyfikuje właściwość `Caption` formularza `UserForm`, a następnie uaktualnia trzy formanty `Label` (`lblFormula`, `lblNumFormat` oraz `lblLocked`).

Zamieszczone poniżej punkty pozwalają zrozumieć sposób działania tego przykładu.

- Formularz `UserForm` jest wyświetlany jako niemodalny, a zatem w czasie jego wyświetlania możesz nadal pracować z arkuszem.
- Kod na początku procedury sprawdza, czy aktywny arkusz jest arkuszem roboczym. Jeżeli tak nie jest, etykietom nadawane są wartości *N/A*.
- Skoroszyt monitoruje aktywną komórkę za pomocą zdarzenia `Selection_Change` (którego procedura obsługi znajduje się w kodzie modułu `ThisWorkbook`).
- Informacje są wyświetlane w formularzu `UserForm` za pomocą etykiet (formantów typu `Label`).

Na rysunku 13.2 pokazano znacznie bardziej zaawansowaną wersję tego przykładu, gdzie wyświetlane są dodatkowe informacje o wybranej komórce. Kod przykładowu jest zbyt obszerny, żeby można go było tutaj zaprezentować, ale zainteresowani mogą się z nim zapoznać w skoroszycie, który znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

The screenshot shows a Microsoft Excel spreadsheet with data for three months (Styczeń, Luty, Marzec) and a summary row (Razem za kwartał). A UserForm titled "Informacje o komórce: B8 (R8C2)" is displayed, containing the following details:

- Options** section:
 - Automatyczna aktualizacja
 - Wyświetl formuły w notacji R1C1
- Wartość:** 8819
- Wyświetlana jako:** 8 819 zł
- Typ komórki:** Currency
- Format liczby:** #,##0 \$
- Formula:** =SUM(B5:B7)
- Nazwa:** (brak)
- Zabezpieczenie:** Zablokowana
- Komentarz do komórki:** (brak)
- Liczba komórek zależnych:** Komórka nie jest używana w formułach.
- Liczba komórek bezpośrednio zależnych:** Komórka nie jest używana w formułach.
- Liczba użytych komórek:** 3
- Liczba komórek użytych bezpośrednio:** 3

Rysunek 13.2. W tym formularzu UserForm wyświetlono informacje na temat aktywnej komórki



Skoroszyt z tym przykładem (*Niemodalny formularz UserForm2.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Poniżej wyszczególniono najważniejsze elementy tej bardziej zaawansowanej wersji.

- Formularz *UserForm* zawiera pole wyboru (*Automatyczna aktualizacja*). Jeżeli jest ono zaznaczone, formularz *UserForm* uaktualnia się automatycznie. Jeżeli pole nie jest zaznaczone, użytkownik w celu uaktualnienia informacji powinien użyć przycisku *Uaktualnij*.
- W skoroszycie zastosowano moduł klasy do monitorowania dwóch zdarzeń dla wszystkich otwartych skoroszytów: *SheetSelectionChange* oraz *SheetActivate*. W rezultacie kod służący do wyświetlania informacji o bieżącej komórce jest wykonywany automatycznie za każdym razem, kiedy w dowolnym skoroszycie wystąpi jedno z wymienionych zdarzeń (przy założeniu, że zaznaczono pole wyboru *Automatyczna aktualizacja*). Niektóre działania (np. zmiana formatu liczby wyświetlanej w komórce) nie powodują wystąpienia żadnego z wymienionych zdarzeń. Z tego powodu w formularzu *UserForm* znajduje się przycisk *Uaktualnij*.

Niemodalne formularze UserForm w Excelu 2013

Wprowadzenie w Excelu 2013 nowego, jednodokumentowego interfejsu użytkownika spowodowało, że działanie formularzy niemodalnych nieco się zmieniło w raczej niekorzystny dla użytkownika sposób. Otóż kiedy formularz niemodalny jest wyświetlany na ekranie, zostaje powiązany z oknem aktywnego skoroszytu, stąd kiedy przełączysz widok na okno innego skoroszytu, formularz niemodalny może po prostu nie być widoczny. Co gorsza, nawet jeżeli jest widoczny, to nie będzie z nowym oknem działał tak, jak mógłbyś się tego spodziewać.

Jeżeli chcesz, aby dany niemodalny formularz *UserForm* był dostępny dla wszystkich okien skoroszytów, musisz wykonać nieco dodatkowej pracy. Technikę tworzenia takich formularzy dobrze ilustruje skoroszyt *Formularz niemodalny — SDI.xlsxm*, który znajdziesz na stronie internetowej naszej książki.

W przykładowym skoroszycie do tworzenia uchwytów niemodalnych formularzy *UserForm* są używane wywołania odpowiednich funkcji Windows API. Skoroszyt wykorzystuje osobny moduł klasy do monitorowania wszystkich zdarzeń Windows Activate. Kiedy dane okno jest aktywowane, kolejna funkcja Windows API zmienia wskaźnik skoroszytu nadzielnego formularza *UserForm* na odpowiednie nowe okno skoroszytu, dzięki czemu formularz będzie zawsze wyświetlany w oknie aktywnego skoroszytu.



Więcej informacji na temat modułów klas znajdziesz w rozdziale 27.

- Liczniuki wyświetlane dla poprzedników i komórek zależnych uwzględniają wyłącznie komórki aktywnego arkusza. Są to ograniczenia właściwości Precedents i Dependents.
- Ponieważ rozmiar wyświetlanych informacji będzie różny dla różnych arkuszy, wykorzystano kod VBA do określenia rozmiaru etykiet oraz odstępów pomiędzy etykietami w pionie, a także zmiany wysokości formularza *UserForm*, jeżeli zajdzie taka konieczność.

Wyświetlanie wskaźnika postępu zadania

Jednym z zagadnień często poruszanych przez programistów Excela jest pytanie o sposoby tworzenia wskaźnika postępu zadania. *Wskaźnik postępu zadania* to element graficzny podobny do termometru rtęciowego, wskazujący postęp wykonywanego zadania (np. długo działającego makra).

W tym podrozdziale przedstawimy sposoby tworzenia trzech rodzajów wskaźników postępu, które można wykorzystać dla:

- makr, które nie są inicjowane przez formularz *UserForm* (samodzielny wskaźnik postępu);
- makr inicjowanych przez formularz *UserForm* — w tym przypadku w formularzu *UserForm* wykorzystywany jest formant *MultiPage* wyświetlający wskaźnik postępu w czasie działania makra;

Wyświetlanie wskaźnika postępu zadania na pasku stanu

Wskaźnik postępu wykonania makra można przy niewielkim nakładzie pracy wyświetlić na pasku stanu Excela. Niestety większość użytkowników nie jest przyzwyczajona do obserwowania paska stanu i woli elementy bardziej wyeksponowane na ekranie.

Aby wpisać tekst w pasku stanu, możesz skorzystać z następującego polecenia:

```
Application.StatusBar = "Proszę czekać..."
```

Oczywiście w czasie wykonywania makra można uaktualniać tekst wyświetlany na pasku stanu. Jeśli na przykład zmienna Pct reprezentuje procent wykonania zadania, to możemy napisać kod, który co jakiś czas wykonuje następującą instrukcję:

```
Application.StatusBar = "Wykonano " & Pct & "% zadania"
```

Po zakończeniu działania makra należy przywrócić stan początkowy paska stanu za pomocą następującej instrukcji:

```
Application.StatusBar = False
```

Jeżeli nie przywróci się stanu początkowego paska stanu, na pasku nadal będzie wyświetlany ostatni komunikat z makra.

- makr inicjowanych przez formularz *UserForm* — w tym przypadku zwiększa się wysokość formularza *UserForm*, a wskaźnik postępu pojawia się u dołu okna dialogowego.

Wskaźnik postępu można zaimplementować, jeżeli w kodzie programu można oszacować, jak duża część zadania została już w danej chwili wykonana. Sposób dokonania takiego oszacowania może być różny w zależności od wykonywanego makra. Na przykład: jeżeli działanie makra polega na zapisywaniu danych do komórek (a znamy liczbę komórek, do których będą one zapisywane), z łatwością napiszemy kod obliczający procent wykonania zadania. Nawet jeżeli nie jesteś w stanie dokładnie oszacować postępu zadania, zawsze warto użyć wskaźnika postępu do poinformowania użytkownika, że makro nadal działa i że Excel nie zawiesił się podczas jego wykonywania.



Zastosowanie wskaźnika postępu nieco spowalnia działanie makra, ponieważ musi ono wykonać dodatkowe operacje. Jeżeli szybkość działania aplikacji ma znaczenie kluczowe, możesz rozważyć rezygnację ze wskaźnika postępu.

Tworzenie samodzielnego wskaźnika postępu zadania

W tym podpunkcie opisano sposób tworzenia samodzielnego wskaźnika postępu zadania — tzn. takiego, który nie jest inicjowany przez wyświetlenie formularza *UserForm*. Nasze przykładowe makro czyści arkusz i zapisuje 20 000 losowych wartości w wybranym zakresie komórek.

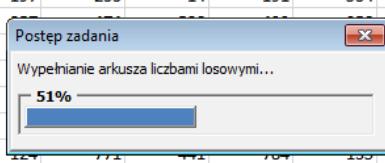
```
Sub GenerateRandomNumbers()
    ' Wstawia losowe wartości do zakresu komórek aktywnego arkusza
    Const RowMax As Integer = 500
    Const ColMax As Integer = 40
    Dim r As Integer, c As Integer
    If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
    Cells.Clear
```

```

For r = 1 To RowMax
    For c = 1 To ColMax
        Cells(r, c) = Int(Rnd * 1000)
    Next c
Next r
End Sub

```

Po dokonaniu kilku modyfikacji tego makra (opisanych poniżej) możemy utworzyć formularz *UserForm* wyświetlający postęp zadania (patrz rysunek 13.3).



A	B	C	D	E	F	G	H	I	J	K	L
1	791	973	168	797	951	304	989	141	752	653	932
2	646	388	479	692	281	771	715	79	85	714	71
3	721	534	426	197	233	14	191	534	570	210	234
4	333	961	859						351	206	727
5	939	795	897						216	844	619
6	638	243	424						89	846	249
7	668	88	587						538	11	716
8	908	192	302						270	717	391
9	379	994	748						634	419	404
10	741	14	872	706	665	342	833	231	118	151	157
11	796	346	885	136	465	164	198	144	852	24	812
12	985	166	764	985	966	27	19	953	105	729	801
13	891	226	751	23	351	877	955	185	789	34	319
14	737	356	856	725	694	991	677	325	954	285	827
15	887	965	351	771	462	474	373	308	293	906	553
16	345	540	277	541	10	756	247	23	638	899	989
17	257	146	938	620	84	99	17	813	462	346	393
18	407	925	406	295	323	91	420	973	378	904	290
19	723	600	17	57	753	148	705	252	643	312	969
20	770	970	872	101	294	14	138	351	649	883	985

Rysunek 13.3. Formularz UserForm wyświetlający postęp działania makra



Skoroszyt z tym przykładem (*Wskaźnik postępu1.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Tworzenie formularza UserForm jako samodzielnego wskaźnika postępu zadania

Aby utworzyć formularz *UserForm* służący do wyświetlania postępu zadania, wykonaj następujące polecenia:

1. Wstaw nowy formularz *UserForm* i ustaw wartość właściwości *Caption* na *Postęp zadania*.
2. Dodaj formant *Frame* i nadaj mu nazwę *FrameProgress*.
3. Dodaj formant *Label* wewnętrz kontenera *Frame* i nadaj mu nazwę *LabelProgress*. Usuń tytuł etykiety (właściwość *Caption*) i ustaw kolor tła (właściwość *BackColor*) na taki, który będzie spełniał Twoje oczekiwania.

Na tym etapie rozmiar etykiety i jej położenie nie mają znaczenia.

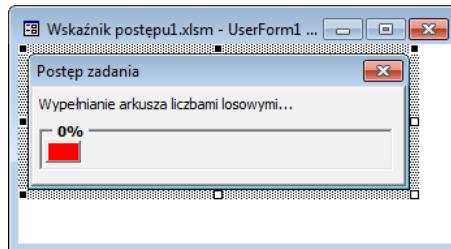
- 4.** Wprowadź dodatkową etykietę powyżej ramki, w której umieścisz opis wykonywanych działań (opcjonalnie).

W naszym przykładzie możemy użyć na przykład takiego opisu: *Wpisywanie wartości losowych...*

- 5.** Dostosuj wygląd formularza *UserForm* i położenie formantów w taki sposób, aby przypominały formularz pokazany na rysunku 13.4.

Rysunek 13.4.

Ten formularz
UserForm będzie służył
jako wskaźnik postępu
zadania



Można oczywiście zastosować dowolne elementy formatowania. Na przykład w formularzu przedstawionym na rysunku 13.4 tak zmodyfikowałem właściwość *SpecialEffect* formantu *Frame*, że wygląda jak „zagłębiony” w oknie dialogowym.

Tworzenie procedur obsługi zdarzeń dla samodzielnego wskaźnika postępu zadania

Zasada działania tego wskaźnika polega na automatycznym uruchomieniu procedury w momencie wyświetlanego formularza *UserForm*. Nie możemy użyć zdarzenia *Initialize*, które następuje *przed* właściwym wyświetleniem formularza *UserForm*. Zamiast tego możesz jednak skorzystać ze zdarzenia *Activate*, które zachodzi w momencie wyświetlenia formularza *UserForm*, zatem idealnie nadaje się do tego celu.

Niżej pokazaną procedurę należy wprowadzić w module kodu formularza *UserForm*. Jej działanie polega na wywołaniu procedury *Main* w momencie wyświetlenia formularza *UserForm*. Procedura *Main* zapisana w module VBA jest makrem, które uruchamia się podczas wyświetlania wskaźnika postępu zadania.

```
Private Sub UserForm_Activate()
    Call GenerateRandomNumbers
End Sub
```

Zmodyfikowaną wersję procedury *GenerateRandomNumbers* (omawianej nieco wcześniej) przedstawiono poniżej. Zauważ, że dodatkowy kod programu śledzi postęp działania procedury i zapisuje go w zmiennej o nazwie *PctDone*.

```
Sub GenerateRandomNumbers()
    ' Wstawia losowe wartości do zakresu komórek aktywnego arkusza
    Dim Counter As Long
    Const RowMax As Long = 500
    Const ColMax As Long = 40
    Dim r As Integer, c As Long
    Dim PctDone As Double
```

```
If TypeName(ActiveSheet) <> "Worksheet" Then Exit Sub
Cells.Clear
Counter = 1
For r = 1 To RowMax
    For c = 1 To ColMax
        Cells(r, c) = Int(Rnd * 1000)
        Counter = Counter + 1
    Next c
    PctDone = Counter / (RowMax * ColMax)
    Call UpdateProgress(PctDone)
Next r
Unload UserForm1
End Sub
```

Procedura GenerateRandomNumbers zawiera dwie pętle. W pętli wewnętrznej znajduje się wywołanie procedury UpdateProgress, która pobiera jeden argument (zmienną PctDone, która reprezentuje postęp działania makra). Zmienna PctDone przechowuje wartości z przedziału od 0 do 100.

```
Sub UpdateProgress(Pct)
    With UserForm1
        .FrameProgress.Caption = Format(Pct, "0%")
        .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
        .Repaint
    End With
End Sub
```

Tworzenie procedury startowej dla samodzielnego wskaźnika postępu zadania

Brakuje nam już tylko procedury wyświetlającej formularz *UserForm*. Wprowadź następującą procedurę w module kodu VBA:

```
Sub ShowUserForm()
    With UserForm1
        .LabelProgress.Width = 0
        .Show
    End With
End Sub
```



Dodatkowym ulepszeniem naszej procedury może być dopasowanie wskaźnika postępu do aktualnego stylu formatowania arkusza. Aby to zrobić, wystarczy w procedurze ShowUserForm umieścić poniższe polecenie:

```
.LabelProgress.BackColor = ActiveWorkbook.Theme.  
ThemeColorScheme.Colors(msoThemeAccent1)
```

Jak działa samodzielny wskaźnik postępu zadania?

W momencie uruchamiania procedury ShowDialog szerokość obiektu Label jest ustalona na wartość 0. Następnie metoda Show formularza UserForm1 wyświetla formularz *UserForm* (będący wskaźnikiem postępu zadania). W momencie wyświetlania formularza UserForm następuje wyzwolenie zdarzenia Activate, które wywołuje procedurę Generate

→ RandomNumbers. Procedura GenerateRandomNumbers zawiera kod, który wywołuje procedurę UpdateProgress za każdym razem, kiedy zmienia się wartość licznika pętli (zmienna *r*). Zauważ, że procedura UpdateProgress wykorzystuje metodę Repaint formularza *UserForm*, bez której zmiany szerokości etykiet nie byłyby widoczne. Przed zakończeniem działania ostatnie polecenie procedury GenerateRandomNumbers powoduje usunięcie z pamięci formularza *UserForm*.

Aby można było dostosować tę technikę do własnych potrzeb, trzeba określić sposób obliczania procentu wykonania zadania i przypisać obliczoną wartość do zmiennej *PctDone*. Sposób obliczania tego procentu może być różny dla różnych aplikacji. Jeżeli kod jest wykonywany w pętli (jak w zaprezentowanym przykładzie), obliczenie procentu wykonania zadania jest łatwe. W przeciwnym razie czasami trzeba obliczyć procent wykonania zadania w kilku miejscach kodu.

Wyświetlanie wskaźnika postępu zadania za pomocą formantu MultiPage

W poprzednim przykładzie makro nie było inicjowane przez formularz *UserForm*. W wielu przypadkach jednak uruchomienie makra następuje po naciśnięciu przez użytkownika przycisku *OK*, znajdującego się na formularzu *UserForm*. Technika, którą opiszymy w tym podrozdziale, jest w takiej sytuacji lepszym rozwiązaniem i przyjmuje następujące założenia:

- projekt jest ukończony i zostały usunięte błędy;
- do zainicjowania makra w projekcie wykorzystano formularz *UserForm* (bez formantu *MultiPage*);
- istnieje sposób zmierzenia postępu wykonania makra.

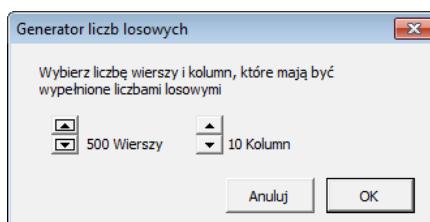


Skoroszyt z tym przykładem (*Wskaźnik postępu2.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Kolejna procedura — podobnie jak w poprzednim przykładzie — wstawia szereg losowych wartości do zakresu komórek aktywnego arkusza. Różnica polega na tym, że tym razem aplikacja zawiera formularz *UserForm*, który pozwala użytkownikowi na określenie liczby wierszy i kolumn, do których zostaną wstawione wartości losowe (patrz rysunek 13.5).

Rysunek 13.5.

Użytkownik może zdefiniować liczbę wierszy i kolumn, do których zostaną wstawione wartości losowe

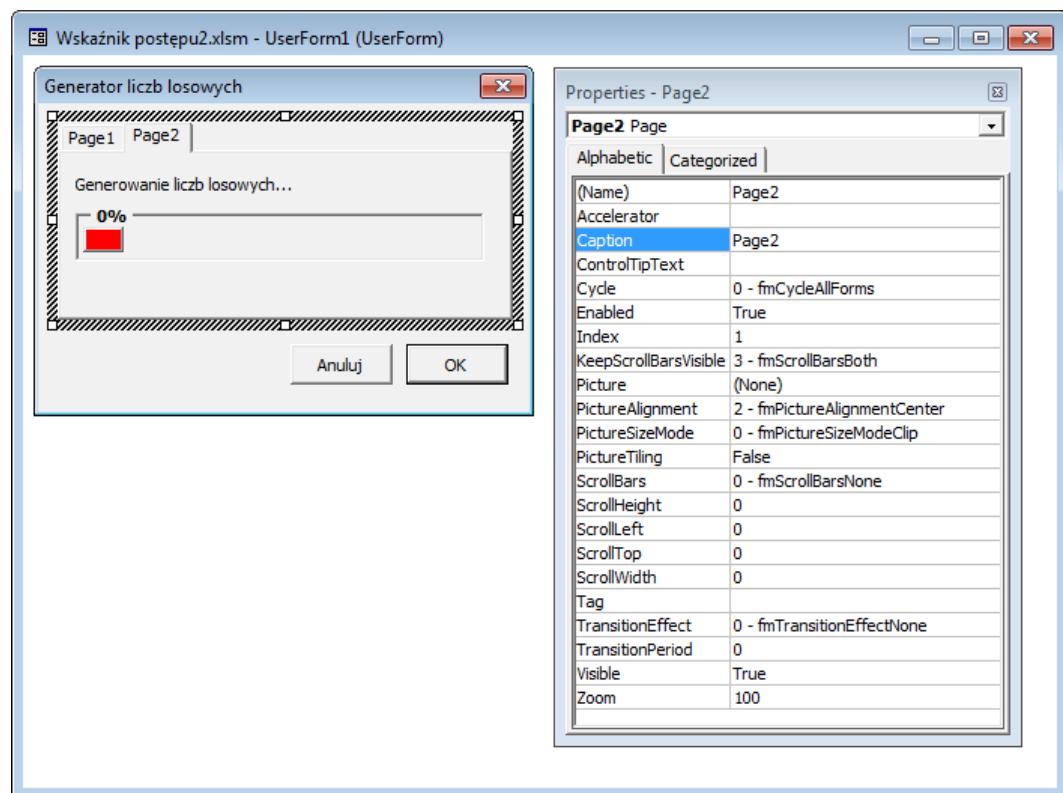


Modyfikowanie formularza UserForm w celu wyświetlania wskaźnika postępu zadania wykorzystującego formant MultiPage

Przyjmijmy, że tworzenie formularza *UserForm* zostało zakończone. Następne zadanie polega na dodaniu formantu *MultiPage*. Na jego pierwszej karcie (Strona1) umieścimy formanty, które znajdowały się w poprzedniej wersji okna. Formanty umieszczone na drugiej karcie (Strona2) będą tworzyć wskaźnik postępu zadania. W momencie rozpoczęcia wykonywania makra kod VBA spowoduje zmianę wartości właściwości *Value* formantu *MultiPage* i w efekcie formanty wyświetlane na pierwszej stronie zostaną ukryte, a formularz wyświetli wskaźnik postępu zadania.

Przygotowania rozpoczynamy od dodania formantu *MultiPage* do formularza *UserForm*. W dalszej kolejności przenosimy wszystkie istniejące formanty formularza *UserForm* na pierwszą kartę formantu *MultiPage*.

Następnie aktywujemy drugą kartę formantu *MultiPage* i konfigurujemy go tak, jak to zostało przedstawione na rysunku 13.6. W zasadzie jest to taki sam zestaw formantów, jakiego używaliśmy w poprzednim przykładzie.



Rysunek 13.6. Na drugiej karcie formantu *MultiPage* zostanie wyświetlony wskaźnik postępu zadania

Aby skonfigurować formant MultiPage, powinieneś wykonać polecenia przedstawione poniżej:

1. Dodaj formant Frame i nadaj mu nazwę FrameProgress.
2. Dodaj formant Label wewnątrz kontenera Frame i nadaj mu nazwę LabelProgress. Usuń tytuł etykiety (właściwość Caption) i ustaw kolor tła (właściwość BackColor) na czerwony.
3. Dodaj kolejną etykietę, w której umieścisz opis wykonywanych działań (opcjonalnie).
4. Uaktywnij formant MultiPage (a nie jego kartę) i ustaw właściwość Style na 2 — fmTabStyleNone (co spowoduje ukrycie zakładek).
5. Aby uwzględnić ukrycie zakładek, będziesz najprawdopodobniej zmienić rozmiar formantu MultiPage.



Najprostszym sposobem zaznaczenia wybranej karty formantu MultiPage w sytuacji, kiedy jego karty nie są widoczne, jest użycie rozwijanej listy w oknie *Properties*.

Aby wyświetlić żądaną kartę, zaznacz formant MultiPage i ustaw wartość właściwości Value na 0 (by wyświetlić kartę Page1), 1 (żeby wyświetlić kartę Page2) i tak dalej.

Utworzenie procedury UpdateProgress dla wskaźnika postępu zadania utworzonego za pomocą formantu MultiPage

W module kodu formularza *UserForm* umieść następującą procedurę:

```
Sub UpdateProgress(Pct)
    With UserForm1
        .FrameProgress.Caption = Format(Pct, "0%")
        .LabelProgress.Width = Pct * (.FrameProgress.Width - 10)
        .Repaint
    End With
End Sub
```

Procedura *UpdateProgress* jest wywoływaną z makra, które jest wykonywane, kiedy użytkownik naciśnie przycisk *OK*. Jej zadaniem jest aktualizowanie wskaźnika postępu zadania.

Modyfikowanie procedury obsługującej wskaźnik postępu zadania utworzony za pomocą kontrolki MultiPage

Teraz musisz zmodyfikować procedurę wykonywaną po kliknięciu przycisku *OK*. Będzie to procedura obsługi zdarzenia *Click* dla przycisku; nadamy jej nazwę *OKButton_Click*. Najpierw, na początku procedury, wprowadzimy poniższą instrukcję:

```
MultiPage1.Value = 1
```

Instrukcja uaktywnia drugą kartę formantu MultiPage (tę, która wyświetla wskaźnik postępu zadania).

Następny krok zależy od aplikacji. Musimy napisać kod do obliczania procentu wykonania zadania i przypisania wyniku do zmiennej *PctDone*. W większości przypadków takie

obliczenie wykonywane jest w pętli. Po wykonaniu obliczenia zastosujemy poniższą instrukcję do uaktualnienia wskaźnika postępu zadania:

```
Call UpdateProgress(PctDone)
```

Jak działa wskaźnik postępu zadania utworzony za pomocą formantu MultiPage

Ta technika jest bardzo prosta i — jak się przekonaliśmy — wykorzystuje jedynie formularz *UserForm*. Kod powoduje przełączanie stron formantu *MultiPage* i przekształca zwykłe okno dialogowe we wskaźnik postępu zadania. Ponieważ karty formantu *MultiPage* są ukryte, całość nie przypomina nawet tradycyjnego wyglądu tego formantu.

Wyświetlanie wskaźnika postępu zadania bez korzystania z kontrolki MultiPage

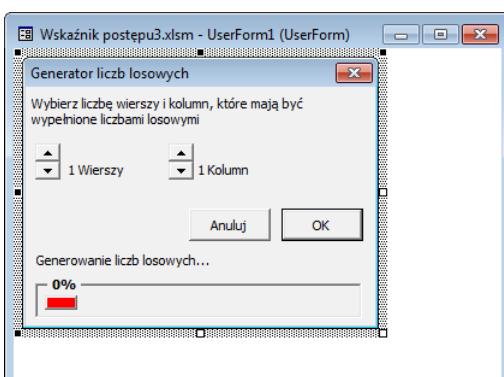
Przykład zaprezentowany w tym punkcie przypomina przykład z punktu poprzedniego. Pokazana technika jest jednak prostsza, ponieważ nie wymaga zastosowania formantu *MultiPage*. Wskaźnik postępu zadania zostanie umieszczony w dolnej części formularza *UserForm*. Normalnie będzie niewidoczny z powodu zmniejszenia wysokości formularza *UserForm*. Kiedy jednak procedura uaktywni opcję wyświetlania wskaźnika, wysokość formularza zostanie zwiększena i wskaźnik postępu pojawi się na ekranie.



Skoroszyt z tym przykładem (*Wskaźnik postępu3.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 13.7 pokazano formularz *UserForm* wyświetlany w oknie edytora VBE.

Rysunek 13.7.
Wskaźnik postępu zadania zostanie ukryty poprzez zmniejszenie wysokości formularza *UserForm*



Właściwość *Height* formularza *UserForm* ma wartość 172. Jednak przed wyświetleniem formularza *UserForm* wartość właściwości *Height* jest zmniejszana do 124 (co oznacza, że formant wskaźnika postępu zadania nie jest widoczny). Dopiero kiedy użytkownik naciśnie przycisk *OK*, procedura zmodyfikuje wartość właściwości *Height* do 172 za pomocą następującego polecenia:

```
Me.Height = 172
```

Na rysunku 13.8 przedstawiono wygląd formularza *UserForm* z wyświetlonym wskaźnikiem postępu.

	A	B	C	D	E	F	G	H	I	J	K	L
1	705	533	579	289	301	774	14	760	814	709		
2	45	414	862	790	373	961	871	56	949	364		
3	524	767	53	562	459	399	522	547	263	279		
4	829	824	589	562	459	399	522	547	243	533		
5	106	999	676	562	459	399	522	547	284	45		
6	295	382	300	562	459	399	522	547	162	646		
7	410	412	712	562	459	399	522	547	80	457		
8	905	261	785	562	459	399	522	547	428	97		
9	561	694	913	562	459	399	522	547	677	502		
10	513	462	353	562	459	399	522	547	60	390		
11	364	489	155	562	459	399	522	547	938	654		
12	506	390	107	562	459	399	522	547	18	210		
13	73	105	331	562	459	399	522	547	827	81		
14	191	678	454	562	459	399	522	547	89	757		
15	401	461	492	207	329	95	589	169	927	97		
16	443	272	872	750	272	673	256	89	30	322		
17	790	297	235	480	254	340	44	482	206	864		
18	588	754	927	331	542	80	634	410	960	114		
19	923	620	347	149	479	219	993	130	28	345		
20	547	922	538	406	847	826	672	721	996	339		

Rysunek 13.8. Gotowy wskaźnik postępu w działaniu

Tworzenie kreatorów

W wielu aplikacjach wykorzystuje się kreatory, które prowadzą użytkownika przez proces wykonywania działań (przykładem może być kreator importu Excela). *Kreator* to nic innego, jak sekwencja okien dialogowych, które pobierają informacje od użytkownika. Często wybory dokonane przez użytkownika w jednym oknie dialogowym mają wpływ na zawartość okien kolejnych. W większości kreatorów użytkownik ma swobodę poruszania się w przód i w tył w sekwencji okien lub może kliknąć *Zakończ*, aby zatwierdzić wartości domyślne.

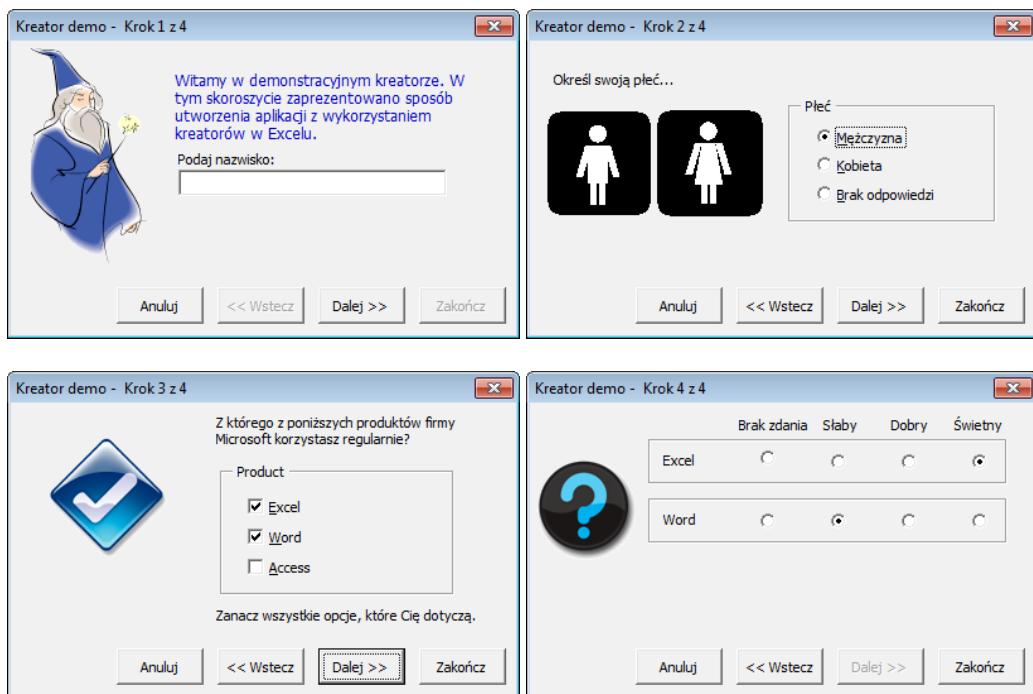
Kreatory można oczywiście tworzyć za pomocą języka VBA i sekwencji formularzy *UserForm*, jednak wydajniejsze jest wykorzystanie pojedynczego formularza *UserForm* i formantu *MultiPage*.

Na rysunku 13.9 pokazano przykład prostego czteroetapowego kreatora, który utworzono z jednego formularza *UserForm* z formantem *MultiPage*. Każde okno kreatora wyświetla się jako oddzielna karta formantu *MultiPage*.



Skoroszyt z tym przykładem (*Demo kreatora.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Sposób utworzenia kreatora opisano w kolejnym podrozdziale.



Rysunek 13.9. W tym kreatorze zastosowano formant MultiPage

Konfigurowanie formantu MultiPage w celu utworzenia kreatora

Pracę nad kreatorem musimy rozpocząć od utworzenia nowego formularza *UserForm* i wstawienia do niego formantu *MultiPage*. Domyślnie formant ten składa się z dwóch kart. Kliknij prawym przyciskiem myszy dowolną kartę *MultiPage* i wybierając pozycję *New Page* z menu podręcznego, wstaw odpowiednią liczbę nowych kart (po jednej dla każdego kroku kreatora). Przykładowy kreator na płycie CD-ROM składa się z czterech kroków, a zatem formant *MultiPage* powinien zawierać cztery karty. Nazwy kart formantu *MultiPage* nie mają znaczenia, ponieważ nie będą wyświetlane na ekranie. Wartość właściwości *Style* po zakończeniu projektowania należy ustawić na 2 — *fmTabStyleNone*.



Podczas pracy z formularzem *UserForm* karty formantu *MultiPage* powinny pozostać widoczne — dzięki temu będzie łatwiejszy dostęp do poszczególnych kart.

Następnie należy wprowadzić odpowiednie formanty na każdej karcie formantu *MultiPage*. Będą one różne dla różnych aplikacji. Aby utworzyć miejsce dla formantów podczas projektowania, czasami trzeba zmienić początkowy rozmiar formantu *MultiPage*.

Dodawanie przycisków do formularza UserForm kreatora

Następną czynnością jest dodanie przycisków sterujących działaniem kreatora. Przyciski zostaną umieszczone poza formantem MultiPage, ponieważ są wyświetlane w każdym kroku kreatora. W większości kreatorów używa się czterech podstawowych przycisków:

- **Anuluj** — anulowanie działania kreatora.
- **Wstecz** — powrót do poprzedniego kroku (podczas wykonywania pierwszego kroku przycisk powinien być niedostępny).
- **Dalej** — przejście do kolejnego kroku (podczas wykonywania ostatniego kroku kreatora przycisk powinien być niedostępny).
- **Zakończ** — zakończenie działania kreatora.

W zaprezentowanym przykładzie przyciski poleceń (obiekty typu CommandButton) noszą nazwy odpowiednio CancelButton, BackButton, NextButton oraz FinishButton.



W niektórych kreatorach użytkownik może kliknąć przycisk **Zakończ** w dowolnym momencie i zaakceptować ustawienia domyślne dla pominiętych kroków kreatora. W innych przypadkach odpowiedź użytkownika jest obowiązkowa. W takich sytuacjach przycisk **Zakończ** powinien być nieaktywny do momentu wprowadzenia wszystkich wymaganych danych. W naszym przykładzie wymagane jest tylko wprowadzenie wartości w polu tekstowym w pierwszym kroku.

Programowanie przycisków kreatora

Każdy z czterech przycisków kreatora wymaga zdefiniowania procedury obsługującej zdarzenie Click. Procedurę obsługi zdarzenia Click dla przycisku CancelButton zaprezentowano poniżej.

```
Private Sub CancelButton_Click()
    Dim Msg As String
    Dim Ans As Integer
    Msg = "Anułować działanie kreatora?"
    Ans = MsgBox(Msg, vbQuestion + vbYesNo, APPNAME)
    If Ans = vbYes Then Unload Me
End Sub
```

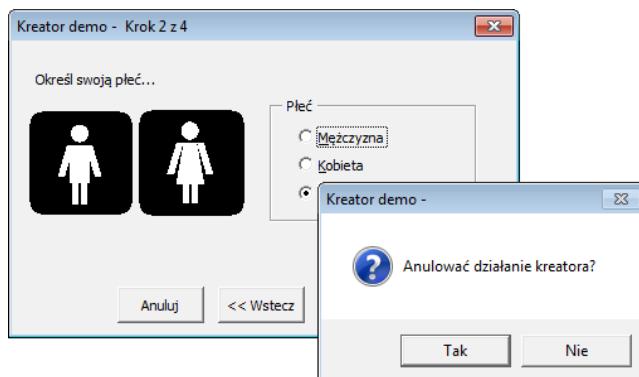
Procedura wykorzystuje funkcję MsgBox (patrz rysunek 13.10) do sprawdzenia, czy użytkownik rzeczywiście chce zakończyć działanie kreatora. Jeżeli kliknie *Tak*, nastąpi zamknięcie formularza UserForm bez wykonania żadnych działań. Takie sprawdzenie jest oczywiście opcjonalne.

Procedury obsługi zdarzeń dla przycisków *Wstecz* oraz *Dalej* zaprezentowano poniżej:

```
Private Sub BackButton_Click()
    MultiPage1.Value = MultiPage1.Value - 1
    UpdateControls
End Sub

Private Sub NextButton_Click()
    MultiPage1.Value = MultiPage1.Value + 1
    UpdateControls
End Sub
```

Rysunek 13.10.
Kliknięcie przycisku
Anuluj spowoduje
wyświetlenie okna
dialogowego z prośbą
o potwierdzenie



Pokazane wyżej dwie procedury są bardzo proste. Ich działanie polega na modyfikacji właściwości Value formantu MultiPage, a następnie wywołaniu procedury UpdateControls (patrz poniżej).

Zadaniem procedury UpdateControls jest uaktywnienie i dezaktywacja przycisków Back ➪Button oraz NextButton.

```
Sub UpdateControls()
    Select Case MultiPage1.Value
        Case 0
            BackButton.Enabled = False
            NextButton.Enabled = True
        Case MultiPage1.Pages.Count - 1
            BackButton.Enabled = True
            NextButton.Enabled = False
        Case Else
            BackButton.Enabled = True
            NextButton.Enabled = True
    End Select

    ' Aktualizacja nagłówka
    Me.Caption = APPNAME & " Krok " -
        & MultiPage1.Value + 1 & " z " -
        & MultiPage1.Pages.Count

    ' Pole Nazwisko jest wymagane
    If tbName.Text = "" Then
        FinishButton.Enabled = False
    Else
        FinishButton.Enabled = True
    End If
End Sub
```

W celu wyświetlenia numeru bieżącego kroku oraz całkowitej liczby kroków procedura modyfikuje właściwość Caption formularza *UserForm* (APPNAME jest stałą publiczną zdefiniowaną w module *Module1*). Następnie sprawdza pole tekstowe *tbName* na pierwszej stronie. Jest to pole obowiązkowe, a zatem kiedy jest puste, kliknięcie przycisku *Zakończ* nie powinno być możliwe. Jeżeli okaże się, że pole *tbName* jest puste, przycisk *Finish* ➪Button będzie nieaktywny.

Zależności programowe w kreatorach

W większości kreatorów wcześniejsze decyzje użytkownika mają wpływ na zawartość kolejnych okien dialogowych. W naszym przykładzie użytkownik określa w kroku 3., jakich produktów używa, a w kroku 4. ocenia te produkty. Przyciski opcji oceny będą widoczne tylko wtedy, gdy wcześniej użytkownik wybierze jakieś produkty.

W programie można to osiągnąć, monitorując zdarzenie Change formantu MultiPage. Za każdym razem, kiedy zmienia się wartość właściwości Value formantu MultiPage (w wyniku kliknięcia przycisków *Wstecz* lub *Dalej*), wywoływana jest procedura MultiPage1_Change. Przed wyświetleniem ostatniej karty formantu MultiPage (krok 4.) procedura sprawdza wartości pól wyboru z kroku 3. i wykonuje odpowiednie korekty okna z kroku 4.

W naszym przykładzie w kodzie użyto dwóch tablic formantów — jednej dla pól wyboru produktów (krok 3.), a drugiej dla kontrolek Frame (krok 4.). Zastosowano pętlę *For ... Next*, która ukrywa nieużywane ramki produktów oraz koryguje ich pozycję w pionie. Jeżeli w kroku 3. nie zaznaczono żadnych produktów, wszystkie formanty w kroku 4. będą ukryte, poza polem tekstowym wyświetlającym tekst *Kliknij Zakończ, aby wyjść* (jeżeli w kroku 1. wprowadzono nazwę) lub *Wprowadzenie nazwiska w kroku 1 jest obowiązkowe* (jeżeli w kroku 1. nie wprowadzono nazwiska). Kod procedury MultiPage1_Change zaprezentowano poniżej.

```
Private Sub MultiPage1_Change()
    Dim TopPos As Long
    Dim FSpace As Long
    Dim AtLeastOne As Boolean
    Dim i As Long

    ' Czy wyświetlić stronę z oceną?
    If MultiPage1.Value = 3 Then
        ' Tworzenie tablicy formantów CheckBox
        Dim ProdCB(1 To 3) As MSForms.CheckBox
        Set ProdCB(1) = cbExcel
        Set ProdCB(2) = cbWord
        Set ProdCB(3) = cbAccess

        ' Tworzenie tablicy formantów Frame
        Dim ProdFrame(1 To 3) As MSForms.Frame
        Set ProdFrame(1) = FrameExcel
        Set ProdFrame(2) = FrameWord
        Set ProdFrame(3) = FrameAccess

        TopPos = 22
        FSpace = 8
        AtLeastOne = False

        ' Pętla przetwarzająca produkty
        For i = 1 To 3
            If ProdCB(i) Then
                ProdFrame(i).Visible = True
                ProdFrame(i).Top = TopPos
                TopPos = TopPos + ProdFrame(i).Height + FSpace
                AtLeastOne = True
            End If
        Next i
    End If
End Sub
```

```

        Else
            ProdFrame(i).Visible = False
        End If
    Next i

    ' Czy wybrano co najmniej jeden produkt?
    If AtLeastOne Then
        lblHeadings.Visible = True
        Image4.Visible = True
        lblFinishMsg.Visible = False
    Else
        lblHeadings.Visible = False
        Image4.Visible = False
        lblFinishMsg.Visible = True
        If tbName = "" Then
            lblFinishMsg.Caption =
                "Wprowadzenie Nazwiska w kroku 1 jest obowiązkowe."
        Else
            lblFinishMsg.Caption =
                "Kliknij Zakończ, aby wyjść."
        End If
    End If
End If
End Sub

```

Wykonywanie zadań za pomocą kreatorów

Kiedy użytkownik kliknie przycisk **Zakończ**, kreator wykonuje swoje zadanie: przenosi informacje z formularza UserForm do kolejnego pustego wiersza w arkuszu. Ta procedura, o nazwie **FinishButton_Click**, jest bardzo prosta. Rozpoczyna działanie od znalezienia kolejnego pustego wiersza w arkuszu, a następnie przypisuje tę wartość do zmiennej (**r**). Pozostała część procedury po prostu przekształca wartości formantów i wprowadza dane do arkusza.

```

Private Sub FinishButton_Click()
    Dim r As Long
    r = Application.WorksheetFunction. -
        CountA(Range("A:A")) + 1

    ' Wstaw nazwisko
    Cells(r, 1) = tbName.Text

    ' Wstaw informację o płci
    Select Case True
        Case obMale: Cells(r, 2) = "Mężczyzna"
        Case obFemale: Cells(r, 2) = "Kobieta"
        Case obNoAnswer: Cells(r, 2) = "Obcy..."
    End Select

    ' Wstaw informacje o produktach
    Cells(r, 3) = cbExcel
    Cells(r, 4) = cbWord
    Cells(r, 5) = cbAccess

    ' Wstaw oceny
    If obExcel1 Then Cells(r, 6) = ""
    If obExcel2 Then Cells(r, 6) = 0

```

```

If obExcel3 Then Cells(r, 6) = 1
If obExcel4 Then Cells(r, 6) = 2
If obWord1 Then Cells(r, 7) = ""
If obWord2 Then Cells(r, 7) = 0
If obWord3 Then Cells(r, 7) = 1
If obWord4 Then Cells(r, 7) = 2
If obAccess1 Then Cells(r, 8) = ""
If obAccess2 Then Cells(r, 8) = 0
If obAccess3 Then Cells(r, 8) = 1
If obAccess4 Then Cells(r, 8) = 2

    ' Zamknij formularz
    Unload Me
End Sub

```

Jeżeli po przetestowaniu kreatora okaże się, że wszystko działa poprawnie, można ustawić wartość właściwości Style formantu MultiPage na 2 — fmTabStyleNone, co spowoduje ukrycie zakładek kart formantu.

Emulacja funkcji MsgBox

Funkcja MsgBox języka VBA (któրą omawialiśmy w rozdziale 10.) jest dość niecodzienna, ponieważ w odróżnieniu od większości funkcji wyświetla okno dialogowe. Z drugiej jednak strony podobnie jak inne funkcje zwraca wartość, a ściślej — liczbę całkowitą reprezentującą przycisk, który nacisnął użytkownik.

W kolejnym przykładzie omówimy funkcję mojego autorstwa, która emuluje działanie funkcji MsgBox języka VBA. Na pierwszy rzut oka wydaje się, że utworzenie takiej funkcji jest stosunkowo proste. Zastanówmy się nad tym dokładniej! Funkcja MsgBox jest niezwykle uniwersalna ze względu na szeroki zakres argumentów, które przyjmuje. W efekcie utworzenie funkcji, która emuluje funkcję MsgBox, stanowi duże wyzwanie.



Celem tego ćwiczenia nie jest utworzenie alternatywnej funkcji wyświetlania komunikatów, ale zademonstrowanie sposobu tworzenia stosunkowo skomplikowanych funkcji z wykorzystaniem formularzy UserForm. Niektórym użytkownikom przyda się ponadto możliwość dostosowania komunikatów do własnych potrzeb (można np. zmienić czcionkę, kolory, tekst przycisków itd.). Jak sam się przekonasz, dostosowanie tej funkcji do własnych potrzeb jest stosunkowo proste.

Moją funkcję emulującą nazwałem MyMsgBox. Emulacja jest bliska oryginału, ale nie jest doskonała. Moja funkcja ma następujące ograniczenia:

- Nie obsługuje argumentu Helpfile (powodującego dodanie przycisku *Pomoc*, którego kliknięcie otwiera plik pomocy).
- Nie obsługuje argumentu Context (umożliwiającego określenie identyfikatora kontekstu dla pliku pomocy).
- Nie obsługuje opcji system modal, która powoduje wstrzymanie działania aplikacji do momentu udzielenia odpowiedzi na pytanie w oknie dialogowym.
- Nie potrafi odtwarzać dźwięku w czasie wywoływania.

Składnia funkcji MyMsgBox jest następująca:

```
MyMsgBox(pytanie[, przyciski] [, tytuł])
```

Składnia jest niemal dokładnie taka sama, jak funkcji MsgBox, z tym, że nie można wprowadzić dwóch ostatnich opcjonalnych argumentów (Helpfile i Context). W funkcji MyMsgBox wykorzystamy te same predefiniowane stałe, które wykorzystano w funkcji MsgBox: vbOKOnly, vbQuestion, vbDefaultButton1 itd.



Aby poznać argumenty funkcji MsgBox języka VBA, powinieneś zatrzymać się na systemie pomocy programu Excel.

Emulacja funkcji MsgBox: kod funkcji MyMsgBox

W funkcji MyMsgBox wykorzystano formularz *UserForm* o nazwie MyMsgBoxForm. Sama funkcja, którą zaprezentowano poniżej, jest bardzo krótka. Większość działań wykonywanych jest w procedurze UserForm_Initialize.

```
Public Prompt1 As String  
Public Buttons1 As Integer  
Public Title1 As String  
Public UserClick As Integer  
  
Function MyMsgBox(ByVal Prompt As String, _  
    Optional ByVal Buttons As Integer,  
    Optional ByVal Title As String) As Integer  
    Prompt1 = Prompt  
    Buttons1 = Buttons  
    Title1 = Title  
    MyMsgBoxForm.Show  
    MyMsgBox = UserClick  
End Function
```

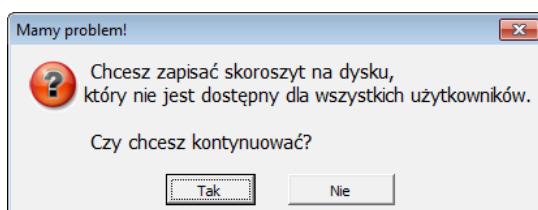


W sieci

Pełny kod funkcji MyMsgBox jest zbyt długi, aby można go było umieścić w niniejszej książce. Skoroszyt z tym przykładem (*Emulacja funkcji MsgBox.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Skoroszyt jest tak skonfigurowany, że możesz samodzielnie przetestować różne opcje.

Działanie funkcji MyMsgBox pokazano na rysunku 13.11. Wygląd okna dialogowego jest bardzo zbliżony do okna oryginalnej funkcji VBA, ale zastosowałem inną czcionkę dla tekstu komunikatu.

Rysunek 13.11.
Wynik działania funkcji emulującej funkcję MsgBox



Jeżeli jesteś posiadaczem systemu dwumonitorowego, to pozycja wyświetlnego formularza *UserForm* niekoniecznie musi znajdować się w centralnej części okna Excela. Aby rozwiązać ten problem, do wyświetlania formularza *UserForm* użyj następującego fragmentu kodu:

```
With MyMsgBoxForm
    .StartUpPosition = 0
    .Left = Application.Left + (0.5 * Application.Width) - (0.5 * .Width)
    .Top = Application.Top + (0.5 * Application.Height) - (0.5 * .Height)
    .Show
End With
```

Poniżej znajduje się kod, którego użyłem do uruchomienia funkcji:

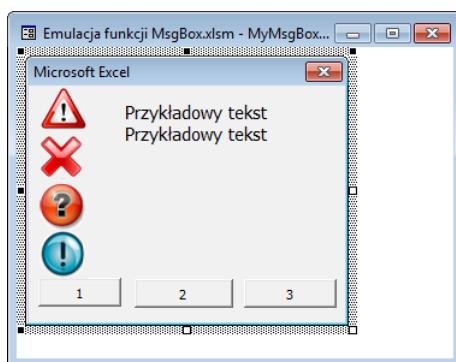
```
Prompt = " Chcesz zapisać skoroszyt na dysku." & vbCrLf
Prompt = Prompt & "który nie jest dostępny dla wszystkich użytkowników." & vbCrLf
Prompt = Prompt & vbCrLf & " Czy chcesz kontynuować?"
Buttons = vbQuestion + vbYesNo
Title = "Mamy problem!"
Ans = MyMsgBox(Prompt, Buttons, Title)
```

Jak działa funkcja MyMsgBox

Zwróćmy uwagę na zastosowanie czterech zmiennych publicznych. Pierwsze trzy (*Prompt1*, *Buttons1* oraz *Title1*) reprezentują argumenty przekazywane do funkcji, natomiast czwarta (*UserClick*) — wartości zwracane przez funkcję. Procedura *UserForm_Initialize* musi pobierać te informacje i wysyłać je z powrotem do funkcji, stąd zastosowanie zmiennych publicznych jest jedynym sposobem na realizację takiego zadania.

Formularz *UserForm*, pokazany na rysunku 13.12, zawiera cztery formanty *Image* (po jednym dla każdej ikony), trzy przyciski polecień (formanty *CommandButton*) oraz pole tekstowe (formant *TextBox*).

Rysunek 13.12.
Formularz *UserForm*
dla funkcji *MyMsgBox*



Początkowo do przechowywania tych czterech ikon używałem formantu *Image*, ale zauważyłem, że dookoła ikon wyświetlane jest delikatne obramowanie. Z tego powodu zamiениłem formant *Image* na *Label*, na którym obramowania ikon nie są wyświetlane.

Procedura `UserForm_Initialize` sprawdza argumenty i wykonuje następujące operacje:

- sprawdza, który obrazek należy wyświetlić (jeżeli wybrano wyświetlenie obrazka), i ukrywa pozostałe;
- sprawdza, które przyciski należy wyświetlić, i ukrywa pozostałe;
- sprawdza, który przycisk jest domyślny;
- wyśrodkowuje przyciski w oknie dialogowym;
- określa podpisy przycisków;
- określa pozycję tekstu w oknie dialogowym;
- określa szerokość i wysokość okna dialogowego (za pomocą wywołania funkcji interfejsu API pobierającej rozdzielcość ekranu);
- wyświetla formularz `UserForm`.

Dodatkowo zdefiniowano trzy procedury obsługi zdarzeń (po jednej dla każdego przycisku). W procedurach tych następuje sprawdzenie, który przycisk kliknięto, i zwrócenie wartości funkcji poprzez ustawienie wartości zmiennej `UserClick`.

Interpretacja drugiego argumentu (*przyciski*) jest dość złożona i może stanowić pewne wyzwanie. Argument ten składa się z kilku stałych, które są do siebie dodawane. Na przykład drugi argument może przyjąć następującą postać:

```
vbYesNoCancel + vbQuestion + vbDefaultButton3
```

Ten argument powoduje utworzenie okna `MsgBox` z trzema przyciskami (*Tak*, *Nie* i *Anuluj*), wyświetlenie ikony pytajnika i ustawienie trzeciego przycisku jako domyślnego. Rzeczywista wartość argumentu wynosi 547 (3+32+512).

Na podstawie tej jednej liczby trzeba uzyskać trzy informacje. W tym celu należy przekształcić argument na postać binarną i sprawdzić ustawienie poszczególnych bitów. Na przykład liczba 547 w postaci binarnej to 1000100011. Cyfry binarne na pozycjach od 4 do 6 określają wyświetlającą ikonę, od 8 do 10 — wyświetlane przyciski, natomiast cyfry na pozycjach 1, 2 informują, który przycisk jest domyślny.

Wykorzystanie funkcji MyMsgBox do emulacji funkcji MsgBox

Jeżeli chcesz wykorzystać funkcję `MyMsgBox` w swoim własnym projekcie, powinieneś najpierw wyeksportować moduł `MyMsgBoxMod` oraz formularz `MyMsgBoxForm`, a następnie zaimportować oba pliki do własnego projektu. Od tej chwili będziesz mógł używać funkcji `MyMsgBox` w swoim programie.

Formularz UserForm z formantami, których położenie można zmieniać

Formularz *UserForm* przedstawiony na rysunku 13.13 zawiera trzy formanty *Image*. Użytkownik może za pomocą myszy przesuwać te obrazy w obrębie okna z obsługą myszy. Nie jestem całkiem przekonany, czy opisana tutaj technika może mieć jakieś zastosowanie praktyczne, ale ten przykład pomoże Ci zrozumieć zdarzenia związane z myszą.

Rysunek 13.13.
Trzy obrazy, które mogą być przesuwane w oknie dialogowym przy użyciu myszy



Skoroszyt z tym przykładem (*Przesuwanie formantów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Każdy z formantów *Image* posiada dwie skojarzone procedury obsługi zdarzeń: *MouseDown* oraz *MouseMove*. Poniżej przedstawiono kody procedur obsługi zdarzeń dla formantu *Image1* (procedury obsługi zdarzeń dla pozostałych formantów są identyczne).

```
Private Sub Image1_MouseDown(ByVal Button As Integer,
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ' Pozycja początkowa przy naciśnięciu przycisku myszy
    OldX = X
    OldY = Y
    Image1.ZOrder 0
End Sub

Private Sub Image1MouseMove(ByVal Button As Integer,
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    ' Przesuń obraz
    If Button = 1 Then
        Image1.Left = Image1.Left + (X - OldX)
        Image1.Top = Image1.Top + (Y - OldY)
    End If
End Sub
```

Naciśnięcie przycisku myszy powoduje wygenerowanie zdarzenia `MouseDown` i wtedy zapamiętywane są współrzędne X oraz Y wskaźnika myszy. Do przechowywania oryginalnej pozycji formantu używane są dwie zmienne publiczne: `OldX` oraz `OldY`. Procedura zmienia również właściwość `ZOrder`, która powoduje, że formant (obraz) jest wyświetlany na pierwszym planie (niejako „na wierzchu” pozostałych obrazów).

Podeczas przesuwania myszy generowany jest ciąg zdarzeń `MouseMove`. Procedura obsługi tego zdarzenia sprawdza stan przycisku myszy. Jeżeli argument `Button` ma wartość 1, oznacza to, że został naciśnięty lewy przycisk myszy. Jeżeli tak, odpowiednio modyfikowane jest położenie powiązanego formantu `Image`.

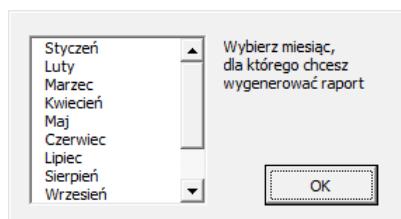
Zwróc uwagę, że wygląd wskaźnika myszy zmienia się w momencie, kiedy ustawisz go nad wybranym obrazem. Dzieje się tak, ponieważ właściwość `MousePointer` jest ustawiona na wartość 15 — `fmMousePointerSizeAll`. Taki styl wskaźnika myszy jest powszechnie używany do zasygnalizowania użytkownikowi, że dany element może zostać przesunięty w inne miejsce.

Formularz *UserForm* bez paska tytułowego

Excel nie posiada mechanizmu pozwalającego na bezpośrednie wyświetlanie formularza *UserForm*, który nie ma paska tytułowego. Na szczęście możemy obejść to ograniczenie przy użyciu kilku funkcji interfejsu Windows API. Na rysunku 13.14 przedstawiono wygląd formularza *UserForm* bez paska tytułowego.

Rysunek 13.14.

Ten formularz
UserForm nie posiada
paska tytułowego



Kolejny przykład formularza *UserForm* pozbawionego paska tytułowego został przedstawiony na rysunku 13.15. W tym oknie dialogowym umieszczono formant `Image` oraz formant `CommandButton`.



W sieci

Oba przykłady zostały umieszczone w skoroszycie *Formularz bez paska tytułowego.xlsx*, który znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Dodatkowo zamieszczono tam również kolejną wersję okna powitalnego, które omawialiśmy w rozdziale 12. Skoroszyt ten, o nazwie *Okno powitalne2.xlsx*, wyświetla okno powitalne bez paska tytułowego.

Wyświetlenie formularza *UserForm* bez paska tytułowego wymaga zastosowania czterech funkcji Windows API: `GetWindowLong`, `SetWindowLong`, `DrawMenuBar` oraz `FindWindowA` (pełne deklaracje funkcji API znajdziesz w pliku na stronie internetowej książki). Procedura `UserForm_Initialize` wywołuje te funkcje w następujący sposób:

Rysunek 13.15.
Kolejny przykład
formularza *UserForm*
pozbawionego paska
tytułowego



```
Private Sub UserForm_Initialize()
    Dim lNgWindow As Long, lFrmHd1 As Long
    lFrmHd1 = FindWindowA(vbNullString, Me.Caption)
    lNgWindow = GetWindowLong(lFrmHd1, GWL_STYLE)
    lNgWindow = lNgWindow And (Not WS_CAPTION)
    Call SetWindowLong(lFrmHd1, GWL_STYLE, lNgWindow)
    Call DrawMenuBar(lFrmHd1)
End Sub
```

Jedną z niedogodności takiego formularza jest fakt, że użytkownik nie może zmienić położenia okna dialogowego pozbawionego paska tytułu. Rozwiązaniem może być zastosowanie procedur obsługi zdarzeń `MouseDown` oraz `MouseMove`, tak jak to zostało opisane w poprzednim podrozdziale.



Ponieważ funkcja `FindWindowA` używa nagłówka formularza *UserForm*, opisana technika nie będzie działała w sytuacji, kiedy właściwości `Caption` formularza zostanie przypisany pusty łańcuch znaków.

Symulacja paska narzędzi za pomocą formularza UserForm

Tworzenie własnego paska narzędzi w wersjach programu Excel wcześniejszych niż 2007 było względnie proste. Niestety, począwszy od wersji 2007 tego programu nie możesz tworzyć własnych pasków narzędzi, a ściślej mówiąc, teoretycznie nadal możesz przy użyciu VBA taki pasek narzędzi utworzyć, ale Excel po prostu zignoruje większość tego typu poleceń języka VBA. Począwszy od Excela 2007, wszystkie niestandardowe paski narzędzi są wyświetlane na karcie *DODATKI* w grupie opcji *Niestandardowe paski narzędzi*. Paski narzędzi umieszczone w tej grupie nie mogą być przesuwane, zdejmowane, nie można zmieniać ich rozmiaru ani ich dokować.

W tym podrozdziale omówimy rozwiązywanie alternatywne: niemodalny formularz *UserForm* symulujący pasek narzędzi. Na rysunku 13.16 przedstawiono wygląd takiego formularza. Dzięki wykorzystaniu odpowiednich wywołań funkcji Windows API pasek tytułowy został nieco skrócony, a sam formularz jest wyświetlany z prostokątnymi narożnikami (zamiast zaokrąglonych). Przycisk *Zamknij* formularza *UserForm* jest również nieco mniejszy niż zwykłe.

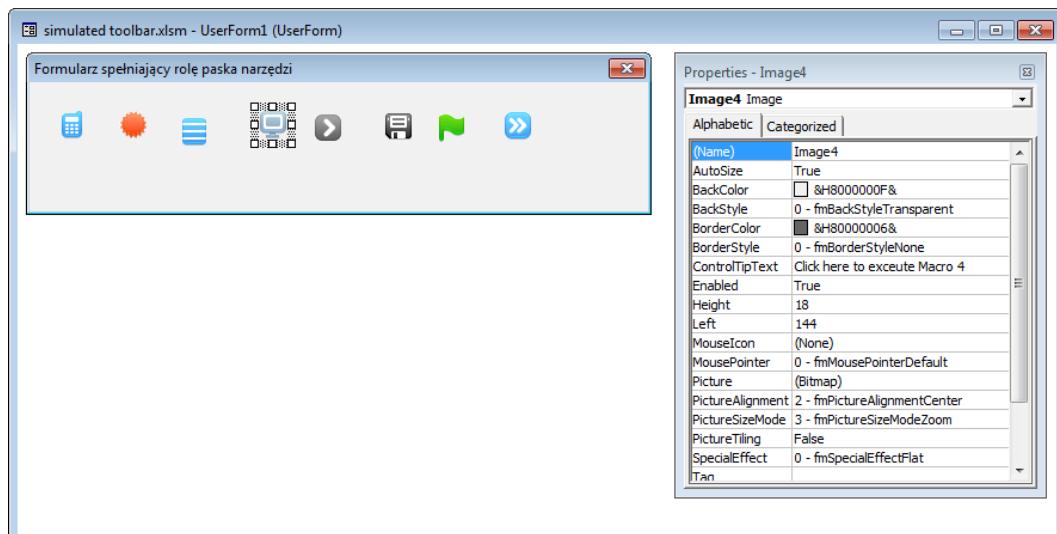
Rysunek 13.16.
Formularz *UserForm*
spełniający rolę
paska narzędzi

	A	B	C	D	E	F	G	H	I	J
1	118	128	222	142	180	267	47	18	280	
2	106	111	160	271	243	203	320	287	225	
3	28	274	179	183	48	235	281	107	340	
4	19	285					6	14	24	
5	43	248					9	329	241	
6	274	148					8	155	340	
7	195	234	31	326	246	116	164	283	341	
8	275	316	335	30	350			77		
9	338	85	63	253	210	329	337	306	287	
10	157	224	64	252	115	342	166	324	279	
11	42	280	302	190	15	342	249	341	175	
12	337	111	314	154	298	283	234	258	319	
13	187	31	316	141	18	309	40	269	32	
14	239	118	234	16	30	306	257	186	309	



Skoroszyt z tym przykładem (*Symulacja paska narzędzi.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na formularzu *UserForm* umieszczono osiem formantów *Image*, z których każdy będzie uruchamiał osobne makro. Na rysunku 13.17 przedstawiono wygląd tego formularza w edytorze VBE. Zwróć uwagę na następujące fakty:



Rysunek 13.17. Formularz *UserForm* symulujący pasek narzędzi

- Poszczególne formanty nie są wyrównane.
- Wyświetlane obrazy są inne niż ich wersje finalne.
- Rozmiar formularza nie został ostatecznie dobrany.
- Pasek tytułowy ma standardowy rozmiar.

Wszystkie detale „kosmetyczne” zostaną załatwione z poziomu kodu VBA, włączając w to „pożyczenie” odpowiednich obrazów ze Wstążki Excela. Na przykład polecenie, które przypisuje odpowiedni obraz do formantu `Image1`, wygląda następująco:

```
Image1.Picture = Application.CommandBars.  
    GetImageMso("ReviewAcceptChange", 32, -32)
```

Więcej informacji na temat korzystania z obrazów ze Wstążki znajdziesz w rozdziale 20.

Kod VBA zajmuje się również odpowiednim wyrównaniem formantów, ustawieniem ich w równych odległościach od siebie i takie dopasowanie rozmiarów formularza *UserForm*, by cały jego obszar był wykorzystany. Dodatkowo za pomocą odpowiednich funkcji Windows API rozmiary paska tytułowego zostaną zmniejszone do takich, jakie ma „prawdziwy” pasek narzędzi. Aby jeszcze bardziej zbliżyć wygląd naszego „paska” do oryginału, tak ustawiemy właściwość `ControlTipText` poszczególnych formantów, aby po ustawieniu wskaźnika myszy nad danym formantem na ekranie pojawiały się okienka podpowiedzi opisujące rolę przycisku.

Jeżeli otworzysz skoroszyt z tym przykładem przekonasz się także, że wygląd poszczególnych formantów nieznacznie zmienia się po ustawieniu nad nimi wskaźnika myszy. Dzieje się tak, ponieważ każdy z formantów `Image` posiada powiązaną z nim procedurę obsługi zdarzenia `MouseMove`, która zmienia właściwość `SpecialEffect` formantu. Poniżej przedstawiamy kod procedury obsługi zdarzenia `MouseMove` dla formantu `Image1` (procedury dla pozostałych formantów są identyczne).

```
Private Sub Image1_MouseMove(ByVal Button As Integer,  
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)  
    Call NormalSize  
    Image1.Width = 26  
    Image1.Height = 26  
End Sub
```

Procedura `Image1_MouseMove` wywołuje procedurę `NormalSize`, która przywraca normalny, domyślny rozmiar wszystkich formantów.

```
Private Sub NormalSize()  
    ' Przywraca normalny rozmiar wszystkich formantów  
    Dim ctl As Control  
    For Each ctl In Controls  
        ctl.Width = 24  
        ctl.Height = 24  
    Next ctl  
End Sub
```

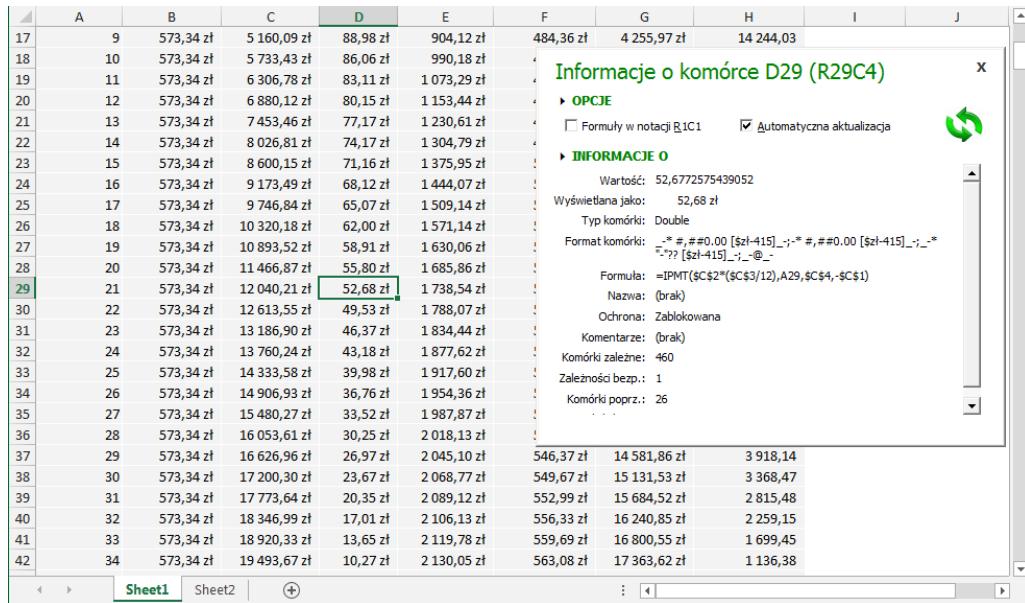
W efekcie działania tych procedur użytkownik po ustawieniu wskaźnika myszy nad formantem odnosi wrażenie „podniesienia” przycisku — dokładnie tak, jak w przypadku rzeczywistego paska narzędzi. Na tym jednak symulacja działania paska narzędzi się kończy. Nie ma możliwości zmiany rozmiaru formularza *UserForm* (na przykład aby zmienić układ paska narzędzi z poziomego na pionowy) i oczywiście nie ma żadnej możliwości zadokowania paska na jednej z krawędzi okna programu Excel.

Emulowanie panelu zadań za pomocą formularza UserForm

W pakiecie Office 2013 funkcjonalność panelu zadań została znacząco rozszerzona i obecnie jest on wykorzystywany do modyfikacji formatowania wielu obiektów, włączając w to również wykresy i obrazy. Panel zadań otrzymał również zupełnie nowy wygląd.

Muszę przyznać, że spędziłem całkiem sporo czasu na próbach emulowania wyglądu panelu zadań za pomocą formularza *UserForm*. Rezultaty moich wysiłków możesz zobaczyć na rysunku 13.18. W przykładzie użyłem takiego samego niemodalnego formularza *UserForm* jak ten, o którym mówiliśmy na początku tego rozdziału (patrz rysunek 13.2). Formularz możesz przenosić z miejsca na miejsce, lapiąc go za wyświetlany tytuł (dokładnie tak samo, jak to się dzieje w przypadku oryginalnego panelu zadań). Mój formularz zawiera również przycisk *Zamknij*, zlokalizowany w prawym górnym rogu, i podobnie jak oryginalny panel zadań, w razie potrzeby wyświetla pionowy pasek przewijania.

```
Me.BackColor = RGB(255, 255, 255)
Frame1.BackColor = RGB(255, 255, 255)
Frame2.BackColor = RGB(255, 255, 255)
```



Rysunek 13.18. Formularz UserForm symulujący panel zadań

Formanty Frame nie mogą mieć przezroczystego tła, stąd musiałem ustawić kolor tła dla każdego z nich z osobna.

Aby utworzyć formularz *UserForm*, który ma jasnoszary kolor tła, pasujący do domyślnego, systemowego motywu graficznego, użyłem następującego polecenia:

RGB(240, 240, 240)

Aby emulować ciemnoszary motyw graficzny, możesz użyć następującego polecenia:

RGB(222, 222, 222)

Myślę, że nieźle poradziłem sobie z oddaniem podstawowego wyglądu panelu zadań, ale jeszcze całkiem sporo pozostało do poprawienia w zakresie jego funkcjonalności. Na przykład poszczególnych sekcji nie można zwijać i rozwijać; nie można również zadowolić formularza na krawędzi ekranu. Oprócz tego użytkownik nie ma możliwości skalowania rozmiaru formularza, choć jest to wykonalne (jak to zrobić, dowiesz się w następnym podrozdziale).



Skoroszyt z tym przykładem (*Emulacja panelu zadań.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci

Formularze UserForm z możliwością zmiany rozmiaru

Excel wykorzystuje kilka okien dialogowych, których rozmiary możesz zmieniać. Jednym z takich okien jest na przykład okno *Menedżer nazw*, którego rozmiary możesz zmieniać, przeciągając przy użyciu myszy jego prawy, dolny narożnik.

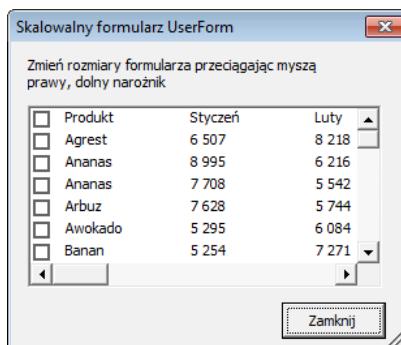
Jeżeli chciałbyś utworzyć takie skalowalne okno formularza *UserForm*, szybko przekonasz się, że nie ma bezpośredniego sposobu realizacji takiego zamierzenia. Jednym z możliwych rozwiązań jest skorzystanie z funkcji Windows API. Taka metoda co prawda działa, ale w praktyce takie rozwiązanie jest dosyć złożone i nie generuje żadnych zdarzeń, stąd kod programu nie może reagować na zmianę rozmiaru formularza. W tym rozdziale omówimy jednak inną, o wiele prostszą metodę tworzenia formularzy *UserForm*, których rozmiary można będzie zmieniać.



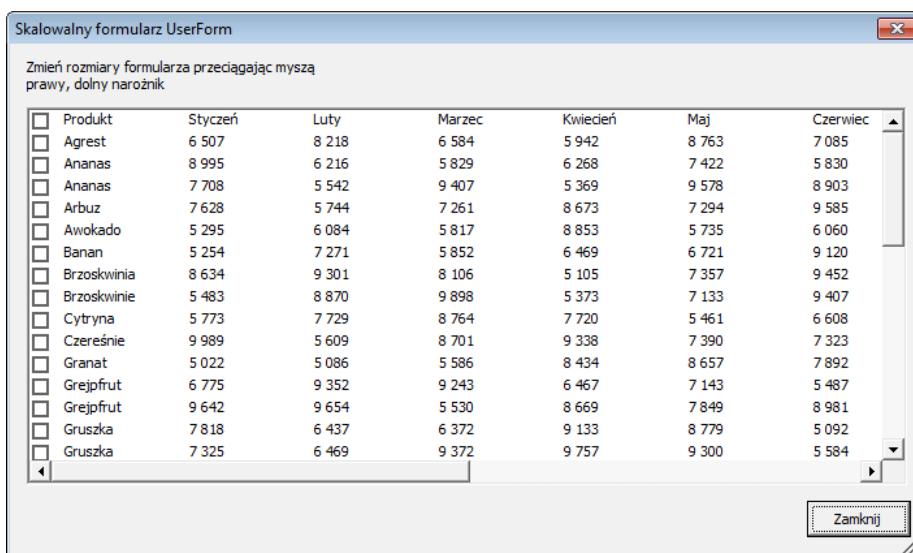
Podziękowania za opracowanie tej metody należą się Andy'emu Pope, ekspertowi w dziedzinie Excela i posiadaczowi tytułu Microsoft MVP, zamieszkałemu w Wielkiej Brytanii. Andy jest jednym z najbardziej kreatywnych deweloperów aplikacji Excela, jakich znam. Więcej ciekawych informacji (i bardzo wiele interesujących przykładów) znajdziesz na jego stronie internetowej, <http://andypope.info>.

Na rysunku 13.19 przedstawiono formularz *UserForm* omawiany w tym podrozdziale. Umieszczono na nim formant *ListBox*, który wyświetla dane z arkusza. Zwróć uwagę na paski przewijania formantu *ListBox*. Ich obecność świadczy o tym, że istnieje o wiele więcej danych do wyświetlenia, niż jest w stanie jednocześnie zmieścić się w polu formantu. Zwróć również uwagę na wygląd prawego, dolnego rogu okna formularza, gdzie znajdziesz dobrze Ci znany uchwyt zmiany rozmiarów okna.

Rysunek 13.19.
Okno skalowalnego formularza *UserForm*



Na rysunku 13.20 przedstawiono ten sam formularz *UserForm*, ale już po zmianie rozmiarów dokonanej przez użytkownika. Zwróć uwagę na fakt, że rozmiary formantu *ListBox* również uległy zmianie, natomiast przycisk *Zamknij* znajduje się w tej samej pozycji względem okna. Okno formularza możesz rozciągnąć aż do wymiarów ograniczonych rozmiarami używanego monitora.



Rysunek 13.20. Wygląd formularza *UserForm* po zmianie rozmiarów



Skoroszyt z tym przykładem (*Skalowalny formularz UserForm.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Cała sztuczka polega na użyciu formantu *Label*, który jest dodawany do formularza *UserForm* w czasie działania programu. Uchwyt zmiany rozmiarów, znajdujący się w prawym, dolnym rogu okna formularza, to w rzeczywistości formant *Label* wyświetlający jeden znak: // (kod znaku 111) czcionki Marlett, z drugiego zestawu znaków. Formant ten (o nazwie *objResizer*) jest dodawany do formularza *UserForm* przez procedurę *UserForm_Initialize* po uruchomieniu programu:

```

Private Sub UserForm_Initialize()
    ' Dodawanie formantu skalowania w prawym, dolnym rogu formularza
    Set objResizer = Me.Controls.Add("Forms.Label.1", MResizer, True)
    With objResizer
        .Caption = Chr(111)
        .Font.Name = "Marlett"
        .Font.Charset = 2
        .Font.Size = 14
        .BackStyle = fmBackStyleTransparent
        .AutoSize = True
        .ForeColor = RGB(100, 100, 100)
        .MousePointer = fmMousePointerSizeNWSE
        .ZOrder
        .Top = Me.InsideHeight - .Height
        .Left = Me.InsideWidth - .Width
    End With
End Sub

```



Pomimo że formant Label jest dodawany w czasie działania programu, procedura obsługi zdarzeń tego formantu została umieszczona w module kodu. Jak widać, dołączenie do modułu kodu obsługi zdarzeń obiektu, który nie istnieje, nie stanowi żadnego problemu.

Opisana technika bazuje na następujących założeniach:

- Użytkownik może zmieniać położenie formantu na formularzu *UserForm* (patrz „Formularz UserForm z formantami, których położenie można zmieniać” we wcześniejszej części rozdziału).
- Istnieją zdarzenia, które pozwalają na identyfikację ruchów myszy i odczytywanie współrzędnych położenia wskaźnika myszy (w naszym przypadku są to zdarzenia MouseDown oraz MouseMove).
- Kod VBA może zmieniać rozmiar formularza *UserForm* w czasie działania programu, natomiast użytkownik nie może tego robić.

Jeżeli podejdziemy do tego zagadnienia w nieco bardziej kreatywny sposób, to okaże się, że możemy zamienić zmiany położenia formantu Label dokonywane przez użytkownika na informacje, które mogą być wykorzystane do zmiany rozmiaru formularza *UserForm*.

Kiedy użytkownik kliknie obiekt objResizer (formant Label), wykonywana jest procedura obsługi zdarzenia objResizer_MouseDown:

```

Private Sub objResizer_MouseDown(ByVal Button As Integer,
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)
    If Button = 1 Then
        LeftResizePos = X
        TopResizePos = Y
    End If
End Sub

```

Procedura ta jest wykonywana tylko w sytuacji, kiedy naciśnięty jest lewy przycisk myszy (czyli inaczej mówiąc, kiedy argument Button ma wartość 1) i wskaźnik myszy znajduje się nad formantem objResizer. Współrzędne X i Y położenia wskaźnika myszy w momencie kliknięcia są zapamiętywane w zmiennych modułowych LeftResizePos oraz Top

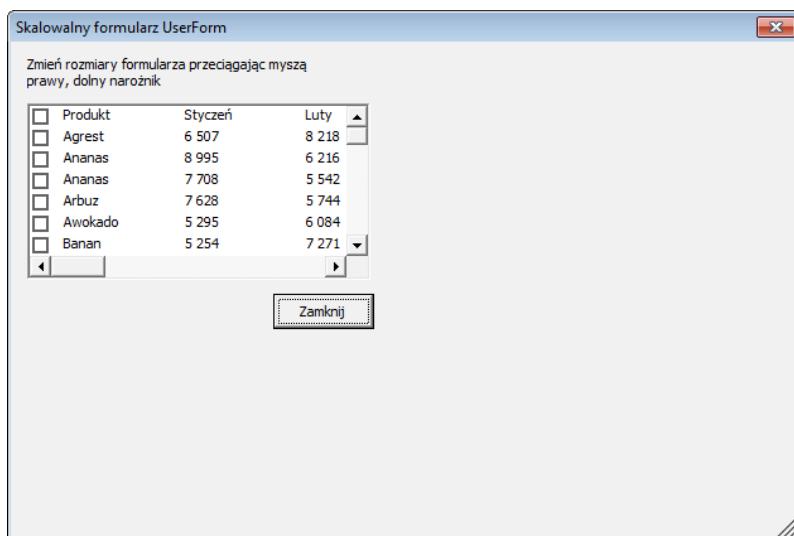
→ResizePos.

Kolejne ruchy myszy powodują wygenerowanie zdarzenia `MouseMove` i wykonywana jest procedura obsługi zdarzenia `objResizer_MouseMove`. Poniżej przedstawiamy kod pierwszej wersji tej procedury:

```
Private Sub objResizer_MouseMove(ByVal Button As Integer,  
    ByVal Shift As Integer, ByVal X As Single, ByVal Y As Single)  
    If Button = 1 Then  
        With objResizer  
            .Move .Left + X - LeftResizePos, .Top + Y - TopResizePos  
            Me.Width = Me.Width + X - LeftResizePos  
            Me.Height = Me.Height + Y - TopResizePos  
            .Left = Me.InsideWidth - .Width  
            .Top = Me.InsideHeight - .Height  
        End With  
    End If  
End Sub
```

Jeżeli dokładnie przeanalizujesz kod tej procedury, z pewnością zauważysz, że właściwości `Width` i `Height` formularza `UserForm` są modyfikowane na podstawie zmiany położenia formantu `objResizer`. Na rysunku 13.21 przedstawiono wygląd formularza `UserForm` po przesunięciu przez użytkownika formantu `Label` w dół i na prawo.

Rysunek 13.21.
Kod VBA zamienia zmianę położenia formantu `Label` na nowe wartości właściwości `Width` i `Height` formularza `UserForm`



Problem oczywiście polega na tym, że inne formanty formularza `UserForm` nie reagują na zmianę rozmiaru formularza. Rozmiar formantu `ListBox` powinien być zwiększyony, a przycisk `CommandButton` powinien zostać przeniesiony w inne miejsce, tak aby nadal pozostawał w dolnej prawej części okna.

Do zmiany rozmiarów formantów umieszczonych na formularzu `UserForm` potrzebny będzie dodatkowy kod VBA, który umieścimy w procedurze obsługi zdarzenia `objResizer_MouseMove`. Zadanie zostanie wykonane przez następujący fragment kodu:

```
' Zmiana rozmiaru formantu ListBox  
On Error Resume Next  
With ListBox1
```

```

    .Width = Me.Width - 22
    .Height = Me.Height - 100
End With
On Error GoTo 0
Zmiana położenia przycisku Zamknij
With CloseButton
    .Left = Me.Width - 70
    .Top = Me.Height - 54
End With

```

Rozmiar i położenie obu formantów są modyfikowane względem zmiany rozmiarów formularza *UserForm* (stąd użycie słowa kluczowego *Me*). Po dodaniu kodu opisanego powyżej nasze okno dialogowe zachowuje się tak, jak moglibyśmy tego oczekiwąć. Użytkownik może je przeskalać do dowolnych rozmiarów, a formanty będą posłusznie reagować na każdą zmianę.

Jest chyba oczywiste, że najtrudniejszym zagadnieniem podczas tworzenia skalowanego okna dialogowego jest zapewnienie odpowiedniego sposobu zmiany rozmiaru formantów. Jeżeli w oknie dialogowym znajdują więcej niż dwa, trzy formanty, całe zagadnienie może stać się znacznie bardziej złożone.

Obsługa wielu przycisków formularza UserForm za pomocą jednej procedury obsługi zdarzeń

Każdy przycisk *CommandButton* na formularzu *UserForm* musi posiadać swoją własną procedurę obsługi zdarzeń. Jeżeli na przykład zdefiniowaliśmy dwa obiekty *CommandButton*, musimy odpowiednio zdefiniować dwie procedury obsługi zdarzeń *Click*:

```

Private Sub CommandButton1_Click()
    ' tutaj należy umieścić kod obsługi zdarzenia
End Sub

Private Sub CommandButton2_Click()
    ' tutaj należy umieścić kod obsługi zdarzenia
End Sub

```

Innymi słowy, nie możemy utworzyć makra, które będzie wykonywane w momencie kliknięcia *dowolnego* przycisku. Każda procedura obsługi zdarzenia *Click* jest „na sztywno” powiązana z konkretnym przyciskiem *CommandButton*. Można jednak spowodować, żeby w każdej procedurze obsługi zdarzenia znajdowało się wywołanie uniwersalnego makra, ale w takim przypadku trzeba przekazać do niego argument wskazujący na to, który przycisk został naciśnięty. W poniższych przykładach naciśnięcie przycisku *CommandButton1* lub *CommandButton2* spowoduje wykonanie procedury *ButtonClick* i przekazanie do niej argumentu informującego, który przycisk został naciśnięty:

```

Private Sub CommandButton1_Click()
    Call ButtonClick(1)
End Sub

Private Sub CommandButton2_Click()
    Call ButtonClick(2)
End Sub

```

Jeżeli w formularzu UserForm skonfigurowano wiele przycisków CommandButton, definiowanie wszystkich procedur obsługi zdarzeń może być uciążliwe. Wygodniejsze byłoby zdefiniowanie pojedynczej procedury rozpoznającej przycisk, który został naciśnięty, i na tej podstawie podejmującej odpowiednie działanie.

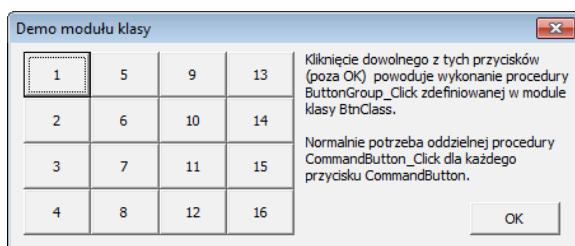
W tym podrozdziale opiszymy sposób obejścia tego ograniczenia poprzez zdefiniowanie nowej klasy i umieszczenie jej w module klasy.



Skoroszyt z tym przykładem (*Wiele przycisków.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Polecenia przedstawione poniżej pozwalają na utworzenie przykładowego formularza *UserForm* pokazanego na rysunku 13.22.

Rysunek 13.22.
*Wiele przycisków
i tylko jedna procedura
obsługi zdarzeń*



1. Utwórz formularz *UserForm* i dodaj kilka przycisków CommandButton.

(W przykładzie umieszczonym na płycie CD-ROM dołączonej do książki zdefiniowano 16 przycisków). W tym przykładzie formularzowi nadano nazwę UserForm1.

2. Dodaj moduł klasy do projektu (wybierz polecenie *Insert/Class Module*), nadaj mu nazwę BtnClass i wprowadź podany poniżej kod.

```
Public WithEvents ButtonGroup As MsForms.CommandButton
```

```
Private Sub ButtonGroup_Click()  
    Dim Msg As String  
    Msg = "Nacisnąłś przycisk: " & ButtonGroup.Name & vbCrLf & vbCrLf  
    Msg = Msg & "Etykieta: " & ButtonGroup.Caption & vbCrLf  
    Msg = Msg & "Lewa współrzędna: " & ButtonGroup.Left & vbCrLf  
    Msg = Msg & "Górna współrzędna: " & ButtonGroup.Top  
    MsgBox Msg, vbInformation, ButtonGroup.Name
```

```
End Sub
```

3. Procedurę ButtonGroup_Click należy zmodyfikować odpowiednio do potrzeb konkretnego przykładu.



Opisaną technikę można dostosować do innych formantów. W tym celu należy zmodyfikować nazwę typu w deklaracji Public WithEvents. Aby na przykład zastosować procedurę dla przycisków opcji (OptionButton) zamiast poleceń (CommandButton), należy użyć deklaracji zbliżonej do tej przedstawionej poniżej:

```
Public WithEvents ButtonGroup As MsForms.OptionButton
```

4. Dodaj moduł VBA i wprowadź podany niżej kod.

```
Sub ShowDialog()
    UserForm1.Show
End Sub
```

Procedura po prostu wyświetla na ekranie formularz *UserForm*.

5. W module kodu formularza *UserForm* wprowadź przedstawiony poniżej kod procedury *UserForm_Initialize*.

```
Dim Buttons() As New BtnClass

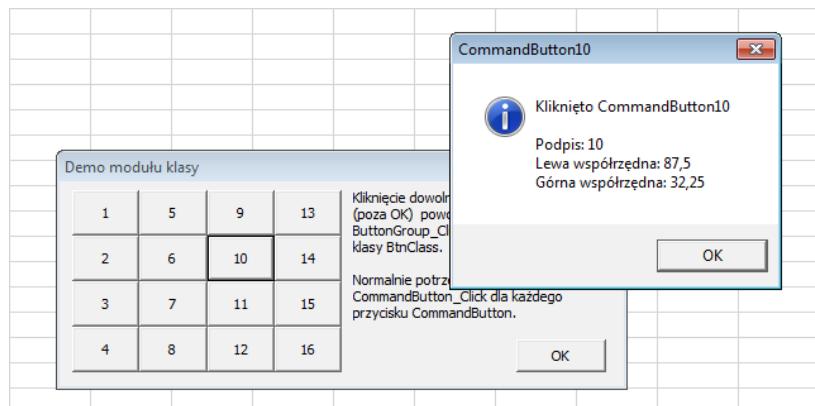
Private Sub UserForm_Initialize()
    Dim ButtonCount As Integer
    Dim ctl As Control
    ' Utworzenie obiektów Button
    ButtonCount = 0
    For Each ctl In UserForm1.Controls
        If TypeName(ctl) = "CommandButton" Then
            Pomiń przycisk OK
            If ctl.Name <> "OKButton" Then
                ButtonCount = ButtonCount + 1
                ReDim Preserve Buttons(1 To ButtonCount)
                Set Buttons(ButtonCount).ButtonGroup = ctl
            End If
        End If
    Next ctl
End Sub
```

Procedura jest wywoływana przez zdarzenie Initialize formularza *UserForm*. Zauważ, że kod wyłącza przycisk *OK* (*OKButton*) z grupy pozostałych przycisków, stąd naciśnięcie przycisku *OK* nie powoduje wykonania procedury *ButtonGroup_Click*.

Po wykonaniu tych operacji, w celu wyświetlania formularza *UserForm* należy wywołać procedurę *ShowDialog*. Kliknięcie dowolnego przycisku (poza przyciskiem *OK*) spowoduje wykonanie procedury *ButtonGroup_Click*. Na rysunku 13.23 pokazano przykład komunikatu wyświetlanego po kliknięciu wybranego przycisku.

Rysunek 13.23.

Procedura
ButtonGroup_Click
wyświetla informację
o tym, który przycisk
został naciśnięty



Wybór koloru za pomocą formularza UserForm

Przykład zaprezentowany w tym podrozdziale to funkcja, która wyświetla okno dialogowe (idea nieco zbliżona do funkcji MsgBox, omawianej wcześniej w tym rozdziale). Nasza funkcja, o nazwie GetAColor, zwraca wartość wybranego koloru:

```
Public ColorValue As Variant  
  
Function GetAColor() As Variant  
    UserForm1.Show  
    GetAColor = ColorValue  
End Function
```

Funkcji GetAColor możesz używać w następujący sposób:

```
UserColor = GetAColor()
```

Wykonanie tego polecenia spowoduje wyświetlenie formularza *UserForm*. Użytkownik wybiera żądzany kolor i naciska przycisk *OK*. Funkcja przypisuje wartość wybranego przez użytkownika koloru do zmiennej UserColor.

Formularz *UserForm*, przedstawiony na rysunku 13.24, składa się z trzech formantów ScrollBar — po jednym dla każdego koloru podstawowego (czerwony, zielony i niebieski). Zakres wartości każdego suwaka może zmieniać się od 0 do 255. W module kodu znajdują się procedury obsługi zdarzeń Change formantów ScrollBar. Na przykład, poniżej zamieszczamy kod procedury, która jest wykonywana po zmianie wartości pierwszego formantu ScrollBar.

```
Private Sub ScrollBarRed_Change()  
    LabelRed.BackColor = RGB(ScrollBarRed.Value, 0, 0)  
    Call UpdateColor  
End Sub
```

Rysunek 13.24.

Okno dialogowe, które pozwala użytkownikowi na zdefiniowanie koloru poprzez zmianę kolorów składowych: czerwonego, zielonego i niebieskiego



Procedura UpdateColor odpowiednio dopasowuje wyświetlającą próbkę koloru oraz aktualizuje wartości RGB wyświetlane w oknie formularza.



Skoroszyt z tym przykładem (*Funkcja GetAColor.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Formularz GetAColor ma jeszcze jedną ciekawą cechę: zapamiętuje ostatnio wybrany kolor. Kiedy funkcja kończy działanie, wartości trzech suwaków kolorów podstawowych są

zapamiętywane w rejestrze systemu Windows przy użyciu następującego kodu (łańcuch APPNAME jest zdefiniowany w module kodu Module1):

```
SaveSetting APPNAME, "Colors", "RedValue", ScrollBarRed.Value
SaveSetting APPNAME, "Colors", "BlueValue", ScrollBarBlue.Value
SaveSetting APPNAME, "Colors", "GreenValue", ScrollBarGreen.Value
```

Procedura UserForm_Initialize pobiera wartości koloru z rejestru i przypisuje je do odpowiednich suwaków kolorów:

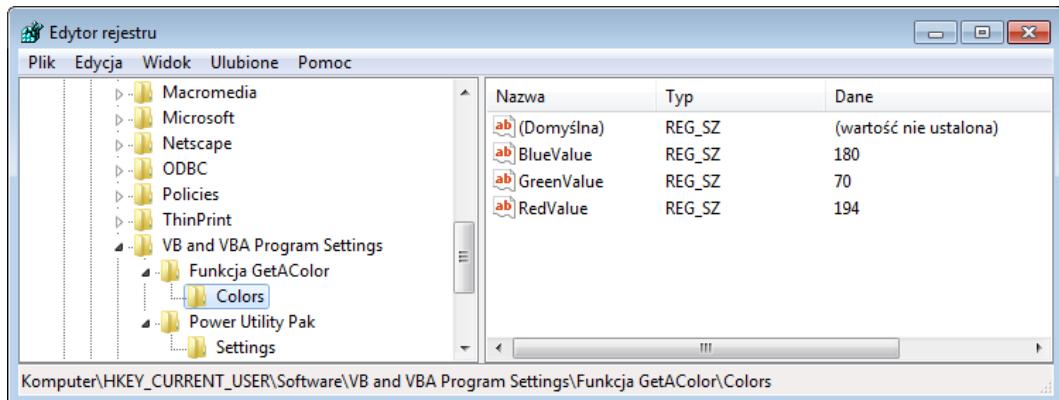
```
ScrollBarRed.Value = GetSetting(APPNAME, "Colors", "RedValue", 128)
ScrollBarGreen.Value = GetSetting(APPNAME, "Colors", "GreenValue", 128)
ScrollBarBlue.Value = GetSetting(APPNAME, "Colors", "BlueValue", 128)
```

Ostatni argument funkcji GetSetting jest wartością domyślną, która jest używana w sytuacji, kiedy dany klucz w rejestrze nie zostanie odnaleziony. W takim przypadku wartości wszystkich trzech kolorów składowych są ustawiane na 128, co w efekcie daje średni szary kolor.

Funkcje SaveSetting i GetSetting zawsze używają następującego klucza rejestru:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\

Na rysunku 13.25 przedstawiono dane zapisane w rejestrze systemu Windows, wyświetcone w programie *Regedit.exe* (*Edytor rejestru*):



Rysunek 13.25. Wartości kolorów składowych są zapisywane w rejestrze systemu Windows i odczytywane za każdym razem, kiedy użyjemy funkcji GetAColor



Więcej szczegółowych informacji o tym, jak Excel używa kolorów, znajdziesz w rozdziale 28.

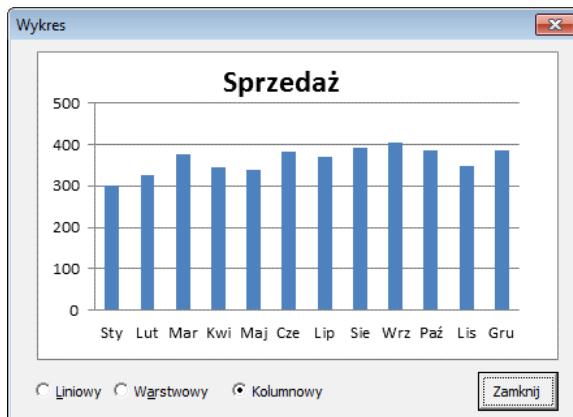
Wyświetlanie wykresów na formularzach UserForm

Co ciekawe, Excel nie oferuje prostego sposobu wyświetlania wykresów na formularzu *UserForm*. Można oczywiście skopiować wykres i wkleić go jako właściwość Picture formantu Image, ale w ten sposób powstanie statyczny obraz wykresu, bez możliwości dynamicznych zmian wartości.

W tym podrozdziale opisano technikę pozwalającą na wyświetlanie wykresów na formularzach *UserForm*. Na rysunku 13.26 przedstawiono formularz *UserForm* z wykresem umieszczonym w formancie *Image*. Sam wykres znajduje się na arkuszu, dzięki czemu formularz *UserForm* zawsze wyświetla aktualny wykres. Zasada działania polega na kopiowaniu wykresu do tymczasowego pliku graficznego i następnie zastosowaniu funkcji *LoadPicture* do przypisania tego pliku do właściwości *Picture* formantu *Image*.

Rysunek 13.26.

Dzięki zastosowaniu małej sztuczki na formularzu *UserForm* możemy wyświetlać „dynamiczne” wykresy



Skoroszyt z tym przykładem (*Wykres na formularzu UserForm.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Aby wyświetlić wykres na formularzu *UserForm*, powinieneś wykonać polecenia opisane poniżej:

1. Utwórz wykres lub wykresy tak, jak to robłeś do tej pory.
2. Wstaw formularz *UserForm* i umieść na nim formant *Image*.
3. Napisz odpowiedni kod VBA, który będzie zapisywał wykres w postaci pliku *GIF*, i następnie ustaw właściwość *Picture* formantu *Image* tak, aby wskazywała na zapisany plik *GIF*.

Aby tego dokonać, powinieneś skorzystać z funkcji *LoadPicture*.

4. Jeżeli chcesz, dodaj inne udogodnienia i wodotryski do programu.

Na przykład, formularz *UserForm* w naszym pliku demo zawiera formantu, które pozwalają na zmianę typu wykresu. Inną modyfikacją może być napisanie kodu, którego zadaniem będzie wyświetlanie wielu wykresów.

Zapisywanie wykresu w postaci pliku *GIF*

Poniższy kod demonstruje sposób zapisywania wykresu do pliku *GIF* (o nazwie *temp.gif*) — w tym przypadku jest to pierwszy obiekt typu *Chart* na arkuszu *Dane*:

```
Set CurrentChart = Sheets("Dane").ChartObjects(1).Chart
Fname = ThisWorkbook.Path & "\temp.gif"
CurrentChart.Export FileName:=Fname, FilterName:="GIF"
```

Modyfikacja właściwości Picture formantu Image

Poniższe polecenie powoduje załadowanie obrazu (reprezentowanego przez zmienną Fname) do formantu Image (zakładamy, że formant Image formularza *UserForm* nosi nazwę Image1):

```
Image1.Picture = LoadPicture(Fname)
```



Opisany powyżej sposób działa całkiem dobrze, ale podczas zapisywania i odtwarzania wykresu występuje niewielkie opóźnienie. W przypadku szybkich komputerów to opóźnienie nie jest jednak odczuwalne.

Tworzenie półprzezroczystych formularzy UserForm

Zazwyczaj formularze *UserForm* są nieprzezroczyste — a co za tym idzie, dokładnie przykrywają to, co znajduje się pod nimi. W razie potrzeby jednak możesz utworzyć półprzezroczysty formularz *UserForm*. Dzięki takiemu rozwiązaniu użytkownik będzie mógł zobaczyć to, co kryje się pod oknem formularza.

Tworzenie półprzezroczystego formularza *UserForm* wymaga użycia szeregu funkcji Windows API. Przejrzystość okna formularza można regulować w zakresie od 0 (formularz całkowicie przezroczysty, czyli inaczej mówiąc, zupełnie niewidoczny) do 255 (okno formularza całkowicie nieprzezroczyste, czyli takie, jakie najczęściej widujemy na ekranie).

Na rysunku 13.27 przedstawiono przykład formularza *UserForm*, którego przejrzystość została ustalona na wartość 128.



Skoroszyt z tym przykładem (*Półprzezroczysty formularz UserForm.xlsxm*) znajdziesz na płycie CD-ROM dołączonej do książki.

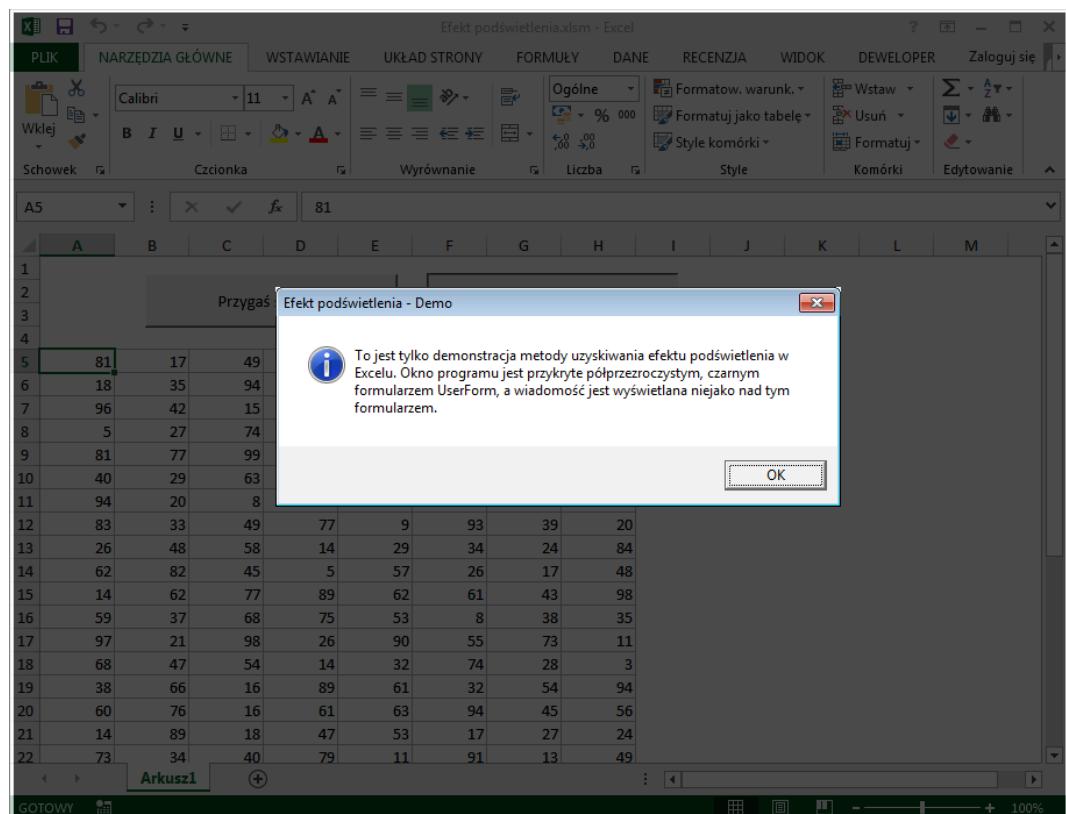
	A	B	C	D	E	F	G	H	I	J	K	L
5	25	9	47	49	84	100	23	55				
6	44	34	10	76	55	73	63	83				
7	41	33	92	33	36	96	55	30				
8	38	61	5	31	1	75	29	86				
9	44	42	8	38	57	83						
10	58	63	70	45	79	2	25	24				
11	72	25	2	53	94	31						
12	38	96	13	44	70	88						
13	32	95	68	24	49	76						
14	43	32	77	37	64	92						
15	76	3	82	27	8	100						
16	97	71	93	10	42	82						
17	13	43	15	71	60	87						
18	6	86	28	56	15	18						
19	38	3	34	71	26	11						
20	3	28	11	52	53	77						
21	1	95	97	32	12	22						
22	12	12	42	13	45	32						
23	49	67	98	53	79	41						
24	67	18	82	4	66	9	63	46				
25	100	64	36	31	13	55	44	52				

Rysunek 13.27. Półprzezroczysty formularz UserForm

Do czego może się nam przydać taki półprzezroczysty formularz *UserForm*? Długo się nad tym zastanawiałem i w końcu doszedłem do wniosku, że jednym z potencjalnych zastosowań może być stworzenie *efektu podświetlenia*. Prawdopodobnie spotkałeś się już z czymś takim na niektórych stronach sieci WWW, gdzie sama zawartość strony jest nieco „przygaszona”, a na ekranie pojawia się w pełnej krasie obraz lub „wyskakuje” okno z reklamą. Z pewnością taki efekt przyciągnie uwagę użytkownika do określonego elementu na ekranie.

Na rysunku 13.28 przedstawiono skoroszyt Excela, który wykorzystuje taki właśnie efekt podświetlenia. Okno Excela jest przygaszone, ale za to okno komunikatu jest wyświetlane normalnie. Jak to działa? Najpierw utworzyłem formularz *UserForm* z czarnym tłem. Następnie napisałem kod programu, który zmieniał rozmiar i położenie formularza tak, aby całkowicie zakrywał okno Excela. Kod realizujący takie zadanie wygląda następująco:

```
With Me
    .Height = Application.Height
    .Width = Application.Width
    .Left = Application.Left
    .Top = Application.Top
End With
```



Rysunek 13.28. Tworzenie efektu podświetlenia okna dialogowego w Excelu

Następnie zmodyfikowałem formularz tak, aby uzyskać efekt półprzezroczystości, co spowodowało wrażenie, że to okno Excela jest „przygaszone”. Teraz pozostało już tylko wyświetlić na tle formularza okno komunikatu (lub kolejny formularz).



Skoroszyt z tym przykładem (*Efekt podświetlenia.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zaawansowane formularze danych

Poniżej zaprezentujemy jeden z najbardziej rozbudowanych formularzy *UserForm*, jakie spotkasz w tej książce. Zaprojektowałem go tak, aby zastąpić standardowy formularz wprowadzania danych, pokazany na rysunku 13.29.

Agent	Data	Obszar	Cena	Liczba sypial	Lazienki	Powierzchnia m2	Typ	Basen	Sprzedany?
Adamek	2007-10-09	Śródmieście	199 000,00 zł	3	2,5	140	Mieszkanie	FALSZ	FALSZ
Adamek	2007-08-19	Śródmieście	214 500,00 zł	4	2,5	173	Dom	PRAWDA	FALSZ
Adamek	2007-04-28	Śródmieście					Dom	FAŁSZ	FAŁSZ
Adamek	2007-07-19	Śródmieście					Dom	FAŁSZ	FAŁSZ
Adamek	2007-02-06	Śródmieście					Dom	PRAWDA	PRAWDA
Jelonek	2007-01-29	Trynek					Dom	FAŁSZ	PRAWDA
Robaczek	2007-04-04	Trynek					Dom	PRAWDA	FAŁSZ
Hirek	2007-02-24	Trynek					Dom	FAŁSZ	FAŁSZ
Rafalski	2007-04-24	Trynek					Dom	PRAWDA	PRAWDA
Adamek	2007-04-21	Sikornik					Dom	PRAWDA	PRAWDA
Szostak	2007-03-24	Trynek					Dom	FAŁSZ	FAŁSZ
Krzaczek	2007-06-09	Trynek					Dom	FAŁSZ	FAŁSZ
Szostak	2007-08-17	Trynek					Dom	FAŁSZ	FAŁSZ
Adamek	2007-06-06	Trynek					Mieszkanie	FAŁSZ	FAŁSZ
Adamek	2007-02-08	Trynek					Mieszkanie	FALSZ	PRAWDA
Raczek	2007-03-30	Trynek					Dom	FAŁSZ	PRAWDA
Barnes	2007-06-26	Sikornik					Dom	FALSZ	FALSZ
Benek	2007-05-12	Śródmieście	229 500,00 zł	4	3	190	Dom	FAŁSZ	PRAWDA
Benek	2007-05-09	Śródmieście	549 000,00 zł	4	3	180	Dom	PRAWDA	FALSZ
Szostak	2007-07-15	Trynek	374 900,00 zł	4	3	365	Dom	FALSZ	FALSZ
Lang	2007-05-03	Trynek	369 900,00 zł	3	2,5	189	Mieszkanie	PRAWDA	FALSZ
Debarkator	2007-01-28	Trynek	360 000,00 zł	4	2,5	185	Mieszkanie	FAŁSZ	PRAWDA

Rysunek 13.29. Formularz danych Excela



Wyświetlanie formularza wprowadzania danych w najnowszych wersjach Excela nie jest wcale takie proste. Polecenie wyświetlające formularz nie jest dostępne bezpośrednio na Wstążce, stąd aby z niego skorzystać, powinieneś najpierw umieścić jego ikonę na pasku narzędzi *Szybki dostęp* lub na wybranej karcie Wstążki. Aby dodać ikonę polecenia do paska narzędzi *Szybki dostęp*, kliknij go prawym przyciskiem myszy i z menu podręcznego, które pojawi się na ekranie wybierz polecenie *Dostosuj pasek narzędzi Szybki dostęp*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Z listy *Wybierz polecenia z, odszukaj i zaznacz opcję Polecenia, których nie ma na Wstążce*, następnie kliknij polecenie *Formularz* i naciśnij przycisk *Dodaj*. Po naciśnięciu przycisku *OK* i zamknięciu okna opcji Excela polecenie *Formularz* będzie widoczne na pasku narzędzi *Szybki dostęp*.

Podobnie jak standardowy formularz danych Excela, tak i mój ulepszony formularz danych operuje na listach danych z arkusza. Jednak — jak możesz przekonać się na podstawie rysunku 13.30 — ma zupełnie inny wygląd i kilka dodatkowych zalet.

The screenshot shows a Microsoft Excel spreadsheet with data in columns A through J. A modal dialog box titled 'J-Walk Enhanced Data Form' is overlaid on the spreadsheet. The dialog has fields for 'Agent' (set to 'Hirek'), 'Data' (set to '2007-02-24'), 'Obszar' (set to 'Trynek'), and 'Cena' (set to '425900'). It also displays 'Liczba sypialni' (5), 'Łazienki' (3), and 'Powierzchnia m2' (224.2). Buttons for 'New', 'Insert', 'Delete', 'Previous', 'Next', 'Undo Entry', and 'Close' are visible. The main spreadsheet table includes columns for Agent, Data, Obszar, Cena, Liczba sypialni, Łazienki, Powierzchnia m2, Typ, Basen, and Sprzedany. The data spans from row 1 to row 25.

Rysunek 13.30. Ulepszony formularz danych opracowany przez autora książki

Opis ulepszonego formularza danych

W ulepszonym formularzu danych wprowadzono usprawnienia wyszczególnione w tabeli 13.1.



Ulepszony formularz danych to produkt komercyjny (przynajmniej w pewnym stopniu). Wersja dla Excela 97 i nowszych jest dostępna na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) — możesz ją rozpowszechniać bez żadnych ograniczeń. Jeżeli chciałbyś dopasować kod programu lub sam formularz *UserForm* do własnych potrzeb, to za niewielką opłatą możesz pobrać pakiet zawierający kompletny kod źródłowy. Szczegółowe informacje znajdziesz na stronie internetowej <http://spreadsheetpage.com>.

Instalacja dodatku — ulepszonego formularza danych

Aby wypróbować ulepszony formularz danych, zainstaluj dodatek, wykonując następujące czynności:

1. Skopij do wybranego katalogu na dysku twardym plik *dataform3.xlam* ze strony internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).
2. Uruchom Excela i przywołaj na ekran okno *Dodatki* (na przykład naciśkając sekwencję klawiszy *lewy Alt, Q, X*).
3. W oknie dialogowym *Dodatki* naciśnij przycisk *Przeglądaj* i na dysku twardym odszukaj i zaznacz plik *dataform3.xlam*, a następnie naciśnij przycisk *OK*.

Tabela 13.1. Porównanie ulepszonego formularza danych ze standardowym formularzem danych Excela

Ulepszony formularz danych	Standardowy formularz danych Excela
Obsługuje dowolną liczbę rekordów i pól.	Ograniczenie do 32 pól.
Okno dialogowe może być wyświetlane w dowolnym rozmiarze, a użytkownik może zmieniać jego rozmiary.	Okno dialogowe dostosowuje rozmiar do liczby pól. Czasami zajmuje cały ekran!
Możliwość wykorzystania zarówno pól tekstowych, jak i pól typu kombi.	W oknie dialogowym można skorzystać wyłącznie z pól tekstowych.
Szerokość nagłówków może być modyfikowana.	Nie można zmieniać szerokości nagłówków.
Można łatwo zmieniać język wykorzystywany w oknie dialogowym (do tej operacji potrzebne jest hasło do odblokowania projektu VBA).	Nie można zmieniać języka używanego w oknie dialogowym.
Rekord wyświetlany w oknie dialogowym jest zawsze widoczny na ekranie i jest wyróżniony, dzięki czemu zawsze wiesz, który rekord jest przetwarzany.	Nie przewija ekranu i nie podświetla bieżącego rekordu.
Początkowo okno dialogowe zawsze wyświetla rekord odpowiadający aktywnej komórce.	Zawsze rozpoczyna wyświetlanie od pierwszego rekordu w bazie danych.
Zamknięcie okna dialogowego powoduje zaznaczenie bieżącego rekordu.	Po zamknięciu okna wybrany rekord jest taki sam, jak przy jego otwieraniu.
Pozwala wprowadzić nowy rekord na dowolnej pozycji w bazie danych.	Nowe rekordy można dodawać tylko na końcu bazy danych.
Zawiera przycisk <i>Undo</i> dla poleceń modyfikacji danych, wstawiania nowego rekordu, usuwania rekordu i wprowadzania nowego rekordu.	Zawiera tylko przycisk <i>Przywróć</i> .
Kryteria wyszukiwania są zapisywane w innym panelu, dzięki czemu zawsze będziesz wiedział, czego szukasz.	Kryteria wyszukiwania nie zawsze są widoczne.
Obsługuje symbole wieloznaczne we wzorcach wyszukiwania (*, ? oraz #).	Formularz danych Excela nie obsługuje symboli wieloznacznych.
Dostępny jest kompletny kod źródłowy w języku VBA, co umożliwia dostosowanie formularza do potrzeb użytkownika.	Formularz danych nie jest napisany w języku VBA, a zatem nie można dostosować go do potrzeb użytkownika.

Po wykonaniu opisanych wyżej poleceń rozszerzony formularz danych będzie dostępny po przejściu na kartę *DANE* i wybraniu polecenia *J-Walk Enhanced DataForm*, znajdującego się w grupie opcji *DataForm*. Nowego formularza danych możesz użyć do pracy z danymi w dowolnym arkuszu lub tabeli.

Puzzle na formularzu UserForm

Kolejnym przykładem w tym rozdziale będą dobrze wszystkim znane puzzle, wyświetlane na formularzu *UserForm* (patrz rysunek 13.31). Puzzle tego typu zostały wynalezione przez Noyesa Chapmana w latach osiemdziesiątych XIX wieku. Napisanie procedury VBA realizującej takie puzzle, oprócz kilku minut niezłą zabawy utworzenie takiego puzzle może być całkiem niezłą okazją do poznania kilku nowych zagadnień.

Rysunek 13.31.
Puzzle na formularzu
UserForm



Naszym zadaniem jest ułożenie wszystkich puzzli (formantów CommandButton) w kolejności rosnącej. Aby przesunąć wybrany element, kliknij puste miejsce znajdujące się obok niego. Pole kombi znajdujące się w prawej, górnej części okna pozwala na wybór rozmiarów puzzli. Do wyboru masz trzy konfiguracje: 3×3, 4×4 i 5×5. Naciśnięcie przycisku *Nowy* powoduje rozpoczęcie nowej gry, a formant typu *Label* przechowuje informację o liczbie wykonanych ruchów.

Aplikacja wykorzystuje moduł klasy do obsługi wszystkich zdarzeń związanych z przyciskami (patrz podrozdział „Obsługa wielu przycisków formularza UserForm za pomocą jednej procedury obsługi zdarzeń”).

Kod VBA tego przykładu jest dosyć rozbudowany, dlatego nie znalazłeś się w książce. Podczas analizowania kodu źródłowego powinieneś pamiętać o kilku sprawach:

- Formanty CommandButton są dodawane do formularza przez kod VBA. Liczba i rozmiar przycisków jest określana za pomocą wartości formantu ComboBox.
- Elementy (puzzle) są mieszane poprzez symulację kilku tysięcy losowych kliknięć przycisków. Innym rozwiązaniem może być po prostu przypisanie poszczególnym elementom losowych wartości, ale czasami mogłoby to skutkować wygenerowaniem nierożwiąwalnych układów elementów.
- Pusty element gry puzzle to po prostu przycisk CommandButton, którego właściwość Visible została ustawiona na wartość False.
- Moduł klasy zawiera jedną procedurę obsługi zdarzeń (MouseUp), która jest wykonywana za każdym razem, kiedy użytkownik kliknie dowolny element puzzle.
- Kiedy użytkownik kliknie dowolny przycisk CommandButton, wartość jego właściwości Caption jest zamieniana z ukrytym przyciskiem, zatem w rzeczywistości żaden przycisk nie zmienia swojego położenia.



Skoroszyt z tym przykładem (*Puzzle.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Video Poker na formularzu UserForm

Na koniec zaprezentujemy dowód na to, że Excel nie musi wcale być nudny. Na rysunku 13.32 przedstawiamy okno formularza *UserForm*, za pomocą którego możemy rozegrać partijkę pokera.

Rysunek 13.32.

Poker w Excelu?
Dlaczego nie!



Program oferuje następujące możliwości:

- Dwa rodzaje gry do wyboru: *Joker's Wild* lub *Jacks Or Better*.
- Wykres przedstawiający historię Twoich zwycięstw (lub porażek).
- Możliwość zmiany stawek.
- Pomoc (wyświetlaną na arkuszu).
- Przycisk bezpieczeństwa, który pozwala na szybkie ukrycie formularza w razie zagrożenia (na przykład niespodziewaną wizytą szefa...).

Jedynie czego tutaj brakuje, to gwar kasyna....



Skoroszyt z tym przykładem (*Video poker.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Jak łatwo się możesz domyślać, kod programu jest zbyt długi, aby go zamieścić w naszej książce, ale mimo to warto się z nim szczegółowo zapoznać. Znajdziesz tam szereg użytecznych wskazówek i technik, włącznie z przykładem zastosowania modułu klasy.

Część IV

Zaawansowane

techniki programowania

W tej części:

Rozdział 14. „Tworzenie narzędzi dla Excela w języku VBA”

Rozdział 15. „Tabele przestawne”

Rozdział 16. „Wykresy”

Rozdział 17. „Obsługa zdarzeń”

Rozdział 18. „Interakcje z innymi aplikacjami”

Rozdział 19. „Tworzenie i wykorzystanie dodatków”

Rozdział 14.

Tworzenie narzędzi dla Excela w języku VBA

W tym rozdziale:

- Narzędzia programu Excel i ich zastosowanie
- Tworzenie narzędzi przy użyciu VBA
- Tworzenie dobrych narzędzi
- Przetwarzanie tekstu w komórkach arkusza
- Gdzie można znaleźć dodatkowe narzędzia dla Excela

Kilka słów o narzędziach dla programu Excel

Narzędzia to inaczej mówiąc ulepszenia programu, wprowadzające do niego użyteczne właściwości lub zwiększające dostępność istniejących funkcji. Narzędzie nie jest produktem końcowym, takim jak na przykład kwartalny raport. Jest to raczej program, który ułatwia tworzenie produktu końcowego. Narzędzie Excela niemal zawsze mają postać dodatków (ang. *add-ins*) wzbogacających Excela o nowe funkcje i mechanizmy.

Excel jest niewątpliwie doskonałym produktem, ale pomimo to wielu użytkowników wciąż zgłasza nowe propozycje ulepszeń. Na przykład użytkownicy, którzy często przetwarzają daty i czas, chcieliby, aby w Excelu pojawiła się możliwość wyświetlania rozwijanego kalendarza, z którego daty mogłyby być wprowadzane bezpośrednio do komórek arkusza. Jeszcze inni z radością powitaliby łatwiejsze metody eksportowania zakresu danych do osobnego pliku czy możliwość zapisywania wykresu w postaci pliku graficznego. To tylko kilka przykładów funkcji, których jeszcze nie znajdziemy w aktualnej wersji Excela, ale które możemy w stosunkowo prosty sposób dodać, tworząc odpowiednie narzędzia.

Narzędzia nie muszą być skomplikowane. Niektóre najbardziej użyteczne narzędzia są w gruncie rzeczy bardzo prostymi programami. Na przykład, czy zauważyleś, że Excel 2013 nie ma na Wstążce polecenia do przełączania wyświetlania podziałów stron na arkuszu? Jeżeli nie chcesz, aby kropkowane linie podziału były wyświetlane na arkuszu, musisz

wybrać się na wycieczkę do okna opcji programu Excel i dopiero tam znajdziesz odpowiednie opcje pozwalające na wyłączenie tych znaczników. Co gorsza, takiego polecenia nie możesz dodać ani do Wstążki, ani do paska narzędzi *Szybki dostęp*.

Poniżej przedstawiamy bardzo prostą procedurę języka VBA, która pozwala na szybkie włączanie i wyłączanie wyświetlania znaczników podziału strony.

```
Sub TogglePageBreaks()
    With ActiveSheet
        .DisplayPageBreaks = Not .DisplayPageBreaks
    End With
End Sub
```

Powyższe makro można zapisać w osobistym skoroszycie z makrami, dzięki czemu będzie zawsze dostępne. Możesz również utworzyć dodatek programu Excel, który będzie zawierał wszystkie Twoje ulubione narzędzia. W celu umożliwienia szybszego dostępu możesz przypisać poszczególne narzędzia do skrótów klawiszowych czy menu podręcznego. W razie potrzeby możesz nawet tak zmodyfikować ustawienia Excela, aby umieścić swoje ulubione narzędzia na pasku narzędzi *Szybki dostęp* lub bezpośrednio na Wstążce.

Jak się niebabem sam przekonasz, tworzenie narzędzi dla Excela to znakomity sposób na to, aby naprawdę znakomity program stał się jeszcze lepszy.

Zastosowanie języka VBA do tworzenia narzędzi

Excel 5, wydany w 1992 roku, to pierwsza wersja Excela z obsługą języka VBA. Kiedy otrzymałem do testów wersję beta Excela 5, potencjał języka VBA wywarł na mnie duże wrażenie. Był o niebo lepszy od posiadającego duże możliwości (ale bardzo nieczytelnego) języka makr XLM, dlatego postanowiłem poznać nowy język i sprawdzić jego możliwości.

Podczas nauki napisałem szereg narzędzi Excela, korzystając wyłącznie z języka VBA. Po myślealem, że nauczę się języka szybciej, jeżeli postawię sobie ambitny cel. W rezultacie powstał produkt, który nazywam *Power Utility Pak for Excel* (więcej szczegółowych informacji na temat tego pakietu znajdziesz na stronie internetowej autora).

Podczas pracy nad tym projektem dowiedziałem się o VBA kilku ciekawych rzeczy:

- Język VBA początkowo może wydawać się trudny, ale w praktyce okazuje się bardzo przystępny.
- Kluczem do opanowania programowania w języku VBA jest eksperymentowanie. Każdy projekt, którym się zajmuję, zwykle składa się z kilkunastu małych eksperymentów programistycznych, wspólnie tworzących końcowy produkt.
- Język VBA pozwala też poszerzać możliwości Excela w sposób spójny z jego interfejsem użytkownika, włączając w to funkcje arkusza i okna dialogowe, a jeżeli chcesz przekroczyć granice wyznaczane przez VBA, możesz przy użyciu kodu XML dostosować Wstążkę do własnych potrzeb.

- Za pomocą Excela można zrobić prawie wszystko. Kiedy natkniemy się na ślepu uliczkę, zwykle okazuje się, że do celu prowadzi inna ścieżka. Trzeba tylko być kreatywnym i wiedzieć, gdzie należy szukać pomocy.

Niewiele pakietów oprogramowania zawiera tak obszerny zestaw narzędzi pozwalających użytkownikowi na rozszerzanie funkcjonalności aplikacji bazowej.

Co decyduje o przydatności narzędzi?

Narzędzia Excela powinny ułatwiać pracę lub przyczyniać się do wzrostu wydajności. Jednak co sprawia, że narzędzia utworzone dla szerszego grona użytkowników są wartościowe? Poniżej utworzyłem listę elementów charakterystycznych dla dobrych narzędzi.

- **Wprowadzają do Excela coś nowego.** Może to być nowa właściwość, połączenie istniejących właściwości lub ułatwienie korzystania z istniejących właściwości.
- **Są uniwersalne.** W idealnej sytuacji narzędzie jest dobre, jeżeli spełnia swoją rolę w jak najszerzym zakresie warunków. Oczywiście napisanie narzędzia ogólnego przeznaczenia jest trudniejsze od stworzenia narzędzia przeznaczonego do wykorzystania w ścisłe określonym środowisku.
- **Są elastyczne.** Najlepsze narzędzia zawierają wiele opcji obsługujących wiele sytuacji i zdarzeń.
- **Wyglądają i działają jak wbudowane polecenia Excela.** Chociaż czasami trudno się oprzeć chęci wprowadzenia własnych elementów, innym użytkownikom łatwiej korzysta się z narzędzi, jeżeli sprawiają wrażenie polecen Excela.
- **Zapewniają pomoc użytkownikom.** Mówiąc inaczej, narzędzia powinny być wyposażone w szczegółową i łatwo dostępną dokumentację.
- **Obsługują błędy.** Użytkownik nigdy nie powinien oglądać komunikatu o błędzie VBA. Jedyne dopuszczalne komunikaty o błędach to te, które sami zdefiniowaliśmy.
- **Efekty ich działania można cofnąć.** Użytkownikom, którzy nie będą zadowoleni z działania naszych narzędzi, należy zapewnić sposób cofnięcia skutków ich działania.

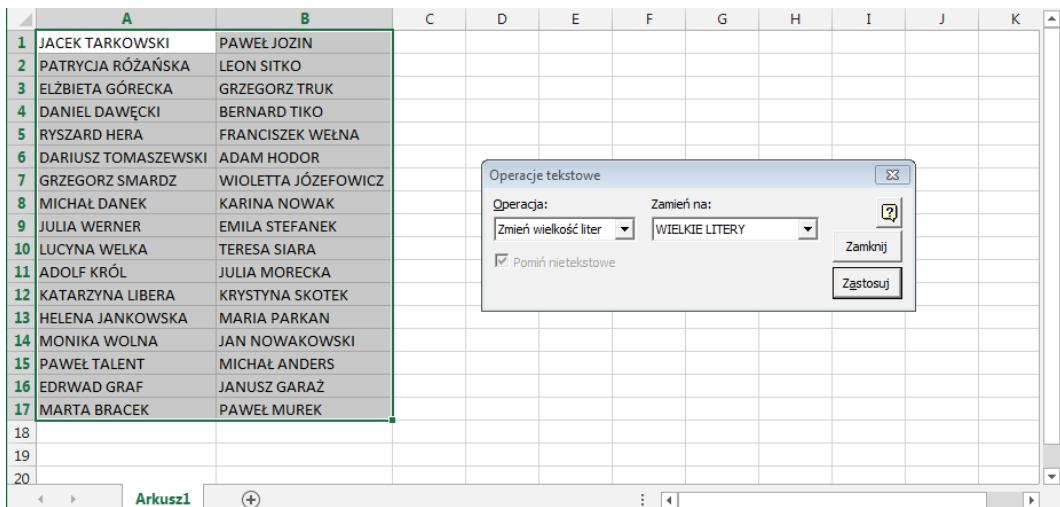
Operacje tekstowe: anatomia narzędzia

W tym podrozdziale opiszę moje autorskie narzędzie do przetwarzania tekstu, które opracowałem jakiś czas temu i z którego korzystam niemal na co dzień. Narzędzie to jest częścią dodatku *Power Utility Pak* i umożliwia użytkownikom wykonywanie działań na tekście zawartym w wybranego zakresu komórek, a w szczególności umożliwia wykonywanie następujących działań:

- Zmianę wielkości liter tekstu (wielkie litery, małe litery, wszystkie wyrazy pisane wielkimi literami, jak w zdaniu, przełącznik małe-wielkie).
- Dodawanie znaków do łańcuchów tekstu (na początku, na końcu, na określonej pozycji).

- Usuwanie znaków z łańcuchów tekstu (z początku, z końca lub z określonej pozycji).
- Usuwanie spacji z łańcuchów tekstu (wszystkich spacji lub nadmiarowych).
- Usuwanie znaków z łańcuchów tekstu (niedrukowalnych, znaków alfabetu, znaków innych niż cyfry, znaków innych niż cyfry lub znaki alfabetu).

Na rysunku 14.1 przedstawiono wygląd okna dialogowego *Operacje tekstowe*.



Rysunek 14.1. Narzędzie *Operacje tekstowe* można wykorzystać do zmiany wielkości liter zaznaczonego tekstu



W sieci

Narzędzie *Operacje tekstowe* jest dostępne na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Jest to samodzielna wersja narzędzia dostępnego w pakiecie Power Utility Pak. Plik, o nazwie *Operacje tekstowe.xlam*, jest standardowym dodatkiem programu Excel. Po zainstalowaniu umieszcza na Wstążce nowe polecenia, dostępne po wybraniu polecenia *Operacje tekstowe*, znajdującego się w grupie poleceń *Inne* na karcie *Narzędzia główne*. Projekt VBA tego narzędzia nie jest chroniony hasłem, więc możesz dokładnie zapoznać się z kodem źródłowym i dowiedzieć się, jak to narzędzie działa.

Kilka słów o programie *Operacje tekstowe*

Excel posiada wiele funkcji arkuszowych, które pozwalają na przetwarzanie łańcuchów tekstu. Na przykład, możesz zamienić małe litery w komórce na wielkie (funkcja *LITERA.WIELKIE*), połączyć łańcuchy tekstu (funkcja *ZŁĄCZ.TEKSTY*), usunąć spacje (funkcja *USUŃ.ZBĘDNE.ODSTĘPY*) i tak dalej. Jednak aby wykonać takie operacje, musisz utworzyć odpowiednie formuły, skopiować je, zamienić formuły na wartości, a następnie wkleić wartości w miejsce oryginalnego tekstu. Krótko mówiąc, przetwarzanie tekstu w Excelu nie jest szczególnie łatwym zadaniem. Czy nie byłoby wygodniej, gdyby istniały odpowiednie narzędzia umożliwiające modyfikację tekstu bez konieczności używania formuł?

Tak na marginesie, wiele pomysłów na dobre narzędzia narodziło się w sytuacji, kiedy ktoś podczas rozmowy zapytał po prostu „Czy nie byłoby wygodniej, gdyby...?”.

Określenie wymagań dla narzędzia Operacje tekstowe

Pierwszą czynnością, jaką powinieneś wykonać podczas tworzenia narzędzia, jest określenie wymagań. Poniżej znajduje się mój oryginalny plan działania, ujęty w kilkunastu punktach.

- Narzędzie będzie charakteryzowało się właściwościami, które opisano na początku niniejszego podrozdziału.
- Będzie umożliwiało użytkownikowi zażądanie wymienionych modyfikacji dla komórek tekstowych i nietekstowych.
- Będzie miało wygląd innych poleceń Excela. Mówiąc inaczej, okno dialogowe narzędzia będzie przypominać wbudowane okna dialogowe Excela.
- Będzie miało formę dodatku i będzie dostępne bezpośrednio ze Wstążki.
- Będzie działało dla bieżącego zakresu zaznaczonych komórek (z wyborem wielokrotnym włącznie). Dodatkowo będzie pozwalało użytkownikowi na zmianę zaznaczenia podczas wyświetlania okna dialogowego.
- Ostatnia wykonana operacja będzie pamiętała i wyświetli się podczas ponownego otwarcia okna dialogowego.
- Narzędzie nie będzie modyfikować komórek zawierających formuły.
- Będzie działało szybko i wydajnie. Jeżeli na przykład użytkownik zaznaczy całą kolumnę, puste komórki zostaną zignorowane.
- Będzie korzystało z okna niemodalnego, dzięki czemu użytkownik będzie mógł trzymać okno narzędzia otwarte na ekranie przez cały czas pracy z arkuszem.
- Rozmiary okna dialogowego narzędzia będą umiarkowane, tak aby nie zasłaniało zbyt dużego obszaru arkusza.
- Użytkownik będzie miał możliwość cofnięcia wykonanych zmian.
- Będzie posiadało obszerny system pomocy dla użytkownika.

Skoroszyt narzędzia Operacje tekstowe

Skoroszyt narzędzia *Operacje tekstowe* to klasyczny plik dodatku zapisany w formacie XLAM. Podczas projektowania narzędzia pracowałem ze standardowym plikiem skoroszytu w formacie XLSM, który po zakończeniu prac projektowych i testowania narzędzia zapisałem w postaci dodatku.

Skoroszyt narzędzia *Operacje tekstowe* składa się z następujących komponentów:

- **Jednego arkusza.** Każdy skoroszyt musi zawierać co najmniej jeden arkusz (wykorzystałem ten fakt i użyłem tego arkusza podczas tworzenia procedury pozwalającej na wycofanie dokonanych zmian, co zostało opisane w punkcie „Implementacja procedury Cofnij” w dalszej części rozdziału).
- **Jednego modułu VBA.** W tym module znajdują się deklaracje zmiennych publicznych i stałych, kod wyświetlający formularz *UserForm* oraz kod obsługi procedury pozwalającej na wycofanie dokonanych zmian.

- **Jednego formularza UserForm.** Formularz zawiera okno dialogowe wykorzystywane przez narzędzie. Kod, który realizuje wybrane przez użytkownika operacje przetwarzające tekst, jest przechowywany w module kodu formularza *UserForm*.



Plik dodatku zawiera również kilka dodatkowych modyfikacji, których dokonałem po to, aby ikona narzędzia była poprawnie wyświetiana na Wstążce programu Excel (patrz „Umieszczanie polecen na Wstążce” w dalszej części tego rozdziału). Niestety modyfikacja polecen na Wstążce wyłącznie przy użyciu VBA nie jest możliwa.

Jak działa narzędzie Operacje tekstowe?

Dodatek *Operacje tekstowe* zawiera odpowiedni fragment kodu RibbonX, który pozwala umieścić na Wstążce programu Excel nowego polecenia, *Operacje tekstowe*, które znajdziesz na karcie *Narzędzia główne*, w grupie *Inne*. Naciśnięcie przycisku tego polecenia powoduje wykonanie procedury *StartTextTools*, która wywołuje procedurę *ShowTextToolsDialog*.



Aby dowiedzieć się, dlaczego do poprawnej pracy narzędzia *Operacje tekstowe* potrzebne są obie procedury *StartTextTools* oraz *ShowTextToolsDialog*, zajrzyj do podrozdziału „Umieszczanie polecen na Wstążce” w dalszej części tego rozdziału.

Użytkownik określa modyfikacje do wykonania, a następnie kliką przycisk *Zastosuj*, aby je wykonać. Modyfikacje są widoczne w arkuszu, ale okno dialogowe w dalszym ciągu wyświetla się na ekranie. Każdą operację można cofnąć, a także można wykonać dodatkowe działania z tekstem. Kliknięcie przycisku *Pomoc* powoduje wyświetlenie okna pomocy, natomiast kliknięcie przycisku *Zamknij* — zamknięcie okna dialogowego. Zwróci uwagę, że jest to *niemodalne* okno dialogowe, a więc w czasie, kiedy się wyświetla, można kontynuować pracę w Excelu. W tym kontekście można powiedzieć, że okno dialogowe naszego narzędzia jest nieco podobne do paska narzędzi.

Instalowanie dodatku

Aby zainstalować wybrany dodatek, na przykład *Operacje tekstowe.xlam*, powinieneś wykonać polecenia opisane poniżej:

1. Przejdz na kartę *PLIK* i z menu wybierz polecenie *Opcje*.
2. W oknie dialogowym *Opcje* programu Excel kliknij kartę *Dodatki*.
3. Z listy rozwijanej *Zarządzaj* wybierz opcję *Dodatki programu Excel* i naciśnij przycisk *Przejdz*. Na ekranie pojawi się okno dialogowe *Dodatki*.
4. Jeżeli dodatek, który chcesz zainstalować, znajduje się już na liście dostępnych dodatków, po prostu zaznacz go.

Jeżeli nie ma go na liście, naciśnij przycisk *Przeglądaj*, a następnie odszukaj i zaznacz żądaną plik dodatku w formacie *XLAM* lub *XLA*.

5. Naciśnij przycisk *OK* i wybrany dodatek zostanie zainstalowany.

Dodatek pozostanie zainstalowany dopóty, dopóki nie usuniesz go z listy zainstalowanych dodatków.

Jeżeli lubisz posługiwać się klawiaturą, zamiast wykonywania kroków 1. – 3. możesz po prostu nacisnąć sekwencję klawiszy *lewy Alt, Q, X*, co przywoła na ekran okno dialogowe *Dodatki*. Jeżeli karta *DODATKI* jest widoczna na Wstążce, możesz również przejść na tę kartę i naciąć przycisk *Dodatki*, znajdujący się w grupie opcji *Dodatki*.

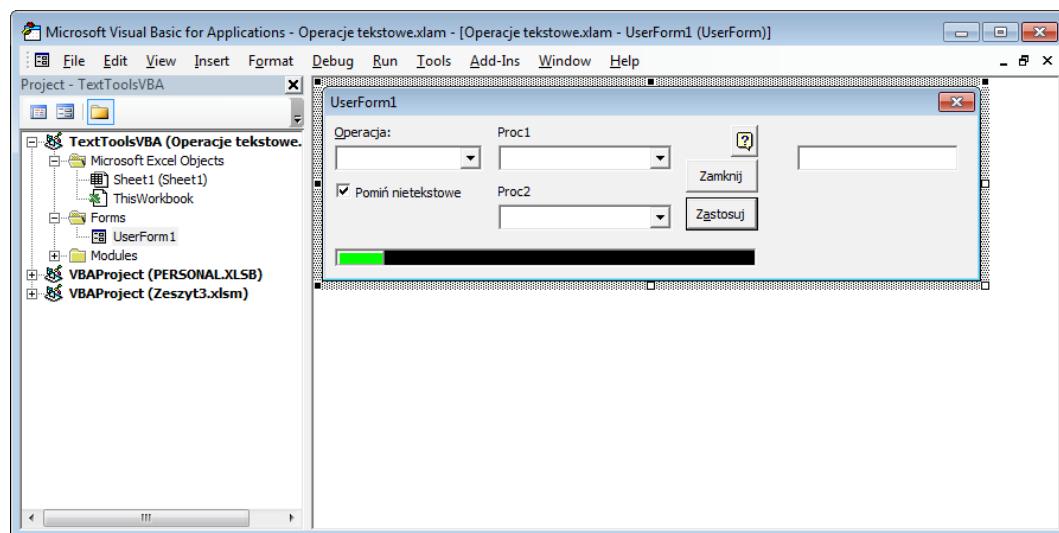


Jeżeli używasz tego narzędzia w Excelu 2013, po przełączeniu do innego skoroszytu okno dialogowe *Operacje tekstowe* nie będzie dostępne. Aby korzystać z tego narzędzia w innych skoroszytach, musisz najpierw zamknąć okno dialogowe *Operacje tekstowe*, następnie aktywować okno innego skoroszytu i ponownie przywołać na ekran okno *Operacje tekstowe*, klikając odpowiednie polecenie na Wstążce.

Formularz UserForm dla narzędzia Operacje tekstowe

Kiedy tworzę narzędzia, zazwyczaj zaczynam od zaprojektowania interfejsu użytkownika. W tym przypadku jest to okno dialogowe wyświetlane użytkownikowi. Tworzenie okna dialogowego zwykle zmusza mnie do przemyślenia całego projektu narzędzia raz jeszcze.

Na rysunku 14.2 pokazano formularz *UserForm* narzędzia *Operacje tekstowe*.



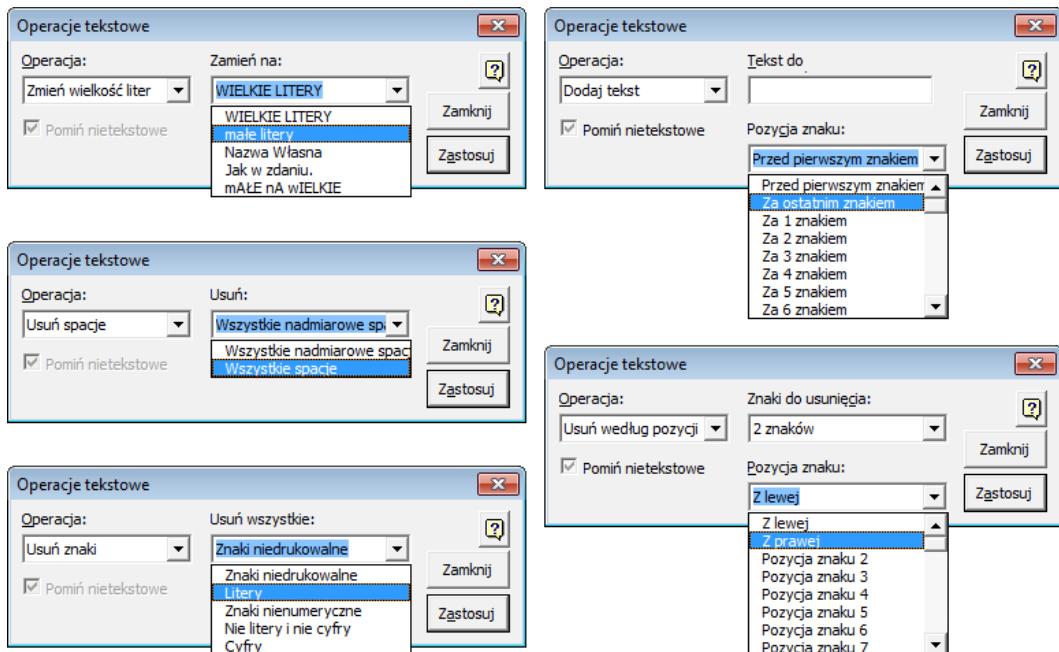
Rysunek 14.2. Formularz *UserForm* dla narzędzia *Operacje tekstowe*

Zwróć uwagę, że formanty w formularzu *UserForm* są ułożone inaczej, niż zostaną faktycznie wyświetcone. Wynika to stąd, że dla różnych opcji zostały zastosowane różne formanty, a pozycje formantów są obsługiwane dynamicznie przez kod programu. Użyte formanty zostały wymienione i opisane poniżej.

- **Pole kombi *Operacja*** — zawsze jest umieszczone z lewej strony okna. Służy do wyboru operacji, która ma być wykonana.
- **Pole kombi *Proc1*** — wykorzystywane jest w większości operacji modyfikacji tekstu do bardziej szczegółowego zdefiniowania działań.
- **Pole kombi *Proc2*** — wykorzystywane jest w dwóch operacjach modyfikacji tekstu (*Wprowadź tekst* oraz *Usuń tekst z pozycji*) do jeszcze bardziej szczegółowego zdefiniowania działań.
- **Pole wyboru *Pomiń nietekstowe*** — jest opcją właściwą dla niektórych operacji.
- **Przycisk pomocy** (ikona pytajnika) służy do wyświetlania pomocy.

- **Przycisk Zamknij** (CloseButton) służy do zamykania formularza *UserForm*.
- **Przycisk Zastosuj** (ApplyButton) służy do wykonywania wskazanych operacji dla zaznaczonego tekstu.
- **Wskaźnik postępu** zadania składa się z etykiety (formant Label) umieszczonej wewnętrz formantu Frame.
- **Pole tekstowe** jest wykorzystywane w opcji *Dodaj tekst*.

Na rysunku 14.3 pokazano wygląd formularza *UserForm* dla każdej z pięciu operacji. Zwróć uwagę na to, że konfiguracja formantów zmienia się w zależności od wybranej opcji.



Rysunek 14.3. Układ formularza *UserForm* jest różny dla różnych operacji

Moduł VBA Module1

Moduł VBA Module1 zawiera deklaracje, prostą procedurę uruchamiającą narzędzie oraz procedurę obsługującą operację wycofywaną wykonanych operacji.

Deklaracje w module VBA Module1

Poniżej wyszczególniono deklaracje umieszczone na początku modułu Module1:

```
Public Const APPNAME As String = "Operacje tekstowe"
Public Const PROGRESSTHRESHOLD = 2000
Public UserChoices(1 To 8) As Variant ' zapisuje ustawienia dla ostatnio wykonywanej operacji
Public UndoRange As Range ' w celu umożliwienia wycofania polecenia
Public UserSelection As Range ' w celu umożliwienia wycofania polecenia
```

Zadeklarowałem stałą publiczną zawierającą łańcuch tekstu przechowujący nazwę aplikacji. Ten łańcuch znaków zastosowano jako tytuł formularza *UserForm* oraz w kilku oknach komunikatów.

Stała PROGRESSTHRESHOLD określa liczbę, która decyduje o wyświetlaniu wskaźnika postępu zadania. Jeżeli określmy dla stałej wartość 2000, wskaźnik postępu zadania wyświetli się tylko wtedy, jeżeli narzędzie będzie wykonywać operacje dotyczące co najmniej 2000 komórek.

W tablicy UserChoices zapisano wartości wszystkich formantów. Te informacje są zapisywane w rejestrze Windows i odtwarzane w czasie uruchomienia narzędzia. Jest to funkcja, którą dodałem dla wygody użytkowania, ponieważ zauważylem, że wielu ludzi często wykonuje takie same operacje za każdym razem, kiedy uruchamiają to narzędzie.

W celu zapisania informacji umożliwiających cofnięcie wykonywanych operacji wykorzystano dwa dodatkowe obiekty typu Range.

Procedura ShowTextToolsDialog w module VBA Module1

Kod procedury ShowTextToolsDialog został przedstawiony poniżej:

```
Sub ShowTextToolsDialog()
    Dim InvalidContext As Boolean
    If Val(Application.Version) < 12 Then
        MsgBox "To narzędzie wymaga Excela w wersji 2007 lub nowszej.", vbCritical
        Exit Sub
    End If
    If ActiveSheet Is Nothing Then InvalidContext = True
    If TypeName(ActiveSheet) <> "Worksheet" Then InvalidContext = True
    If InvalidContext Then
        MsgBox "Zaznacz wybrany zakres komórek.", vbCritical, APPNAME
    Else
        UserForm1.Show vbModeless
    End If
End Sub
```

Procedura rozpoczyna działanie od sprawdzenia wersji programu Excel. Jeżeli wykryta zostanie wersja wcześniejsza niż 2007, użytkownik jest informowany, że to narzędzie wymaga Excela 2007 lub wersji późniejszej.

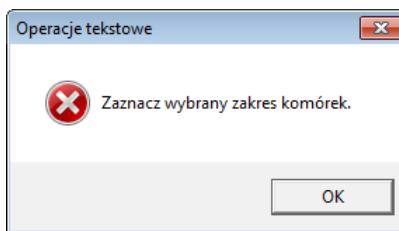


Oczywiście można tak zmodyfikować kod tego narzędzia, że będzie poprawnie pracowało z poprzednimi wersjami Excela. Jednak dla uproszczenia kodu źródłowego nasza aplikacja została napisana tak, że działa poprawnie tylko z Exceliem 2007 i nowszymi.

Jeżeli użytkownik korzysta z odpowiedniej wersji Excela, procedura ShowTextToolsDialog sprawdza aktywny arkusz i upewnia się, że jest to arkusz danych (a nie na przykład arkusz wykresu). Jeżeli któryś warunek nie zostanie spełniony, zmienna InvalidContext zostanie ustawiona na wartość True. Struktura If ... Then ... Else sprawdza wartość tej zmiennej i w zależności od niej wyświetla albo komunikat o błędzie (patrz rysunek 14.4), albo formularz *UserForm*. Zauważ, że metoda Show korzysta z argumentu vbModeless, dzięki czemu okno formularza jest wyświetlane w trybie *niemodalnym* (ang. *modeless*), stąd użytkownik może nadal pracować z Exceliem, kiedy okno jest wyświetlone na ekranie.

Rysunek 14.4.

Ten komunikat wyświetli się, jeżeli nie otwarto skoroszytu lub jeżeli aktywne okno nie jest arkuszem



Zauważ, że kod nie sprawdza, czy zaznaczono zakres komórek. Obsługę tego błędu zawarto w kodzie wykonywanym po kliknięciu przycisku *Zastosuj*.



Podczas projektowania tego narzędzia na potrzeby testowania przypisałem do procedury *ShowTextToolsDialog* skrót klawiszowy *Ctrl+Shift+T*. Zrobiłem tak, ponieważ modyfikacja Wstążki była ostatnim zadaniem na mojej liście, a ja musiałem mieć możliwość wcześniejszego testowania narzędzia. Po dodaniu przycisku na Wstążce usunąłem skrót klawiszowy.

Aby przypisać skrót klawiszowy do makra, naciśnij kombinację klawiszy *Alt+F8* (na ekranie pojawi się okno dialogowe *Makro*). W polu *Nazwa makra* wpisz *ShowTextToolsDialog* i naciśnij przycisk *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje makra*, w którym możesz przypisać do makra wybraną kombinację klawiszy.

Procedura UndoTextTools w module VBA Module1

Procedura *UndoTextTools* jest wykonywana, kiedy użytkownik naciśnie przycisk *Cofnij* (lub naciśnie kombinację klawiszy *Ctrl+Z*). Technika wycofywania poleceń zostanie omówiona w dalszej części niniejszego rozdziału w punkcie „Implementacja procedury *Cofnij*”.

Moduł formularza UserForm1

Wszystkie zasadnicze działania wykonuje kod VBA umieszczony w module formularza *UserForm1*. W tym punkcie pokrótkę opiszymy każdą procedurę umieszczoną w module. Kod przykładu jest zbyt obszerny, żeby można go było zaprezentować tutaj w całości, ale zainteresowani mogą go przejrzeć, otwierając plik *Operacje tekstowe.xlam* umieszczony na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Procedura UserForm_Initialize w module kodu formularza UserForm1

Wykonywana jest przed wyświetlением formularza *UserForm*. Jej działanie polega na określaniu rozmiarów formularza *UserForm* i pobraniu (z rejestru Windows) poprzednio wybranych wartości dla formantów. Procedura dodaje także elementy na liście pola kombi (o nazwie *ComboBoxOperation*), którego wartość decyduje o wykonywanej operacji. Użytkownik może wykonywać następujące operacje:

- Zmiana wielkości liter.
- Dodawanie tekstu.
- Usuwanie znaków według pozycji.

- Usuwanie spacji.
- Usuwanie znaków (według rodzaju).

Procedura ComboBoxOperation_Change w module kodu formularza UserForm1

Wykonywana jest za każdym razem, kiedy użytkownik wybierze opcję w polu kombi ComboBoxOperation. Zadaniem procedury jest wyświetlanie lub ukrywanie odpowiednich formantów w zależności od operacji wybranej przez użytkownika. Na przykład: jeżeli użytkownik wybierze opcję *Zmień wielkość liter*, kod procedury ukryje drugie pole kombi (o nazwie *ComboProc1*) i wypełni pierwsze pole następującymi opcjami:

- WIELKIE LITERY
- małe litery
- Nazwa Własna
- Jak w zdaniu
- MAŁE nA wIELKIE

Procedura ApplyButton_Click w module kodu formularza UserForm1

Wykonywana jest w momencie kliknięcia przycisku *Zastosuj*. Najpierw sprawdza błędy, a następnie wywołuje funkcję *CreateWorkRange* w celu sprawdzenia, czy nie ma pustych komórek w zakresie zaznaczonym do przetwarzania (więcej informacji znajduje się w kolejnym punkcie „Poprawa wydajności narzędzia Operacje tekstowe”).

Procedura *ApplyButton_Click* wywołuje również procedurę *SaveForUndo*, która zapisuje bieżące dane na wypadek, gdyby użytkownik chciał cofnąć operację (więcej informacji znajduje się w punkcie „Implementacja procedury *Cofnij*” w dalszej części niniejszego rozdziału).

Następnie wykorzystuje konstrukcję *Select Case* w celu wywołania odpowiedniej procedury wykonującej operację. Wywoływana jest jedna z następujących procedur:

- ChangeCase
- AddText
- RemoveText
- RemoveSpaces
- RemoveCharacters

Niektóre spośród tych procedur wywołują odpowiednie funkcje. Na przykład procedura *ChangeCase* może wywoływać funkcję *ToggleCase* lub *SentenceCase*.

Procedura CloseButton_Click w module kodu formularza UserForm1

Ta procedura jest wykonywana w momencie kliknięcia przycisku *Zamknij*. Jej działanie polega na zapisaniu bieżących ustawień formantów w rejestrze Windows, a następnie zamknięciu formularza *UserForm*.

Procedura HelpButton_Click w module kodu formularza UserForm1

Wykonywana jest w momencie kliknięcia przycisku *Pomoc*. Jej działanie polega po prostu na wyświetleniu pliku pomocy (którym jest standardowy plik pomocy w skompilowanym formacie HTML).

Poprawa wydajności narzędzia Operacje tekstowe

Procedury narzędzia *Operacje tekstowe* przetwarzają w pętli zaznaczony zakres komórek. Nie ma sensu przetwarzać w pętli komórek, które się nie zmieniają, na przykład pustych komórek lub takich, które zawierają formuły.

Jak wspomniałem wcześniej, procedura *ApplyButton_Click* wywołuje funkcję *CreateWorkRange*, która tworzy i zwraca obiekt Range składający się z wszystkich niepustych i niezawierających formuł komórek z zaznaczonego zakresu. Założymy na przykład, że w kolumnie A tekst umieszczono w zakresie A1:A12. Jeżeli użytkownik zaznaczy całą kolumnę, funkcja *CreateWorkRange* przekształci cały zakres kolumny na podzbior tego zakresu składający się z niepustych komórek (tzn. zakres A:A zostanie przekształcony na zakres A1:A12). Ponieważ puste komórki i formuły nie będą przetwarzane, kod stanie się o wiele wydajniejszy.

Funkcja *CreateWorkRange* pobiera dwa argumenty:

- *Rng* — obiekt typu Range reprezentujący zakres wybrany przez użytkownika.
- *TextOnly* — argument typu Boolean. Jeżeli ma wartość True, funkcja zwróci tylko komórki tekstowe. W innym przypadku będą zwrócone wszystkie niepuste komórki.

```
Private Function CreateWorkRange(Rng, TextOnly)
    ' Tworzy i zwraca obiekt Range
    Set CreateWorkRange = Nothing
    ' Pojedyncza komórka zawierająca formułę
    If Rng.Count = 1 And Rng.HasFormula Then
        Set CreateWorkRange = Nothing
        Exit Function
    End If
    ' Pojedyncza komórkę lub pojedyncza scalona komórkę
    If Rng.Count = 1 Or Rng.MergeCells = True Then
        If TextOnly Then
            If Not IsNumeric(Rng(1).Value) Then
                Set CreateWorkRange = Rng
                Exit Function
            Else
                Set CreateWorkRange = Nothing
                Exit Function
            End If
        Else
    End If
```

```
If Not IsEmpty(Rng(1)) Then
    Set CreateWorkRange = Rng
    Exit Function
End If
End If
On Error Resume Next
Set Rng = Intersect(Rng, Rng.Parent.UsedRange)
If TextOnly = True Then
    Set CreateWorkRange = Rng.SpecialCells(xlConstants, xlTextValues)
    If Err <> 0 Then
        Set CreateWorkRange = Nothing
        On Error GoTo 0
        Exit Function
    End If
Else
    Set CreateWorkRange = Rng.SpecialCells _
        (xlConstants, xlTextValues + xlNumbers)
    If Err <> 0 Then
        Set CreateWorkRange = Nothing
        On Error GoTo 0
        Exit Function
    End If
End If
End Function
```



W funkcji CreateWorkRange często wykorzystywana jest właściwość `SpecialCells`. Aby dowiedzieć się więcej na temat tej właściwości, zarejestruj makro, modyfikując ustawienia w oknie dialogowym *Przechodzenie do — specjalnie* Excela. Możesz je wyświetlić, wciskając klawisz *F5*, a następnie klikając przycisk *Specjalnie* w oknie dialogowym *Przechodzenie do*.

Ważne jest, aby prawidłowo zrozumieć sposób działania okna dialogowego *Przechodzenie do — specjalnie*. Zazwyczaj okno to operuje na zaznaczonym zakresie. Jeżeli na przykład zostanie zaznaczona cała kolumna, wynikiem będzie podzbiór tej kolumny. Jeżeli jednak zaznaczono pojedynczą komórkę, operacje będą dotyczyć całego arkusza. Z tego powodu funkcja `CreateWorkRange` sprawdza liczbę komórek w zakresie przekazanym do niej w postaci argumentu.

Zapisywanie ustawień narzędzia Operacje tekstowe

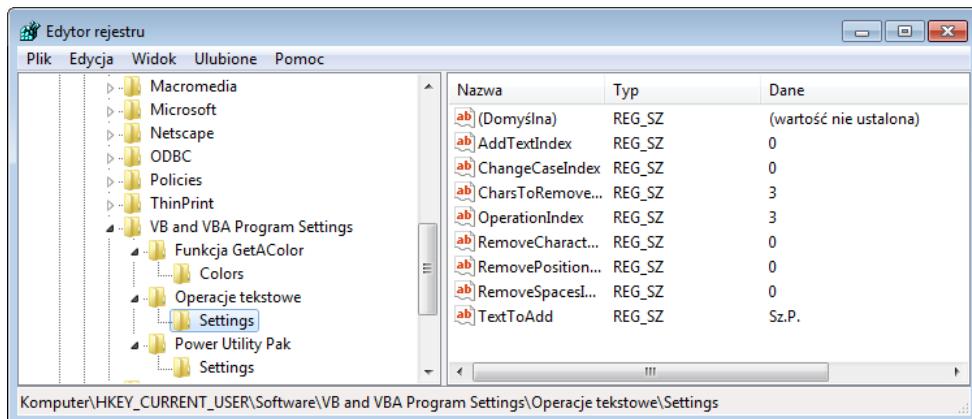
Narzędzie *Operacje tekstowe* ma bardzo przydatną właściwość: pamięta ostatnio wykorzystywane ustawienia. A przecież większość użytkowników, wywołując narzędzie, często korzysta z tego samego zestawu opcji.

Jak wspomniałem, ostatnie ustawienia są zapisywane w rejestrze Windows. Kiedy użytkownik kliknie przycisk *Zamknij*, wywoływana jest funkcja VBA `SaveSetting` zapisująca wartości wszystkich formantów. Podczas uruchamiania narzędzia *Operacje tekstowe* wywoływana jest funkcja `GetSetting` pobierająca te wartości i odpowiednio ustawiająca poszczególne formanty.

W rejestrze Windows ustawienia są zapisane w następującej lokalizacji:

HKEY_CURRENT_USER\Software\VB and VBA Program Settings\Text Tool Utility\Settings

Na rysunku 14.5 pokazano te ustawienia wyświetlane w edytorze rejestru Windows (*regedit.exe*).



Rysunek 14.5. Za pomocą edytora rejestru można sprawdzić ustawienia zapisane w rejestrze

Jeżeli przyjrzyisz się dokładniej kodowi VBA narzędziu *Operacje tekstowe*, z pewnością zauważysz, że do zapisania ustawień zastosowano ósmioelementową tablicę UserChoices. Zamiast tego do zapisania poszczególnych ustawień można by użyć osobnych zmiennych, ale dzięki zastosowaniu tablicy kod jest nieco prostszy.

Fragment kodu przedstawiony poniżej odczytuje ustawienia z rejestrów systemu Windows i zapisuje je w tablicy UserChoices:

```
' Pobierz ustawienia z rejestrów
UserChoices(1) = GetSetting(APPNAME, "Settings", "OperationIndex", 0)
UserChoices(2) = GetSetting(APPNAME, "Settings", "ChangeCaseIndex", 0)
UserChoices(3) = GetSetting(APPNAME, "Settings", "TextToAdd", "")
UserChoices(4) = GetSetting(APPNAME, "Settings", "AddTextIndex", 0)
UserChoices(5) = GetSetting(APPNAME, "Settings", "CharsToRemoveIndex", 0)
UserChoices(6) = GetSetting(APPNAME, "Settings", "RemovePositionIndex", 0)
UserChoices(7) = GetSetting(APPNAME, "Settings", "RemoveSpacesIndex", 0)
UserChoices(8) = GetSetting(APPNAME, "Settings", "RemoveCharactersIndex", 0)
cbSkipNonText.Value = GetSetting(APPNAME, "cbSkipNonText", 0)
```

Kolejny fragment kodu jest wykonywany podczas zamknięcia okna dialogowego. Poszczególne polecenia pobierają ustawienia z tablicy UserChoices i zapisują je w odpowiednich kluczach rejestrów systemu Windows.

```
' Zapisz ustawienia w rejestrach
SaveSetting APPNAME, "Settings", "OperationIndex", UserChoices(1)
SaveSetting APPNAME, "Settings", "ChangeCaseIndex", UserChoices(2)
SaveSetting APPNAME, "Settings", "TextToAdd", UserChoices(3)
SaveSetting APPNAME, "Settings", "AddTextIndex", UserChoices(4)
SaveSetting APPNAME, "Settings", "CharsToRemoveIndex", UserChoices(5)
SaveSetting APPNAME, "Settings", "RemovePositionIndex", UserChoices(6)
SaveSetting APPNAME, "Settings", "RemoveSpacesIndex", UserChoices(7)
SaveSetting APPNAME, "Settings", "RemoveCharactersIndex", UserChoices(8)
SaveSetting APPNAME, "Settings", "cbSkipNonText", cbSkipNonText.Value * -1
```

Implementacja procedury Cofnij

Niestety Excel nie udostępnia bezpośredniego sposobu na wycofywanie wykonanych poleceń z poziomu kodu VBA. Wycofanie zmian dokonanych przez makro VBA jest co prawda możliwe, ale realizacja takiej funkcji wymaga całkiem sporego nakładu pracy. Co więcej, w odróżnieniu od mechanizmu wycofywania poleceń samego Excela, technika wycofywania operacji w narzędziu *Operacje tekstowe* jest tylko jednopoziomowa. Mówiąc inaczej, użytkownik może wycofać tylko ostatnio wykonaną operację. Więcej informacji na temat zastosowania mechanizmu wycofywania poleceń w aplikacjach można znaleźć w ramce „Wycofywanie efektów działania procedur VBA”.

Narzędzie *Operacje tekstowe* zapisuje oryginalne dane w arkuszu. Jeżeli użytkownik zechce cofnąć operacje, dane tam zapisane zostaną z powrotem skopiowane do skoroszytu użytkownika.

Przypomnijmy sobie dwie zmienne publiczne zadeklarowane w module VBA *Module1*, których zadaniem była obsługa wycofywania poleceń:

```
Public UndoRange As Range  
Public UserSelection As Range
```

Przed wykonaniem modyfikacji danych procedura *ApplyButton_Click* wywołuje procedurę *SaveForUndo*. Procedura rozpoczyna się od wykonania trzech instrukcji:

```
Set UserSelection = Selection  
Set UndoRange = WorkRange  
ThisWorkbook.Sheets(1).UsedRange.Clear
```

Zmienna obiektowa *UserSelection* zapisuje aktualne zaznaczenie dokonane przez użytkownika, tak aby mogło być odtworzone po wycofaniu operacji. Obiekt *WorkRange* to obiekt typu *Range*, który jest zwracany przez funkcję *CreateWorkRange* i składa się z komórek zaznaczonego zakresu, które nie są puste i nie zawierają formuł. Trzecie polecenie usuwa z arkusza zapisane w nim poprzednio dane.

Następnie wykonywana jest pętla przedstawiona poniżej:

```
For Each RngArea In WorkRange.Areas  
    ThisWorkbook.Sheets(1).Range  
        (RngArea.Address).Formula = RngArea.Formula  
    Next RngArea
```

W pętli przetwarzane są poszczególne obszary zakresu *WorkRange*, które następnie są zapisywane jako dane w arkuszu. Jeżeli zakres *WorkRange* składa się z ciągłego zakresu komórek, arkusz będzie zawierać tylko jeden obszar.

Po wykonaniu określonej operacji wykorzystywana jest metoda *OnUndo* określająca działania, które należy wykonać w razie wybrania operacji *Cofnij*. Przykładowo po wykonaniu operacji zmiany wielkości liter wykonywana jest następująca instrukcja:

```
Application.OnUndo "Cofnij zmianę wielkości liter", "UndoTextTools"
```

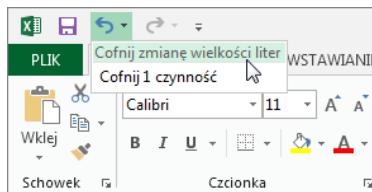
Po jej wykonaniu w menu wycofywania poleceń Excela pojawi się pozycja *Cofnij zmianę wielkości liter* (patrz rysunek 14.6). Jeżeli użytkownik wybierze to polecenie, zostanie wykonana procedura pokazana poniżej:

```

Private Sub UndoTextTools()
    ' Cofnięcie ostatnio wykonanej operacji
    Dim a As Range
    On Error GoTo ErrHandler
    Application.ScreenUpdating = False
    With UserSelection
        .Parent.Parent.Activate
        .Parent.Activate
        .Select
    End With
    For Each a In UndoRange.Areas
        a.Formula = ThisWorkbook.Sheets(1).Range(a.Address).Formula
    Next a
    Application.ScreenUpdating = True
    On Error GoTo 0
    Exit Sub
ErrHandler:
    Application.ScreenUpdating = True
    MsgBox "Nie można cofnąć operacji", vbInformation, APPNAME
    On Error GoTo 0
End Sub

```

Rysunek 14.6.
Narzędzie Operacje tekstowe pozwala na wycofanie tylko ostatnio wykonanej operacji



Wycofywanie efektów działania procedur VBA

Użytkownicy komputerów przyzwyczaili się do możliwości cofania wykonanych operacji. Prawie wszystkie działania wykonywane w Excelu można cofnąć. Co więcej, w począwszy od Excela 2007 Microsoft zwiększył liczbę poziomów wycofania operacji z 16 do 100.

Programując w VBA, zastanawiamy się, czy jest możliwe cofanie efektów procedur. Chociaż odpowiedź brzmi: *Tak*, dokładniejsza odpowiedź powinna brzmieć: *To nie zawsze jest łatwe*.

Cofanie efektów procedur VBA nie jest realizowane automatycznie. Aby było możliwe, procedura musi zapisywać poprzedni stan, który można odtworzyć, kiedy użytkownik wybierze polecenie *Cofnij* (którego przycisk znajduje się na pasku narzędzi *Szybki dostęp*). Sposób zapisania stanu zależy od działań wykonywanych przez procedurę. W przypadkach ekstremalnych konieczne jest zapisanie całego arkusza. Jeżeli z kolei procedura zmodyfikuje zakres, wystarczy zapisać zawartość tego zakresu.

Pamiętaj również, że wykonanie procedury VBA typu Sub powoduje wyczyszczenie stosu Excela, wykorzystywanego przez jego mechanizm wycofywania poleceń. Innymi słowy, po uruchomieniu makra tracisz możliwość wycofania wykonanych wcześniej operacji.

Obiekt Application zawiera metodę *OnUndo*. Umożliwia ona programistę określenie tekstu, który ma się pojawić w menu wycofania, oraz procedury, która ma być wykonana w przypadku wybrania polecenia *Cofnij*. Na przykład poniższa instrukcja spowoduje, że w podręcznym menu wycofania pojawi się pozycja *Cofnij efekty mojego makro*. Jeżeli użytkownik wybierze to polecenie, zostanie wykonana procedura *UndoMyMacro*:

```
Application.OnUndo "Cofnij efekty mojego makro", "UndoMyMacro"
```

Procedura `UndoTextTools` najpierw sprawdza, czy jest aktywny odpowiedni skoroszyt i arkusz, a potem wybiera zakres, który poprzednio wybrał użytkownik. Następnie w pętli przetwarza wszystkie obszary zapisanych danych, dostępne dzięki zmiennej publicznej `UndoRange`. Dane są umieszczane w swojej pierwotnej lokalizacji, co powoduje zastąpienie zmodyfikowanych danych ich oryginalnymi wartościami.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt *Prosty przykład procedury Cofnij.xlsxm*, zawierający prosty przykład uaktywnienia polecenia *Cofnij* po wykonaniu procedury VBA. Procedura w tym przykładzie zamiast w arkuszu zapisuje dane w tablicy. Tablica składa się z elementów o niestandardowym typie danych, które zawierają wartość i adres poszczególnych komórek.

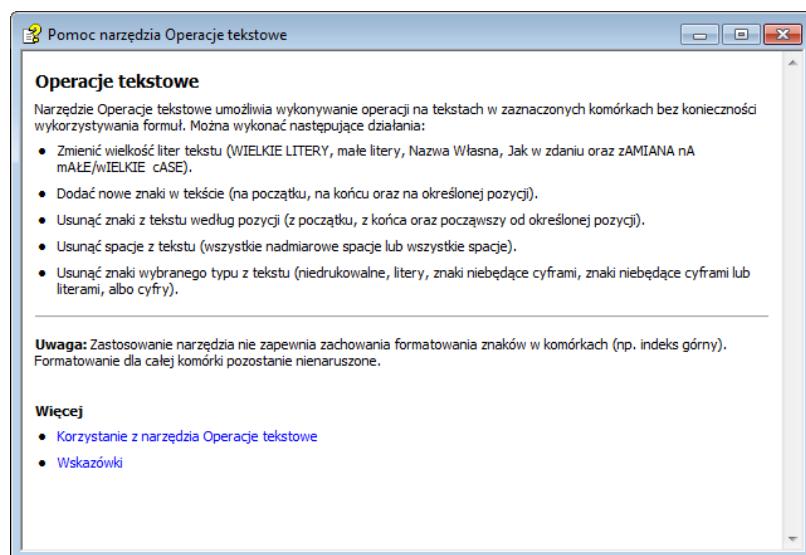
Wyświetlanie pliku pomocy

Na potrzeby narzędzia *Operacje tekstowe* utworzyłem prosty plik pomocy, zapisany w postaci skompilowanego pliku HTML o nazwie *OperacjeTekstowe.chm*. Naciśnięcie przycisku *HelpButton* na formularzu *UserForm* powoduje wykonanie następującej procedury:

```
Private Sub HelpButton_Click()
    Application.Help (ThisWorkbook.Path & "\" & "OperacjeTekstowe.chm")
End Sub
```

Na rysunku 14.7 przedstawiono wygląd jednego z ekranów okna pomocy.

Rysunek 14.7.
Wygląd jednego
z ekranów okna
pomocy



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz wszystkie pliki źródłowe, których użyłem do utworzenia skompilowanego pliku pomocy. Pliki są zlokalizowane w katalogu `\helpsource`. Jeżeli nie miałeś wcześniej zbyt dużych doświadczeń ze skompilowanymi plikami pomocy w formacie HTML, więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 22.

Umieszczanie poleceń na Wstążce

Ostatnim zadaniem, jakie wykonałem podczas tworzenia tego narzędzia, było zapewnienie wygodnego sposobu uruchamiania go przez użytkownika. Przed pojawiением się Excela 2007, wstawienie nowego polecenia do menu lub utworzenie nowego przycisku na pasku narzędzi było względnie prostym rozwiązaniem. Niestety wraz z pojawiением się Wstążki realizacja tego — kiedyś bardzo prostego — zadania nieoczekiwane znacznie się skomplikowała.

Do utworzenia kodu RibbonX, którego zadaniem będzie utworzenie nowej grupy na Wstążce i umieszczenie w niej ikony nowego polecenia użyłem edytora *Custom UI Editor for Microsoft Office*. Edytor ten nie jest częścią pakietu Microsoft Office, ale możesz go znaleźć i pobrać z sieci Internet.

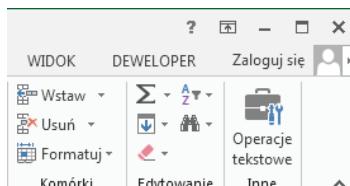


Więcej szczegółowych informacji na temat pracy ze Wstążką oraz edytorem *Custom UI* znajdziesz w rozdziale 20.

Na rysunku 14.8 przedstawiono wygląd Wstążki, na której na karcie *Narzędzia główne* została umieszczona nowa grupa poleczeń o nazwie *Inne*. W grupie tej znajduje się jeden przycisk, po naciśnięciu którego wykonywana jest następująca procedura:

```
Sub StartTextTools(control As IRibbonControl)
    Call ShowTextToolsDialog
End Sub
```

Rysunek 14.8.
Fragment karty
Narzędzia główne,
na której została
umieszczona nowa
grupa poleczeń



Na rysunku 14.9 przedstawiono kod RibbonX wyświetlony w programie *Custom UI Editor*.

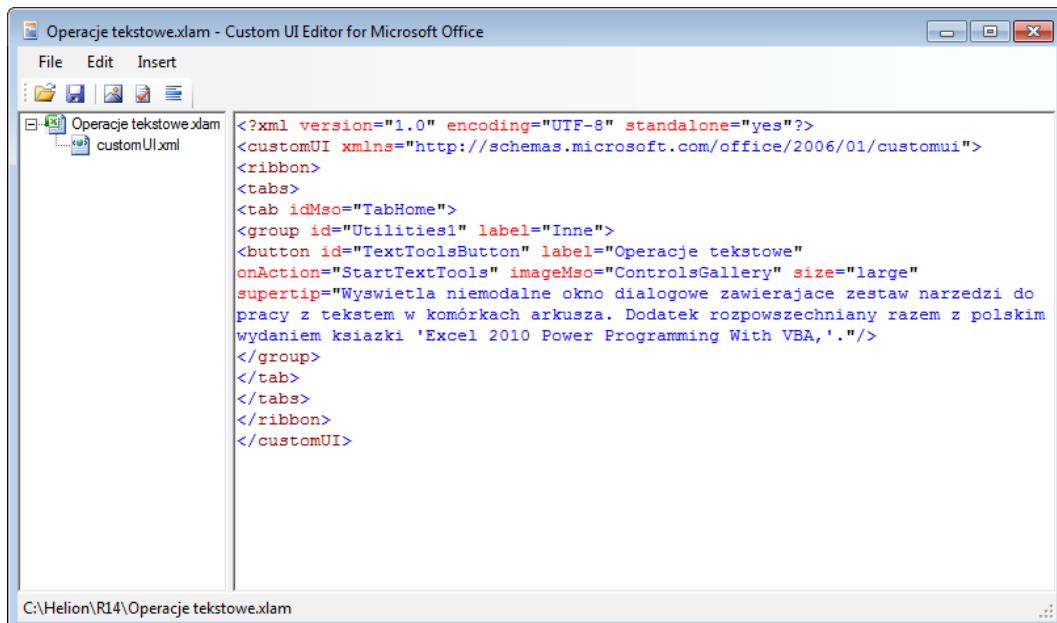


Kiedy tworzysz własne rozszerzenia Wstążki, są one widoczne tylko wtedy, kiedy powiązany z nimi skoroszyt Excela zostanie otwarty i jest aktywny. Na szczęście od tej reguły jest jeden wyjątek. Jeżeli modyfikacje Wstążki zostaną zapisane w pliku dodatku XLAM (tak jak to zrobiliśmy w tym przykładzie), będą widoczne dopóty, dopóki powiązany z nimi dodatek jest załadowany, niezależnie od tego, czy w tym czasie będą otwarte jakieś inne skoroszyty.

Ocena realizacji projektu

W poprzednich punktach opisałem wszystkie komponenty narzędzia *Operacje tekstowe*. W tym momencie warto przyjrzeć się pierwotnym wymaganiom projektowym, aby sprawdzić, czy zostały spełnione. Poniżej wyszczególniono założone cele wraz z komentarzami.

- **Narzędzie będzie posiadało właściwości opisane na początku niniejszego podrozdziału** — spełniony.



Rysunek 14.9. Użycie edytora Custom UI pozwala na uruchomienie narzędzia bezpośrednio ze Wstążki

- **Będzie umożliwiano użytkownikowi wykonanie żądanego modyfikacji dla komórek tekstowych i nietekstowych** — spełniony.
- **Będzie miało wygląd innych poleceń Excela. Mówiąc inaczej, okno dialogowe narzędzia będzie przypominać wbudowane okna dialogowe Excela.** Jak wspomniałem wcześniej, narzędzie *Operacje tekstowe* nieco różni się od okien dialogowych Excela, ponieważ zawiera przycisk *Zastosuj* zamiast przycisku *OK*. Ponadto w odróżnieniu od większości wbudowanych okien dialogowych Excela okno dialogowe narzędzia *Operacje tekstowe* jest niemodalne. Biorąc pod uwagę wygodę korzystania, te różnice można uznać za dopuszczalne.
- **Będzie miało formę dodatku dostępnego ze Wstążki** — spełniony.
- **Będzie działało dla bieżącego zakresu zaznaczonych komórek (włącznie z wyborem wielokrotnym). Dodatkowo będzie możliwa zmiana zaznaczenia podczas wyświetlania okna dialogowego** — spełniony. Ponieważ okno dialogowe nie trzeba zamykać, nie ma potrzeby wykorzystywania formantu *RefEdit*.
- **Ostatnia wykonana operacja będzie pamiętała i wyświetli się podczas ponownego otwarcia okna dialogowego** — spełniony (dzięki wykorzystaniu rejestru Windows).
- **Narzędzie nie będzie modyfikować komórek zawierających formuły** — spełniony.
- **Będzie działało szybko i wydajnie. Jeżeli na przykład użytkownik zaznaczy całą kolumnę, puste komórki w kolumnie zostaną zignorowane** — spełniony.

- Będzie używalo niemodalnego okna dialogowego, dzięki czemu będzie mogło być wyświetlane na ekranie przez cały czas pracy z Exceliem — spełniony.
- Okno narzędzia będzie miało umiarkowane rozmiary tak, aby nie zajmowało zbyt wiele miejsca na ekranie — spełniony.
- Użytkownik będzie miał możliwość cofnięcia wykonanych zmian — spełniony.
- Będzie dostępna obszerna pomoc — spełniony.

Działanie narzędzia Operacje tekstowe

Osoby, które nie do końca rozumieją, jak działa narzędzie, zachęcam do załadowania dodatku i wykorzystania debugera do wykonania kodu krok po kroku. Warto wypróbować narzędzie dla różnych zakresów komórek, z całym arkuszem włącznie. Niezależnie od wybranego zakresu przetworzone będą tylko właściwe komórki; puste zostaną całkowicie zignorowane. Dla arkusza zawierającego tylko jedną komórkę tekstową narzędzie będzie działać równie szybko w przypadku zaznaczenia tylko tej komórki, jak i w przypadku zaznaczenia całego arkusza.

Jeżeli przekształcimy dodatek na standardowy skoroszyt, będziemy mogli zobaczyć sposób zapisania danych w arkuszu dla potrzeb operacji *Cofnij*. Aby przekształcić dodatek na skoroszyt, należy kliknąć dwukrotnie moduł ThisWorkbook. Następnie należy wcisnąć F4 w celu wyświetlenia okna *Properties* i ustawić wartość właściwości IsAddin na False.

Dodatkowe informacje na temat narzędzi Excela

Jeżeli jesteś zainteresowany tworzeniem narzędzi Excela, zachęcam do pobrania wersji próbnej mojego dodatku *Power Utility Pak*. Pakiet PUP zawiera kilkadziesiąt narzędzi (a także wiele nowych funkcji arkuszowych) rozszerzających funkcjonalność programu Excel.

W sieci Internet oprócz pakietu *Power Utility Pak* dostępnych jest wiele innych, interesujących narzędzi, które możesz pobrać i których możesz używać we własnych projektach.

Rozdział 15.

Tabele przestawne

W tym rozdziale:

- Tworzenie tabel przestawnych przy użyciu VBA
- Przykłady procedur VBA tworzących tabele przestawne
- Zastosowanie języka VBA do tworzenia tabel arkusza na bazie tabel podsumowań

Przykład prostej tabeli przestawnej

Tabele przestawne w Excelu to najbardziej efektywne i spektakularne narzędzie do przetwarzania danych. Po raz pierwszy tabele przestawne pojawiły się w Excelu 5 i są konsekwentnie ulepszane w każdej kolejnej, nowej wersji Excela. Ten rozdział nie jest wprowadzeniem w tematykę tabel przestawnych — po prostu zakładamy, że znasz już ten mechanizm i związaną z nim terminologię oraz potrafisz ręcznie tworzyć i modyfikować tabele przestawne.

Jak wiadomo, utworzenie tabeli przestawnej na podstawie danych z bazy danych lub listy umożliwia wykonywanie podsumowań danych w sposób, który bez ich zastosowania nie byłby możliwy. Co więcej, jest to sposób niezwykle szybki i nie wymagający tworzenia formuł. Tabele przestawne możesz również tworzyć i modyfikować za pomocą odpowiednich procedur języka VBA.

W tym podrozdziale przedstawimy prosty przykład zastosowania języka VBA do tworzenia tabel przestawnych.

Na rysunku 15.1 przedstawiono prosty arkusz danych, zawierający cztery pola: RepHand1, Region, Miesiąc i Sprzedaż. Każdy rekord opisuje wielkość sprzedaży danego handlowca w określonym miesiącu.



Skoroszyt z tym przykładem (*Prosta tabela przestawna.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Rysunek 15.1.

Taka tabela jest dobrym punktem wyjścia do utworzenia tabeli przestawnej

	A	B	C	D	E
1	RepHandl	Region	Miesiąc	Sprzedaż	
2	Andrzej	północny	Sty	33 488	
3	Andrzej	północny	Lut	47 008	
4	Andrzej	północny	Mar	32 128	
5	Bogdan	północny	Sty	34 736	
6	Bogdan	północny	Lut	92 872	
7	Bogdan	północny	Mar	76 128	
8	Czesław	południowy	Sty	41 536	
9	Czesław	południowy	Lut	23 192	
10	Czesław	południowy	Mar	21 736	
11	Daniel	południowy	Sty	44 834	
12	Daniel	południowy	Lut	32 002	
13	Daniel	południowy	Mar	23 932	
14					
15					
16					

Tworzenie tabel przestawnych

Na rysunku 15.2 pokazano tabelę przestawną utworzoną na podstawie danych z arkusza. Na rysunku widać również panel *Lista pól tabeli przestawnej*. Tabela przestawna tworzy podsumowania sprzedaży według reprezentantów handlowych oraz miesięcy i składa się z następujących pól:

- Region — pole filtra strony w tabeli przestawnej;
- RepHandl — pole wiersza w tabeli przestawnej;
- Miesiąc — pole kolumny w tabeli przestawnej;
- Sprzedaż — pole danych w tabeli przestawnej, w którym wykorzystano funkcję SUMA.

	A	B	C	D	E	F	G	H	I	J	K
1	Region	(Wszystko)									
2											
3	Suma z Sprzedaż										
4		Sty	Lut	Mar	Suma końcowa						
5	Andrzej	33488	47008	32128	112624						
6	Bogdan	34736	92872	76128	203736						
7	Czesław	41536	23192	21736	86464						
8	Daniel	44834	32002	23932	100768						
9	Suma końcowa	154594	195074	153924	503592						
10											
11											
12											
13											
14											
15											
16											
17											
18											
19											
20											
21											
22											

Rysunek 15.2. Tabela przestawna utworzona na podstawie danych zamieszczonych na rysunku 15.1

Zanim utworzyłem tę tabelę przestawną, włączyłem rejestrator makr. Jako miejsce do celowe tabeli wybrałem nowy arkusz. Wygenerowany kod zaprezentowano poniżej:

```
Sub CreatePivotTable()
    Sheets.Add
    ActiveWorkbook.PivotCaches.Create _
        (SourceType:=xlDatabase, _
        SourceData:="Arkusz1!R1C1:R13C4", _
        Version:=xlPivotTableVersion15).CreatePivotTable _
        TableDestination:="Arkusz2!R3C1", _
        TableName:="TabelaPrzestawnal", _
        DefaultVersion:=xlPivotTableVersion15)
    Sheets("Arkusz2").Select
    Cells(3, 1).Select
    With ActiveSheet.PivotTables("TabelaPrzestawnal").PivotFields("Region")
        .Orientation = xlPageField
        .Position = 1
    End With
    With ActiveSheet.PivotTables ("TabelaPrzestawnal").PivotFields("RepHand1")
        .Orientation = xlRowField
        .Position = 1
    End With
    With ActiveSheet.PivotTables("TabelaPrzestawnal").PivotFields("Miesiąc")
        .Orientation = xlColumnField
        .Position = 1
    End With
    ActiveSheet.PivotTables("TabelaPrzestawnal").AddDataField
        ActiveSheet.PivotTables("TabelaPrzestawnal").PivotFields("Sprzedaż"), _
        "Suma z Sprzedaż", xlSum
End Sub
```

Jeżeli spróbujesz uruchomić to makro, najprawdopodobniej próba taka zakończy się wygenerowaniem błędu. Kiedy przeanalizujesz kod procedury, zobaczyysz, że rejestrator makr jako lokalizacji tabeli przestawnej użył w kodzie bezpośredniego odwołania do nazwy arkusza (Arkusz2). Jeżeli arkusz o takiej nazwie już istnieje (lub zostanie dodany nowy arkusz o innej nazwie), działanie makra zakończy się błędem. Dodatkowo, rejestrator makr „na sztywno” koduje nazwę tabeli przestawnej. Jeżeli w arkuszu utworzyłeś wcześniej jakieś inne tabele przestawne, to z pewnością kolejna tabela nie będzie nosiła nazwy TabelaPrzestawnal.

Z drugiej strony jednak, nawet jeżeli zarejestrowane makro nie będzie działało, to nie jest całkowicie nieprzydatne. Wygenerowany kod daje dobry wgląd w zasady pisania procedur VBA tworzących tabele przestawne.

Analiza zarejestrowanego kodu tworzenia tabeli przestawnej

Kod procedury VBA tworzącej tabelę przestawną może się wydawać niejasny. Aby zarejestrowane makro nabralo sensu, trzeba poznać kilka najważniejszych obiektów, których pełne opisy znajdziesz w pomocy systemowej:

- PivotCaches — kolekcja obiektów PivotCache obiektu Workbook (dane wykorzystywane przez tabelę przestawną są przechowywane w buforze tabeli przestawnej [ang. pivot cache]).

Dane, na podstawie których można utworzyć tabelę przestawną

Tabela przestawną wymaga uporządkowania danych wejściowych w postaci prostokątnego obszaru arkusza, tworzącego coś w rodzaju bazy danych. Dane mogą być przechowywane w wybranym zakresie komórek arkusza (którym może być zarówno tabela, jak i zwykły zakres komórek) lub w zewnętrznym pliku bazy danych. Co prawda Excel potrafi wygenerować tabelę przestawną na podstawie danych z niemal dowolnych baz danych, ale w nie wszystkich przypadkach to się opłaca.

Ogólnie rzecz biorąc, pola w tabeli bazy danych można podzielić na dwa rodzaje:

- **Pola danych** — Zawierają wartości lub inne dane, które będą podsumowywane. Przykładowo, jeżeli mamy bazę danych opisującą konta bankowe, to pole Saldo będzie właśnie polem danych.
- **Pola kategorii** — Zawierają informacje opisujące dane. Na przykładzie wspomnianej wcześniej bazy danych opisującej konta bankowe, można zobaczyć, że pola Data, TypKonta, OtwartePrzez, OddziałBanku oraz Klient są polami kategorii, ponieważ opisują dane przechowywane w polu Saldo.

Tabela bazy danych, która jest odpowiednia do utworzenia na jej podstawie tabeli przestawnej, jest nazywana *tabelą znormalizowaną*. Innymi słowy, każdy rekord tabeli (lub każdy wiersz tabeli) zawiera informacje opisujące dane.

Pojedyncza tabela bazy danych może posiadać dowolną liczbę pól danych oraz pól kategorii. Kiedy tworzysz tabelę przestawną, zazwyczaj chcesz dokonać podsumowania jednego lub więcej pól danych. Wartości w polach kategorii pojawiają się w tabeli przestawnej jako wiersze, kolumny lub filtry danych.

Jeżeli jeszcze masz jakieś wątpliwości, to na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>) znajdziesz skoroszyt o nazwie *Dane znormalizowane.xlsx*. W skoroszycie znajdziesz przykład zakresu danych przed i po normalizacji. Na podstawie znormalizowanej wersji danych możesz utworzyć tabelę przestawną.

- PivotTables — kolekcja obiektów PivotTable obiektu Worksheet.
- PivotFields — kolekcja pól obiektu PivotTable.
- PivotItems — kolekcja elementów danych dla pola określonej kategorii.
- CreatePivotTable — metoda obiektu PivotCache tworząca tabelę przestawną na podstawie danych w buforze tabeli przestawnej.

Optymalizacja wygenerowanego kodu tworzącego tabelę przestawną

Podobnie jak w przypadku większości zarejestrowanych makr zaprezentowany wcześniej przykład nie jest tak efektywny, jak mógłby być, a co więcej, jak już wcześniej wspominaliśmy, próba jego wykonania może zakończyć się błędem. Naszą przykładową procedurę można jednak uprościć, uczynić bardziej zrozumiałą i jednocześnie dodać odpowiedni kod zapobiegający możliwości wystąpienia opisywanego wcześniej błędu. Poniżej przedstawiono ręcznie zoptymalizowany kod procedury, która tworzy dokładnie taką samą tabelę przestawną jak w poprzednim przykładzie.

```

Sub CreatePivotTable()
    Dim PTCache As PivotCache
    Dim PT As PivotTable

    ' Tworzenie bufora
    Set PTCache = ActiveWorkbook.PivotCaches.Create( _
        SourceType:=xlDatabase, _
        SourceData:=Range("A1").CurrentRegion)

    ' Dodawanie nowego arkusza dla tabeli przestawnej
    Worksheets.Add

    ' Tworzenie tabeli przestawnej
    Set PT = ActiveSheet.PivotTables.Add( _
        PivotCache:=PTCache, _
        TableDestination:=Range("A3"))

    ' Dodawanie poszczególnych pól tabeli przestawnej
    With PT
        .PivotFields("Region").Orientation = xlPageField
        .PivotFields("Miesiąc").Orientation = xlColumnField
        .PivotFields("RepHand1").Orientation = xlRowField
        .PivotFields("Sprzedaż").Orientation = xlDataField

        ' nie wyświetlamy etykiet pól
        .DisplayFieldCaptions = False
    End With
End Sub

```

Procedurę CreatePivotTable uproszczono (co przyczynia się do zwiększenia jej przejrzystości), deklarując tylko dwie zmienne obiektowe: PTCache oraz PT. Nowy obiekt PivotCache jest tworzony za pomocą metody Create. Następnie dodawany jest nowy arkusz, który staje się arkuszem aktywnym (na którym zostanie osadzona tabela przestawna). Nowy obiekt PivotTable zostaje utworzony za pomocą metody Add kolekcji PivotTables. W ostatniej sekcji kodu następuje dodanie pól do tabeli przestawnej i określenie lokalizacji poszczególnych pól w tabeli (poprzez przypisanie odpowiednich wartości do właściwości Orientation).

W pierwotnie wygenerowanym makrze znajdował się zakodowany *na sztywno* zakres danych służący do utworzenia obiektu PivotCache ('Arkusz1!R1C1:R13C4') oraz docelowa lokalizacja tabeli przestawnej (Arkusz2). W procedurze CreatePivotTable tabela przestawna jest tworzona na podstawie bieżącego obszaru otaczającego komórkę A1. Dzięki temu zyskujemy pewność, że makro będzie działać poprawnie po wprowadzeniu dodatkowych danych.

Dodanie nowego arkusza jeszcze przed utworzeniem tabeli przestawnej eliminuje konieczność sztywnego kodowania odwołań do arkusza. Kolejną różnicą jest to, że zoptymalizowane makro nie określa nazwy tworzonej tabeli przestawnej. Zamiast tego używamy zmiennej obiektowej PT, dzięki której nie ma potrzeby odwoływanego się do tabeli przestawnej po nazwie.



Kod byłby bardziej uniwersalny, gdyby do przetwarzania kolekcji PivotFields użyto indeksów zamiast literałów. W ten sposób makro działałoby nawet wtedy, gdy użytkownik zmieniłby nagłówki kolumn. Na przykład bardziej uniwersalny kod zawierałby odwołanie PivotFields(1) zamiast PivotFields('Region').

Kompatybilność tabel przestawnych

Jeżeli planujesz udostępnienie skoroszytów zawierających tabele przestawne użytkownikom, którzy korzystają z poprzednich wersji Excela, powinieneś poświęcić chwilę czasu zagadnieniom związanym z kompatybilnością. Jeżeli przyjrzysz się kodowi VBA wygenerowanemu przez rejestrator makr, przedstawionemu w podrozdziale „Tworzenie tabel przestawnych”, to z pewnością zauważysz następujące polecenie:

```
DefaultVersion:=xlPivotTableVersion15
```

Jeżeli Twój skoroszyt jest otwierany w trybie zgodności, to zarejestrowane polecenie będzie miało następującą postać:

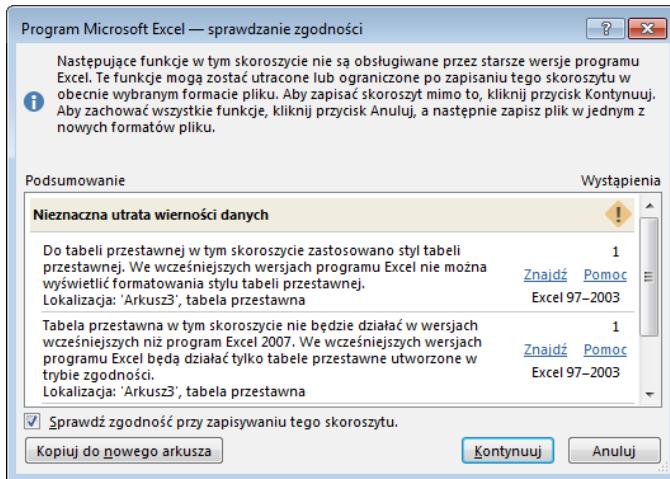
```
DefaultVersion:=xlPivotTableVersion10
```

Przekonasz się również, że zarejestrowany kod jest zupełnie inny, ponieważ począwszy od wersji 2007 Excela w tabelach przestawnych zostały wprowadzone znaczące zmiany.

Założmy, że utworzyłeś tabelę przestawną w Excelu 2013 i przesyłasz taki skoroszyt do użytkownika, który nadal korzysta z Excela 2003. Użytkownik Excela 2003 będzie w stanie zobaczyć tabelę przestawną, ale dane w tabeli nie będą „odświeżalne” — inaczej mówiąc, będzie to po prostu statyczny zestaw danych.

Aby w Excelu 2013 utworzyć tabelę przestawną, która będzie miała zachowaną wstępna kompatybilność z poprzednimi wersjami Excela, musisz zapisać taki skoroszyt w starym formacie XLS, a następnie zamknąć go i ponownie otworzyć. Po wykonaniu takiej operacji tabele przestawne, które utworzysz, będą poprawnie działały z wersjami Excela starszymi niż 2007. Takie zachowanie kompatybilności jest oczywiście okupione tym, że nie będziesz mógł w takich tabelach korzystać z nowych mechanizmów tabel przestawnych, wprowadzonych w nowszych wersjach Excela.

Na szczęście, Excel posiada ciekawy mechanizm o nazwie *Sprawdzanie zgodności*, który wyświetli na ekranie odpowiedni komunikat w sytuacji, kiedy zapisanie pliku w starszym formacie może spowodować utratę lub ograniczenie funkcjonalności skoroszytu (patrz rysunek poniżej). Pamiętaj jednak, że mechanizm ten nie sprawdza kompatybilności makr powiązanych z tabelami przestawnymi. Makra omawiane w tym rozdziale tworzą tabele przestawne, które *nie są kompatybilne* ze starszymi wersjami Excela.



Jak zawsze najlepszym sposobem poznania potrzebnych obiektów, metod i właściwości jest zarejestrowanie wykonywanych działań w makrze. Następnie, aby zrozumieć sposób użycia poszczególnych elementów, warto zapoznać się z tematami pomocy systemowej. Pamiętaj, że niemal w każdym przypadku zarejestrowane makra wymagają modyfikacji. Zamiast rejestrować i poprawiać makro, możesz oczywiście napisać kod od początku, ale wymaga to pewnego doświadczenia.

Tworzenie złożonych tabel przestawnych

W tym podrozdziale zaprezentowano kod VBA tworzący dosyć złożoną tabelę przestawną.

Na rysunku 15.3 zaprezentowano fragment dużej tabeli danych, złożonej z 15 840 wierszy zawierających hierarchiczne dane budżetowe firmy. Korporacja składa się z 5 oddziałów, a każdy oddział z 11 wydziałów. W każdym wydziale są cztery kategorie budżetowe, a każda kategoria zawiera po kilka pozycji. Kwoty budżetu oraz wydatki rzeczywiste są zapisane dla każdego z 12 miesięcy. Naszym celem jest podsumowanie tych informacji za pomocą tabeli przestawnej.

Rysunek 15.3.

Dane z tego skoroszytu będą zestawione w tabeli przestawnej

	A	B	C	D	E	F	G
1	Oddział	Wydział	Kategoria	Pozycja	Miesiąc	Budżet	Wydatki
2	Ameryka Pln	Informatyka	Rekompensaty	Wynagrodzenia	Sty	2583	3165
3	Ameryka Pln	Informatyka	Rekompensaty	Zyski	Sty	4496	2980
4	Ameryka Pln	Informatyka	Rekompensaty	Premie	Sty	3768	3029
5	Ameryka Pln	Informatyka	Rekompensaty	Zlecenia	Sty	3133	2815
6	Ameryka Pln	Informatyka	Rekompensaty	Podatki od wynagrod.	Sty	3559	3770
7	Ameryka Pln	Informatyka	Rekompensaty	Szkolenie	Sty	3099	3559
8	Ameryka Pln	Informatyka	Rekompensaty	Konferencje	Sty	2931	3199
9	Ameryka Pln	Informatyka	Rekompensaty	Rozrywka	Sty	2632	2633
10	Ameryka Pln	Informatyka	Obiekty	Wynajem	Sty	2833	2508
11	Ameryka Pln	Informatyka	Obiekty	Dzierżawa	Sty	3450	2631
12	Ameryka Pln	Informatyka	Obiekty	Narzędzia	Sty	4111	3098
13	Ameryka Pln	Informatyka	Obiekty	Remonty	Sty	3070	2870
14	Ameryka Pln	Informatyka	Obiekty	Rachunki telefonicz.	Sty	3827	4329
15	Ameryka Pln	Informatyka	Obiekty	Inne	Sty	3843	3322
16	Ameryka Pln	Informatyka	Materiały i serwis	Materiały biurowe	Sty	2642	3218
17	Ameryka Pln	Informatyka	Materiały i serwis	Akcesoria komputer.	Sty	3052	4098
18	Ameryka Pln	Informatyka	Materiały i serwis	Książki	Sty	4346	3361
19	Ameryka Pln	Informatyka	Materiały i serwis	Usługi zewnętrzne	Sty	2869	3717
20	Ameryka Pln	Informatyka	Materiały i serwis	Inne	Sty	3328	3116
21	Ameryka Pln	Informatyka	Sprzęt	Sprzęt komputerowy	Sty	3088	2728
22	Ameryka Pln	Informatyka	Sprzęt	Oprogramowanie	Sty	4226	2675
23	Ameryka Pln	Informatyka	Sprzęt	Kserokopiarki	Sty	3780	3514



Skoroszyt z tym przykładem (*Złożona tabela przestawna.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 15.4 pokazano tabelę przestawną utworzoną na podstawie zaprezentowanych wyżej danych. Warto zwrócić uwagę, że tabela przestawna zawiera obliczane pole Odchylenie, które obrazuje różnicę pomiędzy zaplanowaną kwotą budżetu a wydatkami rzeczywistymi.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Oddział	(Wszystko)												
2	Kategoria	(Wszystko)												
3														
4		Sty	Lut	Mar	Kwi	Maj	Cze	Lip	Sie	Wrz	Paź	Lis	Gru	Suma końcowa
5	Bezpieczeństwo													
6	Budżet	419 195	419 294	413 258	421 700	421 875	421 231	417 392	410 715	417 112	430 013	412 302	419 939	5 024 026
7	Wydatki	409 486	418 697	427 401	419 221	421 266	420 388	423 828	424 682	418 400	415 569	410 717	423 031	5 032 686
8	Różnica	9 709	0 597	-14 143	2 479	0 609	0 843	-6 436	-13 967	-1 288	14 444	1 585	-3 092	-8 660
9	Dostawy													
10	Budżet	429 595	429 917	423 975	419 617	433 168	421 732	413 330	429 881	428 164	428 851	412 565	423 607	5 094 402
11	Wydatki	413 835	413 777	414 932	411 949	410 561	422 913	411 481	421 191	420 974	419 458	428 434	416 709	5 006 214
12	Różnica	15 760	16 140	9 043	7 668	22 607	-1 181	1 849	8 690	7 190	9 393	-15 869	6 898	88 188
13	Informatyka													
14	Budżet	422 197	422 057	419 659	417 260	422 848	421 038	421 676	418 093	419 999	418 752	421 106	428 679	5 053 364
15	Wydatki	414 743	438 990	430 545	424 214	411 775	421 909	420 210	414 966	419 913	430 262	417 478	408 644	5 053 649
16	Różnica	7 454	-16 933	-10 886	-6 954	11 073	-0 871	1 466	3 127	0 086	-11 510	3 628	20 035	-0 285
17	Kadry													
18	Budżet	422 053	425 313	418 634	423 038	423 514	419 602	415 197	419 701	422 762	413 741	410 972	422 746	5 037 273
19	Wydatki	424 934	429 275	407 053	429 187	410 258	421 870	428 551	422 469	422 252	421 838	415 125	417 222	5 050 034
20	Różnica	-2 881	-3 962	11 581	-6 149	13 256	-2 268	-13 354	-2 768	0 510	-8 097	-4 153	5 524	-12 761
21	Księgowość													
22	Budżet	422 455	433 317	420 522	417 964	411 820	414 012	427 431	418 530	412 134	421 678	426 602	418 445	5 044 910
23	Wydatki	422 662	413 163	416 522	420 672	431 303	429 993	425 879	415 253	417 401	417 806	425 271	420 026	5 055 951

Rysunek 15.4. Tabela przestawna utworzona na podstawie danych z poprzedniego rysunku



Innym rozwiązaniem jest wstawienie do tabeli nowej kolumny i utworzenie formuły, która będzie obliczała różnicę pomiędzy założonym budżetem a rzeczywistymi wydatkami. Jeżeli jednak dane są pobierane z zewnętrznego źródła (czyli kiedy na przykład zamiast skoroszytu użyjemy zewnętrznej bazy danych), taka opcja może nie być dostępna.

Kod tworzący tabelę przestawną

Poniżej zamieszczamy kod procedury tworzącej tabelę przestawną:

```

Sub CreatePivotTable()
    Dim PTcache As PivotCache
    Dim PT As PivotTable

    Application.ScreenUpdating = False
    ' Usuń ArkuszTabeliPrzestawnej, jeżeli już istnieje
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("ArkuszTabeliPrzestawnej").Delete
    On Error GoTo 0

    ' Utwórz bufor tabeli przestawnej
    Set PTcache = ActiveWorkbook.PivotCaches.Create( _
        SourceType:=xlDatabase, _
        SourceData:=Range("A1").CurrentRegion.Address)

    ' Dodaj nowy arkusz
    Worksheets.Add
    ActiveSheet.Name = "ArkuszTabeliPrzestawnej"
    ActiveWindow.DisplayGridlines = False

    ' Utwórz tabelę przestawną na podstawie danych z bufora
    Set PT = ActiveSheet.PivotTables.Add( _
        PivotCache:=PTcache, _

```

```

TableDestination:=Range("A1"), _
TableName:="Budżet")

With PT
    ' Dodaj pola
    .PivotFields("Kategoria").Orientation = xlPageField
    .PivotFields("Oddział").Orientation = xlPageField
    .PivotFields("Wydział").Orientation = xlRowField
    .PivotFields("Miesiąc").Orientation = xlColumnField
    .PivotFields("Budżet").Orientation = xlDataField
    .PivotFields("Rzeczywiste wydatki").Orientation = xlDataField
    .DataPivotField.Orientation = xlRowField

    ' Dodaj pole obliczeniowe, w którym będziemy obliczać odchylenie budżetu
    .CalculatedFields.Add "Odchylenie", "=Budget-Actual"
    .PivotFields("Odchylenie").Orientation = xlDataField

    ' Określ format liczbowy
    .DataBodyRange.NumberFormat = "0,000"

    ' Nadaj styl tabeli
    .TableStyle2 = "PivotStyleMedium2"

    ' Ukryj nagłówki pól
    .DisplayFieldCaptions = False

    ' Zmień etykiety
    .PivotFields("Suma z Budżet").Caption = " Budżet"
    .PivotFields("Suma z Rzeczywiste wydatki").Caption = " Rzeczywiste wydatki"
    .PivotFields("Suma z Odchylenie").Caption = " Odchylenie"
End With
End Sub

```

Jak działa złożona tabela przestawna?

Procedura CreatePivotTable najpierw usuwa arkusz ArkuszTabeliPrzestawnej, jeżeli taki wcześniej istniał. Następnie tworzy obiekt PivotCache, wstawia nowy arkusz o nazwie ArkuszTabeliPrzestawnej i w końcu tworzy tabelę przestawną na podstawie danych z bufora PivotCache. Do tabeli przestawnej dodawane są następujące pola:

- **Kategoria** — pole filtra strony raportu;
- **Oddział** — pole filtra strony raportu;
- **Wydział** — pole wiersza;
- **Miesiąc** — pole kolumny;
- **Budżet** — pole danych;
- **Rzeczywiste wydatki** — pole danych.

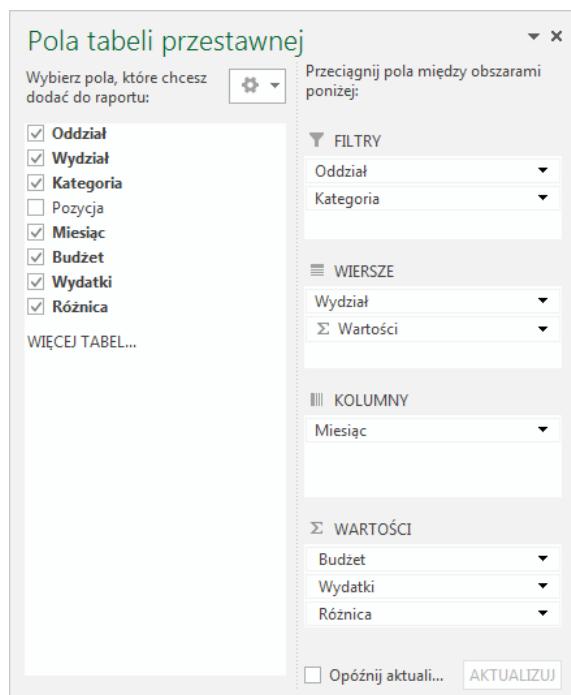
Zwrót uwagę, że właściwość Orientation obiektu DataPivotField jest ustawiana na wartość xlRowField przy użyciu następującego polecenia:

```
.DataPivotField.Orientation = xlRowField
```

Polecenie to określa ogólną orientację tabeli przestawnej i reprezentuje pole Σ Wartości na liście pól tabeli przestawnej (patrz rysunek 15.5). Spróbuj przeciągnąć to pole do obszaru Etykiety kolumn, aby zobaczyć, jaki to będzie miało wpływ na wygląd tabeli przestawnej.

Rysunek 15.5.

Okno dialogowe Lista pól tabeli przestawnej



Następnie wykorzystano metodę `Add kolekcji CalculatedFields` w celu utworzenia obliczanego pola `Odchylenie`, którego wartość jest wyliczana poprzez odjęcie wartości faktycznie wydanej kwoty (pole `RZECZYWISTE_WYDATKI`) od kwoty budżetu (pole `BUDŻET`). Takie pole obliczeniowe jest traktowane jak pole danych.



Aby ręcznie dodać pole obliczeniowe do tabeli przestawnej, kliknij tabelę przestawną, przejdź na kartę **NARZĘDZIA TABEL PRZESTAWNYCH**, potem na kartę **ANALIZA**, naciśnij przycisk **Pola, elementy i zestawy** znajdujący się w grupie opcji **Obliczenia**, a następnie wybierz polecenie **Pole obliczeniowe**. Na ekranie pojawi się okno dialogowe **Wstawianie pola obliczeniowego**.

Na koniec kod procedury dokonuje kilku kosmetycznych poprawek:

- Nadaje odpowiedni format liczbowy obiektowi `DataBodyRange` (który reprezentuje dane całej tabeli przestawnej).
- Nadaje tabeli przestawnej odpowiedni styl formatowania.
- Ukrywa nagłówki pól (jest to odpowiednik następującej sekwencji poleceń: przejdź na kartę **NARZĘDZIA TABEL PRZESTAWNYCH**, następnie na kartę **ANALIZA**, wybierz opcję **Pokaż** i z menu wybierz polecenie **Nagłówki pól**).

- Zmienia nagłówki pól wyświetlane w tabeli przestawnej. Na przykład nagłówek *Suma z Budżet* jest zastępowany nagłówkiem *Budżet*. Zwróć uwagę, że ciąg znaków *Budżet* jest poprzedzony spacją. Excel nie pozwala na użycie nazw nagłówków identycznych z nazwami pól, stąd dodanie pojedynczej spacji pozwala na obejście tego ograniczenia.



Tworząc tę procedurę, najpierw wygenerowałem kod za pomocą rejestratora makr, aby zapoznać się z wieloma właściwościami tabel przestawnych. Taka technika, połączona z informacjami, które znalazłem w pomocy systemowej i które odkryłem podczas wielu eksperymentów metodą prób i błędów, w efekcie dały mi zasób wiadomości niezbędnych do napisania tego programu.

Jednoczesne tworzenie wielu tabel przestawnych

W ostatnim przykładzie utworzymy kilka tabel przestawnych, które będą zawierały podsumowania danych pobranych z ankiety wypełnianej przez klientów. Dane są zapisane w arkuszu w bazie danych (patrz rysunek 15.6) złożonej ze 150 wierszy. Każdy wiersz zawiera dane o płci respondenta oraz odpowiedź od 1 do 5 dla każdej z 14 pozycji ankiety.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Nazwisko	Płeć	Lokalizacja magazynów są dobrane	Godzinny otwarcia są dobrane	Mają tematyczne sa dobrze utrzymane	Dobry kontakt telefoniczny	Podoba mi się witryna WWW	Pracownicy są milni	Pracownicy oferują pomoc	Pracownicy mają wiele	Konkurencyjne ceny	Duży wybór towarów	Podoba mi się reklamy TV	Spośród tych z wysokiej jakości	Odpinie jeśli żadnego	Polecam firmę waszą przyjacielom
2	Osoba1	Mężczyzna	1	4	4	4	1	1	2	1	1	2	5	2	2	1
3	Osoba2	Kobieta	2	5	1	1	4	2	4	3	3	2	5	2	3	3
4	Osoba3	Mężczyzna	1	1	4	2	3	3	2	1	2	3	3	4	3	2
5	Osoba4	Mężczyzna	2	1	3	5	1	2	3	4	2	1	3	4	1	2
6	Osoba5	Kobieta	2	2	5	5	4	2	1	5	5	2	3	4	2	5
7	Osoba6	Kobieta	2	4	3	3	1	1	4	4	4	2	2	2	2	4
8	Osoba7	Kobieta	2	4	5	4	5	3	2	5	4	4	1	5	4	4
9	Osoba8	Mężczyzna	3	2	1	2	3	4	3	1	2	4	3	4	4	2
10	Osoba9	Kobieta	3	4	4	4	5	1	4	1	4	1	2	1	1	4
11	Osoba10	Mężczyzna	2	1	5	5	5	1	4	1	2	2	5	2	2	2
12	Osoba11	Mężczyzna	4	3	3	2	1	2	4	2	1	4	4	2	4	1
13	Osoba12	Kobieta	2	1	4	5	5	5	3	1	4	1	4	3	4	4
14	Osoba13	Kobieta	4	3	4	3	2	5	3	3	2	2	5	2	4	2
15	Osoba14	Kobieta	2	3	4	2	1	1	4	2	1	3	3	1	3	1
16	Osoba15	Kobieta	1	3	5	1	2	2	4	1	3	4	5	5	4	3
17	Osoba16	Mężczyzna	1	4	1	3	4	3	4	4	5	3	4	1	3	3
18	Osoba17	Kobieta	3	4	3	5	5	4	4	3	2	4	2	2	4	2
19	Osoba18	Mężczyzna	1	5	5	3	5	3	4	2	3	2	3	3	2	3
20	Osoba19	Kobieta	1	3	5	4	5	5	5	1	1	5	3	2	5	1
21	Osoba20	Mężczyzna	2	2	5	2	2	5	5	3	1	5	2	4	5	1
22	Osoba21	Mężczyzna	3	4	1	4	5	1	3	1	4	1	2	1	1	4
23	Osoba22	Mężczyzna	2	1	5	5	5	1	2	1	2	2	5	2	2	2
24	Osoba23	Mężczyzna	4	3	4	2	1	2	1	2	1	4	4	1	4	2
25	Osoba24	Kobieta	1	1	2	5	5	3	1	4	1	4	3	1	2	2
26	Osoba25	Kobieta	2	3	4	3	2	5	3	3	2	2	5	2	1	2
27	Osoba26	Mężczyzna	1	3	4	2	1	1	3	2	1	3	3	1	1	1

Rysunek 15.6. Wyniki ankiety będą podsumowane w kilku tabelach przestawnych



Skoroszyt z tym przykładem (*Analiza danych z ankiety.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 15.7 pokazano kilka spośród 28 tabel przestawnych utworzonych przez nasze makro. Dla każdej pozycji ankiety utworzono dwie tabele przestawne — jedna wyświetla dane w procentach, druga rzeczywiste wartości liczbowe.

	A	B	C	D	E	F	G	H	I	J
1	Lokalizacje magazynów są dogodne					Lokalizacje magazynów są dogodne				
2	Liczba z Lokalizacje magazynów są dogodne	Kobieta	Mężczyzna	Suma końcowa		Liczba z Lokalizacje magazynów są dogodne	Kobieta	Mężczyzna	Suma końcowa	
3										
4	Kategorycznie się nie zgadzam	20	26	46		Kategorycznie się nie zgadzam	42,6%	49,1%	46,0%	
5	Nie zgadzam się	13	11	24		Nie zgadzam się	27,7%	20,8%	24,0%	
6	Jestem niezdecydowany	10	6	16		Jestem niezdecydowany	21,3%	11,3%	16,0%	
7	Zgadzam się	3	10	13		Zgadzam się	6,4%	18,9%	13,0%	
8	Calkowicie się zgadzam	1	1	1		Calkowicie się zgadzam	2,1%	0,0%	1,0%	
9	Suma końcowa	47	53	100						
10										
11	Godziny otwarcia są dogodne					Godziny otwarcia są dogodne				
12	Liczba z Godziny otwarcia są dogodne	Kobieta	Mężczyzna	Suma końcowa		Liczba z Godziny otwarcia są dogodne	Kobieta	Mężczyzna	Suma końcowa	
13										
14	Kategorycznie się nie zgadzam	6	9	15		Kategorycznie się nie zgadzam	12,8%	17,0%	15,0%	
15	Nie zgadzam się	5	8	13		Nie zgadzam się	10,6%	15,1%	13,0%	
16	Jestem niezdecydowany	19	16	35		Jestem niezdecydowany	40,4%	30,2%	35,0%	
17	Zgadzam się	15	15	30		Zgadzam się	31,9%	28,3%	30,0%	
18	Calkowicie się zgadzam	2	5	7		Calkowicie się zgadzam	4,3%	9,4%	7,0%	
19	Suma końcowa	47	53	100						
20										
21	Magazyny są dobrze utrzymane					Magazyny są dobrze utrzymane				
22	Liczba z Magazyny są dobrze utrzymane	Kobieta	Mężczyzna	Suma końcowa		Liczba z Magazyny są dobrze utrzymane	Kobieta	Mężczyzna	Suma końcowa	
23										
24	Kategorycznie się nie zgadzam	6	8	14		Kategorycznie się nie zgadzam	12,8%	15,1%	14,0%	
25	Nie zgadzam się	4	3	7		Nie zgadzam się	8,5%	5,7%	7,0%	
26	Jestem niezdecydowany	13	11	24		Jestem niezdecydowany	27,7%	20,8%	24,0%	
27	Zgadzam się	17	20	37		Zgadzam się	36,2%	37,7%	37,0%	
28	Calkowicie się zgadzam	7	11	18		Calkowicie się zgadzam	14,9%	20,8%	18,0%	
29	Suma końcowa	47	53	100						
30										
	Podsumowanie	DaneZAnkiety								

Rysunek 15.7. Tabele przestawne utworzone za pomocą procedury VBA

Powyższe tabele zostały utworzone za pomocą następującej procedury VBA:

```

Sub MakePivotTables()
    ' Procedura tworzy 28 tabel przestawnych
    Dim PTCache As PivotCache
    Dim PT As PivotTable
    Dim SummarySheet As Worksheet
    Dim ItemName As String
    Dim Row As Long, Col As Long, i As Long

    Application.ScreenUpdating = False

    ' Usuń arkusz Podsumowanie, jeżeli już istnieje
    On Error Resume Next
    Application.DisplayAlerts = False
    Sheets("Podsumowanie").Delete
    On Error GoTo 0

    ' Dodaj arkusz Podsumowanie
    Set SummarySheet = Worksheets.Add
    ActiveSheet.Name = "Podsumowanie"

    ' Utwórz bufor tabel przestawnych
    Set PTCache = ActiveWorkbook.PivotCaches.Create( _
        SourceType:=xlDatabase, _
        SourceData:=Sheets("DaneZAnkiety").Range("A1") . . . _
        CurrentRegion)

    Row = 1
    For i = 1 To 6 Step 5 '2 kolumny
        For Col = 1 To 6 Step 5 '2 kolumny
            ItemName = Sheets("DaneZAnkiety").Cells(1, i + 2)
            With Cells(Row, Col)

```

```

        .Value = ItemName
        .Font.Size = 16
    End With

    ' Utwórz tabelę przestawne
    Set PT = ActiveSheet.PivotTables.Add( _
        PivotCache:=PTCache, _
        TableDestination:=SummarySheet.Cells(Row + 1, Col))

    ' Dodaj pola
    If Col = 1 Then 'Tabele częstości
        With PT.PivotFields(ItemName)
            .Orientation = xlDataField
            .Name = "Liczba odpowiedzi"
            .Function = xlCount
        End With
    Else ' Tabele procentowe
        With PT.PivotFields(ItemName)
            .Orientation = xlDataField
            .Name = "Procent"
            .Function = xlCount
            .Calculation = xlPercentOfColumn
            .NumberFormat = "0.0%"
        End With
    End If

    PT.PivotFields(ItemName).Orientation = xlRowField
    PT.PivotFields("Płeć").Orientation = xlColumnField
    PT.TableStyle2 = "PivotStyleMedium2"
    PT.DisplayFieldCaptions = False
    If Col = 6 Then
        ' Dodaj paski danych do ostatniej kolumny
        PT.ColumnGrand = False
        PT.DataBodyRange.Columns(3).FormatConditions. _
            AddDataBar
        With pt.DataBodyRange.Columns(3).FormatConditions(1)
            .BarFillType = xlDataBarFillSolid
            .MinPoint.Modify newtype:=xlConditionValueNumber, newValue:=0
            .MaxPoint.Modify newtype:=xlConditionValueNumber, newValue:=1
        End With
    End If
    Next Col
    Row = Row + 10
Next i

    ' Zamień liczby na opisy
    With Range("A:A,F:F")
        .Replace "1", "Kategorycznie się nie zgadzam"
        .Replace "2", "Nie zgadzam się"
        .Replace "3", "Jestem niezdecydowany"
        .Replace "4", "Zgadzam się"
        .Replace "5", "Całkowicie się zgadzam"
    End With
End Sub

```

Zwróć uwagę na fakt, że wszystkie tabele przestawne zostały utworzone na podstawie jednego obiektu PivotCache.

Tabele przestawne są tworzone w zagnieździonej pętli. Licznik pętli `Col` jest inkrementowany od 1 do 6 za pomocą parametru `Step`. Kod dla drugiej kolumny tabel przestawnych jest nieco inny niż w przypadku pierwszej kolumny, a w szczególności wykonuje następujące, dodatkowe operacje:

- Wyświetla wartości jako procent ogólnej liczby odpowiedzi.
 - Nie wyświetla wierszy sum końcowych poszczególnych tabel
 - Przypisuje format procentowy dla danych tabeli.
 - Wyświetla paski danych formatowania warunkowego.

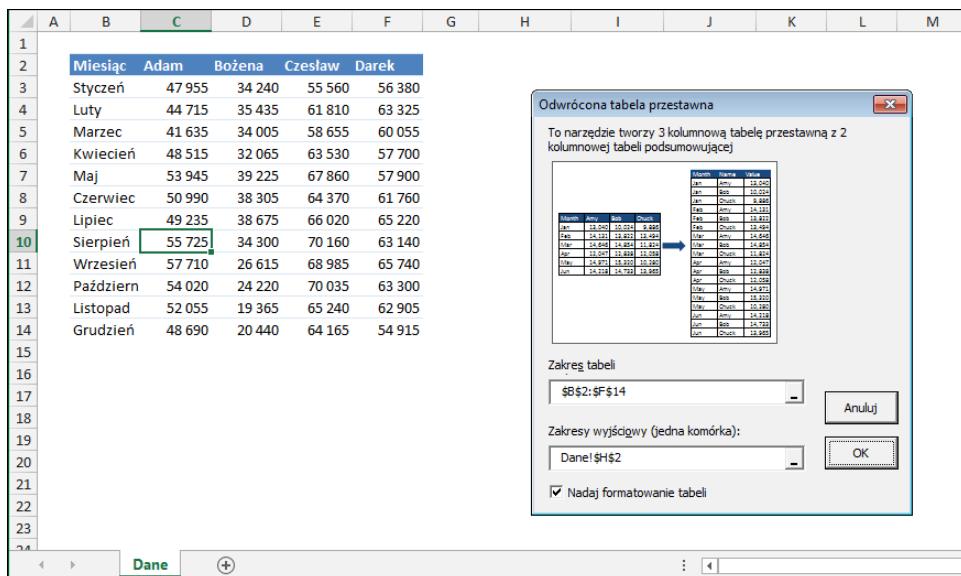
Początkowy wiersz każdej kolejnej tabeli przestawnej jest wyliczany na podstawie zmiennej `Row`. Po utworzeniu tabeli kod zamienia kategorie liczbowe w pierwszej kolumnie na tekst (np. wartość 1 jest zastępowana tekstem *Kategorycznie się nie zgadzam*).

Tworzenie odwróconych tabel przestawnych

Tabela przestawna to inaczej podsumowanie danych znajdujących się w danej tabeli. Ale co można zrobić, jeżeli masz daną tabelę podsumowującą i chcesz z niej utworzyć tabelę danych? Przykład takiej sytuacji przedstawiono na rysunku 15.8. W zakresie B2:F14 znajduje się tabela podsumowująca — bardzo podobna do prostej tabeli przestawnej. W kolumnach I:K znajduje się składająca się z 48 wierszy tabela danych, utworzona na podstawie tabeli podsumowującej. W tej tabeli każdy wiersz zawiera jeden punkt danych, a w pierwszych dwóch kolumnach znajduje się jego opis. Innymi słowy, dane w przetworzonej tabeli są znormalizowane (patrz ramka „Dane, na podstawie których można utworzyć tabelę przestawną” we wcześniejszej części tego rozdziału).

Rysunek 15.8. Tabela podsumowująca (z lewej) zostanie zamieniona na tabele danych (po prawej)

Excel nie posiada żadnych mechanizmów pozwalających na automatyczną zamianę tabeli podsumowującej na tabelę danych, a zatem wykonanie takiej operacji jest zadaniem dla procedury VBA. Po utworzeniu tego makra nieco czasu poświęciłem na utworzenie formularza *UserForm*, przedstawiony na rysunku 15.9. Okno formularza pozwala na pobranie zakresu wejściowego i wyjściowego tabeli i posiada opcję pozwalającą na sformatowanie zakresu wyjściowego jako tabeli.



Rysunek 15.9. Okno dialogowe, w którym możesz zdefiniować zakres wejściowy i wyjściowy tabeli



Skoroszyt z tym przykładem (*Odwrócona tabela przestawna.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Kiedy użytkownik naciśnie przycisk *OK* na formularzu *UserForm*, kod VBA dokona sprawdzenia poprawności zakresów i następnie przy użyciu polecenia przedstawionego poniżej wywoła procedurę *ReversePivot*:

```
Call ReversePivot(SummaryTable, OutputRange, cbCreateTable)
```

Wywołanie przekazuje trzy argumenty:

- *SummaryTable* — obiekt klasy Range reprezentujący tabelę podsumowującą.
- *OutputRange* — obiekt klasy Range reprezentujący górną, lewą komórkę zakresu wyjściowego.
- *CbCreateTable* — obiekt klasy CheckBox znajdujący się na formularzu *UserForm*.

Nasza procedura będzie działać dla tabeli o dowolnym rozmiarze. Liczbę wierszy danych w tabeli wynikowej można wyznaczyć ze wzoru $(r-1)*(c-1)$, gdzie r oraz c reprezentują odpowiednio liczbę wierszy i kolumn tabeli podsumowującej.

Kod procedury ReversePivot przedstawiony został poniżej:

```
Sub ReversePivot(SummaryTable As Range, _
    OutputRange As Range, CreateTable As Boolean)
    Dim r As Long, c As Long
    Dim OutRow As Long, OutCol As Long

    ' Zamiana zakresów
    OutRow = 2
    Application.ScreenUpdating = False
    OutputRange.Range("A1:C3") = Array("Kolumna1", "Kolumna2", "Kolumna3")
    For r = 2 To SummaryTable.Rows.Count
        For c = 2 To SummaryTable.Columns.Count
            OutputRange.Cells(OutRow, 1) = SummaryTable.Cells(r, 1)
            OutputRange.Cells(OutRow, 2) = SummaryTable.Cells(1, c)
            OutputRange.Cells(OutRow, 3) = SummaryTable.Cells(r, c)
            OutRow = OutRow + 1
        Next c
    Next r

    ' Czy zamieniamy na tabelę?
    If CreateTable Then _
        ActiveSheet.ListObjects.Add xlSrcRange, _
        OutputRange.CurrentRegion, , xlYes
End Sub
```

Procedura jest stosunkowo prosta. Kod przechodzi w pętli przez kolejne wiersze i kolumny zakresu wejściowego i zapisuje dane w zakresie wyjściowym. Zakres wyjściowy zawsze będzie składał się z trzech kolumn. Zmienna OutRow przechowuje informacje o bieżącym wierszu zakresu wyjściowego. Na koniec program sprawdza, czy użytkownik zaznaczył opcję formatowania zakresu wyjściowego, i jeżeli tak, program przy użyciu metody Add kolekcji ListObjects zamienia zakres wyjściowy na tabelę danych.

Rozdział 16.

Wykresy

W tym rozdziale:

- Podstawowe wiadomości o wykresach Excela
- Różnice pomiędzy wykresami osadzonymi na arkuszu a arkuszami wykresów
- Model obiektowy Chart
- Zastosowanie metod innych niż rejestrator makr do pracy z wykresami
- Przykłady technik przetwarzania wykresów w języku VBA
- Przykłady złożonych makr przetwarzających wykresy
- Ciekawe i użyteczne techniki tworzenia wykresów
- Wykresy przebiegu w czasie (ang. *Sparklines*)

Podstawowe wiadomości o wykresach

Excel pozwala na tworzenie wielu typów wykresów i daje użytkownikowi możliwość kontrolowania niemal każdego elementu tworzonego wykresu.

Wykres to nic innego, jak zbiór obiektów, z których każdy charakteryzuje się innymi właściwościami i metodami. Z tego powodu wykonywanie działań z wykresami za pomocą języka VBA jest procesem dość złożonym. W tym rozdziale omówiono najważniejsze pojęcia, które trzeba poznać, aby pisać kod VBA generujący lub modyfikujący wykresy. Kluczem do sukcesu, jak się wkrótce przekonasz, jest dokładne zapoznanie się z hierarchią obiektów dotyczących wykresów.

Lokalizacja wykresu

W Excelu wykresy można umieszczać w jednym z dwóch miejsc w skoroszycie:

- **Jako obiekt osadzony na arkuszu danych** — arkusz może zawierać wówczas dowolną liczbę osadzonych wykresów.
- **W osobnym arkuszu** — arkusz wykresu zawiera tylko jeden wykres.

Kilka słów o kompatybilności

Przykłady kodu VBA omawiane w tym rozdziale korzystają z wielu właściwości i metod wykresów, które zostały wprowadzone w Excelu 2013. Na przykład w Excelu 2013 pojawiła się nowa metoda o nazwie AddChart2. Co prawda stara metoda AddChart i wiele innych nadal działa, ale w tym rozdziale skoncentrujemy się na nowych wersjach, które są często znacznie łatwiejsze w użyciu. W rezultacie kod VBA wielu przykładów nie będzie działał poprawnie z poprzednimi wersjami programu Excel (innymi niż 2013).

Większość wykresów jest tworzona ręcznie, przy użyciu polecen z grupy *WSTAWA-Wykresy*. Wykresy można również tworzyć za pomocą kodu VBA. Oczywiście język VBA można również wykorzystać w celu modyfikowania istniejących wykresów.



Najszybszym sposobem utworzenia wykresu w nowym arkuszu jest zaznaczenie danych i wcisnięcie klawiszy *Alt+F1*. Excel osadzi wykres na aktywnym arkuszu danych, wykorzystując domyślny typ wykresu. Aby utworzyć nowy wykres domyślny na arkuszu wykresu, zaznacz zakres danych i naciśnij klawisz *F11*.

Kluczowym elementem przy tworzeniu wykresów jest tzw. *wykres aktywny*, czyli wykres, który jest aktualnie zaznaczony. Kiedy użytkownik kliknie wbudowany wykres lub arkusz wykresu, następuje uaktywnienie obiektu Chart. W języku VBA właściwość ActiveChart zwraca uaktywniony obiekt Chart (jeżeli jakiś istnieje). Można napisać kod wykonujący działania z tym obiektem Chart, podobnie jak można napisać kod wykonujący działania z obiektem Workbook zwracanym poprzez właściwość ActiveWorkbook.

Oto przykład: jeżeli uaktywniono wykres, poniższa instrukcja spowoduje wyświetlenie właściwości Name obiektu Chart:

```
MsgBox ActiveChart.Name
```

Jeżeli wykres nie został uaktywniony, wykonanie tej instrukcji spowoduje wygenerowanie błędu.



Jak się okaże w dalszej części tego rozdziału, aby za pomocą kodu VBA wykonywać działania na wykresie, nie trzeba go uaktywniać.

Rejestrator makr a wykresy

Jeżeli przeczytałeś już inne rozdziały tej książki, to zapewne pamiętasz, że do poznawania obiektów, właściwości i metod bardzo często polecano używanie rejestratora makr. Jak zawsze, zarejestrowane makra są najlepszym materiałem źródłowym do nauki tworzenia procedur VBA pracujących z wykresami — w kodzie wygenerowanym przez rejestrator niemal zawsze znajdziesz poprawne odwołania do interesujących Cię obiektów, właściwości i metod.

Model obiektu Chart

Pierwsza próba analizy modelu obiektu Chart z pewnością wprowadzi Cię w zakłopotanie. Nie będzie w tym nic dziwnego, ponieważ model tego obiektu *jest* bardzo złożony i rozbudowany.

Założymy na przykład, że postanowiliśmy zmienić tytuł wyświetlanego na wbudowanym wykresie. Na szczytce obiektów znajduje się oczywiście obiekt Application (Excel). Zawiera on obiekt Workbook, a ten z kolei zawiera obiekt Worksheet. Obiekt Worksheet zawiera obiekt ChartObject, który zawiera obiekt Chart. Obiekt Chart zawiera obiekt ChartTitle, który posiada właściwość Text. To właśnie w niej jest zapisany tekst wyświetlany jako tytuł wykresu.

Oto inny sposób spojrzenia na tę hierarchię:

```
Application
  Workbook
    Worksheet
      ChartObject
        Chart
          ChartTitle
```

W kodzie VBA trzeba oczywiście ściśle przestrzegać hierarchii modelu obiektowego. Aby na przykład ustawić tytuł wykresu na *Sprzedaż firmy LTD*, należy zapisać następującą instrukcję VBA:

```
Worksheets("Arkusz1").ChartObjects(1).Chart.ChartTitle.Text = "Sprzedaż firmy LTD"
```

W tej instrukcji założono, że aktywny skoroszyt jest obiektem typu Workbook. Instrukcja zadziała dla pierwszego elementu w kolekcji ChartObjects w arkuszu o nazwie Arkusz1. Właściwość Chart zwraca odpowiedni obiekt Chart, natomiast właściwość ChartTitle zwraca obiekt ChartTitle. Na końcu uzyskujemy dostęp do właściwości Text.

Warto zwrócić uwagę na fakt, że powyższe polecenie nie zadziała poprawnie, jeżeli wykres nie będzie miał tytułu. Aby wykresowi nadać domyślny tytuł (czyli po prostu ciąg znaków Tytuł wykresu), możesz użyć następującego polecenia:

```
Worksheets("Arkusz1").ChartObjects(1).Chart.HasTitle = True
```

W przypadku arkusza wykresu, hierarchia obiektów nieco się różni, ponieważ nie zawiera obiektów Worksheet i ChartObject. Poniżej pokazano hierarchię obiektów dla obiektu ChartTitle w przypadku arkusza wykresu:

```
Application
  Workbook
    Chart
      ChartTitle
```

W celu zmiany tytułu wykresu na *Sprzedaż firmy LTD* w kodzie VBA należałoby zastosować poniższą instrukcję:

```
Sheets("Wykres1").ChartTitle.Text = "Sprzedaż firmy LTD"
```

Mówiąc inaczej, arkusz wykresu jest po prostu obiektem klasy Chart, który nie jest zawarty w obiekcie ChartObject. Dla wykresu wbudowanego jego obiektem nadzorującym jest ChartObject, a dla arkusza wykresu — obiekt Workbook.

Obydwie zaprezentowane poniżej instrukcje spowodują wyświetlenie okien komunikatów zawierających słowo *Chart*:

```
MsgBox TypeName(Sheets("Arkusz1").ChartObjects(1).Chart)
MsgBox TypeName(Sheets("Wykres1"))
```



Tworzenie nowego wykresu wbudowanego powoduje dodanie nowego elementu do kolekcji ChartObjects zawartej w określonym arkuszu (w arkuszu nie ma kolekcji Charts). Tworzenie nowego arkusza wykresu powoduje dodanie nowych elementów do kolekcji Charts oraz kolekcji Sheets określonego skoroszytu.

Tworzenie wykresów osadzonych na arkuszu danych

Obiekt ChartObject jest specjalnym rodzajem obiektu Shape i stąd jest elementem kolekcji Shapes. Aby utworzyć nowy wykres, powinieneś użyć metody AddChart2 obiektu Shapes. Polecenie przedstawione poniżej tworzy nowy, pusty wykres osadzony na arkuszu danych:

```
ActiveSheet.Shapes.AddChart2
```

Metoda AddChart2 używa siedmiu argumentów (wszystkie argumenty są opcjonalne):

- **Style** — wartość numeryczna, która definiuje styl (czyli inaczej mówiąc, ogólny wygląd) wykresu.
- **xlChartType** — typ wykresu. Jeżeli ten argument zostanie pominięty, użyty zostanie domyślny typ wykresu. Poszczególne typy wykresów są reprezentowane przez odpowiednie stałe (na przykład xlArea, xlColumnClustered i tak dalej).
- **Left** — pozycja wykresu w poziomie wyrażona w punktach, licząc od lewej. Jeżeli ten argument zostanie pominięty, Excel umieszcza wykres centralnie w poziomie na środku okna.
- **Top** — pozycja wykresu wyrażona w punktach, licząc od górnej krawędzi okna. Jeżeli ten argument zostanie pominięty, Excel umieszcza wykres centralnie w pionie na środku okna.
- **Width** — szerokość wykresu wyrażona w punktach. Jeżeli ten argument zostanie pominięty, Excel domyślnie przyjmuje wartość 354.
- **Height** — wysokość wykresu wyrażona w punktach. Jeżeli ten argument zostanie pominięty, Excel domyślnie przyjmuje wartość 210.
- **NewLayout** — kod numeryczny określający ogólny układ wykresu.

Polecenie przedstawione poniżej tworzy grupowany wykres kolumnowy, wykorzystujący styl 201 oraz układ 5, zlokalizowany 50 pikseli od lewej, 60 pikseli od góry okna, mający 300 pikseli szerokości i 200 pikseli wysokości.

```
ActiveSheet.Shapes.AddChart2 201, xlColumnClustered, 50, 60, 300, 200, 5
```

W wielu przypadkach korzystnym rozwiązaniem podczas tworzenia wykresu może być utworzenie odpowiedniej zmiennej obiektowej. Procedura, której kod zamieszczono poniżej, tworzy wykres liniowy, do którego możesz się odwoływać za pomocą zmiennej obiektowej o nazwie MyChart. Zwróć uwagę na to, że metoda AddChart2 zostaje wywołana tylko z dwoma argumentami; pozostałe pięć argumentów przyjmuje w tej sytuacji wartości domyślne.

```
Sub CreateChart()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.Shapes.AddChart2(212, xlLineMarkers).Chart
End Sub
```

Wykres bez danych nie jest zbyt użyteczny. Dane, które będą prezentowane na wykresie, możesz zdefiniować na dwa sposoby:

- Przed utworzeniem wykresu zaznacz komórki zawierające odpowiednie dane.
- Po utworzeniu wykresu użyj metody SetSourceData obiektu Chart.

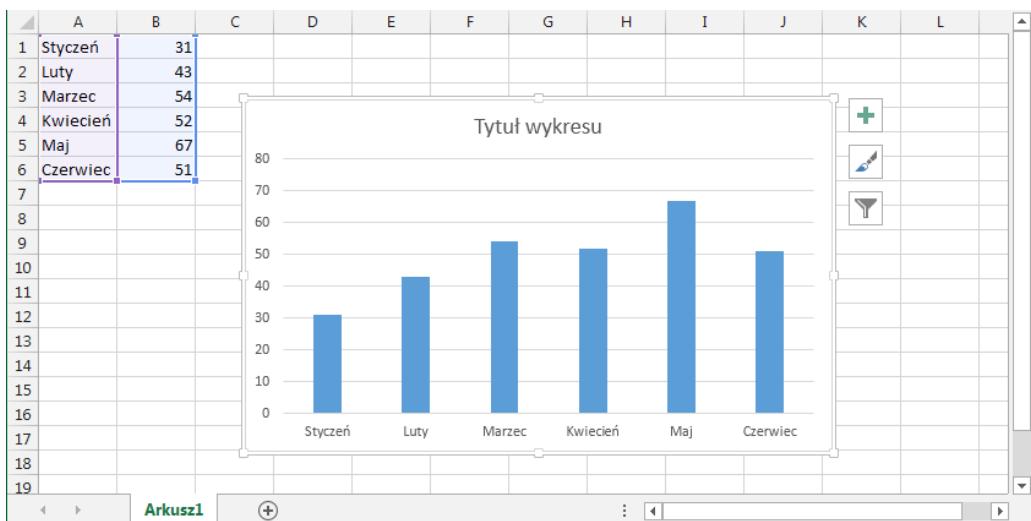
Poniżej zamieszczamy kod prostej procedury, która zaznacza zakres danych i następnie tworzy na ich podstawie wykres.

```
Sub CreateChart2()
    Range("A1:B6").Select
    ActiveSheet.Shapes.AddChart2 201, xlColumnClustered
End Sub
```

Procedura przedstawiona poniżej demonstruje zastosowanie metody SetSourceData. Procedura wykorzystuje dwie zmienne obiektowe: DataRange (dla obiektu Range przechowującego dane) oraz MyChart (dla obiektu Chart). Zmienna obiektowa MyChart jest tworzona w tym samym czasie co wykres.

```
Sub CreateChart3()
    Dim MyChart As Chart
    Dim DataRange As Range
    Set DataRange = ActiveSheet.Range("A1:B6")
    Set MyChart = ActiveSheet.Shapes.AddChart2.Chart
    MyChart.SetSourceData Source:=DataRange
End Sub
```

Procedura, której kod został zamieszczony powyżej, tworzy wykres przedstawiony na rysunku 16.1. Zwróć uwagę, że metoda AddChart2 została wywołana bez żadnych argumentów, stąd do tworzenia wykresu zostały użyte wartości domyślne.



Rysunek 16.1. Ten wykres został utworzony za pomocą kilku wierszy kodu VBA

Tworzenie wykresu na arkuszu wykresu

W poprzedniej sekcji omówiliśmy podstawową procedurę tworzenia wykresu osadzonego na arkuszu danych. Aby utworzyć wykres na arkuszu wykresu, powinieneś użyć metody Add2 kolekcji Charts. Metoda ta posiada kilka opcjonalnych argumentów, które określają pozycję wykresu na arkuszu wykresu, ale nie są bezpośrednio związane z samym wykresem.

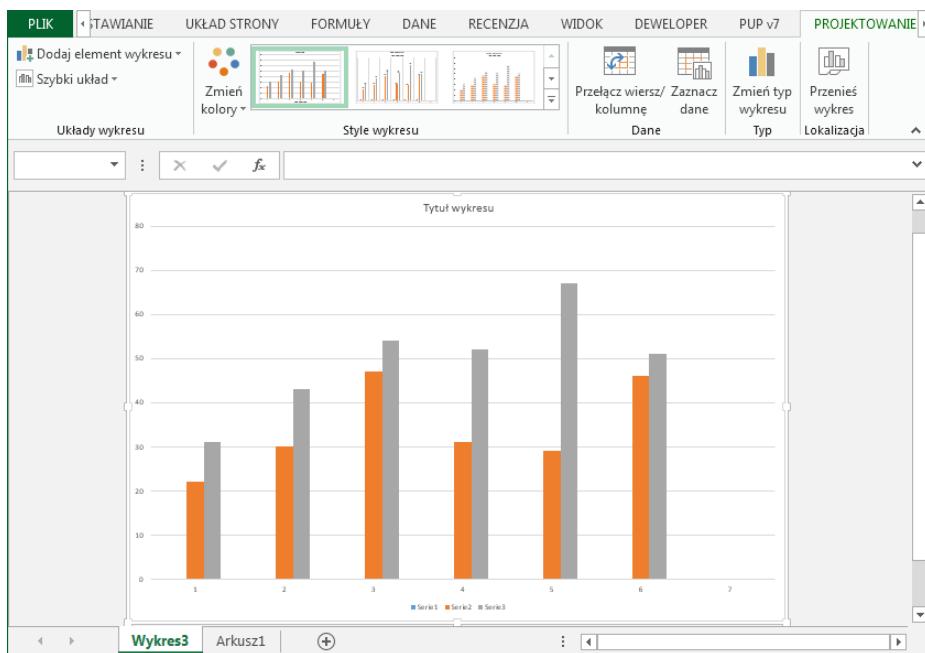
Kolejna procedura, której kod zamieszczono poniżej, tworzy wykres na arkuszu wykresu oraz definiuje zakres danych i typ wykresu.

```
Sub CreateChartSheet()
    Dim MyChart As Chart
    Dim DataRange As Range
    Set DataRange = ActiveSheet.Range("A1:C7")
    Set MyChart = Charts.Add2
    MyChart.SetSourceData Source:=DataRange
    ActiveChart.ChartType = xlColumnClustered
End Sub
```

Rezultat działania powyższej procedury został przedstawiony na rysunku 16.2.

Modyfikowanie wykresów

Nowe mechanizmy wprowadzone w Excelu 2013 powodują, że tworzenie i modyfikowanie wykresów stało się łatwiejsze niż kiedykolwiek wcześniej. Na przykład kiedy wykres zostanie uaktywniony, Excel wyświetla po jego prawej stronie trzy małe ikonki poleceń: *Elementy wykresu* (pozwala na dodawanie lub usuwanie różnych elementów wykresu), *Style wykresu* (za jego pomocą możesz zmienić styl wykresu i paletę kolorów) oraz *Filtры wykresu* (możesz go użyć do selektywnego wyświetlania serii danych).



Rysunek 16.2. Tworzenie wykresu na arkuszu wykresu

Za pomocą kodu VBA możesz wykonywać wszystkie operacje związane z wykresami. Na przykład jeżeli włączysz rejestrator i dodasz lub usuniesz jakiś element wykresu, przekonasz się, że takie operacje były wykonywane za pomocą metody SetElement obiektu Chart. Metoda SetElement pobiera jeden argument (poszczególne elementy są dostępne w postaci predefiniowanych wartości stałych). Na przykład aby dodać do aktywnego wykresu główne linie siatki poziomej, możesz użyć następującego polecenia:

```
ActiveChart.SetElement msoElementPrimaryValueGridLinesMajor
```

Żeby usunąć poziome linie siatki, powinieneś użyć następującego polecenia:

```
ActiveChart.SetElement msoElementPrimaryValueGridLinesNone
```

Listę wszystkich zdefiniowanych wartości stałych znajdziesz w pomocy systemowej (jeżeli lubisz eksperymentować, możesz zamiast tego użyć po prostu rejestratora makr).

Aby zmienić predefiniowany styl wykresu, możesz użyć właściwości ChartStyle. Poszczególne style są ponumerowane i tym razem nie ma dla nich żadnych predefiniowanych wartości stałych. Na przykład by zmienić styl aktywnego wykresu na 215, powinieneś użyć polecenia przedstawionego poniżej:

```
ActiveChart.ChartStyle = 215
```

Poprawnymi wartościami właściwości ChartStyle są liczby całkowite z zakresu od 1 do 48 oraz od 201 do 248. Ta druga grupa stylów składa się z nowych stylów, wprowadzonych w Excelu 2013. Pamiętaj również, że wygląd poszczególnych stylów wykresów nie jest spójny między wersjami Excela. Na przykład styl 48 w Excelu 2010 wygląda inaczej niż w wersji 2013.

Aby zmienić paletę kolorów wykorzystywaną przez aktywny wykres, możesz użyć właściwości ChartColor, ustawiając ją na wartość całkowitą z zakresu od 1 do 26. Na przykład:

```
ActiveChart.ChartColor = 12
```

Właściwość ChartColor to nowy element, który pojawił się w Excelu 2013.

Kiedy połączysz 96 stylów wykresów (ChartStyle) z 26 paletami kolorów (ChartColor), masz do dyspozycji 2496 kombinacji — co wydaje się w zupełności wystarczające do zaspokojenia gustów nawet najbardziej wybrednych użytkowników. Co więcej, jeśli takie predefiniowane opcje Ci nie wystarczają, masz pełną kontrolę nad każdym elementem wykresu. Na przykład fragment kodu VBA przedstawiony poniżej zmienia kolor wypełnienia jednego z punktów serii danych prezentowanej na wykresie:

```
With ActiveChart.FullSeriesCollection(1).Points(2).Format.Fill
    .Visible = msoTrue
    .ForeColor.ObjectThemeColor = msoThemeColorAccent2
    .ForeColor.TintAndShade = 0.4
    .ForeColor.Brightness = -0.25
    .Solid
End With
```

I znów, użycie rejestratora makr do zarejestrowania wprowadzanych do wykresu zmian z pewnością dostarczy Ci odpowiednich informacji na temat modelu obiektowego wykresów, które następnie będziesz mógł wykorzystać w swoich programach.

Wykorzystanie VBA do uaktywnienia wykresu

Kliknięcie wykresu osadzonego na arkuszu danych powoduje uaktywnienie tego wykresu. W kodzie VBA wbudowany wykres można uaktywnić za pomocą metody Activate. Oto przykład procedury VBA będącej odpowiednikiem kliknięcia wykresu osadzonego na arkuszu danych:

```
ActiveSheet.ChartObjects("Wykres1").Activate
```

Jeżeli wykres znajduje się na arkuszu wykresu, powinieneś zastosować następującą instrukcję:

```
Sheets("Wykres1").Activate
```

Zamiast tego możesz również aktywować wykres, wybierając przechowujący go obiekt Shape:

```
ActiveSheet.Shapes("Chart 1").Select
```

Do uaktywnionego wykresu można się odwoływać w kodzie przez właściwość ActiveChart (która zwraca obiekt typu Chart). Na przykład wykonanie poniższej instrukcji spowoduje wyświetlenie nazwy aktywnego wykresu. Jeżeli żaden wykres nie jest w danej chwili aktywny, instrukcja spowoduje wygenerowanie błędu:

```
MsgBox ActiveChart.Name
```

Aby dokonać modyfikacji wykresu za pomocą kodu VBA, nie trzeba go uaktywniać. Efekt wykonania dwóch procedur zamieszczonych poniżej jest dokładnie taki sam: zmiana typu wykresu osadzonego o nazwie *Wykres1* na wykres warstwowy. Pierwsza procedura uaktywnia wykres przed wykonaniem działań, druga tego nie robi.

```
Sub ModifyChart1()
    ActiveSheet.ChartObjects("Wykres1").Activate
    ActiveChart.Type = xlArea
End Sub

Sub ModifyChart2()
    ActiveSheet.ChartObjects("Wykres1").Chart.Type = xlArea
End Sub
```

Przenoszenie wykresu

Wykres osadzony można z łatwością przekształcić w arkusz wykresu. Aby to zrobić ręcznie, wystarczy uaktywnić wykres osadzony i wybrać polecenie *NARZĘDZIA WYKRESÓW/PROJEKTOWANIE/Lokalizacja/Przenieś wykres*. W oknie dialogowym *Przenoszenie wykresu* zaznacz opcję *Nowy arkusz* i podaj nazwę arkusza.

Wykres osadzony można przekształcić w arkusz wykresu również za pomocą kodu VBA. Poniżej zaprezentowano przykład, który przekształca pierwszy obiekt ChartObject osadzony na arkuszu o nazwie Arkusz1 na arkusz wykresu o nazwie MójWykres:

```
Sub MoveChart1()
    Sheets("Arkusz1").ChartObjects(1).Chart.
        Location xlLocationAsNewSheet, "MójWykres"
End Sub
```

Kod pokazany poniżej wykonuje dokładnie odwrotną czynność: przekształca arkusz wykresu o nazwie MójWykres na wykres osadzony na arkuszu danych o nazwie Arkusz1:

```
Sub MoveChart2()
    Charts("MójWykres").Location xlLocationAsObject, "Arkusz1"
End Sub
```

Metoda **Location** dodatkowo uaktywnia przeniesiony wykres.



Uwaga

Wykorzystanie VBA do dezaktywacji wykresu

Jak już wiesz, do aktywowania wykresu możesz użyć metody **Activate**, ale w jaki sposób dezaktywować wybrany wykres (czyli inaczej mówiąc, jak sprawić, aby dany wykres nie był już dłużej „zaznaczony”)?

O ile mi wiadomo, jedynym sposobem skutecznej dezaktywacji wykresu przy użyciu VBA jest zaznaczenie czegoś innego niż wykres. W przypadku wykresów osadzonych możesz użyć do tego celu właściwości **RangeSelection** obiektu **ActiveWindow** i zaznaczyć na przykład zakres komórek, który był zaznaczony przed aktywacją wykresu:

```
ActiveWindow.RangeSelection.Select
```

Jak się nazywasz?

Każdy obiekt `ChartObject` posiada swoją własną nazwę i podobnie każdy obiekt `Chart` w kolekcji `ChartObject` również posiada swoją nazwę. Takie rozwiązańe wydaje się oczywiste, ale mimo wszystko może być nieco mylące. Utwórz nowy wykres na arkuszu `Arkusz1` i aktywuj go. Następnie przejdź do okna `Immediate` i wpisz polecenia przedstawione poniżej (polecenia wpisywane przez użytkownika zostały wyróżnione pogrubioną czcionką):

```
? ActiveSheet.Shapes(1).Name  
Wykres1  
? ActiveSheet.ChartObjects(1).Name  
Wykres1  
? ActiveChart.Name  
Arkusz1 Wykres1  
? Activesheet.ChartObjects(1).Chart.Name  
Arkusz1 Wykres1
```

Jeżeli zmienisz nazwę arkusza, nazwa wykresu również ulegnie zmianie. Do zmiany nazwy obiektu `Chart` możesz użyć pola nazwy (które znajduje się po lewej stronie paska formuł w głównym oknie Excela). Możesz to zrobić również za pomocą odpowiedniego polecenia VBA:

```
Activesheet.ChartObjects(1).Name = "Nowa nazwa"
```

Nie możesz jednak zmienić nazwy obiektu `Chart` znajdującego się w kolekcji `ChartObject`. Jeżeli spróbujesz wykonać taką operację przy użyciu VBA, otrzymasz nieco tajemniczo brzmiący komunikat o braku dostępnej pamięci (ang. *out of memory*).

```
Activesheet.ChartObjects(1).Chart.Name = "Nowa nazwa"
```

Co dziwne, Excel pozwala na użycie nazwy innego istniejącego obiektu `ChartObject` — czyli krótko mówiąc, teoretycznie możesz utworzyć tuzin wykresów osadzonych na arkuszu i każdy z nich może nosić nazwę `Wykres1`. Jeżeli utworzysz kopię wykresu osadzonego, nowy wykres będzie miał taką samą nazwę jak wykres źródłowy.

Wnioski? Powinieneś zwracać szczególną uwagę na takie zachowanie Excela. Jeżeli okaże się, że Twoje makro VBA nie działa tak jak powinno, upewnij się, że na arkuszu nie znajdują się dwa wykresy o takich samych nazwach.

Aby zdezaktywować wykres osadzony na arkuszu wykresu, wystarczy utworzyć fragment kodu, który będzie aktywował inny arkusz.

Sprawdzanie, czy wykres został uaktywniony

Za pomocą makr można wykonać pewne typowe operacje na aktywnym wykresie (czyli na wykresie zaznaczonym przez użytkownika). Na przykład można zmienić typ wykresu, nadać mu wybrany styl formatowania czy eksportować wykres do postaci pliku graficznego.

Powstaje pytanie, w jaki sposób z poziomu kodu VBA sprawdzić, czy użytkownik zaznaczył wykres? Przez zaznaczenie rozumiemy uaktywnienie arkusza wykresu lub

kliknięcie wykresu osadzonego. Pierwsza myśl to sprawdzenie właściwości TypeName obiektu Selection, tak jak w poniższym wyrażeniu:

```
TypeName(Selection) = "Chart"
```

W praktyce takie wyrażenie nigdy nie przyjmuje wartości True. Kiedy wykres jest aktywny, właściwe zaznaczenie będzie dotyczyło innego obiektu zawartego w obiekcie Chart. Na przykład może to być obiekt Series, ChartTitle, Legend, PlotArea i tak dalej.

Rozwiązańiem problemu jest sprawdzenie, czy wartość ActiveChart jest równa Nothing. Jeżeli tak, oznacza to, że wykres nie jest aktywny. Fragment kodu zamieszczony poniżej sprawdza, czy wykres jest aktywny. Jeżeli nie, na ekranie wyświetlany jest odpowiedni komunikat i procedura kończy działanie.

```
If ActiveChart Is Nothing Then  
    MsgBox "Zaznacz wykres."  
Exit Sub  
Else  
    'inne polecenia procedury  
End If
```

Do sprawdzenia, czy wykres jest aktywny, wygodnie jest użyć funkcji VBA. Poniżej przedstawiamy kod funkcji ChartIsSelected, która zwraca wartość True, jeżeli wykres osadzony lub arkusz wykresu jest aktywny. Jeżeli nie, funkcja zwraca wartość False.

```
Private Function ChartIsSelected() As Boolean  
    ChartIsSelected = Not ActiveChart Is Nothing  
End Function
```

Usuwanie elementów z kolekcji ChartObjects lub Charts

Aby usunąć wykres z arkusza, musisz znać jego nazwę lub indeks w kolekcji ChartObject. Polecenie przedstawione poniżej usuwa obiekt ChartObject o nazwie Wykres 1 z aktywnego arkusza:

```
ActiveSheet.ChartObjects("Wykres 1").Delete
```

Pamiętaj, że wiele obiektów ChartObjects może mieć taką samą nazwę. Jeżeli tak się dzieje, możesz usunąć wybrany wykres za pomocą wartości jego indeksu w kolekcji:

```
ActiveSheet.ChartObjects(1).Delete
```

Aby usunąć wszystkie obiekty ChartObject z arkusza, powinieneś użyć metody Delete kolekcji ChartObjects:

```
ActiveSheet.ChartObjects.Delete
```

Możesz również usunąć wykres osadzony za pomocą metody Delete kolekcji Shapes. Polecenie przedstawione poniżej usuwa wykres o nazwie Wykres 1 z aktywnego arkusza:

```
ActiveSheet.Shapes("Wykres 1").Delete
```

Polecenie przedstawione poniżej usuwa wszystkie wykresy osadzone (oraz wszystkie inne kształty) z aktywnego arkusza:

```
Dim shp As Shape  
For Each shp In ActiveSheet.Shapes  
    shp.Delete  
Next shp
```

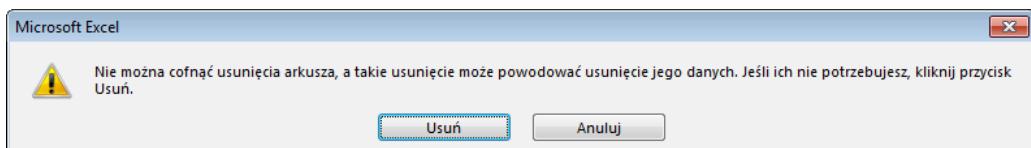
Aby usunąć wybrany arkusz wykresu, musisz znać jego nazwę lub indeks w kolekcji. Polecenie przedstawione poniżej usuwa arkusz wykresu o nazwie Wykres1:

```
Charts("Wykres1").Delete
```

Aby usunąć wszystkie arkusze wykresów z aktywnego skoroszytu, można wykorzystać następującą instrukcję:

```
ActiveWorkbook.Charts.Delete
```

Zwykle usuwanie arkusza powoduje wyświetlenie ostrzeżenia podobnego do pokazanego na rysunku 16.3. Aby makro kontynuowało działanie, użytkownik musi odpowiedzieć na zadane pytanie.



Rysunek 16.3. Próba usunięcia jednego lub kilku arkuszy wykresów powoduje wyświetlenie takiego komunikatu

Jeżeli usuwasz wykres za pomocą makra, możesz zrezygnować z wyświetlania tego ostrzeżenia. Aby je wyeliminować, należy użyć następującego ciągu instrukcji:

```
Application.DisplayAlerts = False  
ActiveWorkbook.Charts.Delete  
Application.DisplayAlerts = True
```

Przetwarzanie wszystkich wykresów w pętli

W niektórych przypadkach musisz wykonać określone operacje na wszystkich wykresach. Procedura przedstawiona poniżej wprowadza zmiany do wszystkich wykresów osadzonych na aktywnym arkuszu. W procedurze zastosowano pętlę, która przechodzi przez wszystkie elementy kolekcji ChartObjects i modyfikuje niektóre właściwości obiektu Chart poszczególnych elementów kolekcji.

```
Sub FormatAllCharts()  
    Dim ChtObj As ChartObject  
    For Each ChtObj In ActiveSheet.ChartObjects  
        With ChtObj.Chart  
            .ChartType = xlLineMarkers  
            .ApplyLayout 3  
            .ChartStyle = 12  
            .ClearToMatchStyle
```

```

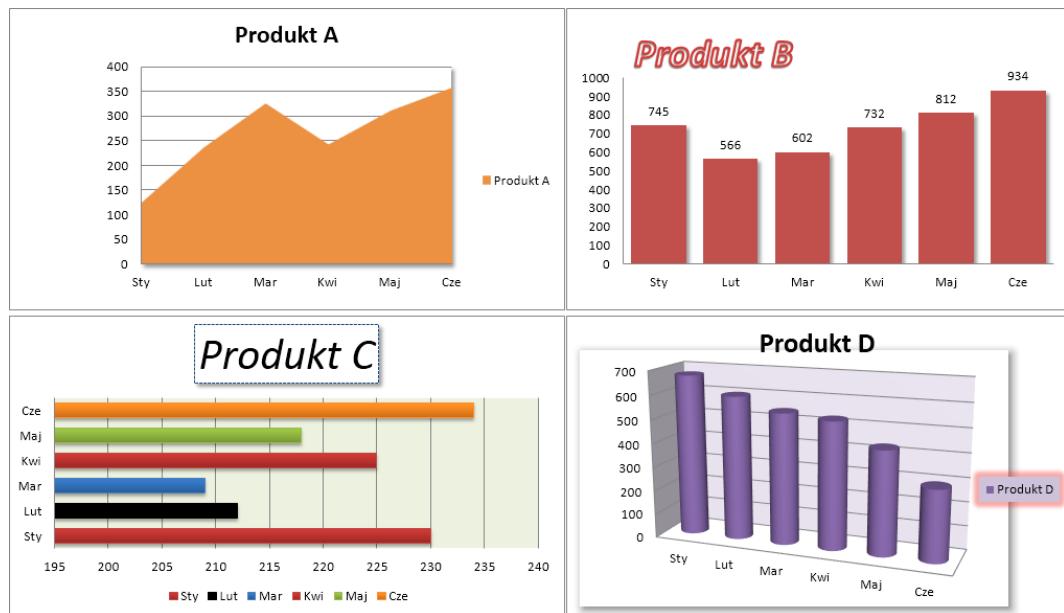
    .SetElement msoElementChartTitleAboveChart
    .SetElement msoElementLegendNone
    .SetElement msoElementPrimaryValueAxisTitleNone
    .SetElement msoElementPrimaryCategoryAxisTitleNone
    .Axes(xlValue).MinimumScale = 0
    .Axes(xlValue).MaximumScale = 1000
    With .Axes(xlValue).MajorGridlines.Format.Line
        .ForeColor.ObjectThemeColor = msoThemeColorBackground1
        .ForeColor.TintAndShade = 0
        .ForeColor.Brightness = -0.25
        .DashStyle = msoLineSysDash
        .Transparency = 0
    End With
    End With
    Next ChtObj
End Sub

```

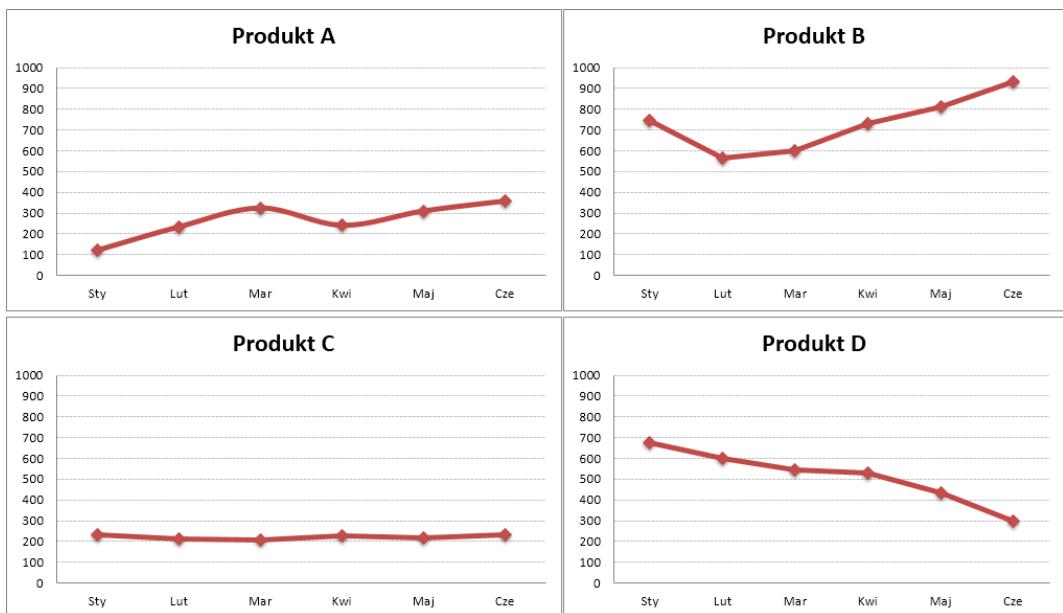


Skoroszyt z tym przykładem (*Formatowanie wszystkich wykresów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 16.4 przedstawiono cztery wykresy, które zostały sformatowane w różny sposób. Na rysunku 16.5 przedstawiono wygląd tych wykresów po uruchomieniu procedury FormatAllCharts.



Rysunek 16.4. Cztery wykresy sformatowane w różny sposób



Rysunek 16.5. Proste makro nadaje wszystkim wykresom jednolity sposób formatowania

Poniższe makro wykonuje to samo działanie, co poprzednie, ale działa dla wszystkich arkuszy wykresów w aktywnym skoroszycie:

```
Sub FormatAllCharts2()
    Dim cht as Chart
    For Each cht In ActiveWorkbook.Charts
        With cht
            .ChartType = xlLineMarkers
            .ApplyLayout 3
            .ChartStyle = 12
            .ClearToMatchStyle
            .SetElement msoElementChartTitleAboveChart
            .SetElement msoElementLegendNone
            .SetElement msoElementPrimaryValueAxisTitleNone
            .SetElement msoElementPrimaryCategoryAxisTitleNone
            .Axes(xlValue).MinimumScale = 0
            .Axes(xlValue).MaximumScale = 1000
            With .Axes(xlValue).MajorGridlines.Format.Line
                .ForeColor.ObjectThemeColor = msoThemeColorBackground1
                .ForeColor.TintAndShade = 0
                .ForeColor.Brightness = -0.25
                .DashStyle = msоЛineSysDash
                .Transparency = 0
            End With
        End With
    Next cht
End Sub
```

Zmiana rozmiarów i wyrównywanie obiektów ChartObject

Obiekt ChartObject posiada standardowe właściwości pozycji (Top oraz Left) i rozmiaru (Width oraz Height), do których można uzyskać dostęp z poziomu kodu VBA. Co ciekawe, na Wstążce programu Excel znajdują się formanty (karta *NARZĘDZIA WYKRESÓW/FORMATOWANIE*, grupa opcji *Rozmiar*) pozwalające na ustawienie właściwości Width oraz Height, ale nie ma formantów dla właściwości Top oraz Left.

Kolejna procedura, której kod zamieszczono poniżej, tak zmienia rozmiar wszystkich obiektów ChartObject w arkuszu, aby były takie same jak wymiary aktywnego wykresu. Dodatkowo zmienia układ obiektów ChartObject w taki sposób, że są one wyświetlane w zdefiniowanej przez użytkownika liczbie kolumn.

```
Sub SizeAndAlignCharts()
    Dim W As Long, H As Long
    Dim TopPosition As Long, LeftPosition As Long
    Dim ChtObj As ChartObject
    Dim i As Long, NumCols As Long

    If ActiveChart Is Nothing Then
        MsgBox "Wybierz wykres bazowy dla zmiany rozmiarów pozostałych wykresów"
        Exit Sub
    End If

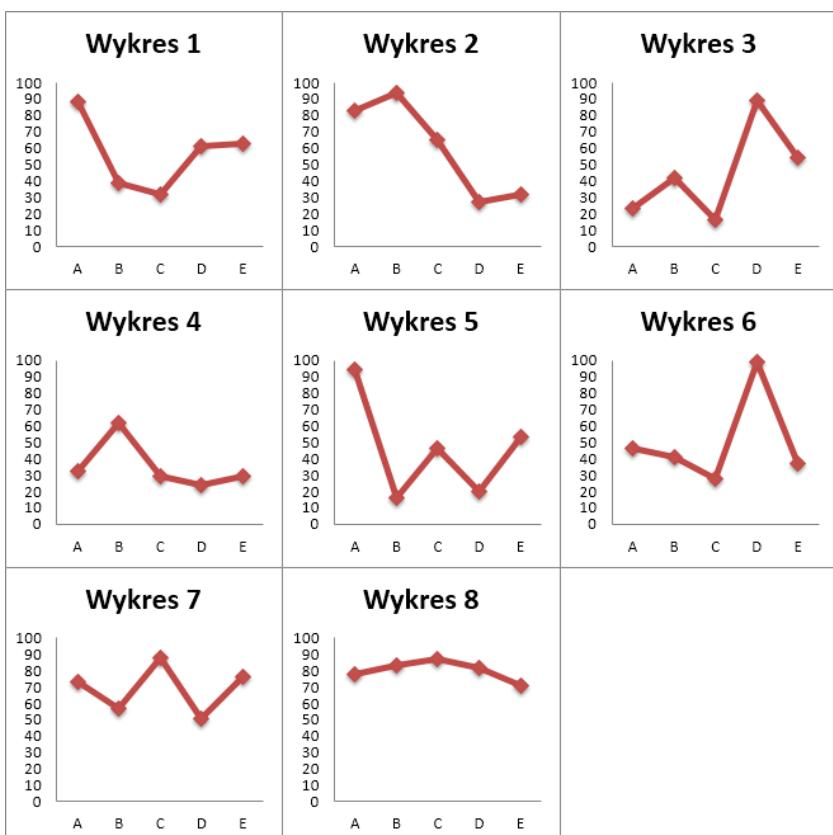
    'Pobierz liczbę kolumn
    On Error Resume Next
    NumCols = InputBox("Podaj liczbę kolumn do wyświetlania wykresów?")
    If Err.Number <> 0 Then Exit Sub
    If NumCols < 1 Then Exit Sub
    On Error GoTo 0

    'Pobierz rozmiar aktywnego wykresu
    W = ActiveChart.Parent.Width
    H = ActiveChart.Parent.Height

    'Jeżeli to konieczne, zmień początkowe pozycje wykresów
    TopPosition = 100
    LeftPosition = 20
    For i = 1 To ActiveSheet.ChartObjects.Count
        With ActiveSheet.ChartObjects(i)
            .Width = W
            .Height = H
            .Left = LeftPosition + ((i - 1) Mod NumCols) * W
            .Top = TopPosition + Int((i - 1) / NumCols) * H
        End With
    Next i
End Sub
```

Jeżeli żaden wykres nie jest aktywny, procedura prosi użytkownika o zaznaczenie wybrane go wykresu, który będzie użyty jako baza do zmiany rozmiarów pozostałych wykresów. Do pobrania liczby kolumn użyta została funkcja InputBox. Wartości właściwości Left i Top są obliczane w pętli.

Na rysunku 16.6 przedstawiono kilka wykresów po aranżacji położenia i dopasowaniu rozmiarów.



Rysunek 16.6. Zastosowanie makra VBA pozwala na szybką zmianę rozmiaru i położenia osadzonych wykresów



Skoroszyt z tym przykładem (Zmiana rozmiaru i wyrównania wykresów.xlsx) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Tworzenie dużej liczby wykresów

Przykład, który będziemy omawiać w tym podrozdziale, demonstruje sposób automatyzacji procesu tworzenia dużej liczby wykresów. Na rysunku 16.7 zaprezentowano fragment zestawu danych, które powinny zostać przedstawione na wykresach. Arkusz zawiera dane dla 50 osób, a naszym celem jest utworzenie dla nich 50 wykresów, które będą jednolicie sformatowane i ładnie wyrównane.

Rysunek 16.7.

Każdy wiersz będzie spełniał rolę serii danych dla osobnego wykresu

	A	B	C	D	E	F	G
1	Nazwisko i imię	Dzień 1	Dzień 2	Dzień 3	Dzień 4	Dzień 5	
2	Adamczyk Jakub	16	13	99	5	80	
3	Adamczyk Katarzyna	70	40	82	22	60	
4	Baran Kacper	79	17	25	19	40	
5	Górcka Daria	52	82	86	77	74	
6	Górski Filip	13	60	28	24	67	
7	Grabowska Alicja	12	75	86	69	35	
8	Jabłoński Bartosz	93	13	89	99	32	
9	Jaworska Dominika	30	14	49	77	73	
10	Kaczmarek Dawid	79	30	36	13	81	
11	Kaczmarek Gabriela	99	14	100	82	49	
12	Kamińska Amelia	11	45	20	91	9	
13	Kamiński Adam	72	81	80	84	2	
14	Kowalski Jan	96	62	11	3	93	
15	Kozłowski Igor	92	70	11	38	5	
16	Krawczyk Hanna	52	39	82	64	94	
17	Lewandowski Dominik	84	75	73	0	87	
18	Malinowski Antoni	25	49	76	69	0	
19	Mazur Anna	39	10	45	41	10	
20	Michałak Kamil	6	96	54	51	0	
21	Michalski Aleksander	40	97	69	42	88	
22	Nowak Julia	45	91	12	26	6	
23	Nowakowska Klaudia	25	81	64	12	89	
24	Olszewski Karol	61	53	86	72	9	
25	Ostrowski Franciszek	1	69	63	74	63	

Aby to zrobić, rozpoczniemy od napisania procedury CreateChart, która będzie pobierała następujące argumenty:

- rng — zakres, który będzie użyty jako seria danych dla wykresu.
- l — odległość wykresu od lewej krawędzi okna, wyrażona w pikselach.
- t — odległość wykresu od górnej krawędzi okna, wyrażona w pikselach.
- w — szerokość wykresu.
- h — wysokość wykresu.

Procedura CreateChart wykorzystuje wymienione wyżej argumenty do utworzenia wykresu liniowego, którego osie będą wyskalowane w zakresie od 0 do 100.

```
Sub CreateChart(rng, l, t, w, h)
    With Worksheets("Arkusz2").Shapes
        AddChart2(332, xlLineMarkers, l, t, w, h).Chart
            .SetSourceData Source:=rng
            .Axes(xlValue).MinimumScale = 0
            .Axes(xlValue).MaximumScale = 100
    End With
End Sub
```

Kiedy byłem już usatysfakcjonowany sposobem działania tej procedury, napisałem kolejną procedurę, o nazwie Make50Charts, która za pomocą pętli For ... Next 50 razy wywołuje procedurę CreateChart. Zwróć uwagę, że zestaw danych dla każdego z wykresów składa się z pierwszego wiersza (nagłówka) oraz odpowiednich serii danych znajdujących się

w wierszach od 2. do 51. Do połączenia tych dwóch zakresów komórek w jeden obiekt Range użyłem metody Union. Obiekt Range jest następnie przekazywany do procedury CreateChart. Kolejnym wyzwaniem było napisanie kodu, który będzie określał współrzędne lewego górnego rogu każdego z wykresów.

```
Sub Make50Charts()
    Dim ChartData As Range
    Dim i As Long
    Dim leftPos As Long, topPos As Long
    ' Usuń wszystkie wykresy (jeżeli istnieją)
    With Worksheets("Arkusz2").ChartObjects
        If .Count > 0 Then .Delete
    End With

    ' Inicjalizowanie pozycji
    leftPos = 0
    topPos = 0

    ' Przechodzenie w pętli przez kolejne zestawy danych
    For i = 2 To 51
        ' Określenie zakresu danych
        With Worksheets("Arkusz1")
            Set ChartData = Union(.Range("A1:F1"), _
                .Range(.Cells(i, 1), .Cells(i, 6)))
        End With

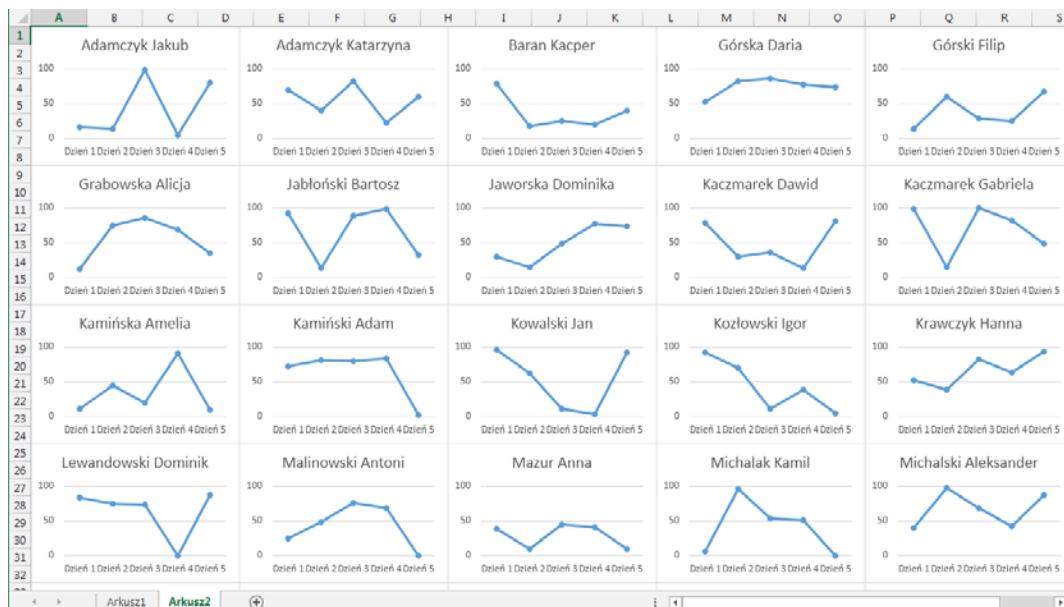
        ' Tworzenie wykresu
        Call CreateChart(ChartData, leftPos, topPos, 180, 120)

        ' Dopasowanie pozycji
        If (i - 1) Mod 5 = 0 Then
            leftPos = 0
            topPos = topPos + 120
        Else
            leftPos = leftPos + 180
        End If
    Next i
End Sub
```

Na rysunku 16.8 przedstawiono część z 50 wykresów automatycznie wygenerowanych za pomocą przedstawionego kodu VBA.

Eksportowanie wykresów

W niektórych przypadkach niezbędne może się okazać zapisanie wykresu Excela w postaci pliku graficznego — na przykład kiedy chcesz zamieścić wykres na stronie internetowej. Jednym z rozwiązań jest użycie programu do tworzenia zrzutów ekranu i skopiowanie obrazu wyświetlanego na ekranie monitora. Innym rozwiązaniem jest napisanie prostej procedury VBA.



Rysunek 16.8. Szereg wykresów utworzonych automatycznie przez makro VBA

Procedura, której kod zamieszczono poniżej, używa metody Export obiektu Chart do zapisania wykresu na dysku w postaci pliku *GIF*.

```

Sub SaveChartAsGIF()
    Dim Fname as String
    If ActiveChart Is Nothing Then Exit Sub
    Fname = ThisWorkbook.Path & "\" & ActiveChart.Name & ".gif"
    ActiveChart.Export FileName:=Fname, FilterName:="GIF"
End Sub

```

Argument *FilterName* może również przyjmować wartości "JPEG" lub "PNG". W większości przypadków pliki w formatach *GIF* lub *PNG* wyglądają lepiej niż *JPEG*. W systemie pomocy programu Excel znajdziesz informację o tym, że metoda Export posiada również trzeci argument: *Interactive*. Jeżeli argument ten przyjmie wartość True, na ekranie pojawi się okno dialogowe, w którym będziesz mógł wybrać parametry eksportu. W praktyce jednak podanie tego argumentu nie ma żadnego wpływu na działanie metody.

Pamiętaj również, że działanie metody Export zakończy się błędem, jeżeli użytkownik nie poda poprawnego formatu zapisu pliku graficznego. Odpowiednie filtry są instalowane podczas instalacji pakietu Office (lub samego Excela).

Eksportowanie wszystkich obiektów graficznych

Jednym ze sposobów eksportu wszystkich obiektów graficznych znajdujących się w skoroszycie jest zapisanie skoroszytu w formacie HTML. Wykonanie takiej operacji spowoduje utworzenie osobnego katalogu, w którym w formacie GIF lub PNG zostaną zapisane wszystkie obiekty graficzne, takie jak wykresy, kształty, obiekty typu clip-art, a nawet skopiowane obrazy zakresów, które możesz umieścić na arkuszu za pomocą polecenia *NARZĘDZIA GŁÓWNE/Schowek/Wklej/Obraz (O)*.

Poniżej zamieszczono kod procedury, która automatyzuje ten proces. Procedura działa z aktywnym skoroszytem.

```
Sub SaveAllGraphics()
    Dim FileName As String
    Dim TempName As String
    Dim DirName As String
    Dim gFile As String

    FileName = ActiveWorkbook.FullName
    TempName = ActiveWorkbook.Path & "\" & _
        ActiveWorkbook.Name & "graphics.htm"
    DirName = Left(TempName, Len(TempName) - 4) & "_files"

    ' Zapisz aktywny skoroszyt w formacie HTML i ponownie otwórz oryginalny plik
    ActiveWorkbook.Save
    ActiveWorkbook.SaveAs FileName:=TempName, FileFormat:=xlHtml
    Application.DisplayAlerts = False
    ActiveWorkbook.Close
    Workbooks.Open FileName

    ' Usuń plik HTML
    Kill TempName

    ' Usuń z katalogu HTML wszystkie pliki oprócz *.PNG
    gFile = Dir(DirName & "\*.*")
    Do While gFile <> ""
        If Right(gFile, 3) <> "png" Then Kill DirName & "\" & gFile
        gFile = Dir
    Loop

    ' Wyświetl wyeksportowane pliki graficzne
    Shell "explorer.exe" & DirName, vbNormalFocus
End Sub
```

Procedura rozpoczyna działanie od zapisania aktywnego skoroszytu. Następnie zapisuje skoroszyt w formacie *HTML*, zamienia plik i ponownie otwiera oryginalny skoroszyt. Dalej procedura usuwa plik *HTML*, ponieważ zależy nam tylko na folderze utworzonym podczas tego procesu (w tym folderze znajdują się wszystkie obiekty graficzne zapisane w plikach graficznych). Następnie procedura przechodzi w pętlę przez pliki w tym folderze i usuwa wszystkie pliki inne niż *PNG*. Na koniec procedura wywołuje funkcję *Shell*, która wyświetla zawartość foldera.



Więcej szczegółowych informacji na temat poleceń przeznaczonych do pracy z plikami znajdziesz w rozdziale 25.



Skoroszyt z tym przykładem (*Eksport wszystkich obiektów graficznych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Zmiana danych prezentowanych na wykresie

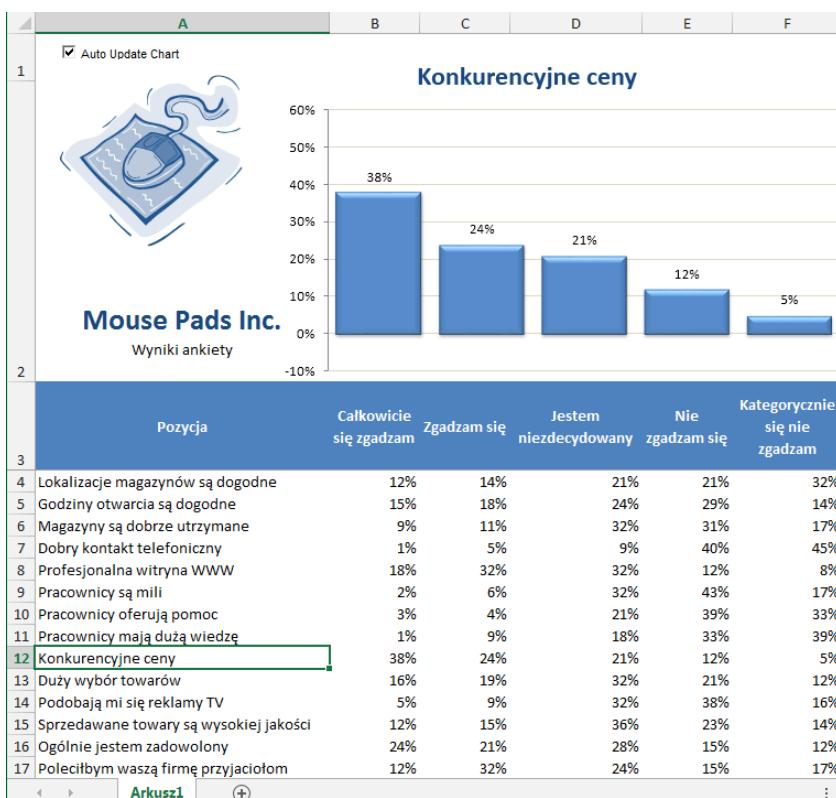
Przykłady zaprezentowane do tej pory w tym rozdziale wykorzystywały do określania zakresu danych prezentowanych na wykresie właściwość `SourceData`. W wielu przypadkach zachodzi jednak konieczność zmiany zakresu danych dla wybranych serii danych. Aby to zrobić, powinieneś skorzystać z właściwości `Values` obiektu `Series`. Obiekt ten posiada również właściwość `XValues`, przechowującą informacje o wartościach na osi poziomej.



Właściwość `Values` odpowiada trzeciemu argumentowi formuły SERIE, a właściwość `XValues` odpowiada drugiemu argumentowi tej formuły. Więcej szczegółowych informacji na ten temat znajdziesz w ramce „Formuła SERIE”.

Modyfikacja danych wykresu na podstawie aktywnej komórki

Na rysunku 16.9 pokazano wykres utworzony na podstawie danych z wiersza aktywnej komórki. Kiedy użytkownik przesunie wskaźnik komórki, wykres zostanie automatycznie uaktualniony.



Rysunek 16.9. Ten wykres zawsze wyświetla dane na podstawie wiersza odpowiadającego aktywnej komórce

W przykładzie użyto procedury obsługi zdarzenia dla obiektu Sheet1. Kiedy użytkownik zmienia zaznaczenie, przenosząc wskaźnik komórki, zachodzi zdarzenie SelectionChange. Procedura obsługi tego zdarzenia (umieszczona w module kodu obiektu Arkusz1) jest następująca:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Excel.Range)
    If CheckBox1 Then Call UpdateChart
End Sub
```

Za każdym razem, kiedy użytkownik przemieści wskaźnik do innej komórki, wykonywana jest procedura Worksheet_SelectionChange. Jeżeli opcja *Automatyczna aktualizacja wykresu* jest zaznaczona (formant ActiveX umieszczony na arkuszu), procedura obsługi zdarzenia wywołuje procedurę UpdateChart, której kod został przedstawiony poniżej:

```
Sub UpdateChart()
    Dim ChtObj As ChartObject
    Dim UserRow As Long
    Set ChtObj = ActiveSheet.ChartObjects(1)
    UserRow = ActiveCell.Row
    If UserRow < 4 Or IsEmpty(Cells(UserRow, 1)) Then
        ChtObj.Visible = False
    Else
        ChtObj.Chart.SeriesCollection(1).Values =
            Range(Cells(UserRow, 2), Cells(UserRow, 6))
        ChtObj.Chart.ChartTitle.Text = Cells(UserRow, 1).Text
        ChtObj.Visible = True
    End If
End Sub
```

Zmienna UserRow zawiera numer wiersza aktywnej komórki. Instrukcja If sprawdza, czy aktywna komórka znajduje się w wierszu z danymi (dane rozpoczynają się od wiersza numer 4). Jeżeli wskaźnik aktywnej komórki znajduje się w wierszu, który nie zawiera danych, obiekt ChartObject zostaje ukryty, a zamiast niego wyświetlany jest komunikat *Nie mogę wyświetlić wykresu*. Jeżeli w wierszu znajdują się dane do wyświetlenia, procedura ustawia właściwość Values obiektu Serie na zakres kolumn 2 – 6 aktywnego wiersza oraz nadaje obiekowi ChartTitle wartość reprezentującą tekst w kolumnie A.



Skoroszyt z tym przykładem (*Wykres aktywnej komórki.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zastosowanie języka VBA do identyfikacji zakresu danych prezentowanych na wykresie

W poprzednim przykładzie przedstawiliśmy sposób użycia właściwości Values obiektu Series do zdefiniowania danych prezentowanych na wykresie. W tym podrozdziale omówiono zastosowanie makr VBA do identyfikacji zakresów serii danych prezentowanych na wykresie, kiedy na przykład trzeba zwiększyć serię danych poprzez dodanie do zakresu nowej komórki.

Formuła SERIE

Dane wykorzystane w każdej serii wykresu określa formuła SERIE, która pojawia się na pasku formuły po zaznaczeniu serii danych na wykresie. Nie jest to formuła w pełnym tego słowa znaczeniu; mówiąc inaczej, nie można użyć jej w komórce i odwrotnie, w formule SERIE nie można wykorzystać funkcji arkuszowych. Można jednak modyfikować jej argumenty.

Formuła SERIE ma następującą składnię:

=SERIE(*nazwa_serii, etykiety_kategorii, wartości, kolejność, rozmiary*)

Argumenty formuły SERIE opisano poniżej:

- *nazwa_serii* (opcjonalny) — odwołanie do komórki zawierającej nazwę serii do wykorzystania w legendzie. Jeżeli wykres składa się tylko z jednej serii danych, argument ten jest wykorzystywany jako tytuł. Można go także podać jako tekst ujęty w cudzysłów. Jeżeli argument zostanie pominięty, Excel utworzy domyślną nazwę serii (np. Seria 1).
- *etykiety_kategorii* (opcjonalny) — odwołanie do zakresu zawierającego etykiety dla osi kategorii. Jeżeli argument zostanie pominięty, Excel zastosuje kolejne liczby całkowite, począwszy od 1. Dla wykresów XY ten argument definiuje etykiety dla osi X. Poprawne jest także odwołanie do nieciągłego zakresu. W takim przypadku adresy zakresów powinny być oddzielone przecinkami i ujęte w nawiasy. Argument można także zdefiniować w postaci tablicy wartości oddzielonych przecinkami (lub tekstu w cudzysłach) ujętych w nawiasy klamrowe.
- *wartości* (wymagany) — odwołanie do zakresu zawierającego wartości dla serii. Dla wykresów XY ten argument definiuje wartości dla osi Y. Poprawne jest także odwołanie do nieciągłego zakresu. W takim przypadku adresy zakresów powinny być oddzielone przecinkami i ujęte w nawiasy. Argument można także zdefiniować w postaci tablicy wartości oddzielonych przecinkami, ujętych w nawiasy klamrowe.
- *kolejność* (wymagany) — liczba całkowita określająca kolejność wykreślania serii. Ten argument ma znaczenie tylko wtedy, gdy wykres składa się z więcej niż jednej serii. Zastosowanie odwołania do komórki nie jest dozwolone.
- *rozmiary* (tylko dla wykresów bąbelkowych) — odwołanie do zakresu zawierającego wartości rozmiarów bąbelków w wykresach bąbelkowych. Poprawne jest także odwołanie do nieciągłego zakresu. W takim przypadku adresy zakresów powinny być oddzielone przecinkami i ujęte w nawiasy. Argument można także zdefiniować w postaci tablicy wartości ujętych w nawiasy klamrowe.

Odwołania do zakresów w formule SERIE zawsze są bezwzględne i zawsze zawierają nazwę arkusza. Oto przykład:

=SERIE(Arkusz1!\$B\$1,,Arkusz1!\$B\$2:\$B\$7,1)

Odwołanie do zakresu może składać się z obszaru nieciągłego. W takim przypadku poszczególne zakresy składowe należy oddzielić średnikami i ująć argument w nawiasy. W poniższym przykładzie formuły SERIE wartości należą do zakresu B2:B3 oraz B5:B7:

=SERIE(,,(Arkusz1!\$B\$2:\$B\$3,Arkusz1!\$B\$5:\$B\$7),1)

Odwołania do zakresów można zastąpić nazwami zakresów. Jeżeli wybierzesz takie rozwiązanie (a nazwy zostały zdefiniowane na poziomie skoroszytu), Excel zmodyfikuje odwołania, wprowadzając w formule SERIE nazwę skoroszytu. Oto przykład:

=SERIE(Arkusz1!\$B\$1,,budżet.xlsx!MojeDane, 1)

Poniżej zamieszczono opis trzech właściwości, które będą potrzebne do wykonania tego zadania.

- Właściwość `Formula` — zwraca lub ustawia formułę SERIE dla serii danych. Po zaznaczeniu serii danych na wykresie w pasku formuły wyświetla się formuła SERIE. Właściwość `Formula` zwraca tę formułę jako łańcuch znaków.
- Właściwość `Values` — zwraca lub ustawia kolekcję wszystkich wartości w serii. Można ją podać jako zakres danych w arkuszu lub tablicę stałych, ale nie można użyć kombinacji obu tych sposobów.
- Właściwość `XValues` — zwraca lub ustawia tablicę wartości osi X dla serii danych wykresu. Właściwość `XValues` można podać jako zakres danych w arkuszu lub tablicę stałych, ale nie można użyć kombinacji obu tych sposobów. Właściwość `XValues` może także być pusta.

Wydawać by się mogło, że zakres danych wykorzystywany przez wybraną serię danych na wykresie, potrzebny w makrze VBA, można uzyskać na podstawie właściwości `Values` obiektu `Series`. Podobnie mógłbyś również założyć, że wartości osi X (etykiety kategorii) można uzyskać z właściwości `XValues`. Teoretycznie taki sposób rozumowania *wydaje się* być prawidłowy. W praktyce jednak tak nie jest.

Kiedy ustawiasz właściwość `Values` obiektu `Series`, możesz określić obiekt `Range` lub tablicę. Jednak odczytywanie tej właściwości zawsze zwraca tablicę. Niestety model obiektowy wykresu nie pozwala na odczytanie obiektu `Range` wykorzystywanego przez obiekt `Series`.

Jednym ze sposobów rozwiązymania tego problemu jest napisanie kodu, który analizuje formułę SERIE i wydobywa z niej adresy zakresów. Brzmi to dosyć prosto, ale w rzeczywistości jest to trudne zadanie — ze względu na złożoność formuły SERIE. Poniżej podano kilka przykładów poprawnych formuł SERIE:

```
=SERIE(Arkusz1!$B$1,Arkusz1!$A$2:$A$4,Arkusz1!$B$2:$B$4,1)  
=SERIE(,,Arkusz1!$B$2:$B$4,1)  
=SERIE(,Arkusz1!$A$2:$A$4,Arkusz1!$B$2:$B$4,1)  
=SERIE("Podsumowanie sprzedaży",,Arkusz1!$B$2:$B$4,1)  
=SERIE(,"Sty","Lut","Mar", Arkusz1!$B$2:$B$4,1)  
=SERIE(,(Arkusz1!$A$2,Arkusz1!$A$4),(Arkusz1!$B$2,Arkusz1!$B$4),1)  
=SERIE(Arkusz1!$B$1,Arkusz1!$A$2:$A$4,Arkusz1!$B$2:$B$4,1,Arkusz1!$C$2:$C$4)
```

Jak widać, formuła SERIE nie zawsze musi posiadać wszystkie argumenty, może używać tablic, a nawet adresów nieciągłych zakresów komórek. Aby było jeszcze trudniej, dla wykresów bąbelkowych istnieje dodatkowy argument (np. w ostatniej formule SERIE na poprzedniej liście). Próba przeanalizowania składni argumentów z pewnością nie jest trywialnym zadaniem nawet dla doświadczonego programisty.

Spędziłem nad tym problemem wiele czasu, aż wreszcie udało mi się znaleźć rozwiązanie. Cała sztuczka polega na analizie formuły SERIE za pomocą funkcji pomocniczej, która przyjmuje argumenty formuły SERIE i zwraca tablicę elementów o wymiarach 2×5 , zawierającą wszystkie informacje o formule.

Później uproszcilem to rozwiązywanie, tworząc cztery nowe funkcje VBA, z których każda przyjmuje jeden argument (odwołanie do obiektu Series) i zwraca tablicę dwuelementową. Są to następujące funkcje:

- SERIESNAME_FROM_SERIES — pierwszy element tablicy zawiera łańcuch opisujący typ danych pierwszego argumentu formuły SERIE (Range, Empty lub String), natomiast drugi element zawiera adres zakresu, pusty łańcuch znaków lub łańcuch znaków.
- XVALUES_FROM_SERIES — pierwszy element zawiera łańcuch opisujący typ danych drugiego argumentu formuły SERIE (Range, Array, Empty lub String), natomiast drugi — adres zakresu, tablicę, pusty łańcuch znaków lub łańcuch znaków.
- VALUES_FROM_SERIES — pierwszy element zawiera łańcuch opisujący typ danych trzeciego argumentu formuły SERIE (Range lub Array), natomiast drugi — adres zakresu lub tablice.
- BUBBLESIZE_FROM_SERIES — pierwszy element tablicy zawiera łańcuch opisujący typ danych piątego argumentu formuły SERIE (Range, Array lub Empty), natomiast drugi — adres zakresu, tablicę lub pusty łańcuch znaków. Ta funkcja działa tylko dla wykresów bąbelkowych.

Warto zwrócić uwagę, że nie ma funkcji pobierającej czwarty argument formuły SERIE (kolejność wykreślania). Ten argument można uzyskać bezpośrednio za pomocą właściwości PlotOrder obiektu Series.



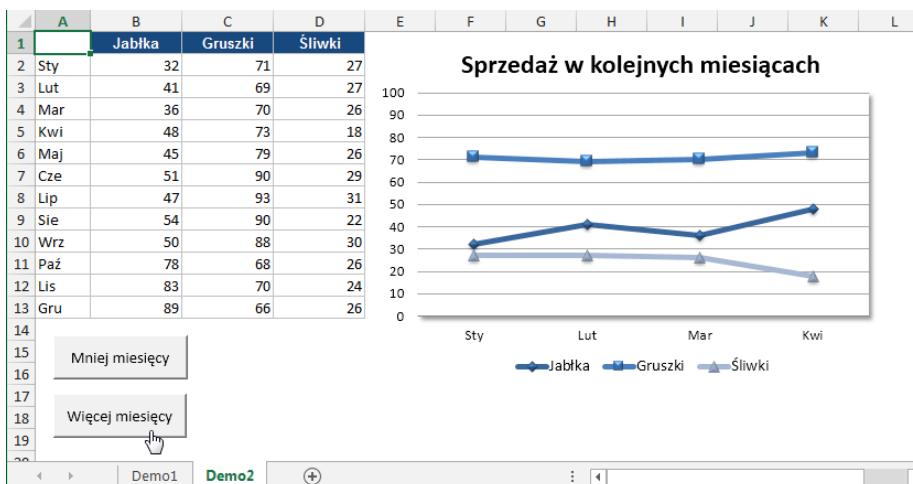
Kod VBA wymienionych funkcji jest zbyt obszerny, aby go tu zamieścić, ale znajdziesz go na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>), w pliku o nazwie *Pobieranie zakresów serii.xlsm*. Kod jest szczegółowo udokumentowany, co umożliwia łatwe przystosowanie go do własnych potrzeb.

Na poniższym przykładzie zademonstrowano działanie funkcji VALUES_FROM_SERIES, która wyświetla adres zakresu wartości dla pierwszej serii danych aktywnego wykresu:

```
Sub ShowValueRange()
    Dim Ser As Series
    Dim x As Variant
    Set Ser = ActiveChart.SeriesCollection(1)
    x = VALUES_FROM_SERIES(Ser)
    If x(1) = "Range" Then
        MsgBox Range(x(2)).Address
    End If
End Sub
```

Zmienna x została zdefiniowana jako Variant i służy do przechowywania dwuelementowej tablicy zwróconej przez funkcję VALUES_FROM_SERIES. Pierwszy element tablicy x zawiera łańcuch znaków opisujący typ danych. W przypadku, gdy jest to łańcuch "Range", w oknie informacyjnym wyświetli się adres zakresu zawartego w drugim elemencie tablicy x.

Na rysunku 16.10 zaprezentowano inny przykład. Wykres zawiera trzy serie danych. Przyciski w arkuszu powodują wykonywanie makr, które rozszerzają i zawiązują poszczególne zakresy danych.



Rysunek 16.10. W tym skoroszytce pokazano, w jaki sposób rozszerzyć lub zwiększyć serie danych za pomocą makra VBA

Kod procedury ContractAllSeries zamieszczono poniżej. Jej działanie polega na przetwarzaniu w pętli kolekcji SeriesCollection i obliczaniu zakresów danych za pomocą funkcji XVALUES_FROM_SERIES oraz VALUES_FROM_SERIES. Następnie zakresy są zmniejszane za pomocą metody Resize.

```
Sub ContractAllSeries()
    Dim s As Series
    Dim Result As Variant
    Dim DRange As Range
    For Each s In ActiveSheet.ChartObjects(1).Chart.SeriesCollection
        Result = XVALUES_FROM_SERIES(s)
        If Result(1) = "Range" Then
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
                Set DRange = DRange.Resize(DRange.Rows.Count - 1)
                s.XValues = DRange
            End If
        End If
        Result = VALUES_FROM_SERIES(s)
        If Result(1) = "Range" Then
            Set DRange = Range(Result(2))
            If DRange.Rows.Count > 1 Then
                Set DRange = DRange.Resize(DRange.Rows.Count - 1)
                s.Values = DRange
            End If
        End If
    Next s
End Sub
```

Procedura ExpandAllSeries jest bardzo podobna. Jej wykonanie powoduje rozszerzenie wszystkich zakresów o jedną komórkę.

Wykorzystanie VBA do wyświetlania dowolnych etykiet danych na wykresie

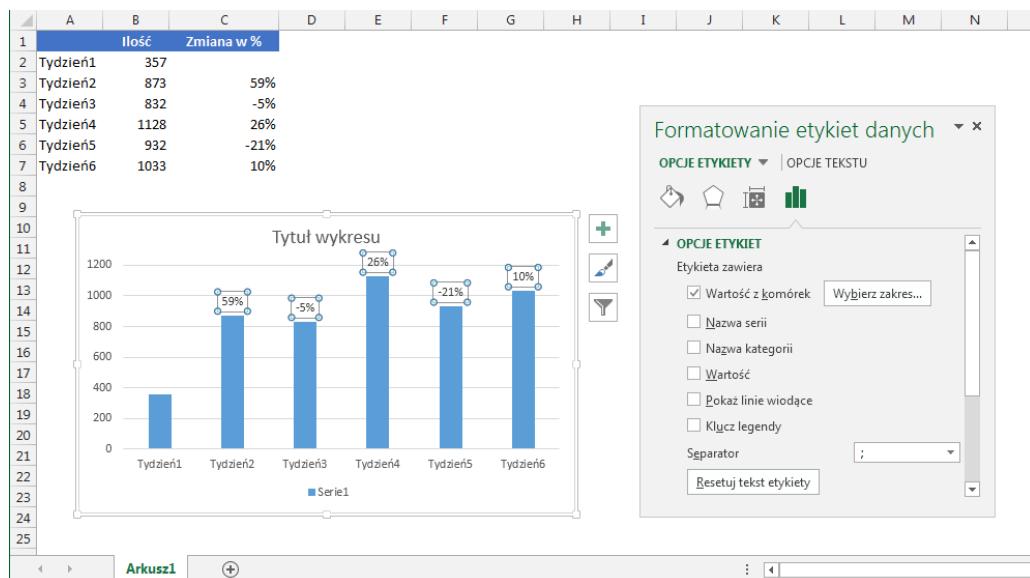


Jeżeli korzystasz wyłącznie z Excela 2013, to chciałbym z przyjemnością zakomunikować, że informacje przedstawione w tym podrozdziale są już właściwie niepotrzebne. Firma Microsoft wreszcie odpowiedziała na żądania tysięcy użytkowników Excela, prosiących o wprowadzenie możliwości definiowania zakresów danych, które będą spełniać rolę etykiet danych na wykresie. Taka funkcja została zaimplementowana w Excelu 2013.

Aby zdefiniować zakres danych, który będzie spełniał rolę etykiet serii danych na wykresie, powinieneś wykonać polecenia przedstawione poniżej:

1. Utwórz wykres i zaznacz serię danych, która będzie spełniała rolę etykiet danych.
2. Kliknij przycisk *Elementy wykresu*, który pojawi się po prawej stronie wykresu, i z menu podręcznego wybierz polecenie *Etykiety danych*.
3. Kliknij przycisk ze strzałką, który pojawi się po prawej stronie polecenia *Etykiety danych*, i z menu podręcznego wybierz polecenie *Więcej opcji*.
4. Na ekranie pojawi się panel zadań *Formatowanie etykiet danych*.
5. Zaznacz opcję *Wartość z komórek*.
6. Excel poprosi Cię o zaznaczenie zakresu danych, który zawiera etykiety.

Na rysunku 16.11 przedstawiono przykład takiego wykresu. Jako serii danych zawierających etykiety użyłem zakresu C2:C7. W przeszłości (czytaj: w poprzednich wersjach Excela) etykiety danych trzeba było tworzyć ręcznie bądź za pośrednictwem odpowiedniego makra VBA.

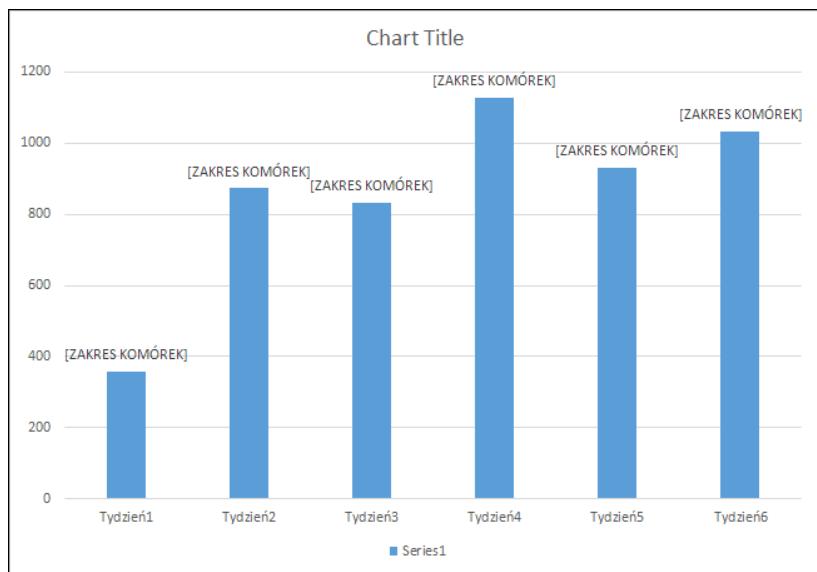


Rysunek 16.11. Etykiety danych, utworzone na podstawie arbitralnie wybranego zakresu komórek, pokazują procentową zmianę liczby zamówień w poszczególnych tygodniach

Jak widać, nowy mechanizm tworzenia etykiet danych jest bardzo efektywny, ale niestety charakteryzuje się zupełnym brakiem kompatybilności wstępnej. Na rysunku 16.12 przedstawiono wygląd takiego wykresu po otwarciu arkusza w Excelu 2010.

Rysunek 16.12.

Etykiety danych utworzone na podstawie arbitralnie wybranego zakresu komórek nie są kompatybilne z poprzednimi wersjami Excela

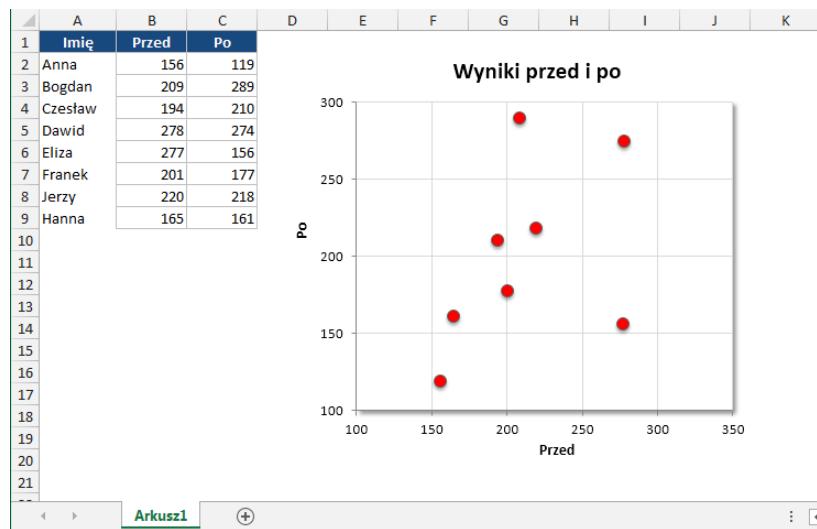


W pozostałej części tego podrozdziału pokażemy, w jaki sposób utworzyć takie etykiety danych, bazujące na wybranym zakresie komórek, za pomocą VBA. Etykiety danych utworzone w ten sposób są kompatybilne z poprzednimi wersjami Excela.

Dla przykładu przeanalizujmy wykres XY pokazany na rysunku 16.13. Bardzo użyteczną funkcją byłaby możliwość wyświetlenia nazwy dla każdego punktu na wykresie.

Rysunek 16.13.

Wykres XY bez etykiet danych



Procedura DataLabelsFromRange działa dla pierwszego wykresu w aktywnym arkuszu. Wyświetla pytanie o zakres, a następnie przetwarza w pętli kolekcję Points i modyfikuje właściwość Text każdego obiektu należącego do kolekcji na wartości z zakresu.

```

Sub DataLabelsFromRange()
    Dim DLRange As Range
    Dim Cht As Chart
    Dim i As Integer, Pts As Integer

    ' Wybór wykresu
    Set Cht = ActiveSheet.ChartObjects(1).Chart

    ' Wyświetlenie pytania o zakres
    On Error Resume Next
    Set DLRange = Application.InputBox _
        (prompt:="Zakres zawierający etykiety danych?", Type:=8)
    If DLRange Is Nothing Then Exit Sub
    On Error GoTo 0

    ' Dodanie etykiet danych
    Cht.SeriesCollection(1).ApplyDataLabels _
        Type:=xlDataLabelsShowValue, _
        AutoText:=True, _
        LegendKey:=False

    ' Przetwarzanie w pętli kolekcji Points i ustawianie etykiet danych
    Pts = Cht.SeriesCollection(1).Points.Count
    For i = 1 To Pts
        Cht.SeriesCollection(1). _
            Points(i).DataLabel.Text = DLRange(i)
    Next i
End Sub

```



Skoroszyt z tym przykładem (*Etykiety danych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 16.14 pokazano wygląd wykresu po uruchomieniu procedury DataLabelsFromRange i wprowadzeniu A2:A9 jako zakresu etykiet.

Etykiety danych na wykresie mogą jednocześnie być łączami do komórek. Aby zmodyfikować procedurę DataLabelsFromRange tak, aby tworzyła również łącza do komórek, powinieneś zmienić polecenie wewnętrz pętli For ... Next w następujący sposób:

```

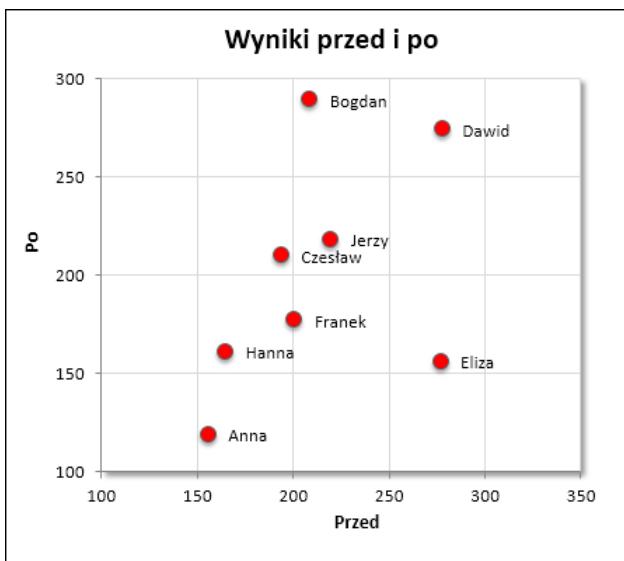
Cht.SeriesCollection(1).Points(i).DataLabel.Text = _
    "=" & "" & DLRange.Parent.Name & "!" &
    DLRange(i).Address(ReferenceStyle:=xlR1C1)

```

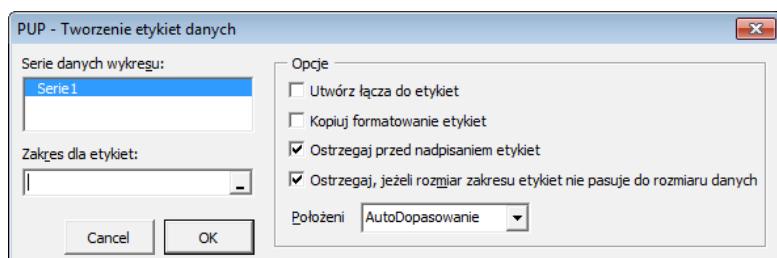


Kod poprzedniej procedury jest dosyć surowy i praktycznie nie ma obsługi błędów. Poza tym działa tylko dla pierwszego obiektu kolekcji Series. Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt PUP — Tworzenie etykiet danych.xlsxm, który zawiera znacznie bardziej rozbudowane narzędzie przeznaczone do modyfikowania etykiet na wykresach. Na rysunku 16.15 przedstawiono wygląd okna dialogowego tego narzędzia. Warto zauważyć, że w bardzo prosty sposób możesz skonwertować ten skoroszyt na dodatek programu Excel.

Rysunek 16.14.
Wykres XY zawierający etykiety danych dzięki zastosowaniu procedury VBA



Rysunek 16.15.
Okno dialogowe narzędzia PUP — Tworzenie etykiet danych

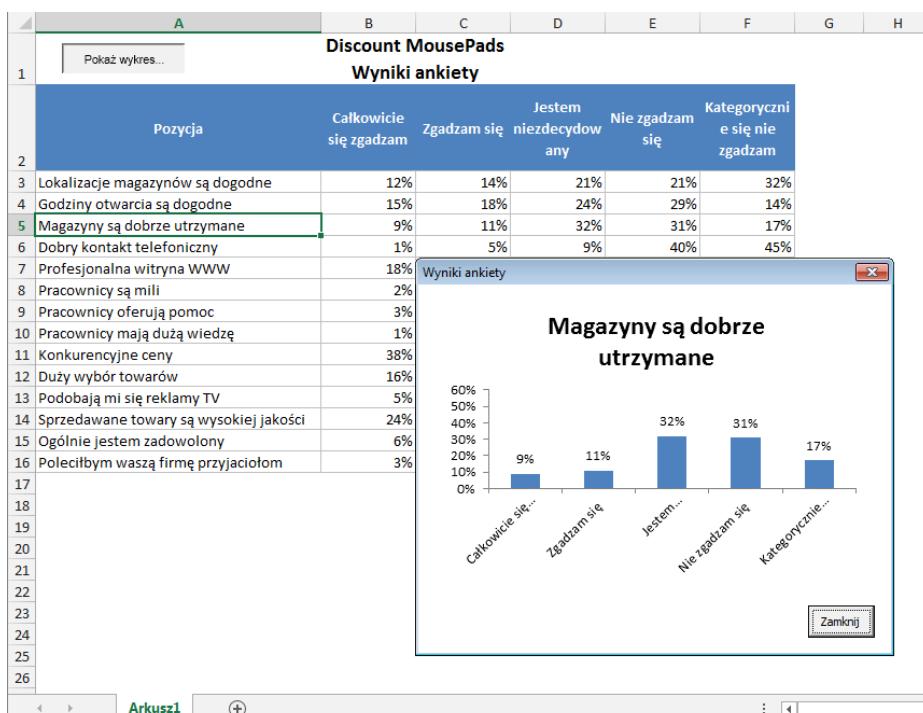


Wyświetlanie wykresu w oknie formularza UserForm

W rozdziale 13. zaprezentowaliśmy sposób wyświetlania wykresu w oknie formularza *UserForm*. Omawiana technika polegała na zapisaniu wykresu do pliku *GIF*, a następnie załadowaniu pliku do formantu *Image* umieszczonego na formularzu *UserForm*.

Przykład omawiany w tym podrozdziale wykorzystuje tę samą technikę, ale z wprowadzonym dodatkowym elementem: wykres jest tworzony „w locie”, na podstawie danych z wiersza, w którym znajduje się aktywna komórka. Efekt działania procedury pokazano na rysunku 16.16.

Okno formularza *UserForm* wykorzystane dla potrzeb tego przykładu jest bardzo proste. Zawiera formant *Image* (obraz) oraz przycisk *Zamknij* (formant *CommandButton*). W arkuszu z danymi znajduje się przycisk, którego naciśnięcie powoduje wykonanie następującej procedury:



Rysunek 16.16. Wykres w oknie formularza UserForm został utworzony „w locie” na podstawie danych z bieżącego wiersza

```
Sub ShowChart()
    Dim UserRow As Long
    UserRow = ActiveCell1.Row
    If UserRow < 2 Or IsEmpty(Cells(UserRow, 1)) Then
        MsgBox "Przenieś kursor do wiersza zawierającego dane."
        Exit Sub
    End If
    CreateChart (UserRow)
    UserForm1.Show
End Sub
```

Wykres jest tworzony na podstawie danych z wiersza, w którym znajduje się aktywna komórka. Jeżeli wskaźnik komórki znajduje się w nieodpowiednim wierszu, procedura wyświetla ostrzeżenie. Jeżeli komórka jest odpowiednia, procedura ShowChart wywołuje procedurę CreateChart w celu utworzenia wykresu, a następnie wyświetla go w oknie formularza *UserForm*.

Procedura CreateChart, której kod zamieszczono poniżej, pobiera jeden argument reprezentujący wiersz z aktywną komórką. Jest to zmodyfikowana i poprawiona wersja kodu procedury zarejestrowanej przez rejestrator makr.

```
Sub CreateChart(r)
    Dim TempChart As Chart
    Dim CatTitles As Range
    Dim SrcRange As Range, SourceData As Range
    Dim FName As String
```

```

Set CatTitles = ActiveSheet.Range("A2:F2")
Set SrcRange = ActiveSheet.Range(Cells(r, 1), Cells(r, 6))
Set SourceData = Union(CatTitles, SrcRange)

    ' Dodaj wykres
    Application.ScreenUpdating = False
    Set TempChart = ActiveSheet.Shapes.AddChart.Chart
    TempChart.SetSourceData Source:=SourceData

    ' Dopusz formatowanie
    With TempChart
        .ChartType = xlColumnClustered
        .SetSourceData Source:=SourceData, PlotBy:=xlRows
        .ChartStyle = 25
        .HasLegend = False
        .PlotArea.Interior.ColorIndex = xlNone
        .Axes(xlValue).MajorGridlines.Delete
        .ApplyDataLabels Type:=xlDataLabelsShowValue, LegendKey:=False
        .Axes(xlValue).MaximumScale = 0.6
        .ChartArea.Format.Line.Visible = False
    End With

    ' Dopusz rozmiar obiektu ChartObject
    With ActiveSheet.ChartObjects(1)
        .Width = 300
        .Height = 200
    End With

    ' Zapisz wykres jako GIF
    FName = ThisWorkbook.Path & Application.PathSeparator & "temp.gif"
    TempChart.Export Filename:=FName, filterName:="GIF"
    ActiveSheet.ChartObjects(1).Delete
    Application.ScreenUpdating = True
End Sub

```

Gdy procedura CreateChart zostanie zakończona, w arkuszu zostanie umieszczony obiekt ChartObject zawierający wykres utworzony na podstawie danych z wiersza aktywnej komórki. Obiekt ChartObject nie będzie jednak widoczny, ponieważ właściwość Screen Updating została wyłączona. Wykres jest eksportowany i usuwany, a następnie ponownie włączana jest właściwość ScreenUpdating (aktualizacja ekranu).

Ostatnia instrukcja procedury ShowChart powoduje załadowanie okna formularza *UserForm*. Poniżej zamieszczono listing procedury UserForm_Initialize. Jej działanie polega po prostu na załadowaniu pliku *GIF* do formantu *Image*.

```

Private Sub UserForm_Initialize()
    Dim FName As String
    FName = Application.DefaultFilePath &
        Application.PathSeparator & "temp.gif"
    UserForm1.Image1.Picture = LoadPicture(FName)
End Sub

```

Skoroszyt z tym przykładem (*Wykres na formularzu UserForm.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Zdarzenia związane z wykresami

Excel obsługuje szereg zdarzeń związanych z wykresami. Na przykład w momencie uaktywniania wykresu generowane jest zdarzenie `Activate`. Zdarzenie `Calculate` zachodzi wtedy, kiedy do wykresu zostaną przesłane nowe lub zmodyfikowane dane. Oczywiście istnieje możliwość napisania kodu VBA wykonywanego w momencie zajścia określonego zdarzenia.



Dodatkowe informacje na temat zdarzeń znajdziesz w rozdziale 17.

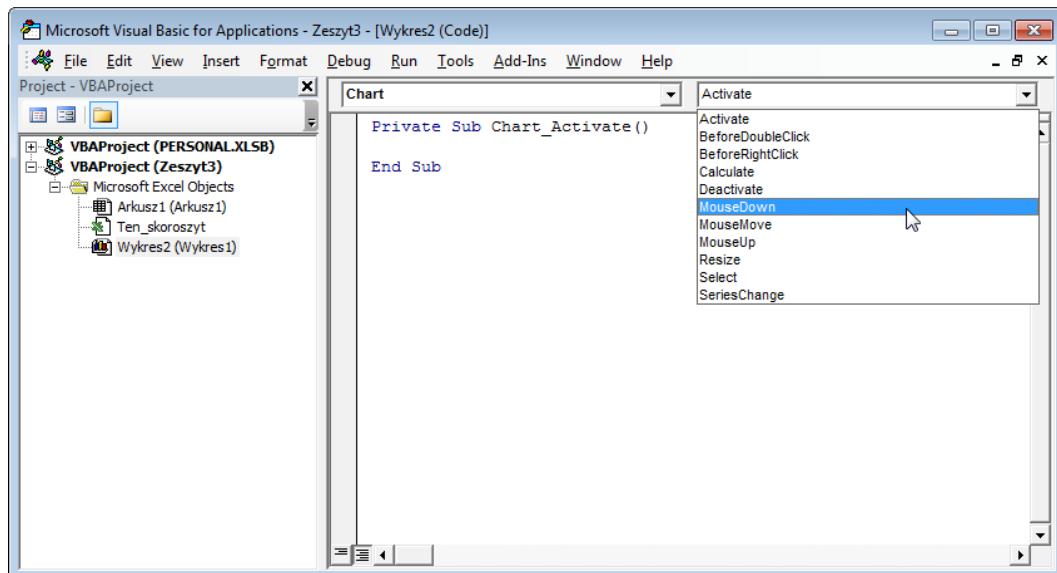
W tabeli 16.1 zestawiono wszystkie zdarzenia związane z wykresami.

Tabela 16.1. Zdarzenia dotyczące obiektu Chart

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
<code>Activate</code>	Uaktywnienie arkusza wykresu lub wykresu osadzonego.
<code>BeforeDoubleClick</code>	Dwukrotne kliknięcie wykresu osadzonego. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku dwukrotnego kliknięcia.
<code>BeforeRightClick</code>	Kliknięcie wykresu osadzonego prawym przyciskiem myszy. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku kliknięcia prawym przyciskiem myszy.
<code>Calculate</code>	Wykreślenie nowych lub zmodyfikowanych danych.
<code>Deactivate</code>	Dezaktywacja wykresu.
<code>MouseDown</code>	Przyciśnięcie przycisku myszy w czasie, kiedy wskaźnik myszy znajduje się nad wykresem.
<code>MouseMove</code>	Zmiana pozycji wskaźnika myszy znajdującego się nad wykresem.
<code>MouseUp</code>	Zwolnienie przycisku myszy w czasie, kiedy wskaźnik myszy znajduje się nad wykresem.
<code>Resize</code>	Zmiana rozmiaru wykresu.
<code>Select</code>	Zaznaczenie elementu na wykresie.
<code>SeriesChange</code>	Modyfikacja wartości punktu danych należącego do wykresu.

Przykład wykorzystania zdarzeń związanych z wykresami

Aby można było zaprogramować procedurę obsługi zdarzenia dotyczącego arkusza wykresu, kod VBA musi się znajdować w module kodu obiektu `Chart`. Aby uaktywnić ten moduł kodu, należy dwukrotnie kliknąć pozycję wykresu w oknie projektu. Następnie w module kodu, z rozwijanej listy `Object` po lewej stronie trzeba wybrać pozycję `Chart`, a z rozwijanej listy `Procedure` po prawej stronie (rysunek 16.17) odpowiednie zdarzenie.



Rysunek 16.17. Wybór zdarzenia w module kodu obiektu Chart



Ponieważ wykresy osadzone nie posiadają własnego modułu kodu, procedura przedstawiona w tym podrozdziale działa tylko dla wykresów umieszczonych w osobnym arkuszu. Dla wykresów osadzonych również można obsługiwać zdarzenia, ale w tym celu należy przeprowadzić wstępne czynności konfiguracyjne obejmujące utworzenie modułu klasy. Zagadnienie to zostało opisane w dalszej części rozdziału w punkcie „Obsługa zdarzeń: wykresy osadzone”.

Pokazany poniżej przykład powoduje wyświetlenie komunikatu w momencie, kiedy użytkownik uaktywni arkusz wykresu lub wybierze dowolny element na wykresie. Najpierw utworzyłem skoroszyt zawierający wykres w osobnym arkuszu, a następnie napisałem trzy procedury obsługi zdarzeń:

- Chart_Activate — wykonywaną w chwili uaktywnienia wykresu,
- Chart_Deactivate — wykonywaną w chwili dezaktywacji wykresu,
- Chart_Select — wykonywaną w momencie zaznaczenia elementu na wykresie.



Skoroszyt z tym przykładem (Zdarzenia — arkusz wykresu.xlsxm) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

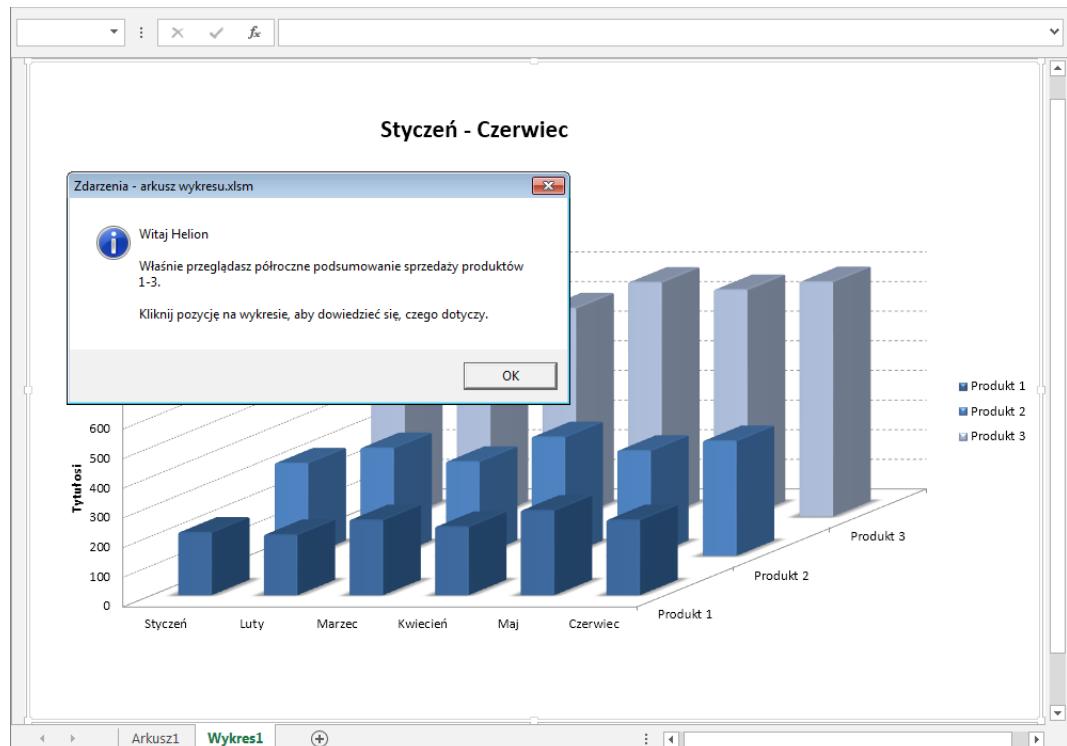
Kod procedury Chart_Activate znajduje się poniżej:

```

Private Sub Chart_Activate()
    Dim msg As String
    msg = "Witaj " & Application.UserName & vbCrLf & vbCrLf
    msg = msg & "Właśnie przeglądzasz półroczone podsumowanie "
    msg = msg & "sprzedaży produktów 1-3. " & vbCrLf & vbCrLf
    msg = msg &
        "Kliknij pozycję na wykresie, aby dowiedzieć się, czego dotyczy. "
    MsgBox msg, vbInformation, ActiveWorkbook.Name
End Sub

```

Procedura powoduje wyświetlenie komunikatu podczas każdego uaktywnienia wykresu. Wygląd komunikatu przedstawiono na rysunku 16.18.



Rysunek 16.18. Uaktywnienie wykresu powoduje uruchomienie procedury Chart_Activate i wyświetlenie komunikatu

Procedura Chart_Deactivate, która znajduje się poniżej, także wyświetla komunikat, tym razem w przypadku dezaktywacji wykresu:

```
Private Sub Chart_Deactivate()
    Dim msg As String
    msg = "Dziękujemy za skorzystanie z wykresu."
    MsgBox msg, , ActiveWorkbook.Name
End Sub
```

Kolejna procedura, Chart_Select, jest wykonywana za każdym razem, kiedy na wykresie zostanie zaznaczony jakiś element:

```
Private Sub Chart_Select(ByVal ElementID As Long, _
    ByVal Arg1 As Long, ByVal Arg2 As Long)
    Dim Id As String
    Select Case ElementID
        Case xlAxis: Id = "Axis"
        Case xlAxisTitle: Id = "AxisTitle"
        Case xlChartArea: Id = "ChartArea"
        Case xlChartTitle: Id = "ChartTitle"
        Case xlCorners: Id = "Corners"
        Case xlDataLabel: Id = "DataLabel"
```

```
Case xlDataTable: Id = "DataTable"
Case xlDownBars: Id = "DownBars"
Case xlDropLines: Id = "DropLines"
Case xlErrorBars: Id = "ErrorBars"
Case xlFloor: Id = "Floor"
Case xlHiLoLines: Id = "HiLoLines"
Case xlLegend: Id = "Legend"
Case xlLegendEntry: Id = "LegendEntry"
Case xlLegendKey: Id = "LegendKey"
Case xlMajorGridlines: Id = "MajorGridlines"
Case xlMinorGridlines: Id = "MinorGridlines"
Case xlNothing: Id = "Nothing"
Case xlPlotArea: Id = "PlotArea"
Case xlRadarAxisLabels: Id = "RadarAxisLabels"
Case xlSeries: Id = "Series"
Case xlSeriesLines: Id = "SeriesLines"
Case xlShape: Id = "Shape"
Case xlTrendline: Id = "Trendline"
Case xlUpBars: Id = "UpBars"
Case xlWalls: Id = "Walls"
Case xlXErrorBars: Id = "XErrorBars"
Case xlYErrorBars: Id = "YErrorBars"
Case Else: Id = "Nieznaný"
End Select

MsgBox "Selection type:" & Id & vbCrLf & Arg1 & vbCrLf & Arg2
End Sub
```

Procedura wyświetla okno informacyjne zawierające typ wybranego elementu oraz wartości argumentów Arg1 i Arg2. Kiedy zajdzie zdarzenie Select, argument ElementID będzie zawierał liczbę całkowitą odpowiadającą elementowi, który został zaznaczony. Argumenty Arg1 i Arg2 zawierają dodatkowe informacje o wybranym elemencie (więcej informacji znajduje się w pomocy online). Procedura Select Case zamienia wbudowane stałe na opisowe łańcuchy znaków.



Ponieważ w kodzie programu nie została umieszczona pełna lista elementów, które mogą występować w obiekcie Chart (nie ma tam na przykład obiektów specyficznych dla wykresów przestawnych), użyłem instrukcji Case Else.

Obsługa zdarzeń dla wykresów osadzonych

Jak wspomniałem w poprzednim punkcie, zdarzenia dotyczące obiektu Chart są automatycznie włączane dla wykresów w osobnych arkuszach, ale nie dla wykresów osadzonych. Aby wykorzystać zdarzenia dla wykresów osadzonych, należy wykonać czynności opisane poniżej.

Utworzenie modułu klasy

W edytorze Visual Basic należy zaznaczyć projekt w oknie *Project*, a następnie wybrać polecenie *Insert/Class Module*. Spowoduje to dodanie nowego (pustego) modułu klasy do projektu. Następnie należy wprowadzić opisową nazwę modułu w oknie *Properties* (np. *c1sChart*). Zmiana nazwy modułu klasy nie jest konieczna, ale jest to dobra praktyka.

Zadeklarowanie publicznego obiektu Chart

Kolejną operacją jest zadeklarowanie zmiennej publicznej, która będzie reprezentować wykres. Zmienna powinna być typu Chart i należy ją zadeklarować w module klasy za pomocą słowa kluczowego WithEvents. Jeżeli słowo kluczowe WithEvents zostanie pominięte, obiekt nie będzie odpowiadał na zdarzenia. Poniżej znajduje się przykład takiej deklaracji:

```
Public WithEvents clsChart As Chart
```

Powiązanie zadeklarowanego obiektu z wykresem

Aby procedury obsługi zdarzeń zaczęły działać, należy powiązać zadeklarowany obiekt w module klasy z wykresem osadzonym. Wykonuje się to, deklarując obiekt typu clsChart (nazwa typu musi być zgodna z nazwą modułu klasy). Powinna to być zmienna obiektowa poziomu modułu zadeklarowana w module VBA (a nie w module klasy). Oto przykład:

```
Dim MyChart As New clsChart
```

Następnie należy napisać kod, który powiąże obiekt clsChart z określonym wykresem. Czynność tę wykonuje poniższa instrukcja:

```
Set MyChart.clsChart = ActiveSheet.ChartObjects(1).Chart
```

Po wykonaniu instrukcji obiekt clsChart w module klasy będzie wskazywał pierwszy wykres osadzony na aktywnym arkuszu. W rezultacie, kiedy zajdą określone zdarzenia, zostaną wykonane odpowiednie procedury obsługi zapisane w module klasy.

Utworzenie procedur obsługi zdarzeń dla klasy Chart

W tym podpunkcie pokażemy, jak należy tworzyć procedury obsługi zdarzeń w module klasy. Jak pamiętasz, w module klasy musi znajdować się deklaracja następującej postaci:

```
Public WithEvents clsChart As Chart
```

Nowy obiekt, zadeklarowany z użyciem słowa kluczowego WithEvents, zostanie wyświetlony na rozwijanej liście *Object* w module klasy. Wybranie obiektu na liście *Object* spowoduje wyświetlenie zdarzeń właściwych dla tego obiektu na liście *Procedure* znajdującej się po prawej stronie ekranu.

W kolejnym przykładzie pokazano prostą procedurę obsługi zdarzeń wykonywaną w momencie uaktywnienia wbudowanego wykresu. Procedura powoduje wyświetlenie okna informacyjnego z nazwą obiektu nadawanego dla obiektu Chart (jest nim obiekt ChartObject):

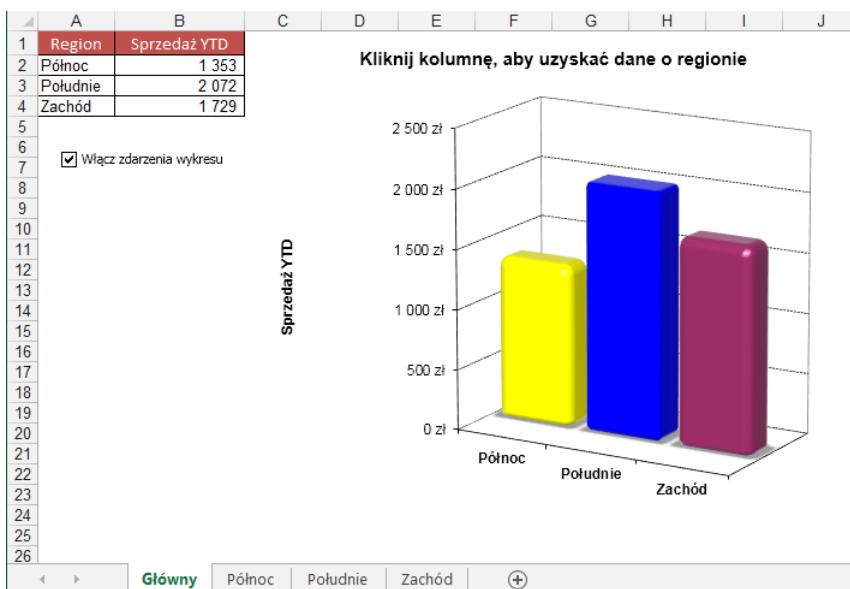
```
Private Sub clsChart_Activate()
    MsgBox "Uaktywniono wykres " & clsChart.Parent.Name
End Sub
```



Skoroszyt ilustrujący zastosowanie tej metody (*Zdarzenia — wykres osadzony.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Przykład zastosowania zdarzeń dla wykresów osadzonych

Przykład zaprezentowany w tym punkcie jest praktyczną ilustracją zagadnień opisanych w punkcie poprzednim. Skoroszyt, którego wygląd pokazano na rysunku 16.19, zawiera wykres osadzony, działający jak mapa obrazkowa, czyli obiekt graficzny powiązany z łączami, których kliknięcie powoduje wykonanie określonych działań. Kliknięcie kolumny na wykresie uaktywnia arkusz wyświetlający szczegółowe dane o wskazanym regionie.



Rysunek 16.19. Ten wykres działa jak mapa obrazkowa

Skoroszyt składa się z czterech arkuszy. Arkusz o nazwie *Główny* zawiera wbudowany wykres. Pozostałe arkusze to *Północ*, *Południe* i *Zachód*. Formuły zapisane w zakresie B1:B4 podsumowują dane w odpowiednich arkuszach i te sumaryczne dane są wykreślane na wykresie. Kliknięcie kolumny na wykresie powoduje wygenerowanie zdarzenia, a procedura obsługi tego zdarzenia uaktywnia odpowiedni arkusz, w którym są wyświetlane szczegółowe informacje o wybranym regionie.

Skoroszyt zawiera zarówno moduł klasy `EmbChartClass`, jak również zwykły moduł VBA `Module1`. Dla celów demonstracyjnych w arkuszu *Główny* umieszczono dodatkową opcję, której zaznaczenie powoduje wykonanie procedury `CheckBox1_Click`. Procedura ta odpowiednio włącza lub wyłącza monitorowanie zdarzeń wykresu.

Dodatkowo w każdym arkuszu znajduje się przycisk powodujący wykonanie makra `ReturntoMain`, które ponownie uaktywnia arkusz *Główny*.

Pelny listing modułu Module1 zamieszczono poniżej:

```

Dim SummaryChart As New EmbChartClass
Sub CheckBox1_Click()
    If Worksheets("Główny").Checkboxes("Check Box 1") = xlOn Then
        ' Włącza zdarzenia wykresu
        Range("A1").Select
        Set SummaryChart.myChartClass =
            Worksheets(1).ChartObjects(1).Chart
    Else
        ' Wyłącza zdarzenia wykresu
        Set SummaryChart.myChartClass = Nothing
        Range("A1").Select
    End If
End Sub

Sub ReturnToMain()
    ' Wywoływana przez kliknięcie przycisku
    Sheets("Główny").Activate
End Sub

```

Pierwsza instrukcja jest deklaracją nowej zmiennej obiektowej SummaryChart typu EmbChartClass, co — jak pamiętamy — jest nazwą modułu klasy. Kiedy użytkownik kliknie przycisk *Włącza obsługę zdarzeń wykresu*, wykres osadzony zostanie przypisany do obiektu, co w rezultacie spowoduje uaktywnienie obsługi zdarzeń dla wykresu. Zawartość modułu klasy EmbChartClass jest następująca:

```

Public WithEvents myChartClass As Chart
Private Sub myChartClass_MouseDown(ByVal Button As Long, _
    ByVal Shift As Long, ByVal X As Long, ByVal Y As Long)

    Dim IDnum As Long
    Dim a As Long, b As Long

    ' Kolejne polecenie zwraca wartości dla
    ' IDnum, a oraz b
    myChartClass.GetChartElement X, Y, IDnum, a, b

    ' Czy seria została kliknięta?
    If IDnum = xlSeries Then
        Select Case b
            Case 1
                Sheets("Północ").Activate
            Case 2
                Sheets("Południe").Activate
            Case 3
                Sheets("Zachód").Activate
        End Select
    End If
    Range("A1").Select
End Sub

```

Kliknięcie wykresu generuje zdarzenie MouseDown, które powoduje wykonanie procedury myChartClass_MouseDown. Procedura wykorzystuje metodę GetChartElement w celu określenia, który element wykresu kliknięto. Metoda GetChartElement zwraca informacje na temat elementu wykresu w postaci współrzędnych X i Y (informacje te są dostępne za pośrednictwem argumentów procedury myChartClass_MouseDown).



Skoroszyt z tym przykładem (*Wykresy — mapa obrazkowa.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Jak ułatwić sobie pracę z wykresami przy użyciu VBA?

W tym podrozdziale opisałem niektóre interesujące rozwiązania, które opracowałem przez lata pracy z Exceliem. Niektóre z nich mogą być użyteczne podczas tworzenia aplikacji, natomiast inne są wyłącznie ciekawostką. Przestudiowanie ich z pewnością umożliwi Ci inne spojrzenie na model obiektowy wykresów Excela.

Drukowanie wykresów osadzonych na arkuszu

Po zaznaczeniu wykresu osadzonego można go wydrukować poprzez wybranie polecenia *PLIK/Drukuj*. Wykres osadzony zostanie wydrukowany na pełnej stronie (dokładnie tak, jak w przypadku wykresów umieszczonego w osobnym arkuszu).

Poniższe makro powoduje wydrukowanie wszystkich wykresów osadzonych na aktywnym arkuszu. Każdy wykres będzie wydrukowany na osobnej stronie:

```
Sub PrintEmbeddedCharts()
    Dim ChtObj As ChartObject
    For Each ChtObj In ActiveSheet.ChartObjects
        ChtObj.Chart.Printout
    Next ChtObj
End Sub
```

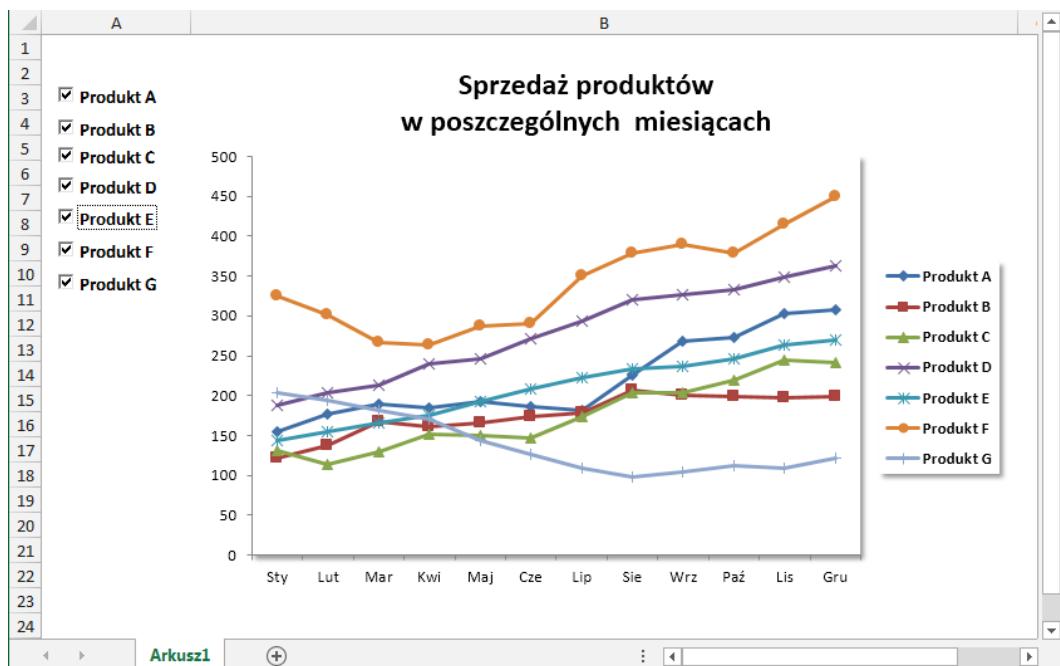
Ukrywanie serii danych poprzez ukrywanie kolumn

Domyślnie wykresy Excela nie wyświetlają danych znajdujących się w ukrytych wierszach i kolumnach. Skoroszyt przedstawiony na rysunku 16.20 ilustruje prosty sposób na umożliwienie użytkownikowi ukrywania i wyświetlania wybranych serii danych. Na wykresie prezentowanych jest siedem serii danych, co może wprowadzać niejakie zamieszanie.

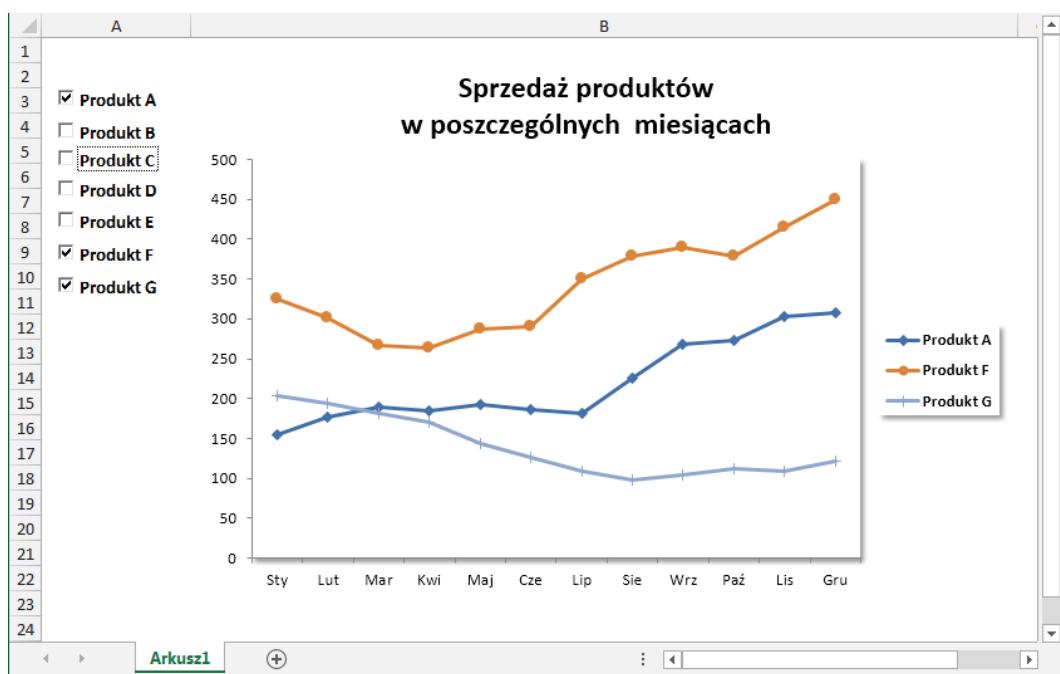
Napisanie kilku prostych makr pozwala użytkownikowi na wykorzystanie opcji (formantów ActiveX) do wskazania, które serie danych mają być wyświetlane. Na rysunku 16.21 przedstawiono ten sam wykres prezentujący już tylko trzy serie danych.

Kolejne serie danych znajdują się w zakresach o zdefiniowanych nazwach Produkt_A, Produkt_B i tak dalej. Każdy formant CheckBox posiada swoją procedurę obsługi zdarzenia Click. Na przykład procedura wykonywana po zaznaczeniu opcji *Produkt A* wygląda następująco:

```
Private Sub CheckBox1_Click()
    ActiveSheet.Range("Produkt_A").EntireColumn.Hidden = _
        Not ActiveSheet.OLEObjects(1).Object.Value
End Sub
```



Rysunek 16.20. Zastosowanie formantów CheckBox do wskazywania wyświetlanego serii danych



Rysunek 16.21. Zagmatwany wykres staje się bardziej przejrzysty, kiedy niektóre kolumny danych zostaną ukryte



Excel 2013 ma nowy mechanizm, który w prosty sposób pozwala na ukrywanie i wyświetlanie wybranych serii danych na wykresie. Kiedy zaznaczyś dany wykres, po jego prawej stronie pojawią się ikony trzech przycisków. Gdy klikniesz ostatni z nich, *Filtры wykresu*, na ekranie pojawi się menu podręczne, zawierające pola wyboru reprezentujące poszczególne serie danych. Aby ukryć wybraną serię, po prostu usuń zaznaczenie odpowiedniej opcji. W podobny sposób możesz ukrywać i wyświetlać również wybrane punkty danych.



Skoroszyt z tym przykładem (*Ukrywanie i wyświetlanie serii danych.xlsxm*) znajdziesz na płycie CD-ROM dołączonej do książki.

Tworzenie wykresów, które nie są połączone z danymi

W normalnych warunkach wykres Excela wykorzystuje dane zapisane wewnątrz zakresu. Modyfikacja danych w zakresie powoduje automatyczne uaktualnienie wykresu. Czasami jednak trzeba odłączyć wykres od zakresu danych i utworzyć *wykres statyczny* (czyli taki, który nigdy się nie zmienia), jak choćby w przypadku wykreślania danych generowanych dla różnych scenariuszy typu *co-jeżeli*, kiedy niektóre wykresy warto zapisać jako punkty odniesienia, dla porównania z innymi scenariuszami.

Istnieją trzy sposoby utworzenia takich wykresów:

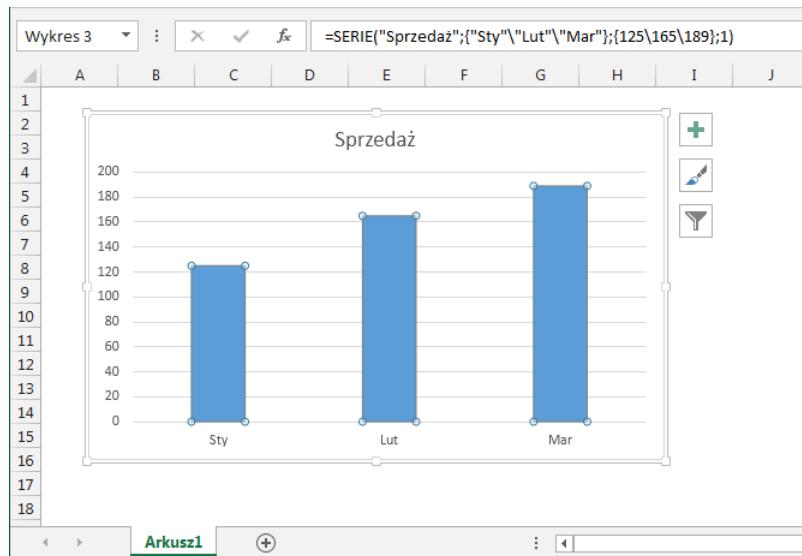
- **Skopiowanie wykresu i wklejenie go jako obrazu.** Aby to zrobić, uaktywnij wykres, wybierz polecenie *NARZĘDZIA GŁÓWNE/Schowek/Wklej/Obraz (O)* (w oknie *Kopiowanie obrazu* zaakceptuj ustawienia domyślne). Następnie kliknij wybraną komórkę i wybierz polecenie *NARZĘDZIA GŁÓWNE/Schowek/Wklej*. W rezultacie na arkuszu pojawi się obraz skopiowanego wykresu.
- **Konwersja odwołań do zakresu na tablice.** Aby to zrobić, kliknij serię danych wykresu, a następnie pasek formuły. Teraz naciśnij *F9*, co powoduje konwersję zakresów na tablicę. Czynność tę należy powtórzyć dla wszystkich serii danych na wykresie.
- **Zastosowanie procedury w języku VBA do przypisania tablicy zamiast zakresu do właściwości XValues lub Values obiektu Series.** Ta technika zostanie opisana poniżej.

Procedura zaprezentowana poniżej tworzy wykres (patrz rysunek 16.22) na podstawie tablic danych. Dane nie są przechowywane w arkuszu. Jak łatwo zauważyć, formuła SERIE zawiera tablice, a nie odwołania do zakresów.

```
Sub CreateUnlinkedChart()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.Shapes.AddChart2.Chart
    With MyChart
        .SeriesCollection.NewSeries
        .SeriesCollection(1).Name = "Sprzedaż"
        .SeriesCollection(1).XValues = Array("Sty", "Lut", "Mar")
        .SeriesCollection(1).Values = Array(125, 165, 189)
        .ChartType = xlColumnClustered
        .SetElement msoElementLegendNone
    End With
End Sub
```

Rysunek 16.22.

Ten wykres wykorzystuje dane z tablic (które nie są przechowywane na arkuszu)



Ponieważ Excel posiada ograniczenia na maksymalną długość formuły SERIE, opisana technika może być wykorzystywana tylko do tworzenia wykresów relatywnie małych serii danych.

Procedura, której kod przedstawiono poniżej, tworzy obraz aktywnego wykresu (oryginalny wykres nie jest usuwany). Procedura działa tylko dla wykresów osadzonych.

```
Sub ConvertChartToPicture()
    Dim Cht As Chart
    If ActiveChart Is Nothing Then Exit Sub
    If TypeName(ActiveSheet) = "Chart" Then Exit Sub
    Set Cht = ActiveChart
    Cht.CopyPicture Appearance:=xlPrinter, _
        Size:=xlScreen, Format:=xlPicture
    ActiveWindow.RangeSelection.Select
    ActiveSheet.Paste
End Sub
```

Kiedy wykres zostanie zamieniony na obraz, możesz nadać mu interesujący wygląd przy użyciu poleceń dostępnych w grupie opcji *NARZĘDZIA OBRAZÓW/Formatowanie/Style obrazu* (patrz przykład na rysunku 16.23).



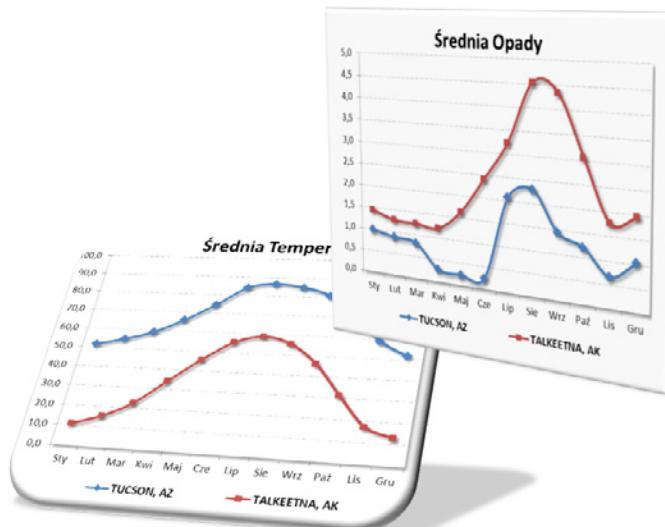
Skoroszyt z tym przykładem (*Wykres statyczny.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wykorzystanie zdarzenia MouseOver do wyświetlania tekstu

Jednym z często spotykanych problemów jest modyfikowanie porad ekranowych dotyczących wykresów. *Porady ekranowe* to niewielkie komunikaty pojawiające się obok wskaźnika myszy w przypadku wskazania wykresu. W poradzie wyświetla się nazwa elementu wykresu oraz (w przypadku serii danych) wartość punktu danych. W modelu obiektu Chart nie ma dostępu do porad, a zatem nie ma sposobu ich modyfikowania.

Rysunek 16.23.

Po zamianie wykresu na obraz możesz przy użyciu dostępnych polecień nadawać mu interesujące efekty graficzne

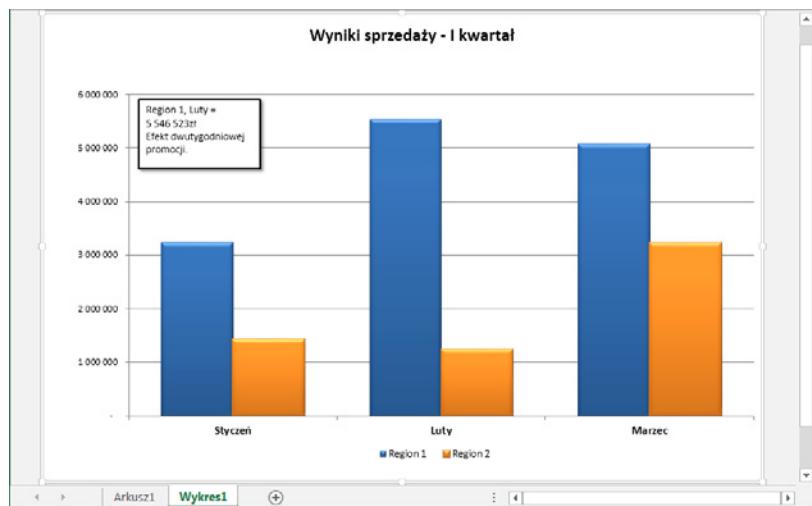


Aby włączyć lub wyłączyć wyświetlanie porad, przejdź na kartę *PLIK* i z menu wybierz polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Przejdź na kartę *Zaawansowane* i odszukaj sekcję *Wykres*. Opcje odpowiadające za wyświetlanie porad ekranowych to *Pokaż nazwy elementów wykresu przy aktywowaniu* oraz *Pokaż wartości punktów danych przy aktywowaniu*.

W tym punkcie omówiono alternatywną technikę wyświetlania porad ekranowych. Na rysunku 16.24 pokazano wykres kolumnowy, wykorzystujący zdarzenie *MouseOver*. Kiedy wskaźnik myszy znajdzie się nad kolumną, w lewym górnym rogu wyświetli się pole tekstowe (obiekt *Shape*) z informacją na temat punktu danych. Informacje są zapisane w zakresie i mogą być dowolnie formułowane.

Rysunek 16.24.

Pole tekstowe wyświetlające informacje na temat punktu danych wskazywanego przez mysz



Zaprezentowaną poniżej procedurę obsługi zdarzenia należy umieścić w module kodu arkusza wykresu:

```
Private Sub Chart_MouseMove(ByVal Button As Long, ByVal Shift As Long, _
    ByVal X As Long, ByVal Y As Long)
    Dim ElementId As Long
    Dim arg1 As Long, arg2 As Long
    On Error Resume Next
    ActiveChart.GetChartElement X, Y, ElementId, arg1, arg2
    If ElementId = xlSeries Then
        ActiveChart.Shapes(1).Visible = msoCTrue
        ActiveChart.Shapes(1).TextFrame.Characters.Text = _
            Sheets("Sheet1").Range("Comments").Offset(arg2, arg1)
    Else
        ActiveChart.Shapes(1).Visible = msoFalse
    End If
End Sub
```

Procedura monitoruje wszystkie ruchy myszą na arkuszu wykresu. Współrzędne myszy są zapisane w zmiennych X i Y przekazywanych do procedury. Argumenty Button i Shift nie są w tej procedurze używane.

Jak poprzednio, tak i tu najważniejszym elementem tej procedury jest metoda GetChartElement. Jeżeli wartością zmiennej ElementId jest xlSeries, oznacza to, że wskaźnik myszy znajduje się nad serią danych. Pole tekstowe zostaje wyświetcone i pojawia się w nim tekst przechowywany w odpowiedniej komórce arkusza, zawierający opisowe informacje na temat danego punktu danych (patrz rysunek 16.25). Jeżeli wskaźnik myszy nie znajduje się nad serią danych, pole tekstowe jest ukrywane.

Rysunek 16.25.

Zakres B7:C9 zawiera informacje o punktach danych wyświetlanych w polu tekstowym wykresu

A	B	C
1 Miesiąc	Region 1	Region 2
2 Styczeń	3 245 151	1 434 343
3 Luty	5 546 523	1 238 709
4 Marzec	5 083 204	3 224 855
5		
6 Komentarze		
7	Region 1, Styczeń = 3 245 151 zł	Region 2, Styczeń = 1 434 343 zł
8	Region 1, Luty = 5 546 523 zł Efekt dwutygodniowej promocji.	Region 2, Luty = 1 238 709 zł
9	Region 1, Marzec = 5 083 204 zł	Region 2, Marzec = 3 224 855 zł Wielka wyprzedaż!
10		
11		

← → Arkusz1 Wykres1 + :

W skoroszycie przykładowym znajdziesz również procedurę Chart_Activate, która wyłącza wyświetlanie standardowych porad ekranowych, oraz procedurę Chart_Deactivate, która przywraca ich wyświetlanie. Kod procedury Chart_Activate wygląda następująco:

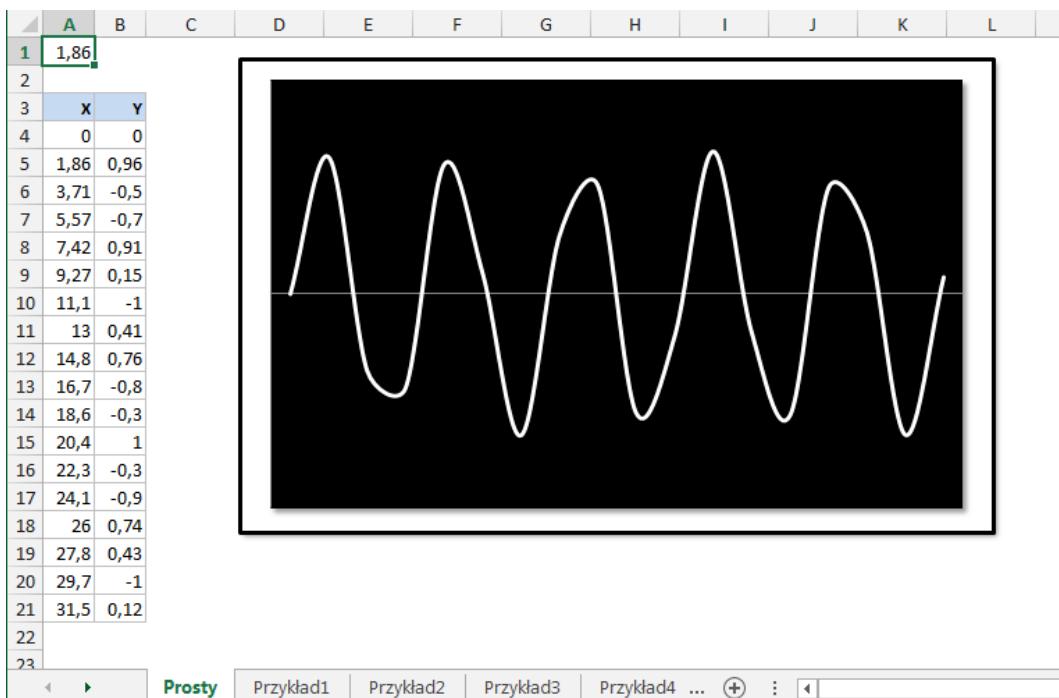
```
Private Sub Chart_Activate()
    Application.ShowChartTipNames = False
    Application.ShowChartTipValues = False
End Sub
```



Skoroszyty z przykładami dla wykresów osadzonych (*Wykresy osadzone — zdarzenie MouseOver.xlsm*) oraz arkuszy wykresów (*Arkusz wykresu — zdarzenie MouseOver.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wykresy animowane

Większość użytkowników nie zdaje sobie sprawy z tego, że Excel potrafi również wyświetlać proste animacje, na przykład animowane kształty i wykresy. Przeanalizujmy wykres XY pokazany na rysunku 16.26.



Rysunek 16.26. Prosta procedura w języku VBA przekształci ten wykres w ciekawą animację

Wartości X (kolumna A) zależą od wartości w komórce A1. Wartość w każdym kolejnym wierszu jest obliczana jako suma wartości z poprzedniego wiersza i wartości w komórce A1. Kolumna B zawiera formuły obliczające funkcję sinus dla wartości z kolumny A. Poniższa prosta procedura tworzy interesującą animację, modyfikując wartość w komórce A1, co powoduje zmianę wartości dla zakresów X i Y. W efekcie otrzymujemy efektowny wykres animowany.

```

Sub SimpleAnimation()
    Dim i As Long
    Range("A1") = 0
    For i = 1 To 150
        DoEvents
        Range("A1") = Range("A1") + 0.035
        DoEvents
    Next i
    Range("A1") = 0
End Sub

```

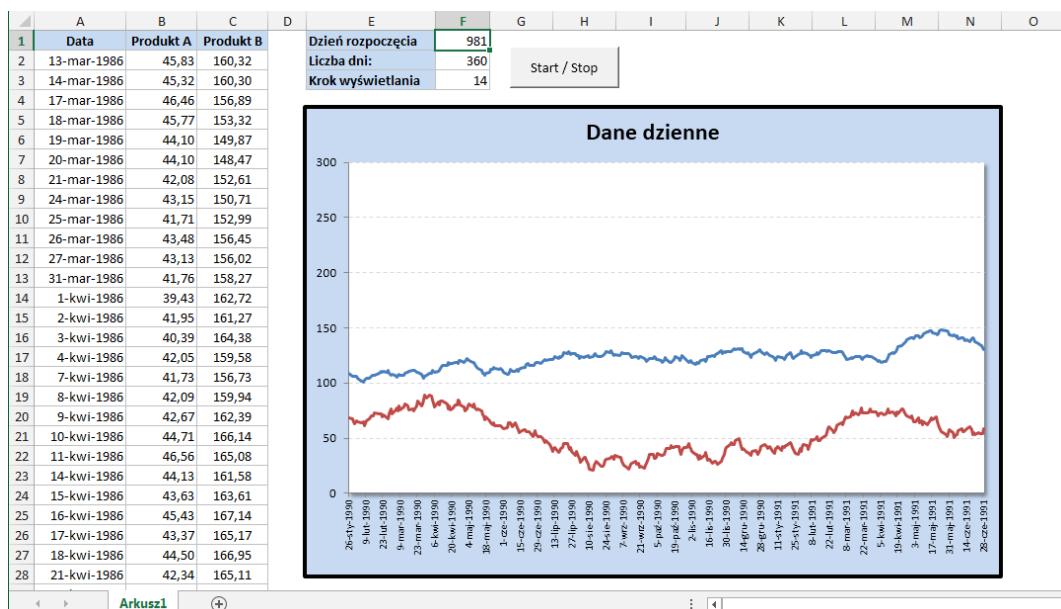
Kluczem do sukcesu podczas tworzenia wykresów animowanych jest zastosowanie jednego lub więcej poleceń `DoEvents`. Wykonanie tego polecenia powoduje chwilowe przekazanie sterowania do systemu operacyjnego, co w momencie powrotu sterowania do Excela powoduje (najwyraźniej...) aktualizację wykresu. Bez zastosowania poleceń `DoEvents` zmiany wykresu dokonywane wewnętrz pętli nie będą wyświetlane.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt (*Wykresy animowane.xlsxm*) zawierający nasz przykładowy animowany wykres oraz kilka innych przykładów animacji.

Przewijanie wykresów

Na rysunku 16.27 przedstawiono wykres zawierający 5128 punktów danych w każdej serii. Wykres mieści tylko fragment danych, ale użytkownik może przewijać wykres w lewo i w prawo, wyświetlając żądane obszary danych.



Rysunek 16.27. Wartości w kolumnie F determinują dane prezentowane na wykresie

W skoroszycie zostały zdefiniowane następujące nazwy:

- StartDay — nazwa dla komórki F1.
- NumDays — nazwa dla komórki F2.
- Increment — nazwa dla komórki F3 (wykorzystywanej do automatycznego przewijania).
- Date — nazwa dla następującej formuły:
=OFFSET(Arkusz1!\$A\$1; StartDay; 0; NumDays; 1)
- ProdA — nazwa dla następującej formuły:
=OFFSET(Arkusz1!\$B\$1; StartDay; 0; NumDays; 1)
- ProdB — nazwa dla następującej formuły:
=OFFSET(Arkusz1!\$C\$1; StartDay; 0; NumDays; 1)

Każda z formuł SERIE na wykresie używa nazw do opisania wartości kategorii i danych. Formuła SERIE dla *Produktu A* wygląda następująco (dla przejrzystości z formuły zostały usunięte nazwy arkusza i skoroszytu):

=SERIES(\$B\$1,Date,ProdA,1)

Formuła SERIE dla *Produktu B* wygląda następująco:

=SERIES(\$C\$1,Date,ProdB,2)

Użycie nazw pozwala użytkownikowi na zdefiniowanie wartości komórek StartDay oraz NumDays i wyświetlenie na wykresie odpowiadającego im podzbioru danych.



Skoroszyt z tym przykładem (*Przewijanie wykresu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Przewijanie wykresu jest realizowane przez względnie proste makro. Przycisk umieszczony na arkuszu powoduje uruchomienie makra, które przewija wykres (lub zatrzymuje jego przewijanie).

```
Public AnimationInProgress As Boolean

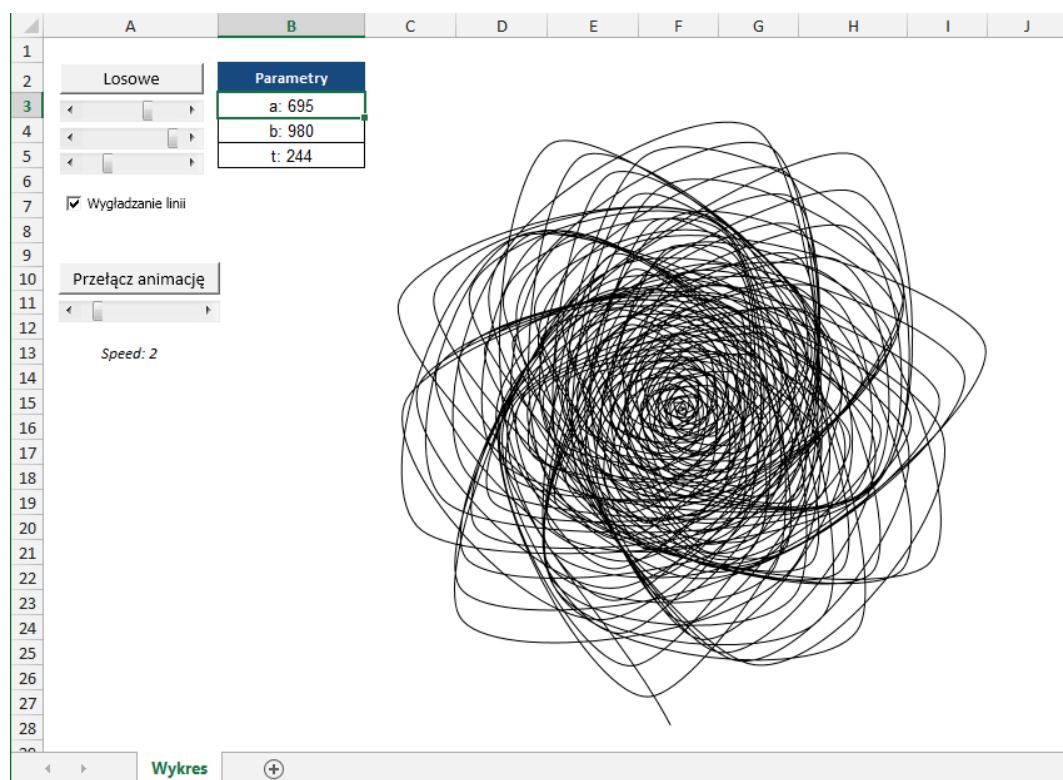
Sub AnimateChart()
    Dim StartVal As Long, r As Long
    If AnimationInProgress Then
        AnimationInProgress = False
    End If
    AnimationInProgress = True
    StartVal = Range("StartDay")
    For r = StartVal To 5219 - Range("NumDays") Step Range("Increment")
        Range("StartDay") = r
        DoEvents
    Next r
    AnimationInProgress = False
End Sub
```

Procedura AnimateChart używa zmiennej publicznej AnimationInProgress do monitorowania statusu animacji. Animacja realizowana jest w pętli, która zmienia wartość w komórce StartDay. Ponieważ wartość ta jest wykorzystywana przez dwie serie danych, wartość początkowa wykresu jest nieustannie aktualizowana. Szybkość przewijania wykresu jest kontrolowana przez wartość komórki Increment.

Do zatrzymania animacji zamiast polecenia Exit Sub użyto polecenia End. Z pewnych względów polecenie Exit Sub nie działa w takiej sytuacji stabilnie i może spowodować nawet zawieszenie się programu Excel.

Tworzenie wykresu krzywych hipocykloidalnych

Nawet tym, którzy nie lubili trygonometrii w szkole, prawdopodobnie spodoba się przykład zaprezentowany w tym punkcie, który opiera się głównie na wykorzystaniu funkcji trygonometrycznych. Skoroszyt pokazany na rysunku 16.28 wyświetla wykres XY zawierający niemal nieskończoną liczbę krzywych hipocykloidalnych. Krzywa hipocykoidalna to ścieżka, jaką kreśli punkt należący do okręgu poruszającego się wewnątrz innego okręgu. Jest to ta sama technika, którą wykorzystywano w popularnej zabawce Spirograph firmy Hasbro.



Rysunek 16.28. Skoroszyt generujący krzywe hipocykloidalne



Skoroszyt z tym przykładem (*Krzywe hipocykloidalne.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W przykładzie wykorzystano wykres XY, na którym zostały ukryte wszystkie elementy z wyjątkiem serii danych. Dane dla osi X i Y są generowane przy użyciu formuł zapisanych w kolumnach A i B. Paski przewijania w górnej części okna umożliwiają modyfikację trzech parametrów sterujących wyglądem wykresu. Dodatkowo na wykresie znajduje się przycisk *Losowo*, który generuje losowe wartości dla trzech parametrów.

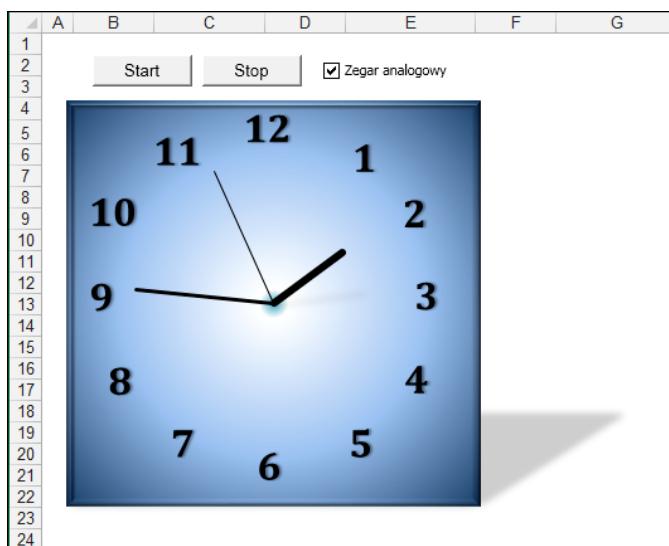
Sam wykres jest bardzo ciekawy, ale robi się *naprawdę* efektowny dopiero po dodaniu animacji, która jest realizowana poprzez zmianę w pętli początkowych wartości serii.

Tworzenie wykresu-zegara

Na rysunku 16.29 pokazano wykres XY, który skonfigurowano w taki sposób, aby wyglądał jak zegar. Co ciekawe, nasz wykres nie tylko wygląda jak zegar, ale również działa jak zegar. Nie przychodzi mi na myśl żadna konkretna sytuacja, w której można by wykorzystać taki zegar w arkuszu, ale jego utworzenie było dla mnie ciekawym wyzwaniem, a oprócz tego tworząc taki wykres, można się po prostu wiele nauczyć.

Rysunek 16.29.

Ten działający zegar jest w rzeczywistości wykresem XY „w przebraniu”

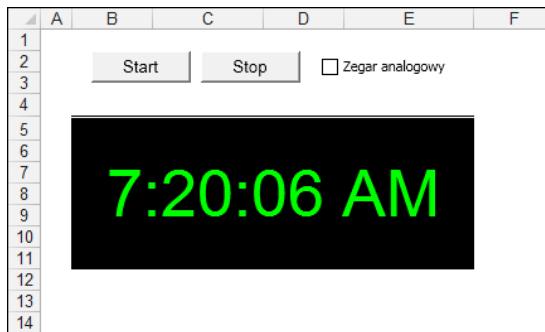


Skoroszyt z tym przykładem (*Wykres — zegar VBA.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Poza zegarem skoroszyt zawiera pole tekstowe, w którym czas jest wyświetlany w postaci zwykłego łańcucha znaków, jak pokazano na rysunku 16.30. Domyślnie to pole jest ukryte, ale można je wyświetlić, anulując zaznaczenie pola wyboru *Zegar analogowy*.

Rysunek 16.30.

Wyświetlanie zegara w postaci cyfrowej jest znacznie łatwiejsze, ale i efekt jest mniej interesujący



Podeczas analizy skoroszytu z tym przykładem warto pamiętać o kilku sprawach:

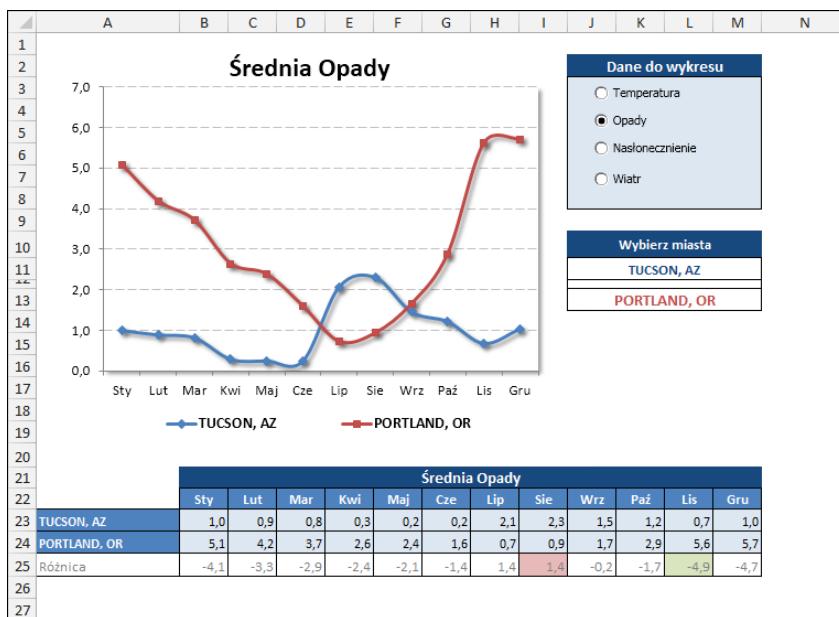
- Zmiennej obiektowej typu ChartObject nadano nazwę ClockChart. Obejmuje ona zakres DigitalClock, wykorzystywany do wyświetlania czasu w postaci cyfrowej.
- Dwa przyciski wykorzystane w arkuszu to formanty formularza. Z każdym z nich jest powiązane makro (StartClock oraz StopClock).
- Pole wyboru (o nazwie cbClockType) to również formant formularza. Kliknięcie obiektu powoduje wykonanie procedury cbClockType_Click, która przełącza wartość właściwości Visible obiektu ChartObject. Kiedy ten obiekt jest niewidoczny, wyświetlany jest zegar w postaci cyfrowej.
- W przykładzie wykorzystano wykres XY z czterema obiektami Series. Serie reprezentują wskaźówkę godzin, minut, sekund oraz 12 liczb (etykiety danych).
- Procedura UpdateClock jest uruchamiana po naciśnięciu przycisku *Uruchom zegar*. W procedurze wykorzystano metodę OnTime obiektu Application do zdefiniowania nowego zdarzenia OnTime, które wydarzy się za sekundę. Mówiąc inaczej, procedura UpdateClock jest automatycznie wywoływana co sekundę.
- W odróżnieniu od większości wykresów nasz wykres nie używa danych z arkusza. Zamiast tego odpowiednie wartości są obliczane w kodzie VBA i przenoszone bezpośrednio do właściwości Values i XValues obiektu Series wykresu.



Chociaż ten zegar jest interesującą ciekawostką, wyświetlanie w arkuszu stale aktualizowanego zegara nie jest dobrym rozwiązaniem. Makro VBA musiałoby działać w tle przez cały czas, a to z pewnością przeszkadzałoby w działaniu innych makr i ujemnie wpływało na ogólną wydajność systemu.

Tworzenie wykresu interaktywnego bez użycia VBA

Ostatni przykład, przedstawiony na rysunku 16.31, jest użyteczną aplikacją umożliwiającą użytkownikowi wybranie dwóch miast (z listy 284 miast USA) i przeglądanie wykresu, na którym porównano miasta w poszczególnych miesiącach w każdej z następujących kategorii: średnia wartość opadów, średnia temperatura, procent dni słonecznych oraz średnia prędkość wiatru.



Rysunek 16.31. W tej aplikacji wykorzystano kilka mechanizmów Excela (ale bez udziału VBA) do wykreszenia miesięcznych danych dotyczących klimatu dwóch wybranych miast w USA

Najbardziej interesującym elementem tej aplikacji jest to, że nie używa żadnych makr VBA — interaktywność zapewniają wbudowane mechanizmy Excela. Miasta są wybierane z listy rozwijanej, utworzonej za pomocą mechanizmu sprawdzania poprawności danych Excela, a z kolei opcje są wybierane za pomocą czterech formantów połączonych z komórkami. Poszczególne elementy są ze sobą powiązane za pomocą zaledwie kilku formuł.

Ten przykład pokazuje, że można utworzyć rozbudowaną i interaktywną aplikację bez konieczności stosowania makr.



Skoroszyt z tym przykładem (*Dane klimatyczne.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W kolejnych podrozdziałach bardziej szczegółowo omówiono czynności wykonane podczas tworzenia tej aplikacji.

Przygotowanie danych do utworzenia wykresu interaktywnego

Dane wejściowe dla wykresu udało mi się zdobyć po kilkunastu minutach przeszukiwania zasobów sieci Internet — na stronach Narodowego Centrum Danych Klimatycznych (ang. *National Climatic Data Center*). Po prostu skopiowałem dane z okna przeglądarki sieciowej, wkleiłem do Excela i trochę uporządkowałem. W efekcie uzyskałem cztery tabele zawierające dane w 13 kolumnach, które nazwałem odpowiednio *DaneOpady*, *DaneTemperatury*, *DaneNaślonecznienie* oraz *DaneWiatry*. Dla zachowania przejrzystości interfejsu aplikacji umieściłem dane w osobnym arkuszu (o nazwie *Dane*).

Tworzenie przycisków opcji dla interaktywnego wykresu

Aby zapewnić użytkownikowi możliwość wybierania danych do zaprezentowania na wykresie, wykorzystałem przyciski opcji (OptionButton) z grupy formantów *Formularze*. Ponieważ dla przycisków opcji tworzy się grupy, cztery przyciski opcji powiązano z tą samą komórką (komórka 03). Komórka 03 zawiera wartość z zakresu od 1 do 4, w zależności od wybranego przycisku opcji.

Musiałem w jakiś sposób uzyskać nazwę tabeli danych na podstawie wartości liczbowej zapisanej w komórce 03. Rozwiążaniem okazało się napisanie odpowiedniej formuły (w komórce 04), wykorzystującej funkcję arkuszową WYBIERZ:

```
=WYBIERZ(03, "DaneTemperatury"; "DaneOpady"; "DaneNasłonecznienie"; "DaneWiatry")
```

Dzięki takiemu rozwiążaniu w komórce 04 znajduje się nazwa jednej z czterech tabel danych. Kolejnym etapem było wprowadzenie pewnych elementów formatowania dla przycisków opcji, tak aby uatrakcyjnić ich wygląd.

Tworzenie listy miast dla wykresu interaktywnego

Nastecną czynnością jest skonfigurowanie aplikacji: utworzenie list rozwijanych w celu umożliwienia użytkownikowi wyboru miast do porównania na wykresie. Dzięki właściwości sprawdzania poprawności danych Excela utworzenie rozwijanej listy w komórce jest bardzo łatwe. Najpierw scalimy kilka komórek w celu rozszerzenia pola. Scalimy komórki J11:M11 i umieścimy tam listę wyboru pierwszego porównywanyego miasta. Obszarowi nadamy nazwę Miasto1. Następnie scalimy komórki J13:M13 i umieścimy listę wyboru drugiego porównywanyego miasta. Powstałemu obszarowi nadamy nazwę Miasto2.

Aby ułatwić działania z listą miast, utworzymy zakres o nazwie ListaMiast, który odwołuje się do pierwszej kolumny w tabeli DaneOpady.

W celu utworzenia rozwijanych list wykonaj następujące czynności:

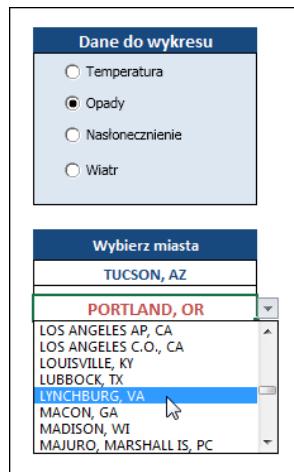
1. Zaznacz obszar J11:M11 (pamiętaj, że są to komórki scalone).
2. Przejdz na kartę *Dane*, naciśnij przycisk *Poprawność Danych* znajdujący się w grupie poleceń *Narzędzia danych* i z menu podręcznego wybierz polecenie *Poprawność danych*. Na ekranie pojawi się okno dialogowe *Sprawdzanie poprawności danych*.
3. W oknie dialogowym *Sprawdzanie poprawności danych* wybierz kartę *Ustawienia*.
4. W polu *Dozwolone* wybierz wartość *Lista*.
5. W polu *Źródło* wprowadź formułę =ListaMiast.
6. Naciśnij przycisk *OK*.
7. Skopiuj obszar J11:M11 do obszaru J13:M13.

Wykonanie takiej operacji spowoduje to zdublowanie ustawień sprawdzania poprawności danych dla drugiego miasta.

Efekty zaprezentowano na rysunku 16.32.

Rysunek 16.32.

Do wybierania miast wykorzystamy listę rozwijaną utworzoną za pomocą mechanizmu Sprawdzanie poprawności danych

**Tworzenie zakresów danych dla wykresu interaktywnego**

Kluczowym elementem tej aplikacji jest wykorzystywanie przez wykres danych z określonego zakresu. Dane pobierane są z odpowiedniej tabeli danych za pomocą formuły, w której wykorzystano funkcję WYSZUKAJ.PIONOWO (patrz rysunek 16.33).

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
20														
21														
22														
	Średnia Opady													
	Sty	Lut	Mar	Kwi	Maj	Cze	Lip	Sie	Wrz	Paź	Lis	Gru		
23	TUCSON, AZ	1,0	0,9	0,8	0,3	0,2	0,2	2,1	2,3	1,5	1,2	0,7	1,0	
24	PORTLAND, OR	5,1	4,2	3,7	2,6	2,4	1,6	0,7	0,9	1,7	2,9	5,6	5,7	
25	Różnica	-4,1	-3,3	-2,9	-2,4	-2,1	-1,4	1,4	1,4	-0,2	-1,7	-4,9	-4,7	
26														
27														

Rysunek 16.33. Na wykresie wykorzystano dane pobrane przez formuły zapisane w zakresie A23:M24

Formuła w komórce A23, wyszukująca dane na podstawie zawartości zakresu Miasto1, ma następującą postać:

```
=WYSZUKAJ.PIONOWO(Miasto1;ADR.POŚR(TabelaDanych),NR.KOLUMNY(),FAŁSZ)
```

Formuła w komórce A24 jest taka sama, ale wyszukuje dane na podstawie zawartości zakresu Miasto2:

```
=WYSZUKAJ.PIONOWO(Miasto2;ADR.POŚR(TabelaDanych),NR.KOLUMNY(),FAŁSZ)
```

Po wprowadzeniu formuł wystarczy je skopiować do następnych 12 kolumn.



Niektórych Czytelników z pewnością zainteresowało użycie funkcji NR.KOLUMNY dla trzeciego argumentu funkcji WYSZUKAJ.PIONOWO. Funkcja zwraca numer kolumny komórki zawierającej formułę. Jest to wygodny sposób pozwalający na uniknięcie kodowania „na sztywno” kolumny do pobrania. Dzięki temu dla wszystkich kolumn można zastosować tę samą formułę.

Wiersz 25 zawiera formuły, które obliczają różnicę danych pomiędzy miastami w poszczególnych miesiącach. Do wyróżnienia komórek zawierających największą różnicę i najmniejszą różnicę użyto mechanizmu formatowania warunkowego.

Etykietę powyżej listy miesięcy generuje formula odwołująca się do komórki TabelaDanych, która tworzy na tej podstawie opisowy tytuł wykresu. Formuła ma następującą postać:

```
= "Średnie " & LEWY(TabelaDanych; DŁ(TabelaDanych) - 4)
```

Utworzenie wykresu interaktywnego

Ostatnia czynność — utworzenie właściwego wykresu — jest bardzo prosta. Wykres liniowy zawiera dwie serie danych i wykorzystuje dane z zakresu A22:M24. Tytuł wykresu jest powiązany z komórką A21. Dane wewnętrz zakresu A22:M24 zmieniają się za każdym razem, kiedy zaznaczyś inny przycisk OptionButton lub z listy rozwijanej zostanie wybrane nowe miasto.

Tworzenie wykresów przebiegu w czasie

Na zakończenie tego rozdziału omówimy tzw. *wykresy przebiegu w czasie* (ang. *Sparklines*), czyli mechanizm, który został wprowadzony w Excelu 2010. Wykresy przebiegu w czasie to miniaturowe wykresy mieszczące się i wyświetlane w jednej komórce. Takie wykresy pozwalają użytkownikowi na szybkie zidentyfikowanie trendu danych w danym okresie czasu. Ze względu na swoją zartą postać, wykresy przebiegu w czasie często są stosowane w grupach.

Na rysunku 16.34 przedstawiono przykłady trzech rodzajów wykresów przebiegu w czasie, obsługiwanych przez Excela.

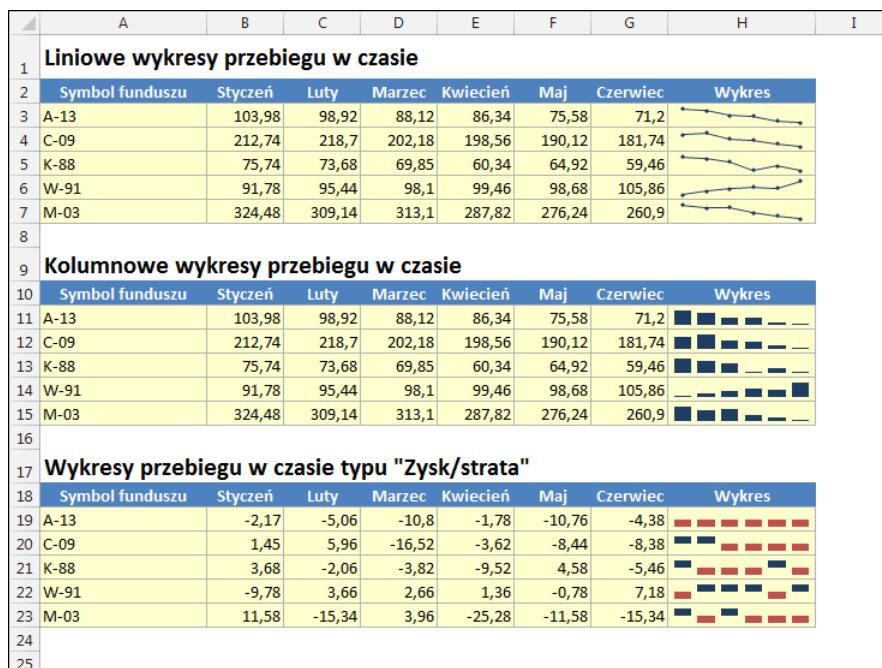
Firma Microsoft dodała wykresy przebiegu w czasie do modelu obiektowego Excela, co oznacza, że możesz używać tych wykresów za pośrednictwem procedur VBA. Na szczytcie hierarchii wykresów przebiegu w czasie znajduje się kolekcja SparklineGroups, która jest kolekcją wszystkich obiektów Sparkline. Obiekt SparklineGroup jest kontenerem obiektów Sparkline. W przeciwnieństwie do tego, czego mógłbyś oczekwać, obiektem nadrzędnym kolekcji SparklineGroups jest obiekt Range, a nie Worksheet, stąd próba wykonania polecenia przedstawionego poniżej po prostu zakończy się błędem:

```
MsgBox ActiveSheet.SparklineGroups.Count
```

Zamiast tego powinieneś użyć właściwości Cell (która zwraca obiekt klasy Range):

```
MsgBox Cells.SparklineGroups.Count
```

Przykład przedstawiony poniżej wyświetla listę adresów poszczególnych grup Sparkline znajdujących się na aktywnym arkuszu.



Rysunek 16.34. Przykłady wykresów przebiegu w czasie

```
Sub ListSparklineGroups()
    Dim sg As SparklineGroup
    Dim i As Long
    For i = 1 To Cells.SparklineGroups.Count
        Set sg = Cells.SparklineGroups(i)
        MsgBox sg.Location.Address
    Next i
End Sub
```

Z blizej nieokreślonych i nieznanych mi powodów, do przechodzenia w pętli przez poszczególne obiekty kolekcji SparklineGroups nie można używać konstrukcji For Each. Zamiast tego musisz odwoływać się kolejnych obiektów za pomocą indeksów.

Poniżej przedstawiamy kolejny przykład użycia VBA do pracy z wykresami przebiegu w czasie. Procedura SparklineReport wyświetla informacje o wszystkich wykresach tego typu, znajdujących się na aktywnym arkuszu.

```
Sub SparklineReport()
    Dim sg As SparklineGroup
    Dim sl As Sparkline
    Dim SGType As String
    Dim SLSheet As Worksheet
    Dim i As Long, j As Long, r As Long

    If Cells.SparklineGroups.Count = 0 Then
        MsgBox "Na aktywnym arkuszu nie znaleziono żadnych wykresów przebiegu w czasie."
        Exit Sub
    End If
```

```
Set SLSheet = ActiveSheet
' Wstaw nowy arkusz przeznaczony na raport
Worksheets.Add

' Nagłówki
With Range("A1")
    .Value = "Wykresy przebiegu w czasie - raport: " & SLSheet.Name &
              " Skoroszyt: " & SLSheet.Parent.Name
    .Font.Bold = True
    .Font.Size = 16
End With
With Range("A3:F3")
    .Value = Array("Grupa nr: ", "Zakres danych wykresu",
                  "Nr w grupie", "Typ", "Nr wykresu:", "Źródło danych")
    .Font.Bold = True
End With
r = 4

' Przetwarzaj w pętli kolejne grupy
For i = 1 To SLSheet.Cells.SparklineGroups.Count
    Set sg = SLSheet.Cells.SparklineGroups(i)
    Select Case sg.Type
        Case 1: SGType = "Liniowy"
        Case 2: SGType = "Kolumnowy"
        Case 3: SGType = "Zysk/strata"
    End Select

    ' Przetwarzaj w pętli kolejne wykresy w grupie
    For j = 1 To sg.Count
        Set sl = sg.Item(j)
        Cells(r, 1) = i ' Nr grupy
        Cells(r, 2) = sg.Location.Address
        Cells(r, 3) = sg.Count
        Cells(r, 4) = SGType
        Cells(r, 5) = j ' Nr wykresu w grupie
        Cells(r, 6) = sl.SourceData
        r = r + 1
    Next j
    r = r + 1
    Next i
End Sub
```

Na rysunku 16.35 przedstawiono wygląd raportu wygenerowanego dla arkusza przedstawionego na rysunku 16.34.



Skoroszyt z tym przykładem (*Wykresy przebiegu w czasie.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

	A	B	C	D	E	F	G	H	I
1	Wykresy przebiegu w czasie - raport: Arkusz1 Skoroszyt: Wykresy przebiegu w czasie.xlsxm								
2									
3	Grupa nr:	Zakres danych wykresu	Nr w grupie	Typ	Nr wykresu:	Źródło danych			
4	1	\$H\$3:\$H\$7	5	Liniowy	1	B3:G3			
5	1	\$H\$3:\$H\$7	5	Liniowy	2	B4:G4			
6	1	\$H\$3:\$H\$7	5	Liniowy	3	B5:G5			
7	1	\$H\$3:\$H\$7	5	Liniowy	4	B6:G6			
8	1	\$H\$3:\$H\$7	5	Liniowy	5	B7:G7			
9									
10	2	\$H\$11:\$H\$15	5	Kolumnowy	1	B11:G11			
11	2	\$H\$11:\$H\$15	5	Kolumnowy	2	B12:G12			
12	2	\$H\$11:\$H\$15	5	Kolumnowy	3	B13:G13			
13	2	\$H\$11:\$H\$15	5	Kolumnowy	4	B14:G14			
14	2	\$H\$11:\$H\$15	5	Kolumnowy	5	B15:G15			
15									
16	3	\$H\$19:\$H\$23	5	Zysk/strata	1	B19:G19			
17	3	\$H\$19:\$H\$23	5	Zysk/strata	2	B20:G20			
18	3	\$H\$19:\$H\$23	5	Zysk/strata	3	B21:G21			
19	3	\$H\$19:\$H\$23	5	Zysk/strata	4	B22:G22			
20	3	\$H\$19:\$H\$23	5	Zysk/strata	5	B23:G23			
21									
22									

Rysunek 16.35. Wyniki działania procedury SparklineReport

Rozdział 17.

Obsługa zdarzeń

W tym rozdziale:

- Typy zdarzeń, które Excel może monitorować
- Najważniejsze informacje o zdarzeniach
- Przykłady zdarzeń związanych ze skorosztami, arkuszami, wykresami oraz formularzami *UserForm*
- Zastosowanie zdarzeń Application do monitorowania wszystkich otwartych skorosztów
- Przykłady przetwarzania zdarzeń związanych z czasem i naciśnięciami klawiszy

Co powinieneś wiedzieć o zdarzeniach

W kilku poprzednich rozdziałach prezentowałem *procedury obsługi zdarzeń* w języku VBA, czyli procedury o specjalnych nazwach wykonywane w momencie wystąpienia określonych zdarzeń. Prostym przykładem jest procedura `CommandButton1_Click` wykonywana po kliknięciu przycisku umieszczonego w formularzu *UserForm* lub na arkuszu. Naciśnięcie przycisku jest zdarzeniem, które uruchamia wykonanie procedury obsługi tego zdarzenia.

Excel może monitorować wiele różnych zdarzeń, które możemy podzielić na następujące kategorie:

- **Zdarzenia dotyczące skorosztów** — występują w kontekście określonego skorosztu. Przykładami są zdarzenia takie jak `Open` (skoroszt został otwarty lub utworzony), `BeforeSave` (próba zapisania skorosztu) oraz `NewSheet` (dodano nowy arkusz).
- **Zdarzenia dotyczące arkuszy** — występują w kontekście określonego arkusza. Przykładami są zdarzenia takie jak `Change` (zmieniła się zawartość komórki w arkuszu), `SelectionChange` (użytkownik przemieścił wskaźnik komórki) oraz `Calculate` (przeliczenie arkusza).
- **Zdarzenia dotyczące wykresów** — występują w kontekście określonego wykresu. Należą do nich na przykład `Select` (zaznaczono obiekt na wykresie) oraz `SeriesChange` (zmodyfikowano wartość punktu danych w serii danych).

W celu monitorowania zdarzeń dla wykresu wbudowanego należy wykorzystać moduł klasy, jak zaprezentowano w rozdziale 16.

- **Zdarzenia dotyczące aplikacji** — występują w kontekście określonej aplikacji Excela. Należą do nich NewWorkbook (utworzono nowy skoroszyt), WorkbookBeforeClose (próba zamknięcia któregoś skoroszytu) oraz SheetChange (zmodyfikowano zawartość komórki w dowolnym z otwartych skoroszytów). W celu monitorowania zdarzeń dotyczących aplikacji należy wykorzystać moduł klasy.
- **Zdarzenia dotyczące formularzy UserForm** — występują w kontekście określonego formularza *UserForm* lub obiektu zawartego w formularzu *UserForm*. Na przykład z formularzem *UserForm* jest związane zdarzenie Initialize (przed wyświetleniem formularza *UserForm*), natomiast z przyciskiem w formularzu *UserForm* jest związane zdarzenie Click (kliknięcie przycisku).
- **Zdarzenia niezwiązane z obiektami** — ostatnia kategoria dotyczy dwóch użytecznych zdarzeń poziomu aplikacji: OnTime oraz OnKey. Zdarzenia te działają nieco inaczej niż pozostałe.

Ten rozdział zorganizowano według wymienionej wyżej listy. W każdym podrozdziale prezentujemy przykłady, które ilustrują poszczególne zdarzenia.

Sekwencje zdarzeń

Jak się nietrudno domyślić, niektóre działania powodują wystąpienie wielu zdarzeń. Na przykład kiedy do skoroszytu wstawiany jest nowy arkusz, zachodzą trzy zdarzenia poziomu aplikacji:

- WorkbookNewSheet, które zachodzi podczas dodawania nowego arkusza;
- SheetDeactivate, które zachodzi podczas dezaktywacji aktywnego arkusza;
- SheetActivate, które zachodzi podczas aktywowania dodanego arkusza.



Sekwencje zdarzeń są nieco bardziej złożone, niż mogłoby się wydawać. Wymienione powyżej zdarzenia zachodzą na poziomie aplikacji. Podczas dodawania nowego arkusza zachodzą dodatkowe zdarzenia na poziomie skoroszytu i arkusza.

Na tym etapie powinieneś po prostu zapamiętać, że zdarzenia zachodzą w określonej kolejności, a wiedza o tym, jaka jest ta kolejność, ma kluczowe znaczenie podczas pisania procedur obsługi. W dalszej części rozdziału wyjaśniono, jak uzyskać informacje o kolejności zdarzeń dla określonego działania (zobacz punkt „Monitorowanie zdarzeń poziomu aplikacji”).

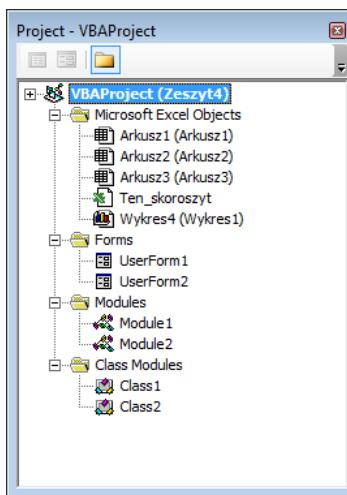
Gdzie należy umieścić procedury obsługi zdarzeń?

Nowicjusze w programowaniu w VBA często zastanawiają się, dlaczego napisane przez nich procedury obsługi zdarzeń nie wykonują się, mimo iż zdarzenie wystąpiło. Niemal zawsze przyczyną jest umieszczenie procedur w niewłaściwym miejscu.

W edytorze Visual Basic wszystkie projekty są wyszczególnione w oknie *Project*. Składniki projektu tworzą listę hierarchiczną, taką jak pokazana na rysunku 17.1.

Rysunek 17.1.

Składniki poszczególnych projektów VBA są wyszczególnione w oknie *Project*



Wszystkie elementy wymienione poniżej posiadają swoje własne moduły kodu:

- obiekty typu Sheet (np. Arkusz1, Arkusz2 itd.) — tego modułu powinieneś używać do przechowywania procedur obsługi zdarzeń dotyczących danego arkusza;
- obiekty typu Chart (tzn. arkusze wykresów) — tego modułu powinieneś używać do przechowywania procedur obsługi zdarzeń dotyczących wykresu;
- obiekt ThisWorkbook — tego modułu powinieneś używać do przechowywania procedur obsługi zdarzeń dotyczących całego skoroszytu;
- moduły VBA ogólnego przeznaczenia (procedur obsługi zdarzeń nigdy nie należy umieszczać w modułach ogólnego przeznaczenia, czyli inaczej mówiąc, w modułach, które nie są związane z obiektami);
- obiekty typu UserForm — tego modułu powinieneś używać do przechowywania procedur obsługi zdarzeń związanych z formularzami *UserForm* lub formatami umieszczonymi na takich formularzach;
- moduły klas — takich modułów powinieneś używać do przechowywania specjalnych procedur obsługi zdarzeń, takich jak zdarzenia na poziomie aplikacji czy zdarzenia generowane przez wykresy osadzone na arkuszu.

Chociaż procedury obsługi zdarzeń muszą być umieszczone w odpowiednich modułach, można w nich wywoływać standardowe procedury zapisane w innych modułach. Na przykład w zaprezentowanej poniżej procedurze obsługi zdarzenia, umieszczonej w module obiektu ThisWorkbook, wywoływana jest procedura *WorkbookSetup*, która może być zapisana w standardowym module VBA:

```
Private Sub Workbook_Open()
    Call WorkbookSetup
End Sub
```

Obsługa zdarzeń w starszych wersjach Excela

Wersje Excela poprzedzające wydanie pakietu Office 97 także obsługiwały zdarzenia, ale techniki programistyczne wymagane do skorzystania z nich znaczco różniły się od tych, które zostały opisane w niniejszym rozdziale.

Jeżeli na przykład zdefiniujemy procedurę `Auto_Open` i zapiszemy ją w zwykłym module VBA, zostanie ona uruchomiona podczas otwierania skoroszytu. Począwszy od Excela 97, obok procedury `Auto_Open` wprowadzono procedurę obsługi zdarzeń `Workbook_Open`, zapisaną w module kodu obiektu `ThisWorkbook` i wykonywaną przed procedurą `Auto_Open`.

Przed pojawiением się Excela 97 często istniała konieczność ręcznego definiowania zdarzeń. Na przykład, aby wykonać procedurę za każdym razem, kiedy w komórce były wprowadzane dane, należało wykonać następujące polecenie:

```
Sheets("Arkusz1").OnEntry = "ValidateEntry"
```

To polecenie powoduje, że Excel wykona procedurę `ValidateEntry`, jeżeli do komórki zostaną wprowadzone dane. W Excelu 97 i wersjach późniejszych wystarczy utworzyć procedurę `Worksheet_Change` i zapisać ją w module kodu obiektu `Arkusz1`.

Dla zachowania zgodności ze starszymi wersjami w Excelu 97 i wersjach nowszych w dalszym ciągu obsługiwany jest starszy mechanizm (choć nie jest on udokumentowany w systemie pomocy). O tych mechanizmach i zdarzeniach wspominamy tylko dlatego, że ciągle możesz spotkać się ze starymi arkuszami, w których mogą znajdować się takie nieco „dziwne” sekwencje procedur obsługi zdarzeń.

Wyłączanie obsługi zdarzeń

Domyślnie wszystkie zdarzenia są włączone. Aby wyłączyć wszystkie zdarzenia, należy wykonać poniższą instrukcję VBA:

```
Application.EnableEvents = False
```

Aby ponownie włączyć usługę zdarzeń, należy wykonać następujące polecenie:

```
Application.EnableEvents = True
```



Wyłączenie obsługi zdarzeń nie ma wpływu na zdarzenia generowane przez formanty formularzy `UserForm` — na przykład na zdarzenie `Click` wygenerowane poprzez kliknięcie przycisku `CommandButton` w formularzu `UserForm`.

Dlaczego czasami zachodzi potrzeba wyłączania zdarzeń? Jednym z najczęstszych powodów jest zapobieganie powstawaniu nieskończonych pętli kaskadowych zdarzeń.

Na przykład założmy, że w komórce A1 arkusza zawsze musi znajdować się liczba o wartości mniejszej lub równej 12. Aby obsłużyć tę sytuację, można napisać kod sprawdzający poprawność danych za każdym razem, kiedy w komórce będą wprowadzane dane. W tym przypadku należy monitorować zdarzenie `Change` obiektu `Worksheet` za pomocą procedury o nazwie `Worksheet_Change`. Procedura sprawdzi dane wprowadzone przez użytkownika i jeżeli wartość wprowadzona w komórce nie będzie mniejsza lub równa 12, wyświetli komunikat i wyzeruje wprowadzoną wartość. Problem polega na tym, że zerowanie zapisu za pomocą kodu VBA generuje nowe zdarzenie `Change`, co powoduje ponowne

wykonanie procedury obsługi zdarzenia. Takie działanie jest niepożądane, a zatem przed wyzerowaniem komórki należy wyłączyć zdarzenia, a następnie ponownie je włączyć, aby można było monitorować następną operację wprowadzania danych.

Innym sposobem zabezpieczenia się przed nieskończonymi pętlami kaskadowych zdarzeń jest zadeklarowanie zmiennej statycznej typu Boolean na początku procedury obsługi zdarzeń, takiej jak na przykład:

```
Static AbortProc As Boolean
```

Kiedy procedura ma wykonać zmiany, należy ustawić zmienną AbortProc na wartość True (w innym przypadku należy się upewnić, czy jest ustawiona na wartość False). Na początku procedury należy wstawić następujący kod:

```
If AbortProc Then  
    AbortProc = False  
    Exit Sub  
End If
```

Kiedy nastąpi ponowne wykonanie procedury obsługi zdarzeń, to dzięki wartości True zmiennej AbortProc procedura natychmiast zakończy działanie. Dodatkowo zmienna AbortProc zostanie ponownie ustawiona na wartość False.



Praktyczny przykład sprawdzania poprawności danych zaprezentowano w punkcie „Monitorowanie zakresu w celu sprawdzenia poprawności danych” w dalszej części tego rozdziału.



Wyłączenie zdarzeń w Excelu dotyczy wszystkich skoroszytów. Jeśli na przykład wyłączymy zdarzenia w procedurze, a następnie otworzymy inny skoroszyt, w którym zdefiniowano procedurę Workbook_Open, procedura ta nie zostanie wykonana.

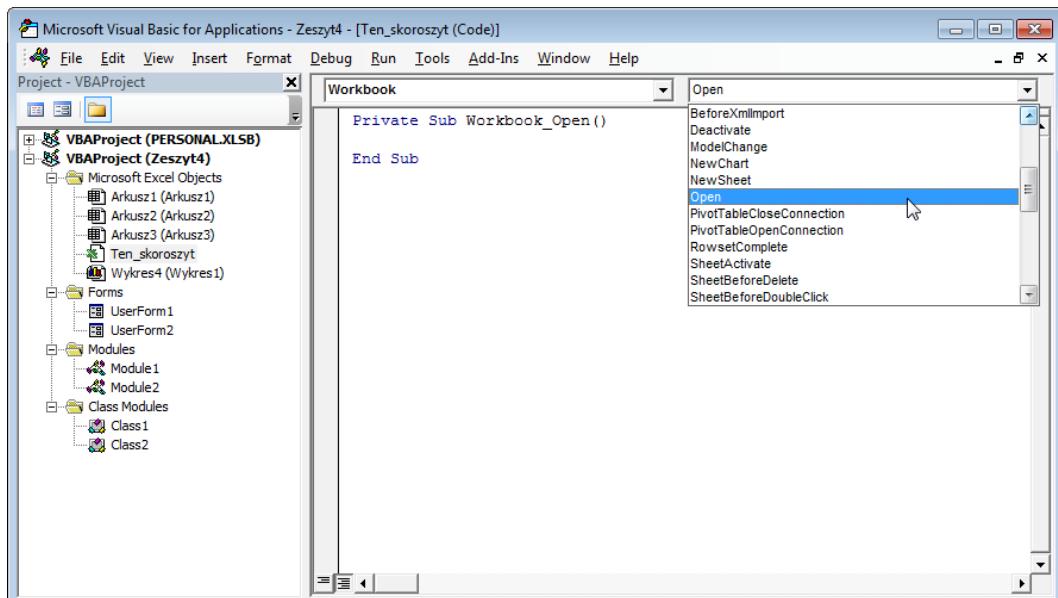
Wprowadzanie kodu procedury obsługi zdarzeń

Każda procedura obsługi zdarzeń posiada nazwę, która jest z góry określona. Poniżej podano kilka przykładów nazw procedur obsługi zdarzeń:

- Worksheet_SelectionChange
- Workbook_Open
- Chart_Activate
- Class_Initialize

Procedurę można zadeklarować, wpisując ją ręcznie, ale znacznie lepszym rozwiązaniem jest pozostawienie tego zadania edytoriowi VBE.

Na rysunku 17.2 pokazano moduł kodu dla obiektu ThisWorkbook. Aby wstawić deklarację procedury, należy wybrać skoroszyt z listy obiektów po lewej stronie oraz zdarzenie z listy procedur po prawej stronie. Wykonanie tych czynności spowoduje utworzenie szkieletu procedury, który zawiera wiersz deklaracji procedury oraz instrukcję End Sub.



Rysunek 17.2. Najlepszym sposobem utworzenia procedury obsługi zdarzenia jest wykorzystanie edytora Visual Basic

Jeżeli na przykład wybierzemy Workbook z listy obiektów i Open z listy procedur, edytor Visual Basic wstawi następującą (pustą) procedurę:

```
Private Sub Workbook_Open()
```

```
End Sub
```

Właściwy kod powinien oczywiście zostać umieszczony pomiędzy tymi dwiema instrukcjami.



Zwrć uwagę, że po wybraniu danego elementu z listy obiektów, na przykład skoroszytu (Workbook) czy arkusza (Worksheet), VBE automatycznie wstawia deklarację procedury. Zazwyczaj nie jest to deklaracja procedury, której oczekujesz. Aby ją zmienić, po prostu wybierz z listy rozwijanej po prawej stronie okna potrzebne Ci zdarzenie i VBE automatycznie zamieni deklarację procedury na odpowiednią dla wybranego zdarzenia.

Procedury obsługi zdarzeń z argumentami

W niektórych procedurach obsługi zdarzeń wykorzystuje się argumenty. Na przykład można utworzyć procedurę obsługi zdarzeń w celu monitorowania zdarzenia SheetActivate dla skoroszytu. Jeżeli skorzystamy z techniki opisanej w poprzednim punkcie, edytor Visual Basic utworzy następującą procedurę:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
```

```
End Sub
```

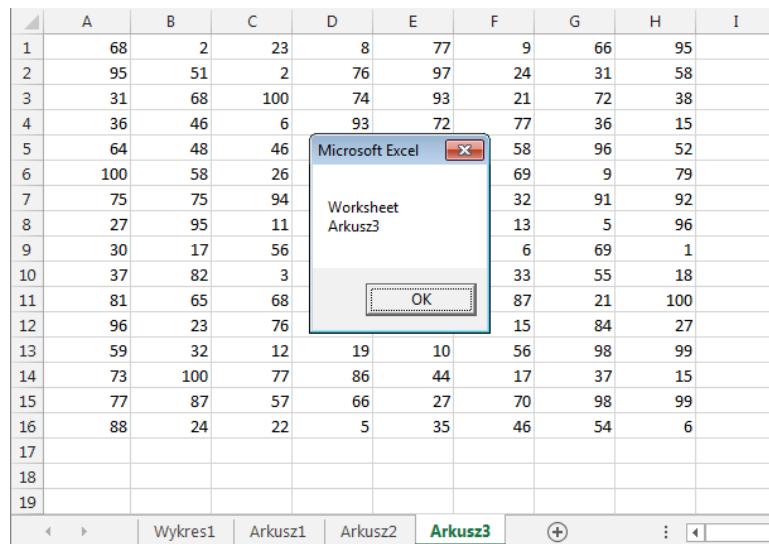
W tej procedurze wykorzystano jeden argument (`Sh`), który reprezentuje uaktywniony arkusz. W tym przypadku argument `Sh` zadeklarowano jako dane typu `Object`, a nie typu `Worksheet`. Wynika to z faktu, że uaktywniany arkusz może być także arkuszem wykresu (typu `Chart`).

W kodzie można oczywiście wykorzystywać dane przekazane jako argumenty. Poniższa procedura jest wykonywana podczas każdego uaktywnienia arkusza. Wyświetla typ i nazwę uaktywnianego arkusza dzięki wykorzystaniu funkcji VBA `TypeName` oraz właściwości `Name` obiektu przekazanego w postaci argumentu:

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    MsgBox TypeName(Sh) & vbCrLf & Sh.Name
End Sub
```

Na rysunku 17.3 przedstawiono komunikat, jaki pojawia się na ekranie, kiedy aktywowany zostanie Arkusz3.

Rysunek 17.3.
Okno komunikatu,
które zostało
wywołane przez
zdarzenie
`SheetActivate`



W niektórych procedurach obsługi zdarzeń wykorzystuje się argument `Cancel` typu `Boolean`. Na przykład deklaracja zdarzenia skoroszytu `BeforePrint` jest następująca:

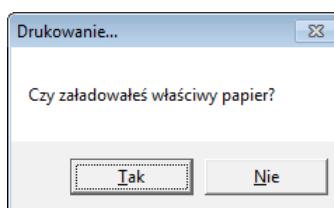
```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
```

Wartość argumentu `Cancel` przekazanego do procedury to `False`, aczkolwiek kod może ustawić wartość `Cancel` na `True`, co spowoduje anulowanie drukowania. Taka sytuacja została przedstawiona w poniższym przykładzie:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Dim Msg As String, Ans As Integer
    Msg = "Czy załadowałeś właściwy papier?"
    Ans = MsgBox(Msg, vbYesNo, "Drukowanie...")
    If Ans = vbNo Then Cancel = True
End Sub
```

Procedura Workbook_BeforePrint jest wykonywana przed wydrukowaniem skoroszytu. Zaprezentowana powyżej procedura powoduje wyświetlenie komunikatu pokazanego na rysunku 17.4. Jeżeli użytkownik kliknie przycisk Nie, argument Cancel zostanie ustawiony na wartość True i operacja drukowania zostanie przerwana.

Rysunek 17.4.
Operację drukowania
można anulować,
modyfikując argument
Cancel



Zdarzenie BeforePrint zachodzi także wtedy, kiedy użytkownik przegląda arkusz na podglądzie wydruku.

Niestety w Excelu nie ma zdarzenia BeforePrint na poziomie arkusza. Z tego powodu w kodzie nie można sprawdzić, który arkusz użytkownik chce wydrukować. Bardzo często jednak prawdziwe może być założenie, że użytkownik chce wydrukować aktywny arkusz, aczkolwiek nie można w żaden sposób sprawdzić, czy użytkownik nie wybrał do wydruku całego skoroszytu.

Zdarzenia poziomu skoroszytu

Zdarzenia poziomu skoroszytu zachodzą w kontekście określonego skoroszytu. W tabeli 17.1 wyszczególniono zdarzenia dotyczące skoroszytu wraz z krótkim opisem każdego z nich. Pełną listę zdarzeń poziomu skoroszytu znajdziesz w pomocy systemowej. Procedury ich obsługi są zapisywane w module kodu obiektu ThisWorkbook.

Tabela 17.1. Zdarzenia poziomu skoroszytu

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
Activate	Uaktywnienie skoroszytu
AddinInstall	Zainstalowanie skoroszytu jako dodatku
AddinUninstall	Odinstalowanie skoroszytu jako dodatku
AfterSave	Zapisanie skoroszytu
BeforeClose	Próba zamknięcia skoroszytu
BeforePrint	Próba wydrukowania albo przeglądania na podglądzie wydruku skoroszytu lub dowolnego fragmentu, który do niego należy
BeforeSave	Próba zapisania skoroszytu
Deactivate	Dezaktywacja skoroszytu
NewSheet	Utworzenie nowego arkusza w skoroszycie
Open	Otwarcie skoroszytu

Tabela 17.1. Zdarzenia poziomu skoroszytu (ciąg dalszy)

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
SheetActivate	Uaktywnienie dowolnego arkusza
SheetBeforeDoubleClick	Dwukrotne kliknięcie dowolnego arkusza. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku dwukrotnego kliknięcia
SheetBeforeRightClick	Kliknięcie dowolnego arkusza prawym przyciskiem myszy. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku kliknięcia prawym przyciskiem myszy
SheetCalculate	Przeliczenie dowolnego arkusza
SheetChange	Modyfikacja dowolnego arkusza przez użytkownika lub zewnętrzne łącze
SheetDeactivate	Dezaktywacja dowolnego arkusza
SheetFollowHyperlink	Kliknięcie hiperłącza w arkuszu
SheetPivotTableUpdate	Uaktualnienie tabeli przestawnej polegające na wprowadzeniu nowych danych
SheetSelectionChange	Modyfikacja zaznaczenia w dowolnym arkuszu
WindowActivate	Uaktywnienie dowolnego okna skoroszytu
WindowDeactivate	Dezaktywacja dowolnego okna skoroszytu
WindowResize	Zmiana rozmiaru dowolnego okna skoroszytu



Jeżeli musisz monitorować zdarzenia w *dowolnym* skoroszycie, powinieneś wykorzystać zdarzenia poziomu aplikacji (więcej informacji znajdziesz w podrozdziale „Zdarzenia poziomu arkusza” w dalszej części rozdziału). W pozostałej części tego podrozdziału zaprezentowano przykłady zastosowania zdarzeń poziomu skoroszytu. Wszystkie procedury należy umieścić w module kodu obiektu ThisWorkbook. Jeżeli umieścisz je w innym module kodu, po prostu nie będą działać.

Zdarzenie Open

Jednym z najczęściej monitorowanych zdarzeń poziomu skoroszytu jest zdarzenie `Open`, które następuje w momencie otwarcia skoroszytu (lub dodatku). Wystąpienie zdarzenia powoduje wykonanie procedury `Workbook_Open`. Procedura ta może wykonać najróżniejsze zadania, najczęściej są to następujące operacje:

- Wyświetlanie komunikatów powitalnych.
- Otwieranie innych skoroszytów.
- Konfigurowanie menu podręcznego.
- Uaktywnianie określonego arkusza lub komórki.
- Sprawdzanie, czy zostały spełnione określone warunki, na przykład czy został zainstalowany określony dodatek.
- Konfigurowanie określonych właściwości automatycznych — na przykład definiowanie określonych kombinacji klawiszy (więcej informacji znajdziesz w punkcie „Zdarzenie OnKey” w dalszej części rozdziału).

- Ustawianie właściwości ScrollArea arkusza (która nie jest zapisana w skoroszycie).
- Ustawienie zabezpieczenia UserInterfaceOnly dla arkuszy w taki sposób, aby kod mógł działać na zabezpieczonych arkuszach (to ustawienie jest argumentem metody Protect i nie jest zapisane w skoroszycie).



Utworzenie odpowiednich procedur obsługi zdarzeń w żaden sposób nie gwarantuje, że będą one wykonywane. Jeżeli użytkownik przytrzyma klawisz *Shift* podczas otwierania skoroszytu, procedura Workbook_Open nie zostanie uruchomiona. Procedura nie będzie również uruchomiona w przypadku otwierania arkusza z wyłączoną obsługą makr.

Poniżej zamieszczono prosty przykład procedury Workbook_Open. Wykorzystano w niej funkcję VBA Weekday w celu określenia dnia tygodnia. Jeżeli jest to piątek, wyświetli się komunikat przypominający użytkownikowi o wykonaniu tygodniowej kopii zapasowej plików. Dla pozostałych dni tygodnia nie są wykonywane żadne działania.

```
Private Sub Workbook_Open()
    If Weekday(Now) = vbFriday Then
        Msg = "Dziś jest piątek. Pamiętaj, aby wykonać "
        Msg = Msg & " cotygodniową kopię zapasową danych!"
        MsgBox Msg, vbInformation
    End If
End Sub
```

Zdarzenie Activate

Poniższa procedura wykonywana jest każdorazowo podczas uaktywniania skoroszytu. W wyniku jej działania aktywne okno zostaje rozciągnięte na cały ekran.

```
Private Sub Workbook_Activate()
    ActiveWindow.WindowState = xlMaximized
End Sub
```

Zdarzenie SheetActivate

Procedura przedstawiona poniżej jest wykonywana za każdym razem, kiedy użytkownik uaktywni dowolny arkusz skoroszytu. W przypadku zwykłego arkusza kod wybiera komórkę A1. W przypadku arkusza będącego wykresem nic się nie dzieje. W procedurze wykorzystano funkcję VBA TypeName w celu sprawdzenia, czy uaktywniany arkusz jest zwykłym arkuszem (w odróżnieniu od arkuszy wykresów).

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    If TypeName(Sh) = "Worksheet" Then Range("A1").Select
End Sub
```

Procedura przedstawiona poniżej ilustruje metodę alternatywną, która nie wymaga sprawdzania typu arkusza — w tej wersji procedury błędy są po prostu ignorowane.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    On Error Resume Next
    Range("A1").Select
End Sub
```

Zdarzenie NewSheet

Poniższa procedura jest wykonywana, kiedy do skoroszytu dodawany jest nowy arkusz. Arkusz jest przekazywany do procedury jako argument. Ponieważ nowy arkusz może być zwykłym arkuszem albo arkuszem wykresu, procedura sprawdza jego typ. Jeżeli jest to zwykły arkusz, kod wstawia datę i znacznik czasowy w komórce A1 nowego arkusza.

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    If TypeName(Sh) = "Worksheet" Then
        Sh.Cells.ColumnWidth = 35
        Sh.Range("A1") = "Arkusz został dodany " & Now()
    End If
End Sub
```

Zdarzenie BeforeSave

Zdarzenie BeforeSave zachodzi, zanim skoroszyt zostanie zapisany. Jak wiadomo, wybranie polecenia *PLIK/Zapisz* czasami powoduje wyświetlenie okna dialogowego *Zapisywanie jako*. Taka sytuacja występuje w przypadku, gdy skoroszyt nigdy wcześniej nie był zapisywany lub jeżeli otwarto go w trybie tylko do odczytu.

Kiedy procedura Workbook_BeforeSave zostanie wywołana, otrzymuje argument (SaveAsUI), który pozwala określić, czy okno dialogowe *Zapisywanie jako* ma zostać wyświetcone na ekranie. Użycie tego argumentu pokazano w poniższym przykładzie:

```
Private Sub Workbook_BeforeSave
    (ByVal SaveAsUI As Boolean, Cancel As Boolean)
    If SaveAsUI Then
        MsgBox "Sprawdź, czy zapisujesz ten plik na dysku J:."
    End If
End Sub
```

Jeżeli użytkownik spróbuje zapisać skoroszyt, zostanie wywołana procedura Workbook_BeforeSave. Jeżeli okno dialogowe *Zapisywanie jako* ma zostać wyświetcone, zmienna SaveAsUI będzie miała wartość True. Procedura zaprezentowana powyżej sprawdza zmienną i wyświetla komunikat tylko wtedy, jeżeli okno dialogowe *Zapisywanie jako* zostanie wyświetcone. Jeżeli w procedurze argument Cancel zostanie ustawiony na wartość True, plik nie zostanie zapisany (lub okno *Zapisywanie jako* nie zostanie wyświetcone).

Zdarzenie Deactivate

W tym przykładzie zilustrowano działanie zdarzenia Deactivate. Procedura obsługi tego zdarzenia zaprezentowana poniżej jest wykonywana przy każdej próbie dezaktywacji skoroszytu. Jej działanie polega na uniemożliwieniu dezaktywacji. Kiedy wystąpi zdarzenie Deactivate, kod ponownie uaktywni skoroszyt i wyświetli odpowiedni komunikat.

```
Private Sub Workbook_Deactivate()
    Me.Activate
    MsgBox "Niestety, nie możesz opuścić tego skoroszytu."
End Sub
```



Nie polecam stosowania procedur (podobnych do przedstawionej powyżej), które próbują przejąć kontrolę nad Exceliem. Może to być bardzo frustrujące i mylące dla użytkownika. Zamiast stosować takie procedury, lepiej nauczyć użytkowników poprawnego korzystania z aplikacji.

Ten prosty przykład pokazuje, jak ważna jest wiedza na temat kolejności zdarzeń. Jeżeli wypróbujemy tę procedurę, zauważymy, że działa poprawnie podczas próby uaktywnienia innego skoroszytu. Należy jednak pamiętać, że zdarzenie poziomu skoroszytu Deactivate jest generowane także w wyniku następujących działań:

- Próba zamknięcia skoroszytu.
- Próba otwarcia nowego skoroszytu.
- Próba minimalizacji skoroszytu.

W związku z tym procedura może działać zupełnie inaczej, niż zamierzałyśmy. Co prawda zabezpiecza przed bezpośrednim uaktywnieniem innego skoroszytu, ale użytkownik może zamknąć skoroszyt, zminimalizować go lub otworzyć nowy. Komunikat wyświetli się na ekranie, ale działania i tak zostaną wykonane.

Zdarzenie BeforePrint

Zdarzenie BeforePrint występuje podczas żądania wydrukowania arkusza (wyświetlenia go na podglądzie wydruku), ale przed faktycznym wydrukowaniem (wyświetleniem podglądu). W procedurze obsługi zdarzenia wykorzystywany jest argument Cancel, który pozwala na anulowanie drukowania lub wyświetlenia podglądu, jeżeli nada się mu wartość True. Niestety nie ma sposobu sprawdzenia, czy zdarzenie BeforePrint zostało wygenerowane w wyniku żądania wydruku, czy też w wyniku próby wyświetlenia arkusza w podglądzie wydruku.

Aktualizacja nagłówka lub stopki

Opcje ustawień nagłówka i stopki w Excelu są bardzo elastyczne i dają duże możliwości, aczkolwiek nadal nie ma możliwości wydrukowania pełnej ścieżki dostępu do skoroszytu w nagłówku lub stopce strony. Procedura obsługi zdarzenia Workbook_BeforePrint zapewnia możliwość wyświetlenia w nagłówku lub stopce bieżącej zawartości komórki za każdym razem, kiedy skoroszyt jest drukowany. Kod procedury przedstawiony poniżej aktualizuje przed wydrukiem skoroszytu zawartość lewej sekcji nagłówka strony, a dokładniej wstawia do niego zawartość komórki A1 arkusza Arkusz1.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Dim sht As Object
    For Each sht In ThisWorkbook.Sheets
        sht.PageSetup.LeftFooter = Worksheets("Arkusz1").Range("A1")
    Next sht
End Sub
```

Procedura przetwarza w pętli wszystkie arkusze skoroszytu i ustawia właściwość LeftFooter obiektu PageSetup na wartość komórki A1 arkusza Arkusz1.

Ukrywanie kolumn przed wydrukiem

Kolejny przykład wykorzystuje procedurę Workbook_BeforePrint do ukrycia kolumn B:D arkusza Arkusz1 przed rozpoczęciem wydruku lub podglądu wydruku.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    ' ukrywa kolumny B:D arkusza Arkusz1 przed rozpoczęciem wydruku lub podglądu wydruku.
    Worksheets("Arkusz1").Range("B:D").EntireColumn.Hidden = True
End Sub
```

Idealnym rozwiązaniem byłoby odkrywanie tych ukrytych kolumn zaraz po zakończeniu drukowania. Z pewnością byłoby miło, gdyby Excel obsługiwał również zdarzenie AfterPrint, ale niestety takie zdarzenie nie istnieje. Nie zmienia to jednak faktu, że istnieje inna metoda pozwalająca na automatyczne odkrywanie ukrytych kolumn arkusza. Zmodyfikowana wersja naszej procedury ustawia zdarzenie OnTime tak, że po upływie 5 sekund od zakończenia drukowania lub podglądu wydruku wywołuje procedurę o nazwie UnhideColumns.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    ' ukrywa kolumny B:D arkusza Arkusz1 przed rozpoczęciem wydruku lub podglądu wydruku.
    Worksheets("Arkusz1").Range("B:D").EntireColumn.Hidden = True
    Application.OnTime Now() + TimeValue("0:00:05"), "UnhideColumns"
End Sub
```

Procedura UnhideColumns powinna zostać umieszczona w standardowym module kodu VBA.

```
Sub UnhideColumns()
    Worksheets("Arkusz1").Range("B:D").EntireColumn.Hidden = False
End Sub
```



Skoroszyt z tym przykładem (*Ukrywanie kolumn przed drukowaniem.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci



Więcej szczegółowych informacji na temat zdarzenia OnTime znajdziesz w punkcie „Zdarzenie OnTime” w dalszej części rozdziału.

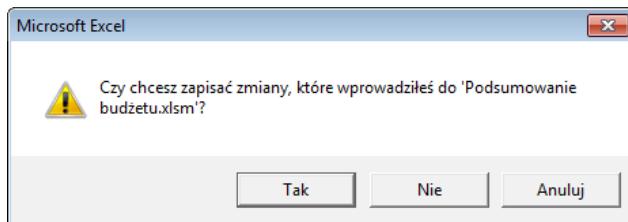
Zdarzenie BeforeClose

Zdarzenie BeforeClose zachodzi przed zamknięciem skoroszytu. Często używa się go w połączeniu z procedurą obsługi zdarzenia Workbook_Open. Na przykład procedurę Workbook_Open można wykorzystać do utworzenia menu podręcznego, a następnie za pomocą procedury Workbook_BeforeClose usunąć to menu przed zamknięciem skoroszytu. W ten sposób menu podręczne aplikacji będzie dostępne tylko wtedy, gdy skoroszyt będzie otwarty.

Niestety, zdarzenie Workbook_BeforeClose nie zostało zbyt dobrze zaimplementowane. Jeśli więc spróbujesz zamknąć skoroszyt, który jeszcze nie został zapisany na dysku, Excel wyświetli na ekranie okno dialogowe z pytaniem, czy chcesz zapisać skoroszyt przed zamknięciem, tak jak to zostało przedstawione na rysunku 17.5. Problem w tym, że zanim użytkownik zobaczył ten komunikat, zdarzenie Workbook_BeforeClose już miało miejsce, stąd nawet jeżeli użytkownik wybierze przycisk *Anuluj*, niczego to nie zmieni, ponieważ procedura obsługi zdarzenia i tak została już dawno wykonana.

Rysunek 17.5.

Kiedy na ekranie pojawia się ten komunikat, procedura Workbook_BeforeClose została już wykonana



Rozważmy następującą sytuację: chcemy wyświetlić menu użytkownika, kiedy otworzy się określony skoroszyt. Do utworzenia menu w momencie otwierania skoroszytu wykorzystamy procedurę Workbook_Open, a następnie zastosujemy procedurę Workbook_BeforeClose w celu usunięcia menu w momencie zamknięcia skoroszytu. Te dwie procedury obsługi zdarzeń znajdują się poniżej. Obie wywołują inne procedury, których tu nie zamieszczono.

```
Private Sub Workbook_Open()
    Call CreateShortcutMenuItems
End Sub

Private Sub Workbook_BeforeClose (Cancel As Boolean)
    Call DeleteShortcutMenuItems
End Sub
```

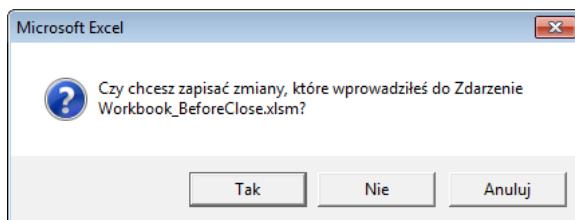
Jak zaznaczyliśmy wcześniej, okno dialogowe z pytaniem *Czy zapisać zmiany w...* zostaje wyświetlone już po wykonaniu procedury Workbook_BeforeClose. Zatem jeżeli użytkownik kliknie *Anuluj*, skoroszyt pozostanie otwarty, ale zniknie menu użytkownika, które zostało usunięte wcześniej!

Jednym z rozwiązań tego problemu jest pominięcie pytania wyświetlanego przez Excel i napisanie własnego kodu procedury Workbook_BeforeClose. Przykładową procedurę zaprezentowano poniżej:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Dim Msg As String
    If Me.Saved = False Then
        Msg = "Czy zapisać zmiany w skoroszycie "
        Msg = Msg & Me.Name & "?"
        Ans = MsgBox(Msg, vbQuestion + vbYesNoCancel)
        Select Case Ans
            Case vbYes
                Me.Save
            Case vbCancel
                Cancel = True
                Exit Sub
        End Select
    End If
    Call DeleteShortcutMenuItems
    Me.Saved = True
End Sub
```

W procedurze wykorzystano właściwość Saved obiektu Workbook w celu sprawdzenia, czy skoroszyt został zapisany. Jeżeli tak, wówczas procedura DeleteShortcutMenuItems jest wykonywana i skoroszyt zostanie zamknięty. Jeżeli jednak skoroszyt nie został jeszcze zapisany, procedura wyświetli okno dialogowe, które spełnia rolę standardowego okna wyświetlanego normalnie przez Excela (patrz rysunek 17.6). Efekty naciśnięcia poszczególnych przycisków są następujące:

Rysunek 17.6.
Komunikat wyświetlany na ekranie przez procedurę obsługi zdarzenia Workbook_BeforeClose



- Przycisk **Tak** — skoroszyt zostaje zapisany, menu jest usuwane i następnie skoroszyt zostaje zamknięty.
- Przycisk **Nie** — procedura ustawia właściwość Saved obiektu Workbook na wartość True (ale nie zapisuje samego pliku), usuwa menu i zamyka plik.
- Przycisk **Anuluj** — zdarzenie BeforeClose jest anulowane i procedura kończy działanie bez usuwania menu użytkownika.



Skoroszyt z tym przykładem (*Zdarzenie Workbook_BeforeClose.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zdarzenia poziomu arkusza

Zdarzenia związane z obiektem Worksheet należą do najbardziej użytecznych, ponieważ większość tego, co dzieje się w Excelu ma związek z arkuszami. Monitorowanie tych zdarzeń pozwala aplikacji na wykonywanie działań, które w innych sytuacjach byłyby niemożliwe do wykonania.

W tabeli 17.2 wyszczególniono zdarzenia dotyczące arkuszy wraz z krótkim opisem każdego z nich.

Tabela 17.2. Zdarzenia poziomu skoroszytu

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
Activate	Uaktywnienie arkusza
BeforeDelete	Próba usunięcia arkusza
BeforeDoubleClick	Dwukrotne kliknięcie arkusza
BeforeRightClick	Kliknięcie arkusza prawym przyciskiem myszy
Calculate	Przeliczenie arkusza
Change	Modyfikacja komórki arkusza dokonana przez użytkownika lub zewnętrzne łącze
Deactivate	Desaktywacja arkusza
FollowHyperlink	Kliknięcie hiperłącza na arkuszu
PivotTableUpdate	Aktualizacja tabeli przestawnej w arkuszu
SelectionChange	Modyfikacja zaznaczenia w arkuszu

Pamiętaj, że kod procedur obsługi zdarzeń poziomu arkusza musi być zapisany w module kodu dla określonego arkusza.



Aby szybko uaktywnić moduł kodu arkusza, trzeba kliknąć prawym przyciskiem myszy zakładkę arkusza, a następnie wybrać polecenie *View Code*.

Zdarzenie Change

Zdarzenie *Change* następuje w przypadku modyfikacji dowolnej komórki arkusza przez użytkownika lub procedurę VBA. Zdarzenie to nie nastąpi, jeżeli w wyniku obliczeń zmieni się wartość formuły lub kiedy do arkusza zostanie dodany obiekt.

Procedura *Worksheet_Change* pobiera obiekt typu *Range* jako argument *Target*. Obiekt ten reprezentuje zmodyfikowaną komórkę lub zakres, które spowodowały wystąpienie zdarzenia. Poniższa procedura zostanie wykonana, kiedy zmieni się zawartość arkusza. Jej działanie polega na wyświetleniu okna informacyjnego z adresem zakresu *Target*:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    MsgBox "Zakres " & Target.Address & " został zmodyfikowany."
End Sub
```

Aby lepiej zrozumieć rodzaje działań, które generują zdarzenie *Change* dla arkusza, wprowadźmy poprzednią procedurę w module kodu obiektu *Worksheet*. Następnie uaktywnijmy Excela i wykonajmy kilka modyfikacji za pomocą różnych technik. Za każdym razem, kiedy nastąpi zdarzenie *Change*, wyświetli się komunikat zawierający adres zakresu, który został zmodyfikowany.

Po uruchomieniu procedury odkryłem kilka interesujących zachowań. Niektóre działania nie generowały zdarzeń, choć powinny, natomiast inne, które pozornie nie powinny tego robić, generowały je!

- Zmiana formatowania komórki nie generuje zdarzenia *Change* (choć wydaje się, że powinna), natomiast kopiowanie formatów przy użyciu okna dialogowego *Wklejanie specjalne* wyzwala zdarzenie *Change*. Zdarzenie to jest również generowane po wykonaniu polecenia *Wyczyść formaty* (karta *NARZĘDZIA GŁÓWNE*, grupa opcji *Edytowanie*, menu polecenia *Wyczyść*).
- Łączenie komórek nie generuje zdarzenia *Change*, nawet w sytuacji, kiedy wskutek łączenia zawartość niektórych komórek zostaje skasowana.
- Dodawanie, edycja lub usuwanie komentarza w komórce nie wyzwala zdarzenia *Change*.
- Wciśnięcie klawisza *Delete* generuje zdarzenie, nawet jeżeli komórka była pusta.
- Modyfikacja komórek za pomocą poleceń Excela czasami wyzwala zdarzenie *Change*, a czasami nie. Na przykład sortowanie zakresu nie wyzwala zdarzenia, natomiast użycie modułu sprawdzania pisowni tak.
- Modyfikacja komórki za pomocą kodu VBA wyzwala zdarzenie *Change*.

Jak widać na podstawie tej listy, w przypadku ważnych aplikacji nie można polegać na zdarzeniu *Change* jako wskaźniku modyfikacji dokonanych w komórkach.

Monitorowanie zmian w wybranym zakresie komórek

Zdarzenie Change występuje w przypadku zmiany dowolnej komórki arkusza. Jednak w większości przypadków interesują nas modyfikacje określonej komórki lub zakresu. Procedura Worksheet_Change pobiera obiekt typu Range jako argument. Obiekt ten reprezentuje komórkę lub komórki, które uległy modyfikacji.

Załóżmy, że w arkuszu zdefiniowano zakres danych wejściowych InputRange i interesują nas wyłącznie modyfikacje wykonane w tym zakresie. Obiekt Range nie obsługuje zdarzenia Change, ale z łatwością można wykonać odpowiednie sprawdzenie wewnątrz procedury Worksheet_Change. Czynność tę zademonstrowano poniżej:

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim MRange As Range
    Set MRange = Range("InputRange")
    If Not Intersect(Target, MRange) Is Nothing Then
        MsgBox "Zmodyfikowana komórka znajduje się w zakresie danych wejściowych."
    End Sub
```

W tym przykładzie wykorzystano zmienną obiektową typu Range o nazwie MRange, która reprezentuje monitorowany zakres w arkuszu. W procedurze wykorzystano funkcję VBA Intersect w celu sprawdzenia, czy zakres Target (przekazany do procedury jako argument) przecina się z zakresem MRange. Funkcja Intersect zwraca obiekt składający się z wszystkich wspólnych komórek obu argumentów. Zwrócenie przez funkcję wartości Nothing oznacza, że oba zakresy nie mają wspólnych komórek. W procedurze wykorzystano operator Not, a zatem wyrażenie zwróci wartość True, jeżeli zakresy mają co najmniej jedną wspólną komórkę. Tak więc jeżeli zmodyfikowany zakres posiada co najmniej jedną wspólną komórkę z zakresem InputRange, wyświetli się komunikat. W innym przypadku procedura zakończy działanie.

Monitorowanie zakresu w celu pogrubienia zawartości komórek zawierających formuły

Poniżej zaprezentowano przykład monitorowania zakresu w celu pogrubienia komórek zawierających formuły.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim cell1 As Range
    For Each cell1 In Target
        If cell1.HasFormula Then cell1.Font.Bold = True
    Next cell1
End Sub
```

Ponieważ obiekt przekazany do procedury Worksheet_Change może się składać z wielokomórkowego zakresu, procedura przetwarza w pętli wszystkie komórki zakresu Target. Zawartość komórek zawierających formuły wyświetla się pogrubioną czcionką. W innym przypadku właściwość Bold jest ustawiana na wartość False.

Procedura działa, ale posiada pewien problem. Co się stanie, kiedy użytkownik usuwa cały wiersz lub kolumnę? W takim przypadku zakres Target będzie się składał z ogromnej liczby komórek. Sprawdzenie wszystkich komórek w pętli For Each zajmie bardzo dużo czasu, a procedura i tak nie znajdzie żadnych formuł.

Zmodyfikowana wersja procedury, której kod przedstawiamy poniżej, rozwiązuje ten problem poprzez zmianę zakresu Target na przecięcie zakresu Target i zakresu używanych komórek arkusza. Sprawdzenie, czy zakres Target jest Not Nothing, jest rozwiązaniem, jeśli użytkownik chce usunąć kolumnę lub wiersz znajdujące się poza używanym zakresem arkusza.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim cell As Range
    Set Target = Intersect(Target, Target.Parent.UsedRange)
    If Not Target Is Nothing Then
        For Each cell In Target
            cell.Font.Bold = cell.HasFormula
        Next cell
    End If
End Sub
```



Skoroszyt z tym przykładem (*Pogrubianie formuł.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Potencjalnie bardzo niekorzystnym efektem ubocznym działania procedur Worksheet_Change może być praktycznie wyłączenie mechanizmu wycofywania poleceń Excela. Stos danych używany przez mechanizm wycofywania poleceń jest kasowany za każdym razem, kiedy wykonywane jest dowolne makro. W przykładzie przedstawionym powyżej wpisywanie danych do komórki wywołuje zmianę jej formatowania, co w konsekwencji powoduje skasowanie stosu danych mechanizmu wycofywania poleceń.

Monitorowanie zakresu w celu sprawdzenia poprawności danych

Właściwość weryfikacji poprawności danych Excela jest bardzo przydatnym narzędziem, ale posiada istotną wadę. W przypadku wklejenia danych do komórki, dla której zastosowano tę właściwość, wklejana wartość nie tylko nie będzie sprawdzona, ale także zostaną usunięte reguły poprawności danych powiązane z komórką! Z tego powodu właściwość weryfikacji poprawności danych staje się praktycznie bezużyteczna w poważnych zastosowaniach. W niniejszym punkcie zademonstrujemy sposób wykorzystania zdarzenia Change arkusza do zdefiniowania własnej procedury sprawdzenia poprawności danych.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz dwie wersje tego przykładu. W pierwszym skoroszycie (*Sprawdzanie poprawności1.xlsxm*) wykorzystano właściwość EnabLeEvents w celu zabezpieczenia przed kaskadowym generowaniem zdarzeń Change, w drugim (*Sprawdzanie poprawności2.xlsxm*) zmienną statyczną (więcej szczegółowych informacji na ten temat znajdziesz w punkcie „Wyłączanie obsługi zdarzeń” we wcześniejszej części tego rozdziału).

Procedura Worksheet_Change, której kod prezentujemy poniżej, jest wykonywana w przypadku modyfikacji komórki przez użytkownika. Sprawdzenie poprawności ogranicza się do zakresu InputRange. Wartości wprowadzane w tym zakresie muszą być liczbami całkowitymi o wartościach pomiędzy 1 a 12.

```
Private Sub Worksheet_Change(ByVal Target As Range)
    Dim VRange As Range, cell As Range
    Dim Msg As String
    Dim ValidateCode As Variant
```

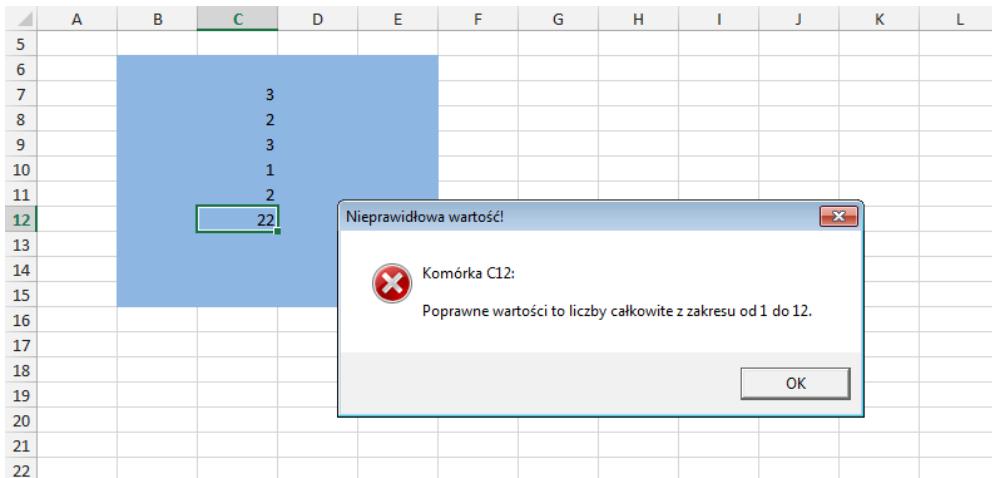
```

Set VRange = Range("InputRange")
If Intersect(VRange, Target) Is Nothing Then Exit Sub
For Each cell In Intersect(VRange, Target)
    ValidateCode = EntryIsValid(cell)
    If TypeName(ValidateCode) = "String" Then
        Msg = "Komórka " & cell.Address(False, False) & ":" &
        Msg = Msg & vbCrLf & vbCrLf & ValidateCode
        MsgBox Msg, vbCritical, "Niewłaściwa zawartość komórki"
        Application.EnableEvents = False
        cell.ClearContents
        cell.Activate
        Application.EnableEvents = True
    End If
    Next cell
End Sub

```

Procedura Worksheet_Change tworzy obiekt Range o nazwie VRange, reprezentujący zakres arkusza, dla którego sprawdzana jest poprawność. Następnie przetwarza w pętli wszystkie komórki zakresu określonego argumentem Target, który reprezentuje zmodyfikowaną komórkę lub komórki. Następuje sprawdzenie, czy poszczególne komórki mieszą się w zakresie, dla którego ma być wykonana weryfikacja poprawności. Jeżeli tak jest, komórka jest przekazywana jako argument do funkcji użytkownika EntryIsValid, zwracającej wartość True, jeżeli dane w komórce są prawidłowe.

Jeżeli wartość w komórce nie jest poprawna, funkcja zwróci ciąg znaków opisujący problem, a użytkownik uzyska informację za pomocą okna informacyjnego (rysunek 17.7). Po zamknięciu okna niepoprawny zapis jest usuwany, a komórka uaktywniana. Warto zwrócić uwagę, że przed wyzerowaniem komórki obsługa zdarzeń jest wyłączana. Gdyby zdarzenia nie były wyłączone, zerowanie komórki wygenerowałoby zdarzenie Change i w efekcie powstałyby pętla nieskończona.



Rysunek 17.7. Okno informacyjne wyświetla komunikat, jeżeli użytkownik wprowadzi niepoprawne dane

Warto również zwrócić uwagę na fakt, że nieprawidłowa wartość w komórce spowoduje wyczyszczenie stosu danych mechanizmu wycofywania poleceń.

Kod procedury EntryIsValid został przedstawiony poniżej:

```

Private Function EntryIsValid(cell) As Variant
    ' Zwraca wartość True, jeżeli komórka jest liczbą całkowitą pomiędzy 1 a 12.
    ' W innym przypadku zwracała łańcuch opisujący problem

    ' Czy wprowadzono liczbę?
    If Not WorksheetFunction.IsNumber(cell) Then
        EntryIsValid = "Wprowadź liczbę."
        Exit Function
    End If
    ' Czy jest to liczba całkowita?
    If CInt(cell) <> cell Then
        EntryIsValid = "Wprowadź liczbę całkowitą."
        Exit Function
    End If
    ' Czy liczba mieści się w zakresie pomiędzy 1 a 12?
    If cell < 1 Or cell > 12 Then
        EntryIsValid = "Wprowadź liczbę z zakresu od 1 do 12."
        Exit Function
    End If
    ' Wszystkie testy zakończyły się sukcesem
    EntryIsValid = True
End Function

```

Technika zaprezentowana powyżej działa, ale może być dosyć trudna do wdrożenia w zastosowaniach praktycznych. Czy nie byłoby łatwiej skorzystać po prostu z istniejącego mechanizmu sprawdzania poprawności danych i po prostu upewnić się, że reguły sprawdzania poprawności nie zostaną przypadkowo usunięte podczas wklejania danych do tego obszaru przez użytkownika? Problem ten został rozwiązany w naszej kolejnej procedurze, której kod zamieszczamy poniżej:

```

Private Sub Worksheet_Change(ByVal Target As Range)
    Dim VT As Long
    ' Czy wszystkie komórki w zakresie mają nadal ustawione
    ' reguły sprawdzania poprawności danych?
    On Error Resume Next
    VT = Range("InputRange").Validation.Type
    If Err.Number <> 0 Then
        Application.Undo
        MsgBox "Ostatnia wykonana operacja została anulowana, ponieważ " & vbCrLf & _
            "jej wykonanie mogłoby usunąć reguły sprawdzania poprawności danych.", _
            vbCritical
    End If
End Sub

```

Nowa procedura obsługi zdarzeń kontroluje rodzaj sprawdzania poprawności danych w zakresie InputRange, którego komórki *powinny* zawierać reguły sprawdzania poprawności danych. Jeżeli zmenna VT zawiera błąd, oznacza to, że jedna bądź więcej komórek w zakresie InputRange już nie posiada zaaplikowanych reguł sprawdzania poprawności danych. Innymi słowy, zmiana komórek arkusza była najprawdopodobniej rezultatem skopiowania danych do zakresu, w którym wcześniej zostały ustawione reguły sprawdzania poprawności danych. W takiej sytuacji procedura wywołuje metodę Undo obiektu Application, wycofuje operację wykonaną przez użytkownika i na koniec wyświetla na ekranie odpowiedni komunikat, przedstawiony na rysunku 17.8.

	A	B	C	D	E	F	G	H	I	J	K	L
5												
6				1								
7				8								
8				2								
9				2								
10				2								
11				12								
12				11								
13												
14												
15												
16												
17												
18												
19												
20												
21												
22												

Rysunek 17.8. Procedura Worksheet_Change zapewnia, że reguły sprawdzania poprawności danych nie zostaną usunięte



Procedura przedstawiona powyżej działa poprawnie tylko wtedy, kiedy komórki znajdujące się w zakresie działania procedury zawierają jakiś rodzaj sprawdzania poprawności wprowadzanych danych.



Przyjemnym efektem ubocznym użycia tej procedury jest fakt, że stos polecenia *Cofnij* nie jest kasowany.



Skoroszyt z tym przykładem (*Sprawdzanie poprawności3.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zdarzenie SelectionChange

W poniższym przykładzie zilustrowano zastosowanie zdarzenia SelectionChange. Procedura jest wykonywana za każdym razem, kiedy użytkownik zaznaczy nowy obszar w arkuszu.

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Cells.Interior.ColorIndex = xlNone
    With ActiveCell
        .EntireRow.Interior.Color = RGB(219, 229, 241)
        .EntireColumn.Interior.Color = RGB(219, 229, 241)
    End With
End Sub
```

Procedura powoduje wyróżnienie wiersza i kolumny odpowiadających aktywnej komórce, co ułatwia jej identyfikację. Pierwsza instrukcja usuwa kolor tła dla wszystkich komórek arkusza. Następnie cały wiersz i kolumna są cieniowane kolorem jasnoniebieskim. Efekt zaprezentowano na rysunku 17.9.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Projekt-1	Projekt-2	Projekt-3	Projekt-4	Projekt-5	Projekt-6	Projekt-7	Projekt-8	Projekt-9	Projekt-10	
2	sty-2007	2158	1527	3870	4863	3927	3993	2175	2143	3965	2885	
3	lut-2007	4254	28	4345	2108	412	2857	3098	87	2181	4461	
4	mar-2007	3631	1240	4208	452	3443	2965	91	2935	170	3044	
5	kwi-2007	724	4939	1619	1721	3631	3487	3581	3082	3729	2335	
6	maj-2007	3060	1034	1646	345	978	526	4422	1390	3566	3063	
7	cze-2007	394	1241	2965	1411	3545	4499	2477	735	2533	4951	
8	lip-2007	2080	3978	3304	1460	4533	3335	2675	1687	2475	3901	
9	sie-2007	411	753	732	1207	1902	4009	1793	2262	916	2887	
10	wrz-2007	2711	95	2267	2634	1944	3920	2020	402	3183	3581	
11	paź-2007	2996	4934	3932	2938	4730	1139	3776	3366	2239	1539	
12	lis-2007	2837	1116	3879	1740	1466	3628	212	780	4518	4769	
13	gru-2007	300	2917	321	1219	841	3554	1924	1786	3967	245	
14	sty-2008	1604	768	2617	3414	4732	863	2993	4184	3432	1977	
15	lut-2008	1662	1380	4590	531	4143	1758	2990	2938	4400	798	
16	mar-2008	1001	3454	4611	4852	456	46	4475	1340	859	1334	
17	kwi-2008	4407	46	4185	4868	2313	2750	4948	4525	3896	473	
18	maj-2008	3948	1292	1462	1977	2418	1816	4810	2803	4973	910	
19	cze-2008	160	2908	3834	2396	4120	2231	3689	486	4751	3174	
20	lip-2008	1131	118	3193	40	1965	424	4802	3379	3645	3293	
21	sie-2008	2480	2564	373	3893	4932	4362	4472	3707	2411	3218	
22	wrz-2008	4949	2649	2335	9	2309	3454	657	3519	658	3619	
23	paź-2008	3268	2652	2164	1898	1598	1237	1524	4752	1237	4708	
24	lis-2008	794	4821	2885	102	2618	1086	3050	3121	2202	2375	
25	gru-2008	2623	985	3737	666	3568	4685	4184	36	1830	4622	

Rysunek 17.9. Kolumna i wiersz, w których znajduje się aktywna komórka, są wyróżnione kolorem niebieskim

Nie należy wykorzystywać tej procedury, jeżeli w arkuszu zastosowano różnokolorowe tła. W takim przypadku zostaną one usunięte. Wyjątkiem są tabele, którym nadano wybrane style formatowania oraz kolor tła będące rezultatem formatowania warunkowego. W obu tych przypadkach oryginalny kolor tła zostanie zachowany. Pamiętaj jednak, że wykonanie procedury Worksheet_SelectionChange niszczy stos mechanizmu wycofywania poleceń, zatem użycie tej procedury w praktyce całkowicie wyłączy możliwość wycofywania poleceń.



Skoroszyt z tym przykładem (*Cleniowanie aktywnego wiersza i kolumny.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zdarzenie BeforeDoubleClick

Możesz utworzyć procedurę VBA, która będzie wykonywana, kiedy użytkownik dwukrotnie kliknie komórkę arkusza. Nasza kolejna procedura (przechowywana w oknie kodu obiektu Sheet) powoduje, że zawartość komórki dwukrotnie klikniętej lewym przyciskiem myszy zostaje pogrubiona (jeżeli nie jest pogrubiona) lub staje się normalna (jeżeli wcześniej była pogrubiona).

```
Private Sub Worksheet_BeforeDoubleClick(  
    ByVal Target As Range, Cancel As Boolean)  
    If Target.Style = "Good" Then  
        Target.Style = "Normal"  
    End If  
End Sub
```

```

    Else
        Target.Style = "Good"
    End If
    Cancel = True
End Sub

```

Jeżeli argument `Cancel` ma wartość `True`, domyślna akcja związana z podwójnym kliknięciem lewym przyciskiem myszy nie jest wykonywana, innymi słowy, podwójne kliknięcie nie przełączy Excela w tryb edycji zawartości komórki. Pamiętaj, że każde takie podwójne kliknięcie komórki usuwa stos danych mechanizmu wycofywania poleceń.

Zdarzenie BeforeRightClick

Kiedy użytkownik kliknie arkusz prawym przyciskiem myszy, wyświetli się menu podrzczne. Jeżeli z jakiegoś powodu chcemy wyłączyć jego wyświetlanie w określonym arkuszu, możemy w tym celu wykorzystać zdarzenie `RightClick`. Poniższa procedura ustawia argument `Cancel` na wartość `True`, co powoduje anulowanie zdarzenia `RightClick`, a tym samym anulowanie wyświetlania menu podrzcznego. Zamiast niego wyświetlane będzie okno informacyjne.

```

Private Sub Worksheet_BeforeRightClick(
    ByVal Target As Range, Cancel As Boolean)
    Cancel = True
    MsgBox "Menu podrzczne jest niedostępne."
End Sub

```

Pamiętaj, że użytkownik nadal ma możliwość wyświetlenia menu podrzcznego po naciśnięciu kombinacji klawiszy `Shift+F10`. Na szczęście tylko niewielka liczba użytkowników Excela zna tę sztuczkę...



Aby dowiedzieć się, jak można przechwycić naciśnięcie kombinacji klawiszy `Shift+F10`, zajrzyj do punktu „Zdarzenie OnKey” w dalszej części tego rozdziału. W rozdziale 21. opisano inne metody wyłączenia menu podrzcznego.

Poniżej przedstawiamy kolejny przykład procedury wykorzystującej zdarzenie `BeforeRightClick`. Procedura sprawdza, czy komórka, która została kliknięta prawym przyciskiem myszy, zawiera wartość numeryczną. Jeżeli tak, procedura wyświetla na ekranie okno *Formatowanie komórek/Liczby* i ustawia argument `Cancel` na wartość `True` (dzięki czemu standardowe menu podrzczne nie jest wyświetlane). Jeżeli komórka nie zawiera wartości liczbowej, nic specjalnego się nie dzieje — na ekranie pojawia się po prostu standardowe menu podrzczne.

```

Private Sub Worksheet_BeforeRightClick(
    ByVal Target As Range, Cancel As Boolean)
    If IsNumeric(Target) And Not IsEmpty(Target) Then
        Application.CommandBars.ExecuteMso ("NumberFormatsDialog")
        Cancel = True
    End If
End Sub

```

Zwrót uwagi, że procedura dodatkowo sprawdza, czy komórka nie jest pusta. Dzieje się tak dlatego, że w kategoriach VBA pusta komórka jest zaliczana do komórek numerycznych.

Zdarzenia dotyczące wykresów

W tym podrozdziale omówimy niektóre zdarzenia powiązane z wykresami. Domyślnie obsługa zdarzeń jest włączona tylko dla wykresów umieszczonych w osobnych arkuszach. Aby wykorzystać zdarzenia dla wykresów wbudowanych, należy utworzyć moduł klasy.



Przykłady zdarzeń dotyczących wykresów zamieszczono w rozdziale 16. Znajdziesz tam również omówienie sposobów tworzenia modułów klas w celu włączenia obsługi zdarzeń dla wykresów wbudowanych.

W tabeli 17.3 wyszczególniono zdarzenia dotyczące wykresów wraz z krótkim opisem każdego z nich.

Tabela 17.3. Zdarzenia dotyczące arkuszy wykresów

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
Activate	Uaktywnienie arkusza wykresu lub wykresu wbudowanego
BeforeDoubleClick	Dwukrotne kliknięcie arkusza wykresu lub samego wykresu. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku dwukrotnego kliknięcia
BeforeRightClick	Kliknięcie arkusza wykresu lub samego wykresu prawym przyciskiem myszy. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku kliknięcia prawym przyciskiem myszy
Calculate	Wykreślenie nowych lub zmodyfikowanych danych
Deactivate	Desaktywacja wykresu
MouseDown	Przyciśnięcie przycisku myszy w czasie, kiedy wskaźnik myszy znajduje się nad wykresem
MouseMove	Zmiana pozycji wskaźnika myszy znajdującego się nad wykresem
MouseUp	Zwolnenie przycisku myszy, kiedy wskaźnik myszy znajduje się nad wykresem
Resize	Zmiana rozmiaru wykresu
Select	Zaznaczenie elementu na wykresie
SeriesChange	Modyfikacja wartości punktu danych należącego do wykresu

Zdarzenia dotyczące aplikacji

W poprzednich podrozdziałach omówiono zdarzenia dotyczące skoroszytów i arkuszy. Zdarzenia te są monitorowane dla określonego skoroszytu. Aby monitorować zdarzenia dla wszystkich otwartych skoroszytów lub wszystkich arkuszy, należy wykorzystać zdarzenia poziomu aplikacji.

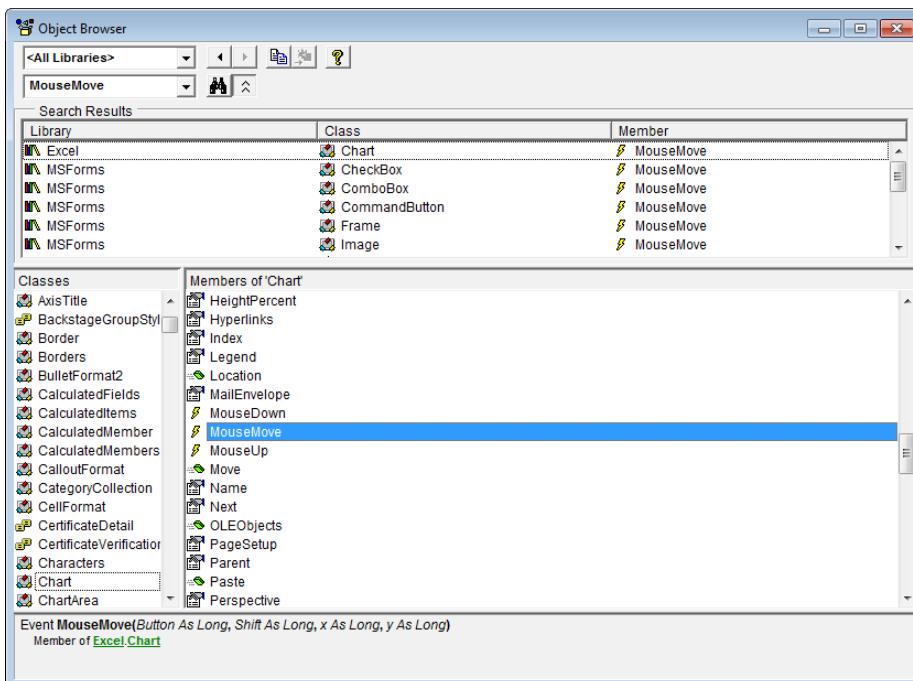


Tworzenie procedur obsługi zdarzeń poziomu aplikacji zawsze wymaga zastosowania modułu klasy oraz pewnych czynności przygotowawczych.

Wykorzystanie przeglądarki obiektów do wyszukiwania zdarzeń

Przeglądarka obiektów (*Object Browser*) umożliwia uzyskanie informacji na temat obiektów oraz ich właściwości i metod. Za jej pomocą można się także dowiedzieć, które obiekty obsługują określone zdarzenia. Założymy dla przykładu, że chcemy się dowiedzieć, jakie obiekty obsługują zdarzenie *MouseMove*. W tym celu uaktywniamy edytor *Visual Basic* i wciskamy klawisz *F2* w celu wyświetlenia okna przeglądarki obiektów. Wybieramy pozycję *<All Libraries>*, a następnie wpisujemy *MouseMove* i klikamy ikonę lornetki (patrz rysunek poniżej).

Przeglądarka obiektów wyświetli listę pasujących pozycji. Zdarzenia są oznaczone małą, żółtą błyskawicą. Na podstawie tej listy można się dowiedzieć, które obiekty obsługują zdarzenie *MouseMove*. Większość obiektów to formanty biblioteki MSForms zawierającej m.in. formularz *UserForm*, ale zdarzenie *MouseMove* obsługuje także obiekt *Chart* Excela.



Lista jest podzielona na trzy kolumny: *Library* (biblioteka), *Class* (klasa) oraz *Members* (składowe). Wyszukiwany element może się znaleźć w każdej z tych kolumn. Wiąże się z tym bardzo istotna uwaga: nazwa zdarzenia, obiektu lub składowej należąca do jednej biblioteki może być taka sama, jak nazwa zdarzenia, obiektu lub składowej należącej do innej biblioteki, choć z reguły nie spełniają one identycznych funkcji. Zatem przed użyciem elementu wyświetlonego w przeglądarce obiektów należy przestudiować jego składnię, która jest wyświetlana na pasku stanu po kliknięciu elementu w przeglądarce. Czasami może się okazać, że w określonej klasie lub bibliotece zdarzenie jest obsługiwane w inny sposób.

W tabeli 17.4 wyszczególniono zdarzenia dotyczące aplikacji wraz z krótkim opisem każdego z nich. W Excelu 2010 pojawiło się kilka nowych zdarzeń, które obsługują chronione widok okien oraz tabele przestawne. Więcej szczegółowych informacji na ten temat znajdziesz w pomocy systemowej.

Tabela 17.4. Najczęściej wykorzystywane zdarzenia dotyczące obiektu Application

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
AfterCalculate	Obliczenia zostały zakończone, dane zostały odświeżone i nie ma żadnych oczekujących zapytań
NewWorkbook	Utworzenie nowego skoroszytu
SheetActivate	Uaktywnienie dowolnego arkusza
SheetBeforeDoubleClick	Dwukrotne kliknięcie dowolnego arkusza. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku dwukrotnego kliknięcia
SheetBeforeRightClick	Kliknięcie dowolnego arkusza prawym przyciskiem myszy. To zdarzenie zachodzi przed domyślnym działaniem wykonywanym w przypadku kliknięcia prawym przyciskiem myszy
SheetCalculate	Przeliczenie dowolnego arkusza
SheetChange	Modyfikacja komórki dowolnego arkusza dokonana przez użytkownika lub zewnętrzne łącze
SheetDeactivate	Dezaktywacja dowolnego arkusza
SheetFollowHyperlink	Kliknięcie hiperłącza
SheetPivotTableUpdate	Aktualizacja danych na wykresie
SheetSelectionChange	Modyfikacja zaznaczenia w dowolnym arkuszu poza arkuszami będącymi wykresami
WindowActivate	Uaktywnienie dowolnego okna skoroszytu
WindowDeactivate	Dezaktywacja dowolnego okna skoroszytu
WindowResize	Zmiana rozmiaru dowolnego okna skoroszytu
WorkbookActivate	Uaktywnienie dowolnego skoroszytu
WorkbookAddinInstall	Zainstalowanie skoroszytu jako dodatku
WorkbookAddinUninstall	Odinstalowanie skoroszytu będącego dodatkiem
WorkbookBeforeClose	Zamknięcie dowolnego otwartego skoroszytu
WorkbookBeforePrint	Wydrukowanie dowolnego otwartego skoroszytu
WorkbookBeforeSave	Zapisanie dowolnego otwartego skoroszytu
WorkbookDeactivate	Dezaktywacja otwartego skoroszytu
WorkbookNewSheet	Utworzenie nowego arkusza w otwartym skoroszycie
WorkbookOpen	Otwarcie skoroszytu

Włączanie obsługi zdarzeń poziomu aplikacji

Aby wykorzystać zdarzenia poziomu aplikacji, wykonaj następujące czynności:

1. Utwórz nowy moduł klasy.
2. Wpisz nazwę tego modułu w polu *Name* okna *Properties* utworzonego modułu klasy.

W języku VBA każdemu nowemu modułowi klasy jest nadawana domyślna nazwa, na przykład Class1, Class2 itd. Zalecanym rozwiążaniem jest nadanie modułowi klasy opisowej nazwy, na przykład c1sApp.

3. W module klasy zadeklaruj publiczny obiekt Application z użyciem słowa kluczowego WithEvents. Na przykład:

```
Public WithEvents XL As Application
```

4. Utwórz zmienną, za pomocą której będziesz odwoływać się do zadeklarowanego obiektu Application w module klasy.

Powinna to być zmienna obiektowa poziomu modułu zadeklarowana w module VBA (nie w module klasy). Oto przykład:

```
Dim X As New c1sApp
```

5. Powiąż zadeklarowany obiekt z obiektem Application.

Taka operacja często jest wykonywana w procedurze Workbook_Open, na przykład:

```
Set X.XL = Application
```

6. Napisz procedury obsługi zdarzeń dla obiektu XL w module klasy.



Opisana procedura jest niemal identyczna z tą, która należało wykonać w celu wykorzystania zdarzeń powiązanych z wykresami osadzonymi na arkuszu. Więcej informacji na ten temat znajdziesz w rozdziale 16.

Sprawdzanie, czy skoroszyt jest otwarty

W przykładzie zaprezentowanym w tym punkcie pokazano sposób monitorowania otwieranych skoroszytów za pomocą informacji zapisywanych do pliku tekstowego w formacie CSV (ang. *Comma Separated Values*), który następnie może być zainportowany do Excela.

Rozpoczniemy od wstawienia nowego modułu klasy i nazwania go c1sApp. Kod modułu klasy zamieszczono poniżej:

```
Public WithEvents AppEvents As Application

Private Sub AppEvents_WorkbookOpen (ByVal Wb As Excel.Workbook)
    Call UpdateLogFile(Wb)
End Sub
```

W kodzie zadeklarowano zmienną AppEvents jako obiekt Application z obsługą zdarzeń. Procedura AppEvents_WorkbookOpen będzie wywoływana przy każdej próbie otwarcia skoroszytu. Procedura obsługi zdarzenia wywoła procedurę UpdateLogFile i przekaże zmienną Wb reprezentującą otwarty skoroszyt. Następnie dodamy moduł VBA i wprowadzimy następujący kod:

```
Dim AppObject As New c1sApp

Sub Init()
    ' Procedura wywoływaną przez procedurę Workbook_Open
    Set AppObject.AppEvents = Application
End Sub
```

```
Sub UpdateLogFile(Wb)
    Dim txt As String
    Dim Fname As String
    txt = Wb.FullName
    txt = txt & "," & Date & "," & Time
    txt = txt & " " & Application.UserName
    Fname = Application.DefaultFilePath & "\logfile.csv"
    Open Fname For Append As #1
    Print #1, txt
    Close #1
    MsgBox txt
End Sub
```

Zwróć uwagę, że zmienna AppObject została zadeklarowana jako typ `clsApp` (nazwa typu odpowiada nazwie modułu klasy). W procedurze `Workbook_Open` znajdującej się w module kodu obiektu `ThisWorkbook` umieszczono wywołanie procedury `Init`. Procedure `Workbook_Open` zaprezentowano poniżej:

```
Private Sub Workbook_Open()
    Call Init
End Sub
```

Procedura `UpdateLogFile` otwiera plik tekstowy lub tworzy go w przypadku, kiedy nie istnieje. Następnie zapisuje najważniejsze informacje o otwartym skoroszycie: nazwę pliku, pełną ścieżkę dostępu, datę, godzinę i nazwę użytkownika.

Procedura `Workbook_Open` wywołuje procedurę `Init`, która podczas otwierania skoroszytu tworzy zmienną obiektową. Ostatnie polecenie używa okna komunikatu do wyświetlenia informacji, które zostały zapisane do pliku CSV. Jeżeli nie chcesz, aby ten komunikat pojawiał się na ekranie, możesz po prostu usunąć to polecenie z kodu procedury.



Skoroszyt z tym przykładem (*Monitorowanie otwartych skoroszytów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Monitorowanie zdarzeń poziomu aplikacji

W zrozumieniu procesu generowania zdarzeń pomocne jest wyświetlanie listy zdarzeń zachodzących podczas wykonywania różnych działań.

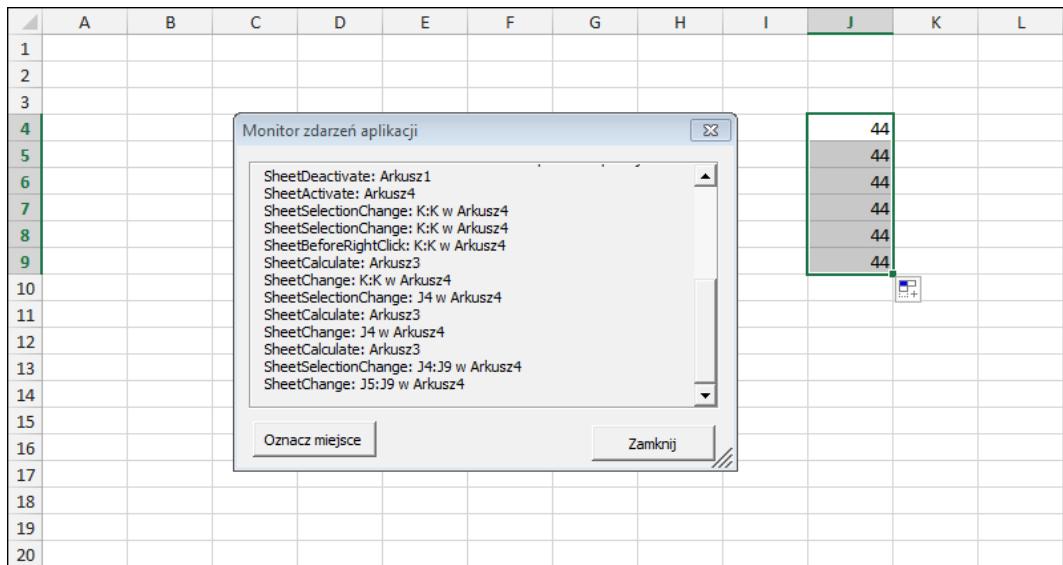
Aby zilustrować takie zdarzenia, utworzyłem aplikację, która na formularzu `UserForm` wyświetla wszystkie zdarzenia poziomu `Application` w miarę ich pojawiania się (patrz rysunek 17.10). Taka aplikacja może bardzo ułatwić zrozumienie typów i sekwencji zachodzących zdarzeń.



Skoroszyt z tym przykładem (*Zdarzenia poziomu aplikacji.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Skoroszyt zawiera moduł klasy, w którym zdefiniowano 21 procedur, po jednej dla każdego zdarzenia. Oto przykład jednej z nich:

```
Private Sub XL_NewWorkbook(ByVal Wb As Excel.Workbook)
    LogEvent "NewWorkbook: " & Wb.Name
End Sub
```



Rysunek 17.10. W tym skoroszycie wykorzystano moduł klasy w celu monitorowania zdarzeń poziomu aplikacji

Każda z tych procedur wywołuje procedurę LogEvent i przekazuje do niej argument składający się z nazwy zdarzenia i obiektu. Procedurę LogEvent zaprezentowano poniżej:

```
Sub LogEvent(txt)
    EventNum = EventNum + 1
    With UserForm1
        With .lblEvents
            .AutoSize = False
            .Caption = .Caption & vbCrLf & txt
            .Width = UserForm1.FrameEvents.Width - 20
            .AutoSize = True
        End With
        .FrameEvents.ScrollHeight = .lblEvents.Height + 20
        .FrameEvents.ScrollTop = EventNum * 20
    End With
End Sub
```

Procedura LogEvent aktualizuje formularz *UserForm*, modyfikując właściwość Caption formantu Label o nazwie *lblEvents*. Procedura koryguje także właściwości ScrollHeight oraz ScrollTop obiektu Frame o nazwie *FrameEvents* zawierającego obiekt *Label*. Dzięki modyfikacji tych właściwości ostatnio dodany tekst jest widoczny, natomiast starszy znika z ekranu. W razie potrzeby możesz również zmieniać rozmiary okna formularza. Do realizacji skalowania została użyta technika opisana wcześniej w rozdziale 13.

Zdarzenia dotyczące formularzy *UserForm*

Istnieje kilka zdarzeń związanych z formularzami *UserForm*. Dodatkowo z każdym formantem umieszczanym na formularzach *UserForm* jest związany osobny zbiór zdarzeń. Zdarzenia dotyczące formularzy *UserForm* zestawiono w tabeli 17.5.

Tabela 17.5. Zdarzenia obsługiwane przez formularz *UserForm*

Zdarzenie	Działania, które powodują wygenerowanie zdarzenia
Activate	Uaktywnienie formularza <i>UserForm</i>
AddControl	Dodanie formantu w fazie wykonywania programu
BeforeDragOver	Wykonywanie operacji przeciągania i upuszczania w czasie, kiedy wskaźnik myszy znajduje się nad formularzem
BeforeDropOrPaste	Próba upuszczenia lub wklejenia danych (tzn. użytkownik zwolnił klawisz myszy)
Click	Kliknięcie myszą w czasie, kiedy wskaźnik myszy znajduje się nad formularzem
DoubleClick	Dwukrotne kliknięcie myszą w czasie, kiedy wskaźnik myszy znajduje się nad formularzem
Deactivate	Desaktywacja formularza <i>UserForm</i>
Error	Błąd spowodowany przez formant — brak możliwości zwrócenia informacji o błędzie do programu wywołującego
Initialize	Żądanie wyświetlenia formularza <i>UserForm</i>
KeyDown	Wciśnięcie klawisza
KeyPress	Wciśnięcie klawisza (kod ANSI)
KeyUp	Zwolnienie klawisza
Layout	Zmiana rozmiaru formularza <i>UserForm</i>
MouseDown	Wciśnięcie klawisza myszy
MouseMove	Przemieszczanie wskaźnika myszy
MouseUp	Zwolnienie klawisza myszy
QueryClose	Żądanie zamknięcia formularza <i>UserForm</i>
RemoveControl	Usunięcie formantu z formularza <i>UserForm</i> w fazie wykonywania programu
Resize	Zmiana rozmiaru formularza <i>UserForm</i>
Scroll	Przewijanie formularza <i>UserForm</i>
Terminate	Usunięcie formularza <i>UserForm</i>
Zoom	Zmiana wielkości formularza <i>UserForm</i>



W wielu przykładach z rozdziałów od 11. do 13. zademonstrowano obsługę zdarzeń dla formularza *UserForm* oraz użytych w nim formantów.

Zdarzenia niezwiązane z obiektami

Wszystkie zdarzenia omówione wcześniej w tym rozdziale były powiązane z obiektami (Application, Workbook, Sheet itd.). W tym podrozdziale omówię dwa dodatkowe zdarzenia: *OnTime* oraz *OnKey*, które nie są związanymi z żadnymi obiektami. Dostęp do nich uzyskuje się za pomocą metod obiektu *Application*.



W odróżnieniu od innych zdarzeń omówionych w tym rozdziale wymienione zdarzenia On definiuje się w ogólnym module VBA (nie w module klasy).

Zdarzenie OnTime

Zdarzenie OnTime występuje o określonej porze dnia. Kod poniższego przykładu powoduje wygenerowanie sygnału dźwiękowego, a następnie wyświetlenie komunikatu o godzinie 15:00:

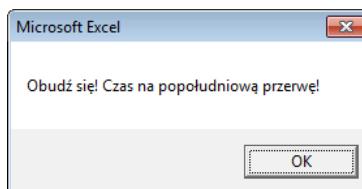
```
Sub SetAlarm()
    Application.OnTime TimeValue("15:00:00"), "DisplayAlarm"
End Sub

Sub DisplayAlarm()
    Beep
    MsgBox "Obudź się! Czas na popołudniową przerwę!"
End Sub
```

W procedurze SetAlarm wykorzystano metodę OnTime obiektu Application w celu skonfigurowania zdarzenia OnTime. Metoda ta pobiera dwa argumenty: godzinę (w tym przykładzie 15:00:00) oraz procedurę, która ma być wykonana o tej godzinie (w tym przypadku DisplayAlarm). Po wykonaniu procedury SetAlarm o godzinie 15:00 zostanie wywołana procedura DisplayAlarm i wyświetli komunikat zaprezentowany na rysunku 17.11.

Rysunek 17.11.

To okno dialogowe zostało zaprogramowane tak, aby wyświetlało się o określonej porze dnia



Aby zaplanować zdarzenie w odniesieniu do czasu bieżącego, a więc takie, które ma nastąpić na przykład za 20 minut, należy zastosować poniższą instrukcję:

```
Application.OnTime Now + TimeValue("00:20:00"), "DisplayAlarm"
```

Metodę OnTime można zastosować także do zaplanowania wykonania procedury na określony dzień. Poniższa instrukcja spowoduje uruchomienie procedury DisplayAlarm o godz. 0:01 1 kwietnia 2014 roku:

```
Application.OnTime DateSerial(2014, 4, 1) + _
    TimeValue("00:00:01"), "DisplayAlarm"
```



Metoda OnTime ma dwa dodatkowe argumenty. Jeżeli chcesz korzystać z tej metody, powinieneś zajrzeć do pomocy systemowej programu Excel.



Skoroszyt z tym przykładem (Zdarzenie OnTime.xlsx) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Zaprezentowane poniżej dwie procedury demonstrują, w jaki sposób zaprogramować powtarzające się zdarzenie. W komórce A1 co pięć sekund wpisywana jest bieżąca godzina. Wykonanie procedury UpdateClock powoduje zapisanie godziny w komórce A1 i jednocześnie zaprogramowanie następnego zdarzenia pięć sekund później. Zdarzenie to ponownie uruchamia procedurę UpdateClock. Aby zatrzymać zdarzenia, należy wykonać procedurę StopClock (która powoduje anulowanie obsługi zdarzenia). Warto zwrócić uwagę, że NextTick jest zmienną poziomu modułu, w której jest zapisana godzina następnego zdarzenia.

```
Dim NextTick As Date

Sub UpdateClock()
    ' Uaktualnienie komórki A1 informacjami o bieżącej godzinie
    ThisWorkbook.Sheets(1).Range("A1") = Time
    ' Zaplanowanie następnego zdarzenia za pięć sekund
    NextTick = Now + TimeValue("00:00:05")
    Application.OnTime NextTick, "UpdateClock"
End Sub

Sub StopClock()
    ' Anulowanie zdarzenia OnTime (zatrzymanie zegara)
    On Error Resume Next
    Application.OnTime NextTick, "UpdateClock", , False
End Sub
```



Zdarzenie OnTime jest obsługiwane nawet po zamknięciu skoroszytu. Mówiąc inaczej, w przypadku zamknięcia skoroszytu bez uruchomienia procedury StopClock skoroszyt sam się otworzy za pięć sekund (przy założeniu, że Excel będzie w dalszym ciągu działać). Aby zabezpieczyć się przed takim działaniem, należy wykorzystać procedurę obsługi zdarzenia Workbook_BeforeClose zawierającą następującą instrukcję:

```
Call StopClock
```



Analogowy zegar z rozdziału 16. to przykład działania powtarzającego się zdarzenia OnTime.

Zdarzenie OnKey

Podczas pracy Excel przez cały czas monitoruje wciskane klawisze. Dzięki temu można skonfigurować klawisz lub kombinację klawiszy, których wcisnięcie spowoduje wykonanie określonej procedury. Jedyne przypadki, kiedy kombinacje klawiszy nie są rozpoznawane, to wprowadzanie formuły lub praca z oknem dialogowym.



Pamiętaj, że kiedy utworzysz procedurę obsługi zdarzenia OnKey, jej zasięg nie będzie ograniczony tylko do jednego skoroszytu. Zmienione odpowiedzi na naciśnięcie klawiszy będą dotyczyć wszystkich otwartych skoroszytów, a nie tylko tego, w którym zlokalizowana jest procedura obsługi tego zdarzenia.

Dodatkowo, jeżeli tworzysz procedurę obsługi zdarzenia OnKey, upewnij się, że zostawiłeś sobie możliwość anulowania tego zdarzenia. Najczęściej stosowanym rozwiązaniem jest użycie do tego celu procedury obsługi zdarzenia Workbook_BeforeClose.

Przykład zastosowania zdarzenia OnKey

W zaprezentowanym poniżej przykładzie wykorzystano metodę OnKey w celu skonfigurowania zdarzenia powstającego w momencie wcisnięcia klawiszy. Zdarzenie powoduje przypisanie procedur do klawiszy *PageDown* oraz *PageUp*. Po wykonaniu procedury *Setup_OnKey* wcisnięcie klawisza *PageDown* spowoduje wykonanie procedury *PgDn_Sub* i przesunięcie kurSORA w dół o jeden wiersz, natomiast wcisnięcie klawisza *PageUp* spowoduje wykonanie procedury *PgUp_Sub* i przesunięcie kurSORA o jeden wiersz w góRę. Inne kombinacje używające klawiszy *PageUp* oraz *PageDown* pozostają bez zmian, stąd na przykład, naciśnięcie kombinacji klawiszy *Ctrl+PageDown* nadal będzie aktywowało kolejny arkusz w skoroszycie.

```
Sub Setup_OnKey()
    Application.OnKey "{PgDn}", "PgDn_Sub"
    Application.OnKey "{PgUp}", "PgUp_Sub"
End Sub

Sub PgDn_Sub()
    On Error Resume Next
    ActiveCell.Offset(1, 0).Activate
End Sub

Sub PgUp_Sub()
    On Error Resume Next
    ActiveCell.Offset(-1, 0).Activate
End Sub
```

Skoroszyt z tym przykładem (*Zdarzenie OnKey.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).



W poprzednim przykładzie zastosowano instrukcję On Error Resume Next w celu zignorowania wygenerowanych błędów. Jeżeli na przykład aktywna komórka znajduje się w pierwszym wierszu, próba przesunięcia kurSORA o jeden wiersz w góRę spowoduje błąd. Błąd powstanie również w przypadku, gdy aktywny arkusz jest wykresem, ponieważ nie ma czegoś takiego, jak aktywna komórka wykresu.

Wykonanie poniższej procedury spowoduje anulowanie obsługi zdarzeń OnKey i przywrócenie normalnego działania klawiszy:

```
Sub Cancel_OnKey()
    Application.OnKey "{PgDn}"
    Application.OnKey "{PgUp}"
End Sub
```

W przeciwnieństwie do tego, czego mógłbyś oczekiwać, zastosowanie pustego ciągu znaków jako drugiego argumentu metody OnKey *nie* spowoduje anulowania zdarzenia OnKey. Zamiast tego Excel zignoruje ciąg znaków i nie wykona żadnych działań. Na przykład poniższa instrukcja informuje Excela, aby ignorował kombinację klawiszy *Alt+F4* (znak procentu reprezentuje klawisz *Alt*):

```
Application.OnKey "%{F4}", ""
```



Chociaż metodę `OnKey` można wykorzystać, aby przypisać klawisz do makra, w celu wykonania takiego zadania lepiej zastosować okno dialogowe *Opcje makra*. Więcej informacji na ten temat można znaleźć w rozdziale 7.

Kody klawiszy

Zwróć uwagę, że w poprzednim przykładzie odwołanie do klawisza `PgDn` zostało umieszczone w nawiasach klamrowych. W tabeli 17.6 przedstawiono zestawienie kodów klawiszy, których możesz używać w procedurach obsługi zdarzenia `OnKey`.

Tabela 17.6. Zestawienie kodów klawiszy dla zdarzenia `OnKey`

Klawisz	Kod
<i>Backspace</i>	{BACKSPACE} lub {BS}
<i>Break</i>	{BREAK}
<i>Caps Lock</i>	{CAPSLOCK}
<i>Delete lub Del</i>	{DELETE} lub {DEL}
<i>Strzałka w dół</i>	{DOWN}
<i>End</i>	{END}
<i>Enter</i>	~ (tylda)
<i>Enter</i> (na klawiaturze numerycznej)	{ENTER}
<i>Escape</i>	{ESCAPE} lub {ESC}
<i>Home</i>	{HOME}
<i>Ins</i>	{INSERT}
<i>Strzałka w lewo</i>	{LEFT}
<i>NumLock</i>	{NUMLOCK}
<i>Page Down</i>	{PGDN}
<i>Page Up</i>	{PGUP}
<i>Strzałka w prawo</i>	{RIGHT}
<i>Scroll Lock</i>	{SCROLLLOCK}
<i>Tab</i>	{TAB}
<i>Strzałka w góre</i>	{UP}
<i>F1 do F15</i>	{F1} do {F15}

Możesz również używać wymienionych w tabeli klawiszy w kombinacji z klawiszami *Shift*, *Ctrl* lub *Alt*. Aby zdefiniować taką kombinację, powinieneś użyć następujących symboli:

- **Shift:** znak plus (+).
- **Ctrl:** znak (^).
- **Alt:** znak procentu (%).

Aby na przykład przypisać procedurę do kombinacji klawiszy *Ctrl+Shift+A*, powinieneś użyć następującego polecenia:

```
Application.OnKey "^+A", "NazwaProcedury"
```

Aby przypisać procedurę do kombinacji klawiszy *Alt+F11* (która normalnie jest używana do uruchomienia edytora VBE), powinieneś użyć następującego polecenia:

```
Application.OnKey "^{F11}", "NazwaProcedury"
```

Wyłączanie menu podręcznego

We wcześniejszej części tego rozdziału omawialiśmy procedurę `Worksheet_BeforeRightClick`, która wyłączała możliwość wyświetlania menu podręcznego. Kod procedury przedstawionej poniżej powinien zostać umieszczony w module kodu obiektu `ThisWorkbook`:

```
Private Sub Worksheet_BeforeRightClick  
    (ByVal Target As Range, Cancel As Boolean)  
    Cancel = True  
    MsgBox "Menu podręczne zostało wyłączone."  
End Sub
```

Zauważymy także, że pomimo to użytkownik nadal miał możliwość wyświetlenia menu podręcznego poprzez naciśnięcie kombinacji klawiszy *Shift+F10*. Aby przechwycić naciśnięcie tej kombinacji klawiszy, dodaj do standardowego modułu kodu VBA następujące procedury:

```
Sub SetupNoShiftF10()  
    Application.OnKey "+{F10}", "NoShiftF10"  
End Sub  
  
Sub TurnOffNoShiftF10()  
    Application.OnKey "+{F10}"  
End Sub  
  
Sub NoShiftF10()  
    MsgBox "Przykro mi, ale ten sposób również nie działa."  
End Sub
```

Po wykonaniu procedury `SetupNoShiftF10` naciśnięcie kombinacji klawiszy *Shift+F10* powoduje wyświetlenie komunikatu przedstawionego na rysunku 17.12. Pamiętaj, że procedura `Worksheet_BeforeRightClick` działa tylko w zakresie swojego własnego skoroszytu, natomiast procedura obsługi naciśnięcia kombinacji klawiszy *Shift+F10* będzie miała zastosowanie dla wszystkich otwartych skoroszytów.



Niektóre klawiatury posiadają specjalny, dodatkowy klawisz pozwalający na otwarcie menu podręcznego. Na mojej klawiaturze klawisz ten jest umieszczony po prawej stronie, pomiędzy klawiszem *Windows* a klawiszem *Ctrl*. Ku mojemu zdziwieniu, zablokowanie naciśnięcia kombinacji klawiszy *Shift+F10* blokuje również taki dedykowany klawisz menu podręcznego.

Rysunek 17.12.

Naciśnięcie kombinacji klawiszy Shift+F10 powoduje wyświetlenie takiego komunikatu

	A	B	C	D	E	F	G	H	I	J
1	52	10	29	16	50	1	27	43		
2	18	93	79	68	74	66	99	88		
3	27	53	31	77	6	53	59	7		
4	64	85	74	46	29	41	66	98		
5	16	18	32	78	72	15	33	1		
6	95	63							45	
7	45	13							51	
8	49	7							51	
9	81	89							54	
10	12	95							3	
11	85	50							80	
12	22	7							2	
13	38	31	69	88	22	18	28		2	
14	34	12	98	22	67	6	72	91		
15	25	98	85	42	54	42	39	98		
16	29	65	88	6	88	35	72	32		
17										
18										



Skoroszyt zawierający wszystkie opisane wyżej przykłady (*Zablokuj menu podręczne.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pwv.htm>). W skoroszycie znajdziesz takie procedury obsługi zdarzeń jak: Workbook_Open, która wywołuje procedurę SetupNoShiftF10, oraz procedurę Workbook_BeforeClose, która wywołuje procedurę TurnOffNoShiftF10.

Rozdział 18.

Interakcje

z innymi aplikacjami

W tym rozdziale:

- Uruchamianie innych aplikacji z poziomu Excela
- Wyświetlanie okien dialogowych Panelu sterowania
- Wykorzystanie automatyzacji do sterowania innymi aplikacjami

Uruchamianie innych aplikacji z poziomu Excela

Uruchamianie innych aplikacji z poziomu Excela jest bardzo często przydatną operacją. Dzięki niej można na przykład z poziomu Excela uruchomić inną aplikację Microsoft Office lub nawet DOS-owy skrypt wsadowy. Można także ułatwić użytkownikowi dostęp do Panelu sterowania i zmianę ustawień systemowych z poziomu aplikacji programu Excel.

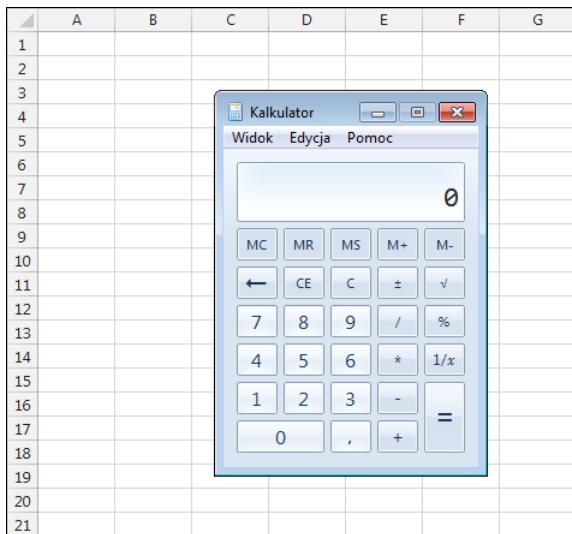
Zastosowanie funkcji Shell języka VBA

Funkcja `Shell` języka VBA powoduje, że uruchamianie innych programów jest stosunkowo proste. Poniżej przedstawiono kod procedury `StartCalc`, która uruchamia program `calc.exe`, czyli popularny Kalkulator systemu Windows.

```
Sub StartCalc()
    Dim Program As String
    Dim TaskID As Double
    On Error Resume Next
    Program = "calc.exe"
    TaskID = Shell(Program, 1)
    If Err <> 0 Then
        MsgBox "Nie mogę uruchomić programu " & Program, vbCritical, "Błąd!"
    End If
End Sub
```

Na rysunku 18.1 zaprezentowano okno aplikacji, która została uruchomiona za pomocą powyższej procedury.

Rysunek 18.1.
*Uruchamianie aplikacji
 Kalkulator z poziomu
 Excela*



Funkcja `Shell` zwraca identyfikator zadania dla aplikacji. Identyfikator ten można następnie wykorzystać do uaktywnienia zadania. Drugi argument funkcji `Shell` określa sposób wyświetlania aplikacji (1 oznacza aktywne okno o domyślnym rozmiarze). Informacje o innych wartościach tego argumentu można uzyskać w systemie pomocy.

Jeżeli wykonanie funkcji `Shell` nie powiedzie się, następuje wygenerowanie błędu. Z tego powodu w powyższej funkcji wykorzystano polecenie `On Error`, które wyświetla komunikat, jeżeli nie może znaleźć pliku wykonywalnego lub jeżeli wystąpi inny błąd.

Pamiętaj, że kod VBA nie zatrzymuje działania po uruchomieniu aplikacji za pomocą funkcji `Shell`. Mówiąc inaczej, funkcja `Shell` uruchamia aplikację w sposób *asynchroniczny*. Jeżeli za wywołaniem funkcji `Shell` w procedurze znajdują się dodatkowe instrukcje, będą one wykonywane równolegle z uruchomionym programem. Jeżeli dowolna instrukcja wymaga interwencji użytkownika (np. wyświetla się okno informacyjne), a aktywna jest inna aplikacja, zaczyna migać pasek tytułu Excela.

Czasami trzeba uruchomić aplikację za pomocą funkcji `Shell` i zatrzymać kod VBA do czasu jej zamknięcia. Może tak się zdarzyć, jeżeli na przykład uruchomiona aplikacja generuje plik, który jest używany w dalszej części kodu. Chociaż nie można wstrzymać wykonywania kodu, możesz utworzyć pętlę, która monitoruje stan aplikacji. Poniżej prezentuję przykład kodu wyświetlającego okno informacyjne w chwili zakończenia działania aplikacji uruchomionej za pomocą funkcji `Shell`.

```

Declare PtrSafe Function OpenProcess Lib "kernel32" _
    (ByVal dwDesiredAccess As Long, _
    ByVal bInheritHandle As Long, _
    ByVal dwProcessId As Long) As Long

Declare PtrSafe Function GetExitCodeProcess Lib "kernel32" _
    (ByVal hProcess As Long,
    TpExitCode As Long) As Long

Sub StartCalc2()
    Dim TaskID As Long

```

```

Dim hProc As Long
Dim lExitCode As Long
Dim ACCESS_TYPE As Integer, STILL_ACTIVE As Integer
Dim Program As String

ACCESS_TYPE = &H400
STILL_ACTIVE = &H103

Program = "Calc.exe"
On Error Resume Next

' Uruchomienie programu przy użyciu funkcji Shell
TaskID = Shell(Program, 1)

' Pobieranie uchwytu aplikacji
hProc = OpenProcess(ACCESS_TYPE, False, TaskID)

If Err <> 0 Then
    MsgBox "Nie mogę uruchomić programu " & Program, vbCritical, "Błąd!"
    Exit Sub
End If

Do 'Pętla
    ' Sprawdź stan procesu
    GetExitCodeProcess hProc, lExitCode
    ' Zewalaj na obsługę zdarzeń
    DoEvents
    Loop While lExitCode = STILL_ACTIVE

    ' Zadanie zakończone, wyświetlanie komunikatu
    MsgBox Program & " zakończył działanie"
End Sub

```

W czasie, kiedy pracuje uruchomiony wcześniej program, procedura wykonuje w pętli Do ... Loop funkcję GetExitCodeProcess, sprawdzając zwracaną wartość (lExitCode). Kiedy program zakończy działanie, zmienna lExitCode przyjmie inną wartość, pętla zakończy działanie i wykonywanie kodu VBA zostanie wznowione.



Skoroszyt z tym przykładem (*Uruchamianie kalkulatora.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Innym sposobem na uruchomienie takiej aplikacji jest utworzenie hiperłącza w komórce arkusza (VBA nie jest do tego potrzebny). Na przykład formula przedstawiona poniżej tworzy w komórce hiperłącze, które po kliknięciu lewym przyciskiem myszy powoduje uruchomienie kalkulatora systemowego:

```
=HYPERLINK("C:\Windows\System32\calc.exe", "Windows Calculator")
```

Tworząc hiperłącze, powinieneś się zawsze upewnić, że prowadzi ono do właściwej lokalizacji. Pamiętaj również, że po kliknięciu takiego łącza na ekranie zazwyczaj pojawi się co najmniej jedno ostrzeżenie. Taka technika działa nie tylko dla programów, ale również dla innych rodzajów plików i powoduje uruchomienie aplikacji powiązanej z danym typem pliku. Na przykład kliknięcie hiperłącza utworzonego przez формуłę przedstawioną poniżej uruchomi domyślny edytor plików tekstowych i załaduje do niego plik wskazywany przez łącze:

```
=HYPERLINK("C:\files\data.txt", "Open the data file")
```

Wyświetlanie okna folderu

Funkcja Shell jest również bardzo użyteczna w sytuacji, kiedy musisz wyświetlić zawartość określonego katalogu przy użyciu programu Windows Explorer. Na przykład polecenie przedstawione poniżej wyświetla folder, w którym jest przechowywany aktywny skoroszyt (zakładając oczywiście, że skoroszyt został już wcześniej zapisany na dysku):

```
If ActiveWorkbook.Path <> "" Then  
    Shell "explorer.exe" & ActiveWorkbook.Path, vbNormalFocus
```

Zastosowanie funkcji ShellExecute interfejsu Windows API

Do uruchamiania innych aplikacji można wykorzystać funkcję ShellExecute należącą do interfejsu Windows API (ang. *Windows Application Programming Interface*). Ważną cechą tej funkcji jest możliwość uruchomienia aplikacji na podstawie nazwy pliku powiązanego z tą aplikacją (o ile określony typ pliku został skojarzony z aplikacją w rejestrze systemu Windows). Na przykład funkcję ShellExecute można wykorzystać do otwarcia dokumentu WWW w domyślnej przeglądarce WWW. Można też uruchomić domyślnego klienta poczty, aby wysłać wiadomość pod wskazany adres e-mail.

Deklarację API funkcji ShellExecute zmieszczono poniżej (ten kod będzie działał tylko z Exceliem 2010 lub nowszym):

```
Private Declare PtrSafe Function ShellExecute Lib "shell32.dll" _  
    Alias "ShellExecuteA" (ByVal hWnd As Long, _  
    ByVal lpOperation As String, ByVal lpFile As String, _  
    ByVal lpParameters As String, ByVal lpDirectory As String, _  
    ByVal nShowCmd As Long) As Long
```

W poniższym przykładzie zademonstrowano sposób wywołania funkcji ShellExecute. Wykorzystano ją do otwarcia pliku graficznego za pomocą programu graficznego, z którym skojarzono pliki *JPG*. Jeżeli wartość zwracana przez tę funkcję jest mniejsza niż 32, oznacza to, że wystąpił błąd.

```
Sub ShowGraphic()  
    Dim FileName As String  
    Dim Result As Long  
    FileName = ThisWorkbook.Path & "\flower.jpg"  
    Result = ShellExecute(0&, vbNullString, FileName, _  
        vbNullString, vbNullString, vbNormalFocus)  
    If Result < 32 Then MsgBox "Wystąpił błąd!"  
End Sub
```

Kolejna procedura otwiera plik tekstowy, korzystając z domyślnego edytora tekstu:

```
Sub OpenTextFile()  
    Dim FileName As String  
    Dim Result As Long  
    FileName = ThisWorkbook.Path & "\textfile.txt"  
    Result = ShellExecute(0&, vbNullString, FileName, _  
        vbNullString, vbNullString, vbNormalFocus)  
    If Result < 32 Then MsgBox "Wystąpił błąd!"  
End Sub
```

Następnym przykładem działa podobnie — tym razem otwiera w domyślnej przeglądarce sieciowej stronę WWW o podanym adresie URL:

```
Sub OpenURL()
    Dim URL As String
    Dim Result As Long
    URL = "http://spreadsheetpage.com"
    Result = ShellExecute(0&, vbNullString, URL, _
        vbNullString, vbNullString, vbNormalFocus)
    If Result < 32 Then MsgBox "Wystąpił błąd!"
End Sub
```

Technikę tę można także wykorzystać do pracy z adresami e-mail. Poniższy kod powoduje otwarcie domyślnego klienta poczty elektronicznej (o ile taki został zdefiniowany) i zaadresowanie wiadomości e-mail do określonego adresata:

```
Sub StartEmail()
    Dim Addr As String
    Dim Result As Long
    Addr = "mailto:nobody@example.com"
    Result = ShellExecute(0&, vbNullString, Addr, _
        vbNullString, vbNullString, vbNormalFocus)
    If Result < 32 Then MsgBox "Wystąpił błąd!"
End Sub
```



Skoroszyt z tymi przykładami (*Funkcja ShellExecute.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Skoroszyt zamieszczony w sieci korzysta z deklaracji API, które są kompatybilne ze wszystkimi wersjami Excela.

Uaktywnianie aplikacji z poziomu Excela

W poprzednim podrozdziale omówiono różne sposoby uruchamiania aplikacji. W praktyce może się okazać, że użycie funkcji `ShellExecute` w przypadku, gdy taką aplikację uruchomiono już wcześniej, może spowodować uruchomienie jej kolejnej kopii. W większości przypadków nie interesuje nas jednak uruchomienie nowej kopii aplikacji, ale *uaktywnienie* aplikacji, która już została wcześniej uruchomiona.

Wykorzystanie instrukcji AppActivate

W zaprezentowanej poniżej procedurze `StartCalculator` wykorzystano instrukcję `AppActivate` w celu uaktywnienia działającej aplikacji (w tym przypadku jest to stary, dobry *Kalkulator* systemu Windows). Argumentem instrukcji `AppActivate` jest tytuł aplikacji (tekst na pasku tytułu). Jeżeli instrukcja `AppActivate` wygeneruje błąd, będzie to oznaczało, że Kalkulator nie został wcześniej uruchomiony. W takim przypadku procedura go uruchomi.

```
Sub StartCalculator()
    Dim AppFile As String
    Dim CalcTaskID As Double
```

```
AppFile = "Calc.exe"
On Error Resume Next
AppActivate "Kalkulator"
If Err <> 0 Then
    Err = 0
    CalcTaskID = Shell(AppFile, 1)
    If Err <> 0 Then MsgBox "Nie można uruchomić kalkulatora."
End If
End Sub
```



Skoroszyt z tym przykładem (*Uruchamianie kalkulatora.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Uaktywnianie aplikacji pakietu Microsoft Office

Jeżeli aplikacja należy do pakietu Office, do jej uaktywnienia można użyć metody `ActivateMicrosoftApp` obiektu `Application`. Na przykład wykonanie poniższej procedury spowoduje uruchomienie Worda:

```
Sub StartWord()
    Application.ActivateMicrosoftApp xlMicrosoftWord
End Sub
```

Jeżeli w momencie wykonywania procedury Word był już uruchomiony, nastąpi jego uaktywnienie. W instrukcji wywołania tej metody można także wykorzystać inne stałe:

- `xlMicrosoftPowerPoint`
- `xlMicrosoftMail` (aktywuje program Outlook)
- `xlMicrosoftAccess`
- `xlMicrosoftFoxPro`
- `xlMicrosoftProject`

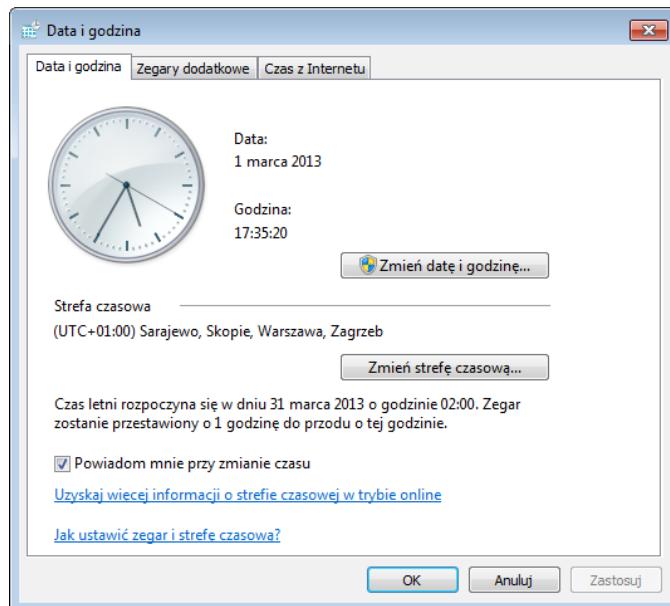
Uruchamianie okien dialogowych Panelu sterowania

W systemie Windows dostępnych jest kilka różnych okien dialogowych i kreatorów, do których w większości uzyskuje się dostęp z Panelu sterowania. Czasami trzeba wyświetlić jedno lub kilka tych narzędzi z poziomu aplikacji Excela. Może to być na przykład okno dialogowe właściwości daty i godziny, jak pokazano na rysunku 18.2.

Kluczem do wywoływanego okien dialogowych innego systemu jest wykonanie aplikacji `rundll32.exe` za pomocą funkcji `Shell` języka VBA.

Rysunek 18.2.

Za pomocą kodu VBA można wyświetlić okna dialogowe Panelu sterowania



Wykonanie poniższej procedury spowoduje wyświetlenie okna dialogowego *Właściwości: Data i godzina*:

```
Sub ShowDateTimeDlg()
    Dim Arg As String
    Dim TaskID As Double
    Arg = "rundll32.exe shell32.dll, Control_RunDLL timedate.cpl"
    On Error Resume Next
    TaskID = Shell(Arg)
    If Err <> 0 Then
        MsgBox ("Nie można uruchomić aplikacji .")
    End If
End Sub
```

Poniżej podano ogólną składnię wywołania aplikacji *rundll32.exe*:

rundll32.exe shell32.dll, Control_RunDLL nazwa_pliku.cp1.@n,t

gdzie:

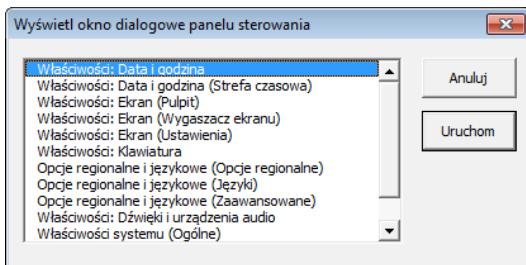
- *nazwa_pliku.cp1* — nazwa jednego z plików CPL z Panelu sterowania;
- *n* — numer apletu wewnątrz pliku CPL (numery zaczynają się od zera);
- *t* — numer zakładki (w przypadku apletów zawierających wiele zakładek).



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt *Okna Panelu sterowania.xls*, wyświetlający 12 apletów Panelu sterowania. Wygląd aplikacji zaprezentowano na rysunku 18.3.

Rysunek 18.3.

Skoroszyt z tym oknem dialogowym pokazuje sposób uruchamiania systemowych okien dialogowych z poziomu Excela



Wykorzystanie automatyzacji w programie Excel

W Excelu można napisać makro zarządzające pracą innej aplikacji, na przykład Worda. Mówiąc ściślej, za pomocą makra w Excelu można zarządzać serwerem automatyzacji Worda. W takiej sytuacji Excel jest *aplikacją klientką*, natomiast Word *serwerem*. Można też napisać aplikację sterującą Exceliem w Visual Basicu. Proces zarządzania aplikacją przez inną aplikację czasami nazywa się łączaniem i osadzaniem obiektów (ang. *Object Linking and Embedding — OLE*) lub po prostu *automatyzacją*.

Zalety automatyzacji są niezaprzeczalne. Na przykład programista, który chce wygenerować wykres, może po prostu wykorzystać inną aplikację, uzyskać dostęp do jej obiektów, pobrać obiekt Chart, a następnie wykorzystywać jego właściwości i metody. Automatyzacja w pewnym sensie zaciera granice pomiędzy aplikacjami. Użytkownik Excela może pracować z obiektem Accessa i zupełnie nie zdawać sobie z tego sprawy.



Niektóre aplikacje, takie jak na przykład Excel, mogą działać zarówno jako aplikacja-klient, jak i jako aplikacja-serwer. Inne mogą działać wyłącznie jako aplikacje typu klient lub wyłącznie jako aplikacje typu serwer.

W tym podrozdziale zademonstrowano sposób użycia języka VBA w celu korzystania z obiektów innych aplikacji. W przykładach posługuję się programem Microsoft Word, ale pojęcia te w równym stopniu dotyczą innych aplikacji udostępniających obiekty do automatyzacji — a takich aplikacji jest coraz więcej.

Działania z obiektami innych aplikacji z wykorzystaniem automatyzacji

Do osadzania w arkuszu Excela obiektu, na przykład dokumentu Worda, służy polecenie *Wstaw obiekt*, znajdujące się w grupie opcji *Tekst*, na karcie *WSTAWIANIE*. Dodatkowo można utworzyć obiekt i wykonywać z nim działania za pomocą kodu VBA (działania te stanowią sedno mechanizmu automatyzacji). W takim przypadku zazwyczaj mamy pełny dostęp do obiektów. Dla programistów taka technika zwykle jest korzystniejsza od osadzania obiektów w arkuszach. W przypadku osadzania obiektów użytkownik musi znać sposób posługiwanego się aplikacją obiektu automatyzacji. Natomiast jeżeli do wykonywania działań z obiektem wykorzystamy język VBA, możemy tak zaprogramować obiekt, aby działania z nim sprowadzały się do prostych czynności, na przykład kliknięcia przycisku.

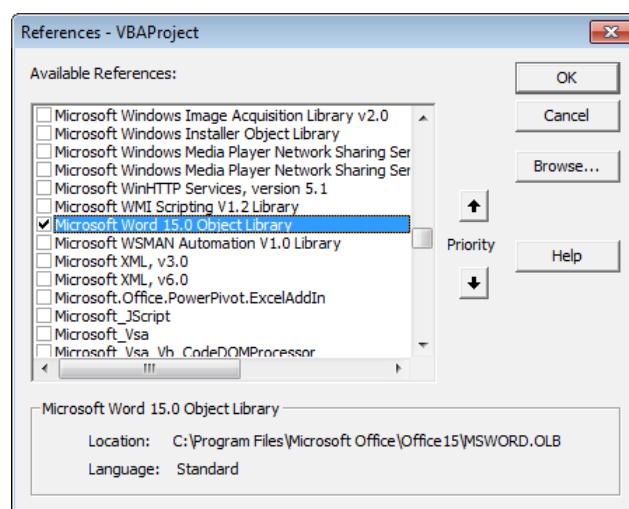
Wczesne i późne wiązanie

Przed wykonaniem działań z obiektem zewnętrznym należy utworzyć jego instance. Można to zrobić na dwa sposoby: wykorzystując wczesne wiązanie (ang. *early binding*) lub późne wiązanie (ang. *late binding*). Termin *wiązanie* dotyczy dopasowywania utworzonych przez programistę wywołań funkcji do właściwego kodu implementującego funkcje.

Wczesne wiązanie

Aby wykorzystać wczesne wiązanie, należy utworzyć odwołanie do biblioteki obiektów poprzez wybranie polecenia *Tools/References* w edytorze Visual Basic. Na ekranie wyświetli się okno dialogowe podobne do tego, które zaprezentowano na rysunku 18.4. Aby utworzyć odwołanie, odszukaj i zaznacz na liście żądaną bibliotekę.

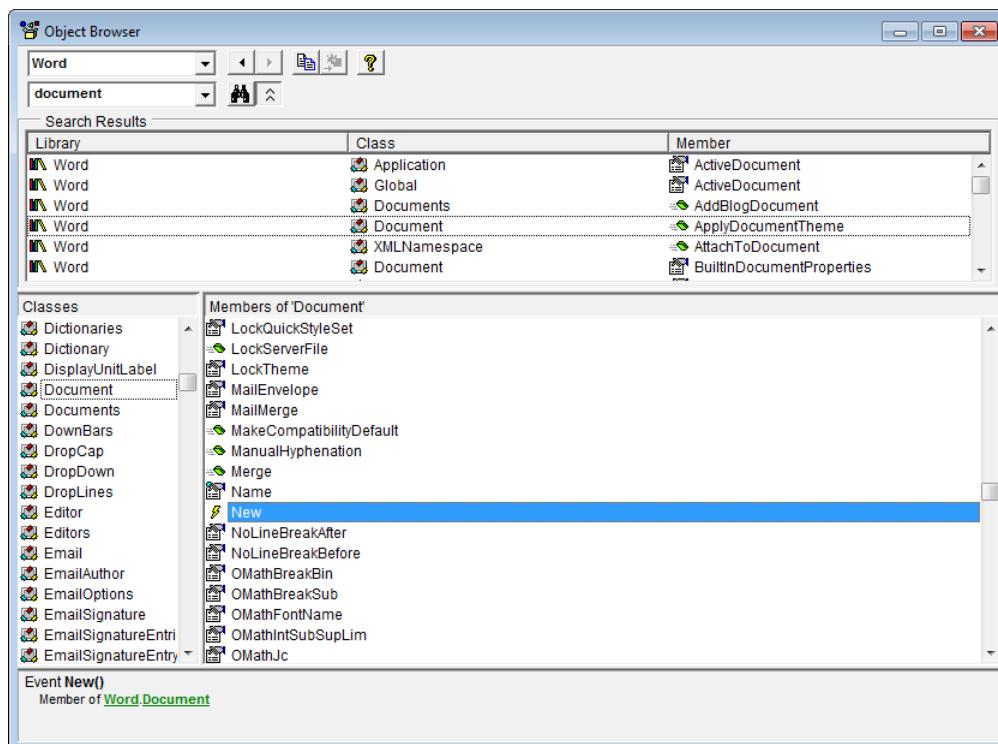
Rysunek 18.4.
Dodawanie odwołania
do pliku biblioteki
obiektów



Po zdefiniowaniu odwołania do biblioteki obiektów można wykorzystać przeglądarkę obiektów (jak pokazano na rysunku 18.5) do przeglądania nazw obiektów oraz ich metod i właściwości. Aby uruchomić przeglądarkę obiektów, należy wcisnąć F2 w edytorze Visual Basic.

W przypadku użycia wczesnego wiązania należy zdefiniować odwołanie do konkretnej wersji biblioteki obiektów. Na przykład możesz wprowadzić odwołanie do biblioteki *Microsoft Word 10.0 Object Library* (dla Worda 2002), *Microsoft Word 11.0 Object Library* (dla Worda 2003), *Microsoft Word 12.0 Object Library* (dla Worda 2007), *Microsoft Word 14.0 Object Library* (dla Worda 2010) lub *Microsoft Word 15.0 Object Library* (dla Worda 2013). Po zdefiniowaniu odwołania powinieneś utworzyć obiekt, korzystając z następującego polecenia:

```
Dim WordApp As New Word.Application
```



Rysunek 18.5. Za pomocą przeglądarki obiektów można uzyskać informacje o obiektach, do których zdefiniowano odwołania

Zastosowanie wczesnego wiązania do tworzenia obiektów, polegającego na skonfigurowaniu odwołania do biblioteki obiektów zazwyczaj jest wydajniejszym i szybszym rozwiązaniem niż użycie wiązania późnego. Wczesne wiązanie można jednak zastosować tylko wtedy, gdy obiekt jest zapisany w osobnym pliku biblioteki typu lub obiektu. Dodatkowo użytkownicy korzystający z aplikacji muszą zainstalować kopię określonej biblioteki.

Kolejną zaletą wczesnego wiązania jest możliwość wykorzystania stałych zdefiniowanych w bibliotece obiektu. Word, podobnie jak Excel, zawiera wiele predefiniowanych stałych, które można wykorzystać w kodzie VBA — ale tylko w przypadku wczesnego wiązania. W przypadku późnego wiązania można jedynie skorzystać z rzeczywistych wartości, a nie stałych.

Kolejną zaletą wczesnego wiązania jest umożliwienie wykorzystania przeglądarki obiektów edytora Visual Basic oraz opcji automatycznego wyświetlanego listy składowych ułatwiającej dostęp do właściwości i metod (ang. *Auto List Members*). Mechanizmy te są niedostępne w przypadku wykorzystania późnego wiązania, ponieważ typ obiektu jest znany dopiero w fazie wykonywania programu.

Późne wiązanie

W fazie wykonywania programu wykorzystuje się funkcję `CreateObject` w celu utworzenia obiektu lub funkcję `GetObject` w celu uzyskania zapisanego instancji obiektu. Taki obiekt jest deklarowany jako ogólny typ `Object`, a odwołanie do niego jest określone w fazie wykonywania programu.

Późne wiązanie można wykorzystać nawet wtedy, kiedy nie jest znana wersja aplikacji zainstalowanej w systemie użytkownika. Na przykład wykonanie poniższego kodu spowoduje utworzenie obiektu typu `Word` (kod działa dla Worda 97 i wersji późniejszych):

```
Dim WordApp As Object  
Set WordApp = CreateObject("Word.Application")
```

W przypadku zainstalowania wielu wersji Worda można utworzyć obiekt dla określonej wersji. Na przykład poniższa instrukcja utworzy obiekt Worda 2010:

```
Set WordApp = CreateObject("Word.Application.14")
```

Klucz rejestru Windows dla obiektu automatyzacji Worda, a także odwołanie do obiektu `Application` w języku VBA są takie same: `Word.Application`. Nie jest to jednak odwołanie do tego samego elementu. Deklaracja obiektu jako `As Word.Application` lub jako `As New Word.Application` dotyczy obiektu `Application` w bibliotece Word. Jednak wywołanie funkcji `CreateObject("Word.Application")` dotyczy nazwy, pod jaką występuje najwyższa wersja Worda w rejestrze Windows. Nie jest to uniwersalna zasada dla wszystkich obiektów automatyzacji, ale obowiązuje dla głównych składników pakietu Office 2013. Jeżeli użytkownik zastąpi Worda 2010 Wordem 2013, funkcja `CreateObject("Word.Application")` dalej będzie działać prawidłowo, choć tym razem będzie odwoływać się do nowej aplikacji. Jeżeli jednak usuniemy Worda 2013, wykonanie funkcji `CreateObject("Word.Application.15")`, w której użyto nazwy symbolicznej odpowiadającej Wordowi 2013, nie powiedzie się.

Funkcja `CreateObject` wykorzystana dla obiektu automatyzacji, na przykład `Word.Application` lub `Excel.Application`, zawsze tworzy *nową instancję* obiektu automatyzacji. Oznacza to, że zawsze zostanie uruchomiona nowa kopia tej części aplikacji, która bierze udział w automatyzacji. Nawet jeżeli dana instancja obiektu automatyzacji już działa, utworzony zostanie jego nowy egzemplarz, a następnie obiekt określonego typu.

Aby wykorzystać bieżący egzemplarz lub uruchomić aplikację i spowodować, aby został załadowany plik, należy skorzystać z funkcji `GetObject`.



W celu automatyzacji aplikacji pakietu Office zaleca się wykorzystanie wcześniego wiązania i odwołań do najwcześniej wersji produktu, która, jak się spodziewamy, jest zainstalowana w systemach klienckich. Jeżeli na przykład chcemy wykorzystać automatyzację dla Worda 2007, Worda 2010 oraz Worda 2013, w celu zachowania zgodności z wszystkimi trzema wersjami powinniśmy wykorzystać bibliotekę Worda 2007. Oczywiście oznacza to, że nie będzie można wykorzystać właściwości dostępnych w nowszych wersjach Worda.

Prosty przykład późnego wiązania

W poniższym przykładzie zademonstrowano sposób utworzenia obiektu Worda za pomocą późnego wiązania. Procedura tworzy obiekt, wyświetla numer wersji, zamienia aplikację Worda, a następnie niszczy obiekt (zwalniając tym samym używany przez niego obszar pamięci):

```
Sub GetWordVersion()
    Dim WordApp As Object
    Set WordApp = CreateObject("Word.Application")
    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```



Obiekt Worda utworzony w tej procedurze będzie niewidoczny. Aby zobaczyć obiekt w czasie, kiedy są z nim wykonywane działania, należy ustawić jego właściwość `Visible` na wartość `True`, tak jak pokazano poniżej:

```
WordApp.Visible = True
```

Przykład ten można też zaprogramować przy użyciu wczesnego wiązania. Zanim jednak będzie to możliwe, należy wybrać polecenie *Tools/References*, aby ustawić odwołania do biblioteki obiektów Worda i wprowadzić następujący kod:

```
Sub GetWordVersion()
    Dim WordApp As New Word.Application
    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```

Na rysunku 18.6 pokazano, w jaki sposób funkcja *Auto List Members* wspomaga mechanizm wczesnego wiązania. Jeżeli wykorzystujesz późne wiązanie, VBE nie zapewnia takiego wsparcia.

Funkcja `GetObject` a `CreateObject`

Obie funkcje VBA, `GetObject` i `CreateObject`, zwracają odwołanie do obiektu, ale działają w inny sposób.

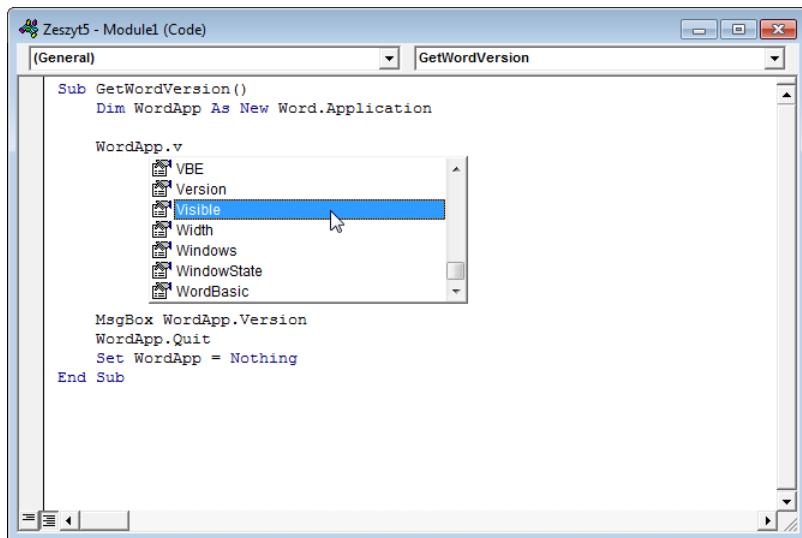
Funkcja `CreateObject` służy do tworzenia interfejsu dla nowego egzemplarza aplikacji. Należy z niej skorzystać, jeżeli aplikacja nie działa. Nawet jeśli egzemplarz aplikacji został wcześniej uruchomiony, funkcja spowoduje uruchomienie nowego egzemplarza aplikacji. Na przykład poniższa instrukcja uruchomi Excela, a obiekt zwrócony w zmiennej `XLApp` będzie odwołaniem do utworzonego obiektu `Excel.Application`:

```
Set XLApp = CreateObject("Excel.Application")
```

Funkcję `GetObject` wykorzystuje się z aplikacją, która już działa, lub w celu uruchomienia aplikacji z jednokrotnym załadowaniem pliku. Na przykład poniższa instrukcja spowoduje uruchomienie Excela i jednokrotnie załadowanie pliku *MójPlik.xlsx*. Obiekt zwrócony w zmiennej `XLBook` będzie odwołaniem do obiektu `Workbook` (pliku *MójPlik.xlsx*).

```
Set XLBook = GetObject("C:\MójPlik.xlsx")
```

Rysunek 18.6.
Użycie wczesnego wiązania pozwala na wykorzystanie wsparcia ze strony funkcji Auto List Members



Sterowanie Wordem z poziomu Excela

W przykładzie zaprezentowanym w tym punkcie zademonstrujemy sesję automatyzacji z wykorzystaniem Worda. Procedura MakeMemos utworzy trzy notatki zdefiniowane przez użytkownika w Wordzie, a następnie zapisze każdy dokument do pliku. Informacje wykorzystane do utworzenia notatek są zapisane w arkuszu Excela, tak jak pokazano na rysunku 18.7.

	A	B	C	D	E	F	G
1	Region1	477	881 466 zł				
2	Region2	375	753 459 zł				
3	Region3	762	1 545 987 zł				

Tworzenie notatek na podstawie danych z arkusza

Wiadomość:
Miesięczne dane dotyczące sprzedaży w Waszym regionie zostały zamieszczone poniżej. Informacje te uzyskano z centralnej bazy danych.
Dobra robota - strajacie się nadal otrzymywać takie rezultaty!
W przypadku pytań proszę o kontakt telefoniczny.

Rysunek 18.7. Word automatycznie generuje trzy notatki na podstawie danych zapisanych w arkuszu Excela

Procedura MakeMemos rozpoczęta działanie od utworzenia obiektu WordApp. Procedura przetwarza w pętli trzy wiersze danych zapisanych w arkuszu Arkusz1 i wykorzystuje metody i właściwości Worda do utworzenia każdej notatki i zapisania jej na dysku.

Tekst notatki zapisano w zakresie o nazwie Notatka (w komórce E6). Wszystkie działania odbywają się w tle: oznacza to, że Word jest niewidoczny.

```
Sub MakeMemos()
    ' Procedura tworzy notatki w Wordzie za pomocą mechanizmu automatyzacji
    Dim WordApp As Object
    Dim Data As Range, message As String
    Dim Records As Integer, i As Integer
    Dim Region As String, SalesAmt As String, SalesNum As String
    Dim SaveAsName As String

    ' Uruchomienie Worda i utworzenie obiektu (późne wiązanie)
    Set WordApp = CreateObject("Word.Application")

    ' Informacje z arkusza
    Set Data = Sheets("Arkusz1").Range("A1")
    Message = Sheets("Arkusz1").Range("Notatka")

    ' Przetwarzanie w pętli wszystkich rekordów w arkuszu Arkusz1
    Records = Application.CountA(Sheets("Arkusz1").Range("A:A"))
    For i = 1 To Records
        ' Aktualizacja informacji o postępie zadania na pasku stanu
        Application.StatusBar = "Przetwarzanie rekordu " & i

        ' Przypisanie bieżących danych do zmiennych
        Region = Data.Cells(i, 1).Value
        SalesNum = Data.Cells(i, 2).Value
        SalesAmt = Format(Data.Cells(i, 3).Value, "#,000")

        ' Określenie nazwy pliku
        SaveAsName = Application.DefaultFilePath &
            "\\" & Region & ".docx"

        ' Wysyłanie poleceń do Worda
        With WordApp
            .Documents.Add
            With .Selection
                .Font.Size = 14
                .Font.Bold = True
                .ParagraphFormat.Alignment = 1
                .TypeText Text:="N O T A T K A"
                .TypeParagraph
                .TypeParagraph
                .Font.Size = 12
                .ParagraphFormat.Alignment = 0
                .Font.Bold = False
                .TypeText Text:="Data:" & vbTab &
                    Format(Date, "d mmmm, yyyy")
                .TypeParagraph
                .TypeText Text:="Do:" & vbTab & "Menedżer " & vbTab & Region
                .TypeParagraph
                .TypeText Text:="Od:" & vbTab &
                    Application.UserName
                .TypeParagraph
                .TypeParagraph
                .TypeText Message
                .TypeParagraph
                .TypeParagraph
            End With
        End With
    Next i
End Sub
```

```

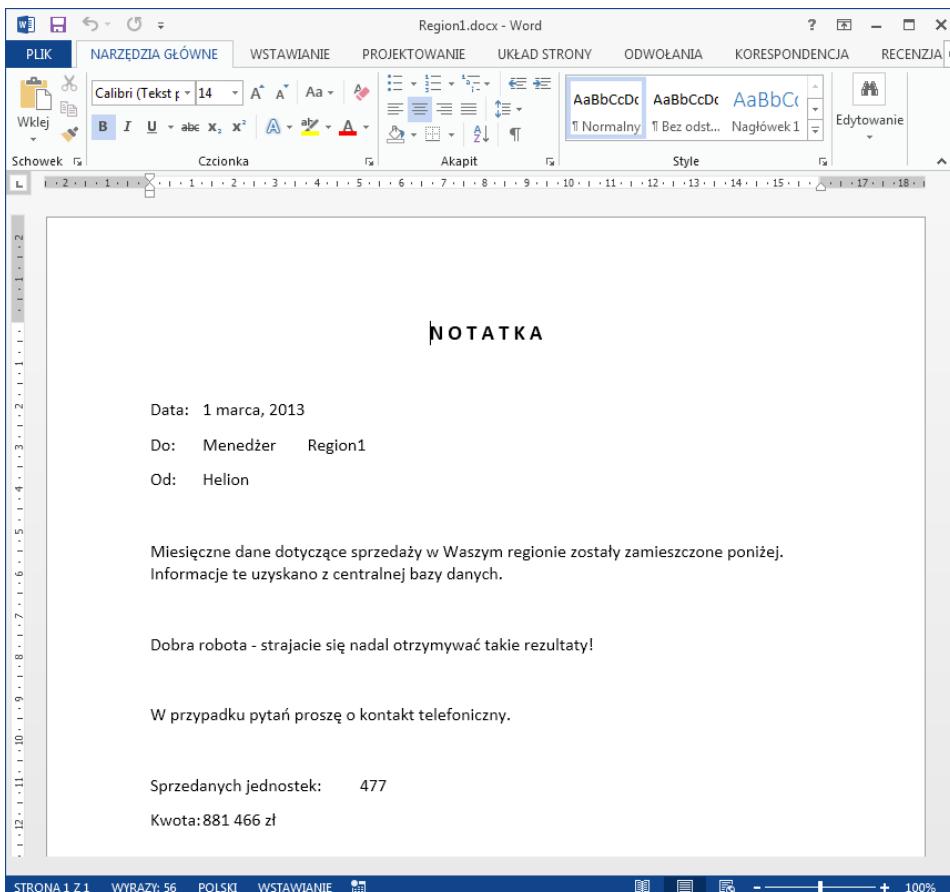
        .TypeText Text:="Sprzedanych jednostek:" & vbTab & SalesNum
        .TypeParagraph
        .TypeText Text:="Kwota:" & vbTab & _
            Format(SalesAmt, "#,##0 zł")
    End With
    .ActiveDocument.SaveAs FileName:=SaveAsName
End With
Next i

' Zniszczenie obiektu
WordApp.Quit
Set WordApp = Nothing

' Odtworzenie paska stanu
Application.StatusBar = ""
MsgBox Records & " notatki utworzono i zapisano w " & _
    ThisWorkbook.Path
End Sub

```

Na rysunku 18.8 zaprezentowano dokument utworzony za pomocą procedury MakeMemos.



Rysunek 18.8. Ten dokument Worda został utworzony za pomocą procedury Excela



Skoroszyt z tym przykładem (*Tworzenie notatek.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Tworzenie powyższego makra składało się z kilku etapów. Najpierw zarejestrowałem w Wordzie makro obejmujące działania tworzenia nowego dokumentu, wprowadzania i formatowania tekstu oraz zapisywania pliku. Dzięki temu uzyskałem informacje o właściwościach i metodach potrzebnych do wykonania zadania. Następnie skopiowałem makro do modułu Excela z zastosowaniem konstrukcji *With ... End With*. Na początku każdej instrukcji pomiędzy słowami kluczowymi *With* i *End With* wprowadziłem kropkę. Oryginalne makro zawierało na przykład następującą instrukcję:

```
Documents.Add
```

Instrukcję tę zmodyfikowałem w następujący sposób:

```
With WordApp  
    .Documents.Add  
    ' pozostałe instrukcje  
End With
```

W makrze, które zarejestrowałem w Wordzie, występowało kilka wbudowanych stałych. Ponieważ w tym przykładzie wykorzystałem późne wiązanie, musiałem zastąpić stałe ich rzeczywistymi wartościami. Wartości te odczytałem w oknie *Immediate* edytora Visual Basic.

Zarządzanie Exceliem z poziomu innej aplikacji

W razie potrzeby możesz również zarządzać Exceliem z poziomu innej aplikacji (np. programu napisanego w innym języku programowania lub procedury VBA w Wordzie). Przykładowo, możesz wykonać obliczenia w Excelu i zwrócić wynik do Worda.

Obiekty Excela możemy tworzyć w następujący sposób:

- Obiekt Application za pomocą funkcji `CreateObject("Excel.Application")`.
- Obiekt Workbook za pomocą funkcji `CreateObject("Excel.Sheet")`.
- Obiekt Chart za pomocą funkcji `CreateObject("Excel.Chart")`.

Poniżej prezentuję procedurę umieszczoną w module VBA dokumentu Worda 2013. Procedura tworzy obiekt Worksheet Excela (którego nazwa symboliczna to `Excel.Sheet`) na podstawie istniejącego skoroszytu i wkleja go do dokumentu edytora Word.

```
Sub MakeLoanTable()  
    Dim XLSheet As Object  
    Dim LoanAmt  
    Dim Wbook As String  
  
    ' Zapytanie o wartości  
    LoanAmt = InputBox("Kwota pożyczki?")  
    If LoanAmt = "" Then Exit Sub
```

```
' Czyszczenie dokumentu
ThisDocument.Content.Delete

' Tworzenie obiektu Sheet
Wbook = ThisDocument.Path & "\Obliczanie pożyczki.xlsx"
Set XLSheet = GetObject(Wbook, "Excel.Sheet").ActiveSheet

' Umieszczanie wartości na arkuszu
XLSheet.Range("LoanAmount") = LoanAmt
XLSheet.Calculate

' Wstawianie nagłówka strony
Selection.Style = "Title"
Selection.TypeText "Kwota pożyczki: " & _
    Format(LoanAmt, "#,##0 zł")
Selection.TypeParagraph
Selection.TypeParagraph

' Kopiowanie danych z arkusza i wklejanie do dokumentu
XLSheet.Range("DataTable").Copy
Selection.Paste

Selection.TypeParagraph
Selection.TypeParagraph

' Kopiowanie wykresu i wklejanie do dokumentu
XLSheet.ChartObjects(1).Copy
Selection.PasteSpecial _
    Link:=False, _
    DataType:=wdPasteMetafilePicture, _
    Placement:=wdInLine

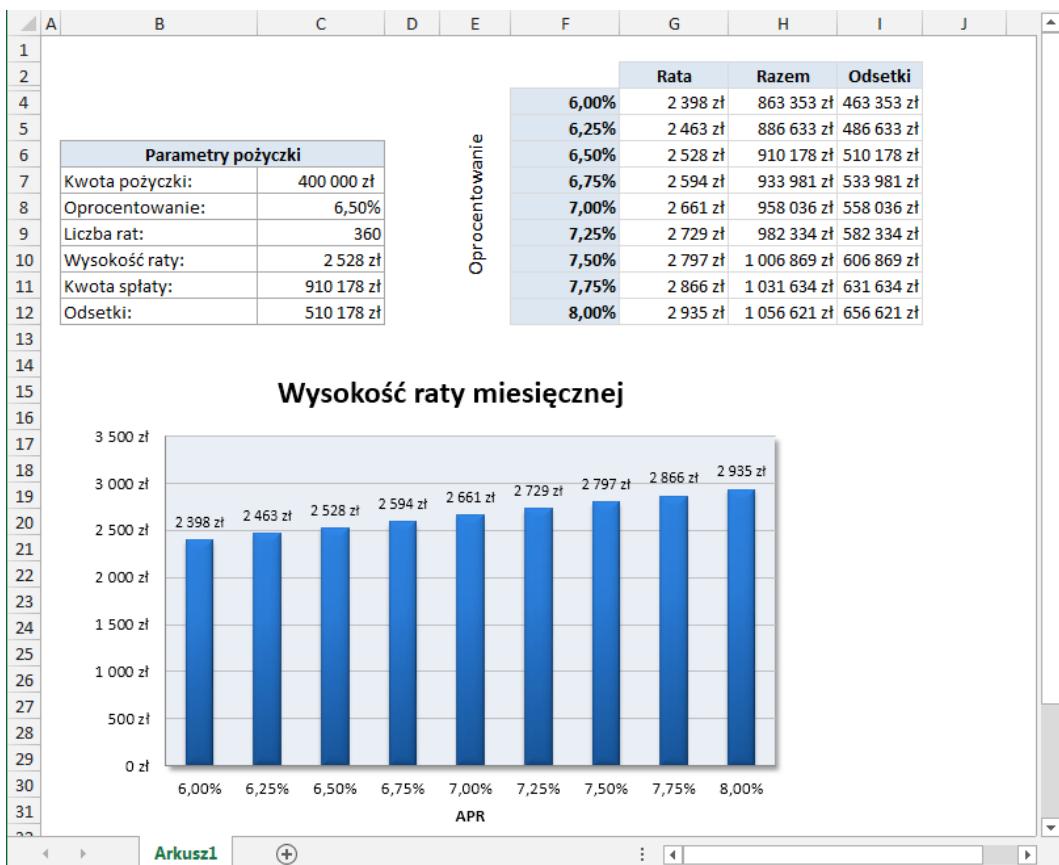
' Usuwanie obiektu
Set XLSheet = Nothing
End Sub
```



W sieci Dokument Worda z tym przykładem (*Automatyzacja Excela.docm*) oraz skoroszyt Excela (*Obliczanie pożyczki.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Po otwarciu dokumentu Worda uruchom makro *MakeLoanTable*. Aby to zrobić, przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Oblicz kredyt*, znajdujący się w grupie opcji *Kredyt*.

Skoroszyt wykorzystany w procedurze Worda pokazano na rysunku 18.9. Procedura *MakeExcelChart* wyświetla pytanie do użytkownika o wartość pożyczki i wstawia ją do komórki C7 arkusza (komórka nosi nazwę *LoanAmount*).

Przeliczenie arkusza powoduje aktualizację danych w tabeli zlokalizowanej w zakresie F2:I12 (o nazwie *DataTable*) i aktualizuje wykres. Dane *DataTable* wraz z wykresem są następnie kopowane z obiektu Excela i wklejane do nowego dokumentu Worda. Wyniki pokazano na rysunku 18.10.

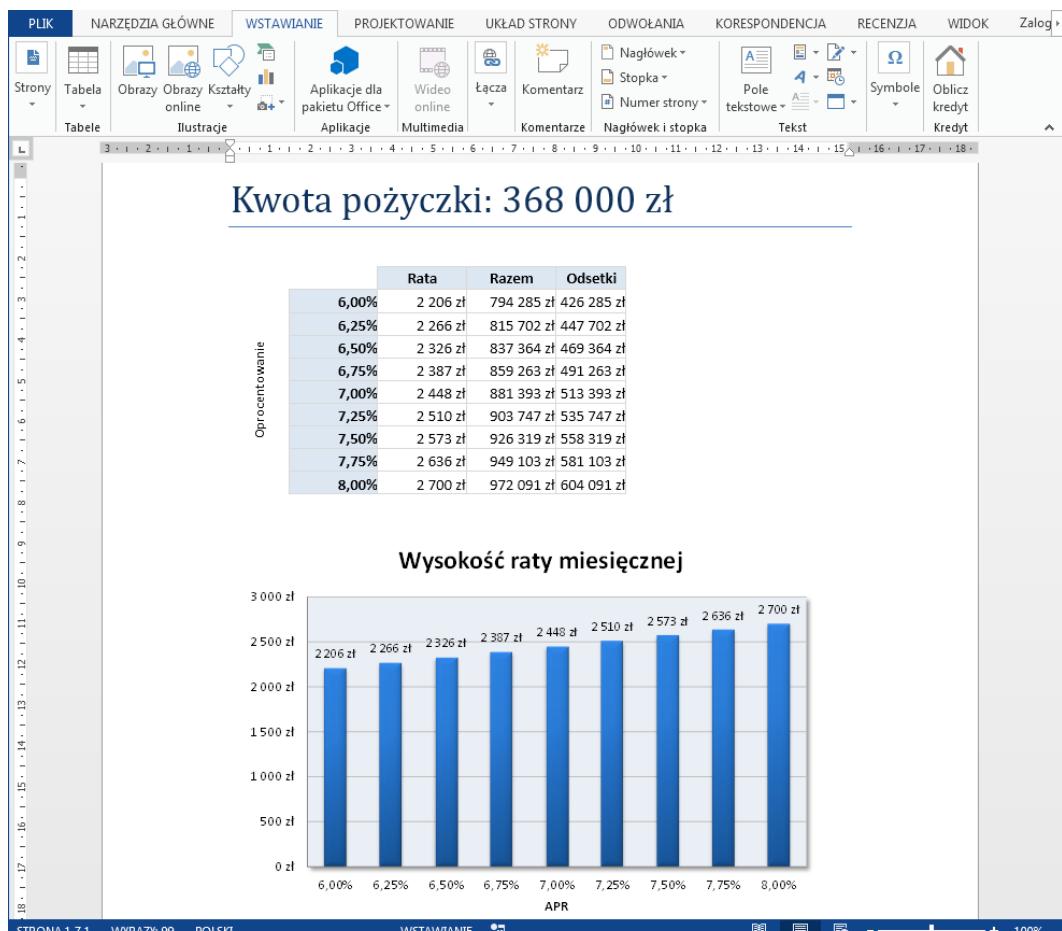


Rysunek 18.9. Ten arkusz jest wykorzystywany przez procedurę VBA w dokumencie Worda

Wysyłanie spersonalizowanych wiadomości e-mail z wykorzystaniem Outlooka

W przykładzie zaprezentowanym w tym podrozdziale zademonstrowano sesję automatyzacji z wykorzystaniem programu Microsoft Outlook.

Na rysunku 18.11 pokazano arkusz zawierający dane wykorzystane w wiadomościach e-mail: nazwisko, adres e-mail oraz kwotę premii. Procedura `SendMail` przetwarza w pętli wiersze w arkuszu, pobiera dane i tworzy indywidualną wiadomość (zapisaną w zmiennej `Msg`).



Rysunek 18.10. Procedura VBA Worda wykorzystała Excela do utworzenia tego dokumentu

Rysunek 18.11.
Informacje zapisane w tym arkuszu są wykorzystywane w wiadomościach e-mail programu Outlook Express

	A	B	C	D
1	Nazwisko	Email	Premia	
2	Jan Jasiński	jjasinski@mojadomena.pl	12 000,00 zł	
3	Bogdan Kowalski	bkowalski@mojadomena.pl	13 500,00 zł	
4	Franciszek Dolas	fdolas@mojadomena.pl	11 250,00 zł	
5	Anna Nowak	anowak@mojadomena.pl	14 000,00 zł	
6				
7				

```
Sub SendEmail()
    ' Wykorzystuje wczesne wiązanie
    ' Wymaga odwołania do biblioteki obiektów Outlooka
    Dim OutlookApp As Outlook.Application
    Dim MItem As Outlook.MailItem
    Dim cell As Range
    Dim Subj As String
    Dim EmailAddr As String
    Dim Recipient As String
```

```

Dim Bonus As String
Dim Msg As String

' Utworzenie obiektu Outlooka
Set OutlookApp = New Outlook.Application

' Przetwarzanie wierszy w pętli
For Each cell In Columns("B").Cells.SpecialCells(xlCellTypeConstants)
    If cell.Value Like "*@*" Then
        Pobranie danych
        Subj = "Roczna premia"
        Recipient = cell.Offset(0, -1).Value
        EmailAddr = cell.Value
        Bonus = Format(cell.Offset(0, 1).Value, "0 000 zŁ")
    End If
    Next
End Sub

' Utworzenie tekstu wiadomości
Msg = "Szanowny(a) Pan(i) " & Recipient & vbCrLf & vbCrLf
Msg = Msg & "Z przyjemnością zawiadamiam, że Pana(i) roczna premia wynosi"
Msg = Msg & Bonus & vbCrLf & vbCrLf
Msg = Msg & "Wiesław Różański" & vbCrLf
Msg = Msg & "Prezes zarządu"

' Utworzenie wiadomości e-mail i jej wysłanie
Set MItem = OutlookApp.CreateItem(olMailItem)
With MItem
    .To = EmailAddr
    .Subject = Subj
    .Body = Msg
    .Send
End With
End If
Next
End Sub

```

Na rysunku 18.12 przedstawiono wygląd jednej z wiadomości, wyświetlonej w programie Outlook.

W tym przykładzie wykorzystano wcześnie wiązanie, a zatem jest wymagane odwołanie do biblioteki obiektów Outlooka. Zwróćmy uwagę, że wykorzystano dwa obiekty: Outlook oraz MailItem. Obiekt Outlook jest tworzony za pomocą następującej instrukcji:

```
Set OutlookApp = New Outlook.Application
```

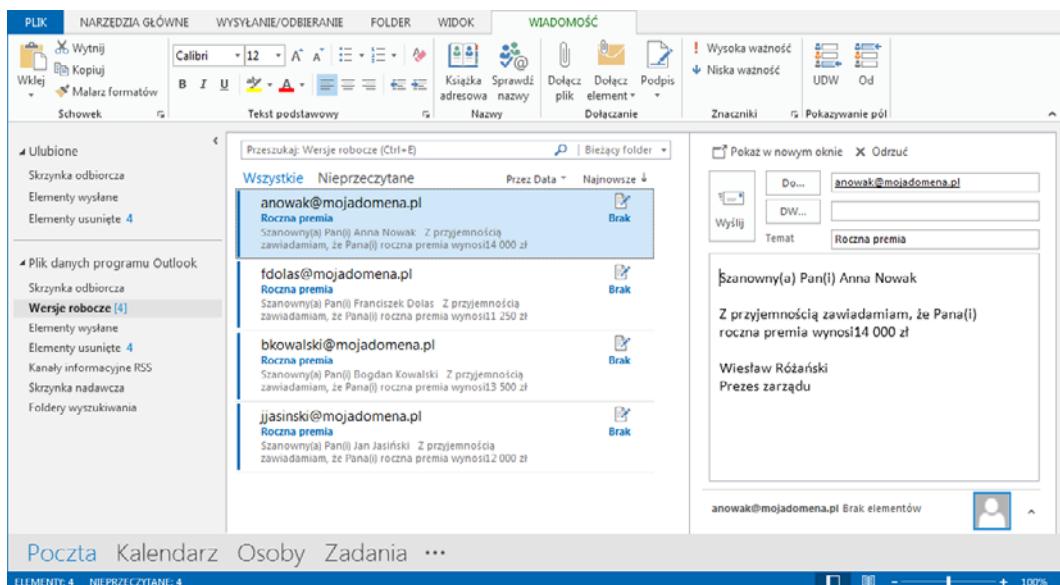
Obiekt MailItem jest tworzony za pomocą następującej instrukcji:

```
Set MItem = OutlookApp.CreateItem(olMailItem)
```

Kod ustawia właściwości To, Subject oraz Body, a następnie wykorzystuje metodę Send w celu wysłania wiadomości.



Aby zamiast wysyłania zapisać wiadomości w folderze *Wersje robocze*, powinieneś zamiast metody Send użyć metody Save. Taka zmiana będzie bardzo użyteczna zwłaszcza na etapie testowania i usuwania błędów procedury.



Rysunek 18.12. Wiadomość w programie Outlook utworzona przez makro Excela



Skoroszyt z tym przykładem (*Wysyłanie wiadomości — Outlook.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>). Aby go uruchomić, trzeba mieć zainstalowany program Microsoft Outlook. Na płycie znajdziesz również zmodyfikowaną wersję tego skoroszytu, w której wykorzystano późne wiązanie (skoroszyt o nazwie *Outlook — późne wiązanie.xlsxm*).

Wysyłanie wiadomości e-mail z załącznikami z poziomu Excela

Jak zapewne wiesz, Excel pozwala na wysyłanie arkuszy lub skoroszytów za pomocą poczty elektronicznej. Oczywiście do automatyzacji tego typu zadań można wykorzystać język VBA. Procedura zaprezentowana poniżej używa metody `SendMail` do wysłania aktywnego skoroszytu (jako załącznik) pod adres *jankowalski@mojadomena.pl*. Tematem wiadomości jest tekst *Mój skoroszyt*.

```
Sub SendWorkbook()
    ActiveWorkbook.SendMail "jankowalski@mojadomena.pl", "Mój skoroszyt"
End Sub
```



Metoda `SendMail` korzysta z domyślnego klienta poczty elektronicznej.

Aby z całego skoroszytu wysłać pocztą elektroniczną tylko jeden arkusz, należy skopiować arkusz do nowego (tymczasowego) skoroszytu, przesłać go jako załącznik, a następnie zamknąć plik tymczasowy. Poniżej zaprezentowano przykład wysłania arkusza *Arkusz1* z aktywnego skoroszytu, który zostanie dołączony do wiadomości o tytule *Mój arkusz*. Zauważ, że po wysłaniu skopiowane arkusze znajdują się w aktywnym skoroszycie.

```
Sub Sendsheet()
    ActiveWorkbook.Worksheets("Arkusz1").Copy
    ActiveWorkbook.SendMail "jankowalski@mojadomena.pl", "Mój arkusz"
    ActiveWorkbook.Close False
End Sub
```

W poprzednim przykładzie plik będzie miał domyślną nazwę skoroszytu (np. *Zeszyt2.xlsx*). Aby nadać załącznikowi składającemu się z pojedynczego arkusza bardziej opisową nazwę, należy zapisać tymczasowy skoroszyt i usunąć go po wysłaniu. Poniższa procedura powoduje zapisanie arkusza Arkusz1 do pliku o nazwie *mój plik.xlsx*. Po wysłaniu tymczasowego skoroszytu jako załącznika e-mail do usunięcia pliku wykorzystywana jest instrukcja Kill języka VBA.

```
Sub SendOneSheet()
    Dim Filename As String
    Filename = "mój plik.xlsx"
    ActiveWorkbook.Worksheets("Arkusz1").Copy
    ActiveWorkbook.SaveAs Filename
    ActiveWorkbook.SendMail "jankowalski@mojadomena.com", "Mój arkusz"
    ActiveWorkbook.Close False
    Kill Filename
End Sub
```



Niestety Excel nie pozwala na automatyzację procesu zapisywania skoroszytu w formacie PDF i wysyłania go w postaci załącznika wiadomości poczty elektronicznej. W razie potrzeby możesz jednak zautomatyzować przynajmniej pewną część tego procesu. Procedura SendSheetAsPDF, której kod przedstawiono poniżej, zapisuje aktywny arkusz w pliku PDF i następnie wyświetla okno tworzenia wiadomości domyślnego klienta poczty elektronicznej (z plikiem PDF dołączonym do wiadomości). Dzięki takiemu rozwiązaniu pozostało Ci tylko dopisać adresata i nacisnąć przycisk *Wyślij*.

```
Sub SendSheetAsPDF()
    CommandBars.ExecuteMso ("FileEmailAsPdfEmailAttachment")
End Sub
```

Kiedy Excelowi brakuje sił do wykonania zadania, czas wezwać na odsiecz Outlooka. Procedura, której kod zamieszczono poniżej, zapisuje aktywny skoroszyt w postaci pliku PDF i wykorzystuje Outlooka do utworzenia wiadomości z załącznikiem w postaci pliku PDF.

```
Sub Sen Sub SendAsPDF()
    ' Wykorzystuje wczesne wiązanie
    ' Wymaga odwołania do biblioteki obiektów Outlooka
    Dim OutlookApp As Outlook.Application
    Dim MItem As Object
    Dim Recipient As String, Subj As String
    Dim Msg As String, Fname As String

    ' Szczegóły wiadomości
    Recipient = "myboss@xrediyh.com"
    Subj = "Raport sprzedaży"
    Msg = "Siema szefie! W załączniu plik PDF, o który prosiłeś. "
    Msg = Msg & vbNewLine & vbNewLine & "-Franek"
    Fname = Application.DefaultFilePath & "\" & _
        ActiveWorkbook.Name & ".pdf"
```

' *Tworzenie załącznika*

```
ActiveSheet.ExportAsFixedFormat _  
    Type:=xlTypePDF, _  
    Filename:=Fname
```

' *Tworzenie obiektu Outlooka*

```
Set OutlookApp = New Outlook.Application
```

' *Tworzenie i wysyłanie wiadomości*

```
Set MItem = OutlookApp.CreateItem(olMailItem)  
With MItem  
    .To = Recipient  
    .Subject = Subj  
    .Body = Msg  
    .Attachments.Add Fname  
    .Save 'to Drafts folder  
    '.Send  
End With  
Set OutlookApp = Nothing
```

' *Usuwanie pliku*

```
Kill Fname  
End Sub
```

Skoroszyt z tym przykładem (*Wysyłanie PDF z Outlooka.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Rozdział 19.

Tworzenie i wykorzystanie dodatków

W tym rozdziale:

- Czym są dodatki i dlaczego są tak ważne?
- Menedżer dodatków Excela
- Tworzenie dodatków
- Czym pliki *XLAM* różnią się od plików *XLSM*
- Praca z dodatkami za pomocą kodu VBA
- Jak sprawdzić, czy dodatek został poprawnie zainstalowany

Czym są dodatki?

Jedną z właściwości Excela, która najbardziej przydaje się programistom, jest możliwość tworzenia dodatków. Tworzenie dodatków sprawia, że aplikacje wyglądają bardziej profesjonalne, a jak się sam przekonasz, dodatki mają pewną przewagę nad standardowymi plikami skoroszytów.

Ogólnie rzecz biorąc, *dodatki* są mechanizmami, które wzbogacają arkusze o nowe właściwości funkcyjne. Na przykład pakiet *Analysis ToolPak*, jeden z najpopularniejszych dodatków dostarczanych z Exceliem, umożliwia wykonywanie obliczeń statystycznych oraz analitycznych, których standardowo nie ma w Excelu.

Niektóre dodatki (m.in. właśnie *Analysis ToolPak*) dostarczają nowych funkcji arkuszowych, które można stosować w formułach. Nowe właściwości zwykle dobrze integrują się z oryginalnym interfejsem i użytkownik odnosi wrażenie, że są częścią samego programu.

Porównanie dodatku ze standardowym skoroszytem

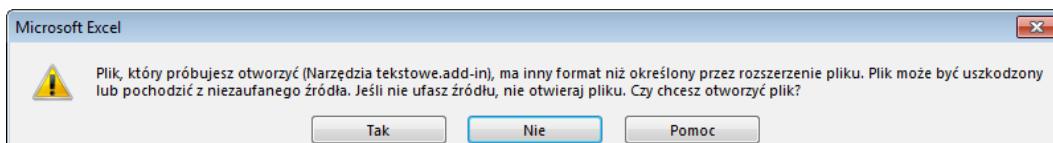
Każdy użytkownik Excela mający odpowiednią wiedzę potrafi utworzyć dodatek na podstawie normalnego skoroszytu. Nie są do tego potrzebne żadne dodatkowe programy ani

narzędzia. Teoretycznie na dodatki można przekształcić wszystkie skoroszyty, ale nie każdy skoroszyt dobrze się do tego nadaje. Dodatki Excela to w zasadzie normalne skoroszyty XLSM, z następującymi różnicami:

- Właściwość `IsAddin` obiektu `ThisWorkbook` ma wartość `True`. Domyślnie dla skoroszytów ta właściwość ma wartość `False`.
- Okno skoroszytu jest ukryte w taki sposób, że nie można go odkryć za pomocą polecenia `Odkryj` znajdującego się w grupie opcji `Okno` na karcie `WIDOK`. Oznacza to, że nie można wyświetlić arkuszy lub wykresów z dodatku bez użycia kodu, który kopiuje arkusz do standardowego skoroszytu.
- Dodatki nie należą do kolekcji `Workbooks` — zamiast tego należą do kolekcji `AddIns`. W razie potrzeby *można* jednak uzyskać dostęp do dodatku za pośrednictwem kolekcji `Workbooks` (zobacz punkt „Pliki XLAM — przynależność do kolekcji z poziomu VBA” w dalszej części rozdziału).
- Dodatki można instalować i usuwać za pomocą okna dialogowego `Dodatki`. Po zainstalowaniu dodatki są dostępne dla wszystkich sesji Excela.
- Okno dialogowe *Makro* (wywoływanie poleceniem *Makra* znajdującym się w grupie opcji *Kod* na karcie *DEVELOPER* lub poleceniem *Makra* znajdującym się w grupie opcji *Makra* na karcie *WIDOK*) nie wyświetla nazw makr zapisanych w dodatku.
- Indywidualne funkcje arkuszowe zapisane w dodatku można stosować w formułach bez konieczności poprzedzania nazwy funkcji nazwą źródłowego pliku skoroszytu.



W przeszłości Excel pozwalał na nadawanie plikom dodatków dowolnych rozszerzeń. Począwszy od Excela 2007 możesz ponownie korzystać z takich plików, ale jeśli plik dodatku będzie miał rozszerzenie inne niż *XLA* lub *XLAM*, na ekranie pojawi się okno dialogowe zawierające ostrzeżenie, które zostało przedstawione na rysunku 19.1. Ostrzeżenie pojawia się nawet wówczas, gdy dodatek został już zainstalowany i jest uruchamiany automatycznie podczas uruchamiania programu Excel, a także wtedy, gdy taki plik znajduje się w zaufanej lokalizacji. Microsoft nazywa ten mechanizm zabezpieczeń *utwardzaniem rozszerzeń* (ang. *extension hardening*).



Rysunek 19.1. Jeżeli plik dodatku ma niestandardowe rozszerzenie, Excel wyświetla na ekranie odpowiednie ostrzeżenie

Po co tworzy się dodatki?

Aplikacje Excela można przekształcić na dodatki w celu:

- **Ograniczenia dostępu do kodu i arkuszy** — Kiedy rozpowszechniamy aplikacje w formie dodatku i zabezpieczymy projekt VBA hasłem, użytkownicy nie będą mogli przeglądać lub modyfikować arkuszy ani kodu VBA, który jest z nimi związany. Jeżeli w aplikacji używasz własnych rozwiązań, przekształcenie jej

na postać dodatku zapobiega kopiowaniu kodu przez innych użytkowników, a przynajmniej mocno utrudnia do niego dostęp.

- **Zapobiegania pomyłkom** — Jeżeli użytkownik załaduje aplikację jako dodatek, nie będą widoczne jego arkusze. Dzięki temu istnieje mniejsze prawdopodobieństwo wprowadzenia w błąd początkujących użytkowników. W odróżnieniu od ukrytego skoroszytu dodatku nie można odkryć.
- **Uproszczenia dostępu do funkcji arkusza** — Funkcje arkuszowe definiowane w dodatku nie wymagają stosowania kwalifikatora nazwy skoroszytu. Jeżeli na przykład zapiszemy funkcję użytkownika o nazwie MOVAVG w skoroszycie *Newfuncs.xlsxm*, użycie tej funkcji w innym skoroszycie będzie wymagało zastosowania instrukcji o następującej składni:

```
=Newfuncs.xlsxm!MOVAVG(A1:A50)
```

Natomiast jeżeli funkcja zostanie zapisana w dodatku, możesz skorzystać z instrukcji o prostszej składni, w której nie trzeba używać odwołania do pliku:

```
=MOVAVG(A1:A50)
```

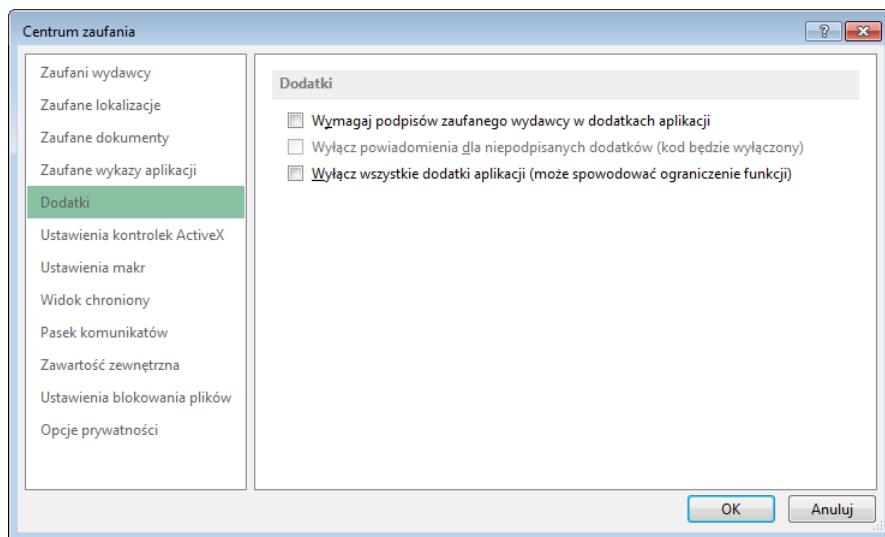
- **Ułatwienia użytkownikom dostępu do właściwości aplikacji**
— Po zidentyfikowaniu położenia dodatku zostanie on umieszczony w oknie dialogowym *Dodatki* wraz z przyjazną nazwą i opisem działania.
- **Uzyskania lepszej kontroli nad ładowaniem aplikacji** — Dodatki można otwierać automatycznie przy uruchomieniu Excela, niezależnie od katalogu, w którym zostały zapisane.
- **Uniknięcia wyświetlania pytań systemowych podczas zamykania** — Podczas zamykania dodatku nie jest wyświetlane okno dialogowe z pytaniem o zapisanie zmian.



Możliwość korzystania z dodatków przez użytkownika jest określana przez ustawienia zabezpieczeń w oknie dialogowym *Centrum zaufania* (patrz rysunek 19.2). Aby przywołać to okno na ekran, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Bezpieczeństwo makr*, znajdujący się w grupie opcji *Kod*. Jeżeli karta *DEVELOPER* nie jest wyświetlana na Wstążce, przejdź na kartę *PLIK* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Centrum zaufania* i następnie naciśnij przycisk *Ustawienia Centrum zaufania*.

Dodatki COM

Excel obsługuje także dodatki *COM* (ang. *Component Object Model*). Pliki te mają rozszerzenia *.dll* lub *.exe*. Dodatek *COM* można napisać w taki sposób, że będzie działać z wszystkimi aplikacjami pakietu *Office*, które obsługują dodatki. Ponieważ kod dodatków *COM* jest skompilowany, zapewniają one większy poziom bezpieczeństwa. W odróżnieniu od dodatków *XLAM* dodatki *COM* nie mogą zawierać arkuszy lub wykresów Excela. Dodatki *COM* można tworzyć za pomocą Visual Basic .NET. Dokładne omówienie tworzenia dodatków *COM* wykracza daleko poza ramy tej książki.



Rysunek 19.2. Za pomocą tych opcji możesz zdecydować, jakie dodatki mogą być używane

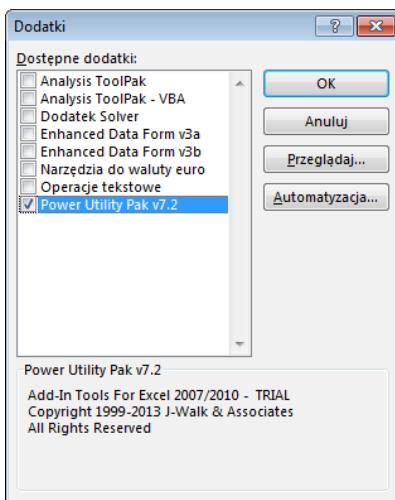
Menedżer dodatków Excela

Najlepszym sposobem ładowania i zamykania dodatków jest wykorzystanie okna dialogowego *Dodatki* w Excelu. Aby przywołać to okno na ekran, możesz użyć jednej z metod przedstawionych poniżej:

- Przejdź na kartę *PLIK* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Dodatki*, a następnie z listy rozwijanej *Zarządzaj* wybierz opcję *Dodatki programu Excel* i naciśnij przycisk *Przejdz*.
- Przejdź na kartę *DEVELOPER* i wybierz polecenie *Dodatki*, znajdujące się w grupie opcji *Dodatki*. Pamiętaj, że domyślnie karta *DEVELOPER* nie jest wyświetlaną na Wstążce.
- Naciśnij sekwencję klawiszy lewy Alt, Q, X.

Na rysunku 19.3 przedstawiono wygląd okna *Dodatki*. Na liście rozwijanej znajdziesz nazwy wszystkich dostępnych w danej chwili dodatków, a zaznaczone pola wyboru wskazują, które dodatki są aktualnie otwarte. Aby otworzyć (zainstalować) bądź zamknąć (odinstalować) wybrany dodatek, powinieneś zaznaczyć odpowiednie pole wyboru lub usunąć jego zaznaczenie. Kiedy zamkñasz wybrany dodatek, nie jest on usuwany z systemu i pozostaje na liście dostępnych dodatków, tak że możesz powrócić do niego i włączyć go później. Aby wyszukać na dysku nowe dodatki i dołączyć je do listy, powinieneś skorzystać z przycisku *Przeglądaj*.

Rysunek 19.3.
Okno dialogowe
Dodatki



Większość dodatków możesz także otwierać za pomocą polecenia *Otwórz* z karty *PLIK*. Ponieważ jednak dodatek nigdy nie będzie aktywnym skoroszytem, nie można go zamknąć za pomocą polecenia *PLIK/Zamknij*. Dodatek można usunąć albo poprzez zamknięcie Excela i jego ponowne załadowanie, albo poprzez wykonanie odpowiedniego kodu VBA, jak na przykład:

```
Workbooks("mojodatek.xlam").Close
```

Otwarcie dodatku za pomocą polecenia *PLIK/Otwórz* powoduje otwarcie pliku, ale po wykonaniu tej czynności dodatek nie jest oficjalnie zainstalowany.

Otwarcie dodatku czasami nie powoduje żadnych widocznych zmian w samym Excelu, ale niemal zawsze zmienia w pewien sposób interfejs użytkownika: na Wstążce pojawia się nowe polecenie czy jedno lub kilka nowych poleceń zostaje dodane do menu podręcznego. Na przykład otwarcie dodatku *Analysis ToolPak* powoduje, że na karcie *DANE* w grupie opcji *Narzędzia danych* pojawia się nowy przycisk o nazwie *Analiza warunkowa*. Kiedy zainstalujesz dodatek *Narzędzia do waluty euro*, na karcie *Formuły* pojawi się nowa grupa opcji o nazwie *Solutions*.

Jeżeli w dodatku znajdują się jedynie nowe funkcje użytkownika, pojawią się one w oknie dialogowym *Wstawianie funkcji* a także w odpowiedniej grupie biblioteki funkcji na Wstążce.



Jeżeli otwierasz dodatek utworzony w wersji wcześniejszej niż Excel 2007, wszelkie modyfikacje interfejsu przeprowadzane przez dodatek nie będą wyglądały tak, jak powinny. Zamiast tego, aby uzyskać dostęp do interfejsu takiego dodatku, będziesz musiał przejść na kartę *DODATKI* i tam skorzystać z poleceń w grupie *Polecenia menu* oraz *Paski narzędzi*.

Tworzenie dodatków

Jak wspomniałem wcześniej, teoretycznie każdy skoroszyt można przekształcić w dodatek, ale w praktyce nie wszystkie skoroszyty się do tego nadają. Skoroszyt, który chcesz przekształcić na dodatek, musi zawierać makra (w przeciwnym wypadku taki dodatek będzie po prostu bezużyteczny).

Najlepszym rozwiązańiem jest przekształcenie na dodatek skoroszytu, który zawiera uniwersalne makra. Skoroszyt składający się tylko z arkuszy nie będzie dostępny, ponieważ z definicji wszystkie arkusze dodatku są ukryte i użytkownik nie ma do nich bezpośredniego dostępu. Możesz jednak napisać kod, który skopiuje wszystkie części arkuszy z dodatku do widocznego skoroszytu.

Utworzenie dodatku na podstawie skoroszytu jest proste. Aby przekształcić standardowy skoroszyt na dodatek, powinieneś wykonać następujące polecenia:

1. Utwórz aplikację i sprawdź, czy wszystko działa poprawnie.
2. Nie zapomnij, że należy zapewnić sposób uruchomienia makra lub makr w dodatku.
Więcej szczegółowych informacji na temat modyfikowania interfejsu programu Excel znajdziesz w rozdziałach 20. i 21.
3. Uaktywnij edytor Visual Basic i wybierz skoroszyt w oknie *Project*.
4. Wybierz polecenie *Tools/xxx Properties* (gdzie xxx reprezentuje nazwę projektu), a następnie kliknij kartę *Protection*. Zaznacz pole wyboru *Lock Project for Viewing* i wprowadź hasło (dwukrotnie). Naciśnij przycisk *OK*.

Ten krok jest wymagany tylko wtedy, kiedy chcesz zablokować możliwość przeglądania makr lub modyfikowania okien *UserForm* przez innych użytkowników.

5. Ponownie uaktywnij Excela, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Panel dokumentu*, znajdujący się w grupie polecień *Modyfikowanie*.
6. Na ekranie pojawi się panel *Właściwości dokumentu*. W polu *Tytuł* wpisz tytuł dodatku, a w polu *Komentarze* wprowadź jego bardziej szczegółowy opis.

Ten krok nie jest wymagany, ale dzięki niemu korzystanie z dodatku jest łatwiejsze, gdyż w oknie dialogowym *Dodatki* wyświetla się opisowy tekst.

7. Przejdź na kartę *PLIK* i z menu wybierz polecenie *Zapisz jako*. Na ekranie pojawi się okno dialogowe *Zapisywanie jako*.
8. W oknie dialogowym *Zapisywanie jako* wybierz z listy rozwijanej *Zapisz jako typ* opcję *Dodatek programu Excel (*.xlam)*.

Excel zaproponuje zapisanie dodatku w domyślnym folderze dodatków, ale w razie potrzeby możesz zapisać dodatek w dowolnie wybranym folderze.

9. Naciśnij przycisk *Zapisz*.

Kopia skoroszytu zostanie zapisana z rozszerzeniem *.xlam*, a oryginalny skoroszyt pozostanie otwarty.

10. Zamknij oryginalny skoroszyt i zainstaluj utworzony dodatek.

11. Przetestuj działanie dodatku i upewnij się, że wszystko działa poprawnie.

Jeżeli nie, dokonaj odpowiednich modyfikacji kodu programu. Nie zapomnij zachować wprowadzonych zmian! Ponieważ dodatki nie pojawiają się w oknie Excela, wszelkie dokonane modyfikacje muszą być zapisywane z poziomu edytora VBE.

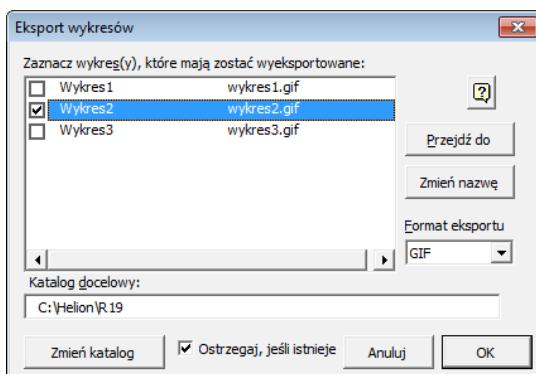


Skoroszyt przekształcany na dodatek musi zawierać co najmniej jeden arkusz komórek, który w momencie tworzenia dodatku musi być arkuszem aktywnym. Jeżeli na przykład aktywnym arkuszem jest arkusz wykresu, opcja *Dodatek programu Excel (*.xlam)* nie pojawi się w oknie dialogowym *Zapisywanie jako*.

Przykład tworzenia dodatku

W tym podrozdziale omówiono czynności, jakie należy wykonać, aby utworzyć użyteczny dodatek. W przykładzie wykorzystamy narzędzie, które utworzyłem do eksportowania wykresów do osobnych plików graficznych. Narzędzie tworzy dodatkową grupę poleceń na karcie kontekstowej *NARZĘDZIA GŁÓWNE/Projektowanie*. Na rysunku 19.4 przedstawiono główne okno dialogowe tego programu. Samo narzędzie jest dosyć złożone i zapoznanie się z jego sposobem działania może wymagać poświęcenia pewnego czasu.

Rysunek 19.4.
Skoroszyt *Eksport wykresów* zostanie przekształcony w użyteczny dodatek



Skoroszyt *Eksport wykresów.xlsxm* zawierający narzędzie *Eksport wykresów* znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Plik ten można wykorzystać w celu utworzenia opisanego poniżej dodatku.

Kilka słów na temat haseł

Pamiętaj, że firma Microsoft nigdy nie twierdziła, że Excel jest produktem, który zapewnia pełne bezpieczeństwo kodu źródłowego przed nieautoryzowanym dostępem. Owszem, mechanizmy ochrony wykorzystujące hasła są wystarczające, aby zablokować przypadkowym użytkownikom dostęp do różnych komponentów skoroszytu. Jeżeli musisz mieć absolutną pewność, że nikt nie będzie miał dostępu do kodu źródłowego aplikacji czy formuł, to być może wybranie Excela jako platformy deweloperskiej nie będzie najlepszym rozwiązaniem.

W tym przykładzie będziemy posługiwać się skoroszytem, który został już wcześniej utworzony i przetestowany. Skoroszyt składa się z następujących komponentów:

- **Arkusz o nazwie Arkusz1** — Ten arkusz nie jest używany, ale musi być obecny ze względu na fakt, że każdy dodatek musi posiadać co najmniej jeden arkusz.
- **Formularz *UserForm* o nazwie *UserForm1*** — Okno tego formularza spełnia rolę zasadniczego interfejsu użytkownika. Moduł kodu tego formularza *UserForm* zawiera kilka procedur obsługującego zdarzeń.
- **Formularz *UserForm* o nazwie *UserForm2*** — Ten formularz jest wyświetlany, kiedy użytkownik naciśnie przycisk *Zmień nazwę* w celu zmiany nazwy pliku eksportowanego wykresu.
- **Formularz *UserForm* o nazwie *UserForm3*** — Okno tego formularza jest wyświetlane po otwarciu skoroszytu i zawiera kilka informacji o sposobie uruchomienia narzędzia *Eksport wykresów*. Na tym formularzu znajduje się również pole wyboru *Nie wyświetlaj więcej tego okna*.
- **Moduł VBA o nazwie *Module1*** — Ten moduł zawiera kilka procedur, między innymi główną procedurę o nazwie *StartExportCharts*, wyświetlającą okno formularza *UserForm1*.
- **Moduł kodu *ThisWorkbook*** — Ten moduł zawiera procedurę *Workbook_Open*, która odczytuje zapisane wcześniej ustawienia i wyświetla wiadomość powitalną.
- **Kod XML** — Kod, którego zadaniem jest modyfikacja Wstążki. Modyfikacja interfejsu odbywa się na zewnątrz programu Excel. Więcej szczegółowych informacji na temat kodu RibbonX używanego do modyfikacji Wstążki znajdziesz w rozdziale 20.

Tworzenie opisu dla dodatku

Aby wprowadzić tytuł i opis tworzonego dodatku, przejdź na kartę *DEVELOPER* i na górnym pasku narzędziowym naciśnij przycisk *Panel dokumentu*, znajdujący się w grupie poleceń *Modyfikowanie*. Na ekranie poniżej Wstążki pojawi się panel właściwości dokumentu.

W polu *Tytuł* wpisz tytuł tworzonego dodatku. Wprowadzony tekst będzie pojawiały się na liście w oknie dialogowym *Dodatki*. W polu *Komentarze* wprowadź bardziej szczegółowy opis dodatku. Ta informacja będzie pojawiała się po wybraniu dodatku w dolnej części okna dialogowego *Dodatki*.

Nadanie tytułu i wprowadzenie opisu dodatku jest co prawda czynnością opcjonalną, ale bardzo zalecaną.

Tworzenie dodatku

Aby utworzyć dodatek, wykonaj następujące czynności:

1. Uaktywnij edytor Visual Basic i w oknie *Project* wybierz skoroszyt, który chcesz przekształcić na dodatek.

2. Wybierz polecenie *Debug/Compile*.

Wybranie tego polecenia wymusza komplikację kodu VBA oraz pozwala znaleźć i poprawić błędy składni. Podczas zapisywania skoroszytu jako dodatku Excel utworzy dodatek nawet wtedy, gdy kod zawiera błędy składni.

3. Wybierz polecenie *Tools/xxx Properties*, aby wyświetlić okno dialogowe właściwości projektu (xxx reprezentuje nazwę projektu). Kliknij zakładkę *General* i wprowadź nową nazwę projektu.

Domyślnie wszystkim projektom VB jest nadawana nazwa *VBProject*.

W prezentowanym przykładzie nazwę projektu zmieniono na *ExpCharts*. Zmiana nazwy projektu nie jest obowiązkowa, ale zalecana.

4. Zapisz skoroszyt po raz ostatni używając jego nazwy z rozszerzeniem **.XLSM*.

Zapisanie skoroszytu — podobnie jak poprzednio — nie jest obowiązkowe, ale dzięki temu będziesz dysponował aktualną kopią dodatku w postaci pliku *XLSM* (bez hasła).

5. W oknie dialogowym *Project Properties* kliknij zakładkę *Protection*. Zaznacz pole wyboru *Lock Project for Viewing*, a następnie wprowadź hasło (dwukrotnie) i naciśnij przycisk *OK*.

Kod projektu nadal będzie można przeglądać bez podawania hasła. Zabezpieczenie hasłem będzie aktywne od następnego otwarcia pliku. Jeżeli nie chcesz zabezpieczać projektu hasłem, możesz pominąć ten punkt.

6. Wróć do Excela, przejdź na kartę *PLIK* i z menu wybierz polecenie *Zapisz jako*.

Na ekranie pojawi się okno dialogowe *Zapisywanie jako*.

7. Z listy rozwijanej *Zapisz jako typ* wybierz opcję *Dodatek programu Excel (*.xlam)*.**8.** Naciśnij przycisk *Zapisz*.

Excel utworzy nowy dodatek, a oryginalna wersja skoroszytu w postaci pliku *XLSM* pozostanie otwarta.

Excel zaproponuje zapisanie dodatku w domyślnym folderze dodatków, ale w razie potrzeby możesz zapisać dodatek w dowolnie wybranym folderze.

Instalowanie dodatku

Aby uniknąć pomyłek, przed instalacją dodatku utworzonego na podstawie skoroszytu *XLSM* należy zamknąć ten skoroszyt.

Aby zainstalować wybrany dodatek wykonaj następujące polecenia:

1. Przejdź na kartę *Plik* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Dodatki*.

Menedżer dodatków Excela

Dodatki możesz instalować i usuwać za pomocą okna dialogowego *Dodatki*, w którym znajdziesz listę wszystkich dostępnych dodatków (zainstalowane dodatki są wyróżnione zaznaczonym polem wyboru obok ich nazwy).

Posługując się terminologią języka VBA można powiedzieć, że w oknie dialogowym *Dodatki* wyświetla się właściwości *Title* wszystkich obiektów AddIn należących do kolekcji AddIns. Wszystkie dodatki, dla których zaznaczono pole wyboru, mają ustawioną właściwość *Installed* na wartość *True*.

Dodatek można zainstalować, zaznaczając pole wyboru znajdujące się obok jego nazwy. Aby zamknąć otwarty dodatek, wystarczy anulować zaznaczenie pola wyboru powiązanego z nazwą dodatku. Aby dodać dodatek do listy, powinieneś użyć przycisku *Przeglądaj* do odszukania pliku. Domyślnie w oknie dialogowym *Dodatki* można przeglądać pliki następujących typów:

- **XLAM** — dodatek programu Excel 2007 lub nowszej wersji, utworzony na podstawie pliku *XLSM*.
- **XLA** — dodatek programu Excel w wersji wcześniejszej niż Excel 2007, utworzony na podstawie pliku *XLS*.
- **XLL** — samodzielny, skompilowany plik *DLL*.

Kliknięcie przycisku *Automatyzacja* pozwala na przeglądanie dodatków *COM*. Warto zwrócić uwagę, że w oknie dialogowym *Serwery automatyzacji* wyświetla się wiele plików, a lista ta nie ogranicza się tylko do dodatków *COM* działających w Excelu.

Plik dodatku można wprowadzić do kolekcji AddIns, używając metody *Add*. Za pomocą kodu VBA nie można jednak usunąć dodatku z listy. Można też otworzyć dodatek, ustawiając właściwość *Installed* obiektu AddIn na wartość *True*. Ustawienie tej właściwości na wartość *False* spowoduje zamknięcie dodatku.

Menedżer dodatków zapisuje stan instalacji dodatków w rejestrze Windows w momencie zamknięcia Excela. Dzięki temu wszystkie zainstalowane dodatki są automatycznie otwierane przy jego następnym uruchomieniu.

2. Z listy rozwijanej *Zarządzaj* wybierz opcję *Dodatki programu Excel* i naciśnij przycisk *Przejdź*.

Na ekranie pojawi się okno dialogowe *Dodatki*.

3. Kliknij przycisk *Przeglądaj* i odszukaj dodatek, który właśnie utworzyłeś.

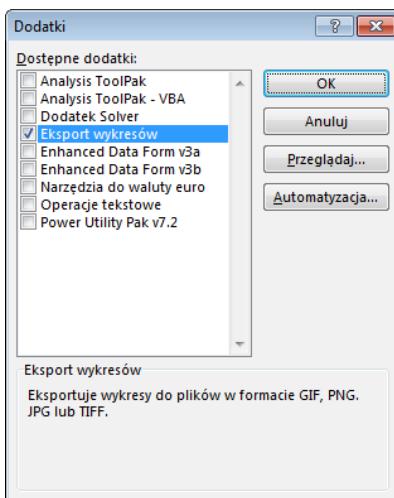
Odnaleziony dodatek zostanie wyświetlony na liście w oknie dialogowym *Dodatki*. Jak pokazano na rysunku 19.5, w oknie dialogowym *Dodatki* wyświetla się również opis dodatku, który wprowadziłeś w oknie dialogowym *Właściwości*.

4. Kliknij *OK*, aby zamknąć okno dialogowe i otworzyć dodatek.

Po otwarciu dodatku *Eksport wykresów* na karcie kontekstowej *NARZĘDZIA WYSZÓW/Projektowanie* pojawi się nowa grupa, o nazwie *Eksport wykresów*, w której znajdziesz przyciski dwóch nowych poleceń. Pierwsze z nich wyświetla okno dialogowe *Eksport wykresów*, a drugie wyświetla plik pomocy.

Dodatek możesz również wywołać, naciskając kombinację klawiszy *Ctrl+Shift+E*.

Rysunek 19.5.
Okno dialogowe
Dodatki z zaznaczonym
nowym dodatkiem



Testowanie dodatków

Po zainstalowaniu dodatku warto pomyśleć o przeprowadzeniu dodatkowych testów. Na przykład aby przetestować nasz dodatek, otwórz nowy skoroszyt, utwórz kilka wykresów i wypróbuj różne funkcje dodatku *Eksport wykresów*. Wypróbuj różne ustawienia i spróbuj spowodować wygenerowanie błędu. Do przeprowadzenia testu możesz również skorzystać z pomocy osoby, która nie zna tej aplikacji i nie wie, jak się nią posługiwać.

Jeżeli odkryjesz jakieś błędy, możesz poprawić kod bezpośrednio w dodatku (oryginalny skoroszyt nie jest potrzebny). Po wprowadzeniu zmian zapisz plik, wybierając z menu głównego edytora VBE polecenie *File/Save*.

Dystrybucja dodatków

Utworzony dodatek można udostępnić innym użytkownikom, dając im kopię pliku *XLAM* wraz z instrukcją instalacji (skoroszyt *XLSM* nie jest im potrzebny). Jeżeli zablokowałeś plik za pomocą hasła, to użytkownicy, którzy nie znają hasła, nie będą mogli przeglądać ani modyfikować kodu.

Modyfikowanie dodatku

Aby zmodyfikować dodatek, należy go otworzyć i odblokować. W celu odblokowania należy uaktywnić edytor Visual Basic, po czym dwukrotnie kliknąć nazwę projektu w oknie *Project*. Wyświetli się pytanie o hasło. Po podaniu właściwego hasła należy wprowadzić poprawki, a następnie zapisać plik w edytorze Visual Basic, wybierając polecenie *File/Save*.

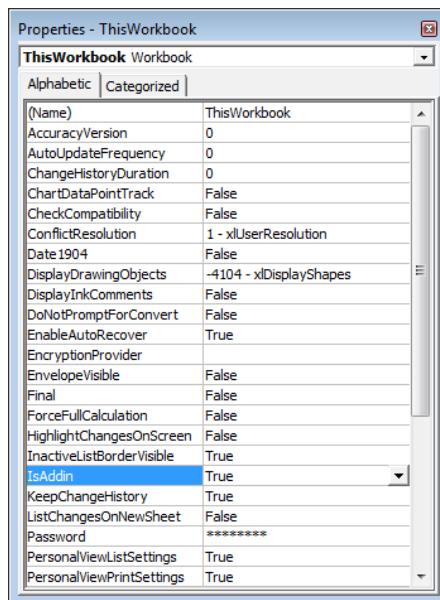
Tworzenie dodatku: lista kontrolna

Przed opublikowaniem utworzonego dodatku warto poświęcić kilka chwil, aby odpowiedzieć na podane niżej pytania.

- Czy dodatek został przetestowany dla wszystkich obsługiwanych platform i wersji Excela?
- Czy nadano projektowi VB opisową nazwę? Domyslnie wszystkie projekty VB otrzymują nazwę *VBProject*. Będzie lepiej, jeżeli zmienimy tę nazwę na bardziej opisową.
- Czy w dodatku przyjęliśmy jakieś założenia dotyczące struktury katalogów użytkownika lub nazw katalogów?
- Czy w oknie dialogowym *Dodatki* wyświetla się właściwa nazwa i opis dodatku?
- Czy funkcje VBA, które nie mają być używane w arkuszu, zostały zadeklarowane jako prywatne (*Private*)? Jeżeli nie, takie funkcje wyświetla się w oknie dialogowym *Wstawianie funkcji*.
- Czy pamiętałeś, aby usunąć wszystkie polecenia *Debug.Print* z kodu dodatku?
- Czy dla sprawdzenia, że dodatek nie zawiera błędów składniowych, pamiętałeś o jego skompilowaniu?
- Czy wzięliśmy pod uwagę ustawienia międzynarodowe?
- Czy dodatek został zoptymalizowany pod względem szybkości?Więcej informacji na ten temat znajdziesz w podrozdziale „Optymalizacja wydajności dodatków” w dalszej części tego rozdziału.

Jeżeli informacje z dodatku są zapisane w arkuszu, to aby przeglądać ten arkusz w Excelu, należy ustawić właściwość *IsAddIn* dodatku na wartość *False*. Można to wykonać w oknie Properties (patrz rysunek 19.6) po zaznaczeniu obiektu *ThisWorkbook*. Po wprowadzeniu zmian, a przed zapisaniem pliku należy ponownie ustawić właściwość *IsAddIn* na wartość *True*. Jeżeli pozostawimy właściwość *IsAddIn* ustawioną na wartość *False*, Excel nie pozwoli zapisać pliku z rozszerzeniem *XLAM*.

Rysunek 19.6.
Przekształcanie
dodateku w zwykły
skoroszyt



Porównanie plików XLAM i XLSM

Ten podrozdział rozpoczniemy od porównania pliku dodatku *XLAM* z odpowiadającym mu plikiem źródłowym *XLSM*. W dalszej części omówimy metody optymalizacji wydajności dodatków i opiszemy techniki zmniejszania rozmiaru pliku, dzięki którym dodatek będzie się ładował szybciej i zużywał mniej miejsca na dysku i w pamięci.

Dodatek utworzony na podstawie pliku źródłowego *XLSM* ma dokładnie ten sam rozmiar, co oryginał. Kod VBA zapisany w plikach *XLAM* nie jest skompresowany ani zoptymalizowany w jakikolwiek sposób, a zatem wykorzystanie dodatków nie powoduje wzrostu szybkości działania aplikacji.

Pliki XLAM — przynależność do kolekcji z poziomu VBA

Dodatki należą do kolekcji *AddIns*, ale nie są oficjalnymi elementami kolekcji *Workbooks*. Do dodatku można się jednak odwołać za pomocą metody *Workbooks* obiektu *Application*, jeżeli jako indeks podamy nazwę pliku. Poniższa instrukcja tworzy zmienną obiektową reprezentującą dodatek o nazwie *Myaddin.xlam*:

```
Dim TestAddin As Workbook  
Set TestAddin = Workbooks("Myaddin.xlam")
```

Do dodatków nie można się odwoływać za pomocą numerów indeksów w kolekcji *Workbooks*. Jeżeli wykorzystamy poniższy kod do przetworzenia w pętli kolekcji *Workbooks*, skoroszyt *Myaddin.xlam* nie zostanie wyświetlony:

```
Dim w As Workbook  
For Each w in Application.Workbooks  
    MsgBox w.Name  
Next w
```

Z kolei poniższa pętla *For ... Next* spowoduje wyświetlenie dodatku *Myaddin.xlam* przy założeniu, że dodatek ten wyświetla się w oknie dialogowym *Dodatki Excela*:

```
Dim a as Addin  
For Each a in Application.AddIns  
    MsgBox a.Name  
Next a
```

Widoczność plików XLSM i XLAM

Zwykłe skoroszyty wyświetlane są w jednym lub kilku oknach. Na przykład poniższa instrukcja wyświetla liczbę okien aktywnego skoroszytu:

```
MsgBox ActiveWorkbook.Windows.Count
```

Widoczność okien skoroszytu można zmieniać za pomocą polecenia *Ukryj* znajdującego się w grupie opcji *Okno* na karcie *Widok* lub zmieniając wartość właściwości *Visible* przy użyciu kodu VBA. Poniższy kod ukrywa wszystkie okna aktywnego skoroszytu:

```
Dim Win As Window  
For Each Win In ActiveWorkbook.Windows  
    Win.Visible = False  
Next Win
```

Pliki dodatków nigdy nie są widoczne i oficjalnie nie posiadają okien, pomimo że zawierają niewidoczne arkusze. W efekcie dodatki nie pojawiają się na liście okien po wybraniu polecenia *Przelacz okna* (karta *WIDOK*, grupa opcji *Okno*). Nawet jeżeli skoroszyt *Myaddin.xlam* jest otwarty, poniższa instrukcja zwróci zawsze wartość 0:

```
MsgBox Workbooks("Myaddin.xlam").Windows.Count
```

Arkusze i wykresy w plikach XLSM i XLAM

Pliki dodatków, podobnie jak normalne pliki skoroszytów, mogą zawierać dowolną liczbę arkuszy lub wykresów. Jednak, jak już wcześniej wspomniano, aby plik *XLSM* można było przekształcić na dodatek, musi zawierać co najmniej jeden arkusz. W większości przypadków, ten arkusz pozostanie pusty.

Jeżeli dodatek jest otwarty, można uzyskać dostęp do jego arkuszy z poziomu kodu VBA w taki sam sposób, jak do arkuszy zwykłego skoroszytu. Ponieważ jednak pliki dodatków nie należą do kolekcji *Workbooks*, do dodatku można odwoływać się wyłącznie na podstawie nazwy, a nie na podstawie numeru indeksu. Zaprezentowany poniżej przykład wyświetla wartość z komórki A1 pierwszego arkusza w pliku *Myaddin.xlam* (przy założeniu, że jest otwarty):

```
MsgBox Workbooks("Myaddin.xlam").Worksheets(1).Range("A1").Value
```

Jeżeli dodatek zawiera arkusz, którego zawartość chcemy wyświetlić, możemy go skopiować do otwartego skoroszytu albo utworzyć nowy skoroszyt i zapisać w nim arkusz.

Na przykład poniższy kod kopiuje pierwszy arkusz z dodatku i umieszcza go w aktywnym skoroszycie (jako ostatni arkusz):

```
Sub CopySheetFromAddin()  
    Dim AddinSheet As Worksheet  
    Dim NumSheets As Long  
    Set AddinSheet = Workbooks("Myaddin.xlam").Sheets(1)  
    NumSheets = ActiveWorkbook.Sheets.Count  
    AddinSheet.Copy After:=ActiveWorkbook.Sheets(NumSheets)  
End Sub
```

Zwrót uwagi na fakt, że powyższa procedura działa poprawnie nawet wtedy, kiedy projekt VBA dodatku jest chroniony hasłem.

Utworzenie nowego skoroszytu na bazie arkusza wewnątrz dodatku jest jeszcze łatwiejsze:

```
Sub CreateNewWorkbook()  
    Workbooks("Myaddin.xlam").Sheets(1).Copy  
End Sub
```



W poprzednich przykładach przyjęliśmy założenie, że kod znajduje się w pliku, który nie jest dodatkiem. W kodzie VBA dodatku, aby odwołać się do własnych arkuszy i zakresów, zawsze należy wykorzystywać obiekt `ThisWorkbook`. Na przykład instrukcja umieszczona w kodzie modułu VBA dodatku, wyświetlająca wartość z komórki A1 arkusza Arkusz1 należącego do tego dodatku, powinna mieć następującą postać:

```
MsgBox ThisWorkbook.Sheets("Arkusz1").Range("A1").Value
```

Dostęp do procedur VBA w dodatku

Korzystanie z procedur VBA w dodatku nieco różni się od korzystania z procedur VBA w zwykłym skoroszycie *XLSM*. Gdy zostanie wybrane polecenie *Makra* (karta *WIDOK*, grupa opcji *Makra*), w oknie dialogowym *Makro* nazwy makr znajdujących się w otwartym dodatku nie zostaną wyświetcone. Wygląda to niemal tak, jakby Excel zabraniał dostępu do tych procedur.



Jeżeli znasz nazwę procedury w dodatku, możesz bezpośrednio wprowadzić ją w oknie dialogowym *Makro* i kliknąć *Uruchom* w celu jej wykonania. Takie procedury Sub muszą znajdować się w ogólnym module kodu VBA, a nie w module kodu obiektu.

Ponieważ procedury zawarte w dodatku nie są wymienione w oknie dialogowym *Makro*, należy wykorzystać inne sposoby, aby uzyskać do nich dostęp. Mogą to być metody bezpośredni (takie, jak klawisze skrótów, nowe polecenia na Wstążce lub nowe polecenia w menu podręcznym) oraz pośrednie (np. procedury obsługi zdarzeń). Dobrze do tego celu nadaje się procedura *OnTime*, za pomocą której można uruchomić kod o określonej porze dnia.

Do wykonania procedury zapisanej w dodatku można wykorzystać metodę *Run* obiektu *Application*:

```
Application.Run "Myaddin.xlam!DisplayNames"
```

Jeszcze inny sposób polega na wykorzystaniu polecenia *Tools/References* w edytorze Visual Basic w celu zdefiniowania odwołania do dodatku. Po wykonaniu tej czynności można wywoływać procedury dodatku w kodzie VBA bez podawania nazwy pliku. Nie trzeba nawet wykorzystywać metody *Run*. Można wywołać procedurę bezpośrednio, jeżeli tylko nie została zadeklarowana jako prywatna. Poniższa instrukcja spowoduje wykonanie procedury *DisplayNames*, o ile zdefiniowano odwołanie do dodatku:

```
Call DisplayNames
```



Nawet po zdefiniowaniu odwołania do dodatku nazwy jego makr nie będą wyświetlane w oknie dialogowym *Makro*.

Funkcje zdefiniowane w dodatku działają identycznie, jak funkcje zdefiniowane w skoroszycie *XLSM*. Dostęp do nich jest prosty, gdyż ich nazwy wyświetlają się w oknie dialogowym *Wstawianie funkcji* w kategorii *Użytkownika* (domyślnie). Wyjątek stanowi sytuacja, kiedy funkcję zadeklarowano ze słowem kluczowym *Private*. W takim przypadku nazwy funkcji nie wyświetlają się w oknie dialogowym *Wstawianie funkcji*. Z tego powodu należy deklarować jako prywatne te funkcje, które są wykorzystywane wyłącznie przez inne procedury VBA, a nie są przeznaczone do wykorzystania w formułach arkusza.

Jak wspominaliśmy wcześniej, funkcje arkuszowe zapisane w dodatku można wykorzystywać bez podawania nazwy skoroszytu. Aby na przykład z arkusza należącego do innego skoroszytu zaadresować funkcję użytkownika o nazwie MOVAVG, zapisaną w skoroszycie *Newfuncs.xlsm*, trzeba użyć następującego zapisu:

```
=Newfuncs.xlsm!MOVAVG(A1:A50)
```

Natomiast jeżeli funkcję zapisano w pliku dodatku, który jest otwarty, można pominąć odwołanie do pliku i skorzystać z instrukcji o następującej składni:

```
=MOVAVG(A1:A50)
```

Pamiętaj, że skoroszyt wykorzystujący funkcje zdefiniowane w dodatku będzie przechowywał łącze do tego dodatku. Z tego powodu dodatek musi być dostępny za każdym razem, kiedy będziesz pracował z takim skoroszytem.

Podglądanie zabezpieczonego dodatku

Okno dialogowe *Makro* nie wyświetla nazw procedur zawartych w dodatku. Co zatem zrobić, jeżeli chcemy uruchomić taką procedurę, a dodatek jest zabezpieczony i nie można przeglądać kodu, aby poznać nazwy procedur? Należy wykorzystać przeglądarkę obiektów!

Aby zilustrować zagadnienie, zainstalujemy dodatek *Narzędzia do waluty euro*. Jest to zabezpieczony dodatek rozpowszechniany razem z Exceliem, a zatem nie można bezpośrednio przeglądać jego kodu. Po zainstalowaniu dodatek ten tworzy na karcie *Formuły* nową grupę, o nazwie *Solutions*. Po naciśnięciu przycisku *Euro Conversion* na ekranie pojawia się okno dialogowe *Euro Conversion*, za pomocą którego możesz dokonać konwersji wartości w zakresie komórek zawierających dane walutowe.

Aby określić nazwę procedury, która wyświetla to okno dialogowe, powinieneś wykonać następujące polecenia:

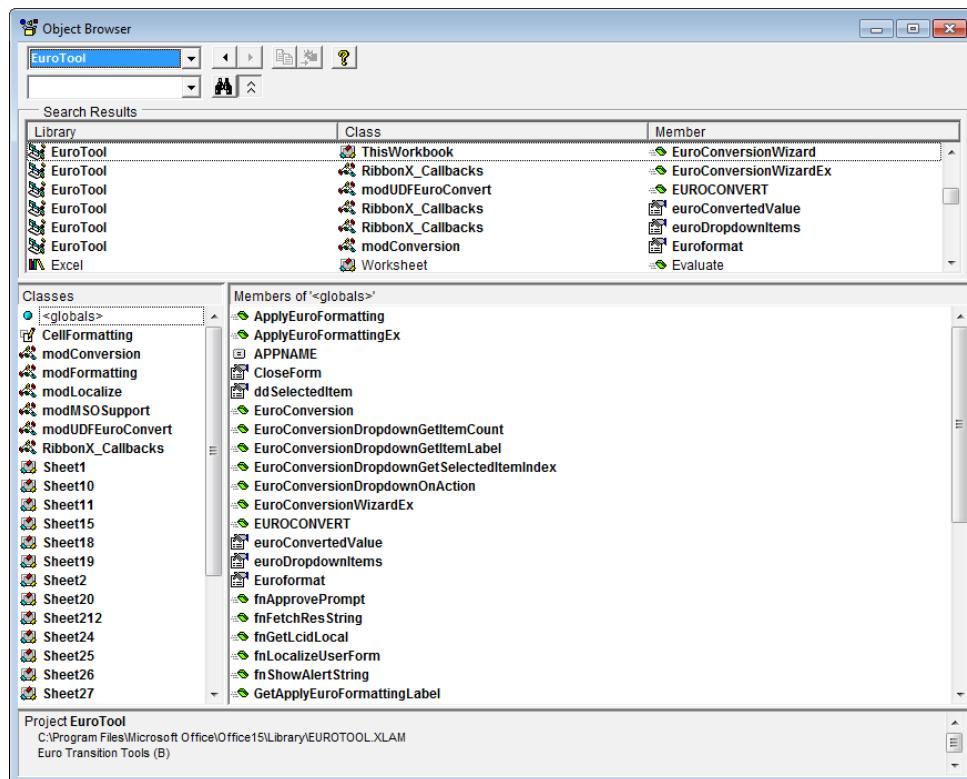
1. Uaktywnij edytor Visual Basic i w oknie *Project* zaznacz projekt *EuroTool.XLAM*.
2. Wciśnij *F2*, aby uaktywnić przeglądarkę obiektów (*Object Browser*).
3. Z listy rozwijanej bibliotek wybierz opcję *EuroTool*. Na ekranie zostanie wyświetlona lista klas dodatku *EuroTool.XLAM*, jak pokazano na rysunku poniżej.
4. Zaznaczaj różne opcje na liście *Classes*, aby przejrzeć obiekty składowe zawarte w tych klasach.

W naszym przykładzie możemy stwierdzić, że dodatek *Narzędzie do waluty euro* składa się z całkiem sporej liczby arkuszy. Excel pozwala na kopowanie arkuszy z chronionych dodatków, stąd jeżeli chcesz rzucić okiem na któryś z arkuszy dodatku, przejdź do okna *Immediate* i skopiuj arkusz do nowego skoroszytu np. przy użyciu następującego polecenia:

```
Workbooks("eurotool.xlam").Sheets(1).Copy
```

Jeżeli chcesz sprawdzić wszystkie arkusze, możesz wykonać następujące polecenie, które dokonuje konwersji dodatku do normalnego skoroszytu:

```
Workbooks("eurotool.xlam").IsAddin = False
```



Na rysunku zamieszczonym poniżej przedstawiamy wygląd fragmentu arkusza skopiowany z dodatku *EuroTool.XLAM*. Arkusz ten, podobnie jak pozostałe, zawiera informacje wykorzystywane do wyświetlania różnych wersji językowych dodatku.

Jak widać, zagadnienie robi się interesujące, ale mimo to nadal nie pomogło nam jeszcze odnaleźć nazwy poszukiwanej procedury.

Nasz dodatek posiada wiele procedur. Metodą prób i błędów sprawdziłem kilka obiecujących procedur, ale żadna z nich nie spowodowała wyświetlenia okna dialogowego dodatku. Następnie sprawdziłem listę procedur przechowywanych w module *ThisWorkbook* i zauważylem tam procedurę o nazwie *EuroConversionWizard*. Spróbowałem ją uruchomić, ale próba zakończyła się wyświetleniem komunikatu o błędzie. Następnie spróbowałem wykonać polecenie przedstawione poniżej:

```
Application.Run "eurotool.xlam!ThisWorkbook.EuroConversionWizard"
```

Sukces! Wykonanie powyższego polecenia spowodowało wyświetlenie na ekranie okna dialogowego dodatku *Narzędzia do waluty euro*.

Mając takie informacje, możesz już napisać swój własny kod VBA uruchamiający konwerter waluty — zakładając oczywiście, że znajdziesz odpowiedni powód, aby tak postąpić.

A	B
1 frmEuro	Caption
2 Accelerator	
3	Μετατροπή σε ευρώ
4	Δεδουλένα για τη μετατροπή σε ευρώ
5 Π	Περιοχή πρέλευσης:
6 ε	Περιοχή προορισμού:
7	Μετατροπή νομιμοποίης μονάδας
8 A	Από:
9 Σ	Σε:
10 M	Μορφή εξόδου:
11 Γ	Για προχωρημένους
12	OK
13	Άκυρο
14	<unused>
15	<unused>
16	<unused>
17	<unused>
18	<unused>
19	<unused>
20	<unused>
21	<unused>
22	<unused>
23	<unused>
24 frmAdvanced	Caption
25 Accelerator	
26	Επιλογές ευρώ για προχωρημένους
27	Επιλογές τύπου
28 M	Μετατροπή μόνο σε τιμές
29 E	Ερώτηση για μετατροπή των τύπων
<input type="button" value="◀"/> <input type="button" value="▶"/> 1028 1030 1031 1032 1033 1035 1036 1040 1041 ... <input type="button" value="⊕"/> : <input type="button" value="◀"/> <input type="button" value="▶"/>	

Przetwarzanie dodatków za pomocą kodu VBA

W tym podrozdziale zaprezentujemy informacje potrzebne do napisania procedur VBA wykonujących działania na dodatkach.

Kolekcja AddIns zawiera wszystkie dodatki, o których Excel posiada aktualnie jakieś informacje. Mogą to być zarówno dodatki zainstalowane, jak i niezainstalowane. Składowe kolekcji AddIns można wyświetlić w oknie dialogowym *Dodatki*. Zainstalowane są te dodatki, dla których zaznaczono pola wyboru.



Już w Excelu 2010 pojawiła się nowa kolekcja o nazwie AddIns2. Jest to taka sama kolekcja jak AddIns, ale w jej skład wchodzą również dodatki, które zostały otwarte za pomocą polecenia *Otwórz*. W poprzednich wersjach Excela dostęp do tak otwartych dodatków wymagał użycia odpowiedniego makra XLM.

Dodawanie nowych elementów do kolekcji AddIns

Pliki dodatków składające się na kolekcję AddIns mogą znajdować się w dowolnym miejscu. Excel utrzymuje listę niektórych spośród tych plików wraz z ich lokalizacją w rejestrze systemu Windows. W przypadku Excela 2013 lista jest przechowywana w następującym kluczu:

HKEY_CURRENT_USER\Software\Microsoft\Office\15.0\Excel\Add-in Manager

Do przeglądania tego klucza rejestru można wykorzystać edytor rejestru (*regedit.exe*). Pamiętaj, że standardowe dodatki dostarczane wraz z Exceliem nie pojawiają się w tym kluczu rejestru. Ponadto pliki dodatków zapisane w katalogu podanym poniżej również pojawiają się na liście okna dialogowego *Dodatki*, ale nie będzie ich w rejestrze:

```
C:\Program Files (x86)\Microsoft Office\Office15\Library
```

Nowy obiekt AddIn można dodać do kolekcji AddIns ręcznie lub za pomocą kodu VBA. Aby ręcznie umieścić dodatek w kolekcji, przywołaj na ekran okno dialogowe *Dodatki*, kliknij przycisk *Przeglądaj* i odszukaj dodatek.

Aby dołączyć nowy element do kolekcji AddIns za pomocą kodu VBA, należy skorzystać z metody Add kolekcji. Oto przykład:

```
Application.AddIns.Add "c:\pliki\nowyddodatek.xlam"
```

Po wykonaniu tej instrukcji w kolekcji AddIns zostanie umieszczony nowy element, który wyświetli się także na liście w oknie dialogowym *Dodatki*. Jeżeli dodatek został już wcześniej dodany do kolekcji, nic się nie stanie i nie zostanie wygenerowany błąd.

Jeżeli dodatek wprowadzany do kolekcji znajduje się na wymiennym nośniku danych (np. dysku CD-ROM), metodę Add można także wykorzystać do skopiowania go do katalogu bibliotek Excela. Wykonanie poniższej instrukcji spowoduje skopiowanie pliku *Myaddin.xlam* z dysku E i dodanie go do kolekcji AddIns. Drugi argument (w tym przypadku ma wartość True) decyduje o tym, czy dodatek będzie skopiowany. Jeżeli dodatek znajduje się na dysku twardym, drugi argument można zignorować.

```
Application.AddIns.Add "e:\Myaddin.xlam", True
```



Dołączenie nowego skoroszytu do kolekcji AddIns nie powoduje jego zainstalowania. Aby zainstalować taki dodatek powinieneś ustawić jego właściwość Installed na wartość True.



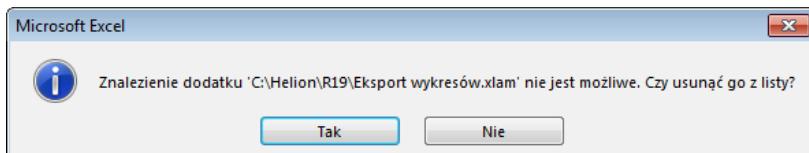
Rejestr Windows będzie uaktualniony tylko wtedy, kiedy Excel zostanie poprawnie zamknięty. Jeżeli działanie Excela zostanie zakończone na przykład na skutek awarii, nazwy wprowadzonych dodatków nie zostaną poprawnie zapisane w rejestrze, a zatem po ponownym uruchomieniu Excela nie będą częścią kolekcji AddIns.

Usuwanie elementów z kolekcji AddIns

Co ciekawe, nie ma bezpośredniej możliwości usunięcia dodatku z kolekcji AddIns. Kolekcja AddIns nie posiada metody Delete lub Remove. Jednym ze sposobów jego usunięcia z okna dialogowego *Dodatki* jest edycja rejestru Windows (za pomocą programu *regedit.exe*). Po usunięciu nazwy z rejestru dodatek nie wyświetli się w oknie dialogowym *Dodatki* przy następnym uruchomieniu Excela. Metoda ta nie jest skuteczna dla wszystkich plików dodatków, a tylko dla tych, których nazwy są zapisane w rejestrze.

Innym sposobem usunięcia dodatku z kolekcji AddIns jest usunięcie, przeniesienie lub zmiana nazwy pliku *XLAM* (lub *XLA*) dodatku. W takim przypadku przy następnej próbie zainstalowania lub odinstalowania dodatku wyświetli się ostrzeżenie podobne do tego, które pokazano na rysunku 19.7.

Rysunek 19.7.
Jeden ze sposobów
usunięcia elementu
kolekcji AddIns



Właściwości obiektu AddIn

Obiekt AddIn to pojedynczy element kolekcji AddIns. Aby wyświetlić nazwę pliku pierwszego elementu kolekcji AddIns, można wykorzystać następującą instrukcję:

```
Msgbox AddIns(1).Name
```

Obiekt AddIn ma 14 właściwości, o których można dowiedzieć się więcej z systemu pomocy. Pięć z nich to właściwości ukryte. Niektóre pojęcia dotyczące obiektu AddIn mogą być nieco mylące, a zatem w kolejnych podpunktach omówimy kilka ważniejszych właściwości.

Właściwość Name obiektu AddIn

Ta właściwość przechowuje nazwę pliku dodatku. Jest to właściwość tylko do odczytu, a zatem nie można zmienić nazwy pliku poprzez modyfikację wartości tej właściwości.

Właściwość Path obiektu AddIn

Ta właściwość przechowuje informacje o napędzie i ścieżce dostępu do pliku dodatku. Nie zawiera ostatniego ukośnika i nazwy pliku.

Właściwość FullName obiektu AddIn

Ta właściwość przechowuje informacje o napędzie, ścieżce dostępu oraz nazwie pliku dodatku. Właściwość FullName jest nieco nadmiarowa, ponieważ informacje te można odczytać także z właściwości Name i Path. Poniższe dwie instrukcje powodują wyświetlenie dokładnie takich samych komunikatów:

```
MsgBox AddIns(1).Path & "\" & AddIns(1).Name  
MsgBox AddIns(1).FullName
```

Właściwość Title obiektu AddIn

Jest to ukryta właściwość, która przechowuje opisową nazwę dodatku. Jej wartość jest tekstem, który wyświetla się w oknie dialogowym *Dodatki*. Jest to właściwość tylko do odczytu. Jedynym sposobem dodania lub modyfikacji właściwości Title jest modyfikacja właściwości dokumentu (aby przywołać na ekran panel właściwości dokumentu, przejdź na kartę *DEVELOPER* i naciśnij przycisk *Panel dokumentu*, znajdujący się w grupie poleceń *Modyfikowanie*). Tego polecenia powinieneś użyć dla pliku *XLSM* przed przekształceniem go na dodatek. Innym rozwiązaniem jest kliknięcie prawym przyciskiem myszy pliku dodatku w oknie Eksploratora Windows i wybranie z menu podręcznego

polecenia *Właściwości*. Następnie powinieneś przejść na kartę *Podsumowanie* i tam w wybranych polach dokonać odpowiednich wpisów. Jeżeli dany plik jest otwarty w Excelu, nie możesz skorzystać z tej metody.

Zazwyczaj dostęp do elementu kolekcji uzyskuje się poprzez jej właściwość *Name*. Kolekcja *AddIns* jest pod tym względem nieco inna, w tym celu wykorzystuje właściwość *Title*. W poniższym przykładzie pokazano sposób wyświetlenia nazwy pliku dla dodatku *Analysis ToolPak* (tzn. *analys32.xls*), którego wartość właściwości *Title* to "Analysis ToolPak":

```
Sub ShowName()
    MsgBox AddIns("Analysis ToolPak").Name
End Sub
```

Można oczywiście odwołać się do określonego dodatku za pomocą numeru indeksu, jednak w olbrzymiej większości przypadków będziemy korzystać z właściwości *Name*.

Właściwość Comments obiektu AddIn

W tej ukrytej właściwości jest zapisany tekst, jaki wyświetla się w oknie dialogowym *Dodatki* po zaznaczeniu określonego dodatku. Właściwość *Comments* jest tylko do odczytu. Jedynym sposobem jej modyfikacji jest wykorzystanie okna dialogowego *Właściwości* przed przekształceniem skoroszytu na dodatek. Komentarze mogą mieć długość do 255 znaków, ale w oknie dialogowym wyświetla się tylko około 100 znaków.

Jeżeli kod Twojej procedury dokona próby odczytania właściwości *Comments* dodatku, który nie posiada zdefiniowanego komentarza, to taka operacja zakończy się błędem.

Właściwość Installed obiektu AddIn

Właściwość *Installed* ma wartość *True*, jeżeli dodatek jest aktualnie zainstalowany — tzn. zaznaczony w oknie dialogowym *Dodatki*. Ustawienie właściwości *Installed* na wartość *True* powoduje otwarcie dodatku. Ustawienie jej na wartość *False* powoduje zamknięcie dodatku. Poniżej pokazano przykład sposobu instalacji (a tym samym otwarcia) dodatku *Analysis ToolPak* za pomocą kodu VBA:

```
Sub InstallATP()
    AddIns("Analysis ToolPak").Installed = True
End Sub
```

Gdy procedura zostanie wykonana, w oknie dialogowym *Dodatki* pole wyboru obok nazwy dodatku *Analysis ToolPak* będzie zaznaczone. Jeżeli dodatek zainstalowano wcześniej, ustawienie właściwości *Installed* na wartość *True* nie przyniesie żadnego efektu. Aby usunąć dodatek (odinstalować go), wystarczy ustawić właściwość *Installed* na wartość *False*.



Jeżeli dodatek został otwarty za pomocą polecenia *PLIK/Otwórz*, nie jest on uznawany za otwarty, stąd jego właściwość *Installed* ma w takiej sytuacji wartość *False*. Dodatek jest prawidłowo zainstalowany tylko wtedy, kiedy jego nazwa pojawia się w oknie dialogowym *Dodatki* oraz pole wyboru obok nazwy jest zaznaczone.

Procedura ListAllAddIns (jej kod zamieszczono poniżej) tworzy tabelę, w której zamieszczane są informacje na temat wszystkich elementów kolekcji AddIns, i wyświetla następujące właściwości: Name, Title, Installed, Comments oraz Path.

```
Sub ListAllAddins()
    Dim ai As AddIn
    Dim Row As Long
    Dim Table1 As ListObject
    Cells.Clear
    Range("A1:E1") = Array("Nazwa", "Tytuł", "Zainstalowany", _
                           "Komentarz", "Ścieżka")
    Row = 2
    On Error Resume Next
    For Each ai In AddIns
        Cells(Row, 1) = ai.Name
        Cells(Row, 2) = ai.Title
        Cells(Row, 3) = ai.Installed
        Cells(Row, 4) = ai.Comments
        Cells(Row, 5) = ai.Path
        Row = Row + 1
    Next ai
    On Error GoTo 0
    Range("A1").Select
    ActiveSheet.ListObjects.Add
    ActiveSheet.ListObjects(1).TableStyle = _
        "TableStyleMedium2"
End Sub
```

Na rysunku 19.8 przedstawiono wyniki działania tej procedury. Jeżeli dokonasz modyfikacji kodu tak, aby użyć kolekcji AddIns2, tabela będzie zawierała również informacje o dodatkach, które zostały otwarte za pomocą polecenia *PLIK/Otwórz* (o ile takie dodatki są otwarte). Kolekcja AddIns2 jest dostępna wyłącznie w Excelu 2010 i nowszych wersjach.

A	B	C	D	E
Nazwa	Tytuł	Zainstalowany	Komentarze	Ścieżka
ANALYS32.XLL	Analysis ToolPak	FAŁSZ		C:\Program Files\Microsoft Office\Office15\Library\Analysis
ATPVBAEN.XLAM	Analysis ToolPak - VBA	FAŁSZ		C:\Program Files\Microsoft Office\Office15\Library\Analysis
SOLVER.XLAM	Dodatek Solver	FAŁSZ		C:\Program Files\Microsoft Office\Office15\Library\SOLVER
Eksport wykresów.xlam	Eksport wykresów	PRAWDA	Eksportuje wykresy do plików w formacie GIF, PNG, JPG lub TIFF. Enhanced Alternative to Excel's Data Form (Version 3.0a) Copyright 1997-2008, J-Walk & Associates	C:\Helion\R19
dataform3.xls	Enhanced Data Form v3a	FAŁSZ	Enhanced Alternative to Excel's Data Form (Version 3.0b) Copyright 1997-2010, J-Walk & Associates	C:\Helion\R13\dataform
dataform3.xlam	Enhanced Data Form v3b	FAŁSZ	Associates	C:\Helion\R13\dataform
EUROTOOL.XLAM	Narzędzia do waluty euro	FAŁSZ		C:\Program Files\Microsoft Office\Office15\Library
Operacje tekstowe.xlam	Operacje tekstowe	FAŁSZ	Dodatek zawierający szereg narzędzi do przetwarzania tekstu. © 2007 JWALK & Associates.	C:\Helion\R14
pup7.xlam	Power Utility Pak v7.2	FAŁSZ	All Rights Reserved. Add-In Tools For Excel 2007/2010 - TRIAL Copyright 1999-2013 J-Walk & Associates	C:\Program Files\pup7

Rysunek 19.8. Tabela z informacjami o wszystkich elementach kolekcji AddIns



W sieci



Uwaga

Skoroszyt z tym przykładem (*Informacje o dodatkach.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Aby przekonać się, czy określony skoroszyt jest dodatkiem, wystarczy sprawdzić wartość właściwości `IsAddIn` tego skoroszytu. Nie jest to właściwość tylko do odczytu, a zatem ustawiając właściwość `IsAddIn` na wartość `True`, można także przekształcić skoroszyt na dodatek.

Twierdzenie odwrotne jest również prawdziwe — poprzez ustawienie właściwości `IsAddIn` na wartość `False` można przekształcić dodatek na skoroszyt. Po wykonaniu tej czynności arkusze dodatku będą widoczne w Excelu nawet wtedy, kiedy projekt VB dodatku jest zabezpieczony. Korzystając z tej techniki przekonałem się, że większość okien dialogowych dodatku *Solver.xlam* to nie formularze *UserForm*, ale stare, tradycyjne arkusze dialogowe Excela 5/95. Oprócz tego dowiedziałem się, że w dodatku *Solver.xlam* zdefiniowanych jest ponad 500 nazw zakresów.

Korzystanie z dodatku jak ze skoroszytu

Jak wspominaliśmy już wcześniej, istnieją dwa sposoby otwierania dodatków: za pomocą okna dialogowego *Dodatki* lub za pomocą polecenia *PLIK/Otwórz*. Pierwsza metoda jest preferowana z następującego powodu: otwarcie dodatku za pomocą polecenia *PLIK/Otwórz* nie powoduje ustawienia jego właściwości `Installed` na wartość `True`. Z tego powodu nie można zamknąć pliku za pomocą okna dialogowego *Dodatki*. Jedynym sposobem zamknięcia takiego dodatku jest użycie następującej instrukcji VBA:

```
Workbooks("Myaddin.xlam").Close
```



Ostrzeżenie

Zastosowanie metody `Close` dla zainstalowanego dodatku powoduje usunięcie go z pamięci, ale nie ustawia właściwości `Installed` na wartość `False`. Z tego powodu dodatek będzie w dalszym ciągu zaznaczony w oknie dialogowym *Dodatki*, tak jakby był zainstalowany. Może to być bardzo mylące. Właściwym sposobem usunięcia zainstalowanego dodatku jest ustawienie jego właściwości `Installed` na wartość `False`.

Jak się już zapewne sam zdążyłeś zorientować, mechanizm dodatków w Excelu jest nieco dziwaczny, a co gorsza, nie był aktualizowany i poprawiany od wielu lat (z wyjątkiem dodania kolekcji `AddIns2`). Z tego względu powinieneś zwracać szczególną uwagę na zagadnienia dotyczące instalowania i odinstalowywania dodatków.

Zdarzenia związane z obiektami `AddIn`

Z obiektem `AddIn` są związane dwa zdarzenia: `AddInInstall` (generowane w momencie instalacji) oraz `AddInUninstall` (generowane w momencie odinstalowania). Procedury obsługi tych zdarzeń należy umieścić w module kodu obiektu `ThisWorkbook`.

Poniższy kod wyświetla komunikat po zainstalowaniu dodatku:

```
Private Sub Workbook_AddInInstall()
    MsgBox "Dodatek " & ThisWorkbook.Name & " został zainstalowany."
End Sub
```



Nie wolno mylić zdarzenia AddInInstall ze zdarzeniem Open. Zdarzenie AddInInstall zachodzi tylko wtedy, gdy dodatek jest instalowany po raz pierwszy, a nie za każdym razem, kiedy jest otwierany. Jeżeli kod ma być wykonywany przy każdej operacji otwierania dodatku, należy wykorzystać procedurę Workbook_Open.



Więcej szczegółowych informacji na temat zdarzeń znajdziesz w rozdziale 17.

Optymalizacja wydajności dodatków

Jeżeli poprosimy tuzin programistów o zautomatyzowanie określonego zadania, prawdopodobnie otrzymamy tuzin różnych rozwiązań, a niektóre z nich najprawdopodobniej będą działały nieco lepiej od pozostałych.

Poniżej podano kilka wskazówek, których wykorzystanie pozwoli zoptymalizować kod pod względem szybkości działania. Pamiętaj, że te wskazówki odnoszą się ogólnie do kodu VBA, a nie tylko programowania dodatków.

- Podczas zapisywania danych do arkusza lub wykonywania innych działań, które powodują zmianę wyświetlanych informacji, powinieneś ustawać właściwość Application.ScreenUpdating na wartość False.
- Zawsze powinieneś deklarować typ danych i jeżeli to możliwe, unikać typu Variant. Aby wymusić konieczność deklarowania wszystkich zmiennych, wprowadźmy na początku każdego modułu instrukcję Option Explicit.
- Aby uniknąć długich odwołań do obiektów, powinieneś tworzyć zmienne obiektowe. Jeżeli na przykład pracujesz z obiektem Series dla wykresu, utwórz zmienną obiektową za pomocą następującego kodu:

```
Dim S1 As Series  
Set S1 = ActiveWorkbook.Sheets(1).ChartObjects(1).  
    Chart.SeriesCollection(1)
```

- Wszędzie, gdzie to możliwe, powinieneś deklarować zmienne obiektowe podając ich konkretny typ. Należy unikać deklarowania ich jako zmiennych ogólnego typu Object.
- Aby ustawić wiele właściwości dla jednego obiektu lub wywołać wiele metod, powinieneś używać konstrukcji With ... End With.
- Zawsze powinieneś usuwać nadmiarowy kod. Jest to szczególnie ważne, jeżeli do utworzenia procedur używałeś rejestratora makr.
- Jeżeli to możliwe, operacje na danych powinieneś wykonywać, wykorzystując tablice VBA, a nie zakresy arkusza. Operacje odczytywania i zapisywania danych do arkusza trwają znacznie dłużej od działań w pamięci. Nie jest to jednak uniwersalna zasada. Dla uzyskania najlepszych wyników warto przetestować obie opcje.
- Jeżeli kod Twojej procedury zapisuje dużo danych do arkuszy, powinieneś rozważyć zmianę trybu przeliczania arkusza na ręczny. Wykonanie takiej prostej zmiany może w znaczący sposób przyspieszyć działanie programu. Poniżej przedstawiamy

polecenie, którego wykonanie zmienia tryb przeliczania arkusza z automatycznego na ręczny:

```
Application.Calculation = xlCalculationManual
```

- Powinieneś unikać łączenia formantów *UserForm* z komórkami arkuszy. W takim przypadku może nastąpić przeliczanie arkusza za każdym razem, kiedy użytkownik zmieni formant *UserForm*.
- Przed utworzeniem dodatku powinieneś pamiętać o skompilowaniu kodu. Może to spowodować zwiększenie rozmiaru pliku, ale wyeliminuje konieczność komplikacji kodu przed wykonaniem procedury.

Problemy z dodatkami

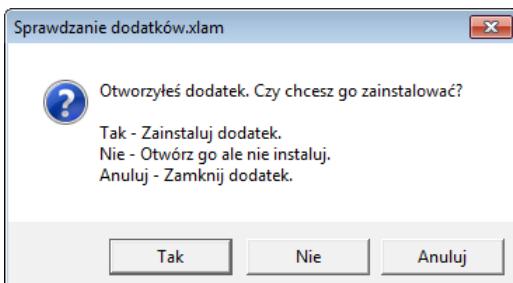
Dodatki to świetny mechanizm, ale jak to w życiu: nie ma nic za darmo. Dodatki stawiają przed programistą określone *wyzwania*. W tym podrozdziale opisano kilka problemów, które programista musi rozwiązać, jeżeli chce tworzyć dodatki dla szerokiego grona użytkowników.

Zapewnienie, że dodatek został zainstalowany

W niektórych przypadkach musisz się upewniać, że dodatek został poprawnie zainstalowany — tzn. otwarty za pomocą okna dialogowego *Dodatki*, a nie za pomocą polecenia *PLIK/Otwórz*. W tym podrozdziale omówiono technikę, która pozwala na sprawdzenie, w jaki sposób dodatek został otwarty, i daje użytkownikowi możliwość zainstalowania dodatku, jeżeli nie został poprawnie zainstalowany.

Jeżeli dodatek nie zostanie poprawnie zainstalowany, procedura wyświetla na ekranie odpowiedni komunikat (patrz rysunek 19.9). Aby zainstalować dodatek, naciśnij przycisk *Tak*. Jeżeli naciśniesz przycisk *Nie*, plik zostanie otwarty, ale nie będzie zainstalowany. Naciśnięcie przycisku *Anuluj* zamknie plik.

Rysunek 19.9.
Próba niewłaściwego
otwarcia dodatku
powoduje wyświetlenie
takiego komunikatu



Poniżej przedstawiamy moduł kodu dla obiektu *ThisWorkbook* z tego przykładu. Procedura wykorzystuje fakt, że dla skoroszytu zdarzenie *AddInInstall* występuje przed zdaniem *Open*.

```
Dim InstalledProperly As Boolean

Private Sub Workbook_AaddinInstall()
    InstalledProperly = True
End Sub

Private Sub Workbook_Open()
    Dim ai As AddIn, NewAi As AddIn
    Dim M As String
    Dim Ans As Integer
    'Czy dodatek został zainstalowany za pomocą okna Dodatki?
    If InstalledProperly Then Exit Sub

    'Czy należy do kolekcji AddIns?
    For Each ai In AddIns
        If ai.Name = ThisWorkbook.Name Then
            If ai.Installed Then
                MsgBox "Dodatek został poprawnie zainstalowany!". _
                    vbInformation, ThisWorkbook.Name
                Exit Sub
            End If
        End If
    Next ai

    'Dodatek nie należy do kolekcji AddIns, wyświetlamy komunikat
    M = "Właśnie otwarteś plik dodatku. Czy chcesz go zainstalować?"
    M = M & vbCrLf
    M = M & vbCrLf & "Tak - zainstaluj dodatek. "
    M = M & vbCrLf & "Nie - otwórz dodatek, ale go nie instaluj."
    M = M & vbCrLf & "Anuluj - Zamknij dodatek."
    Ans = MsgBox(M, vbQuestion + vbYesNoCancel, _
        ThisWorkbook.Name)
    Select Case Ans
        Case vbYes
            ' Umieszcza dodatek w kolekcji AddIns i go instaluje.
            Set NewAi =
                Application.AddIns.Add(ThisWorkbook.FullName)
            NewAi.Installed = True
        Case vbNo
            ' Nie podejmuj żadnej akcji, pozostaw plik otwarty
        Case vbCancel
            ThisWorkbook.Close
    End Select
End Sub
```

Procedura działa w następujących scenariuszach:

- Dodatek został otwarty automatycznie, ponieważ był to dodatek już zainstalowany, widoczny w oknie *Dodatki* i jego nazwa była zaznaczona. Na ekranie nie jest wyświetlany żaden komunikat.
- Użytkownik użył okna *Dodatki* do zainstalowania dodatku. Na ekranie nie jest wyświetlany żaden komunikat.

- Dodatek został otwarty ręcznie (za pomocą polecenia *Plik/Otwórz*) i nie należy do kolekcji *AddIns*. Na ekranie jest wyświetlany odpowiedni komunikat i użytkownik musi wybrać jedną z trzech możliwych akcji.
- Dodatek został otwarty ręcznie (za pomocą polecenia *Plik/Otwórz*) i należy do kolekcji *AddIns*, ale nie jest zainstalowany (jego nazwa w oknie *Dodatki* nie jest zaznaczona). Na ekranie jest wyświetlany odpowiedni komunikat i użytkownik musi wybrać jedną z trzech możliwych akcji.

Tak na marginesie, nasza procedura może być wykorzystywana do uproszczenia instalacji dodatku, który przekazujesz komuś innemu. Wystarczy, że powiesz tej osobie, aby dwukrotnie kliknęła plik dodatku lewym przyciskiem myszy i w odpowiedzi na pojawienie się komunikatu naciągnęła przycisk *Tak*. Jeszcze lepszym rozwiązaniem będzie oczywiście taka modyfikacja kodu, aby dodatek sam się poprawnie instalował, bez wyświetlania żadnego komunikatu.



Dodatek zawierający naszą procedurę (*Sprawdzanie dodatków.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Spróbuj otworzyć go za pomocą obu metod (okno dialogowe *Dodatki* oraz polecenie *PLIK/Otwórz*).

Odwolowywanie się do innych plików z poziomu dodatku

Należy zachować szczególną ostrożność, rozpowszechniając dodatek wykorzystujący inne pliki. Nie można niczego zakładać odnośnie do struktury plików systemu, w którym użytkownicy będą wykonywali aplikację. Najprostszym podejściem jest wymaganie umieszczenia wszystkich plików aplikacji w jednym katalogu. Następnie można wykorzystać właściwość Path skoroszytu aplikacji w celu utworzenia odwołań do wszystkich pozostałych plików.

Jeżeli na przykład w aplikacji wykorzystano plik pomocy, należy zapewnić, aby został on skopiowany do tego samego katalogu, w którym umieszczono aplikację. Następnie w celu sprawdzenia, czy można odnaleźć plik, wykonamy poniższy kod:

```
Sub GetHelp()  
    Application.Help ThisWorkbook.Path & "\PlikPomocy.chm"  
End Sub
```

Jeżeli w aplikacji wykorzystałś wywołania interfejsu API do standardowych bibliotek DLL systemu Windows, można założyć, że system Windows je odnajdzie. Jeżeli jednak wykorzystałś własne biblioteki DLL, powinieneś dopilnować, aby zostały zainstalowane w katalogu *Windows\System* (czasami ten katalog ma inną nazwę). Aby uzyskać dokładną ścieżkę katalogu *System*, można skorzystać z funkcji interfejsu Windows API *GetSystemDirectory*.

Wykrywanie właściwej wersji Excela dla dodatku

Jak już zapewne wiesz, użytkownicy starszych wersji Excela mogą otwierać skoroszyty Excela 2007 (i nowsze) po zainstalowaniu *Pakietu zgodności formatu plików pakietu Microsoft Office*. Jeżeli Twój dodatek wykorzystuje właściwości unikatowe dla Excela 2007

lub nowszych wersji, należy wyświetlić ostrzeżenie dla użytkowników próbujących otworzyć ten dodatek za pomocą wcześniejszych wersji. Można to zrobić za pomocą powyższego kodu:

```
Sub CheckVersion()
    If Val(Application.Version) < 12 Then
        MsgBox "Ten dodatek działa tylko z Exceliem 2007 lub wersją nowszą."
        ThisWorkbook.Close
    End If
End Sub
```

Właściwość `Version` obiektu `Application` zawiera łańcuch znaków, na przykład może mieć wartość `12.0a`. Z tego powodu w zaprezentowanej procedurze wykorzystano funkcję VBA `Val`, która ignoruje wszystkie znaki, począwszy od pierwszej napotkanej litery.



Więcej informacji na temat kompatybilności różnych wersji Excela znajdziesz w rozdziale 24.

Część V

Tworzenie aplikacji

W tej części:

Rozdział 20. „Praca ze Wstążką”

Rozdział 21. „Praca z menu podręcznym”

Rozdział 22. „Tworzenie systemów pomocy w aplikacjach”

Rozdział 23. „Tworzenie aplikacji przyjaznych dla użytkownika”

Rozdział 20.

Praca ze Wstążką

W tym rozdziale:

- Wstążka, czyli nowy interfejs programu Excel z punktu widzenia użytkownika
- Jak używać VBA do pracy ze Wstążką
- Modyfikacja Wstążki przy użyciu kodu RibbonX
- Przykłady skoroszytów modyfikujących Wstążkę
- Przykład procedury tworzącej stary, dobry pasek narzędzi

Wprowadzenie do pracy ze Wstążką

Pierwszym elementem, który rzucał się w oczy po uruchomieniu Excela 2007, był jego zupełnie nowy wygląd. Stare, dobrze wszystkim znane menu i paski narzędzi odeszły w niepamięć i zostały zastąpione kartami interfejsu o wdzięcznej nazwie *Wstążka*. Pomimo iż pod pewnymi względami Wstążka może nieco przypominać tradycyjne menu i paski narzędzi, to jednak szybko przekonasz się, że jest to zupełnie inny, nowy interfejs użytkownika.

Doświadczeni użytkownicy Excela z pewnością zauważyli, że tradycyjne menu Excela z każdą kolejną wersją robiło się coraz bardziej złożone, a paski narzędzi w pewnym momencie stały się wręcz przeładowane ogromną liczbą przycisków poleceń, co po części było rezultatem poniekąd słusznego założenia, że niemal każda nowa funkcja musi być łatwo dostępna dla użytkownika. W poprzednich wersjach taka strategia prowadziła do zwiększania liczby poleceń dostępnych w menu i tworzenia nowych pasków narzędzi. Projektanci firmy Microsoft postanowili zmierzyć się z tym wyzwaniem i Wstążką, czyli nowym interfejsem programu Excel, jest właśnie rezultatem tych starań.

Reakcje użytkowników na wprowadzenie nowego interfejsu najlagodniej można określić słowem *mieszane*. Jak to z każdą nowością bywa, niektórzy użytkownicy pokochali Wstążkę od pierwszego wejrzenia, podczas gdy uczucia innych są dosyć negatywne. Osobiście zaliczyłbym się raczej do tej pierwszej grupy. Po używaniu kilku nowych wersji wersji Excela naprawdę trudno byłoby powrócić do skomplikowanego menu Excela 2003.

Moim skromnym zdaniem bardziej zaawansowani użytkownicy Excela doświadczają czegoś na kształt łagodnego wstrząsu, kiedy przekonają się, że dobrze im znane skróty klawiszowe i sekwencje poleceń w nowej wersji po prostu nie działają. Z drugiej strony, początkujący użytkownicy Excela będą mogli szybko wdrożyć się do pracy z tym programem bez konieczności przedzierania się przez irytująco złożone, wielopoziomowe menu i przeładowane paski narzędzi.

Mając na względzie głównie użytkowników, którzy zetknęli się ze Wstążką po raz pierwszy, w kolejnych podrozdziałach postaram się ją nieco przybliżyć.

Polecenia dostępne na Wstążce zmieniają się w zależności od tego, która karta jest aktywna w danej chwili. Poszczególne karty Wstążki grupują polecenia o podobnym przeznaczeniu. Poniżej zamieszczono krótki przegląd kart Wstążki:

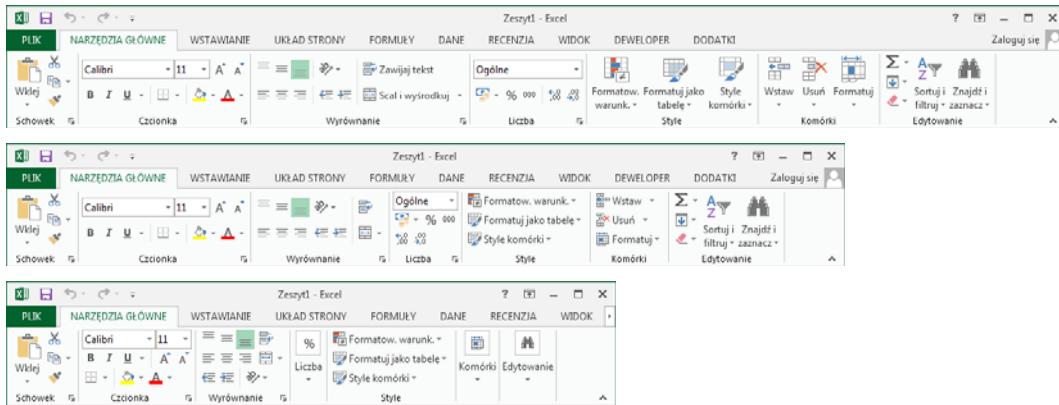
- **NARZĘDZIA GŁÓWNE** — Prawdopodobnie właśnie z tą kartą będziesz spędzał najwięcej czasu. Znajdziesz tutaj podstawowe polecenia obsługujące schowek systemowy, formatowanie, style, wstawianie i usuwanie wierszy i kolumn oraz wiele innych poleceń edycyjnych.
- **WSTAWIANIE** — Przejdź na tę kartę, kiedy będziesz chciał wstawić do arkusza nowy element, taki jak tabela, diagram, wykres, symbole itp.
- **UKŁAD STRONY** — Na tej karcie znajdziesz polecenia modyfikujące wygląd arkusza oraz ustawienia związane z drukowaniem.
- **FORMUŁY** — Poleceń znajdujących się na tej karcie możesz używać do wstawiania formuł, definiowania nazw zakresów, korzystania z narzędzi inspekcji formuł oraz sterowania sposobem wykonywania obliczeń.
- **DANE** — Tutaj znajdziesz polecenia związane z przetwarzaniem danych.
- **RECENZJA** — Na tej karcie znajdziesz narzędzia pozwalające na sprawdzenie pisowni, tłumaczenie tekstu na inne języki, dodawanie komentarzy i ochronę arkusza.
- **WIDOK** — Karta *WIDOK* kryje szereg poleceń pozwalających na sterowanie sposobem wyświetlania arkusza. Niektóre polecenia z tej karty są również dostępne na pasku stanu.
- **DEVELOPER** — Ta karta domyślnie nie jest widoczna na Wstążce. Znajdują się na niej polecenia użyteczne dla programistów. Aby wyświetlić kartę *DEVELOPER* na Wstążce, kliknij Wstążkę prawym przyciskiem myszy, z menu podręcznego wybierz polecenie *Dostosuj Wstążkę* i w oknie dialogowym *Opcje programu Excel*, które pojawi się na ekranie, zaznacz opcję *DEVELOPER*.
- **DODATKI** — Karta *DODATKI* jest widoczna tylko wtedy, kiedy załadujesz skoroszyt lub dodatek, który modyfikuje menu lub paski narzędzi (poprzez użycie obiektów *CommandBar*). Ponieważ menu użytkownika i niestandardowe paski narzędzi nie są już dostępne, takie modyfikacje są wyświetlane na karcie *Dodatki*.

Wygląd poleceń na Wstążce zmienia się w zależności od szerokości okna programu Excel. Kiedy okno Excela jest zbyt wąskie, aby zmieściły się wszystkie polecenia, Wstążka automatycznie adaptuje się do nowego otoczenia i może się wydawać, że niektóre polecenia

z niej znikają — w rzeczywistości jednak nadal są dostępne. Na rysunku 20.1 przedstawiono trzy widoki karty *NARZĘDZIA GŁÓWNE*. Na pierwszym rysunku wszystkie polecenia tej karty są widoczne. W drugim przypadku szerokość okna Excela została nieco zmniejszona. Zauważ, że etykiety niektórych przycisków zniknęły, a niektóre przyciski są wyraźnie mniejsze. Na ostatnim, trzecim rysunku, przedstawiono nieco ekstremalny przypadek, kiedy szerokość okna Excela została bardzo radykalnie zmniejszona. Jak łatwo zauważać, niektóre grupy poleceń zamieniły się w pojedyncze ikony, ale kiedy klikniesz taką ikonę, na ekranie rozwinię się lista wszystkich dostępnych poleceń z tej grupy.



Jeżeli chcesz ukryć Wstążkę, tak aby maksymalnie powiększyć roboczy obszar arkusza, powinieneś po prostu dwukrotnie kliknąć dowolną kartę. Wstążka zniknie (na ekranie pozostaną widoczne jedynie karty polecen), dzięki czemu będziesz mógł wyświetlić około pięciu wierszy arkusza więcej. Kiedy będziesz chciał skorzystać ze Wstążki, po prostu kliknij dowolną kartę i polecenia ponownie pojawią się (tymczasowo) na ekranie. Aby na stałe przywrócić wyświetlanie pełnej Wstążki, dwukrotnie kliknij dowolną kartę. Zamiast tego do przełączania wyświetlania Wstążki możesz używać kombinacji klawiszy *Ctrl+F1* lub kliknąć ikonę , znajdująca się obok ikony pomocy systemowej, w prawej części paska tytułowego głównego okna Excela.



Rysunek 20.1. Wygląd karty *NARZĘDZIA GŁÓWNE* przy różnych szerokościach okna Excela

Obiekt CommandBar w Excelu 2007

W Excelu 97 został wprowadzony zupełnie nowy sposób obsługi menu użytkownika i pasków narzędzi. Tego typu elementy interfejsu użytkownika zostały zdefiniowane jako obiekty klasy *CommandBar*. Element interfejsu znany powszechnie jako pasek narzędzi jest w rzeczywistości jednym z trzech typów obiektów tej klasy:

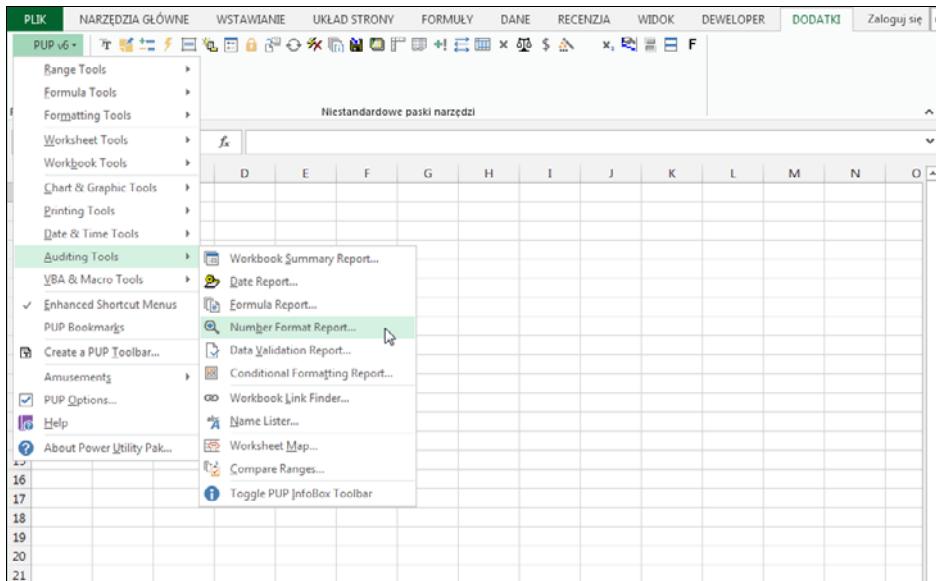
- **Pasek narzędzi** (ang. *Toolbar*) — to pasek, na którym zostały umieszczone przyciski jednego lub więcej poleceń.
- **Pasek menu** (ang. *Menu bar*) — Excel posiadał dwa wbudowane paski menu: pasek menu arkusza oraz menu wykresu.
- **Menu podręczne** (ang. *Shortcut menu*) — to menu, które pojawia się na ekranie po kliknięciu obiektu prawym przyciskiem myszy.

Ze względu na konieczność zachowania kompatybilności Excel 2013 nadal obsługuje obiekty klasy CommandBar — ale ich funkcjonalność została znaczco zredukowana. Użytkownik nie ma już możliwości utworzenia swojego własnego paska narzędzi, aczkolwiek programiści VBA nadal mogą tworzyć obiekty klasy CommandBar i ich używać (więcej szczegółowych informacji na ten temat znajdziesz w podrozdziale „Tworzenie pasków narzędzi w starym stylu” w dalszej części rozdziału). Główny problem polega jednak na tym, że większość właściwości i metod obiektów CommandBar w Excelu 2007 i nowszych wersjach jest po prostu ignorowana. Na przykład paski narzędzi użytkownika i niestandardowe menu pojawiają się tylko na karcie DODATKI. Właściwości sterujące rozmiarami i położeniem pasków narzędzi po prostu nie działają (stąd nie można również tworzyć „pływających” pasków narzędzi).

Na rysunkach zamieszczonych poniżej przedstawiono wygląd niestandardowego menu i paska narzędzi użytkownika w Excelu 2003 oraz wygląd tych samych elementów w Excelu 2013. Po-mimo że takie elementy interfejsu użytkownika nadal działają w Excelu 2013, to jednak wyraźnie nie tak to miało wyglądać w zamysle projektanta (czyli w tym przypadku moim!). Nie trzeba chyba dodawać, że wielu programistów VBA będzie musiał przeprojektować interfejs użytkownika swoich starszych aplikacji.

W dalszej części tego rozdziału zaprezentujemy prosty przykład tworzenia niestandardowego paska narzędzi przy użyciu obiektu CommandBar (patrz „Tworzenie pasków narzędzi w starym stylu” w dalszej części rozdziału). Szczegółowe informacje na temat tworzenia tradycyjnych pasków narzędzi i niestandardowych menu użytkownika znajdziesz w wydaniu Excel 2003 tej książki.

Excel 2013 nadal pozwala jednak na dostosowywanie menu podręcznego przy użyciu obiektów CommandBar, aczkolwiek trzeba zauważać, że nowy, jednodokumentowy interfejs Excela powoduje, że takie rozwiązanie staje się w przypadku wielu aplikacji zupełnie bezużyteczne. W takich sytuacjach znacznie lepszym i preferowanym rozwiązaniem jest modyfikacja menu podręcznego przy użyciu kodu RibbonX — więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 21.



VBA i Wstążka

A teraz nadszedł czas na ważne pytanie: co programista może zrobić ze Wstążką przy użyciu VBA? Odpowiedź jest prosta: niewiele.

Poniżej przedstawiam krótkie zestawienie tego, co możesz zrobić ze Wstążką przy użyciu VBA.

- Możesz sprawdzić, czy dany formant jest włączony.
- Możesz sprawdzić, czy dany formant jest widoczny.
- Możesz sprawdzić, czy dany formant został naciśnięty (dotyczy przycisków i pól wyboru).
- Możesz pobrać etykietę formantu, etykietę ekranową lub superetykietę (czyli bardziej rozbudowany opis funkcji i przeznaczenia formantu).
- Możesz wyświetlić obraz powiązany z formantem.
- Możesz uruchomić polecenie powiązane z wybranym formantem.

Poniżej zamieszczono kolejną listę, tym razem będącą zestawieniem operacji, które prawdopodobnie chciałbyś zrobić ze Wstążką, ale niestety nie jest to możliwe.

- Nie możesz sprawdzić, która karta jest aktywna.
- Nie możesz aktywować wybranej karty.
- Nie możesz dodawać nowych kart.
- Nie możesz tworzyć nowych grup polecen na karcie.
- Nie możesz dodawać nowych formantów.
- Nie możesz usuwać istniejących formantów.
- Nie możesz zablokować działania istniejących formantów.
- Nie możesz ukrywać istniejących formantów.



Począwszy od Excela 2010 użytkownik może modyfikować Wstążkę, korzystając z karty *Dostosowywanie Wstążki* okna dialogowego *Opcje programu Excel*. Niestety, takich zmian nie możesz dokonać przy użyciu VBA.

Dostęp do poleceń Wstążki

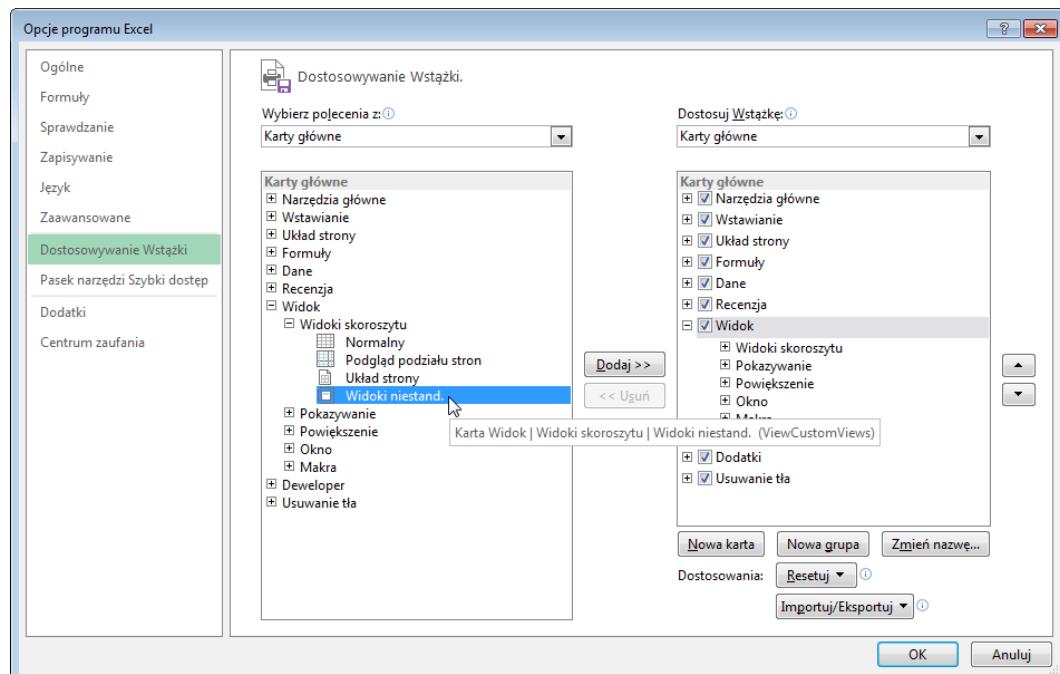
Krótko i bez ogródek — Excel posiada ponad 1700 formantów Wstążki. Każdy formant Wstążki posiada swoją nazwę, której możesz używać w kodzie VBA podczas pracy z tym formantem.

Na przykład polecenie przedstawione poniżej wyświetla na ekranie okno dialogowe, które pokazuje status formantu *ViewCustomViews* (przycisk polecenia *Widoki niestandard.* znajduje się na karcie *WIDOK* w grupie polecen *Widoki skoroszytu*).

```
MsgBox Application.CommandBars.GetEnabledMso("ViewCustomViews")
```

Domyślnie ten formant jest włączony, ale — co jest zupełnie niezrozumiałe — jeżeli w skoroszycie znajduje się tabela (utworzona za pomocą polecenia *Wstawianie/Tabele/Tabela*), to formant *ViewCustomViews* zostaje zablokowany. Innymi słowy, w skoroszycie możesz korzystać z widoków niestandardowych albo z tabel — ale nigdy z obu tych elementów jednocześnie.

Odszukanie nazwy danego formantu jest zadaniem, które musisz wykonać ręcznie. Aby to zrobić, powinieneś najpierw wyświetlić kartę *Dostosowywanie Wstążki* okna dialogowego *Opcje programu Excel*. Następnie odszukaj na liście żądane polecenie i ustaw nad jego ikoną wskaźnik myszy. Po krótkiej chwili nazwa formantu pojawi się w postaci etykiety ekranowej (patrz rysunek 20.2).



Rysunek 20.2. Do określenia nazwy formantu możesz użyć karty *Dostosowywanie* okna dialogowego *Opcje programu Excel*

Niestety nie możesz napisać procedury VBA, która przejdzie w pętli kolejno przez wszystkie formanty Wstążki i wyświetli ich nazwy.



Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>) znajdziesz skoroszyt zawierający nazwy wszystkich formantów Wstążki Excela 2013 i Excela 2010. Na rysunku 20.3 przedstawiono fragment tego skoroszytu (*Nazwy formantów Wstążki.xlsx*).

A	B	C	D	E
1 Nazwa formantu	Typ	Nazwa zestawu kart	Nazwa karty	Nazwa grupy
2 FileNewDefault	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
3 FileOpen	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
4 FileSave	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
5 FileSendAsAttachment	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
6 FilePrintQuick	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
7 FilePrintPreview	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
8 Spelling	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
9 Undo	gallery	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
10 Redo	gallery	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
11 SortAscendingExcel	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
12 SortDescendingExcel	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
13 FileOpenRecentFile	button	Brak (pasek Szybki dostęp)	Pasek Szybki dostęp	
14 TabHome	tab	Brak (Karty główne)		
15 GroupClipboard	group	Brak (Karty główne)	TabHome	
16 PasteMenu	splitButton	Brak (Karty główne)	TabHome	GroupClipboard
17 Paste	button	Brak (Karty główne)	TabHome	GroupClipboard
18 PasteGallery	gallery	Brak (Karty główne)	TabHome	GroupClipboard
19 PasteUsingTextImportWizard	button	Brak (Karty główne)	TabHome	GroupClipboard
20 PasteRefreshableWebQuery	button	Brak (Karty główne)	TabHome	GroupClipboard
21 PasteSpecialDialog	button	Brak (Karty główne)	TabHome	GroupClipboard
22 Cut	button	Brak (Karty główne)	TabHome	GroupClipboard
23 CopySplitButton	splitButton	Brak (Karty główne)	TabHome	GroupClipboard
24 Copy	button	Brak (Karty główne)	TabHome	GroupClipboard
25 Copy	button	Brak (Karty główne)	TabHome	GroupClipboard
26 CopyAsPicture	button	Brak (Karty główne)	TabHome	GroupClipboard
27 FormatPainter	control	Brak (Karty główne)	TabHome	GroupClipboard
28 ShowClipboard	button (dialogBc	Brak (Karty główne)	TabHome	GroupClipboard
29 GroupFont	group	Brak (Karty główne)	TabHome	
30 Font	comboBox	Brak (Karty główne)	TabHome	GroupFont
31 FontSize	comboBox	Brak (Karty główne)	TabHome	GroupFont
32 FontSizeIncrease	button	Brak (Karty główne)	TabHome	GroupFont

Rysunek 20.3. Skoroszyt zawierający informacje o formantach Wstążki

Praca ze Wstążką

W poprzedniej sekcji zaprezentowaliśmy przykład zastosowania metody GetEnabledMso obiektu CommandBars. Poniżej przedstawiono listę wszystkich metod obiektu CommandBars dostępnych do pracy ze Wstążką. Wszystkie metody pobierają jeden argument: idMso, reprezentujący nazwę polecenia.

- ExecuteMso — uruchomienie polecenia.
- GetEnabledMso — zwraca wartość True, jeżeli dany formant jest włączony.
- GetImageMso — zwraca obraz powiązany z formantem.
- GetLabelMso — zwraca etykietę formantu.
- GetPressedMso — zwraca wartość True, jeżeli dany formant jest naciśnięty (dotyczy przycisków przełączników i pól wyboru).
- GetScreenTipMso — zwraca tekst etykiety ekranowej formantu (tekst, który pojawia się na formancie).

- GetSupertipMso — zwraca tekst superetykiety formantu (opis formantu, który pojawia się, kiedy ustawisz wskaźnik myszy nad formantem).

Niektóre z tych metod są mało przydatne. Dlaczego na przykład programista miałby chcieć pobrać tekst etykiety ekranowej formantu? Nie potrafię sobie wyobrazić praktycznego zastosowania takiej metody.

Polecenie VBA przedstawione poniżej przełącza wyświetlanie panelu *Zaznaczanie i widoczność* (funkcja wprowadzona w Excelu 2007, ułatwiająca zaznaczanie obiektów na arkuszu):

```
Application.CommandBars.ExecuteMso "SelectionPane"
```

Kolejne polecenie wyświetla okno dialogowe *Wklejanie specjalne* (jeżeli schowek systemowy jest pusty, próba wykonania tego polecenia zakończy się wystąpieniem błędu):

```
Application.CommandBars.ExecuteMso "PasteSpecialDialog"
```

Poniżej przedstawiono polecenie, które wyświetla informację, czy pasek formuły jest widoczny (czyli inaczej mówiąc, wyświetla stan formantu *Pasek formuły*, znajdującego się na karcie *WIDOK* w grupie *Pokazywanie*):

```
MsgBox Application.CommandBars.GetPressedMso "ViewFormulaBar"
```

Aby ukryć lub wyświetlić pasek formuł, powinieneś użyć następującego polecenia:

```
Application.CommandBars.ExecuteMso "ViewFormulaBar"
```

By upewnić się, że pasek formuł *jest widoczny* na ekranie, powinieneś użyć następującego fragmentu kodu:

```
With Application.CommandBars  
    If Not .GetPressedMso("ViewFormulaBar") Then .ExecuteMso "ViewFormulaBar"  
End With
```

Aby upewnić się, że pasek formuł *nie jest widoczny* na ekranie, powinieneś użyć następującego fragmentu kodu:

```
With Application.CommandBars  
    If .GetPressedMso("ViewFormulaBar") Then .ExecuteMso "ViewFormulaBar"  
End With
```

Możesz również zupełnie nie zwracać uwagi na Wstążkę i zamiast tego ustawić właściwość DisplayFormulaBar obiektu Application na wartość odpowiednio True lub False. Polecenie przedstawione poniżej wyświetla na ekranie pasek formuł (lub nie wywołuje żadnego efektu, jeżeli pasek jest już widoczny):

```
Application.DisplayFormulaBar = True
```

Następne polecenie wyświetla wartość True, jeżeli formant *Scal i wyśrodkuj* jest włączony (formant ten jest zablokowany, jeżeli arkusz jest chroniony lub jeżeli aktywna komórka znajduje się w tabeli).

```
MsgBox Application.CommandBars.GetEnabledMso("MergeCenter")
```

Procedura, której kod przedstawiono poniżej, umieszcza na aktywnym arkuszu formant ActiveX i używa metody GetImageMso do wyświetlenia obrazu (ikony) powiązanego z formantem **Znajdź i zaznacz** (karta *NARZĘDZIA GŁÓWNE/Edytowanie*):

```
Sub ImageOnSheet()
    Dim MyImage As OLEObject
    Set MyImage = ActiveSheet.OLEObjects.Add _
        (ClassType:="Forms.Image.1", _
        Left:=50, _
        Top:=50)
    With MyImage.Object
        .AutoSize = True
        .BorderStyle = 0
        .Picture = Application.CommandBars. _
            GetImageMso("FindDialog", 32, 32)
    End With
End Sub
```

Aby wyświetlić ikonę ze Wstążki na formancie *Image* (o nazwie *Image1*) osadzonym na formularzu *UserForm*, możesz użyć procedury przedstawionej poniżej:

```
Private Sub UserForm_Initialize()
    With Image1
        .Picture = Application.CommandBars. GetImageMso _
            ("FindDialog", 32, 32)
        .AutoSize = True
    End With
End Sub
```

Aktywowanie karty

Microsoft nie udostępnia żadnej bezpośredniej metody aktywowania wybranej karty Wstążki z poziomu procedury VBA. Jeżeli jednak naprawdę musisz dokonać czegoś takiego, jedynym rozwiązaniem pozostaje użycie metody *SendKeys*, która symuluje naciśnięcie klawiszy. Aby na przykład aktywować kartę *NARZĘDZIA GŁÓWNE*, powinieneś nacisnąć klawisz *Alt* i następnie klawisz *G*. Naciśnięcie klawisza *Alt* powoduje wyświetlenie na ekranie etykiet z podpowiedziami. Aby ukryć podpowiedzi, należy nacisnąć klawisz *F6*. Mając takie informacje, możemy utworzyć polecenie, które wysyła sekwencję klawiszy niezbędną do aktywacji karty *NARZĘDZIA GŁÓWNE*.

```
Application.SendKeys "%g{F6}"
```

Aby uniknąć wyświetlania na ekranie etykiet z podpowiedziami, możesz wyłączyć aktualizację ekranu:

```
Application.ScreenUpdating = False
Application.SendKeys "%h{F6}"
Application.ScreenUpdateing=True
```

Argumenty metody *SendKeys* aktywujące pozostałe karty Wstążki są następujące:

- **WSTAWIANIE** — "%v{F6}"
- **UKŁAD STRONY** — "%a{F6}"
- **FORMUŁY** — "%m{F6}"

Zachowywanie zmian konfiguracji interfejsu użytkownika

Począwszy od Excela 2010 użytkownik może dokonywać zmian konfiguracji Wstążki oraz paska narzędzi *Szybki dostęp* i bez większych problemów umieszczać na nich dodatkowe polecenia. W jaki sposób Excel przechowuje informacje o takich zmianach konfiguracji?

Zmiany konfiguracji Wstążki oraz paska narzędzi *Szybki dostęp* są przechowywane w pliku o nazwie *Excel.officeUI*. Lokalizacja tego pliku może się zmieniać. Na moim komputerze plik ten jest zlokalizowany w następującym katalogu:

C:\Użytkownicy\<nazwa_użytkownika>\AppData\Local\Microsoft\Office

Więcej szczegółowych informacji na temat pliku *Excel.officeUI* znajdziesz w rozdziale 3.

- **DANE** — "%u{F6}"
- **RECENZJA** — "%r{F6}"
- **WIDOK** — "%o{F6}"
- **DEVELOPER** — "%q{F6}"
- **DODATKI** — "%x{F6}"



Jak zawsze, użycie metody *SendKeys* powinieneś traktować jako ostatnią deskę ratunku. Powinieneś również zdawać sobie sprawę z tego, że takie rozwiązanie nie zawsze będzie działać tak, jak tego oczekiwaneś. Na przykład jeżeli uruchomisz polecenie przedstawione powyżej w sytuacji, kiedy na ekranie jest wyświetlony formularz *UserForm*, kody klawiszy zostaną wysłane do formularza, a nie do Wstążki.

Dostosowywanie Wstążki do własnych potrzeb

Przy użyciu VBA nie możesz dokonywać żadnych modyfikacji Wstążki. Zamiast tego musisz utworzyć specjalny kod RibbonX i wstawić go do pliku skoroszytu — co odbywa się całkowicie poza Exceliem. W języku VBA możesz jednak tworzyć tzw. procedury zwrotne (ang. *callback procedures*). *Procedura zwrotna* to makro VBA, które jest wykonywane po aktywacji danego formantu Wstążki.

Kod RibbonX to specjalny rodzaj kodu XML opisującego rodzaj formantu, jego położenie na Wstążce, wygląd i operacje, które są wykonywane po aktywowaniu tego formantu. W tej książce nie będziemy się zajmować kodem RibbonX — zagadnienie jest na tyle złożone, że swobodnie można by mu poświęcić osobną książkę. Zaprezentuję tutaj jednak kilka prostych przykładów, które pozwolą Ci zapoznać się ze sposobami modyfikacji interfejsu użytkownika programu Excel i podjąć decyzję, czy jest to coś, z czym chciałbyś się zapoznać bardziej dokładnie.



Więcej szczegółowych informacji na temat samej struktury skoroszytu Excela znajdziesz w rozdziale 3. W tym podrozdziale opisujemy sposób, w jaki możesz przeglądać informacje przechowywane wewnątrz plików *XLSX*.

Wyświetlanie błędów

Zanim zaczniesz modyfikować ustawienia Wstążki, powinieneś włączyć wyświetlanie błędów kodu RibbonX. Aby to zrobić, przejdź na kartę *PLIK* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kartę *Zaawansowane*, przejdź do sekcji *Ogólne* i zaznacz opcję *Pokaż błędy interfejsu użytkownika dodatku*.

Po włączeniu tej opcji Excel w momencie otwierania skoroszytu będzie wyświetlał komunikaty błędów kodu RibbonX (oczywiście o ile takie błędy wystąpią) — co będzie bardzo przydatne podczas wyszukiwania i usuwania błędów.

Prosty przykład kodu RibbonX

W tej sekcji zaprezentujemy krok po kroku przykład modyfikacji Wstążki programu Excel. W naszym przykładzie utworzymy na karcie *Dane* Wstążki nową grupę poleceń o nazwie *Niestandardowe* i umieścimy w niej dwa nowe przyciski poleceń, o nazwach *Witaj!* i *Do widzenia!*. Naciśnięcie tych przycisków będzie uruchamiało odpowiednie makra VBA.



Metoda przedstawiona w tym podrozdziale jest nuancka i podatna na błędy. W praktyce zdecydowana większość projektantów aplikacji nie postępuje w taki sposób — zamiast tego używają oni specjalnego oprogramowania, które powoduje, że proces modyfikacji interfejsu jest prosty i wygodny.

Aby utworzyć skoroszyt zawierający kod RibbonX modyfikujący Wstążkę, powinieneś wykonać następujące polecenia:

1. Utwórz nowy skoroszyt Excela, wstaw moduł kodu VBA i wprowadź dwie procedury zwrotne, których kod przedstawiono poniżej.

```
Sub HelloWorld(control As IRibbonControl)
    MsgBox "Witaj świecie!"
End Sub
```

```
Sub GoodbyeWorld(control As IRibbonControl)
    ThisWorkbook.Close
End Sub
```

Utworzone procedury będą uruchamiane po naciśnięciu nowych przycisków na Wstążce:

2. Zapisz skoroszyt pod nazwą *Modyfikacja Wstążki.xlsxm*.
3. Zamknij skoroszyt.
4. Przejdź na dysku do folderu, w którym zapiszałeś skoroszyt *Modyfikacja Wstążki.xlsxm* i utwórz folder o nazwie *customUI*.
5. Przejdź do tego folderu i przy użyciu edytora tekstu (takiego jak *Notatnik*) utwórz plik tekstowy o nazwie *customUI.xml* i umieść w nim następujący kod RibbonX¹:

¹ Aby uzyskać polskie znaki powinieneś zapisać plik *customUI.xml* w formacie UTF-8 — przyp. tłum.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon>
<tabs>
<tab idMso="TabData">
<group id="Group1" label="Niestandardowe">
<button id="Button1"
label="Witaj!" 
size="normal"
onAction="HelloWorld"
imageMso="HappyFace" />
<button id="Button2"
label="Do widzenia!"
size="normal"
onAction="GoodbyeWorld"
imageMso="DeclineInvitation" />
</group>
</tab>
</tabs>
</ribbon>
</customUI>
```

**Uwaga**

Jeżeli Twój system jest tak skonfigurowany, że rozszerzenia nazw znanych typów plików nie są wyświetlane, powinieneś wyłączyć tę opcję tak, aby rozszerzenia plików były zawsze widoczne. Aby to zrobić, uruchom Eksploratora Windows i z menu głównego wybierz polecenie *Narzędzia/Opcje folderów*. Na ekranie pojawi się okno dialogowe *Opcje folderów*. Przejdź na kartę *Widok* i usuń zaznaczenie opcji *Ukryj rozszerzenia znanych typów plików*.

6. Przy użyciu Eksploratora Windows dodaj do nazwy pliku *Modyfikacja Wstążki.xlsm* rozszerzenie *.zip*.

Nazwa pliku po wykonaniu tej operacji powinna wyglądać następująco:
Modyfikacja Wstążki.xlsm.zip.

7. Przeciągnij folder *customUI*, który utworzyłeś w punkcie 4, na plik *Modyfikacja Wstążki.xlsm.zip*.

System Windows traktuje pliki ZIP jak foldery, stąd operacje typu przeciągnij i upuść są dozwolone.

8. Dwukrotnie kliknij plik *Modyfikacja Wstążki.xlsm.zip*, aby go otworzyć.

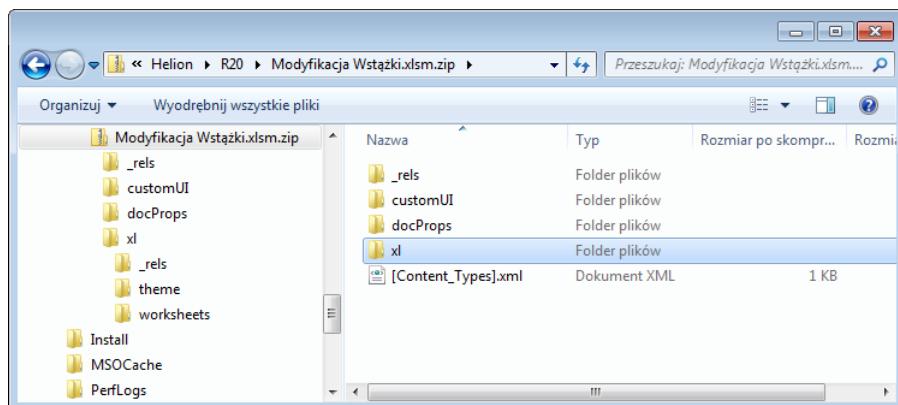
Na rysunku 20.4 przedstawiono zawartość pliku ZIP. Jak łatwo zauważyc, w pliku znajduje się kilka folderów.

9. W pliku ZIP dwukrotnie kliknij folder *_rels*.

W tym folderze znajduje się tylko jeden plik, o nazwie *.rels*.

10. Przeciągnij plik *.rels* na zewnątrz pliku ZIP (na przykład na pulpit).

11. Otwórz plik *.rels* (jest to plik w formacie XML) w dowolnym edytorze tekstu, takim jak na przykład *Notatnik*.



Rysunek 20.4. Skoroszyt Excela wyświetlony jako plik ZIP

12. Odszukaj znacznik </Relationships> i dodaj przed nim następujący wiersz kodu:

```
<Relationship Type="http://schemas.microsoft.com/office/2006/
relationships/ui/extensibility" Target="/customUI/customUI.xml"
Id="12345" />
```

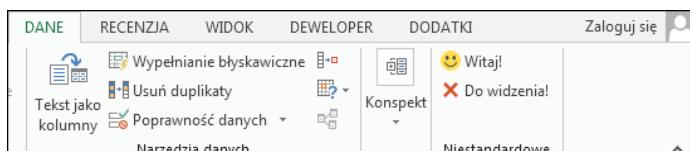
13. Zapisz plik .rels i przeciągnij go ponownie do pliku ZIP, nadpisując oryginalną wersję pliku.

14. Usuń rozszerzenie .zip, przywracając tym samym początkową nazwę pliku *Modyfikacja Wstążki.xlsxm*.

Otwórz skoroszyt w Excelu. Jeżeli wszystko poszło dobrze, na karcie *DANE* powinieneś zobaczyć nową grupę poleceń, w której znajdują się dwa nowe przyciski (patrz rysunek 20.5).

Rysunek 20.5.

Kod RibbonX utworzył nową grupę poleceń, w której znajdują się dwa przyciski



Skoroszyt z tym przykładem (*Modyfikacja Wstążki.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Pamiętaj, że modyfikacje Wstążki są powiązane z konkretnym skoroszytem. Innymi słowy, nowa grupa poleceń Wstążki będzie wyświetlana tylko wtedy, kiedy aktywny będzie skoroszyt zawierający odpowiedni kod RibbonX. Jest to zupełnie nowe rozwiązanie w stosunku do sposobów modyfikacji interfejsu użytkownika stosowanych w wersjach wcześniejszych niż Excel 2007.



Aby modyfikacje Wstążki były wyświetlane dla dowolnego skoroszytu, powinieneś dokonać konwersji skoroszytu na dodatek lub dodać kod RibbonX do skoroszytu makr osobistych.

Jeżeli dojdzieš do wniosku, że modyfikacja Wstążki programu Excel nie jest warta włożonego wysiłku, nie przejmuj się. Powstały już specjalne narzędzia, które zdecydowanie upraszczają opisany powyżej proces.

Prosty przykład kodu RibbonX — podejście 2.

W tym podrozdziale opiszemy krok po kroku sposób postępowania podczas modyfikacji interfejsu użytkownika programu Excel przy użyciu programu o nazwie *Custom UI Editor for Microsoft Office*. Używając tego programu, nadal musisz ręcznie wpisywać cały kod RibbonX, ale program dokonuje automatycznego sprawdzenia składni kodu, eliminuje konieczność żmudnego, ręcznego modyfikowania plików kodu RibbonX i skoroszytu oraz potrafi automatycznie wygenerować deklaracje odpowiednich procedur zwrotnych VBA, które wystarczy skopiować i wkleić w odpowiednie miejsce modułu kodu VBA skoroszytu.



Bezpłatna kopię edytora *Custom UI Editor for Microsoft Office* możesz łatwo znaleźć w sieci, wpisując w wyszukiwarce sieciowej hasło *office custom ui editor* (nie podajemy tutaj konkretnych adresów stron, z których można pobrać ten program, ponieważ od czasu do czasu ulegają one zmianie).

Aby dodać nową grupę i przyciski (takie, jak to zostało opisane w poprzednim przykładzie), korzystając z edytora *Custom UI Editor for Microsoft Office*, powinieneś wykonać polecenia opisane poniżej:

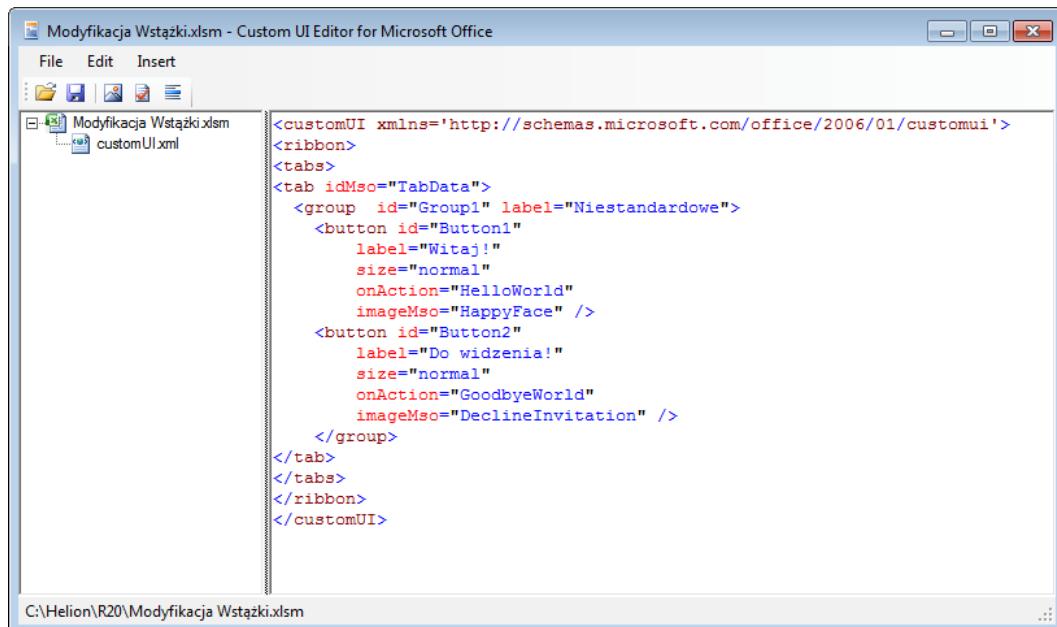
1. Przejdź do Excela, utwórz nowy skoroszyt i zapisz go w formacie skoroszytu Excela z obsługą makr (*XLSM*).
2. Zamknij skoroszyt.
3. Uruchom edytor *Custom UI Editor for Microsoft Office*.
4. Z menu głównego wybierz polecenie *File/Open* i odszukaj plik zapisany w punkcie 1.
5. Z menu głównego wybierz polecenie *Insert/Office 2007 Custom UI Part*.

Wybranie tej opcji powoduje, że plik skoroszytu będzie kompatybilny zarówno z Exceliem 2007, jak i z nowszymi wersjami.

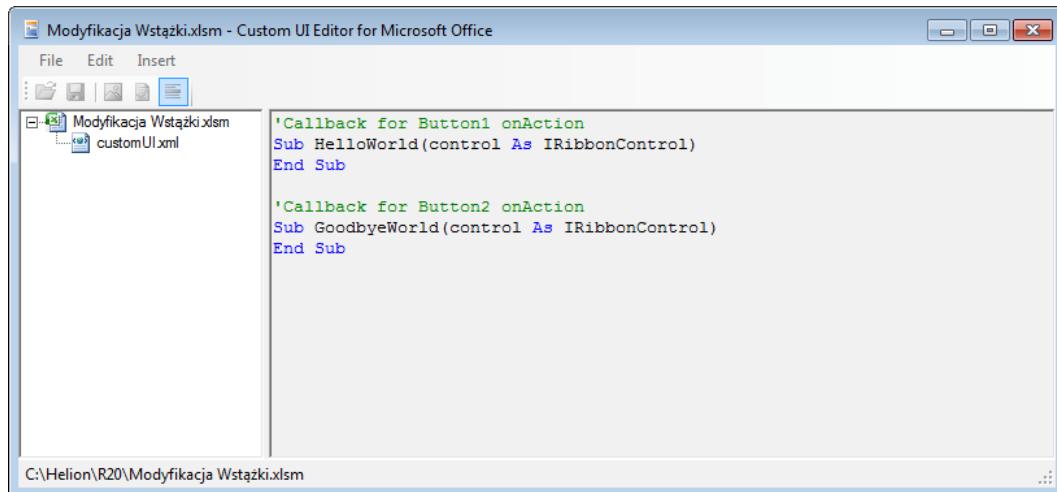
6. Wprowadź kod RibbonX przedstawiony na rysunku 20.6.
7. Naciśnij przycisk *Validate*, znajdujący się na pasku narzędzi, aby sprawdzić, czy w kodzie nie ma błędów.
8. Naciśnij przycisk *Generate Callbacks* i skopiuj kod, który pojawi się na ekranie.

Edytor wygeneruje dwie procedury zwrotne VBA (patrz rysunek 20.7).

Zaznacz i skopiuj kod procedur — za chwilę wkleisz ten kod do modułu VBA w skoroszycie.



Rysunek 20.6. Okno główne edytora Custom UI Editor for Microsoft Office



Rysunek 20.7. Kod dwóch procedur zwrotnych wygenerowany przez edytor Custom UI Editor for Microsoft Office

9. Kliknij węzeł *customUI.xml*, znajdujący się w lewej części okna.
10. Z menu głównego wybierz polecenie *File/Save*, a następnie *File/Close*.
11. Przejdz do Excela i otwórz skoroszyt.
12. Naciśnij kombinację klawiszy *lewy Alt+F11*, aby przejść do edytora VBE.

13. Wstaw nowy moduł VBA do projektu i wklej do niego kod skopiowany w punkcie 8.
14. Dodaj wiersz kodu z poleceniem MsgBox do każdej z dwóch procedur, dzięki czemu będziesz mógł sprawdzić poprawność ich działania.

Jak widać, praca z edytorem *Custom UI Editor for Microsoft Office* jest zdecydowanie łatwiejsza niż ręczna modyfikacja skoroszytu.

Sekcja CUSTOM UI

W punkcie 5. poprzedniego przykładu wstawiliśmy do pliku sekcję *Office 2007 Custom UI Part*. Taki wybór powoduje, że modyfikacje skoroszytu będą kompatybilne zarówno z Excellem w wersji 2007, jak i z nowszymi wersjami. Innym rozwiązaniem jest wstawienie sekcji *Office 2010 Custom UI Part*. Jeżeli wstawisz kod RibbonX w sekci *Office 2010 Custom UI Part*, to modyfikacje skoroszytu nie będą kompatybilne z Excellem 2007.



W czasie kiedy powstawała ta książka, program *Custom UI Editor for Microsoft Office* nie miał jeszcze opcji wstawiania sekcji *Office 2013 Custom UI Part*.

Jeżeli Twoja aplikacja nie korzysta z żadnych poleceń, które występują wyłącznie w wersji 2010 lub w nowszych wersjach, najlepszym rozwiązaniem będzie zastosowanie sekcji *Office 2007 Custom UI Part*. Pamiętaj także, że skoroszyt może posiadać jednocześnie obie sekcje, *Office 2007 Custom UI Part* oraz *Office 2010 Custom UI Part*. Takie rozwiązanie jest stosowane w sytuacji, kiedy w zależności od wersji Excela chcesz załadować inny kod RibbonX.

Zauważ, że pierwsze polecenie kodu RibbonX dla sekcji *Office 2010 Custom UI Part* musi zostać zmienione, tak aby kod odwoływał się do następującej przestrzeni nazw:

```
<customUI xmlns='http://schemas.microsoft.com/office/2009/07/customui'>
```



W czasie kiedy powstawał ten rozdział, definicja przestrzeni nazw dla sekcji *Office 2013 Custom UI Part* nie była jeszcze znana.

Jeżeli użyjesz nieprawidłowego znacznika *customUI*, edytor *Custom UI Editor for Microsoft Office* poinformuje Cię o tym podczas sprawdzania poprawności kodu.

Procedury zwrotne VBA

Jak pamiętasz, w skoroszycie znajdują się dwie procedury zwrotne VBA, *HelloWorld* oraz *GoodbyeWorld*. Nazwy procedur odpowiadają parametrom znacznika *onAction* w kodzie RibbonX. Parametr *onAction* jest jedynym elementem łączącym kod RibbonX z procedurami zwrotnymi VBA.

Obie procedury VBA posiadają argument o nazwie *control*, który jest obiektem klasy *IRibbonControl*. Obiekt ten posiada trzy właściwości, z których możesz korzystać w procedurach VBA:

- Context — to uchwyt aktywnego okna zawierającego Wstążkę, która wywołała procedurę zwrotną. Na przykład polecenia przedstawionego poniżej możesz używać do pobierania nazwy skoroszytu zawierającego kod RibbonX:

```
control.Context.Caption
```

- Id — zawiera nazwę formantu podanego jako parametr metody Id.
- Tag — zawiera dowolny tekst powiązany z formantem.

Procedury zwrotne VBA mogą być dowolnie złożone.

Plik .rels

Wstawienie pliku zawierającego kod RibbonX nie daje żadnego efektu, dopóki nie zostaną zdefiniowane relacje pomiędzy plikiem dokumentu i plikiem zawierającym modyfikacje Wstążki. Opis relacji, w formacie *XML*, jest przechowywany w pliku *.rels*, znajdującym się w folderze o nazwie *_rels*. Poniżej przedstawiono zapis relacji dla przykładu omawianego w poprzedniej sekcji:

```
<Relationship Type="http://schemas.microsoft.com/office/2006/
relationships/ui/extensibility" Target="/customUI/customUI.xml"
Id="12345" />
```

Parametr Target wskazuje na plik *customUI.xml* zawierający kod RibbonX. Parametr Id zawiera dowolny ciąg znaków, który musi być unikatowy dla tego pliku (inaczej mówiąc, żaden inny znacznik *<Relationship>* nie może używać takiego samego ciągu Id).

Jeżeli używasz edytora *Custom UI Editor for Microsoft Office*, nie musisz się przejmować plikiem *.rels*, gdyż wszystkie niezbędne zmiany w tym pliku dokonywane są automatycznie.

Kod RibbonX

A teraz czas na element kluczowy element naszej układanki. Tworzenie kodu XML definiującego modyfikacje interfejsu użytkownika nie jest prostym zadaniem. Jak już wspominaliśmy, w tej książce nie będziemy omawiać zagadnień związanych z tworzeniem kodu RibbonX. Znajdziesz tutaj jedynie kilka prostych przykładów, ale bardziej szczegółowych informacji będziesz musiał poszukać w innych źródłach.

Kiedy rozpoczynasz samodzielne próby modyfikacji Wstążki, powinieneś zacząć od gotowych, działających przykładów dostępnych w sieci Internet i następnie dokonywać małych modyfikacji i często sprawdzać ich efekty. Poświęcenie wielu godzin na analizowanie wyglądającego perfekcyjnie — a mimo to generującego błędy — kodu tylko po to, aby się na koniec przekonać, że w XML-u rozróżniane są małe i wielkie litery, może być naprawdę frustrującym doświadczeniem. ID w kodzie XML to nie to samo, co Id.

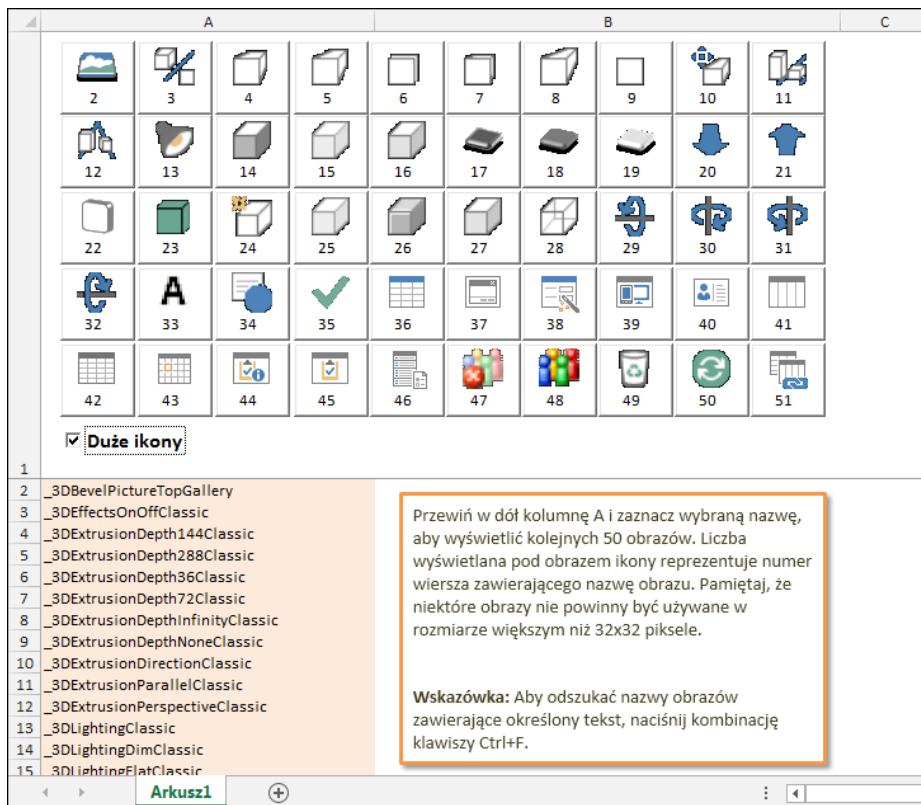


W pewne zakłopotanie może wprowadzić Cię parametr *imageMso*, odpowiadający za obraz, który zostanie umieszczony na przycisku Wstążki. W pakiecie Microsoft Office masz do dyspozycji ponad 1000 obrazów, których możesz użyć jako ikon przycisków na Wstążce. Każdy z obrazów posiada swoją unikatową nazwę. Więcej szczegółowych informacji na ten temat znajdziesz w ramce „Korzystanie z obrazów *imageMso*”.

Korzystanie z obrazów imageMso

Microsoft Office 2010 posiada ponad 1000 obrazów ikon powiązanych z różnymi poleceniami. Do umieszczania własnych przycisków na Wstążce możesz używać dowolnego z tych obrazów, pod warunkiem oczywiście, że znasz ich nazwy.

Na rysunku poniżej przedstawiono wygląd skoroszytu zawierającego wszystkie dostępne obrazy imageMso wraz z nazwami. Jednorazowo na ekranie możesz wyświetlić około 50 obrazów wraz z nazwami. Skoroszyt z obrazami (*Przeglądanie obrazów imageMso.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).



Obrazy imageMso możesz również umieszczać w formancie Image na formularach *UserForm*. Poleceń przedstawione poniżej przypisuje obraz imageMso o nazwie *ReviewAcceptChanges* do właściwości Picture formantu Image o nazwie *Image1*, umieszczonego na formularzu *UserForm*. Rozmiar obrazu został zdefiniowany jako 32×32 piksele.

```
Image1.Picture = Application.CommandBars.  
GetImageMso("ReviewAcceptChange", 32, 32)
```

Kolejny przykład kodu RibbonX

W tej sekcji zaprezentujemy kolejny przykład użycia kodu RibbonX do modyfikacji Wstążki. Przykładowy skoroszyt tworzy nową grupę na karcie *Układ strony* i dodaje pole wyboru odpowiadające za przełączanie wyświetlanego znaczników podziału stron.



Uwaga

Pomimo iż Excel posiada ponad 1700 poleceń, nie ma wśród nich polecenia przełączającego wyświetlanie znaczników podziału stron. Po wydrukowaniu arkusza lub włączeniu funkcji podglądu wydruku jedynym sposobem ukrycia znaczników podziału stron jest użycie odpowiedniej opcji w oknie *Opcje programu Excel*, stąd nasz przykład może być bardzo użyteczny w praktyce.

Przytoczony przykład jest dosyć ciekawy, ponieważ wymaga, aby nowy formant Wstążki był zsynchronizowany z aktywnym arkuszem. Na przykład: jeżeli aktywujesz arkusz, na którym znaczniki podziału stron nie są wyświetlane, pole wyboru nie powinno być zaznaczone. Jeżeli aktywujesz arkusz, na którym znaczniki podziału stron są wyświetlane, pole wyboru powinno być zaznaczone. Dodatkowo znaczniki podziału strony nie odnoszą się do arkuszy wykresów, zatem po aktywowaniu takiego arkusza pole wyboru powinno być zablokowane.

Kod RibbonX

Kod RibbonX przedstawiony poniżej tworzy nową grupę poleceń na karcie *Układ strony* i umieszcza w niej formant CheckBox:

```
<customUI  
    xmlns="http://schemas.microsoft.com/office/2006/01/customui"  
    onLoad="Initialize">  
    <ribbon>  
        <tabs>  
            <tab idMso="TabPageLayoutExcel">  
                <group id="Group1" label="Inne">  
                    <checkbox id="Checkbox1"  
                        label="Znaczniki podziału strony"  
                        onAction="TogglePageBreakDisplay"  
                        getPressed="GetPressed"  
                        getEnabled="GetEnabled"/>  
                </group>  
            </tab>  
        </tabs>  
    </ribbon>  
</customUI>
```

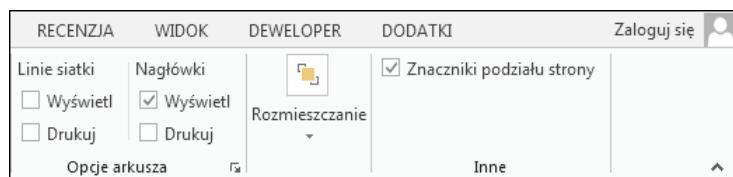
Przedstawiony kod RibbonX zawiera odwołania do czterech procedur zwrotnych VBA (które opiszę nieco później):

- Initialize — wykonywana w momencie otwarcia skoroszytu.
- TogglePageBreakDisplay — wykonywana po kliknięciu pola wyboru przez użytkownika.
- GetPressed — wykonywana, kiedy formant zostaje unieważniony (użytkownik aktywuje inny arkusz).
- GetEnabled — wykonywana, kiedy formant zostaje unieważniony (użytkownik aktywuje inny arkusz).

Na rysunku 20.8 przedstawiono wygląd nowego formantu Wstążki.

Rysunek 20.8.

To pole wyboru jest zsynchonizowane z wyświetlaniem znaczników podziału stron w aktywnym skoroszycie



Kod VBA

Znacznik CustomUI zawiera parametr onLoad, definiujący odwołanie do procedury zwrotnej Initialize, której kod zamieszczono poniżej (kod procedury jest umieszczony w standardowym module VBA):

```
Public MyRibbon As IRibbonUI
Sub Initialize(Ribbon As IRibbonUI)
    ' Wykonywana podczas otwierania skoroszytu
    Set MyRibbon = Ribbon
End Sub
```

Procedura Initialize tworzy obiekt klasy IRibbonUI o nazwie MyRibbon. Zauważ, że MyRibbon został zadeklarowany jako zmienna publiczna, dzięki czemu jest dostępny dla innych procedur w tym module.

Następnie utworzyliśmy prostą procedurę obsługi zdarzeń, która jest wykonywana za każdym razem, kiedy arkusz jest aktywowany. Procedura jest zlokalizowana w module kodu ThisWorkbook i wywołuje procedurę CheckPageBreakDisplay.

```
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    Call CheckPageBreakDisplay
End Sub
```

Procedura CheckPageBreakDisplay *unieważnia* pole wyboru, czyli innymi słowy, usuwa wszystkie dane związane z tym formantem.

```
Sub CheckPageBreakDisplay()
    ' Wykonywana, kiedy arkusz jest aktywowany
    MyRibbon.InvalidateControl ("Checkbox1")
End Sub
```

Kiedy formant zostanie unieważniony, wywoływane są procedury GetPressed oraz GetEnabled.

```
Sub GetPressed(control As IRibbonControl, ByRef returnedVal)
    ' Wykonywana po unieważnieniu pola wyboru
    On Error Resume Next
    returnedVal = ActiveSheet.DisplayPageBreaks
End Sub
```

```
Sub GetEnabled(control As IRibbonControl, ByRef returnedVal)
    ' Wykonywana po unieważnieniu pola wyboru
    returnedVal = TypeName(ActiveSheet) = "Worksheet"
End Sub
```

Zauważ, że argument returnedVal jest przekazywany przez odwołanie (ByRef). Oznacza to, że kod procedury jest w stanie zmienić jego wartość — i dokładnie tak się dzieje. W procedurze GetPressed zmiennej returnedVal jest przypisywana wartość właściwości DisplayPageBreaks aktywnego arkusza. W rezultacie parametr Pressed formantu ma wartość True, jeżeli znaczniki podziału stron są wyświetlane (i pole wyboru jest zaznaczone). W przeciwnym wypadku pole wyboru nie jest zaznaczone.

W procedurze GetEnabled zmieniona jest ustawiana na wartość True, jeżeli aktywnym arkuszem jest arkusz danych (a nie arkusz wykresu), stąd formant pola wyboru jest aktywny tylko wtedy, kiedy aktywnym arkuszem jest arkusz danych.

Ostatnią procedurą VBA jest zdefiniowana przez parametr onAction procedura Toggle →PageBreak, która jest wykonywana, kiedy użytkownik zaznacza lub usuwa zaznaczenie pola wyboru:

```
Sub TogglePageBreakDisplay(control As IRibbonControl, pressed As Boolean)
    ' Wykonywana po kliknięciu pola wyboru
    On Error Resume Next
    ActiveSheet.DisplayPageBreaks = pressed
End Sub
```

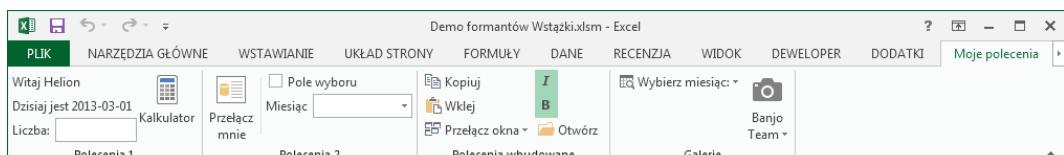
Argument pressed ma wartość True, kiedy użytkownik zaznaczy pole wyboru, a wartość False, kiedy usunie zaznaczenie tego pola. Kod procedury odpowiednio ustawia wartość właściwości DisplayPageBreaks.



Skoroszyt z tym przykładem (*Wyświetlanie znaczników podziału stron.xlsxm*) został zamieszczony na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Na stronie internetowej znajdziesz również dodatek utworzony na podstawie tego skoroszytu (o nazwie *Dodatek — Wyświetlanie znaczników podziału stron.xlsxm*), który udostępnia nowe polecenie Wstążki dla wszystkich skoroszytów. Dodatek wykorzystuje moduł klasy do monitorowania zdarzeń związanych z aktywacją arkusza dla wszystkich skoroszytów. Więcej szczegółowych informacji na temat zdarzeń znajdziesz w rozdziale 17., a więcej szczegółowych informacji na temat modułów klas zamieszczone w rozdziale 27.

Demo formantów Wstążki

Na rysunku 20.9 przedstawiono wygląd nowej karty Wstążki, o nazwie *Moje polecenia*, na której zostały utworzone cztery grupy formantów. W podrozdziale omówiono kod RibbonX tego przykładu i powiązane z nim procedury zwrotne VBA.



Rysunek 20.9. Nowa karta Wstążki zawierająca cztery grupy formantów



Skoroszyt z tym przykładem (*Demo formantów Wstążki.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Tworzenie nowej karty

Kod RibbonX tworzący nową kartę Wstążki wygląda następująco:

```
<ribbon>
<tabs>
<tab id="CustomTab" label="Moje polecenia">
</tab>
</tabs>
</ribbon>
```



Jeżeli chcialibyś utworzyć zminimalizowaną wersję interfejsu użytkownika, powinieneś użyć atrybutu startFromScratch znacznika ribbon. Jeżeli nadasz temu atrybutowi wartość True, wszystkie wbudowane karty Wstążki zostaną ukryte.

```
<ribbon startFromScratch="true" >
```

Tworzenie nowej grupy

Skoroszyt *Demo formantów Wstążki.xlsx* tworzy na karcie *Moje narzędzia* cztery grupy poleceń. Kod odpowiedzialny za tę operację wygląda następująco:

```
<group id="Group1" label="Polecenia 1">
</group>
<group id="Group2" label="Polecenia 2">
</group>
<group id="Group3" label="Polecenia wbudowane">
</group>
<group id="Group4" label="Galerie">
</group>
```

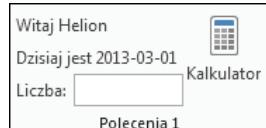
Parы znaczników `<group> </group>` są zlokalizowane wewnętrz znaczników `<tab> </tab>`, które są odpowiedzialne za utworzenie nowej karty.

Tworzenie formantów

Poniżej przedstawiono kod RibbonX, który tworzy formanty pierwszej grupy (*Polecenia 1*), pokazanej na rysunku 20.10. Zwróć uwagę, że formanty zostały zdefiniowane w pierwszej grupie znaczników `<group> </group>`.

Rysunek 20.10.

Nowa grupa, w której umieszczono cztery formanty



```
<group id="Group1" label="Polecenia 1">
<labelControl id="Label1" getLabel="getLabel1" />
<labelControl id="Label2" getLabel="getLabel2" />

<editBox id="EditBox1"
showLabel="true"
label="Liczba:"
onChange="EditBox1_Change"/>
<button id="Button1"
```

```
label="Kalkulator"
size="large"
onAction="ShowCalculator"
imageMso="Calculator" />
</group>
```

Dwa formanty typu Label posiadają powiązane procedury zwrotne VBA (o nazwach odpowiednio getLabel1 oraz getLabel2). Kod tych procedur został zamieszczony poniżej:

```
Sub getLabel1(control As IRibbonControl, ByRef returnedVal)
    returnedVal = "Witaj " & Application.UserName
End Sub
Sub getLabel2(control As IRibbonControl, ByRef returnedVal)
    returnedVal = "Dzisiejsza data: " & Date
End Sub
```

Po załadowaniu kodu RibbonX te dwie procedury są wykonywane i etykiety formantów Label są dynamicznie aktualizowane i wyświetlają nazwę użytkownika i bieżącą datę.

Formant editBox posiada procedurę zwrotną obsługi zdarzenia onChange, o nazwie EditBox1_Change, która wyświetla wartość pierwiastka kwadratowego wartości wprowadzonej w polu EditBox1 (lub komunikat o błędzie, jeżeli wartość pierwiastka nie może zostać obliczona). Kod procedury EditBox1_Change wygląda następująco:

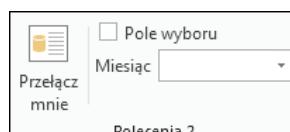
```
Sub EditBox1_Change(control As IRibbonControl, text As String)
    Dim squareRoot As Double
    On Error Resume Next
    squareRoot = Sqr(text)
    If Err.Number = 0 Then
        MsgBox "Wartość pierwiastka kwadratowego liczby " & text & " to: "
        =>& squareRoot
    Else
        MsgBox "Wpisz liczbę dodatnią.", vbCritical
    End If
End Sub
```

Ostatnim formantem w grupie *Polecenia* jest prosty przycisk, którego parametr onAction wywołuje procedurę VBA o nazwie ShowCalculator. Procedura do uruchomienia kalkulatora systemu Windows wykorzystuje funkcję Shell.

```
Sub ShowCalculator(control As IRibbonControl)
    On Error Resume Next
    Shell "calc.exe", vbNormalFocus
    If Err.Number <> 0 Then MsgBox "Nie mogę uruchomić kalkulatora!"
End Sub
```

Na rysunku 20.11 przedstawiono formanty drugiej grupy, o nazwie *Polecenia 2*.

Rysunek 20.11.
Trzy nowe formanty
umieszczone
na Wstążce



Kod RibbonX tworzący drugą grupę przycisków wygląda następująco:

```
<group id="Group2" label="Polecenia 2">
<toggleButton id="ToggleButton1"
    size="large"
    imageMso="FileManageMenu"
    label="Przełącz mnie"
    onAction="ToggleButton1_Click" />
<separator id="sep1" />
<checkBox id="Checkbox1"
    label="Pole wyboru"
    onAction="Checkbox1_Change"/>
<comboBox id="Combo1"
    label="Miesiąc"
    onChange="Combo1_Change">
    <item id="Month1" label="Styczeń" />
    <item id="Month2" label="Luty" />
    <item id="Month3" label="Marzec" />
    <item id="Month4" label="Kwiecień" />
    <item id="Month5" label="Maj" />
    <item id="Month6" label="Czerwiec" />
    <item id="Month7" label="Lipiec" />
    <item id="Month8" label="Sierpień" />
    <item id="Month9" label="Wrzesień" />
    <item id="Month10" label="Październik" />
    <item id="Month11" label="Listopad" />
    <item id="Month12" label="Grudzień" />
</comboBox>
</group>
```

W tej grupie poleceń znajdziesz formanty takie jak `toggleButton`, `separator`, `checkbox` oraz `comboBox`. Nie ma tutaj niczego nadzwyczajnego. Wszystkie formanty — z wyjątkiem formantu `separator`, który wstawia pionową linię oddzielającą — posiadają przypisane procedury zwrotne, które po prostu wyświetlają na ekranie status danego formantu.

```
Sub ToggleButton1_Click(control As IRibbonControl, ByRef returnedVal)
    MsgBox "Stan przełącznika: " & returnedVal
End Sub

Sub Checkbox1_Change(control As IRibbonControl, pressed As Boolean)
    MsgBox "Wartość pola wyboru: " & pressed
End Sub

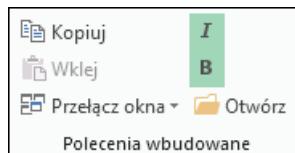
Sub Combo1_Change(control As IRibbonControl, text As String)
    MsgBox text
End Sub
```



W polu `comboBox` możesz również wpisywać swój własny tekst. Jeżeli chcesz, aby możliwości wyboru zostały zredukowane tylko do wartości z predefiniowanej listy, powinieneś użyć formantu `dropDown`.

W trzeciej grupie nowej karty znajdują się formanty wybranych, wbudowanych poleceń Excela (patrz rysunek 20.12). Aby w niestandardowej grupie poleceń użyć formantu wbudowanego, musisz znać jego nazwę (parametr `idMso`).

Rysunek 20.12.
W tej grupie znajdziesz kilka formantów wbudowanych



Kod RibbonX tworzący trzecią grupę wygląda następująco:

```
<group id="Group3" label="Polecenia wbudowane">
    <control idMso="Copy" label="Kopiuj" />
    <control idMso="Paste" label="Wklej" enabled="true" />
    <control idMso="WindowSwitchWindowsMenuExcel"
        label="Przełącz okna" />
    <control idMso="Italic" />
    <control idMso="Bold" />
    <control idMso="FileOpen" />
</group>
```

Formanty w tej grupie nie posiadają procedur zwrotnych, ponieważ wykonują standardowe, wbudowane polecenia Excela.

Na rysunku 20.13 przedstawiono wygląd ostatniej, czwartej grupy, składającej się z dwóch galerii.

Rysunek 20.13.
W tej grupie polecień znajdziesz dwie galerie



Kod RibbonX tworzący wspomniane dwie galerie w czwartej grupie poleczeń jest następujący:

```
<group id="Group4" label="Galerie">
    <gallery id="Gallery1"
        imageMso="ViewAppointmentInCalendar"
        label="Wybierz miesiąc:"
        columns="2" rows="6"
        onAction="MonthSelected" >
        <item id="January" label="Styczeń" imageMso="QuerySelectQueryType"/>
        <item id="February" label="Luty" imageMso="QuerySelectQueryType"/>
        <item id="March" label="Marzec" imageMso="QuerySelectQueryType"/>
        <item id="April" label="Kwiecień" imageMso="QuerySelectQueryType"/>
        <item id="May" label="Maj" imageMso="QuerySelectQueryType"/>
        <item id="June" label="Czerwiec" imageMso="QuerySelectQueryType"/>
        <item id="July" label="Lipiec" imageMso="QuerySelectQueryType"/>
        <item id="August" label="Sierpień" imageMso="QuerySelectQueryType"/>
        <item id="September" label="Wrzesień" imageMso="QuerySelectQueryType"/>
        <item id="October" label="Październik" imageMso="QuerySelectQueryType"/>
        <item id="November" label="Listopad" imageMso="QuerySelectQueryType"/>
        <item id="December" label="Grudzień" imageMso="QuerySelectQueryType"/>
        <button id="Today"
            label="Dzisiaj..."
            imageMso="ViewAppointmentInCalendar"
            onAction="ShowToday"/>
    </gallery>
<gallery id="Gallery2"
        imageMso="ViewImageInImageList"
        label="Galerie"
        columns="2" rows="3"
        onAction="ImageListImageSelected" >
        <item id="ImageListImage1" imageMso="ImageListImage1" />
        <item id="ImageListImage2" imageMso="ImageListImage2" />
        <item id="ImageListImage3" imageMso="ImageListImage3" />
    </gallery>
</group>
```

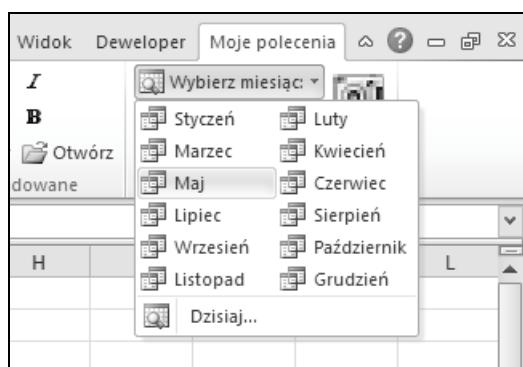
```

</gallery>
<gallery id="Gallery2"
    label="Banjo Team"
    size="large"
    columns="4"
    itemWidth="100" itemHeight="125"
    imageMso= "Camera"
    onAction="OnAction">
    <item id="bp01" image="bp01" />
    <item id="bp02" image="bp02" />
    <item id="bp03" image="bp03" />
    <item id="bp04" image="bp04" />
    <item id="bp05" image="bp05" />
    <item id="bp06" image="bp06" />
    <item id="bp07" image="bp07" />
    <item id="bp08" image="bp08" />
    <item id="bp09" image="bp09" />
    <item id="bp10" image="bp10" />
    <item id="bp11" image="bp11" />
    <item id="bp12" image="bp12" />
    <item id="bp13" image="bp13" />
    <item id="bp14" image="bp14" />
    <item id="bp15" image="bp15" />
</gallery>
</group>

```

Na rysunku 20.14 przedstawiono wygląd pierwszej galerii, wyświetlającej w dwóch kolumnach listę nazw miesięcy. Parametr `onAction` powoduje wykonanie procedury zwrotnej `MonthSelected`, która wyświetla nazwę wybranego miesiąca (przechowywaną przez parametr `id`).

Rysunek 20.14.
Galeria nazw miesięcy
(zawiera dodatkowy
przycisk)



```

Sub MonthSelected(control As IRibbonControl,
    id As String, index As Integer)
    MsgBox "Wybrałeś " & id
End Sub

```

W dolnej części galerii *Wybierz miesiąc* znajdziesz również przycisk *Dzisiaj*, który posiada swoją własną procedurę zwrotną:

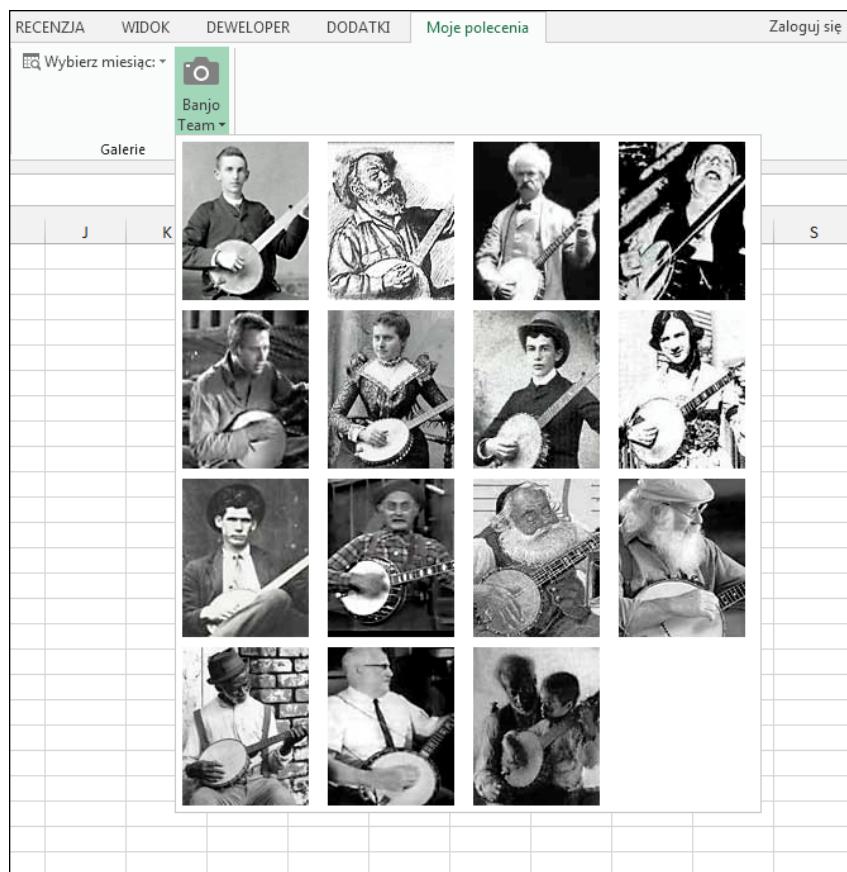
```

Sub ShowToday(control As IRibbonControl)
    MsgBox "Dzisiaj jest " & Date
End Sub

```

Druga galeria, przedstawiona na rysunku 20.15, wyświetla 15 zdjęć.

Rysunek 20.15.
Galeria zdjęć



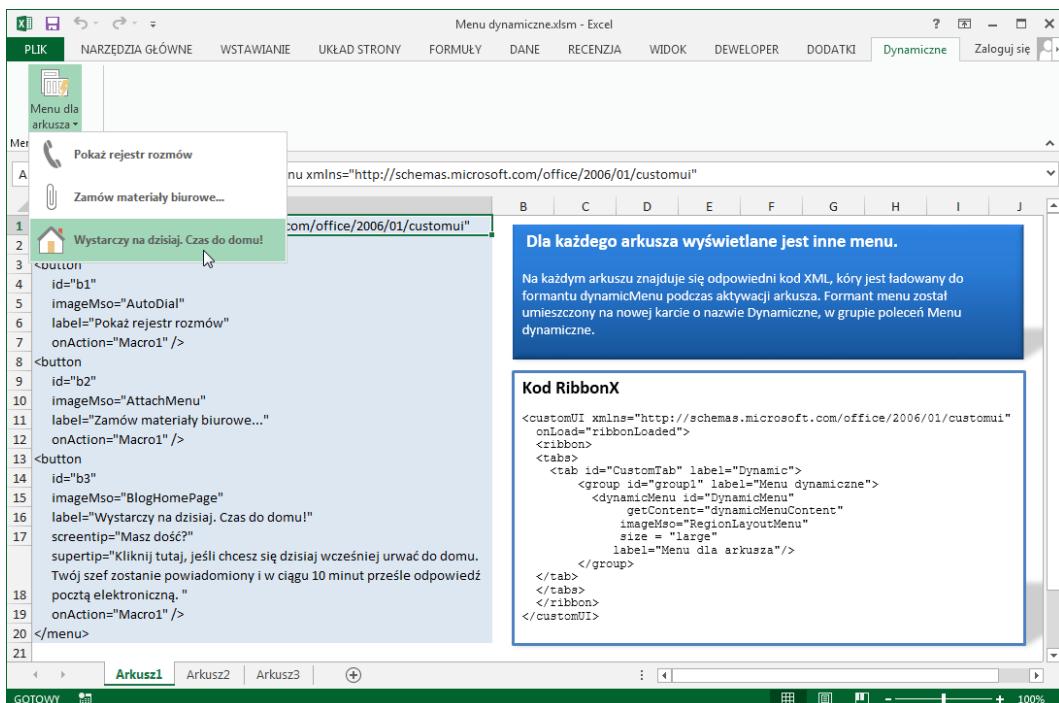
Zdjęcia są przechowywane w pliku skoroszytu, w folderze *customUI/images*. Dołożenie obrazów wymaga również obecności folderu *_rels*, zawierającego listę relacji między plikami. Aby przekonać się, jak to działa, dodaj do pliku skoroszytu rozszerzenie *.zip* i zapoznaj się z jego zawartością.

Przykład użycia formantu DynamicMenu

Jednym z najbardziej interesujących formantów Wstążki jest formant `dynamicMenu`, który pozwala na przekazywanie za pośrednictwem VBA danych XML do formantu — a takie rozwiązanie jest bazą do utworzenia menu kontekstowego, zmieniającego zawartość w zależności od kontekstu, w jakim zostało użyte.

Tworzenie i konfiguracja formantów `dynamicMenu` nie jest prostym zadaniem, ale z punktu widzenia VBA to właśnie ten formant oferuje największą elastyczność podczas dynamicznego modyfikowania Wstążki.

Aby zilustrować działanie formantu dynamicMenu, utworzyłem skoroszyt, który wyświetla inne menu dla każdego z trzech arkuszy znajdujących się w tym skoroszycie. Na rysunku 20.16 przedstawiono wygląd menu, które pojawia się po uaktywnieniu arkusza o nazwie Arkusz1. Kiedy dany arkusz zostaje aktywowany, procedura VBA wysyła kod XML odpowiedni dla danego arkusza. Na potrzeby tego przykładu kod XML jest przechowywany bezpośrednio na arkuszach, tak aby łatwiej można go było przeanalizować. W praktyce zamiast tego kod XML może być przechowywany w odpowiedniej zmiennej tekstowej w kodzie programu.



Rysunek 20.16. Formant dynamicMenu pozwala na tworzenie menu, którego zawartość zmienia się w zależności od kontekstu

Kod RibbonX tworzący nową kartę, nową grupę i wreszcie sam formant dynamicMenu wygląda następująco:

```

<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <onLoad="ribbonLoaded">
    <ribbon>
      <tabs>
        <tab id="CustomTab" label="Dynamiczne">
          <group id="group1" label="Pokaz menu dynamicznego">
            <dynamicMenu id="DynamicMenu"
              getContent="dynamicMenuContent"
              imageMso="RegionLayoutMenu"
              size = "large"
              label="Menu inne dla każdego arkusza"/>
          </group>
        </tab>
      </tabs>
    </ribbon>
  </onLoad>
</customUI>

```

```
</tabs>
</ribbon>
</customUI>
```

W przykładzie musimy mieć możliwość unieważnienia Wstążki za każdym razem, kiedy użytkownik aktywuje nowy arkusz. Do realizacji takiego zadania użyłem tej samej techniki, z której korzystaliśmy w przykładzie dotyczącym wyświetlania znaków podziału strony (patrz podrozdział „Kolejny przykład kodu RibbonX”) zadeklarowałem publiczną zmienną obiektową MyRibbon klasy IRibbonUI. Za każdym razem, kiedy aktywowany jest nowy arkusz, używamy procedury obsługi zdarzenia Workbook_SheetActivate, która wywołuje procedurę UpdateDynamicRibbon.

```
Sub UpdateDynamicRibbon()
    ' Unieważnia Wstążkę w celu wymuszenia wywołania procedury zwrotnej dynamicMenuContent
    On Error Resume Next
    MyRibbon.Invalidate
    If Err.Number <> 0 Then
        MsgBox "Utracono obiekt reprezentujący Wstążkę. Zapisz skoroszyt, zamknij
        → i ponownie otwórz"
    End If
End Sub
```

Procedura UpdateDynamicRibbon unieważnia obiekt MyRibbon, który wymusza wywołanie procedury zwrotnej VBA o nazwie dynamicMenuContent (procedura, do której odwołuje się parametr getContent w kodzie RibbonX). Zwróć uwagę na kod obsługujący błędy. Pewne modyfikacje kodu VBA mogą spowodować zniszczenie obiektu MyRibbon, który jest tworzony podczas otwierania skoroszytu. Próba unieważnienia obiektu, który nie istnieje, powoduje wygenerowanie błędu i wyświetlenie komunikatu informującego, że skoroszyt powinien zostać zapisany, zamknięty i ponownie otwarty. Niestety, zamknięcie i ponowne otwarcie skoroszytu jest jedyną metodą ponownego utworzenia obiektu MyRibbon.

Poniżej przedstawiono kod procedury dynamicMenuContent. Procedura przechodzi w pętli przez komórki kolumny A aktywnego skoroszytu, odczytuje kod XML i zapamiętuje go w zmiennej o nazwie XMLcode. Kiedy cały kod XML zostanie dołączony do zmiennej, jest przekazywany do argumentu returnedVal. Efektem takiego działania jest to, że formant dynamicMenu otrzymuje nowy kod i dzięki temu wyświetla inny zestaw poleceń menu.

```
Sub dynamicMenuContent(control As IRibbonControl, _
    ByRef returnedVal)
    Dim r As Long
    Dim XMLcode As String
    ' Odczytaj kod XML z aktywnego skoroszytu
    For r = 1 To Application.CountA(Range("A:A"))
        XMLcode = XMLcode & ActiveSheet.Cells(r, 1) & " "
    Next r
    returnedVal = XMLcode
End Sub
```

Skoroszyt z tym przykładem (*Menu dynamiczne.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Więcej wskazówek dotyczących modyfikacji Wstążki

Na zakończenie podrozdziału przedstawię kilka dodatkowych wskazówek, o których powinieneś pamiętać podczas samodzielnej eksploracji i modyfikacji Wstążki programu Excel:

- Kiedy pracujesz ze Wstążką, upewnij się, że włączyłeś wyświetlanie komunikatów o błędach. Więcej szczegółowych informacji na ten temat znajdziesz w ramce „Wyświetlanie błędów” we wcześniejszej części tego rozdziału.
- Pamiętaj, że w kodzie RibbonX rozróżniane są małe i wielkie litery.
- Identyfikatory ID wszystkich formantów bazują na języku angielskim i są takie same we wszystkich wersjach językowych Excela. Dzięki takiemu rozwiązaniu modyfikacje Wstążki działają zawsze i wszędzie, niezależnie od tego, jakiej wersji językowej Excela używasz.
- Modyfikacje Wstążki są widoczne tylko wtedy, kiedy skoroszyt zawierający kod RibbonX jest aktywny. Aby modyfikacje Wstążki były widoczne dla wszystkich skoroszytów, kod RibbonX musi znajdować się w dodatku.
- Formanty wbudowane automatycznie zmieniają szerokość, adaptując się do zmian szerokości okna Excela. W Excelu 2007 formanty niestandardowe nie są skalowane, ale w Excelu 2010 i nowszych już tak.
- Dodawanie i usuwanie wbudowanych formantów Wstążki nie jest możliwe.
- Możesz ukrywać wybrane karty Wstążki. Na przykład kod RibbonX, który przedstawiono poniżej, ukrywa trzy karty Wstążki:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon>
    <tabs>
      <tab idMso="TabPageLayoutExcel" visible="false" />
      <tab idMso="TabData" visible="false" />
      <tab idMso="TabReview" visible="false" />
    </tabs>
  </ribbon>
</customUI>
```

- W razie potrzeby możesz również ukrywać wybrane grupy poleceń znajdujące się na danej karcie. Kod RibbonX przedstawiony poniżej ukrywa cztery grupy poleceń znajdujących się na karcie *WSTAWIANIE*, pozostawiając widoczną jedynie grupę *Wykresy*.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
  <ribbon>
    <tabs>
      <tab idMso="TabInsert">
        <group idMso="GroupInsertTablesExcel" visible="false" />
        <group idMso="GroupInsertIllustrations" visible="false" />
        <group idMso="GroupInsertLinks" visible="false" />
        <group idMso="GroupInsertText" visible="false" />
      </tab>
    </tabs>
  </ribbon>
</customUI>
```

- Do formantu wbudowanego możesz przypisać dowolne makro. Takie postępowanie jest określane mianem *zmiany przeznaczenia formantu*. Przykładowo kod RibbonX przedstawiony poniżej przechwytyuje trzy polecenia wbudowane:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<commands>
    <command idMso="FileSave" onAction="mySave"/>
    <command idMso="FilePrint" onAction="myPrint"/>
    <command idMso="FilePrintQuick" onAction="myPrint"/>
</commands>
</customUI>
```

- Możesz również napisać kod, który blokuje jeden lub więcej wbudowanych formantów. Poniżej przedstawiono przykładowy kod RibbonX, który blokuje przycisk *Obiekt clipart*, znajdujący się na karcie *WSTAWIANIE*:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<commands>
    <command idMso="ClipArtInsert" enabled="false"/>
</commands>
</customUI>
```

- Jeżeli masz dwa lub więcej skoroszytów (lub dodatki), które dodają formanty do tej samej grupy polecień na Wstążce, musisz upewnić się, czy korzystają z tej samej przestrzeni nazw. Aby to zrobić, sprawdź zawartość znacznika *<CustomUI>*, który znajdziesz na początku kodu RibbonX.

Tworzenie pasków narzędzi w starym stylu

Jeżeli dojdziesz do wniosku, że dostosowywanie Wstążki do własnych potrzeb wymaga zbyt wielkiego nakładu pracy, być może wystarczające okaże się utworzenie prostego, bardziej tradycyjnego paska narzędzi przy użyciu poprzedniej wersji obiektu CommandBar (przed Exceliem 2007) Taka technika znakomicie nadaje się do użycia w dowolnym skoroszycie, z którego będziesz korzystał. Utworzenie takiego paska to również prosty sposób na zapewnienie szybkiego dostępu do wielu makr.

W tym podrozdziale omówię przykładowy kod, który z łatwością możesz zaadaptować do własnych potrzeb. Nie będę się jednak zbytnio zagłębiać w opisywanie tej techniki. Więcej szczegółowych informacji na temat obiektów CommandBar znajdziesz w zasobach sieci Internet lub w wydaniu tej książki omawiającej Excela 2003. Pamiętaj, że obiekty CommandBar potrafią znacznie więcej, niż zostało to zaprezentowane na naszych przykładach.

Ograniczenia funkcjonalności tradycyjnych pasków narzędzi w Excelu 2007 i nowszych wersjach

Decydując się na tworzenie tradycyjnego paska narzędzi w Excelu 2007 lub nowszym, powinieneś pamiętać o następujących ograniczeniach:

- Nie możesz tworzyć pasków swobodnie „pływających”.
- Paski zawsze będą wyświetlane na karcie *Dodatki*, w grupie poleceń *Niestandardowe paski narzędzi* (razem z innymi paskami narzędzi).

- Niektóre właściwości i metody obiektu CommandBar są przez Excela po prostu ignorowane.

Kod tworzący pasek narzędzi

Kod przedstawiony w tej sekcji zakłada, że masz skoroszyt zawierający dwa makra (o nazwach Macro1 oraz Macro2). Oprócz tego zakładam również, że chcesz, aby Twój pasek narzędzi był tworzony podczas otwierania skoroszytu i usuwany po jego zamknięciu.



Jeżeli korzystasz z Excela 2007 lub Excela 2010, niestandardowe paski narzędzi są widoczne zawsze, niezależnie od tego, który skoroszyt jest w danej chwili aktywny. Jednak w Excelu 2013 taki pasek narzędzi będzie widoczny tylko w skoroszycie, w którym został utworzony, oraz w nowych skoroszytach, utworzonych, kiedy oryginalny skoroszyt był aktywny.

Procedury, których kod przedstawiono poniżej, powinieneś umieścić w module kodu ThisWorkbook. Pierwsza procedura wywołuje inną procedurę, która podczas otwierania skoroszytu tworzy nowy pasek narzędzi. Druga procedura wywołuje procedurę, która po zamknięciu skoroszytu usuwa pasek narzędzi.

```
Private Sub Workbook_Open()
    Call CreateToolbar
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteToolbar
End Sub
```



W rozdziale 17. opisano potencjalnie bardzo istotny problem ze zdarzeniem Workbook_BeforeClose. Komunikat *Czy chcesz zapisać...* pojawia się na ekranie już po zakończeniu działania procedury obsługi zdarzenia Workbook_BeforeClose, zatem jeżeli użytkownik naciśnie przycisk *Anuluj*, skoroszyt pozostanie otwarty, ale niestandardowe elementy menu zostaną już wcześniej usunięte. W rozdziale 17. znajdziesz również sposób na ominięcie tego problemu.

Kod procedury CreateToolbar przedstawiono poniżej:

```
Const TOOLBARNAME As String = "Mój_pasek"

Sub CreateToolbar()
    Dim TBar As CommandBar
    Dim Btn As CommandBarButton

    ' Usunięcie istniejącego paska narzędzi (jeżeli istnieje)
    On Error Resume Next
    CommandBars(TOOLBARNAME).Delete
    On Error GoTo 0

    ' Tworzenie nowego paska narzędzi
    Set TBar = CommandBars.Add
    With TBar
        .Name = TOOLBARNAME
        .Visible = True
    End With
```

```

    ' Dodaj przycisk
    Set Btn = TBar.Controls.Add(Type:=msoControlButton)
    With Btn
        .FaceId = 300
        .OnAction = "Macro1"
        .Caption = "Tutaj możesz umieścić opis procedury Macro1"
    End With

    ' Dodaj kolejny przycisk
    Set Btn = TBar.Controls.Add(Type:=msoControlButton)
    With Btn
        .FaceId = 25
        .OnAction = "Macro2"
        .Caption = " Tutaj możesz umieścić opis procedury Macro2 "
    End With
End Sub

```



Skoroszyt z tym przykładem (*Tradycyjny pasek narzędzi.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Na rysunku 20.17 przedstawiono tradycyjny pasek narzędzi, na którym zostały umieszczone dwa przyciski.



Rysunek 20.17. Tradycyjny pasek narzędzi, znajdujący się na karcie Dodatki, w grupie opcji Niestandardowe paski narzędzi

W programie wykorzystujemy zmienną modułową o nazwie TOOLBAR, która przechowuje nazwę naszego paska narzędzi. Nazwa paska jest wykorzystywana również w procedurze DeleteToolbar, więc użycie stałej zapewnia, że obie procedury pracują z tą samą nazwą paska.

Procedura rozpoczyna działanie od usunięcia istniejącego paska o tej samej nazwie (oczywiście o ile taki pasek już istnieje). Dołączenie tego fragmentu kodu do procedury jest bardzo użyteczne zwłaszcza w fazie wyszukiwania i usuwania błędów, a jednocześnie eliminuje możliwość powstania błędów spowodowanych próbą utworzenia paska narzędzi o takiej samej nazwie.

Pasek narzędzi jest tworzony przy użyciu metody Add obiektu CommandBars. Dwa przyciski są dodawane za pomocą metody Add obiektu Control. Każdy przycisk posiada trzy właściwości:

- FaceID — liczba, która reprezentuje obraz wyświetlany na przycisku. Więcej szczegółowych informacji na temat obrazów FaceID znajdziesz w rozdziale 21.
- OnAction — nazwa makra, które jest uruchamiane po naciśnięciu przycisku.
- Caption — etykieta ekranowa, która pojawia się na ekranie, kiedy ustawisz nad przyciskiem wskaźnik myszy.



Zamiast ustawiać właściwość FaceID, możesz ustawić właściwość Picture na dowolny obraz imageMso. Na przykład polecenie przedstawione poniżej wyświetla zielony znak zapytania:

```
.Picture = Application.CommandBars.GetImageMso _  
("AcceptInvitation", 16, 16)
```

Więcej szczegółowych informacji na temat obrazów imageMso znajdziesz w ramce „Korzystanie z obrazów imageMso”.

Po zamknięciu skoroszytu uruchamiana jest procedura obsługi zdarzenia Workbook_BeforeClose, która wywołuje procedurę DeleteToolbar:

```
Sub DeleteToolbar()  
On Error Resume Next  
CommandBars(TOOLBARNAME).Delete  
On Error GoTo 0  
End Sub
```

Rozdział 21.

Praca z menu podręcznym

W tym rozdziale:

- Jak zidentyfikować menu podręczne
- Jak dostosować menu podręczne do własnych potrzeb
- Jak wyłączyć menu podręczne
- Jak korzystać ze zdarzeń podczas pracy z menu podręcznym
- Jak utworzyć zupełnie nowe menu podręczne

Obiekt CommandBar

Obiekt CommandBar jest wykorzystywany przez trzy elementy interfejsu programu Excel:

- Niestandardowe paski narzędzi.
- Niestandardowe menu.
- Niestandardowe menu podręczne (dostępne po kliknięciu prawym przyciskiem myszy).

Począwszy od Excela 2007, obiekty CommandBar mają dosyć dziwny status. Jeżeli utworzysz procedurę VBA, która będzie modyfikowała menu lub pasek narzędzi, Excel przechwyci ten kod i po prostu zignoruje wiele jego poleceń. Jak wspomniano w rozdziale 20., modyfikacje menu i pasków narzędzi dokonywane za pomocą obiektów CommandBar pojawiają się na karcie *DODATKI* w grupach poleceń *Polecenia menu* lub *Niestandardowe paski narzędzi*, stąd praktyczne zastosowania obiektu CommandBar w Excelu są teraz ograniczone w zasadzie do operacji związanych z menu podręcznym.



Nowy, jednodokumentowy interfejs Excela 2013 bardzo mocno zmienił sposób działania niestandardowych pasków narzędzi. Procedury VBA, które do tej pory były używane do tworzenia takich pasków narzędzi, mogą nie działać poprawnie w Excelu 2013. Więcej szczegółowych informacji na ten temat znajdziesz w dalszej części tego rozdziału (patrz podrozdział „Co nowego w Excelu 2013”).

W tym podrozdziale omówimy podstawowe zagadnienia związane z obiektami CommandBar.

Rodzaje obiektów CommandBar

Excel obsługuje trzy rodzaje obiektów CommandBar, rozróżnianych za pomocą ich właściwości Type, która może przyjmować jedną z trzech wartości:

- msoBarTypeNormal — pasek narzędzi (Type = 0).
- msoBarTypeMenuBar — pasek menu (Type = 1).
- msoBarTypePopUp — menu podręczne (Type = 2).

Nawet jeżeli paski narzędzi i menu nie są używane w Excelu 2007 i w nowszych wersjach, wymienione elementy interfejsu użytkownika nadal są dołączone do modelu obiektowego ze względu na konieczność zachowania kompatybilności ze starszymi aplikacjami. Pamiętaj jednak, że próba wyświetlenia obiektu CommandBar typu 0 lub typu 1 w Excelu nowszym niż wersja 2003 nie daje żadnych efektów. Na przykład polecenie przedstawione poniżej powodowało w Excelu 2003 wyświetlenie standardowego paska narzędzi:

```
CommandBars("Standard").Visible = True
```

W nowszych wersjach Excela takie polecenie jest po prostu ignorowane.

W tym rozdziale skoncentruję się wyłącznie na obiektach CommandBar typu 2 (czyli menu podręcznym).

Wyświetlanie menu podręcznych

W Excelu 2013 istnieje 67 menu podręcznych. Skąd o tym wiem? Po prostu uruchomiłem procedurę ShowShortcutMenuNames, która przechodzi w pętli przez wszystkie istniejące obiekty CommandBar. Jeżeli wartość właściwości Type jest równa msoBarTypePopUp (wbudowana stała o wartości 2), procedura wyświetla w arkuszu wartość indeksu obiektu w kolekcji i nazwę obiektu. Kod procedury zamieszczono poniżej.

```
Sub ShowShortcutMenuNames()
    Dim Row As Long
    Dim cbar As CommandBar
    Row = 1
    For Each cbar In CommandBars
        If cbar.Type = msoBarTypePopUp Then
            Cells(Row, 1) = cbar.Index
            Cells(Row, 2) = cbar.Name
            Cells(Row, 3) = cbar.Controls.Count
            Row = Row + 1
        End If
    Next cbar
End Sub
```

Na rysunku 21.1 przedstawiono fragment wyników działania tej procedury. Wartości indeksów obiektów menu podręcznego znajdują się w zakresie od 22 do 153. Zwróć również uwagę, że nie wszystkie nazwy są unikatowe, na przykład obiekty CommandBar o indeksach 35 i 38 mają taką samą nazwę: Cell. Dzieje się tak, ponieważ kliknięcie komórki prawym przyciskiem myszy w momencie, kiedy arkusz jest wyświetlany w trybie podglądu podziału stron, powoduje wyświetlenie na ekranie innego menu podręcznego niż w widoku normalnym.

Rysunek 21.1.
Proste makro wygenerowało listę wszystkich dostępnych menu podręcznych

	A	B	C	D	E
1	22	PivotChart Menu	6		
2	34	Workbook tabs	16		
3	35	Cell	27		
4	36	Column	13		
5	37	Row	13		
6	38	Cell	21		
7	39	Column	19		
8	40	Row	19		
9	41	Ply	11		
10	42	XLM Cell	15		
11	43	Document	9		
12	44	Desktop	5		
13	45	Nondefault Drag and Drop	11		
14	46	AutoFill	12		
15	47	Button	12		
16	48	Dialog	4		
17	49	Series	5		
18	50	Plot Area	8		



Skoroszyt z tym przykładem (*Pokaż nazwy menu podręcznych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pwv.htm>).

Odwolania do elementów kolekcji CommandBars

Do wybranego obiektu CommandBar możesz się odwołać za pomocą wartości jego indeksu w kolekcji lub za pomocą właściwości Name. Na przykład oba wyrażenia przedstawione poniżej odwołują się do menu podręcznego wyświetlanego w momencie kliknięcia przymytkiem myszy nagłówka kolumny w Excelu 2013:

```
Application.CommandBars (36)
Application.CommandBars("Column")
```

Kolekcja CommandBars jest zawarta w obiekcie Application. Kiedy odwołujesz się do tej kolekcji z poziomu standardowego modułu kodu VBA lub z modułu kodu arkusza, możesz pominiąć odwołanie do obiektu Application. Na przykład polecenie przedstawione poniżej (umieszczone w standardowym module kodu VBA) wyświetla nazwę obiektu o wartości indeksu w kolekcji CommandBars równej 42:

```
MsgBox CommandBars(42).Name
```

Kiedy odwołujesz się do kolekcji CommandBars z modułu kodu obiektu ThisWorkbook, musisz poprzedzić odwołanie nazwą obiektu Application, tak jak to zostało przedstawione na poniższym przykładzie:

```
MsgBox Application.CommandBars(42).Name
```



Niestety numery indeksów elementów kolekcji CommandBars w różnych wersjach Excela mogą się zmieniać. Na przykład w Excelu 2010 obiekty CommandBar 12 i 40 mają nazwę Cell. W Excelu 2013 CommandBar o numerze indeksu równym 40 reprezentuje obiekt Row. Z tego powodu znacznie lepszym rozwiązaniem jest używanie w odwołaniach do elementów kolekcji CommandBars nazw obiektów zamiast numerów indeksów.

Odwolania do formantów obiektu CommandBar

Obiekt CommandBar zawiera obiekty klasy Control, które reprezentują przyciski lub menu. Do wybranego formantu możesz się odwoływać za pomocą jego właściwości Index lub Caption. Poniżej przedstawiamy kod prostej procedury, która wyświetla etykietę pierwszego elementu menu podrzędnego Cell:

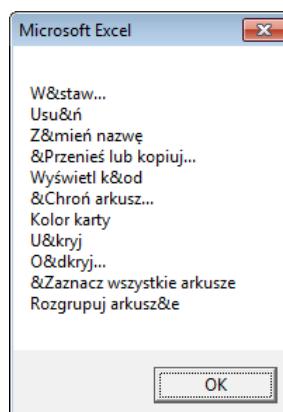
```
Sub ShowCaption()
    MsgBox Application.CommandBars("Cell").Controls(1).Caption
End Sub
```

Kolejna procedura wyświetla wartości właściwości Caption poszczególnych formantów znajdujących się w menu podrzędnym, które pojawia się, kiedy klikniesz prawym przyciskiem myszy kartę arkusza (to menu podrzęczne nosi nazwę Ply):

```
Sub ShowCaptions()
    Dim txt As String
    Dim ctl As CommandBarControl
    For Each ctl In CommandBars("Ply").Controls
        txt = txt & ctl.Caption & vbCrLf
    Next ctl
    MsgBox txt
End Sub
```

Po uruchomieniu tej procedury na ekranie pojawi się okno dialogowe, przedstawione na rysunku 21.2. Znak & (ang. *ampersand*) jest używany do wskazania podkreślonej litery menu — czyli inaczej do wskazania klawisza, który jest wykorzystywany jako skrót klawiszowy do uruchamiania danego polecenia menu.

Rysunek 21.2.
Wyświetlanie wartości właściwości Caption formantów menu



W niektórych przypadkach obiekty Control w menu podrzędnym zawierają inne obiekty Control. Na przykład formant Filtruj menu podrzędnego Cell zawiera szereg innych formantów. Formant Filtruj spełnia rolę kontenera podmenu, a zawarte w nim formanty reprezentują poszczególne elementy podmenu.

Polecenie przedstawione poniżej wyświetla pierwszy element podmenu polecenia *Filtruj*:

```
MsgBox CommandBars("Cell").Controls("Filtruj").Controls(1).Caption
```

Odszukiwanie wybranego formantu

Jeżeli tworzysz kod, który będzie wykorzystywany w różnych wersjach językowych Excela, powinieneś unikać odwoływania się do poszczególnych elementów menu za pomocą właściwości Caption. Wartości właściwości Caption dla danego formantu zmieniają się w zależności od wersji językowej Excela, stąd taki kod może nie działać poprawnie, jeżeli zostanie uruchomiony w innej wersji językowej Excela.

Zamiast tego powinieneś użyć metody FindControl w połączeniu z właściwością ID formantu (która jest niezależna od wersji językowej programu). Założymy, że chcesz wyłączyć polecenie Wytnij, które pojawia się w menu podręcznym po kliknięciu prawym przyciskiem myszy nagłówka kolumny. Jeżeli używasz wyłącznie polskiej wersji Excela, polecenie przedstawione poniżej załatwi sprawę:

```
CommandBars("Column").Controls("Wytnij").Enabled = False
```

Aby upewnić się jednak, że nasze polecenie będzie również działało z innymi wersjami językowymi Excela (na przykład wersją angielską), musisz poznać wartość właściwości ID tego polecenia. Po wykonaniu polecenia przedstawionego poniżej przekonasz się, że wartość ID tego polecenia wynosi 21:

```
MsgBox CommandBars("Column").Controls("Wytnij").ID
```

Teraz, aby wyłączyć to polecenie, powinieneś użyć następującego polecenia:

```
CommandBars.FindControl(ID:=21).Enabled = False
```

Pamiętaj, że nazwy obiektów CommandBar reprezentujących menu podręczne nie są zależne od wersji językowej programu, zatem odwołania takie jak na przykład CommandBar("Column") będą działać zawsze.

Właściwości formantów obiektu CommandBar

Formanty obiektu CommandBar posiadają cały szereg właściwości określających ich wygląd i sposób działania. Poniżej przedstawiamy listę najbardziej użytecznych właściwości formantów obiektu CommandBar:

- Caption — tekst, który jest wyświetlany na formancie. Jeżeli formant ma wyłącznie postać obrazu, tekst właściwości Caption pojawia się, kiedy ustawisz wskaźnik myszy nad tym formantem.
- ID — unikatowy, numeryczny identyfikator formantu.
- FaceID — liczba reprezentująca obraz wyświetlany obok tekstu formantu.
- Type — wartość określająca, czy formant jest przyciskiem (`msoControlButton`), czy podmenu (`msoControlPopup`).
- Picture — reprezentuje obraz wyświetlany obok tekstu formantu.
- BeginGroup — ma wartość True, jeżeli przed formantem wyświetlany jest pasek separatora.
- OnAction — nazwa makra VBA, które jest wykonywane po kliknięciu formantu przez użytkownika.
- BuiltIn — ma wartość True, jeżeli formant jest wbudowanym formantem programu Excel.

- Enabled — ma wartość True, jeżeli formant jest włączony.
- Visible — ma wartość True, jeżeli formant jest widoczny. Wiele menu podręcznych posiada ukryte formanty.
- ToolTipText — tekst, który pojawia się, kiedy użytkownik ustawi wskaźnik myszy nieruchomo nad formantem (nie odnosi się do menu podręcznego).

Wyświetlanie wszystkich elementów menu podręcznego

Procedura ShowShortcutMenuItems, której kod przedstawiono poniżej, tworzy tabelę, w której wyświetlane są wszystkie elementy pierwszego poziomu poszczególnych menu podręcznych. Dla każdego formantu w tabeli zapisywane są wartości właściwości Index, Name, ID, Caption, Type, Enabled oraz Visible.

```
Sub ShowShortcutMenuItems()
    Dim Row As Long
    Dim Cbar As CommandBar
    Dim ctl As CommandBarControl
    Range("A1:G1") = Array("Indeks", "Nazwa", "ID", "Etykieta", _
                           "Typ", "Włączone", "Widoczne")
    Row = 2
    Application.ScreenUpdating = False
    For Each Cbar In Application.CommandBars
        If Cbar.Type = 2 Then
            For Each ctl In Cbar.Controls
                Cells(Row, 1) = Cbar.Index
                Cells(Row, 2) = Cbar.Name
                Cells(Row, 3) = ctl.ID
                Cells(Row, 4) = ctl.Caption
                If ctl.Type = 1 Then
                    Cells(Row, 5) = "Przycisk"
                Else
                    Cells(Row, 5) = "Podmenu"
                End If
                Cells(Row, 6) = ctl.Enabled
                Cells(Row, 7) = ctl.Visible
                Row = Row + 1
            Next ctl
        End If
    Next Cbar
    ActiveSheet.ListObjects.Add(xlSrcRange, _
                               Range("A1").CurrentRegion, , xlYes).Name = "Table1"
End Sub
```

Na rysunku 21.3 przedstawiono fragment wyników działania tej procedury.



Skoroszyt z tym przykładem (*Pokaż elementy menu podręcznych.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

A	B	C	D	E	F	G
Indeks	Nazwa	ID	Etykieta	Typ	Włączone	Widoczne
1	22 PivotChart Menu	460	&Ustawienia pola	Przycisk	PRAWDA	PRAWDA
2	22 PivotChart Menu	1604	&Opcje...	Przycisk	FAŁSZ	PRAWDA
4	22 PivotChart Menu	459	O&dśwież dane	Przycisk	PRAWDA	PRAWDA
5	22 PivotChart Menu	3956	U&kryj przyciski pól wykresu przestawnego	Przycisk	FAŁSZ	PRAWDA
6	22 PivotChart Menu	30254	For&muly	Podmenu	PRAWDA	PRAWDA
7	22 PivotChart Menu	5416	&Usuń pole	Przycisk	FAŁSZ	PRAWDA
8	34 Workbook tabs	957	Arkusz1	Przycisk	PRAWDA	PRAWDA
9	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
10	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
11	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
12	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
13	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
14	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
15	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
16	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
17	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
18	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
19	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
20	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
21	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
22	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
23	34 Workbook tabs	957	&Lista arkuszy	Przycisk	PRAWDA	FAŁSZ
24	35 Cell	21	Wy&nij	Przycisk	PRAWDA	PRAWDA
25	35 Cell	19	&Kopij	Przycisk	PRAWDA	PRAWDA
26	35 Cell	22	Wkl&ej	Przycisk	PRAWDA	PRAWDA
27	35 Cell	21437	Wklej spe&cjalnie...	Przycisk	PRAWDA	PRAWDA
28	35 Cell	3624	Wklej t&abelę	Przycisk	PRAWDA	PRAWDA
29	35 Cell	3181	&Wstaw...	Przycisk	PRAWDA	PRAWDA
30	35 Cell	292	&Usuń...	Przycisk	PRAWDA	PRAWDA
31	35 Cell	3125	Wyczys&ć zawartość	Przycisk	PRAWDA	PRAWDA
32	35 Cell	24508	Szybk&a analiza	Przycisk	PRAWDA	PRAWDA
33	35 Cell	31623	Wykresy pr&zbiegu w czasie	Podmenu	PRAWDA	FAŁSZ
34	35 Cell	31402	Filt&uj	Podmenu	PRAWDA	PRAWDA
35	35 Cell	31435	&Sortuj	Podmenu	PRAWDA	PRAWDA

Rysunek 21.3. Wyświetlanie elementów pierwszego poziomu wszystkich menu podręcznych

Wykorzystanie VBA do dostosowywania menu podręcznego

W tej sekcji zaprezentujemy kilka praktycznych przykładów zastosowania VBA do modyfikacji menu podręcznego Excela. Przykłady pozwolą Ci na zorientowanie się w tym, co możesz zrobić z menu podręcznym, i oczywiście mogą być w dowolny sposób modyfikowane i wykorzystywane w Twoich własnych projektach.

Co nowego w Excelu 2013

Jeżeli w poprzednich wersjach używałeś VBA do pracy z menu podręcznym, powinieneś zwrócić szczególną uwagę na to, że w Excelu 2013 zaszły bardzo istotne zmiany w tym zakresie.

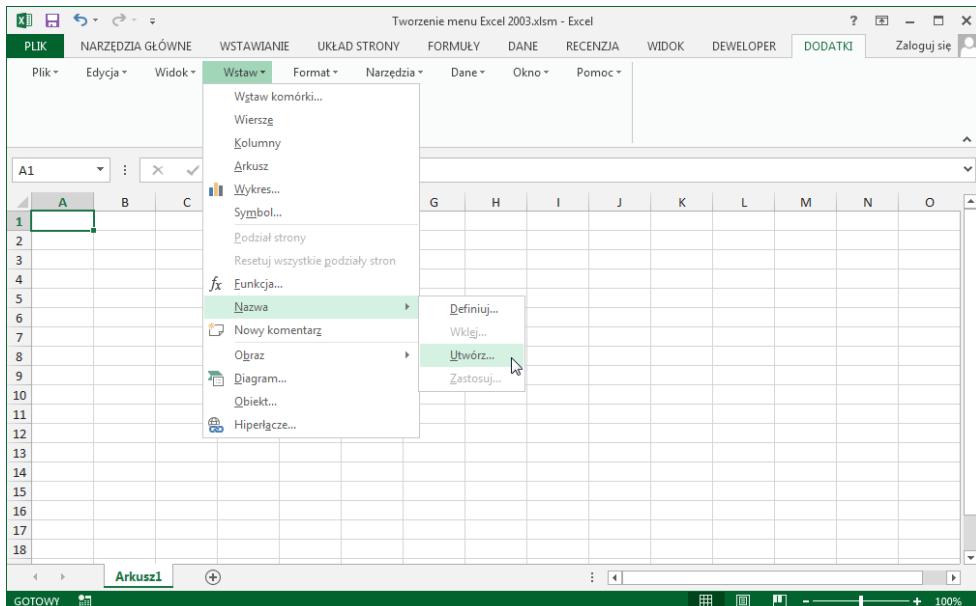
Wyświetlanie menu Excela 2003

Jedno z wbudowanych menu podrzcznych nosi nazwę Built-in Menus i zawiera elementy menu głównego używanego w Excelu 2003 (ostatnia wersja Excela posiadająca klasyczne menu, przed wprowadzeniem Wstążki). Menu nie jest przypisane do żadnego obiektu, ale w razie potrzeby możesz je wyświetlić przy użyciu następującego polecenia VBA:

```
Application.CommandBars("Built-in Menus").ShowPopup
```

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz przykładowy skoroszyt o nazwie *Tworzenie menu Excel 2003.xlsxm*, zawierający kod VBA wyświetlający menu Excela 2003 w postaci paska narzędzi na karcie Dodatek. W rezultacie, po wykonaniu takiego polecenia możesz korzystać w Excelu 2013 z menu Excela 2003.

Na rysunku poniżej przedstawiono wygląd takiego rozwiązania.



Niektóre polecenia nie będą już jednak tutaj działały i oczywiście w takim menu nie ma żadnych nowych opcji, które zostały zaimplementowane w nowszych wersjach Excela, stąd przedstawiona sztuczka to bardziej ciekawostka niż użyteczne narzędzie.

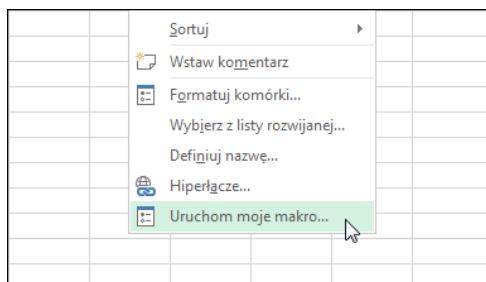
W poprzednich wersjach, jeżeli Twój kod wprowadzał jakieś modyfikacje do menu podrzcznego, to takie zmiany były widoczne we wszystkich skoroszytach. Na przykład jeżeli dodałeś nowe polecenie do menu podrzcznego wyświetlanego po kliknięciu komórki prawym przyciskiem myszy, to taka zmiana była widoczna *we wszystkich* otwartych skoroszytach (oraz w tych, które otworzyłeś później). Innymi słowy, zmiany w menu podrzcznym były wprowadzane *na poziomie aplikacji*.

Dostosowywanie menu podręcznego za pomocą kodu RibbonX

Do wprowadzania zmian w menu podręcznym i dostosowywania go do własnych potrzeb możesz używać również odpowiednio przygotowanego kodu RibbonX. Kiedy skoroszyt zawierający taki kod zostanie otwarty, wprowadzone zmiany są widoczne tylko w tym skoroszycie. Aby modyfikacje menu podręcznego były widoczne we wszystkich skoroszytach, powinieneś umieścić odpowiedni kod RibbonX w dodatku Excela.

Poniżej zamieszczamy przykład prostego kodu RibbonX, który wprowadza modyfikacje do menu podręcznego widocznego po kliknięciu komórki prawym przyciskiem myszy. Jak widać na załączonym niżej rysunku, kod RibbonX dodaje nowe polecenie zaraz za polemieniem *Hiperłącze*.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <contextMenus>
        <contextMenu idMso="ContextMenuCell">
            <button id="MyMenuItem"
                label="Uruchom moje makro..." 
                insertAfterMso="HyperlinkInsert"
                onAction="MyMacro"
                imageMso="AdvancedFileProperties"/>
        </contextMenu>
    </contextMenus>
</customUI>
```



Możliwość zastosowania kodu RibbonX do modyfikacji menu podręcznego została wprowadzona dopiero w Excelu 2010, stąd taka technika nie będzie działała w Excelu 2007.

Jak już wspominaliśmy w rozdziale 20., do wprowadzania kodu RibbonX do skoroszytu musimy użyć programu zewnętrznego¹.

Excel 2013 wykorzystuje nowy, jednodokumentowy interfejs użytkownika, co ma znaczący wpływ na sposób działania menu podręcznego. Zmiany wprowadzane w menu podręcznym działają tylko w oknie aktywnego skoroszytu. Kiedy uruchomisz kod modyfikujący menu podręczne w aktywnym skoroszycie, menu podręczne w oknach pozostałych skoroszytów nie zostanie w żaden sposób zmienione. Takie zachowanie jest diametralnie inne od sposobu, w jaki to działało w poprzednich wersjach Excela.

Kolejna ciekawostka — jeżeli użytkownik otworzy skoroszyt (lub utworzy nowy skoroszyt) w sytuacji, kiedy w aktywnym skoroszycie zostało zmodyfikowane menu podręczne, nowe okno skoroszytu niejako „dziedziczy” takie ustawienia menu podręcznego z aktywnego skoroszytu. Jeżeli napiszesz procedurę, która usuwa zmodyfikowane menu podręczne, to takie menu zostaje usunięte tylko z oryginalnego skoroszytu.

¹ W tym przypadku był to program *Custom UI Editor For Microsoft Office* — przyp. tłum.

Jak widać, pomimo tego, że modyfikacje wprowadzane do menu podręcznego miały działać tylko w jednym skoroszycie, pojawia się tutaj potencjalny problem — jeżeli użytkownik otworzy nowy skoroszyt, to będzie w nim wyświetlane zmodyfikowane menu podręczne. Rozwiążaniem może być wprowadzenie takich zmian do kodu, aby makra uruchamiane z poziomu menu podręcznego działały tylko w oryginalnym skoroszycie, dla którego były przeznaczone.

Jeżeli chcesz używać menu podręcznego jako wygodnego sposobu uruchamiania makr zdefiniowanych w dodatku, to nowe pozycje w menu podręcznym będą widoczne tylko w tych skoroszytach, które zostaną otwarte lub utworzone *po* otwarciu i zainstalowaniu dodatku.

Wnioski? W przeszłości, kiedy otwierałeś skoroszyt lub dodatek zawierający zmodyfikowane menu podręczne, mogłeś być pewien, że takie zmiany będą widoczne we wszystkich skoroszytach. W Excelu 2013 niestety takiej pewności mieć już nie możesz.

Resetowanie menu podręcznego

Metoda Reset przywraca menu podręczne do oryginalnego, początkowego stanu. Procedura, której kod zamieszczono poniżej, przywraca początkowy stan menu podręcznego Cell.

```
Sub ResetCellMenu()
    CommandBars("Cell").Reset
End Sub
```

W Excelu 2013 takie wywołanie metody Reset będzie miało wpływ na menu podręczne, które pojawia się po kliknięciu komórki prawym przyciskiem myszy tylko w oknie aktywnego skoroszytu.

Jak już wspomniano wcześniej, Excel posiada dwa menu podręczne o nazwie Cell. Po przednia procedura resetuje tylko ustawienia pierwszego z nich (indeks 35). Aby zresetować drugie menu podręczne, zamiast nazwy menu powinieneś użyć jego numeru indeksu (38). Pamiętaj jednak, że numery indeksów w różnych wersjach Excela są różne. Poniżej przedstawiamy kod lepszej wersji procedury przywracającej obie instancje menu podręcznego Cell do stanu początkowego.

```
Sub ResetCellMenu()
    Dim cbar As CommandBar
    For Each cbar In Application.CommandBars
        If cbar.Name = "Cell" Then cbar.Enabled = False
    Next cbar
End Sub
```

Procedura, której kod zamieszczono poniżej, przywraca wszystkie wbudowane paski narzędzi do ich początkowego stanu.

```
Sub ResetAll()
    Dim cbar As CommandBar
    For Each cbar In Application.CommandBars
        If cbar.Type = msoBarTypePopup Then
            cbar.Reset
            cbar.Enabled = True
        End If
    Next cbar
End Sub
```

W Excelu 2013 procedura ResetAllShortcutMenus będzie działała tylko w zakresie okna aktywnego skoroszytu. Aby zresetować menu podręczne we wszystkich otwartych skoroszytach, musimy użyć nieco bardziej złożonej procedury:

```
Sub ResetAllShortcutMenus2()
    ' Procedura działa dla wszystkich otwartych skoroszytów
    Dim cbar As CommandBar
    Dim activeWin As Window
    Dim win As Window
    ' Zapamiętuje bieżące aktywne okno
    Set activeWin = ActiveWindow
    ' Przechodzi w pętli przez kolejne okna skoroszytów
    Application.ScreenUpdating = False
    For Each win In Windows
        If win.Visible Then
            win.Activate
            For Each cbar In Application.CommandBars
                If cbar.Type = msoBarTypePopup Then
                    cbar.Reset
                    cbar.Enabled = True
                End If
            Next cbar
        End If
    Next win
    ' Aktywuje oryginalne okno skoroszytu
    activeWin.Activate
    Application.ScreenUpdating = True
End Sub
```

Kod rozpoczyna działanie od zapamiętania bieżącego aktywnego okna skoroszytu w zmiennej obiektowej activeWin. Następnie procedura przechodzi w pętli przez wszystkie otwarte okna skoroszytów i po kolei je aktywuje (ale pomija okna ukryte, ponieważ aktywowanie okna ukrytego spowodowałoby jego wyświetlenie na ekranie). Dla każdego aktywnego okna procedura przechodzi w kolejnej pętli przez wszystkie obiekty CommandBar i resetuje te, które reprezentują menu podręczne. Po zakończeniu procedury reaktywuje oryginalne okno skoroszytu.



Skoroszyt z omawianymi wyżej przykładami (*Resetowanie menu podręcznego.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Wyłączanie menu podręcznego

Właściwość Enabled pozwala na włączanie bądź wyłączanie menu podręcznego. Na przykład możesz ustawić tę właściwość tak, że kliknięcie komórki prawym przyciskiem myszy nie będzie powodowało wyświetlenia menu podręcznego. Polecenie, którego kod zamieszczono poniżej, wyłącza wyświetlanie menu podręcznego Cell:

```
Application.CommandBars("Cell").Enabled = False
```

Aby przywrócić wyświetlanie menu podręcznego, wystarczy po prostu ustawić jego właściwość Enabled na wartość True.

Jeżeli chcesz wyłączyć wyświetlanie *wszystkich* menu podręcznych, możesz użyć następującej procedury:

```
Sub DisableAllShortcutMenus()
    Dim cb As CommandBar
    For Each cb In CommandBars
        If cb.Type = msoBarTypePopup Then _
            cb.Enabled = False
    Next cb
End Sub
```

Wyłączanie wybranych elementów menu podręcznego

Często możesz spotkać się z sytuacją, w której użyteczne byłoby wyłączenie wybranego elementu menu podręcznego w czasie działania aplikacji. Kiedy dany element menu jest wyłączony, tekst polecenia ma kolor szary i jego klikanie nie przynosi żadnych rezultatów. Procedura, której kod zamieszczono poniżej, wyłącza polecenia *Ukryj* znajdujące się w menu podręcznych *Row* oraz *Column*.

```
Sub DisableHideMenuItems()
    CommandBars("Column").Controls("Ukryj").Enabled = False
    CommandBars("Row").Controls("Ukryj").Enabled = False
End Sub
```

Procedura przedstawiona powyżej nie uniemożliwia użytkownikowi korzystania z innych sposobów ukrywania wierszy lub kolumn, takich jak na przykład polecenia z grupy *Komórki* znajdującej się na karcie *NARZĘDZIA GŁÓWNE*.

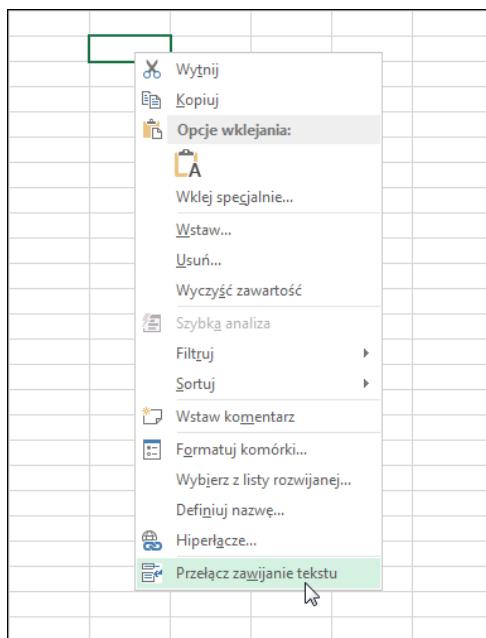
Dodawanie nowego elementu do menu podręcznego Cell

Procedura *AddToShortcut*, której kod zamieszczono poniżej, dodaje nowy element — *Przełącz zawijanie tekstu* do menu podręcznego *Cell*. Jak pamiętasz, Excel posiada dwa menu podręczne o nazwie *Cell*. Procedura omawiana w tym przykładzie modyfikuje standardeowe menu *Cell*, dostępne po kliknięciu komórki prawym przyciskiem myszy (a nie menu *Cell* dostępne w trybie podglądu podziału stron).

```
Sub AddToShortcut()
    ' Dodaje nowy element do menu podręcznego Cell
    Dim Bar As CommandBar
    Dim NewControl As CommandBarButton
    DeleteFromShortcut
    Set Bar = CommandBars("Cell")
    Set NewControl = Bar.Controls.Add _
        (Type:=msoControlButton)
    With NewControl
        .Caption = "Przełącz za&wijanie tekstu"
        .OnAction = "ToggleWordWrap"
        .Picture = Application.CommandBars.GetImageMso _
            ("WrapText", 16, 16)
        .Style = msoButtonIconAndCaption
    End With
End Sub
```

Na rysunku 21.4 przedstawiono wygląd nowego elementu menu podręcznego, dostępnego po kliknięciu komórki prawym przyciskiem myszy.

Rysunek 21.4.
*Nowy element
utworzony w menu
podręcznym Cell*



Pierwsze polecenie po bloku deklaracji zmiennych wywołuje procedurę `DeleteFromShortcut` (która omówimy za chwilę). Wywołanie tej procedury zapewnia, że w menu podręcznym Cell pojawi się tylko jedno polecenie *Przelącz zawijanie tekstu*. Zwróć uwagę na fakt, że klawiszem skrótu dla tego polecenia jest *W*, a nie *T*, ponieważ klawisz *T* jest już wykorzystywany przez polecenie *Wytnij*.

Właściwość Picture jest ustawiana poprzez odwołanie do obrazu wykorzystywanego przez polecenie *Zawijaj tekst*, znajdujące się na Wstążce. Więcej szczegółowych informacji na temat korzystania z obrazów używanych przez polecenia Wstążki znajdziesz w rozdziale 20.

Po wybraniu polecenia *Przelącz zawijanie tekstu* wykonywane jest makro VBA zdefiniowane przez właściwość `OnAction`. W tym przypadku nasze makro nosi nazwę `ToggleWordWrap`.

```
Sub ToggleWrapText()
    On Error Resume Next
    CommandBars.ExecuteMso ("WrapText")
    If Err.Number <> 0 Then MsgBox "Nie mogę przełączyć zawijania tekstu"
End Sub
```

Uruchomienie tej procedury powoduje po prostu wykonanie polecenia `WrapText` Wstążki. Jeżeli próba wykonania tej procedury zakończy się niepowodzeniem (na przykład kiedy arkusz jest chroniony), na ekranie zostanie wyświetlony odpowiedni komunikat o wystąpieniu błędu.

Procedura DeleteFromShortcut usuwa nowy element menu podręcznego Cell.

```
Sub DeleteFromShortcut()
    On Error Resume Next
    CommandBars("Cell").Controls ("Przełącz za&wijanie tekstu").Delete
End Sub
```

W większości przypadków będziesz chciał dodawać i usuwać elementy menu całkowicie automatycznie — dodawać nowe elementy menu podręcznego, kiedy skoroszyt jest otwierany, i usuwać je, kiedy zostanie zamknięty. Aby to zrobić, wystarczy w module kodu ThisWorkbook utworzyć dwie proste procedury obsługujące zdarzenia:

```
Private Sub Workbook_Open()
    Call AddToShortCut
End Sub

Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call DeleteFromShortcut
End Sub
```

Procedura Workbook_Open jest wykonywana, kiedy skoroszyt jest otwierany, a procedura Workbook_BeforeClose jest wykonywana przed zamknięciem skoroszytu. To jest dokładnie to, o co nam chodziło.

Co ciekawe, jeżeli używasz procedur modyfikujących menu podręczne wyłącznie w Excelu 2013, nie musisz usuwać wprowadzonych modyfikacji przed zamknięciem skoroszytu, ponieważ takie zmiany są aplikowane tylko i wyłącznie do okna aktywnego skoroszytu.



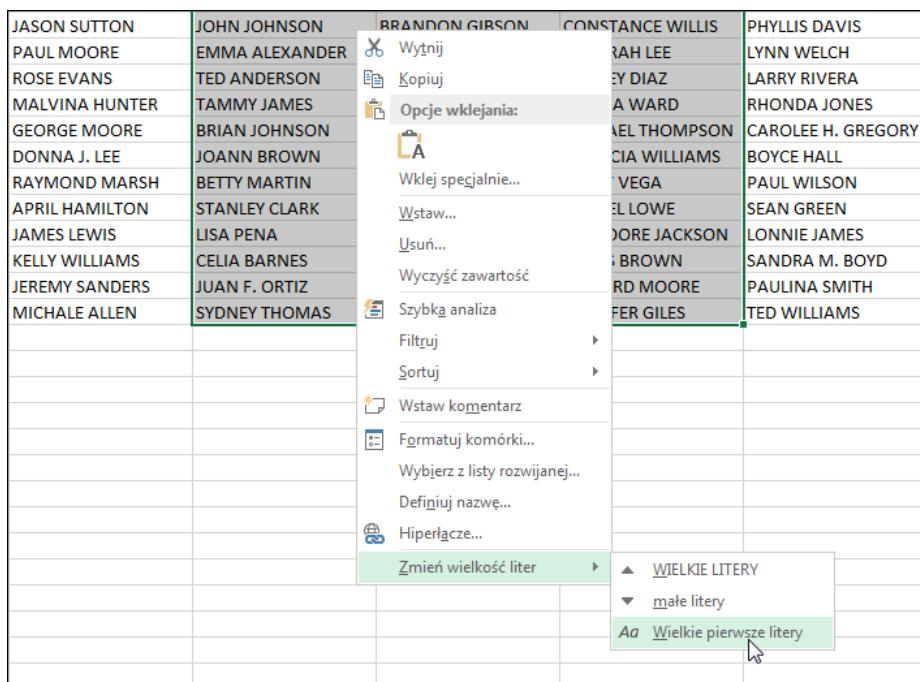
Skoroszyt z tym przykładem (*Tworzenie nowych elementów menu podręcznego.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>). W pliku tym zamieszczono również kolejną wersję tego makra, która dodaje nowe elementy do menu podręcznego wszystkich otwartych okien skoroszytów.

Dodawanie nowego podmenu do menu podręcznego

Przykład opisany w tej sekcji dodaje nowe podmenu, składające się z trzech elementów, do menu podręcznego Cells. Na rysunku 21.5 przedstawiono arkusz, w którym zaznaczony obszar został kliknięty prawym przyciskiem myszy. Poszczególne polecenia podmenu wywołują odpowiednie makra, które zmieniają wielkość liter tekstu znajdującego się w zaznaczonych komórkach.

Kod procedury, która tworzy nowe podmenu wraz z nowymi poleceniami, zamieszczono poniżej:

```
Sub Add_submenu()
    ' Dodaje nowe podmenu do 6 różnych menu podręcznych
    Dim Bar As CommandBar
    Dim NewMenu As CommandBarControl
    Dim NewSubmenu As CommandBarButton
```



Rysunek 21.5. To menu podręczne posiada nowe podmenu zawierające trzy nowe polecenia

```

DeleteSubMenu
Set Bar = CommandBars("Cell")
'Dodaj podmenu
Set NewMenu = Bar.Controls.Add _
(Type:=msoControlPopup)
NewMenu.Caption = "Zmień wielkość liter"
NewMenu.BeginGroup = True
'Dodaj pierwszy element podmenu
Set NewSubmenu = NewMenu.Controls.Add _
(Type:=msoControlButton)
With NewSubmenu
    .FaceId = 38
    .Caption = "&WIELKIE LITERY"
    .OnAction = "MakeUpperCase"
End With
'Dodaj drugi element podmenu
Set NewSubmenu = NewMenu.Controls.Add _
(Type:=msoControlButton)
With NewSubmenu
    .FaceId = 40
    .Caption = "&małe litery"
    .OnAction = "MakeLowerCase"
End With
'Dodaj trzeci element podmenu
Set NewSubmenu = NewMenu.Controls.Add _
(Type:=msoControlButton)
With NewSubmenu
    .FaceId = 476
    .Caption = "&Wielkie pierwsze litery"

```

```

    .OnAction = "MakeProperCase"
End With
End Sub

```

Najpierw dodawane jest podmenu, którego właściwość Type jest ustawiana na wartość msoControlPopup. Następnie tworzone są trzy podmenu, z których każde ma inną wartość właściwości OnAction.

Kod usuwający podmenu jest znacznie prostszy:

```

Sub DeleteSubMenu()
On Error Resume Next
CommandBars("Cell").Controls("Cha&nge Case").Delete
End Sub

```



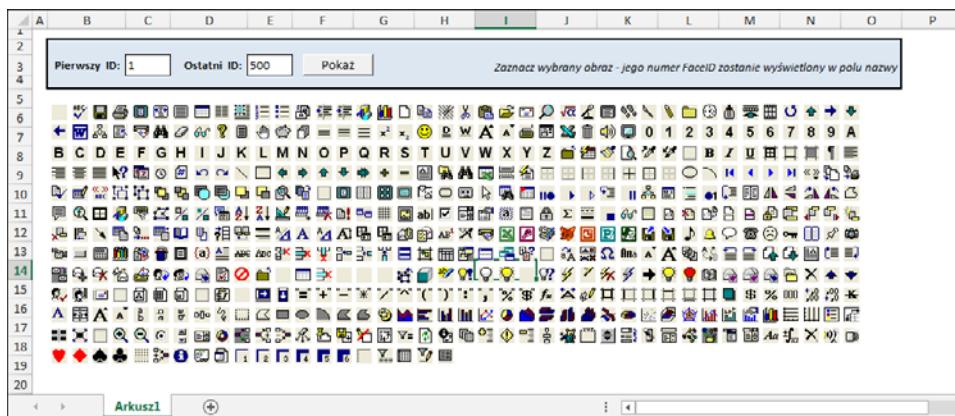
Skoroszyt z tym przykładem (*Tworzenie nowego podmenu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Odszukiwanie obrazów FaceID

Ikona polecenia, która jest wyświetlana w menu podręcznym, jest zdefiniowana przez jedną z dwóch właściwości:

- Picture — wybranie tej opcji pozwala na użycie wybranego obrazu imageMso ze Wstążki. Przykład użycia takiego obrazu znajdziesz w sekcji „Dodawanie nowego elementu do menu podręcznego Cell” we wcześniejszej części tego rozdziału.
- FaceID — to jest najprostsze rozwiązanie, ponieważ FaceID to po prostu wartość numeryczna reprezentująca jeden z setek dostępnych obrazów.

Ale jak zorientować się, jakie obrazy są reprezentowane przez poszczególne wartości FaceID? Excel nie ma bezpośredniej metody wykonania takiego podglądu, zatem utworzyłem prostą aplikację, która pozwala na przeglądanie tych obrazów. Po uruchomieniu wprowadź początkowy i końcowy numer FaceID, naciśnij przycisk i odpowiednie obrazy zostaną wyświetcone w arkuszu. Każdy z obrazów posiada nazwę odpowiadającą wartości FaceID. Na rysunku poniżej przedstawiono wygląd arkusza wyświetlającego obrazy o wartościach FaceID z zakresu od 1 do 500. Skoroszyt z tą aplikacją (*Pokaż obrazy FaceID.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Ograniczanie zasięgu modyfikacji menu podręcznego do jednego skoroszytu

Jak już nie raz wspominaliśmy, w Excelu 2013 modyfikacje wprowadzane do menu podręcznego działają wyłącznie dla okna aktywnego skoroszytu (nazwijmy go *skoroszytem A*). Na przykład założmy, że chcesz w *skoroszycie A* wstawić dodatkowe polecenie do menu podręcznego, które pojawia się na ekranie po kliknięciu dowolnej komórki prawym przyciskiem myszy. Jeżeli jednak po wprowadzeniu takich modyfikacji użytkownik utworzy nowy skoroszyt, to „odziedziczy” on również takie zmodyfikowane menu podręczne. Jeżeli chcesz, aby zmodyfikowane menu podręczne działało tylko wtedy, kiedy aktywnym skoroszytem jest *skoroszyt A*, musisz do makra wykonywanego przez menu podręczne wprowadzić trochę dodatkowego kodu.

Założymy, że napisałś procedurę, która dodaje do menu podręcznego polecenie uruchamiające makro o nazwie *MojeMakro*. Aby ograniczyć zasięg działania takiej zmiany tylko do oryginalnego skoroszytu, w którym została zdefiniowana, możesz użyć na przykład takiej procedury:

```
Sub MyMacro()
    If Not ActiveWorkbook Is ThisWorkbook Then
        MsgBox "To polecenie menu nie działa w tym skoroszycie."
    Else
        ' [... Tutaj umieść kod Twojego makra ...]
    End If
End Sub
```

Menu podręczne i zdarzenia

Przykłady zaprezentowane w tym rozdziale ilustrują różne techniki programowania menu podręcznego z wykorzystaniem zdarzeń.



Więcej szczegółowych informacji na temat programowania obsługi zdarzeń znajdziesz w rozdziale 17.

Automatyczne tworzenie i usuwanie menu podręcznego

Jeżeli chcesz dokonać automatycznej modyfikacji menu podręcznego w momencie otwierania skoroszytu, powinieneś skorzystać ze zdarzenia *Workbook_Open*. Procedura obsługi tego zdarzenia, której kod — zlokalizowany w module *ThisWorkbook* — zamieszczono poniżej, wykonuje procedurę *ModifyShortcut* (kodu tej procedury nie został zamieszczony):

```
Private Sub Workbook_Open()
    Call ModifyShortcut
End Sub
```

Aby przywrócić menu podręczne do stanu początkowego, powinieneś użyć procedury przedstawionej poniżej, która jest wykonywana przed zamknięciem skoroszytu. Procedura obsługi zdarzenia wywołuje procedurę *RestoreShortcut* (nie pokazuję kodu tej procedury):

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call RestoreShortcut
End Sub
```

Jeżeli używasz procedur modyfikujących menu podręczne wyłącznie w Excelu 2013, nie musisz usuwać wprowadzonych modyfikacji przed zamknięciem skoroszytu, ponieważ takie zmiany są aplikowane tylko i wyłącznie do okna aktywnego skoroszytu i znikają po jego zamknięciu.

Wyłączanie lub ukrywanie elementów menu podręcznego

Kiedy wybrany element menu podręcznego jest wyłączony, tekst jego polecenia jest wyświetlanym kolorze szarym, a jego klikanie nie przynosi żadnych efektów. Kiedy wybrany element menu jest ukryty, po prostu nie jest widoczny w menu podręcznym. Możesz oczywiście napisać odpowiedni kod VBA, który w miarę potrzeb będzie włączał i wyłączał poszczególne polecenia menu. Podobnie możesz utworzyć kod, który będzie ukrywał polecenia menu. Kluczowym zagadnieniem w takim przypadku jest oczywiście użycie odpowiedniego zdarzenia.

Na przykład fragment kodu przedstawiony poniżej wyłącza w menu podręcznym polecenie *Zmień wielkość liter* (które zostało dodane do menu podręcznego *Cell* w jednym z poprzednich przykładów), kiedy aktywowany jest Arkusz2. Kod procedury powinien zostać umieszczony w module kodu arkusza Arkusz2.

```
Private Sub Worksheet_Activate()
    CommandBars("Cell").Controls("Zmień wielkość liter").Enabled = False
End Sub
```

Aby ponownie włączyć to polecenie menu, kiedy Arkusz2 zostanie zdezaktywowany, powinieneś dołączyć procedure, której kod zamieszczono poniżej. Efektem działania tych dwóch procedur jest to, że polecenie *Zmień wielkość liter*, znajdujące się w menu podręcznym *Cell*, jest dostępne dla wszystkich arkuszy z wyjątkiem arkusza o nazwie Arkusz2.

```
Private Sub Worksheet_Deactivate()
    CommandBars("Cell").Controls("Zmień wielkość liter").Enabled = True
End Sub
```

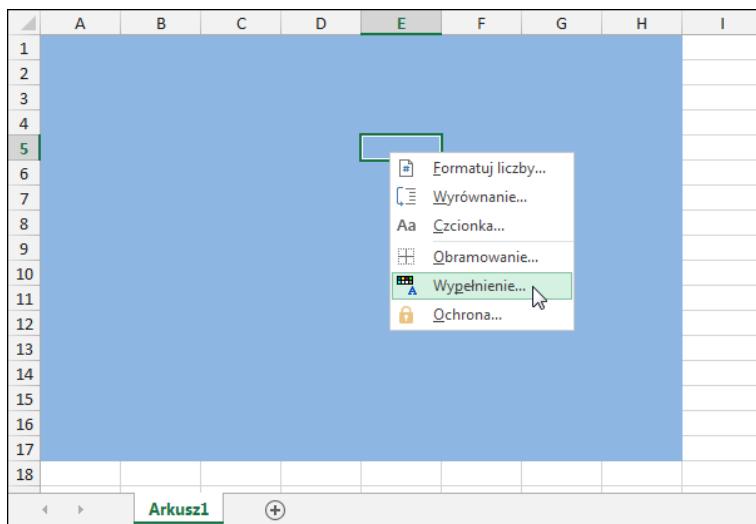
Jeżeli zamiast wyłączania chcesz po prostu ukryć wybrany element menu podręcznego, powinieneś zamiast właściwości *Enabled* użyć właściwości *Visible*.

Tworzenie kontekstowych menu podręcznych

W razie potrzeby możesz utworzyć zupełnie nowe menu podręczne i wyświetlać je na ekranie w odpowiedzi na określone zdarzenie. Procedura, której kod zamieszczono poniżej, tworzy nowe menu podręczne o nazwie *MyShortcut* i dodaje do niego sześć poleceń. Poszczególne elementy menu mają właściwości *OnAction* ustawione na wykonanie prostej procedury, która wyświetla jedną z kart okna dialogowego *Formatowanie komórek* (patrz rysunek 21.6).

Rysunek 21.6.

Nowe menu podręczne pojawia się tylko wtedy, kiedy użytkownik kliknie prawym przyciskiem myszy szary obszar arkusza



```
Sub CreateShortcut()
    Set myBar = CommandBars.Add
        (Name:="MyShortcut", Position:=msoBarPopup)

    ' Dodaj element menu
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Formatuj liczby..."
        .OnAction = "ShowFormatNumber"
        .FaceId = 1554
    End With

    ' Dodaj element menu
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Wyrównanie..."
        .OnAction = "ShowFormatAlignment"
        .FaceId = 217
    End With

    ' Dodaj element menu
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Czcionka..."
        .OnAction = "ShowFormatFont"
        .FaceId = 291
    End With

    ' Dodaj element menu
    Set myItem = myBar.Controls.Add(Type:=msoControlButton)
    With myItem
        .Caption = "&Obramowanie..."
        .OnAction = "ShowFormatBorder"
        .FaceId = 149
        .BeginGroup = True
    End With
End Sub
```

```
' Dodaj element menu
Set myItem = myBar.Controls.Add(Type:=msoControlButton)
With myItem
    .Caption = "Wy&pełnienie..."
    .OnAction = "ShowFormatPatterns"
    .FaceId = 1550
End With

' Dodaj element menu
Set myItem = myBar.Controls.Add(Type:=msoControlButton)
With myItem
    .Caption = "&Ochrona..."
    .OnAction = "ShowFormatProtection"
    .FaceId = 2654
End With
End Sub
```

Po utworzeniu menu podręcznego możesz wyświetlać je przy użyciu metody ShowPopup. Poniższa procedura, której kod został umieszczony w module obiektu Worksheet, jest wykonywana po kliknięciu przez użytkownika prawym przyciskiem myszy w komórce:

```
Private Sub Worksheet_BeforeRightClick
    (ByVal Target As Excel.Range, Cancel As Boolean)
    If Union(Target.Range("A1"), Range("data")).Address = _
        Range("data").Address Then
        CommandBars("MyShortcut").ShowPopup
        Cancel = True
    End If
End Sub
```

Jeżeli użytkownik kliknie prawym przyciskiem myszy komórkę, która znajduje się w zakresie komórek o nazwie data, na ekranie pojawi się niestandardowe menu podręczne. Ustawienie argumentu Cancel na wartość True zapewnia, że normalne menu podręczne nie zostanie wyświetlone. Zwróć uwagę, że standardowy minipasek narzędzi również nie jest wyświetlany.

Nasze niestandardowe menu podręczne możesz również wyświetlać bez użycia myszy. Aby to zrobić, wystarczy utworzyć prostą procedurę i przypisać do niej wybrany klawisz skrótu, naciskając przycisk *Opcje* w oknie dialogowym *Makro*.

```
Sub ShowMyShortcutMenu()
    ' Klawisz skrótu: Ctrl+Shift+M
    CommandBars("MyShortcut").ShowPopup
End Sub
```



Skoroszyt z tym przykładem (*Nowe menu podręczne.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci

Rozdział 22.

Tworzenie systemów pomocy w aplikacjach

W tym rozdziale:

- Tworzenie systemów pomocy w aplikacjach
- Tworzenie systemów pomocy przy użyciu wbudowanych komponentów Excela
- Wyświetlanie plików utworzonych za pomocą systemu HTML Help
- Łączenie plików pomocy z aplikacją
- Inne sposoby wyświetlania plików pomocy HTML Help

Systemy pomocy w aplikacjach Excela

Jeżeli tworzysz aplikację, która z natury rzeczy nie jest zupełnie trywialna, powinieneś pomyśleć o zaimplementowaniu w niej takiego czy innego systemu pomocy dla użytkownika końcowego. Dzięki temu korzystanie z aplikacji stanie się łatwiejsze, a programista nie będzie marnował czasu, odpowiadając na dziesiątki telefonów z prostymi pytaniami. Dodatkowo użytkownik będzie zawsze miał pod ręką potrzebne informacje — krótko mówiąc, taki „podręcznik użytkownika” nie zginie gdzieś pod stertą książek w Twoim biurze.

Pomoc online

W przeszłości zawsze określałem system pomocy Excela jako *pomoc online*. Istotnie, jest to ogólna nazwa dla tego rodzaju instrukcji. Jednak w ostatnich latach terminu *online* zaczęto używać w odniesieniu do informacji dostępnych w sieci Internet. W związku z tym wyrażenie *pomoc online* wprowadzało niektórych użytkowników w zakłopotanie, ponieważ informacje były w rzeczywistości zapisane na ich lokalnych dyskach twardych.

Uwagi na temat przykładów przedstawionych w tym rozdziale

W wielu przykładach omawianych w tym rozdziale do zademonstrowania różnych sposobów tworzenia systemów pomocy użyto prostej aplikacji, która wykorzystuje dane zapisane w arkuszu do generowania i drukowania formularzy.

Jak można zobaczyć na zamieszczonym niżej rysunku, w komórkach arkusza wyświetlna jest całkowita liczba rekordów bazy danych (komórka C2 — wartość obliczona za pomocą formuły), bieżący numer rekordu (C3), pierwszy rekord do wydrukowania (C4) oraz ostatni rekord do wydrukowania (C5). Aby wyświetlić określony rekord, należy wprowadzić wybraną wartość w komórce C3. Aby wydrukować ciąg formularzy, należy wprowadzić numery pierwszego i ostatniego rekordu w komórkach C4 i C5.

The screenshot shows an Excel spreadsheet with a mail merge template for a postcard. The spreadsheet has columns A through I and rows 1 through 36. Row 1 contains column headers. Rows 2-5 show calculated values: C2 (Całkowita liczba rekordów: 6), C3 (Bieżący rekord: 3), C4 (Pierwszy rekord do wydrukowania: 3), and C5 (Ostatni rekord do wydrukowania: 6). A menu bar at the top includes 'Drukuj formularze' (Print form), 'Przeglądaj lub edytuj dane' (View or edit data), and 'Pomoc' (Help). The main area of the spreadsheet contains a blue elephant illustration and text for a postcard:

Słoń dla każdego
skrytka pocztowa 34
00-950 Warszawa

1 marzec 2013

Maria Kowalska
Mickiewicza 4 m. 34
63-111 Kraków
woj. małopolskie

Szanowni Państwo!

Dziękujemy za zainteresowanie adopcją słonia. Obecne zapotrzebowanie przekracza nasze możliwości, a zatem każdy wniosek jest dokładnie analizowany. O naszej decyzji poinformujemy Państwa za około sześć do ośmiu tygodni.

Proszę pamiętać, że zajmowanie się słoniem zajmuje dużo czasu.
Z naszego doświadczenia wynika także, że słonie niezbyt dobrze się czują w obszarach wysoko zurbanizowanych. Szczególnie nie lubią mieszkać w wysokich wieżowcach.

Z poważaniem,

James R. Jones

James R. Jones
Dyrektor

Aplikacja jest bardzo prosta, ale składa się z kilku oddzielnych komponentów, które prezentują różne sposoby wyświetlania pomocy kontekstowej.

Skoroszyt składa się z następujących elementów:

- Szablon — arkusz zawierający tekst szablonu listu.
- Dane — arkusz zawierający bazę danych mającą siedem pól.
- ArkuszPomoc — arkusz występujący tylko w tych przykładach, w których tekst pomocy jest zapisany w arkuszu.
- ModDruk — moduł w języku VBA zawierający makra służące do drukowania szablonów.
- ModPomoc — moduł w języku VBA zawierający makra zarządzające wyświetlaniem pomocy. Treść tego modułu zależy od rodzaju prezentowanego systemu pomocy.
- UserForm1 — formularz *UserForm* występujący tylko w tych technikach wyświetlania pomocy, w których wykorzystywany jest obiekt *UserForm*.

Aby uniknąć nieporozumień, do określania systemu pomocy zapewnianego przez samą aplikację używałem dotąd określenia *pomoc systemowa*. Jednak wraz z wydaniem Excela 2003 historia zatoczyła koło. System pomocy Excela 2003 i nowszych wersji działa (opcjonalnie) w trybie online. W zależności od preferencji możesz albo przeglądać informacje dostępne lokalnie, albo (mając do dyspozycji połączenie z siecią Internet) połączyć się ze stroną internetową firmy Microsoft i wyszukiwać najbardziej aktualne tematy pomocy. Wyjątkiem od tej reguły jest system pomocy Excela 2013 dla VBA, który działa tylko i wyłącznie w trybie online, stąd do poprawnego działania wymaga stałego połączenia z siecią Internet.

Systemy pomocy można tworzyć na wiele sposobów, od bardzo prostych do skomplikowanych. Wybrana metoda zależy od tematyki tworzonej aplikacji, jej złożoności oraz poniesionego nakładu pracy. W przypadku niektórych aplikacji wystarczy kilka instrukcji opisujących sposób ich uruchomienia. W innych trzeba utworzyć rozbudowany system pomocy wraz z wyszukiwarką. W większości przypadków potrzebne jest rozwiązanie pośrednie.

W tym rozdziale systemy pomocy podzielimy na dwie kategorie:

- **Nieoficjalne systemy pomocy** — w takich systemach do wyświetlania pomocy używane są zwykłe standardowe komponenty Excela (takie jak na przykład formularze *UserForm*). Zamiast tego możesz również zapewnić odpowiednią pomoc w postaci plików tekstowych, dokumentów edytora Word czy plików w formacie PDF.
- **Oficjalne systemy pomocy** — skompilowane pliki *CHM*, utworzone przy użyciu *Microsoft HTML Help Workshop*.

Przygotowanie skompilowanego pliku pomocy nie jest zadaniem łatwym, ale warto je wykonać, jeżeli aplikacja jest złożona lub przeznaczona dla wielu użytkowników.



Począwszy od pakietu Microsoft Office 2007, firma Microsoft zarzuciła stosowanie plików pomocy w formacie CHM na korzyść zupełnie nowego (i o wiele bardziej skomplikowanego) systemu pomocy o nazwie MS Help 2. W naszej książce nie będziemy się zajmować tym systemem.



Wszystkie przykłady zaprezentowane w tym rozdziale są dostępne na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>). Ponieważ większość przykładów składa się z wielu różnych plików, każdy przykład został umieszczony w osobnym folderze.

Systemy pomocy wykorzystujące komponenty Excela

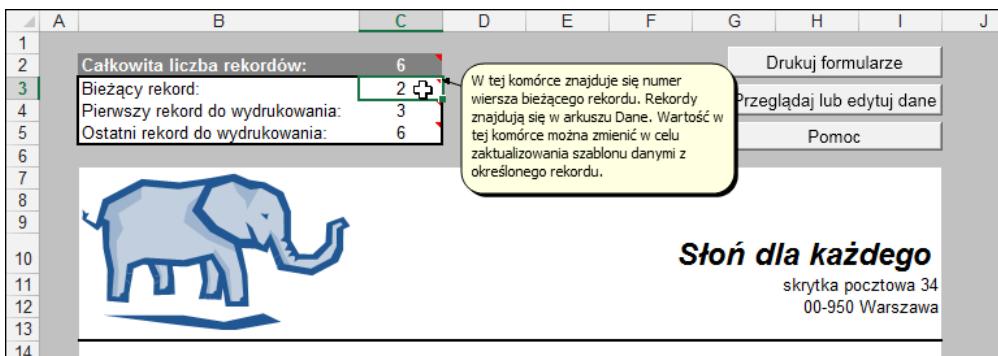
Jedną z najprostszych metod definiowania systemu pomocy jest wykorzystanie wbudowanych właściwości Excela. Najważniejszą zaletą tej metody jest fakt, że nie trzeba uczyć się tworzenia plików pomocy w języku HTML — co stanowi jeden z najważniejszych problemów i może zająć więcej czasu niż opracowanie całej aplikacji.

W tym podrozdziale zamiesczę przegląd różnych technik pomocy, w których wykorzystuje się następujące wbudowane komponenty Excela:

- **Komentarze do zawartości komórek** — jest to jeden z najprostszych sposobów tworzenia systemów pomocy.
- **Pola tekstowe** — wystarczy zdefiniować proste makro, które wyświetla pole tekstowe zawierające tekst pomocy.
- **Arkusz** — prostym sposobem definiowania systemu pomocy jest wstawienie nowego arkusza, wprowadzenie informacji i nadanie arkuszowi nazwy *Pomoc*; kiedy użytkownik kliknie zakładkę arkusza, nastąpi jego uaktywnienie.
- **Formularz UserForm** — istnieje wiele technik wyświetlania pomocy za pomocą okien formularzy *UserForm*.

Wykorzystanie komentarzy do tworzenia systemów pomocy

Jednym z najprostszych systemów pomocy są komentarze do zawartości komórek. Technika ta najbardziej nadaje się do opisywania typu danych wejściowych, które należy wprowadzać w komórkach. Kiedy użytkownik umieści wskaźnik myszy w komórce zawierającej komentarz, wyświetla się niewielkie okno wskazówki ekranowej (patrz rysunek 22.1). Kolejną zaletą takiego systemu pomocy jest to, że nie wymaga definiowania żadnych makr.



Rysunek 22.1. Wykorzystanie komentarza do wyświetlania pomocy

Komentarze w komórkach można wyświetlać automatycznie. Poniższa instrukcja VBA, którą możesz umieścić w procedurze Workbook_Open, zapewnia wyświetlanie znaczników we wszystkich komórkach, w których zdefiniowano komentarze:

```
Application.DisplayCommentIndicator = xlCommentIndicatorOnly
```



Skoroszyt z tym przykładem (*Komentarze\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).



Wiele użytkowników zapewne nie zdaje sobie z tego sprawy, ale w komentarzach można również wyświetlać obrazy. Aby to zrobić, kliknij obramowanie komentarza prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Formatuj komentarz*. Na ekranie pojawi się okno dialogowe *Formatowanie komentarza*. Przejdź na kartę *Kolory i linie*. Rozwiń listę *Kolor* i wybierz opcję *Efekty wypełnienia*. W oknie dialogowym *Efekty wypełnienia*, które pojawi się na ekranie, przejdź na kartę *Obraz* i naciśnij przycisk *Wybierz obraz*.

Innym sposobem wyświetlania pomocy jest wykorzystanie polecenia *Poprawność danych*, które powoduje wyświetlenie okna dialogowego umożliwiającego wprowadzenie kryteriów poprawności danych dla komórki lub zakresu. Aby przywołać to okno na ekran, przejdź na kartę *DANE*, naciśnij przycisk *Poprawność danych*, znajdujący się w grupie opcji *Narzędzia danych*, i wybierz z menu podręcznego polecenie *Poprawność danych*. Możesz spokojnie zignorować opcje sprawdzania poprawności danych, przejść na kartę *Komunikat wejściowy* i zdefiniować wiadomość, która będzie wyświetlana na ekranie w momencie uaktywnienia tej komórki. Objętość tekstu jest ograniczona do około 250 znaków.

Wykorzystanie pól tekstowych do wyświetlania pomocy

Stosunkowo łatwo można opracować system pomocy wykorzystujący pola tekstowe. Aby to zrobić, wystarczy utworzyć pole tekstowe (przejdź na kartę *WSTAWIANIE* i naciśnij przycisk *Pole tekstowe*, znajdujący się w grupie opcji *Tekst*), wprowadzić tekst pomocy i odpowiednio go sformatować.



Zamiast pola tekstowego możesz użyć innego kształtu i umieścić na nim tekst pomocy. Aby wybrać inny kształt, przejdź na kartę *WSTAWIANIE*, naciśnij przycisk *Kształty*, znajdujący się w grupie opcji *Ilustracje*, i wybierz z menu podręcznego żądaną kształt. Teraz wystarczy już tylko umieścić na nim tekst pomocy.

Przykład kształtu wykorzystanego do wyświetlania pomocy pokazano na rysunku 22.2. Aby uzyskać wrażenie unoszenia się obiektu pomocy nad arkuszem, do kształtu dodano efekt cienia.

Przez większość czasu nie będziesz chciał, aby pole tekstowe było widoczne. Do jego ukrywania i wyświetlania można zdefiniować przycisk związany z makrem, które odpowiednio ustawia właściwość *Visible* pola tekstowego. Przykład takiego makra zaprezentowano poniżej. W tym przypadku pole tekstowe występuje pod nazwą *HelpText*.

```
Sub ToggleHelp()  
    ActiveSheet.TextBoxes("HelpText").Visible = _  
        Not ActiveSheet.TextBoxes("HelpText").Visible  
End Sub
```

Aplikacja Szablon listu

Opis: Niniejsza aplikacja umożliwia drukowanie listów firmy Słoń dla każdego.

Określanie rekordów do wydrukowania: w komórce C4 należy wprowadzić pierwszy rekord do wydrukowania, natomiast w komórce C5 - ostatni rekord. Aby wydrukować tylko jeden rekord, należy w obu komórkach - C4 i C5 wprowadzić te same wartości.

Drukowanie listów: przed wydrukowaniem listów, należy sprawdzić, czy wprowadzono pierwszy i ostatni rekord do wydrukowania. Następnie należy kliknąć przycisk *Drukuj listy*.

Przeglądanie lub edycja danych: należy kliknąć przycisk *Przeglądaj lub edytuj dane*. Uaktywni się arkusz *Dane*, gdzie można przeglądać lub modyfikować rekordy.

Przeglądanie określonego rekordu: dane wyświetlane w szablonie listu odpowiadają rekordowi, którego numer wyświetla się w komórce C3. Aby przeglądać określony rekord, należy wprowadzić numer rekordu w komórce C3.

(Aby powrócić do aplikacji kliknij przycisk *Włącz/wyłącz pomoc*)

Proszę pamiętać, że zajmowanie się słoniem zajmuje dużo czasu.
Z naszego doświadczenia wynika także, że słonie nieźle dobrze się czują w obszarach wysoce zurbanizowanych.
Szczególnie nie lubią mieszkać w wysokich wieżowcach.

Rysunek 22.2. Wykorzystanie kształtu do wyświetlania pomocy dla użytkownika



Skoroszyt z tym przykładem (*PoleTekstowe\Szablon listu.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wykorzystanie arkusza do wyświetlania tekstu pomocy

Innym łatwym sposobem utworzenia systemu pomocy w aplikacji jest zdefiniowanie makra, które uaktywnia oddzielny arkusz z tematami pomocy. Wystarczy powiązać makro z przyciskiem w arkuszu, przyciskiem na pasku narzędzi lub pozycją menu, potem kliknąć... i mamy prosty system pomocy.

Na rysunku 22.3 pokazano prosty arkusz wykorzystywany jako system pomocy. Zdefiniowałem w nim zakres komórek zawierający tekst pomocy, sformatowany tak, aby wyglądał jak strona z notatnika.

Aby uniemożliwić użytkownikowi przewijanie arkusza ArkuszPomoc poza obszar, w którym zdefiniowano tekst pomocy, za pomocą makra ustalono właściwość ScrollArea arkusza. Ponieważ ta właściwość nie jest zapisana w arkuszu, nie ma konieczności jej ustawiania w momencie uaktywniania skoroszytu.

Rysunek 22.3.
Użycie oddzielnego arkusza jest łatwym sposobem zdefiniowania systemu pomocy

```
Sub ShowHelp()
    ' Aktywuj arkusz pomocy
    Worksheets("HelpSheet").Activate
    ActiveSheet.ScrollArea = "A1:C35"
    Range("A1").Select
End Sub
```

Arkusz został także zabezpieczony w celu uniemożliwienia użytkownikowi dokonywania zmian w tekście. Dodatkowo został „zamrożony” pierwszy wiersz, tak aby był zawsze widoczny przycisk *Powrót do szablonu*, niezależnie od tego, do którego miejsca użytkownik przewinął tekst pomocy.

Niestety podstawową wadą korzystania z tej techniki jest to, że główny obszar roboczy arkusza nie jest widoczny. Rozwiązaniem może być napisanie makra, które otwiera nowe okno w celu wyświetlenia arkusza z tekstem pomocy.



Skoroszyt z tym przykładem (*Arkusz\Szablon listu.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pwm.htm>).

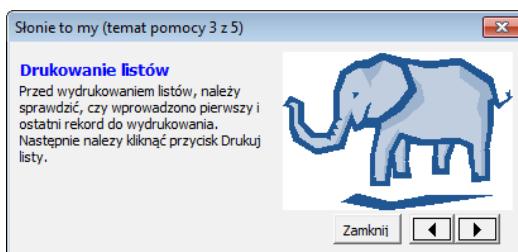
Wyświetlanie pomocy w oknie formularza UserForm

Inny sposób tworzenia systemu pomocy polega na wyświetlaniu tekstu pomocy w oknach formularza *UserForm*. Kilka technik tworzenia systemów pomocy w oparciu o formularze *UserForm* opisano w kolejnych podpunktach.

Zastosowanie etykiet do wyświetlania tekstu pomocy

Na rysunku 22.4 zaprezentowano okno formularza *UserForm*, w którym umieszczono dwie etykiety (obiekty *Label*): jedną dla nagłówka tematu oraz drugą dla właściwego tekstu pomocy. Formant *SpinButton* umożliwia użytkownikowi nawigację pomiędzy tematami pomocy. Sam tekst pomocy jest zapisany w arkuszu, gdzie tematy są przechowywane w kolumnie A arkusza, a tekst w kolumnie B.

Rysunek 22.4.
Kliknięcie przycisków
nawigacyjnych
(w postaci formantu
SpinBox) powoduje
zmianę tekstu
wyświetlanego
za pomocą etykiet



Kliknięcie przycisku powoduje wykonanie poniższej procedury. Jej działanie polega na ustawieniu właściwości *Caption* dwóch etykiet na tekst w odpowiednim wierszu arkusza (o nazwie *ArkuszPomoc*).

```
Private Sub SpinButton1_Change()
    HelpTopic = SpinButton1.Value
    LabelTopic.Caption = Sheets("ArkuszPomoc").Cells(HelpTopic, 1)
    LabelText.Caption = Sheets("ArkuszPomoc").Cells(HelpTopic, 2)
    Me.Caption = APPNAME & " (temat pomocy " & HelpTopic & " z " & SpinButton1.Max & ")"
End Sub
```

APPNAME to stała globalna, w której przechowywana jest nazwa aplikacji.



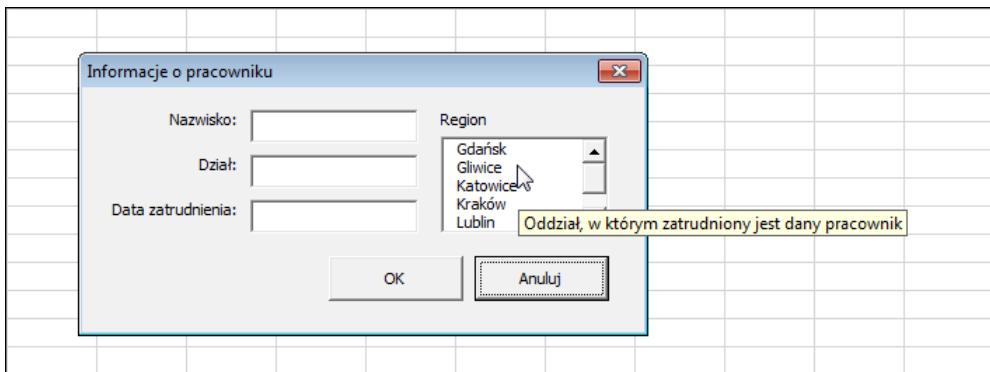
Skoroszyt z tym przykładem (*UserForm1\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Zastosowanie przewijanej etykiety do wyświetlania tekstu pomocy

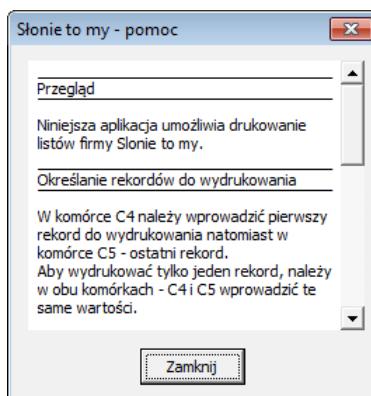
W tej technice tekst pomocy jest wyświetlany za pomocą jednej etykiety umieszczonej w ramce (obiekt *Frame*) wyposażonej w pionowy pasek przewijania (sama etykieta nie może posiadać paska). Przykład okna formularza *UserForm* skonfigurowanego w ten sposób zaprezentowano na rysunku 22.5.

Zastosowanie właściwości ControlTipText formularza UserForm

Każdy formant formularza *UserForm* posiada właściwość *ControlTipText*, która może przechowywać krótki, tekstowy opis formantu. Kiedy użytkownik ustawi wskaźnik myszy nad wybranym formantem, wyświetlana jest podpowiedź ekranowa z opisem formantu. Na rysunku poniżej zaprezentowano przykład takiego rozwiązania.



Rysunek 22.5.
Wstawienie etykiety
wewnętrz ramki
umożliwia przewijanie
etykiety



Tekst wyświetlany za pomocą etykiety jest odczytywany z arkusza o nazwie *ArkuszPomoc* w momencie inicjalizacji okna *UserForm*. Poniżej przedstawiono kod procedury *UserForm_Initialize* dla tego arkusza.

```
Private Sub UserForm_Initialize()
    Dim LastRow As Long
    Dim r As Long
    Dim txt As String
    Me.Caption = APPNAME & " Help"
    LastRow = Sheets("ArkuszPomoc").Cells(Rows.Count, 1).End(xlUp).Row
    txt = ""
    For r = 1 To LastRow
        txt = txt & Sheets("ArkuszPomoc").Cells(r, 1) _
            .Text & vbCrLf
    Next r
    With Label1
        .Top = 0
    End With
End Sub
```

```

    .Caption = txt
    .Width = 260
    .AutoSize = True
End With
With Frame1
    .ScrollHeight = Label1.Height
    .ScrollTop = 0
End With
End Sub

```

Kod modyfikuje właściwość ScrollHeight obiektu Frame, aby zapewnić obsługę przewijania dla całego tekstu zapisanego w etykiecie. Podobnie jak w poprzednim przykładzie, tak i tu APPNAME jest globalną stałą zawierającą nazwę aplikacji.

Ponieważ za pomocą etykiety nie można wyświetlać sformatowanego tekstu, wykorzystałem znaki podkreślenia w arkuszu ArkuszPomoc w celu podkreślenia tytułów tematów pomocy.

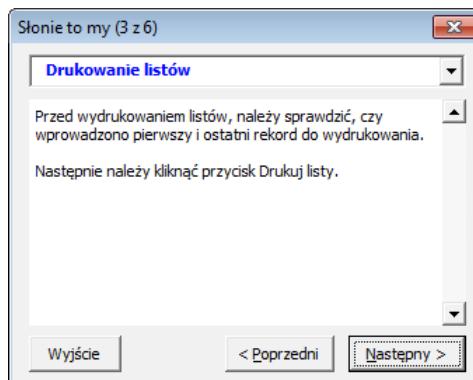


Skoroszyt z tym przykładem (*UserForm2\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Wykorzystanie pola kombi do wybierania tematów pomocy

Przykład zaprezentowany w tym podpunkcie jest ulepszoną wersją przykładu omawianego w poprzednim punkcie. Na rysunku 22.6 pokazano okno formularza *UserForm* zawierające formant ComboBox oraz etykietę (formant Label). Użytkownik może wybrać temat z listy rozwijanej lub przeglądać tematy, odpowiednio klikając przyciski *Poprzedni* i *Następny*.

Rysunek 22.6.
Wybór tematu z listy rozwijanej



Takie rozwiązanie jest nieco bardziej skomplikowane, ale oferuje większe możliwości. Do wyświetlania tekstu o dowolnej długości wykorzystano opisaną poprzednio technikę polegającą na umieszczeniu etykiety wewnętrznych przewijanej ramki.

Tekst pomocy jest zapisany w arkuszu o nazwie ArkuszPomoc w dwóch kolumnach (A i B). W pierwszej kolumnie są zapisane nagłówki tematów, natomiast w drugiej — tekst. W procedurze UserForm_Initialize dodawane są pozycje pola kombi. CurrentTopic jest zmienną poziomu modułu, której wartość odpowiada numerowi tematu pomocy.

```
Private Sub UpdateForm()
    ComboBoxTopics.ListIndex = CurrentTopic - 1
    Me.Caption = HelpFormCaption &
        " (" & CurrentTopic & " of " & TopicCount & ")"
    
    With LabelText
        .Caption = HelpSheet.Cells(CurrentTopic, 2)
        .AutoSize = False
        .Width = 212
        .AutoSize = True
    End With
    With Frame1
        .ScrollHeight = LabelText.Height + 5
        .ScrollTop = 1
    End With
    
    If CurrentTopic = 1 Then
        NextButton.SetFocus
    ElseIf CurrentTopic = TopicCount Then
        PreviousButton.SetFocus
    End If
    PreviousButton.Enabled = CurrentTopic <> 1
    NextButton.Enabled = CurrentTopic <> TopicCount
End Sub
```



Skoroszyt z tym przykładem (*UserForm3\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pzw.htm>).

Wyświetlanie pomocy w oknie przeglądarki sieciowej

W tym podrozdziale omówiono dwa sposoby wyświetlania pomocy w oknie przeglądarki sieciowej.

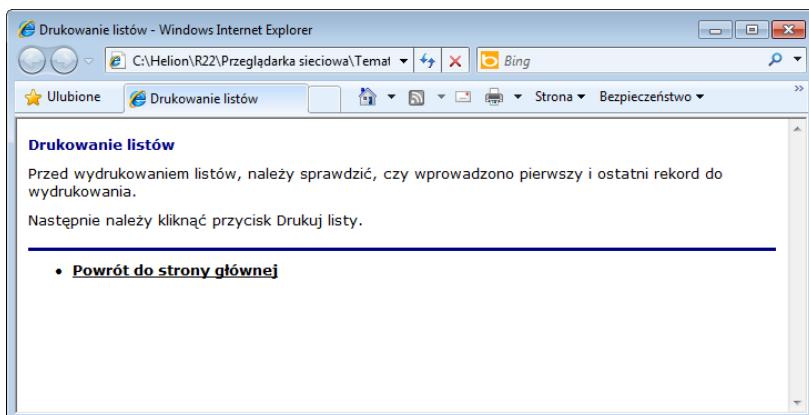
Zastosowanie plików w formacie HTML

Kolejnym sposobem wyświetlania pomocy dla aplikacji programu Excel jest utworzenie jednego lub więcej plików w formacie HTML i utworzenie łącza, którego kliknięcie będzie wyświetlało plik pomocy w oknie przeglądarki sieciowej. Pliki HTML mogą być przechowywane lokalnie na dysku twardym lub na przykład w korporacyjnej sieci intranet. Łącze do pliku pomocy możesz utworzyć w dowolnej komórce arkusza (nie musisz tworzyć żadnego makra). Na rysunku 22.7 przedstawiono przykład pliku pomocy wyświetlonego w oknie przeglądarki sieciowej.

Proste w użyciu edytory HTML są powszechnie dostępne, zatem system pomocy dla Twojej aplikacji może być w zależności od potrzeb albo bardzo prosty, albo bardzo złożony. Wadą tej metody jest jednak to, że oprócz skoroszytu musisz przekazywać użytkownikom niezadko dużą liczbę dodatkowych plików HTML. Jednym z rozwiązań tego problemu jest zastosowanie plików w formacie MHTML, które omówimy już za moment.

Rysunek 22.7.

Wyświetlanie pomocy w oknie przeglądarki sieciowej



Skoroszyt z tym przykładem (*Przeglądarka sieciowa\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zastosowanie plików w formacie MHTML

Formant MHTML (ang. *MIME HyperText Markup Language*) to format pozwalający na zapisywanie całej zawartości stron internetowych w jednym pliku (tworzenie pliku archiwum stron internetowych). Zawartość plików MHTML może być wyświetlana przy użyciu przeglądarki sieciowej Internet Explorer (i kilku innych przeglądarek sieciowych).

Dużą zaletą zastosowania plików w formacie MHTML do utworzenia pomocy systemowej jest fakt, że takie pliki można tworzyć bezpośrednio z poziomu Excela. Aby to zrobić, przygotuj tekst pomocy na dowolnej liczbie arkuszy, następnie przejdź na kartę *PLIK* i wybierz z menu polecenie *Zapisz jako*. Na ekranie pojawi się okno dialogowe *Zapisywanie jako*. Rozwiń listę *Zapisz jako typ* i wybierz z niej opcję *Jednoplikowa strona sieci Web (*.mht; *.mhtml)*. Pamiętaj, że w tym formacie pliku makra VBA nie zostaną zapisane.

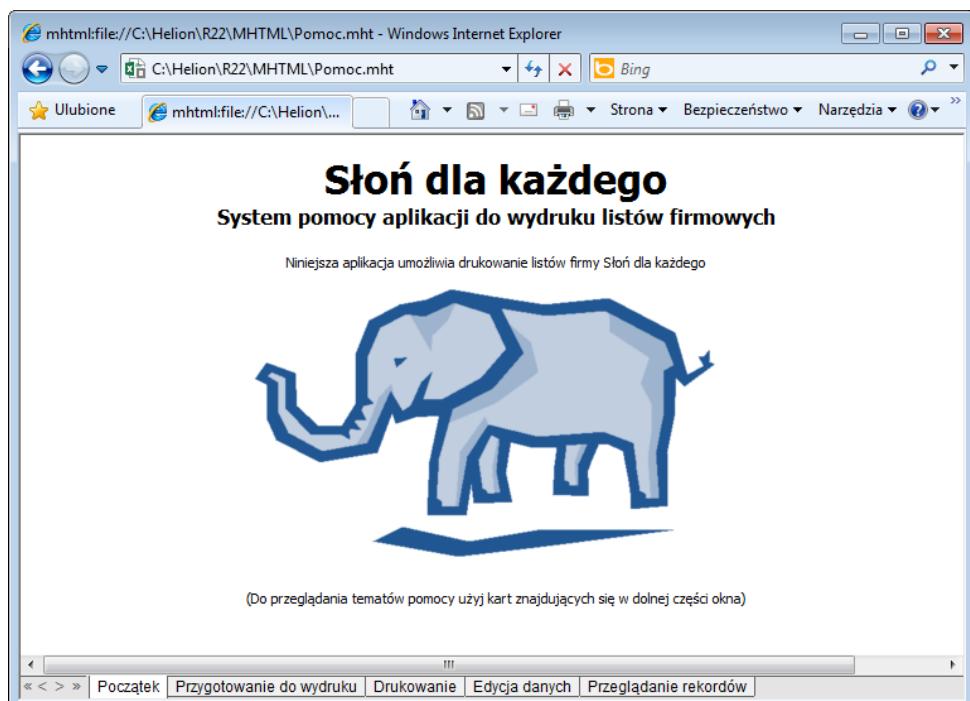
W Excelu możesz utworzyć hiperłącze, którego kliknięcie będzie wyświetlało plik pomocy zapisany w formacie MHTML.

Na rysunku 22.8 przedstawiono wygląd pliku MHTML wyświetlonego w oknie przeglądarki Internet Explorer. Zwróć uwagę na przyciski kart, znajdujące się w dolnej części okna pomocy, które pozwalają na szybkie przechodzenie do wybranych tematów pomocy. Jak się zapewne domyślasz, poszczególne karty w oknie pomocy odpowiadają kartom poszczególnych arkuszy skoroszytu, którego użyjemy do utworzenia pliku MHTML.



Skoroszyt z tym przykładem (*MHTML\Szablon listu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Z nieznanych przyczyn niektóre wersje programu Internet Explorer nie potrafią wyświetlać plików MHTML wywoływanych za pomocą hiperłączy bezpośrednio z Excela w sytuacji, kiedy nazwa pliku lub ścieżka zawiera spację. Gdy hiperłącze zawiedzie, przykładowy skoroszyt do wyświetlania zawartości pliku MHTML wykorzystuje wywołanie funkcji Windows API (`ShellExecute`).



Rysunek 22.8. Wyświetlanie pliku MHTML w oknie przeglądarki sieciowej



Jeżeli zapisujesz w formacie MHTML skoroszyty zawierające wiele arkuszy, otrzymany plik będzie zawierał kod JavaScript. Otwieranie takiego pliku w przeglądarce może spowodować wyświetlenie na ekranie ostrzeżenia.

Wykorzystanie systemu HTML Help

Obecnie w aplikacjach Windows jednym z najczęściej używanych systemów pomocy jest *HTML Help*, w którym są wykorzystywane skompilowane pliki z rozszerzeniem *.chm*. System *HTML Help* zastąpił starszy system pomocy *Windows Help System (WinHelp)*, w którym wykorzystywano pliki *HLP*. Obydwa systemy umożliwiają programistę powiązanie identyfikatora kontekstu z określonym tematem pomocy. Dzięki temu można wyświetlać temat pomocy w określonym kontekście.

Office XP był ostatnią wersją pakietu Microsoft Office, która wykorzystywała system *HTML Help*. Każda kolejna wersja pakietu Office używała innego systemu pomocy. Chociaż *HTML Help* nie potrafi naśladować wyglądu i sposobu działania nowego systemu Microsoft Office Help, to jednak jest nadal bardzo popularny ze względu na prostotę, funkcjonalność i łatwość tworzenia tematów pomocy, przynajmniej dla niezbyt rozbudowanych aplikacji.

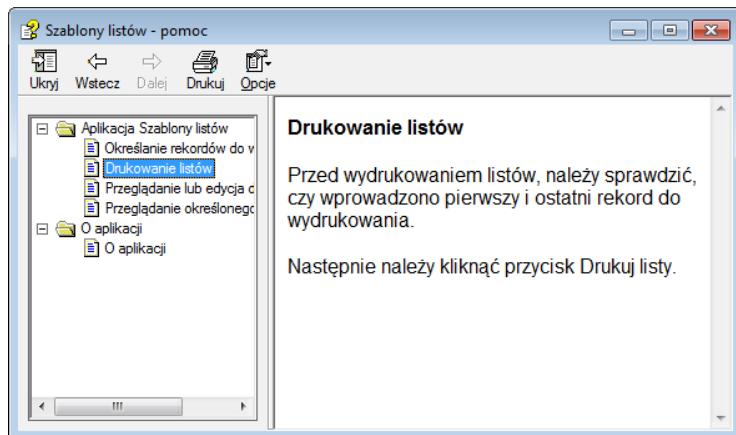
W tym podrozdziale zwięzle opiszę program wspomagający tworzenie plików systemu *HTML Help*. Szczegółowy opis tworzenia takich systemów pomocy wykracza poza ramy tej książki. Jednak większość potrzebnych informacji wraz z odpowiednimi przykładami można znaleźć w sieci Internet.



Tym, którzy zamierają tworzyć rozbudowane systemy pomocy, zalecam zakupienie odpowiedniego narzędzia, znacznie ułatwiającego wykonanie zadania. Takie oprogramowanie wykonuje za programistę wiele żmudnych operacji. Wśród systemów wspomagających tworzenie systemów pomocy są programy typu freeware, shareware oraz produkty komercyjne.

Microsoft opracował system *HTML Help* jako standard systemów pomocy w aplikacjach użytkowych. Działanie systemu polega na skompilowaniu serii plików HTML w celu utworzenia spójnego systemu pomocy. Dodatkowo można utworzyć spis treści i indeks, a także zdefiniować słowa kluczowe w celu utworzenia zaawansowanych hiperłączy. W systemie *HTML Help* można także wykorzystywać dodatkowe narzędzia, takie jak pliki graficzne, formanty ActiveX, skrypty oraz kod DHTML (ang. *Dynamic HTML*). Przykład systemu pomocy *HTML Help* pokazano na rysunku 22.9.

Rysunek 22.9.
Przykład systemu pomocy *HTML Help*

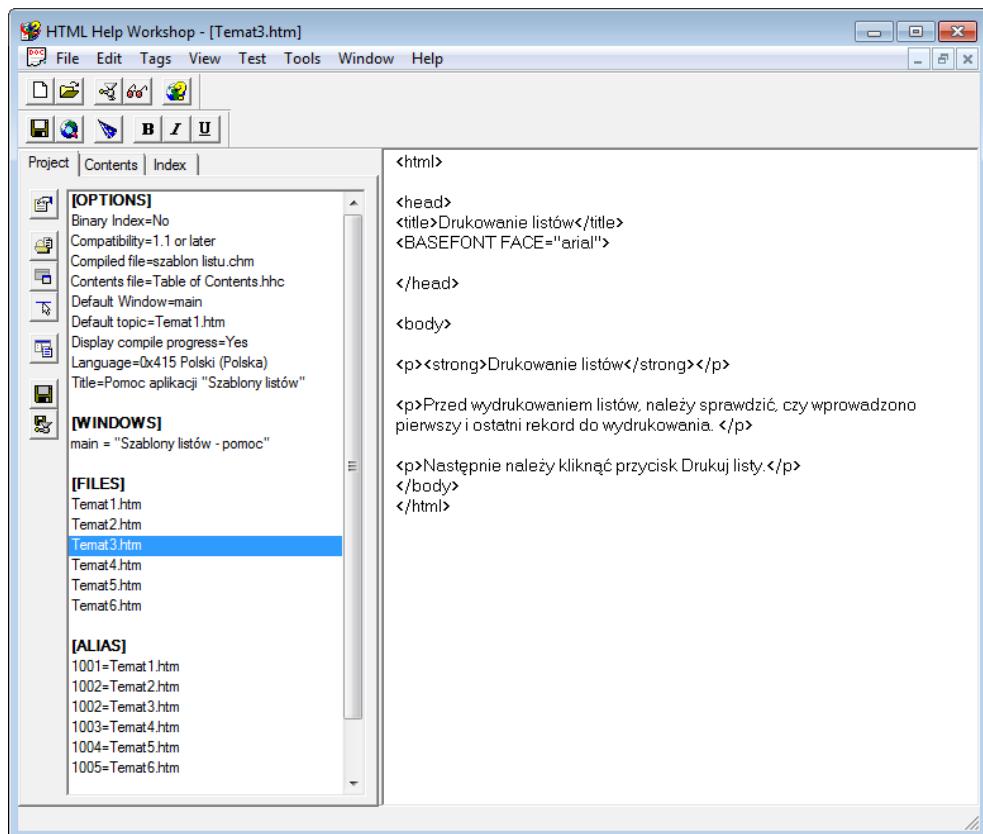


Skoroszyt z tym przykładem (*HTML Help\Szablon listu.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

Pliki pomocy *HTML Help* są wyświetlane w przeglądarce *HTML Help Viewer*, która wykorzystuje mechanizmy przeglądarki *Internet Explorer*. Informacje pojawiają się w oknie, a spis treści, indeks oraz narzędzia wyszukiwania w wydzielonym panelu. Dodatkowo tekst pomocy może zawierać standardowe hiperłącza, służące do wyświetlania dodatkowych tematów lub nawet dokumentów z sieci Internet. Co istotne, w systemie *HTML Help* istnieje możliwość dostępu do plików zapisanych w witrynach WWW. W ten sposób można kierować użytkowników do źródła najaktualniejszych informacji, które nie były dostępne w momencie tworzenia systemu pomocy.

Do tworzenia systemów pomocy *HTML Help* potrzebny jest specjalny kompilator. Program *HTML Help Workshop* i jego dokumentację można pobrać za darmo ze strony internetowej Microsoft MSDN: <http://msdn.microsoft.com>. Po otwarciu tej strony w przeglądarce sieciowej poszukaj hasła *HTML Help Workshop*.

Na rysunku 22.10 przedstawiono okno programu *HTML Help Workshop* z załadowanym plikiem projektu, przy użyciu którego wygenerowany został plik pomocy przedstawiony na rysunku 22.9.



Rysunek 22.10. Zastosowanie programu *HTML Help Workshop* do tworzenia pliku pomocy

Wykorzystanie metody Help do wyświetlania pomocy w formacie HTML Help

Metoda `Help` obiektu `Application` umożliwia wyświetlanie pomocy w jednym z dwóch formatów: *HLP* systemu *WinHelp* oraz *CHM* systemu *HTML Help*. Metoda działa nawet wtedy, kiedy w pliku pomocy nie zostaną zdefiniowane żadne identyfikatory kontekstu. Jej składnia jest następująca:

```
Application.Help(plikPomocy, ID_KontekstuPomocy)
```

Obydwa argumenty są opcjonalne. W przypadku pominięcia nazwy pliku pomocy wyświetlony zostanie plik pomocy Excela. W przypadku pominięcia argumentu opisującego identyfikator kontekstu wyświetlony będzie domyślny temat podanego pliku pomocy.

Wyświetlanie tematów pomocy programu Excel

W niektórych przypadkach bardzo użyteczna może być możliwość wyświetlania tematów pomocy programu Excel za pomocą kodu VBA. Każdy z tematów pomocy ma co prawda swój unikatowy identyfikator ID, ale odszukanie danego identyfikatora w Excelu 2013 nie jest takie trywialne. Założymy na przykład, że chcesz udostępnić użytkownikowi tematy pomocy dotyczące funkcji arkuszo-wej AGGREGATE.

Realizację takiego zadania powinieneś rozpoczęć od znalezienia identyfikatora tematu pomocy (ang. *Topic ID*). Aby to zrobić, odszukaj interesujący Cię temat pomocy w systemie pomocy Excela, wyświetl go, zaznacz prowadzące do niego łącze i naciśnij kombinację klawiszy *Ctrl+C*, aby je skopiować. Następnie przejdź do pustej komórki arkusza i wklej skopiowane łącze, naciskając kombinację klawiszy *Ctrl+V*. Teraz przywołaj na ekran okno dialogowe *Edytowanie hiperłącza*, na przykład klikając komórkę prawym przyciskiem myszy i wybierając z menu podręcznego polecenie *Edytuj hiperłącze*. Przekonasz się, że adres pomocy dla funkcji AGGREGATE wygląda następująco:

```
ms-help://MS.EXCEL.15.1033/EXCEL/content/HA102753246.htm
```

Identyfikatorem tematu pomocy jest nazwa pliku HTML bez rozszerzenia *.htm*, czyli w naszym przypadku identyfikator to ciąg znaków HA102753246. Teraz możesz użyć identyfikatora tematu jako argumentu wywołania metody *ShowHelp* obiektu *Assistance*. Przykładowe wywołanie może wyglądać następująco:

```
Application.Assistance.ShowHelp "HA102753246"
```

Po wykonaniu takiego polecenia system pomocy Excela powinien wyświetlić informacje na temat funkcji AGGREGATE.

Innym rozwiązaniem jest wykorzystanie metody *SearchHelp*. Jako argument metody wystarczy podać poszukiwaną frazę, a na ekranie zostanie wyświetlony szereg tematów pomocy pasujących do podanego wzorca. A oto przykład takiego rozwiązania:

```
Application.Assistance.SearchHelp "funkcja AGGREGATE"
```

Poniższy kod wyświetla domyślny temat pliku *MojaAplikacja.chm* przy założeniu, że plik ten znajduje się w tym samym katalogu, co skoroszyt, z którego go wywołano. Zwróćmy uwagę, że drugi argument został pominięty.

```
Sub ShowHelpContents()
    Application.Help ThisWorkbook.Path & "\MojaAplikacja.chm"
End Sub
```

Poniższa instrukcja wyświetla temat pomocy o identyfikatorze kontekstu 1002 z pliku pomocy *HTML Help* o nazwie *MojaAplikacja.chm*:

```
Application.Help ThisWorkbook.Path & "\MojaAplikacja.chm", 1002
```

Łączenie pliku pomocy z aplikacją

Plik pomocy systemu *HTML Help* można powiązać z aplikacją Excela albo za pomocą okna dialogowego *Project Properties*, albo poprzez zdefiniowanie kodu w języku VBA.

W edytorze *Visual Basic* należy wybrać polecenie *Tools/xxx Properties* (gdzie *xxx* jest nazwą projektu). Wyświetli się okno dialogowe *Project Properties*, w którym należy kliknąć zakładkę *General* i wprowadzić skompilowany plik pomocy systemu *HTML Help* dla projektu. Plik powinien mieć rozszerzenie *.chm*.

Poniższa instrukcja powoduje powiązanie aplikacji z systemem pomocy *Mojefunkcje.chm*. Przyjęto założenie, że plik pomocy znajduje się w tym samym katalogu, co skoroszyt.

`ThisWorkbook.VBProject.HelpFile = ThisWorkbook.Path & "\Mojefunkcje.chm"`



Uwaga

Jeżeli próba wykonania powyższego polecenia zakończy się wystąpieniem błędu, musisz zezwolić na programowy dostęp do modelu obiektowego projektu VBA. Aby to zrobić, przejdź do Excela, następnie kliknij kartę *DEVELOPER* i naciśnij przycisk *Bezpieczeństwo makr*, znajdujący się w grupie poleceń *Kod*. Na ekranie pojawi się okno dialogowe *Centrum zaufania*. Przejdz do prawej części okna i zaznacz opcję *Ufaj dostępowi do modelu obiektowego projektu VBA*.

Po skojarzeniu pliku pomocy z aplikacją tematy pomocy wyświetla się w następujących sytuacjach:

- kiedy użytkownik wciśnie klawisz *F1* w momencie, gdy w oknie dialogowym *Wstawianie funkcji* jest zaznaczona funkcja użytkownika;
- kiedy użytkownik wciśnie klawisz *F1* w momencie, gdy jest wyświetlone okno formularza *UserForm*. W tym przypadku wyświetli się temat pomocy dotyczący aktywnego formantu.

Przypisanie tematów pomocy do funkcji VBA

Utworzoną w języku VBA funkcję użytkownika, przeznaczoną do wykorzystania w arkuszach, można powiązać z plikiem pomocy oraz identyfikatorem kontekstu. Gdy te elementy zostaną przypisane, wciśnięcie klawisza *F1* w momencie, kiedy funkcja jest zaznaczona w oknie dialogowym *Wstawianie funkcji*, spowoduje wyświetlenie pomocy na temat tej funkcji.

Aby określić identyfikator kontekstu pomocy dla funkcji użytkownika, wykonaj następujące czynności:

1. Utwórz funkcję.
2. Upewnij się, czy z projektem skojarzono plik pomocy (opis tej czynności zamieszczono w poprzednim punkcie).
3. W edytorze *Visual Basic* wciśnij *F2*, aby uaktywnić przeglądarkę obiektów (*Object Browser*).
4. Na rozwijanej liście *Project/Library* wybierz projekt.
5. W oknie *Classes* zaznacz moduł zawierający funkcje.
6. Zaznacz funkcję w oknie *Members Of*.
7. Kliknij funkcję prawym przyciskiem myszy i z menu podręcznego wybierz pozycję *Properties*.

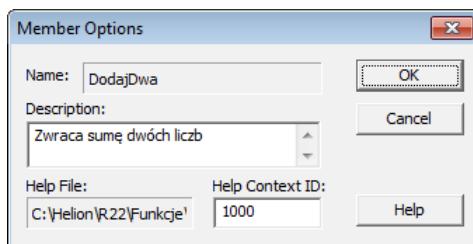
Na ekranie pojawi się okno dialogowe *Member Options*, jak pokazano na rysunku 22.11.

8. Wprowadź identyfikator kontekstu pomocy dla funkcji.

Możesz także wprowadzić opis funkcji.

Rysunek 22.11.

Definiowanie identyfikatora kontekstu pomocy dla funkcji użytkownika w oknie dialogowym Member Options



W oknie dialogowym *Member Options* nie można określić pliku pomocy. Zawsze używany jest plik pomocy związanego z projektem.

Do ustawienia identyfikatora kontekstu pomocy oraz pliku pomocy dla funkcji użytkownika można wykorzystać kod VBA. W tym celu należy posłużyć się metodą *MacroOptions*.

W poniższej procedurze użyto jej w celu określenia opisu, pliku pomocy oraz identyfikatora kontekstu pomocy dla dwóch funkcji użytkownika (*DodaJDwa* oraz *Kwadrat*):

```
Sub SetOptions()
    ' Ustawienie opcji dla funkcji AddTwo
    Application.MacroOptions Macro:="DodaJDwa", _
        Description:="Zwraca sumę dwóch liczb", _
        HelpFile:=ThisWorkbook.Path & "\mojefunkcje.chm", _
        HelpContextID:=1000
    ArgumentDescriptions:=Array("Pierwsza liczba do sumowania", _
        "Druga liczba do sumowania")

    ' Ustawienie opcji dla funkcji Squared
    Application.MacroOptions Macro:="Kwadrat", _
        Description:="Zwraca kwadrat liczby podanej jako argument", _
        HelpFile:=ThisWorkbook.Path & "\mojefunkcje.chm", _
        HelpContextID:=2000
    ArgumentDescriptions:=Array("Liczba, która zostanie podniesiona do kwadratu")
End Sub
```

Po wykonaniu tych procedur użytkownik może wyświetlić temat pomocy bezpośrednio z okna dialogowego *Wstawianie funkcji*, naciskając klawisz *F1* lub klikając łącze pomocy dla danej funkcji.



Skoroszyt z tym przykładem (*Funkcje\Moje funkcje.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Rozdział 23.

Tworzenie aplikacji przyjaznych dla użytkownika

W tym rozdziale:

- Czym jest aplikacja przyjazna dla użytkownika?
- Bliskie spotkanie z Kreatorem amortyzacji pożyczek, który generuje arkusze z harmonogramem spłaty pożyczki o stałej wysokości raty
- Zagadnienia i techniki programowania używane podczas tworzenia Kreatora amortyzacji pożyczek
- Wskazówki dotyczące projektowania aplikacji (lista kontrolna)

Czym jest aplikacja przyjazna dla użytkownika?

Termin „aplikacja przyjazna dla użytkownika” zarezerwowałem dla takich aplikacji, którymi z powodzeniem mogą posługiwać się użytkownicy o minimalnej wiedzy na temat Excela. Takie aplikacje pozwalają na uzyskanie zadowalających wyników nawet dla użytkowników, którzy nie mają żadnego doświadczenia w pracy z Exceliem.

Opisany w tym rozdziale Kreator amortyzacji pożyczek sklasyfikowałem właśnie jako aplikację przyjazną dla użytkownika, ponieważ skuteczne korzystanie z niego nie wymaga od użytkownika wiedzy na temat zawiłości Excela. Wystarczy, że użytkownik odpowie na kilka prostych pytań, a kreator utworzy przydatny i wygodny arkusz z gotowymi formułami.

Kreator amortyzacji pożyczek

Kreator amortyzacji pożyczek generuje arkusz zawierający harmonogram spłaty pożyczki o stałym oprocentowaniu. Harmonogram przedstawia informacje o pożyczce dla kolejnych miesięcy. Wyświetlane są takie dane o pożyczce, jak kwota płatności z wyszczególnieniem kapitału i odsetek oraz saldo pożyczki po spłacie.

Alternatywnym sposobem utworzenia takiego harmonogramu jest zdefiniowanie pliku szablonu. Jak się jednak szybko przekonasz, zastosowanie kreatora jest lepsze z kilku powodów.

Na rysunku 23.1 przedstawiono przykładowy harmonogram spłaty pożyczki wygenerowany przez naszego Kreatora amortyzacji pożyczek.

Rysunek 23.1.

Harmonogram amortyzacji wyświetla szczegółowy pożyczki zaciągniętej na 30 lat

	A	B	C	D	E	F	G	H
1	Harmonogram spłat pożyczki							
2	Przygotowany przez:	Najlepszy Bank w mieście!						
3	Data utworzenia:	1 marca 2013						
4								
5	Cena zakupu:	389 000 zł						
6	Wkład własny (%):	20,0%						
7	Wkład własny:	77 800 zł						
8	Kwota pożyczki:	311 200 zł						
9	Okres spłaty (miesiące):	360						
10	Oprocentowanie:	5,80%						
11	Pierwsza rata:	2013-03-01						
12								
13	Numer raty	Rok	Miesiąc	Rata	Odsetki	Kapitał	Saldo	
14	1	2013	3	1 825,98 zł	1 504,13 zł	321,84 zł	310 878,16 zł	
15	2	2013	4	1 825,98 zł	1 502,58 zł	323,40 zł	310 554,76 zł	
16	3	2013	5	1 825,98 zł	1 501,01 zł	324,96 zł	310 229,80 zł	
17	4	2013	6	1 825,98 zł	1 499,44 zł	326,53 zł	309 903,27 zł	
18	5	2013	7	1 825,98 zł	1 497,87 zł	328,11 zł	309 575,16 zł	
19	6	2013	8	1 825,98 zł	1 496,28 zł	329,70 zł	309 245,46 zł	
20	7	2013	9	1 825,98 zł	1 494,69 zł	331,29 zł	308 914,17 zł	
21	8	2013	10	1 825,98 zł	1 493,09 zł	332,89 zł	308 581,28 zł	
22	9	2013	11	1 825,98 zł	1 491,48 zł	334,50 zł	308 246,78 zł	
23	10	2013	12	1 825,98 zł	1 489,86 zł	336,12 zł	307 910,67 zł	
24	2013 Suma			18 259,75 zł	14 970,42 zł	3 289,33 zł	307 910,67 zł	
25	11	2014	1	1 825,98 zł	1 488,23 zł	337,74 zł	307 572,93 zł	
26	12	2014	2	1 825,98 zł	1 486,60 zł	339,37 zł	307 233,55 zł	
27	13	2014	3	1 825,98 zł	1 484,96 zł	341,01 zł	306 892,54 zł	
28	14	2014	4	1 825,98 zł	1 483,31 zł	342,66 zł	306 549,88 zł	
29	15	2014	5	1 825,98 zł	1 481,66 zł	344,32 zł	306 205,56 zł	
30	16	2014	6	1 825,98 zł	1 479,99 zł	345,98 zł	305 859,58 zł	
	«	»	Arkusz1	+				:

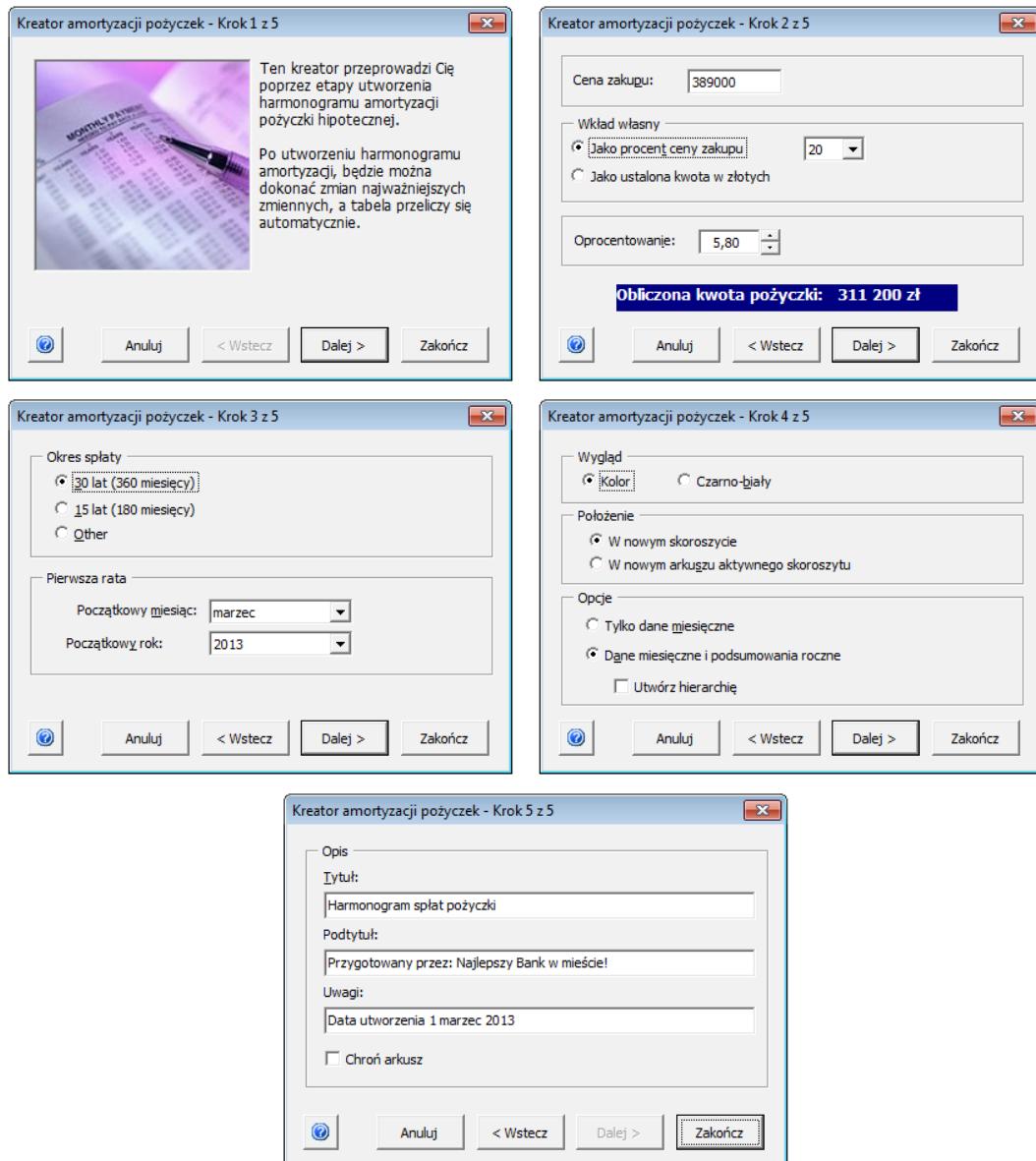


Dodatek z tym przykładem (*Kreator amortyzacji pożyczki.xlsx*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>). Dodatek nie jest zabezpieczony hasłem.

Obsługa Kreatora amortyzacji pożyczek

Kreator amortyzacji pożyczek składa się z pięciu okien dialogowych (kroków), za pomocą których są pobierane informacje od użytkowników. W typowy dla kreatorów sposób użytkownik może poruszać się w przód i w tył pomiędzy poszczególnymi oknami. Kliknięcie przycisku *Zakończ* powoduje utworzenie nowego arkusza. Jeżeli użytkownik kliknie *Zakończ*, zanim zostaną udzielone odpowiedzi na wszystkie pytania, wykorzystane będą wartości domyślne. Kliknięcie przycisku *Anuluj* powoduje zamknięcie okna UserForm bez wykonania żadnych działań.

W tej aplikacji wykorzystano jeden formularz *UserForm* zawierający formant *MultiPage*, służący do wyświetlania okien dialogowych dla pięciu kroków. Poszczególne okna dialogowe zaprezentowano na rysunku 23.2.



Rysunek 23.2. Pięć kolejnych okien dialogowych Kreatora amortyzacji pożyczek

Struktura skoroszytu Kreatora amortyzacji pożyczek

Kreator amortyzacji pożyczek składa się z następujących komponentów:

- FormMain — okno formularza *UserForm* służące jako główny interfejs użytkownika;
- FormHelp — okno formularza *UserForm*, w którym wyświetlana jest treść pomocy;
- FormMessage — okno formularza *UserForm*, które wyświetla informację powitalną po otwarciu dodatku. Użytkownik może wyłączyć wyświetlanie tego okna.
- HelpSheet — arkusz zawierający tekst pomocy;
- ModMain — moduł w języku VBA zawierający procedury wyświetlające główne okno formularza *UserForm*;
- ThisWorkbook — moduł kodu dla tego obiektu zawierający procedurę obsługi zdarzeń *Workbook_Open*.

Oprócz tego skoroszyt zawiera nieco kodu RibbonX XML, który tworzy na karcie *WSTAWIANIE* przycisk uruchamiający kreatora. Kod RibbonX dołączony do skoroszytu nie jest widoczny w Excelu.

Jak działa Kreator amortyzacji pożyczek?

Kreator amortyzacji pożyczek jest dodatkiem, stąd należy go zainstalować za pomocą okna dialogowego *Dodatki*. Aby przywołać na ekran to okno, przejdź na kartę *PLIK* i wybierz z menu polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. W oknie dialogowym opcji Excela kliknij kategorię *Dodatki*. Następnie z listy rozwijanej *Zarządzaj* wybierz opcję *Dodatki programu Excel* i naciśnij przycisk *Przejdź*. Na ekranie pojawi się okno dialogowe *Dodatki*. Naciśnij przycisk *Przeglądaj*, a następnie odszukaj żądanego pliku dodatku. Po zainstalowaniu dodatek pozostaje aktywny dla wszystkich sesji programu Excel. Pamiętaj jednak, że nasz kreator będzie działał równie dobrze, kiedy otworzymy go za pomocą polecenia *PLIK/Otwórz*.

Historia Kreatora amortyzacji pożyczek

Początkowo Kreator amortyzacji pożyczek był prostą aplikacją, która później przekształciła się w stosunkowo skomplikowany projekt. Moim podstawowym celem było zademonstrowanie jak największej liczby technik projektowych, a jednocześnie opracowanie użytecznego produktu. Chciałbym powiedzieć, że dokładnie przewidziałem, jaki będzie ostateczny efekt, zanim przystąpiłem do projektowania, ale... musiałbym skłamać.

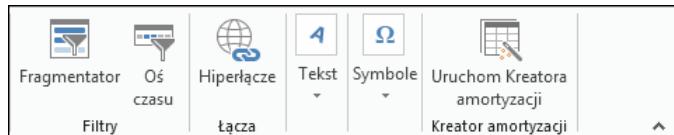
Mój pierwotny zamiar był znacznie mniej ambitny. Chciałem jedynie utworzyć aplikację, która pobiera dane wejściowe od użytkownika i tworzy arkusz. Jednak po rozpoczęciu tworzenia aplikacji zacząłem myśleć o sposobach jej usprawnienia. W efekcie napotkałem kilka ślepych uliczek. Niektórzy pewnie nazwą moje wysiłki stratą czasu, ale te falstarty były istotną częścią procesu projektowania.

Wykonałem ten projekt w ciągu jednego (długiego) dnia oraz spędziłem całkiem sporo dodatkowych godzin na jego ulepszaniu i testowaniu. Wersja dołączona do tego wydania książki zawiera kilka dodatkowych funkcji i kosmetycznych poprawek.

Modyfikacja interfejsu użytkownika

Każdy dodatek musi mieć jakiś sposób uruchamiania. W tym przypadku dołączyłem do naszego dodatku prosty kod RibbonX, który tworzy na karcie *WSTAWIANIE* nową grupę o nazwie *Kreator amortyzacji* i dodaje do niej przycisk uruchamiający kreatora (patrz rysunek 23.3). Naciśnięcie tego przycisku wykonuje procedurę *StartAmortizationWizard*, która wyświetla na ekranie okno formularza *FormMain*.

Rysunek 23.3.
Przycisk kreatora
w nowej grupie poleceń
na karcie Wstawianie



Kod RibbonX, który tworzy nowy przycisk na Wstążce, wygląda następująco:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
<ribbon>
<tabs>
<tab idMso="TabInsert">
<group id="gpUtils" label="Kreator amortyzacji">
<button id="b1"
size="large"
imageMso="CreateQueryFromWizard"
label="Uruchom Kreatora amortyzacji"
supertip="Kliknij tutaj aby uruchomić Kreatora amortyzacji pożyczek."
onAction="StartAmortizationWizard"/>
</group>
</tab>
</tabs>
</ribbon>
</customUI>
```

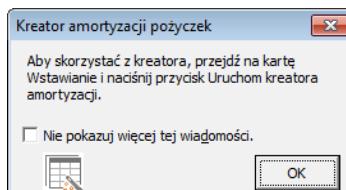


Więcej szczegółowych informacji na temat modyfikacji Wstążki znajdziesz w rozdziale 20.

Wyświetlanie wiadomości powitalnej

W ciągu ostatnich lat zainstalowałem wiele różnych dodatków programu Excel i muszę przyznać, że wiele z nich nie dawało użytkownikowi najmniejszej wskazówki, jak z nich korzystać po zainstalowaniu. Mając zatem na względzie zamiar utworzenia aplikacji jak najbardziej przyjaznej dla użytkownika, utworzyłem formularz *UserForm*, który jest automatycznie wyświetlany po załadowaniu skoroszytu. Komunikat umieszczony na formularzu po prostu informuje użytkownika o tym, jak uruchomić kreatora. Wygląd okna tego formularza został przedstawiony na rysunku 23.4.

Rysunek 23.4.
Wygląd formularza
UserForm
wyświetlanego
po załadowaniu
dodatku *Kreator
amortyzacji pożyczek*



Co ważne, na formularzu znajduje się opcja pozwalająca na wyłączenie tej wiadomości, tak że nie będzie już wyświetlana przy kolejnych otwarciaach skoroszytu.

Poniżej znajdziesz kod procedury `WorkBook_Open`, która wyświetla na ekranie okno formularza:

```
Private Sub Workbook_Open()
    If GetSetting(APPNAME, "Defaults", "ShowMessage", "Yes") = "Yes" Then
        FormMessage.Show
    End If
End Sub
```

Decyzja użytkownika co do wyświetlania okna podczas kolejnych uruchomień jest zapisywana w rejestrze systemu Windows. Nazwa klucza w rejestrze odpowiada nazwie aplikacji (przechowywanej w globalnej stałej `APPNAME`). Domyślnie decyzja ma wartość "Yes", dzięki czemu okno formularza jest wyświetlane co najmniej raz.

Poniżej znajdziesz kod procedury, która jest wykonywana, kiedy użytkownik naciśnie przycisk *OK*:

```
Private Sub OKButton_Click()
    If cbMessage Then
        SaveSetting APPNAME, "Defaults", "ShowMessage", "No"
    Else
        SaveSetting APPNAME, "Defaults", "ShowMessage", "Yes"
    End If
    Unload Me
End Sub
```

Kiedy użytkownik zaznaczy opcję *Nie pokazuj więcej tej wiadomości*, klucz zapisywany w rejestrze będzie miał wartość "No", dzięki czemu okno formularza nie zostanie już ponownie wyświetcone.

Iinicjalizacja formularza FormMain

Procedura `UserForm_Initialize` formularza `FormMain` wykonuje kilka istotnych czynności:

- Ustawia wartość właściwości `Style` formantu `MultiPage` na `fTabStyleNone`. Karty formantu są widoczne w edytorze VBE, dzięki czemu można łatwiej modyfikować formularz `UserForm`.
- Ustawia wartość właściwości `Value` formantu `MultiPage` na 0, dzięki czemu zawsze wyświetla się pierwsza karta formantu, niezależnie od wartości tej właściwości w momencie wykonywania ostatniej operacji zapisu skoroszytu.
- Dodaje elementy do trzech pól kombi wykorzystywanych w formularzu.
- Wywołuje procedurę `GetDefaults`, która pobiera ostatnio używane ustawienia z rejestrów Windows (więcej informacji znajduje się w podpunkcie „Zapisywanie i odtwarzanie ustawień domyślnych”).
- Sprawdza, czy skoroszyt jest aktywny. Jeżeli żaden skoroszyt nie jest aktywny, procedura wyłącza przycisk opcji umożliwiający utworzenie nowego arkusza w aktywnym skoroszycie. W Excelu 2013 skoroszyt jest zawsze aktywny ze

względzie na nowy, jednodokumentowy interfejs użytkownika, jednak pozostawiłem kod VBA bez zmian, ponieważ jest niezbędny do poprawnego działania ze starszymi wersjami Excela.

- Jeżeli skoroszyt jest aktywny, dodatkowy test sprawdza, czy struktura skoroszytu jest zabezpieczona. Jeżeli tak, procedura wyłącza przycisk opcji umożliwiający użytkownikowi utworzenie arkusza w aktywnym skoroszycie.

Przetwarzanie zdarzeń podczas wyświetlania formularza UserForm

Moduł kodu formularza FormMain zawiera kilka procedur obsługi zdarzeń Click i Change dla formantów formularza *UserForm*.



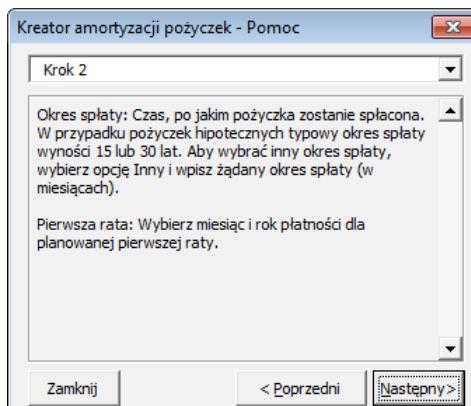
Kliknięcie przycisków *Dalej* i *Wstecz* decyduje o tym, która karta formantu *MultiPage* zostanie wyświetlona. Procedura *MultiPage1_Change* modyfikuje właściwość *Caption* formularza *UserForm* i odpowiednio uaktywnia przyciski *Dalej* i *Wstecz*. Więcej szczegółowych informacji na temat tworzenia kreatorów znajdziesz w rozdziale 13.

Wyświetlanie pomocy w kreatorze

Istnieje wiele możliwości wyświetlania tekstu pomocy. Wybrałem prosty sposób, wykorzystując formularz *UserForm* pokazany na rysunku 23.5. Wyświetla on tekst pomocy zapisany w arkuszu. Jak można zauważyć, jest to kontekstowy system pomocy. Kiedy użytkownik kliknie przycisk *Pomoc*, wyświetlany temat pomocy odpowiada bieżącej karcie formantu *MultiPage*.

Rysunek 23.5.

Pomoc jest wyświetlana w formularzu *UserForm*, do którego jest kopowany tekst zapisany w arkuszu



Arkusze dodatku nie są widoczne. Aby wyświetlić arkusz zawierający treść pomocy dla dodatku, powinieneś tymczasowo ustawić właściwość *IsAddin* dodatku na wartość *False*. Jednym ze sposobów na wykonanie takiej operacji jest zaznaczenie projektu VBA w oknie *Project* edytora VBE i wykonanie w oknie *Immediate* następującego polecenia:

```
ThisWorkbook.IsAddin = False
```



Więcej informacji na temat sposobów przenoszenia tekstu z arkusza do formularza *UserForm* znajdziesz w rozdziale 22.

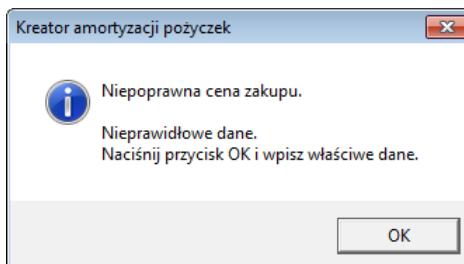
Tworzenie nowego arkusza

Kiedy użytkownik kliknie przycisk *Zakończ*, kreator rozpoczyna wykonywanie działań. Procedura obsługi zdarzenia *Click* dla tego przycisku wykonuje działania opisane poniżej.

- Wywołuje funkcję *DataIsValid*, która sprawdza poprawność danych wprowadzanych przez użytkownika. Jeżeli dane są poprawne, funkcja zwraca wartość *True* i program kontynuuje działanie. W przypadku odnalezienia niepoprawnych danych uaktywniany jest ten formant, którego ustawienia należy poprawić, a funkcja zwraca opisowy komunikat o błędzie (rysunek 23.6).

Rysunek 23.6.

W przypadku odnalezienia niepoprawnych danych automatycznie uaktywniany jest formant zawierający błąd



- Jeżeli dane wprowadzone przez użytkownika są poprawne, procedura tworzy nowy arkusz w aktywnym albo w nowym skoroszycie, w zależności od opcji zaznaczonej przez użytkownika.
- Zapisuje do arkusza *parametry pożyczki* (cena zakupu, wysokość pierwszej wpłaty, kwota pożyczki, okres kredytowania i oprocentowanie). Wymaga to zastosowania kilku instrukcji *If*, ponieważ pierwszą wpłatę można wprowadzić jako procent ceny zakupu lub jako kwotę w złotych.
- Zapisuje do arkusza nagłówki kolumn.
- Pod nagłówkami kolumn zapisywany jest pierwszy wiersz zawierający formuły. Wiersz ten różni się od pozostałych, ponieważ zapisane w nim formuły odnoszą się do danych w sekcji parametrów pożyczki. Pozostałe formuły odnoszą się do poprzedniego wiersza. Zwróćmy uwagę, że w formułach zastosowano nazwane zakresy. Są to nazwy zdefiniowane na poziomie arkusza, a zatem użytkownik może zapisać więcej niż jeden harmonogram amortyzacji w tym samym skoroszycie.
- Dla odwołań, w których nie zastosowano nazwy, wykorzystałem notację z numerem wiersza i kolumny. Jest ona znacznie prostsza od próby określenia rzeczywistych adresów komórek.
- Zapisuje do arkusza drugi wiersz formuł, a następnie kopiuje po jednym wierszu dla każdego miesiąca.
- Jeżeli użytkownik wybierze opcję dla podsumowań rocznych, procedura oblicza sumy częściowe za pomocą metody *Subtotal*. Przy okazji jest to przykład pokazujący zastosowanie wbudowanej właściwości Excela, dzięki której programista może uniknąć naprawdę dużej ilości kodowania.
- Ponieważ tworzenie podsumowań częściowych dla kolumny *Saldo* jest niewłaściwe, procedura zastępuje formuły w tej kolumnie formułami obliczającymi saldo na koniec roku.

- Kiedy Excel oblicza sumy częściowe, jednocześnie tworzy konspekt. Jeżeli użytkownik nie chce wyświetlania konspektu, procedura wykorzystuje metodę ClearOutline w celu jej usunięcia.
- Następnie procedura wykonuje formatowanie komórek: wprowadza format liczb oraz autoformatowanie w przypadku, kiedy użytkownik wybierze prezentację wyników w kolorze.
- Wygenerowany harmonogram amortyzacji jest konwertowany do postaci tabeli, której nadawany jest odpowiedni styl formatowania (w zależności od opcji koloru wybranej przez użytkownika).
- Kolejną czynnością jest dostosowanie szerokości kolumn, zablokowanie tytułów poniżej wiersza nagłówka i zabezpieczenie formuł oraz innych ważnych komórek, których nie należy modyfikować.
- Jeżeli opcja zabezpieczania arkusza zostanie włączona (punkt 5), ochrona arkusza jest włączana, ale bez użycia hasła.
- Na koniec procedura SaveDefaults zapisuje bieżące wartości formantów formularza *UserForm* do rejestru Windows. Wartości te staną się nowymi ustawieniami domyślnymi, kiedy użytkownik będzie tworzył harmonogram amortyzacji następnym razem (więcej informacji znajduje się w kolejnym podpunkcie, „Zapisywanie i odtwarzanie ustawień domyślnych”).

Zapisywanie i odtwarzanie ustawień domyślnych

W formularzu FormMain zawsze wyświetlały się ostatnie ustawienia, które program zapamiętał i wykorzystał jako nowe wartości domyślne. Dzięki temu można z łatwością generować wiele harmonogramów amortyzacji różniących się jednym lub kilkoma parametrami, co umożliwia uzyskanie odpowiedzi na pytania typu *co-jedeli?*. Właściwość tę zapewnia zapisanie wartości w rejestrze Windows, a następnie odtwarzanie ich podczas inicjowania formularza *UserForm*. Kiedy aplikacja jest używana po raz pierwszy, w rejestrze nie ma żadnych wartości, zatem są wykorzystywane wartości domyślne zapisane w formantach formularza *UserForm*.

Zamieszczona poniżej procedura GetDefaults przetwarza w pętli wszystkie formanty formularza *UserForm*. Jeżeli przetwarzany formant to pole tekstowe, pole kombi, przycisk opcji, pole wyboru lub pokrętło, wywoływana jest funkcja VBA GetSetting, która odczytuje wartość z rejestru. Trzecim argumentem funkcji GetSetting jest wartość, która ma być użyta, jeżeli w rejestrze nie zostanie odnaleziona określona wartość. W tym przypadku wykorzystywana jest wartość formantu wprowadzona w czasie projektowania. APPNAME jest globalną stałą zawierającą nazwę aplikacji.

```
Sub GetDefaults()
    ' Odczytanie domyślnych ustawień z rejestru
    Dim ctl As Control
    Dim CtrlType As String

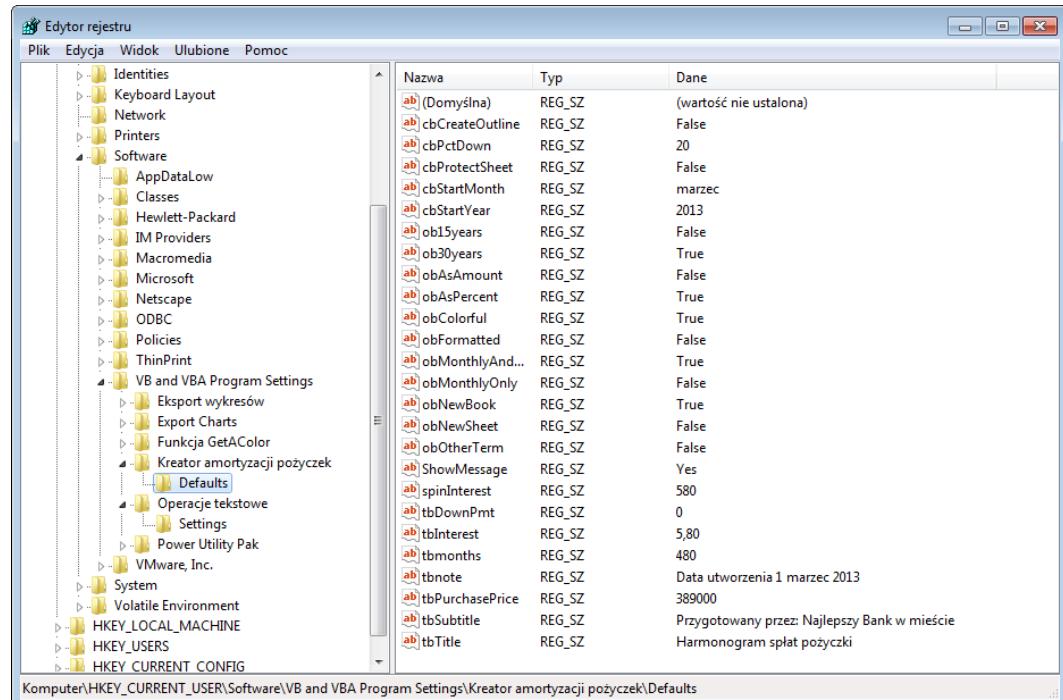
    For Each ctl In Me.Controls
        CtrlType = TypeName(ctl)
        If CtrlType = "TextBox" Or _
            CtrlType = "ComboBox" Or _
```

```

CtrlType = "OptionButton" Or _
CtrlType = "CheckBox" Or _
CtrlType = "SpinButton" Then
    ctl.Value = VBA.GetSetting
        (APPNAME, "Domyślne", ctl.Name, ctl.Value)
    End If
Next ctl
End Sub

```

Na rysunku 23.7 pokazano, jak wyglądają wartości zapisane rejestrze systemu Windows, wyświetlane za pomocą programu *Edytor rejestru* (*regedit.exe*).



Rysunek 23.7. Domyślne wartości wykorzystywane przez kreatora są zapisane w rejestrze Windows

Zaprezentowana poniżej procedura SaveDefaults jest podobna. Wykorzystano w niej funkcję VBA SaveSetting w celu zapisania bieżących wartości do rejestrzu.

```

Sub SaveDefaults()
    ' Zapisanie bieżących ustawień do rejestrzu
    Dim ctl As Control
    Dim CtrlType As String

    For Each ctl In Me.Controls
        CtrlType = TypeName(ctl)
        If CtrlType = "TextBox" Or _
            CtrlType = "ComboBox" Or _
            CtrlType = "OptionButton" Or _
            CtrlType = "CheckBox" Or _
            CtrlType = "SpinButton" Then

```

```
        SaveSetting APPNAME, "Domyślne", ctl.Name, CStr(ctl.Value)
    End If
Next ctl
End Sub
```

W kodzie wykorzystano funkcję `CStr` w celu przekształcenia poszczególnych ustawień na ciągi znaków. Ma to na celu zapobieżenie problemom użytkowników korzystających z ustawień regionalnych dla języków innych niż angielski. Bez konwersji na ciągi znaków wartości `True` i `False` przed zapisaniem do rejestru zostałyby przetłumaczone. Niestety, wartości *nie* zostałyby przetłumaczone w drugą stronę w przypadku pobierania ustawień — a to spowodowałoby błąd.

W instrukcjach `SaveSetting` i `GetSetting` zawsze wykorzystywany jest następujący klucz rejestru:

```
HKEY_CURRENT_USER\Software\VB and VBA Program Settings\
```

Potencjalne usprawnienia Kreatora amortyzacji pożyczek

Bardzo często słyszy się opinię, że nigdy nie można skończyć aplikacji — a jedynie można przestać nad nią pracować. Nawet bez głębokiego zastanowienia przychodzi mi na myśl kilka usprawnień naszego Kreatora amortyzacji pożyczek:

- Dodanie opcji wyświetlania skumulowanych podsumowań dla odsetek i kapitału.
- Wprowadzenie obsługi pożyczek ze zmiennym oprocentowaniem i wykonywanie harmonogramów na podstawie przewidywanych scenariuszy zmian oprocentowania.
- Dodanie opcji formatowania (np. bez miejsc po przecinku, bez znaków waluty itp.).
- Dodanie możliwości definiowania nagłówków i stopek strony przez użytkowników.

Wskazówki dotyczące projektowania aplikacji

Analizując aplikację napisaną przez innego programistę, czasami mam problemy ze zrozumieniem jej logiki. Aby ułatwić zrozumienie mojej pracy innym, umieściłem w kodzie liczne komentarze, natomiast w poprzednich punktach opisałem ogólnie przepływ sterowania w aplikacji. Jeżeli jednak ktoś chce dokładnie zrozumieć działanie tej aplikacji, polecam skorzystanie z debugera i przeanalizowanie kodu krok po kroku.

Na przykładzie Kreatora amortyzacji pożyczek zademonstrowałem kilka ciekawych pojęć i technik, które powinny zainteresować programistów Excela. Są to między innymi:

- Modyfikacja przycisków Wstążki.
- Wykorzystanie kreatora utworzonego na podstawie okien `UserForm` do pobierania informacji od użytkowników.
- Dynamiczne ustawianie właściwości `Enabled` formantów.
- Powiązanie pola tekstowego (`TextBox`) z przyciskiem pokrętła (`SpinButton`).

Projektowanie aplikacji przyjaznych dla użytkownika — lista kontrolna czynności

Projektując aplikację, która ma być przyjazna dla użytkownika, należy pamiętać o wielu sprawach. Poniższa lista może służyć jako wzorzec czynności, które należy wykonać i sprawdzić.

- **Czy okna dialogowe można obsługiwać całkowicie z klawiatury?** Pamiętaj o dodaniu klawiszy skrótów oraz dokładnie sprawdź kolejność kart.
- **Czy przyjęłeś jakieś założenia dotyczące katalogów?** Jeżeli aplikacja odczytuje lub zapisuje pliki, nie można założyć, że istnieje określony katalog oraz że jest to katalog bieżący.
- **Czy zapewniłeś możliwość anulowania okien dialogowych?** Nie wolno zakładać, że użytkownicy zakończą pracę we wszystkich oknach dialogowych poprzez kliknięcie przycisku *OK*.
- **Czy założyłeś, że nie są otwarte inne arkusze?** Jeżeli podczas testowania aplikacja była jedynym otwartym skoroszytem, mogłeś łatwo przeoczyć zdarzenia, które występują w czasie, kiedy są otwarte inne skoroszyty.
- **Czy założyłeś, że skoroszyt jest widoczny?** Z Excela można korzystać również w sytuacji, kiedy nie jest widoczny żaden skoroszyt.
- **Czy zoptymalizowałeś aplikację pod względem szybkości jej działania?** Aplikację można przyspieszyć, na przykład deklarując typy zmiennych oraz definiując zmienne obiektowe.
- **Czy procedury zostały właściwie udokumentowane?** Czy kod nadal będzie zrozumiały, jeżeli spróbujesz przeanalizować go za sześć miesięcy?
- **Czy dołączyłeś odpowiednią dokumentację użytkownika?** Wykonanie dokumentacji często eliminuje pytania użytkowników dotyczące obsługi aplikacji (lub przynajmniej zmniejsza liczbę pytań).
- **Czy dokładnie przeanalizowałeś aplikację?** Istnieje prawdopodobieństwo, że aplikacja nie będzie od razu doskonała. Poświęć trochę czasu na jej poprawianie.

- Wyświetlanie pomocy kontekstowej dla użytkownika.
- Definiowanie nazw komórek za pomocą języka VBA.
- Tworzenie i kopiowanie formuł za pomocą języka VBA.
- Tworzenie tabel za pomocą języka VBA.
- Odczytywanie i zapisywanie danych do i z rejestru Windows.

Tworzenie aplikacji przyjaznych dla użytkownika w Excelu nie jest proste. Trzeba przewidzieć, w jaki sposób użytkownicy będą korzystać z aplikacji. Chociaż starałem się, aby aplikacja przedstawiona w niniejszym rozdziale była w maksymalnym stopniu odporna na nierozsądne działania użytkowników, nie przeprowadziłem dla niej gruntownego testowania w „produkcyji”. W związku z tym nie zdziwiłbym się, gdyby w pewnych warunkach jej działanie zakończyło się błędem.

Część VI

Inne zagadnienia

W tej części:

Rozdział 24. „Problem kompatybilności aplikacji”

Rozdział 25. „Operacje na plikach wykonywane za pomocą kodu VBA”

Rozdział 26. „Operacje na składnikach języka VBA”

Rozdział 27. „Moduły klas”

Rozdział 28. „Praca z kolorami”

Rozdział 29. „Często zadawane pytania na temat programowania w Excelu”

Rozdział 24.

Problem kompatybilności aplikacji

W tym rozdziale:

- Jak zwiększyć prawdopodobieństwo, że Twoja aplikacja w Excelu 2013 będzie działała poprawnie ze starszymi wersjami tego programu
- Deklaracje funkcji API, które działają poprawnie zarówno z 32- i 64-bitową wersją Excela 2013, jak i z poprzednimi wersjami tego programu
- Zagadnienia, o których powinieneś pamiętać podczas tworzenia wielojęzycznych aplikacji Excela

Co to jest kompatybilność?

Wśród użytkowników komputerów termin *kompatybilność* (lub inaczej *zgodność*) jest używany bardzo często i oznacza zdolność działania programu w różnych środowiskach sprzętowych, programowych lub sprzętowo-programowych. Na przykład oprogramowania napisanego dla systemu z rodziny Windows nie można też bezpośrednio uruchamiać w innych systemach operacyjnych, takich jak MacOS czy Linux.

W tym rozdziale omówiono szereg zagadnień związanych z tym, jak aplikacje Excela 2013 działają z wcześniejszymi wersjami Excela dla Windows oraz Excela dla systemu MacOS. Fakt, iż dwie wersje Excela korzystają z tego samego formatu plików, nie zawsze jest wystarczający do zapewnienia pełnej zgodności zawartości plików. Na przykład Excel 97, Excel 2000, Excel 2002, Excel 2003 oraz Excel 2008 dla systemu MacOS teoretycznie korzystają z tego samego formatu plików, ale nie zmienia to w niczym faktu, że występują pomiędzy nimi znaczne problemy z kompatybilnością. Samo otwarcie skoroszytu lub dodatku w określonej wersji Excela wcale nie oznacza, że uda Ci się poprawnie uruchomić zapisane w nich makra języka VBA. Inny przykład: zarówno Excel 2013, jak i Excel 2007 używają tego samego formatu zapisu plików. Jeżeli jednak Twoja aplikacja korzysta z mechanizmów i funkcji, które są dostępne jedynie w Excelu 2010 czy 2013, nie możesz oczekwać, że nagle użytkownicy Excela 2007 będą w magiczny sposób mieli dostęp do tych nowych funkcji.

Pakiet zgodności formatu plików pakietu Microsoft Office

Jeżeli planujesz udostępniać swoje aplikacje napisane w programie Excel 2013 innym użytkownikom, którzy nadal korzystają ze starszych wersji Excela, masz do wyboru dwie możliwości:

- Zawsze zapisuj dokumenty Excela w starszym formacie XLS.
- Upewnij się, że użytkownicy końcowi zainstalowali na swoich komputerach *Pakiet zgodności formatu plików pakietu Microsoft Office* (ang. *Microsoft Office Compatibility Pack*).

Pakiet zgodności formatu plików pakietu Microsoft Office można bezpłatnie pobrać ze strony internetowej firmy Microsoft (www.microsoft.com). Po zainstalowaniu tego pakietu użytkownicy pakietów Office XP oraz Office 2003 mogą otwierać, modyfikować i zapisywać dokumenty, skoroszyty i prezentacje w nowych formatach, używanych przez takie programy jak Word, Excel czy PowerPoint.

Pamiętaj, że *Pakiet zgodności formatu plików pakietu Microsoft Office* nie wyposaża starszych wersji Excela we wszystkie nowe mechanizmy i funkcje Excela 2007 i nowszych wersji, ale po prostu pozwala użytkownikowi starszych wersji na otwieranie i zapisywanie plików w nowym formacie.



Najnowsza wersja pakietu Microsoft Office jest dostępna również w wersji sieciowej oraz dla urządzeń mobilnych, działających pod kontrolą systemu Windows RT, takich jak tablety czy smartfony, co zapewne jeszcze bardziej skomplikuje zagadnienia związane z kompatybilnością plików i aplikacji. Warto zauważyć, że takie mobilne i sieciowe wersje pakietu Office nie obsługują VBA, dodatków oraz innych funkcji, które wykorzystują formanty ActiveX.

Excel jest produktem, który przez cały czas się zmienia i w praktyce nie ma możliwości zapewnienia stu procentowej kompatybilności dokumentów i aplikacji pomiędzy wersjami. Niestety zgodności pomiędzy wersjami nie da się uzyskać automatycznie i aby ją osiągnąć, w większości przypadków trzeba dolożyć sporo wysiłku i dodatkowej pracy.

Rodzaje problemów ze zgodnością

Istnieje kilka kategorii potencjalnych problemów ze zgodnością, czy jak woli, kompatybilnością. Wymienimy je teraz poniżej i omówimy bardziej szczegółowo w dalszej części tego rozdziału.

- **Problemy dotyczące formatu plików** — skoroszyty Excela można zapisać w kilku różnych formatach plików. W starszych wersjach Excela możesz napotkać poważne problemy z otwieraniem plików zapisanych w formacie używanym przez nowsze wersje. Więcej szczegółowych informacji na temat udostępniania dokumentów Excela w wersjach od 2007 do 2013 znajdziesz w ramce „*Pakiet zgodności formatu plików pakietu Microsoft Office*”.
- **Problemy związane z nowymi właściwościami** — dla każdego użytkownika jest chyba oczywiste, że funkcje i mechanizmy, które pojawiły się w określonej wersji Excela, nie są dostępne w jego wcześniejszych wersjach.
- **Problemy firmy Microsoft** — z niezrozumiałych powodów za wiele problemów z kompatybilnością aplikacji firmy Microsoft odpowiedzialna jest... sama firma Microsoft. Na przykład — jak już wspomniano w rozdziale 21. — numery indeksów poleceń menu w różnych wersjach Excela nie są spójne.

- **Problemy zgodności pomiędzy systemami Windows a MacOS** — jeżeli aplikacja ma działać na obu platformach, z pewnością będziesz musiał poświęcić wiele czasu na rozwiązywanie różnorodnych problemów z kompatybilnością.
- **Problemy z „bitami”** — Excel 2010 był pierwszą wersją Excela, która była dostępna zarówno w wersji 32-bitowej, jak i 64-bitowej. Jeżeli kod VBA Twojej aplikacji korzysta z funkcji API, musisz brać pod uwagę potencjalne problemy, które mogą się pojawić w sytuacji, kiedy aplikacja będzie musiała działać na różnych wersjach Excela.
- **Problemy różnic pomiędzy wersjami narodowymi** — jeżeli z aplikacji mają korzystać użytkownicy posługujący się różnymi wersjami językowymi Excela, będziesz musiał często rozwiązywać dodatkowe problemy dotyczące zgodności.

Po przeczytaniu tego rozdziału powinno być oczywiste, że istnieje tylko jeden sposób na zapewnienie kompatybilności: musisz przetestować swoją aplikację na wszystkich dostępnych platformach oraz ze wszystkimi dostępnymi wersjami programu Excel.



Uwaga

Czytelnicy, którzy spodziewali się znaleźć w tym rozdziale pełną listę problemów z kompatybilnością Excela, będą rozczarowani. O ile mi wiadomo, taka pełna lista po prostu nie istnieje, a co więcej, jestem przekonany, że jej utworzenie byłoby bardzo trudne — po prostu takie problemy występują zbyt często i są zbyt złożone.



Wskazówka

Dobrym źródłem informacji, pozwalającym zidentyfikować wiele błędów różnych wersji Excela oraz poznać potencjalne problemy zgodności, jest baza wiedzy firmy Microsoft. Jej adres to <http://search.support.microsoft.com>. Informacje zawarte w tej bazie bardzo często pozwolą Ci na zidentyfikowanie i poprawienie błędów występujących w różnych wersjach Excela.

Unikaj używania nowych funkcji i mechanizmów

Jeżeli Twoja aplikacja musi pracować zarówno z Exceliem 2013, jak i wcześniejszymi wersjami tego programu, musisz unikać stosowania funkcji i mechanizmów, które zostały dodane do Excela po pojawienniu się na rynku najstarszej z wykorzystywanych wersji Excela. Innym rozwiązaniem jest próba selektywnej implementacji odpowiednich funkcji i mechanizmów, czyli inaczej mówiąc, kod programu może określić, jaka wersja Excela jest używana do jego wykonania, i użyć funkcji odpowiednich dla danej wersji.

Określanie numeru wersji Excela

Wersję Excela można określić za pomocą właściwości `Version` obiektu `Application`. Zwrócona wartość jest ciągiem znaków, a zatem czasami trzeba ją przekształcić na liczbę. Idealnie nadaje się do tego funkcja `Val` języka VBA. Na przykład poniższa funkcja zwraca wartość `True`, jeżeli użytkownik wykorzystuje Excela 2007 lub nowszą wersję:

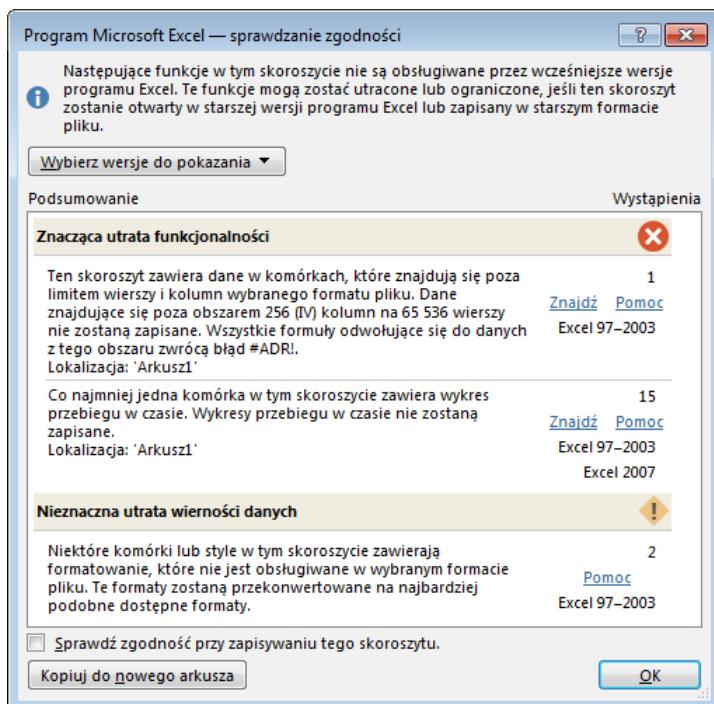
```
Function XL120rLater()
    XL110rLater = Val(Application.Version) >= 12
End Function
```

Excel 2007 to wersja numer 12, Excel 2010 to wersja numer 14, a Excel 2013 to wersja 15. Jak widać, nie istnieje wersja Excela o numerze 13, co niektórzy przypisują powszechnie znanym przesądom.

Programiści aplikacji VBA powinni dbać o to, aby nie używać żadnych obiektów, właściwości ani metod, które nie są dostępne w poprzednich wersjach. Ogólnie rzecz biorąc, najbezpieczniejszym rozwiązaniem jest zaprojektowanie aplikacji według zasady najmniejszego wspólnego mianownika. Na przykład aby zapewnić kompatybilność aplikacji z Exceliem 2003 i wersjami późniejszymi, powinieneś utworzyć aplikację przy użyciu Excela 2003 i później przetestować działanie aplikacji na nowszych wersjach programu.

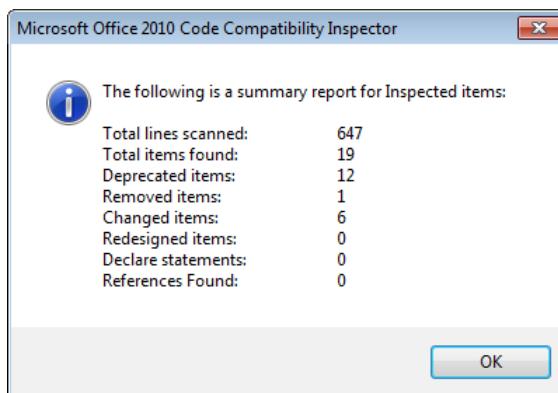
Bardzo użytecznym mechanizmem, wprowadzonym już w Excelu 2007 jest kreator sprawdzania zgodności, którego okno dialogowe zostało przedstawione na rysunku 24.1. Aby go uruchomić, przejdź na kartę *PLIK*, kliknij kategorię *Informacje*, rozwiń listę *Wyszukaj problemy* i następnie z menu podręcznego wybierz polecenie *Sprawdź zgodność*. Kreator sprawdzania zgodności sprawdza, czy dany skoroszyt używa funkcji, które mogłyby spowodować potencjalne problemy we wcześniejszych wersjach Excela.

Rysunek 24.1.
Kreator sprawdzania
zgodności



Niestety, kreator sprawdzania zgodności nawet nie próbuje zaglądać do kodu VBA — który zwykle jest jedną z najpoważniejszych przyczyn niekompatybilności aplikacji. W razie potrzeby jednak z witryny internetowej firmy Microsoft możesz pobrać pakiet *Microsoft Office Code Compatibility Inspector* (poszukaj na stronie <http://www.microsoft.com>). Pakiet instaluje dodatek Excela i dodaje kilka nowych poleceń na karcie *Developer*. Narzędzie pomaga zlokalizować potencjalne problemy z kompatybilnością kodu VBA. W czasie kiedy powstawała niniejsza książka, pakiet *Microsoft Office Code Compatibility Inspector* nie został jeszcze zaktualizowany dla wersji Excela 2013 (ale mimo to można go zainstalować). Do tej pory kilka razy zdarzyło mi się używać tego pakietu, ale szczerze mówiąc, nie okazał się zbyt użyteczny. Przykładowe podsumowanie raportu wygenerowanego przez to narzędzie zostało przedstawione na rysunku 24.2.

Rysunek 24.2.
Podsumowanie raportu
wygenerowanego przez
pakiet Microsoft Office
Code Compatibility
Inspector



Czy aplikacja będzie działać na komputerach Macintosh?

Jednym z najczęstszych problemów, jakie muszą rozwiązywać programiści, jest zgodność z komputerami Macintosh. Excel dla komputerów Macintosh stanowi zaledwie niewielką część rynku Excela i dlatego wielu programistów po prostu go ignoruje. Dobra wiadomość jest taka, że na szczęście pliki zapisane w starym, dobrym formacie XLS są zgodne pomiędzy obydwooma platformami. Zła wiadomość, to fakt, że właściwości obsługiwane na obu platformach nie są jednak identyczne, a zgodność makr VBA jest daleka od ideału. Co gorsza, najnowsza wersja Excela, 2008, dla platformy MacOS po prostu nie obsługuje VBA.

Do określenia platformy, na jakiej działa aplikacja, można zastosować kod VBA. Zaprezentowana poniżej funkcja wykorzystuje właściwość OperatingSystem obiektu Application i zwraca wartość True dla dowolnej wersji systemu Windows (tzn. jeśli zwracany ciąg znaków zawiera tekst *Win*):

```
Function WindowsOS() As Boolean
    If Application.OperatingSystem like "*Win*" Then
        WindowsOS = True
    Else
        WindowsOS = False
    End If
End Function
```

Istnieje wiele różnic pomiędzy wersją Excela dla systemu Windows oraz dla systemu MacOS. Niektóre z nich są subtelne (np. różne czcionki domyślne), a inne dość istotne. Na przykład Excel dla systemu MacOS nie obsługuje formantów ActiveX. Ponadto domyślnie wykorzystuje system daty 1904, a zatem w niektórych skoroszytach daty mogą być wcześniejsze o cztery lata. Z kolei w Excelu dla Windows domyślnie wykorzystywany jest system dat 1900. W systemie MacOS data o wartości 1 oznacza 1 stycznia 1904 roku, natomiast w Excelu dla Windows ta sama wartość daty oznacza 1 stycznia 1900 roku.

Inne ograniczenia dotyczą funkcji interfejsu Windows API: w Excelu dla systemu MacOS takie funkcje nie będą działać. Jeżeli w Twojej aplikacji takie funkcje odgrywają istotną rolę, powinieneś opracować rozwiązanie zastępcze.

Jeżeli w kodzie wykorzystywane są ścieżki i nazwy plików, należy zastosować odpowiedni separator ścieżki (dwukropki dla systemu MacOS, odwrotny ukośnik dla systemu Windows). W takiej sytuacji najlepiej unikać kodowania separatora ścieżki „na twardo” i wykorzystać język VBA w celu jego określenia. Poniższa instrukcja powoduje przypisanie separatora ścieżki do zmiennej o nazwie PathSep:

```
PathSep = Application.PathSeparator
```

Po wykonaniu tej instrukcji można wykorzystać zmienną PathSep zamiast zapisanego „na twardo” dwukropka lub lewego ukośnika.

Zamiast tworzyć jeden plik zgodny z obiema platformami, większość programistów projektuje aplikację dla jednej platformy (zazwyczaj jest nią Excel dla Windows), a następnie modyfikuje ją w taki sposób, aby działała na drugiej. Jeśli więc aplikacja musi działać w dwóch systemach, zwykle utrzymuje się jej dwie oddzielne wersje.

Jest tylko jeden sposób sprawdzenia, czy aplikacja jest zgodna z wersją Excela dla systemu MacOS: należy dokładnie ją przetestować w tym systemie i opracować rozwiązania zastępcze dla tych procedur, które nie działają prawidłowo.



Ron de Bruin, Microsoft Excel MVP z Holandii, jest autorem strony internetowej, omawiającej między innymi wiele przykładów procedur VBA i zagadnień kompatybilności pomiędzy Exceliem 2011 dla komputerów Mac i Exceliem dla platformy Windows. Więcej szczegółowych informacji na ten temat znajdziesz na stronie <http://www.rondebruin.nl/mac.htm>.

Praca z 64-bitową wersją Excela

Excela w wersji 2010 i 2013 możesz zainstalować zarówno w wersji 32-bitowej, jak i w wersji 64-bitowej. Z tej ostatniej możesz oczywiście skorzystać tylko w sytuacji, kiedy używasz 64-bitowej wersji systemu Windows. Ponieważ 64-bitowa wersja Excela korzysta z większej przestrzeni adresowej 64-bitowego systemu Windows, może w efekcie obsługiwać dużo większe skoroszyty, niż obsługuje wersja 32-bitowa.

Większość użytkowników nie będzie jednak potrzebowała wersji 64-bitowej, ponieważ zwykły przeciętny użytkownik nie pracuje z tak dużymi zbiorami danych. Pamiętaj również, że użycie 64-bitowej wersji Excela wcale nie implikuje wzrostu szybkości działania programu, a wręcz można powiedzieć, że niektóre operacje mogą być wykonywane wolniej.

Ogólnie rzecz biorąc, skoroszyty utworzone w 32-bitowej wersji Excela będą poprawnie działać w wersji 64-bitowej. Pewne problemy mogą się zdarzyć jedynie w sytuacji, kiedy skoroszyt będzie zawierał kod VBA wykorzystujący wywołania funkcji API. Deklaracje 32-bitowych funkcji API nie będą się poprawnie komplilować w 64-bitowej wersji Excela.

Na przykład deklaracja przedstawiona poniżej działa poprawnie z 32-bitową wersją Excela, a jej użycie w wersji 64-bitowej spowoduje wystąpienie błędu komplikacji:

```
Declare Function GetWindowsDirectoryA Lib "kernel32" _  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Z kolei deklaracja funkcji API przedstawiona poniżej działa poprawnie zarówno z 32-bitową, jak i 64-bitową wersją Excela 2010 i 2013, ale nie będzie działała z wersjami wcześniejszymi:

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Aby używać wywołań funkcji API zarówno w 32-bitowej, jak i 64-bitowej wersji Excela, powinieneś zadeklarować dwie wersje funkcji, korzystając z dwóch warunkowych dyrektyw kompilatora:

- VBA7 zwraca wartość True w sytuacji, kiedy kod używa wersji 7. VBA (dostarczana z pakietem Office 2010).
- Win64 zwraca wartość True w sytuacji, kiedy kod jest uruchamiany na 64-bitowej wersji Excela.

Poniżej przedstawiamy przykład zastosowania wspomnianych wcześniej dyrektyw kompilatora do zadeklarowania funkcji API, które będą poprawnie działały zarówno w 32-bitowej, jak i 64-bitowej wersji Excela:

```
#If VBA7 And Win64 Then  
    Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" _  
        (ByVal lpBuffer As String, ByVal nSize As Long) As Long  
#Else  
    Declare Function GetWindowsDirectoryA Lib "kernel32" _  
        (ByVal lpBuffer As String, ByVal nSize As Long) As Long  
#End If
```

Pierwsze wyrażenie `Declare` jest używane w sytuacji, kiedy VBA7 oraz Win64 mają wartość True — co jest prawdziwe tylko w przypadku 64-bitowej wersji Excela 2010. We wszystkich pozostałych przypadkach używane jest drugie wyrażenie `Declare`.

Tworzenie aplikacji dla wielu wersji narodowych

Ostatnim problemem zgodności, jaki omówimy w tym rozdziale, są ustawienia międzynarodowe. Excel jest dostępny w wielu różnych wersjach językowych. Polecenie przedstawione poniżej wyświetla kod kraju używanej wersji Excela:

```
MsgBox Application.International(xlCountryCode)
```

Kod 1 odpowiada wersji Excela dla języka angielskiego (USA). Pozostałe kody języków zestawiono w tabeli 24.1.

Tabela 24.1. Kody języków dla Excela

Kod języka	Kraj/region	Język
1	Stany Zjednoczone	angielski
7	Rosja	rosyjski
30	Grecja	grecki
31	Holandia	holenderski
33	Francja	francuski
34	Hiszpania	hiszpański
36	Węgry	węgierski
39	Włochy	włoski
42	Czechy	czeski
45	Dania	duński
46	Szwecja	szwedzki
47	Norwegia	norweski
48	Polska	polski
49	Niemcy	niemiecki
55	Brazylia	portugalski
66	Tajlandia	tajski
81	Japonia	japoński
82	Korea	koreański
84	Wietnam	wietnamski
86	Chiny	chiński (uproszczony)
90	Turcja	turecki
91	Indie	hinduski
92	Pakistan	urdu
351	Portugalia	portugalski
358	Finlandia	fiński
886	Tajwan	chiński (tradycyjny)
966	Arabia Saudyjska	arabski
972	Izrael	hebrajski
982	Iran	perski

Excel obsługuje również pakiety językowe, zatem jedna kopia Excela może działać w kilku wersjach językowych. Informacja o aktualnie używanej wersji językowej jest istotna dla dwóch obszarów: interfejsu użytkownika oraz trybu uruchamiania programów.

Aktualnie używaną wersję językową możesz określić przy użyciu polecenia przedstawionego poniżej:

```
Msgbox Application.LanguageSettings.LanguageID(msoLanguageIDUI)
```

Identyfikator języka dla języka polskiego to 1045.

Jeżeli z aplikacji będą korzystały osoby posługujące się innymi językami, powinieneś zapewnić użycie właściwego języka w oknach dialogowych. Powinieneś także zidentyfikować ustawienia międzynarodowe dla symbolu dziesiętnego oraz separatora tysięcy. W Polsce są to odpowiednio przecinek i spacja, jednak użytkownicy w innych krajach stosują inne znaki. Kolejnym problemem jest format daty i czasu: w niektórych krajach używa się zupełnie nielogicznego formatu *miesiąc/dzień/rok*, w innych *dzień-miesiąc-rok*, a jeszcze w innych *rok-miesiąc-dzień*.

Jeżeli tworzymy aplikację, której będą używać jedynie osoby z naszej firmy, nie musimy martwić się problemami zgodności z wersjami międzynarodowymi. Jeżeli jednak firma posiada biura na całym świecie lub jeśli planujemy rozpowszechniać aplikację poza krajem, powinniśmy zwrócić uwagę na zagadnienia zgodności, aby zapewnić poprawne działanie aplikacji w innych wersjach językowych Excela. Zagadnienia te opiszę w kolejnym podrozdziale.

Aplikacje obsługujące wiele języków

Oczywistym problemem związanym z aplikacjami międzynarodowymi jest język aplikacji. Jeżeli na przykład używamy okien dialogowych, powinniśmy zadbać o to, aby tekst wyświetlał się w języku użytkownika. Na szczęście nie jest to zbyt skomplikowane (przy założeniu, że potrafimy przetłumaczyć tekst lub znamy kogoś, kto potrafi).



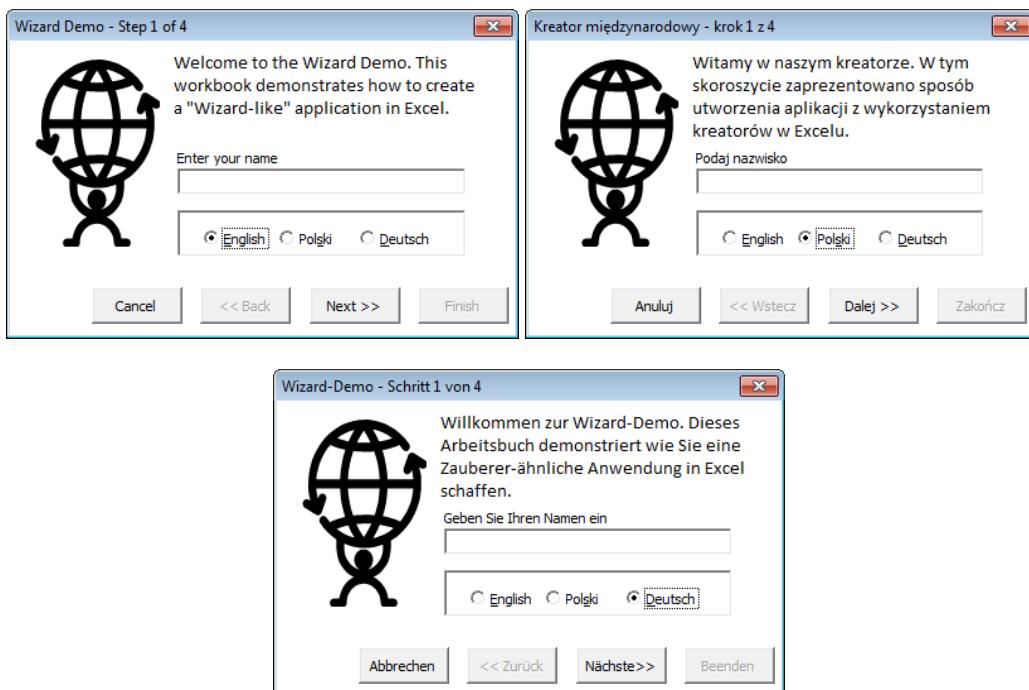
Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>) został zamieszczony skoroszyt z kreatorem skonfigurowanym dla języków polskiego, angielskiego lub niemieckiego. Przykład znajduje się w skoroszycie o nazwie *Kreator międzynarodowy.xlsx*.

W oknie dialogowym pierwszego kroku kreatora umieszczone są trzy przyciski opcji, które użytkownik może wykorzystać do wyboru języka. Tekst dla trzech wersji językowych został zapisany w arkuszu.

Procedura UserForm_Initialize zawiera kod, który sprawdza właściwość International i na tej podstawie próbuje odgadnąć język, jakim posługuje się użytkownik:

```
Select Case Application.International(xlCountryCode)
Case 48 'Polski
    UserLanguage = 2
Case 49 'Niemiecki
    UserLanguage = 3
Case Else 'Domyślny angielski
    UserLanguage = 1 'default
End Select
```

Na rysunku 24.3 przedstawiono wygląd trzech wersji językowych kreatora (angielskiej, polskiej i niemieckiej).



Rysunek 24.3. Wygląd trzech wersji językowych kreatora (angielskiej, polskiej i niemieckiej)

Obsługa języka w kodzie VBA

Ogólnie rzecz biorąc, nie musimy przejmować się językiem, w którym piszemy kod VBA. W Excelu wykorzystywane są dwie biblioteki obiektów: biblioteka obiektów Excela oraz biblioteka obiektów VBA. Podczas instalacji Excela domyślnie rejestrowana jest angielskojęzyczna wersja obu bibliotek, niezależnie od wersji językowej Excela.

Wykorzystanie właściwości lokalnych

Jeżeli kod wyświetla informacje z arkusza (np. formułę lub adres zakresu), najlepiej wyświetlić je w lokalnym języku. Na przykład poniższa instrukcja wyświetla formułę zapisaną w komórce A1:

```
MsgBox Range("A1").Formula
```

W przypadku aplikacji międzynarodowych lepszym rozwiązaniem jest wykorzystanie właściwości `FormulaLocal` zamiast właściwości `Formula`:

```
MsgBox Range("A1").FormulaLocal
```

Dla kilku innych właściwości istnieją wersje lokalne. Wyszczególniono je w tabeli 24.2 (szczegółowe informacje na ten temat znajdują się w systemie pomocy).

Tabela 24.2. Właściwości, które posiadają wersje lokalne

Vłaściwość	Wersja lokalna	Zwracana zawartość
Address	AddressLocal	Adres
Category	CategoryLocal	Kategoria funkcji (tylko makra XLM)
Formula	FormulaLocal	Formuła
FormulaR1C1	FormulaR1C1Local	Formuła w notacji <i>WIKI</i>
Name	NameLocal	Nazwa
NumberFormat	NumberFormatLocal	Format liczby
RefersTo	RefersToLocal	Odwołanie
RefersToR1C1	RefersToR1C1Local	Odwołanie z wykorzystaniem notacji <i>WIKI</i>

Identyfikacja ustawień systemu

Nie możesz z góry zakładać, że system użytkownika będzie skonfigurowany tak samo jak system, w którym tworzysz aplikację. W przypadku aplikacji międzynarodowych powinieneś zwrócić uwagę na następujące ustawienia:

- **symbol dziesiętny** — znak wykorzystywany w celu oddzielenia części dziesiętnej od części całkowitej liczby;
- **separatot tysięcy** — znak wykorzystywany do oddzielenia trzycyfrowych grup w liczbach;
- **separatot list** — znak wykorzystywany do oddzielenia elementów na liście.

Bieżące ustawienia separatorów możesz uzyskać za pomocą właściwości `International` obiektu `Application`. Na przykład poniższa instrukcja wyświetla symbol dziesiętny, który nie zawsze przecież musi być przecinkiem:

```
MsgBox Application.International(xlDecimalSeparator)
```

Za pomocą właściwości `International` można uzyskać 45 ustawień międzynarodowych. Wyszczególniono je w tabeli 24.3.

Tabela 24.3. Stałe dla właściwości `International`

Stała	Znaczenie
<code>xlCountryCode</code>	Kod kraju dla wersji Excela
<code>xlCountrySetting</code>	Bieżące ustawienia kraju w <i>Panelu sterowania</i> systemu Windows
<code>xlDecimalSeparator</code>	Symbol dziesiętny
<code>xlThousandsSeparator</code>	Separatot tysięcy
<code>xlListSeparator</code>	Separatot list
<code>xlUpperCaseRowLetter</code>	Wielka litera dla wiersza (dla odwołań w stylu <i>WIKI</i>)

Tabela 24.3. Stałe dla właściwości *International* — ciąg dalszy

Stała	Znaczenie
xlUpperCaseColumnLetter	Wielka litera dla kolumny
xlLowerCaseRowLetter	Mała litera dla wiersza
xlLowerCaseColumnLetter	Mała litera dla kolumny
xlLeftBracket	Znak używany zamiast lewego nawiasu kwadratowego ([) w odwołaniach w stylu <i>WIKI</i>
xlRightBracket	Znak używany zamiast prawego nawiasu kwadratowego (]) w odwołaniach w stylu <i>WIKI</i>
xlLeftBrace	Znak używany zamiast lewego nawiasu klamrowego ({) w literałach tablicowych
xlRightBrace	Znak używany zamiast prawego nawiasu klamrowego (}) w literałach tablicowych
xlColumnSeparator	Znak używany w celu oddzielenia kolumn w literałach tablicowych
xlRowSeparator	Znak używany w celu oddzielenia wierszy w literałach tablicowych
xlAlternateArraySeparator	Alternatywny separator elementów tablic do wykorzystania w przypadku, gdy bieżący separator tablic jest taki sam jak symbol dziesiętny
xlDateSeparator	Separator daty (-)
xlTimeSeparator	Separator godziny (:)
xlYearCode	Symbol roku w formatach liczbowych (y)
xlMonthCode	Symbol miesiąca (m)
xlDayCode	Symbol dnia (d)
xlHourCode	Symbol godziny (h)
xlMinuteCode	Symbol minuty (m)
xlSecondCode	Symbol sekundy (s)
xlCurrencyCode	Symbol waluty
xlGeneralFormatName	Nazwa standardowego formatu liczb
xlCurrencyDigits	Liczba cyfr dziesiętnych wykorzystywanych w formatach walutowych
xlCurrencyNegative	Wartość reprezentująca format waluty dla ujemnych kwot
xlNoncurrencyDigits	Liczba cyfr dziesiętnych wykorzystywanych w formatach innych niż walutowe
xlMonthNameChars	Zawsze zwraca trzy znaki w celu zachowania zgodności wstecz. W systemie Windows skróty nazw miesięcy mogą mieć dowolną długość
xlWeekdayNameChars	Zawsze zwraca trzy znaki w celu zachowania zgodności wstecz. W systemie Windows skróty nazw dni tygodnia mogą mieć dowolną długość
xlDateOrder	Liczba całkowita reprezentująca kolejność elementów daty
xl24HourClock	True, jeżeli w systemie wykorzystuje się 24-godzinny format czasu; False, jeżeli wykorzystuje się 12-godzinny format czasu

Tabela 24.3. Stałe dla właściwości International — ciąg dalszy

Stała	Znaczenie
xlNonEnglishFunctions	True, jeżeli w systemie funkcje nie są wyświetlane w języku angielskim
xlMetric	True, jeżeli jest wykorzystywany system metryczny; False, jeżeli w systemie wykorzystywany jest angielski system miar
xlCurrencySpaceBefore	True, jeżeli przed symbolem waluty dodawana jest spacja
xlCurrencyBefore	True, jeżeli wartości kwot poprzedza symbol waluty; False, jeśli symbol waluty jest umieszczany za kwotą
xlCurrencyMinusSign	True, jeżeli w systemie wykorzystuje się znak minus dla liczb ujemnych; False, jeżeli są wykorzystywane nawiasy okrągłe
xlCurrencyTrailingZeros	True, jeżeli dla zerowych wartości walutowych są wyświetlane końcowe zera
xlCurrencyLeadingZeros	True, jeżeli dla zerowych wartości walutowych są wyświetlane wiodące zera
xlMonthLeadingZero	True, jeżeli w liczbach oznaczających numery miesięcy jest wyświetlane wiodące zero
xlDayLeadingZero	True, jeżeli w liczbach oznaczających numery dni jest wyświetlane wiodące zero
xl4DigitYears	True, jeżeli lata w systemie zapisuje się w postaci czterech cyfr; False, jeżeli zapisuje się je w postaci dwóch cyfr
xlMDY	True, jeżeli formatem daty jest miesiąc/dzień/rok dla daty długiej; False w przypadku formatu dzień/miesiąc/rok
xlTimeLeadingZero	True, jeżeli w ciągach znaków oznaczających godziny są wyświetlane wiodące zera

Ustawienia daty i godziny

Jeżeli w aplikacji, która ma być używana w innych krajach, zapisywane są sformatowane daty, należy zapewnić ich wyświetlanie w formacie znanym użytkownikowi. Najlepszym sposobem jest obliczenie daty za pomocą funkcji VBA DateSerial. Excel sam zastosuje wówczas odpowiednie formatowanie, używając krótkiego formatu daty użytkownika.

W zaprezentowanej poniżej procedurze wykorzystano funkcję DateSerial w celu przypisania daty do zmiennej StartDate. Tak ustawniona data jest następnie zapisywana w komórce A1 z zastosowaniem lokalnego formatu daty krótkiej.

```
Sub WriteDate()
    Dim StartDate As Date
    StartDate = DateSerial(2013, 4, 15)
    Range("A1") = StartDate
End Sub
```

Aby zastosować dodatkowe formatowanie dla daty, można napisać kod, który wykonuje te działania po wprowadzeniu danych w komórce. W Excelu jest dostępnych kilka formatów daty i czasu, a także kilka formatów liczb. Opisano je w systemie pomocy (szukaj haseł takich jak *formaty daty i czasu* czy *formaty liczbowe*).

Rozdział 25.

Operacje na plikach wykonywane za pomocą kodu VBA

W tym rozdziale:

- Przegląd podstawowych mechanizmów przetwarzania plików tekstowych przy użyciu VBA
- Najczęściej wykonywane operacje na plikach
- Różne metody otwierania plików tekstowych
- Wyświetlanie rozszerzonych informacji o plikach (na przykład szczegóły plików multimedialnych)
- Przykłady odczytywania i zapisywania plików tekstowych z poziomu programu VBA
- Przykład programu dokonującego eksportu zakresu komórek do plików w formacie HTML oraz XML
- Pakowanie i rozpakowywanie plików
- Zastosowanie obiektów ADO do importowania danych

Najczęściej wykonywane operacje na plikach

Wiele aplikacji programu Excel wykonuje różne operacje na plikach zewnętrznych. Czasami trzeba wyświetlić listę plików w katalogu, usunąć pliki, zmienić im nazwy itd. Excel pozwala oczywiście na importowanie i eksportowanie różnych typów plików tekstowych, często jednak wbudowane właściwości obsługi plików są niewystarczające. Dobrym przykładem będzie tutaj sytuacja, gdy trzeba dokonać eksportu zakresu komórek do prostego dokumentu HTML.

W tym rozdziale dowiesz się, jak używać języka Visual Basic for Applications (VBA) do realizacji zarówno tych najczęściej wykonywanych, jak i tych mniej popularnych operacji na plikach oraz jak pracować bezpośrednio na plikach tekstowych.

W Excelu można wykonywać operacje na plikach na dwa sposoby:

- **Za pomocą tradycyjnych instrukcji i funkcji języka VBA** (ta metoda działa we wszystkich wersjach Excela).
- **Za pomocą obiektu FileSystemObject wykorzystującego bibliotekę Microsoft Scripting Library** (ta metoda działa w Excelu 2000 i późniejszych wersjach).



Poprzednie wersje programu Excel obsługiwały również obiekt FileSearch, ale od wersji 2007 mechanizm ten został usunięty. W nowej wersji Excela próba uruchomienia makra wykorzystującego obiekt FileSearch zakończy się po prostu wyświetleniem komunikatu o błędzie.

Kolejne podrozdziały zawierają opis wymienionych metod wraz z odpowiednimi przykładami.

Zastosowanie poleceń języka VBA do wykonywania operacji na plikach

Zestawienie poleceń VBA, które można wykorzystać do wykonywania operacji na plikach, zostało zamieszczone w tabeli 25.1. Większość poleceń nie wymaga specjalnego komentatorza, a wszystkie są opisane w systemie pomocy Excela.

Tabela 25.1. Polecenia operacji na plikach w języku VBA

Nazwa polecenia	Opis działania
ChDir	Zmienia bieżący katalog
ChDrive	Zmienia bieżący napęd
Dir	Zwraca nazwę pliku lub katalog pasujący do określonego wzorca lub atrybutu pliku
FileCopy	Kopiuje plik
FileDateTime	Zwraca datę i godzinę ostatniej modyfikacji pliku
FileLen	Zwraca rozmiar pliku w bajtach
GetAttr	Zwraca wartość reprezentującą atrybut pliku
Kill	Usuwa plik
MkDir	Tworzy nowy katalog
Name	Zmienia nazwę pliku lub katalogu
RmDir	Usuwa pusty katalog
SetAttr	Zmienia atrybut pliku

W dalszej części tego rozdziału znajdziesz szereg przykładów ilustrujących zastosowanie niektórych poleceń.

Funkcja VBA sprawdzająca, czy istnieje dany plik

Poniższa funkcja zwraca wartość True, jeżeli określony plik istnieje, lub wartość False, jeżeli plik nie zostanie odnaleziony. Jeżeli funkcja Dir zwraca pusty ciąg znaków, oznacza to, że nie można odnaleźć żadanego pliku i funkcja FileExists zwraca wartość False.

```
Function FileExists(fname) As Boolean
    FileExists = Dir(fname) <> ""
End Function
```

Argumentem funkcji FileExists jest pełna ścieżka dostępu do pliku wraz z jego nazwą. Funkcję można wykorzystać w arkuszu lub wywołać z poziomu procedury VBA. A oto przykład wywołania takiej funkcji:

```
MyFile = "c:\Budżet\2013-propozycja budżetu.docx"
Msgbox FileExists(MyFile)
```

Funkcja VBA sprawdzająca, czy istnieje dany katalog

Poniższa funkcja zwraca wartość True, jeżeli określony katalog istnieje, lub wartość False, jeżeli katalog nie zostanie odnaleziony:

```
Function PathExists(pname) As Boolean
    ' Zwraca wartość True, jeżeli katalog istnieje
    On Error Resume Next
    PathExists = (GetAttr(pname) And vbDirectory) = vbDirectory
End Function
```

Argument pname ma postać łańcucha tekstu, który zawiera ścieżkę katalogu (bez nazwy pliku). Znak ukośnika zamkający ścieżkę jest opcjonalny. Poniżej przedstawiamy przykład wywołania takiej funkcji.

```
MyFolder = "c:\użytkownicy\jan\pulpit\pobieranie\
MsgBox PathExists(MyFolder)
```



Skoroszyt zawierający funkcje FileExists oraz PathExists (*Funkcje plikowe.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Procedura VBA wyświetlająca listę plików w katalogu

Poniższa procedura wyświetla w aktywnym arkuszu listę plików z określonego katalogu wraz z rozmiarem i datą modyfikacji pliku:

```
Sub ListFiles()
    Dim Directory As String
    Dim r As Long
    Dim f As String
    Dim FileSize As Double
    Directory = "F:\Excel\Budżet"
    r = 1
    ' Wstaw nagłówki
    Cells(r, 1) = "Nazwa pliku"
    Cells(r, 2) = "Rozmiar"
```

```

Cells(r, 3) = "Data/godzina"
Range("A1:C1").Font.Bold = True
' Pobierz pierwszy plik
f = Dir(Directory, vbReadOnly + vbHidden + vbSystem)
Do While f <> ""
    r = r + 1
    Cells(r, 1) = f
    ' Poprawka na pliki o wielkości ponad 2GB
    FileSize = FileLen(Directory & f)
    If FileSize < 0 Then FileSize = FileSize + 4294967296#
    Cells(r, 2) = FileSize
    Cells(r, 3) = FileDateTime(Directory & f)
    ' Pobierz następny plik
    f = Dir()
Loop
End Sub

```

Przykładowy wynik działania procedury `ListFiles` pokazano na rysunku 25.1.

Rysunek 25.1.
Wynik działania procedury `ListFiles`

A	B	C	D
1 Folder: C:\Helion\R25\	Rozmiar	Data/czas	
2 Eksport-import-CSV.xlsm	30672	2010-10-18 08:55	
3 Excel Log.xlsxm	15863	2010-10-20 08:56	
4 Export do HTML.xlsxm	20232	2010-10-18 08:55	
5 Export do XML.xlsxm	19696	2010-10-18 08:55	
6 Funkcje plikowe.xlsxm	13583	2010-10-18 08:55	
7 Informacja o plikach.xlsxm	22400	2010-10-18 08:55	
8 Lista plików - rekurencja.xlsxm	22067	2010-10-18 08:55	
9 Lista plików.xlsxm	22881	2010-10-18 08:55	
10 Pakowanie plików ZIP.xlsxm	17716	2010-10-18 08:55	
11 Pokaż informację o napędach.xlsxm	20120	2010-10-18 08:55	
12 Rozpakowywanie plików ZIP.xlsxm	15900	2010-10-20 08:56	
13			
14			



Funkcja `FileLen` języka VBA wykorzystuje dane typu `Long`, stąd w przypadku plików większych niż 2 GB będzie zwracała niepoprawny rozmiar pliku (liczbę ujemną). Kod procedury sprawdza, czy funkcja `FileLen` zwróciła wartość ujemną i jeżeli tak, dokonuje odpowiednich poprawek.

Zwróć uwagę, że procedura dwukrotnie wykorzystuje funkcję `Dir`. Za pierwszym razem (wywołanie z argumentem) funkcja pobiera pierwszą znalezioną nazwę pliku. Kolejne wywołania w pętli (bez argumentu) powodują pobranie nazw kolejnych plików. Jeżeli nie ma więcej plików, funkcja zwraca pusty ciąg znaków.



Skoroszyt z bardziej zaawansowaną wersją tej procedury, umożliwiającą wybranie katalogu za pomocą okna dialogowego (*Lista plików.xlsxm*), znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Pierwszym argumentem funkcji `Dir` może być nazwa plików podana w postaci wzorca (przy użyciu symboli wieloznacznych). Na przykład: aby uzyskać listę plików programu Excel, możesz użyć polecenia przedstawionego poniżej:

```
f = Dir(Directory & "*.xl???", vbReadOnly + vbHidden + vbSystem)
```

Wykonanie tego polecenia spowoduje pobranie z podanego katalogu nazwy pierwszego pliku zgodnego z wzorcem `*.xl??`. Takie użycie symboli wieloznacznych powoduje, że zwarcane są nazwy plików posiadających czteroznakowe rozszerzenie zaczynające się od liter XL. Na przykład, mogą to być pliki o rozszerzeniach `.xlsx`, `.xltm` czy `.xlam`. Drugi argument funkcji `Dir` umożliwia wprowadzenie atrybutów plików (definiowanych w postaci wbudowanych stałych). W tym przykładzie funkcja `Dir` pobiera nazwy plików, które mają ustawione następujące atrybuty: bez atrybutów, tylko do odczytu, plik ukryty oraz plik systemowy.

Aby procedura wyświetlała również pliki Excela zapisane w starszych formatach (takich jak na przykład `.xls` czy `.xla`), powinieneś użyć następującego wzorca:

`*.xl*`

W tabeli 25.2 zamieszczono zestawienie stałych, które mogą być argumentami funkcji `Dir`.

Tabela 25.2. Zestawienie stałych, które mogą być argumentami funkcji Dir

Nazwa stałej	Wartość	Opis
<code>vbNormal</code>	0	Plik bez atrybutów, jest to domyślne ustawienie atrybutów dla tej funkcji
<code>vbReadOnly</code>	1	Pliki tylko do odczytu
<code>vbHidden</code>	2	Pliki ukryte
<code>vbSystem</code>	4	Pliki systemowe
<code>vbVolume</code>	8	Etykieta woluminu. Jeżeli w wywołaniu funkcji został użyty jakikolwiek inny atrybut, ten atrybut będzie ignorowany
<code>vbDirectory</code>	16	Katalogi. Ten atrybut... po prostu nie działa. Wywołanie funkcji <code>Dir</code> z atrybutem <code>vbDirectory</code> nie zwraca nazw podkatalogów



Jeżeli używasz funkcji `Dir` do przechodzenia w pętli przez kolejne pliki i wywoływanie innych procedur przetwarzających pliki, upewnij się, że nie zawierają one polecenia `Dir` — za każdym razem może być aktywna tylko jedna instancja polecenia `Dir`.

Rekurencyjna procedura VBA wyświetlająca listę plików w katalogu

Procedura przedstawiona w tym podrozdziale tworzy listę plików znajdujących się w danym katalogu oraz wszystkich jego podkatalogach. Sama procedura jest nieco nietypowa, ponieważ zawiera wywołania do samej siebie — takie rozwiązanie nazywamy *rekurencją*.

```
Public Sub RecursiveDir(ByVal CurrDir As String, Optional ByVal Level As Long)
    Dim Dirs() As String
    Dim NumDirs As Long
    Dim FileName As String
    Dim PathAndName As String
    Dim i As Long
    Dim FileSize As Double
```

Upewnij się, że ścieżka kończy się znakiem lewego ukośnika
`If Right(CurrDir, 1) <> "\" Then CurrDir = CurrDir & "\ "`

```

' Wstaw nagłówki kolumn do aktywnego arkusza
Cells(1, 1) = "Ścieżka"
Cells(1, 2) = "Nazwa pliku"
Cells(1, 3) = "Rozmiar"
Cells(1, 4) = "Data/godzina"
Range("A1:D1").Font.Bold = True

' Pobierz pliki
FileName = Dir(CurrDir & "\*.*", vbDirectory)
Do While Len(FileName) <> 0
    If Left(FileName, 1) <> "." Then ' Bieżący katalog
        PathAndName = CurrDir & FileName
        If (GetAttr(PathAndName) And vbDirectory) = vbDirectory Then
            'zapamiętaj odnalezione katalogi
            ReDim Preserve Dirs(0 To NumDirs) As String
            Dirs(NumDirs) = PathAndName
            NumDirs = NumDirs + 1
        Else
            'Zapisz ścieżkę i nazwę pliku na arkuszu
            Cells(WorksheetFunction.CountA(Range("A:A")) + 1, 1) = _
                CurrDir
            Cells(WorksheetFunction.CountA(Range("B:B")) + 1, 2) = _
                FileName
            'Poprawka na pliki o rozmiarze powyżej 2GB
            Filesize = FileLen(PathAndName)
            If Filesize < 0 Then Filesize = Filesize + 4294967296#
            Cells(WorksheetFunction.CountA(Range("C:C")) + 1, 3) = FileSize
            Cells(WorksheetFunction.CountA(Range("D:D")) + 1, 4) = _
                FileDateTime(PathAndName)
        End If
    End If
    FileName = Dir()
Loop
' Rekurencyjne przetwarzanie odnalezionych katalogów
For i = 0 To NumDirs - 1
    RecursiveDir Dirs(i), Level + 2
Next i
End Sub

```

Procedura pobiera tylko jeden argument, CurrDir, reprezentujący przetwarzany katalog. Informacja o poszczególnych odnalezionych plikach jest wyświetlana na aktywnym arkuszu. Nazwy podkatalogów odnalezione podczas rekurencyjnego przetwarzania plików są zapamiętywane w tablicy o nazwie Dirs. Kiedy w bieżącym katalogu nie ma już więcej plików do przetwarzania, procedura wywołuje samą siebie, pobierając jako argument wywołania nazwę kolejnego podkatalogu z tablicy Dirs. Procedura kończy działanie po zakończeniu przetwarzania wszystkich podkatalogów zapisanych w tablicy Dirs.

Ponieważ procedura RecursiveDir wymaga podania odpowiedniego argumentu, musi być wywoływana z poziomu innej procedury, na przykład za pomocą następującego polecenia:

```
Call RecursiveDir("c:\nazwa_katalogu\")
```

Skoroszyt z tym przykładem (*Lista plików — rekurencja.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zastosowanie obiektu FileSystemObject

Obiekt *FileSystemObject* należy do biblioteki *Windows Scripting Host* i zapewnia dostęp do systemu plików komputera. Obiekt ten często jest wykorzystywany na stronach WWW zawierających skrypty (np. *VBScript* lub *JavaScript*). Można z niego korzystać w Excelu 2000 i późniejszych wersjach.



Mechanizm *Windows Scripting Host* jest czasami wykorzystywany do rozpowszechniania wirusów komputerowych, dlatego funkcja ta w wielu systemach jest wyłączona. Projektując aplikacje, które będą wykorzystywane na wielu komputerach, należy o tym pamiętać i zachować szczególną ostrożność.

Nazwa *FileSystemObject* może być nieco myląca, ponieważ obiekt ten w rzeczywistości składa się z szeregu innych obiektów, z których każdy posiada swoje osobne, ścisłe określone przeznaczenie:

- **Drive** — reprezentuje napęd dyskowy lub całą kolekcję napędów dyskowych.
- **File** — reprezentuje plik lub kolekcję plików.
- **Folder** — reprezentuje folder lub kolekcję folderów.
- **TextStream** — reprezentuje strumień tekstu odczytywany, zapisywany lub dołączany do pliku tekstowego.

Aby skorzystać z obiektu *FileSystemObject*, powinieneś najpierw utworzyć instancję tego obiektu. Możesz tego dokonać na dwa sposoby: za pomocą tzw. metody wczesnego wiązania (ang. *early binding*) lub metody późnego wiązania (ang. *late binding*).

Metoda późnego wiązania wykorzystuje sekwencję dwóch poleceń, na przykład:

```
Dim FileSys As Object  
Set FileSys = CreateObject("Scripting.FileSystemObject")
```

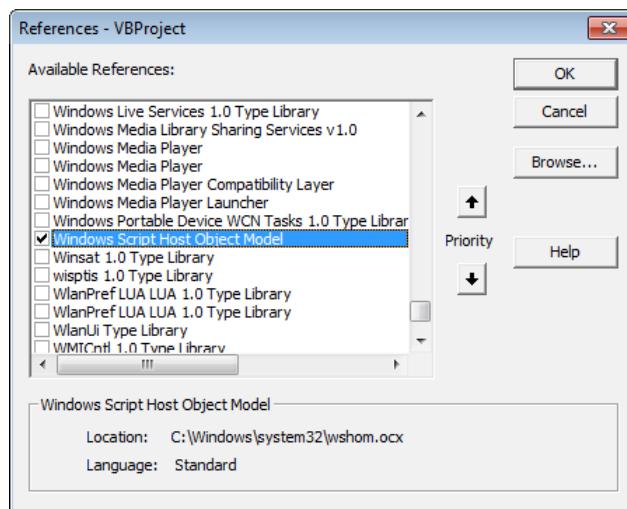
Zwróć uwagę, że zmienna obiektowa *FileSys* została zadeklarowana jako ogólny typ *Object*, a nie jako konkretny typ obiektowy — rodzaj obiektu zostanie ustalony podczas działania programu.

Metoda wczesnego wiązania wymaga utworzenia odwołania do modelu obiektowego *Windows Scripting Host*. Aby to zrobić, powinieneś w edytorze VBE wybrać z menu głównego polecenie *Tools/References* i następnie w oknie dialogowym *References* zaznaczyć odpowiednią opcję (patrz rysunek 25.2). Po utworzeniu odwołania możesz utworzyć obiekt za pomocą następującej sekwencji poleceń:

```
Dim FileSys As Object  
Set FileSys = CreateObject("Scripting.FileSystemObject")
```

Zastosowanie metody wczesnego wiązania pozwala na skorzystanie z mechanizmu *Auto List Members* edytora VBE, który znakomicie ułatwia wpisywanie oraz identyfikację odpowiednich właściwości i metod obiektów. Co więcej, dzięki temu możesz również skorzystać z przeglądarki obiektów (ang. *Object Browser*) i sprawdzić informacje na temat danego obiektu. Aby to zrobić, wystarczy po wejściu do VBE nacisnąć klawisz *F2*.

Rysunek 25.2.
Tworzenie odwołania
do modelu obiektowego
Windows Script Host



W kolejnych przykładach poniżej przedstawimy zastosowanie obiektu `FileSystemObject` do wielu różnych zadań.

Zastosowanie obiektu `FileSystemObject` do sprawdzenia, czy dany plik istnieje

Poniższa funkcja pobiera jeden argument (ścieżkę wraz z nazwą pliku) i jeżeli plik istnieje, zwraca wartość True:

```
Function FileExists3(fname) As Boolean
    Dim FileSys As Object 'FileSystemObject
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    FileExists3 = FileSys.FileExists(fname)
End Function
```

Funkcja tworzy nowy obiekt `FileSystemObject` o nazwie `FileSys`, a następnie sprawdza właściwość `FileExists` tego obiektu.

Zastosowanie obiektu `FileSystemObject` do sprawdzenia, czy istnieje dany katalog

Poniższa funkcja pobiera jeden argument (katalog) i zwraca wartość True, jeżeli ten katalog istnieje:

```
Function PathExists2(pathname) As Boolean
    Dim FileSys As Object 'FileSystemObject
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    PathExists2 = FileSys.FolderExists(path)
End Function
```

Wykorzystanie obiektu FileSystemObject do wyświetlenia informacji o wszystkich dostępnych napędach dyskowych

Procedura przedstawiona poniżej używa obiektu FileSystemObject do pobrania i wyświetlenia różnych informacji na temat dostępnych napędów dyskowych. Procedura przetwarza w pętli kolekcję Drives i zapisuje wartości różnych właściwości do arkusza.

Na rysunku 25.3 przedstawiono wyniki działania procedury w systemie posiadającym pięć napędów dyskowych: zawierającym napęd dyskietek, dwa twarde dyski, napęd CD-ROM oraz dysk sieciowy. Wyświetlane dane obejmują literę napędu, informację o gotowości urządzenia, typ urządzenia, nazwę wolumenu, całkowity rozmiar oraz ilość dostępnego miejsca.

	A	B	C	D	E	F	G
1	Napęd	Gotowy	Typ	Nazwa wolumenu	Rozmiar	Dostępne	
2	A	FAŁSZ	Dysk wymienny				
3	C	PRAWDA	Dysk twardy	SYSTEM	429 475 717 120	300 317 327 360	
4	D	PRAWDA	Dysk twardy	Dane	524 258 337 600	201 623 633 920	
5	F	PRAWDA	Dysk twardy	BACKUP	1 048 516 675 200	476 232 778 910	
6	Z	PRAWDA	Napęd CD-ROM	Helion-Backup	730 028 032	0	
7							
8							

Rysunek 25.3. Wynik działania procedury ShowDriveInfo



Skoroszyt z tym przykładem (*Pokaż informację o napędach.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>).

```

Sub ShowDriveInfo()
    Dim FileSys As FileSystemObject
    Dim Drv As Drive
    Dim Row As Long
    Set FileSys = CreateObject("Scripting.FileSystemObject")
    Cells.ClearContents
    Row = 1
    ' Nagłówki kolumn
    Range("A1:F1") = Array("Napęd", "Gotowy", "Typ", "Nazwa wolumenu", _
                           "Rozmiar", "Dostępne")
    On Error Resume Next
    ' Pętla przetwarzająca kolejne napędy
    For Each Drv In FileSys.Drives
        Row = Row + 1
        Cells(Row, 1) = Drv.DriveLetter
        Cells(Row, 2) = Drv.IsReady
        Select Case Drv.DriveType
            Case 0: Cells(Row, 3) = "Nieznany"
            Case 1: Cells(Row, 3) = "Dysk wymienny"
            Case 2: Cells(Row, 3) = "Dysk twardy"
            Case 3: Cells(Row, 3) = "Dysk sieciowy"
            Case 4: Cells(Row, 3) = "Napęd CD-ROM"
            Case 5: Cells(Row, 3) = "RAM Disk"
        End Select
        Cells(Row, 4) = Drv.VolumeName
    Next
End Sub

```

```

Cells(Row, 5) = Drv.TotalSize
Cells(Row, 6) = Drv.AvailableSpace
Next Drv
'Utwórz tabelę
ActiveSheet.ListObjects.Add xlSrcRange, _
    Range("A1").CurrentRegion, , xlYes
End Sub

```



W rozdziale 9. znajdziesz opis innej metody pobierania informacji o napędach dyskowych, wykorzystującej funkcje API systemu Windows.

Wyświetlanie rozszerzonych informacji o plikach

Przykładowy program, który zamieszczono w tym podrozdziale, pozwala na wyświetlanie właściwości wszystkich plików zlokalizowanych w danym katalogu. Rodzaj dostępnych informacji zależy od typu poszczególnych plików. Na przykład, pliki graficzne posiadają takie właściwości, jak producent i model aparatu fotograficznego, a pliki muzyczne takie właściwości jak wykonawca, tytuł, czas odtwarzania itd.

Rzeczywiste właściwości plików zależą od wersji używanego systemu Windows. Na przykład Windows Vista obsługuje 267 właściwości plików, a Windows 7 jeszcze więcej. Poniżej zamieszczamy kod procedury, która tworzy listę właściwości plików i umieszcza ją na aktywnym arkuszu:

```

Sub ListFileProperties()
    Dim i As Long
    Dim objShell As Object 'IShellDispatch4
    Dim objFolder As Object 'Folder3

    ' Utwórz obiekt
    Set objShell = CreateObject("Shell.Application")

    ' Wybierz folder
    Set objFolder = objShell.Namespace("C:\\")

    ' Pobierz i zapisz listę właściwości
    For i = 0 To 500
        Cells(i + 1, 1) =
            objFolder.GetDetailsOf(objFolder.Items, i)
    Next i
End Sub

```



Niestety wartości poszczególnych właściwości w różnych wersjach systemu Windows nie są spójne. Na przykład w systemie Windows 2000 właściwość **Title** jest przechowywana jako właściwość numer 11, w systemie Windows XP ta sama właściwość ma numer 10, w systemie Windows Vista ma numer 21, a w systemie Windows 7 numer 22.

Poniżej zamieszczamy kod procedury **FileInfo**, która wykorzystuje obiekt **ShellApplication**. Procedura przy użyciu funkcji **GetDirectory** (kodu tej procedury nie zamieszczamy) prosi użytkownika o wybranie katalogu i następnie wyświetla pierwszych 41 właściwości poszczególnych plików przechowywanych w tym katalogu.

```

Sub FileInfo()
    Dim c As Long, r As Long, i As Long
    Dim FileName As Object 'FolderItem2
    Dim objShell As Object 'IShellDispatch4
    Dim objFolder As Object 'Folder3

    ' Utwórz obiekt
    Set objShell = CreateObject("Shell.Application")

    ' Wybierz folder
    Set objFolder = objShell.Namespace(GetDirectory)
    ' Wstaw nagłówki na arkuszu
    Worksheets.Add
    c = 0
    For i = 0 To 40
        c = c + 1
        Cells(1, c) = objFolder.GetDetailsOf(objFolder.Items, i)
    Next i

    ' Przetwarz kolejne pliki
    r = 1
    For Each FileName In objFolder.Items
        c = 0
        r = r + 1
        For i = 0 To 40
            c = c + 1
            Cells(r, c) = objFolder.GetDetailsOf(FileName, i)
        Next i
    Next FileName
    ' Utwórz tablice
    ActiveSheet.ListObjects.Add xlSrcRange, Range("A1").CurrentRegion
End Sub

```

Na rysunku 25.4 przedstawiono przykładowy wynik działania tej procedury.

	A	B	C	D	E	F	G
1	Nazwa	Rozmiar	Typ elementu	Data modyfikacji	Data utworzenia	Data dostępu	Atrybuty
2	Eksport-import-CSV.xlsm	29,9 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
3	Excel Log.xlsxm	15,4 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-20 09:56	2013-03-01 20:49	2013-03-01 20:49	A
4	Export do HTML.xlsxm	19,7 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
5	Export do XML.xlsxm	19,2 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
6	Filtrowanie pliku tekstowego		Folder plików	2013-03-01 20:49	2013-03-01 20:49	2013-03-01 20:49	D
7	Funkcje plikowe.xlsxm	13,2 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
8	Informacja o plikach.xlsxm	21,8 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
9	Lista plików - rekurencja.xlsxm	21,5 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
10	Lista plików.xlsxm	22,3 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
11	Pakowanie plików ZIP.xlsxm	17,3 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
12	Pokaż informację o napędach.xlsxm	19,6 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-18 09:55	2013-03-01 20:49	2013-03-01 20:49	A
13	Przykład ADO01		Folder plików	2013-03-01 20:49	2013-03-01 20:49	2013-03-01 20:49	D
14	Przykład ADO02		Folder plików	2013-03-01 20:49	2013-03-01 20:49	2013-03-01 20:49	D
15	r20		Folder plików	2013-03-01 21:13	2013-03-01 21:13	2013-03-01 21:13	D
16	Rozpakowywanie plików ZIP.xlsxm	15,5 KB	Arkusz programu Microsoft Excel z obsługą makr	2010-10-20 09:56	2013-03-01 20:49	2013-03-01 20:49	A
17							
18							

Rysunek 25.4. Tabela zawierająca pełne informacje o plikach w danym katalogu

W naszym przykładzie do utworzenia obiektu Shell.Application wykorzystujemy metodę późnego wiązania, stąd obiekty są deklarowane w sposób ogólny. Jeżeli chciałbyś użyć metody wczesnego wiązania, powinieneś w edytorze VBE wybrać z menu głównego polecenie *Tools/References* i następnie w oknie dialogowym *References* utworzyć odpowiednie odwołanie do biblioteki *Microsoft Shell Controls and Automation*.



Skoroszyt z tym przykładem (*Informacja o plikach.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Operacje z plikami tekstowymi

W języku VBA istnieje szereg poleceń, które pozwalają na wykonywanie niskopoziomowych operacji na plikach. Wspomniane polecenia operacji wejścia-wyjścia dają znacznie większą kontrolę nad plikami niż zwykłe opcje importowania i eksportowania plików dostępne w Excelu.

Wyróżniamy trzy metody dostępu do pliku:

- **Dostęp sekwencyjny** — to metoda najbardziej popularna, pozwalająca odczytywać i zapisywać pojedyncze znaki lub całe wiersze danych.
- **Dostęp losowy** — wykorzystywany tylko w przypadku tworzenia aplikacji bazodanowych (nie powinno się tego robić w języku VBA, ponieważ istnieją lepsze techniki).
- **Dostęp binarny** — wykorzystywany jest w celu odczytywania lub zapisywania dowolnego bajta w pliku, na przykład podczas operacji zapisywania lub wyświetlania mapy bitowej (ten sposób jest bardzo rzadko wykorzystywany w języku VBA).

Ponieważ w języku VBA rzadko wykorzystuje się losowy lub binarny dostęp do plików, w tym rozdziale skoncentrujemy się na plikach o dostępie sekwencyjnym, w których wiersze danych są odczytywane kolejno od początku pliku. W przypadku zapisywania, dane są zapisywane i dołączane na końcu pliku.



W metodzie odczytywania i zapisywania plików tekstowych opisanej w tej książce zastosowano tradycyjne pojęcie kanału danych. Innym, alternatywnym rozwiązaniem jest zastosowanie podejścia obiektowego. Obiekt `FileSystemObject` zawiera obiekt `TextStream`, który można wykorzystać do odczytywania lub zapisywania plików tekstowych. Obiekt `FileSystemObject` należy do biblioteki *Windows Scripting Host*. Jak wspomniałem wcześniej, ta właściwość jest wyłączona w wielu systemach ze względu na duże ryzyko rozpowszechniania wirusów.

Otwieranie plików tekstowych

Do otwierania plików do zapisu lub odczytu służy instrukcja VBA `Open` (nie należy jej mylić z metodą `Open` obiektu `Workbook`). Zanim będziesz mógł odczytywać lub zapisywać dane, plik musi zostać wcześniej otwarty.

Instrukcja `Open` jest dość uniwersalna i ma całkiem złożoną składnię:

```
Open ścieżka For tryb [Access dostęp] [blokada] _  
As [#]numer_pliku [Len=rozmiar_rekordu]
```

- **ścieżka** (wymagany) — nazwa i ścieżka (opcjonalnie) pliku, który ma być otwarty.
- **tryb** (wymagany) — może mieć jedną z poniższych wartości:

- Append — tryb dostępu sekwencyjnego, który pozwala na czytanie danych lub ich dołączanie na końcu pliku;
- Input — tryb dostępu sekwencyjnego, który pozwala na czytanie danych, ale nie pozwala na ich zapisywanie;
- Output — tryb dostępu sekwencyjnego, który pozwala na czytanie danych lub ich zapisywanie (w tym trybie zawsze jest tworzony nowy plik, a istniejący wcześniej plik o tej samej nazwie jest usuwany);
- Binary — tryb dostępu losowego, który pozwala na odczytywanie lub zapisywanie danych bajt po bajcie;
- Random — tryb dostępu losowego pozwalający na czytanie lub zapis informacji w blokach, których rozmiar określa ostatni argument instrukcji Open — *rozmiar_rekordu*.
- *dostęp* (opcjonalny) — określa rodzaj operacji dozwolonych do wykonania z plikiem. Może mieć wartość Read (czytanie), Write (zapisywanie) lub Read Write (czytanie i zapisywanie).
- *blokada* (opcjonalny) — przydaje się w przypadku używania pliku jednocześnie przez wielu użytkowników. Dopuszczalne wartości to Shared (współdzielony), Lock Read (blokada odczytu), Lock Write (blokada zapisu) oraz Lock Read Write (blokada odczytu i zapisu).
- *numer_pliku* (wymagany) — numer pliku w zakresie od 1 do 511. Aby uzyskać następny wolny numer pliku, można skorzystać z funkcji FreeFile (opis funkcji FreeFile można znaleźć w punkcie „Przydzielanie numeru pliku” w dalszej części rozdziału).
- *rozmiar_rekordu* (opcjonalny) — rozmiar rekordu (dla plików o dostępie losowym) lub rozmiar bufora (dla plików o dostępie sekwencyjnym).

Odczytywanie plików tekstowych

Standardowa procedura odczytywania danych z pliku tekstowego w języku VBA składa się z następujących kroków:

1. Otwarcie pliku za pomocą instrukcji Open.
2. Określenie pozycji w pliku za pomocą funkcji Seek (opcjonalnie).
3. Odczytywanie danych z pliku (za pomocą instrukcji Input, Input # lub Line Input #).
4. Zamknięcie pliku za pomocą instrukcji Close.

Zapisywanie danych do plików tekstowych

Standardowa procedura zapisywania danych do pliku tekstowego jest następująca:

1. Otwarcie lub utworzenie pliku za pomocą instrukcji Open.
2. Określenie pozycji w pliku za pomocą funkcji Seek (opcjonalnie).

3. Zapis danych do pliku za pomocą instrukcji Write # lub Print #.
4. Zamknięcie pliku za pomocą instrukcji Close.

Przydzielanie numeru pliku

Większość programistów VBA po prostu przydziela odpowiedni numer pliku i podaje go jako argument instrukcji Open, na przykład:

```
Open "mójplik.txt" For Input As #1
```

Gdy taka instrukcja zostanie wykonana, w dalszej części kodu można się odwoływać do pliku jako do #1.

Jeżeli plik jest otwierany w czasie, kiedy inny jest już otwarty, kolejny plik można oznaczyć jako #2:

```
Open "innego.txt" For Input As #2
```

Innym sposobem uzyskania numeru pliku jest użycie funkcji FreeFile w celu pobrania uchwytu do pliku. Po wykonaniu tej funkcji można odwoływać się do pliku za pomocą zmiennej. Oto przykład:

```
FileHandle = FreeFile  
Open "mójplik.txt" For Input As FileHandle
```

Określanie lub ustawianie pozycji w pliku

W przypadku sekwencyjnego dostępu do plików znajomość bieżącej lokalizacji wewnętrz pliku jest rzadko potrzebna. Jeżeli jednak z jakiegoś powodu taka informacja jest potrzebna, możesz użyć funkcji Seek.

Import i eksport plików tekstowych w Excelu

Excel obsługuje trzy typy plików tekstowych:

- **CSV** (ang. *Comma-Separated Value*) — kolumny danych są rozdzielane przecinkami, a każdy wiersz kończy się znakiem powrotu karetki (w niektórych narodowych wersjach Excela zamiast przecinka używany jest średnik).
- **PRN** — kolumny danych są wyrównywane przez pozycje znaków, a każdy wiersz kończy się znakiem powrotu karetki. Takie pliki są nazywane również plikami o *stałej szerokości kolumn*.
- **TXT** (pliki z danymi rozdzielanymi znakami tabulacji) — kolumny danych są rozdzielane znakami tabulacji, a każdy wiersz kończy się znakiem powrotu karetki.

Jeżeli spróbujesz otworzyć plik tekstowy za pomocą polecenia *PLIK/Otwórz*, na ekranie może pojawić się okno *Kreatora importu tekstu*, ułatwiającego poprawne zdefiniowanie poszczególnych kolumn. Jeżeli plik tekstowy jest rozdzielany znakami tabulacji lub spacji, Excel zazwyczaj otwiera plik bez wyświetlenia kreatora. Jeżeli dane nie zostaną poprawnie odczytane, zamknij plik i spróbuj zmienić rozszerzenie jego nazwy na .txt.

Kreator konwersji tekstu na kolumny jest niemal identyczny, ale działa poprawnie tylko w przypadku danych zapisanych w pojedynczej kolumnie. Aby go uruchomić, przejdź na kartę *DANE* i naciśnij przycisk *Tekst jako kolumny*, znajdujący się w grupie opcji *Narzędzia danych*.

Instrukcje pozwalające na odczytywanie i zapisywanie plików

W języku VBA znajdziemy kilka instrukcji pozwalających na odczytywanie i zapisywanie danych do pliku.

Do odczytywania danych z plików o dostępie sekwencyjnym służą trzy instrukcje:

- Input — odczytuje z pliku określona liczbę znaków.
- Input # — odczytuje dane z pliku, przypisując wartości do serii zmiennych oddzielonych od siebie przecinkami.
- Line Input # — odczytuje cały wiersz danych, ograniczony znakami powrotu karetki i (lub) wysunięcia wiersza.

Do zapisywania danych w plikach o dostępie sekwencyjnym służą dwie instrukcje:

- Write # — zapisuje do pliku ciąg wartości, gdzie kolejne wartości są od siebie oddzielone przecinkami i ujęte w apostrofy. W przypadku zakończenia instrukcji średnikiem po wartościach nie jest wprowadzana sekwencja znaków CR LF. Dane zapisywane do pliku za pomocą instrukcji Write # zazwyczaj są odczytywane z pliku za pomocą instrukcji Input #.
- Print # — zapisuje do pliku ciąg wartości, gdzie kolejne wartości są od siebie oddzielone znakiem tabulacji. W przypadku zakończenia instrukcji średnikiem po wartościach nie jest wprowadzana sekwencja znaków CR LF. Dane zapisywane do pliku za pomocą instrukcji Print # zazwyczaj są odczytywane z pliku za pomocą instrukcji Line Input # lub Input.

Przykłady wykonywania operacji na plikach

W tym podrozdziale przedstawimy kilka przykładów ilustrujących różne techniki wykonywania operacji na plikach tekstowych.

Importowanie danych z pliku tekstopowego

Procedura przedstawiona poniżej odczytuje dane z pliku tekstopowego, a następnie umieszcza każdy wiersz danych w osobnej komórce (począwszy od aktywnej komórki):

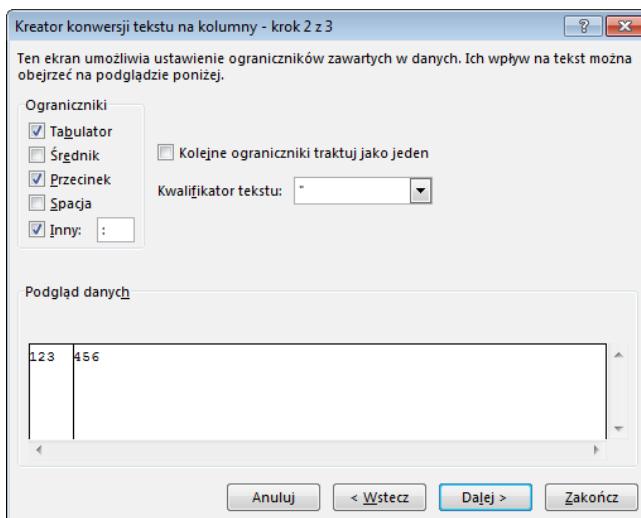
```
Sub ImportData()
    Open "c:\helion\mój_plik.txt" For Input As #1
    r = 0
    Do Until EOF(1)
        Line Input #1, data
        ActiveCell.Offset(r, 0) = data
        r = r + 1
    Loop
    Close #1
End Sub
```

W praktyce taka procedura nie będzie jednak zbyt przydatna, ponieważ każdy wiersz danych jest wpisywany do osobnej komórki. W takiej sytuacji o wiele łatwiejszym rozwiązaniem będzie po prostu bezpośrednie otwarcie pliku tekstowego za pomocą polecenia *PLIK/Otwórz*.

Kiedy Excel niepoprawnie importuje dane...

Czy zdarzyło Ci się kiedyś, że próbowałeś zaimportować plik CSV lub wkleić skopiowane dane do arkusza i okazało się, że Excel niepoprawnie rozdzielił dane do poszczególnych komórek? Jeżeli tak, to przyczyną jest zapewne funkcjonowanie mechanizmu *Tekst jako kolumny*. Poniżej przedstawiamy wygląd drugiego ekranu kreatora konwersji tekstu na kolumny, który pozwala na podzielenie pojedynczej kolumny danych oddzielonych separatorami na wiele kolumn.

W tym przypadku zdefiniowane zostały trzy ograniczniki (separatory) danych: tabulator, przecinek oraz dwukropki.



Dzielenie tekstu na poszczególne kolumny jest bardzo użytecznym mechanizmem. Problem polega jednak na tym, że Excel próbuje być pomocny i zapamiętuje ustawienia separatorów dla operacji importowania kolejnych plików CSV i wklejania skopiowanych danych. Czasami zapamiętanie takich ustawień rzeczywiście może być bardzo przydatne, ale jednak w praktyce chyba bardziej przeszko-dza niż pomaga. Aby usunąć ustawienia separatorów, powinieneś przywołać na ekran kreatora konwersji, usunąć zaznaczenie zbędnych separatorów i nacisnąć przycisk *Anuluj*.

Jeżeli importujesz lub wklejasz dane za pomocą makra, nie ma ono żadnej bezpośrednią możliwości sprawdzenia lub skasowania ustawień separatorów. Rozwiązaniem tego problemu może być swego rodzaju zasymulowanie operacji konwersji tekstu na kolumny. Procedura, której kod prezentujemy poniżej, dokonuje właśnie takiej operacji, symulując usunięcie zaznaczeń wszystkich separatorów (ale nie dokonuje żadnych modyfikacji arkusza).

```
Sub ClearTextToColumns()
    On Error Resume Next
    If IsEmpty(Range("A1")) Then Range("A1") = "XYZZY"
    Range("A1").TextToColumns Destination:=Range("A1"),
        DataType:=xlDelimited,
        TextQualifier:=xlDoubleQuote,
```

```
ConsecutiveDelimiter:=False, _
Tab:=False, _
Semicolon:=False, _
Comma:=False, _
Space:=False, _
Other:=False, _
OtherChar:="""
If Range("A1") = "XYZZY" Then Range("A1") = ""
If Err.Number <> 0 Then MsgBox Err.Description
End Sub
```

Makro zakłada, że arkusz jest aktywny i nie jest chroniony. Zwróć uwagę, że zawartość komórki A1 nie zostanie zmieniona, ponieważ przy wywołaniu metody `TextToColumns` nie zostały zdefiniowane żadne operacje. Jeżeli komórka A1 jest pusta, kod wstawia do niej tymczasowy łańcuch tekstu (ponieważ metoda `TextToColumns` nie będzie działać poprawnie, jeżeli komórka jest pusta). Przed zakończeniem działania procedury tymczasowy łańcuch tekstu jest usuwany.

Eksportowanie zakresu do pliku tekstowego

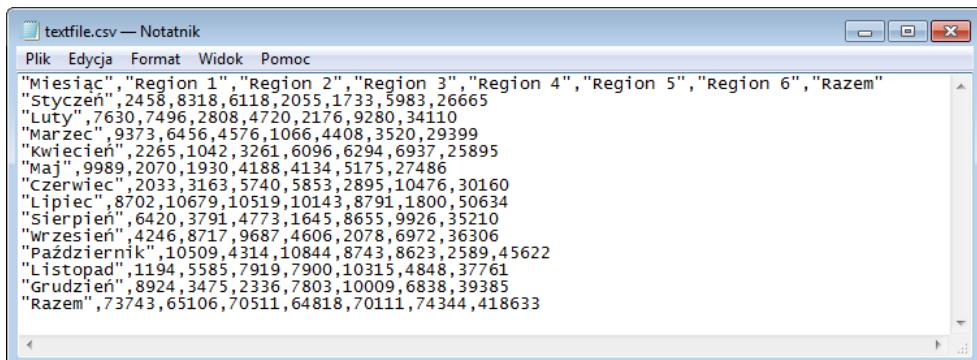
Procedura przedstawiona poniżej zapisuje dane z zaznaczonego zakresu komórek arkusza do pliku tekstowego w formacie CSV. Excel potrafi oczywiście bezpośrednio eksportować dane do pliku w formacie CSV, ale w taki sposób można eksportować tylko całe arkusze, podczas gdy nasza procedura działa dla dowolnego, zaznaczonego obszaru arkusza.

```
Sub ExportRange()
Dim Filename As String
Dim NumRows As Long, NumCols As Integer
Dim r As Long, c As Integer
Dim Data
Dim ExpRng As Range
Set ExpRng = Selection
NumCols = ExpRng.Columns.Count
NumRows = ExpRng.Rows.Count
Filename = Application.DefaultFilePath & "\textfile.csv"
Open Filename For Output As #1
For r = 1 To NumRows
    For c = 1 To NumCols
        Data = ExpRng.Cells(r, c).Value
        If IsNumeric(Data) Then Data = Val(Data)
        If IsEmpty(ExpRng.Cells(r, c)) Then Data = ""
        If c <> NumCols Then
            Write #1, Data;
        Else
            Write #1, Data
        End If
    Next c
    Next r
Close #1
End Sub
```

W procedurze dwukrotnie wykorzystano funkcję `Write #`. Pierwsza instrukcja kończy się średnikiem, a zatem sekwencja CR LF nie będzie zapisywana. Dla ostatniej komórki w wierszu, w drugiej instrukcji `Write #`, nie użyto średnika, dzięki czemu następny zapis do pliku zostanie umieszczony w nowym wierszu.

Do zapisania zawartości komórek wykorzystana została zmienna o nazwie Data. Jeżeli komórka zawiera format liczbowy, zmienna jest przekształcana na liczbę. Dzięki tej czynności dane liczbowe nie zostaną zapisane ze znakami cudzysłowu. Jeżeli komórka jest pusta, wartość jej właściwości Value wynosi 0. Z tego powodu kod sprawdza, czy komórki nie są puste (za pomocą funkcji IsEmpty) i wstawia pusty ciąg znaków zamiast wartości 0.

Na rysunku 25.5 przedstawiono przykładową zawartość pliku będącego wynikiem działania procedury.



"Miesiąć"	"Region 1"	"Region 2"	"Region 3"	"Region 4"	"Region 5"	"Region 6"	"Razem"	
"Styczeń"	2458	8318	6118	2055	1733	5983	26665	
"Luty"	7630	7496	2808	4720	2176	9280	34110	
"Marzec"	9373	6456	4576	1066	4408	3520	29399	
"Kwiecień"	2265	1042	3261	6096	6294	6937	25895	
"Maj"	9989	2070	1930	4188	4134	5175	27486	
"Czerwiec"	2033	3163	5740	5853	2895	10476	30160	
"Lipiec"	8702	10679	10519	10143	8791	1800	50634	
"Sierpień"	6420	3791	4773	1645	8655	9926	35210	
"Wrzesień"	4246	8717	9687	4606	2078	6972	36306	
"Październik"	10509	4314	10844	8743	8623	2589	45622	
"Listopad"	1194	5585	7919	7900	10315	4848	37761	
"Grudzień"	8924	3475	2336	7803	10009	6838	39385	
"Razem"	73743	65106	70511	64818	70111	74344	418633	

Rysunek 25.5. Ten plik tekstowy został wygenerowany za pomocą kodu VBA



Skoroszyt z tym przykładem (*Eksport-import-CSV.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Importowanie pliku tekstowego do zakresu

Procedura przedstawiona poniżej odczytuje dane z pliku CSV utworzonego w poprzednim przykładzie, a następnie zapisuje uzyskane wartości do arkusza, rozpoczynając od aktywnej komórki. Program odczytuje każdy znak i przetwarza wiersze danych w celu wyszukania przecinków oddzielających kolumny i usunięcia cudzysłowów przed zapisaniem danych do arkusza.

```
Sub ImportRange()
    Dim ImpRng As Range
    Dim Filename As String
    Dim r As Long, c As Integer
    Dim txt As String, Char As String * 1
    Dim Data
    Dim i As Integer

    Set ImpRng = ActiveCell
    On Error Resume Next
    Filename = Application.DefaultFilePath & "\textfile.csv"
    Open Filename For Input As #1
    If Err <> 0 Then
        MsgBox "Nie znaleziono pliku: " & Filename, vbCritical, "BŁĄD"
        Exit Sub
    End If

    Do While Not EOF(1)
        Line Input #1, txt
        For i = 1 To Len(txt)
            Char = Mid(txt, i, 1)
            If Char = "," Then
                Data(c) = Data(c) & ","
            Else
                Data(c) = Data(c) & Char
            End If
        Next i
        Data(c + 1) = ""
        c = c + 1
    Loop
    ImpRng.Value = Data
End Sub
```

```
End If
r = 0
c = 0
txt = ""
Application.ScreenUpdating = False
Do Until EOF(1)
    Line Input #1, Data
    For i = 1 To Len(Data)
        Char = Mid(Data, i, 1)
        If Char = "," Then 'przecinek
            ActiveCell.Offset(r, c) = txt
            c = c + 1
            txt = ""
        ElseIf i = Len(Data) Then 'koniec wiersza
            If Char <> Chr(34) Then txt = txt & Char
            ActiveCell.Offset(r, c) = txt
            txt = ""
        ElseIf Char <> Chr(34) Then
            txt = txt & Char
        End If
        Next i
        c = 0
        r = r + 1
    Loop
    Close #1
    Application.ScreenUpdating = True
End Sub
```



Procedura pokazana powyżej poradzi sobie z większością danych, ale ma pewną wadę: nie potrafi poprawnie przetwarzać danych zawierających przecinki lub znaki cudzysłowu. Dodatkowo zimportowane daty będą otoczone znakami #, na przykład #2013-05-12#.

Rejestrowanie wykorzystania Excela

Kod zaprezentowany w tym punkcie zapisuje dane do pliku tekstowego podczas każdej operacji uruchamiania i zamykania Excela. Aby zaprezentowana procedura działała niezawodnie, musi być umieszczona w skoroszycie, który otwiera się za każdym razem, kiedy uruchamiamy Excela — do tego celu idealnie nadaje się osobisty arkusz makr (ang. *Personal Macro Workbook*).

Poniższa procedura jest umieszczona w module kodu obiektu ThisWorkbook i jest wykonywana podczas otwierania pliku:

```
Private Sub Workbook_Open()
    Open Application.Path & "\excelusage.txt" For Append As #1
    Print #1, "Uruchomienie programu Excel " & Now
    Close #1
End Sub
```

Procedura dodaje wiersz do pliku o nazwie *excelusage.txt*. Nowy wiersz zawiera bieżącą datę i godzinę i może mieć następującą postać:

Uruchomienie programu Excel 2013-03-16 19:27:43

Pokazana poniżej procedura wykonuje się podczas zamykania skoroszytu. Jej działanie polega na dodaniu do pliku tekstowego wiersza zawierającego frazę *Zakończenie pracy programu* wraz z bieżącą datą i godziną.

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Open Application.Path & "\excelusage.txt" _
        For Append As #1
    Print #1, "Zakończenie pracy programu Excel " & Now
    Close #1
End Sub
```



Skoroszyt z tym przykładem (Excel Log.xlsxm) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Więcej szczegółowych informacji na temat procedur obsługi zdarzeń Workbook_Open oraz Workbook_BeforeClose znajdziesz w rozdziale 17.

Filtrowanie zawartości pliku tekstuowego

W przykładzie zaprezentowanym poniżej zademonstrujemy metodę jednoczesnego przetwarzania dwóch plików tekstowych. Procedura FilterFile odczytuje dane z pliku tekstuowego (*infile.txt*) i kopiuje wiersze zawierające określony ciąg znaków (na przykład "Styczeń") do drugiego pliku tekstuowego (*output.txt*).

```
Sub FilterFile()
    Open ThisWorkbook.Path & "\infile.txt" For Input As #1
    Open Application.DefaultFilePath & "\output.txt" For Output As #2
    TextToFind = "Styczeń"
    Do Until EOF(1)
        Line Input #1, data
        If InStr(1, data, TextToFind) Then
            Print #2, data
        End If
    Loop
    Close #1 ' Zamknij wszystkie pliki
End Sub
```



Skoroszyt z tym przykładem (Filtrowanie pliku tekstuowego.xlsxm) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Eksportowanie zakresu komórek do pliku HTML

Przykład, który omówimy w tym podrozdziale, ilustruje, w jaki sposób wyeksportować zakres komórek do pliku w formacie HTML. Pliki w formacie HTML to pliki tekstowe zawierające specjalne znaczniki formatowania, które opisują sposób wyświetlania informacji w przeglądarce sieciowej.

Dlaczego zatem nie skorzystamy z polecenia Excela *PLIK/Zapisz jako* i jako typ pliku wybierzemy *Strona sieci Web?* Otóż, procedura zaprezentowana poniżej ma istotną zaletę: nie tworzy niepotrzebnie złożonego kodu HTML. Kiedy za pomocą procedury ExportToHTML

wyeksportowałem zakres 70 komórek, uzyskałem plik o rozmiarze 2,6 kB. Następnie do tego samego zadania wykorzystałem polecenie *Plik/Zapisz jako/Strona sieci Web*. Uzyskany plik miał rozmiar 15,8 kB — ponad sześć razy więcej.

Trzeba jednak pamiętać, że procedura ExportToHTML nie zapisuje formatowania komórek. Jedyne zapisywane elementy formatowania to pogrubienie i pochylenie oraz wyrównanie poziome. Poza tym istnieje dodatkowe poważne ograniczenie: procedura nie obsługuje scalonych komórek. Nie zmienia to jednak w niczym faktu, że poniższą procedurę można wykorzystać w wielu sytuacjach jako bazę, którą później dostosujemy do indywidualnych wymagań użytkownika.

```
Sub ExportToHTML()
    Dim Filename As Variant
    Dim TDOpenTag As String, TDCloseTag As String
    Dim CellContents As String
    Dim Rng As Range
    Dim r As Long, c As Integer

    ' Użyj zaznaczonego zakresu komórek
    Set Rng = Application.Intersect(ActiveSheet.UsedRange, Selection)
    If Rng Is Nothing Then
        MsgBox "Brak danych do eksportu.", vbCritical
        Exit Sub
    End If
    ' Pobierz nazwę pliku
    Filename = Application.GetSaveAsFilename( _
        InitialFileName:="zakres.htm",
        fileFilter:="HTML Files (*.htm), *.htm")
    If Filename = False Then Exit Sub

    ' Otwórz plik tekstowy
    Open Filename For Output As #1

    ' Zapisz znaczniki
    Print #1, "<HTML>"
    Print #1, "<TABLE BORDER=0 CELLPADDING=3>"

    ' Przetwarzanie komórek w pęli
    For r = 1 To Rng.Rows.Count
        Print #1, "<TR>"
        For c = 1 To Rng.Columns.Count
            Select Case Rng.Cells(r, c).HorizontalAlignment
                Case xlHAlignLeft
                    TDOpenTag = "<TD ALIGN=LEFT>"
                Case xlHAlignCenter
                    TDOpenTag = "<TD ALIGN=CENTER>"
                Case xlHAlignGeneral
                    If IsNumeric(Rng.Cells(r, c)) Then
                        TDOpenTag = "<TD ALIGN=RIGHT>"
                    Else
                        TDOpenTag = "<TD ALIGN=LEFT>"
                    End If
                Case xlHAlignRight
                    TDOpenTag = "<TD ALIGN=RIGHT>"
            End Select

            TDCloseTag = "</TD>"
            If Rng.Cells(r, c).Font.Bold Then
```

```

TDOpenTag = TDOpenTag & "<B>"
TDCloseTag = "</B>" & TDCloseTag
End If
If Rng.Cells(r, c).Font.Italic Then
    TDOpenTag = TDOpenTag & "<I>"
    TDCloseTag = "</I>" & TDCloseTag
End If
CellContents = Rng.Cells(r, c).Text
Print #1, TDOpenTag & CellContents & TDCloseTag
Next c
Print #1, "</TR>
Next r
' Zamknij tabelę
Print #1, "</TABLE>"
Print #1, "</HTML>
' Zamknij plik
Close #1
' Powiadom użytkownika
MsgBox Rng.Count & " komórek wyeksportowano do pliku " & Filename
End Sub

```

Procedura rozpoczyna działanie od określenia zakresu komórek, które mają zostać wyeksportowane, i dokonuje tego, odszukując punkt przecięcia zaznaczonego zakresu i wykorzystanego obszaru w arkuszu. Takie rozwiązanie zapewnia, że nie będą przetwarzane całe wiersze lub kolumny. Następnie użytkownik jest proszony o podanie nazwy pliku, po czym plik zostaje otwarty. Większość działań wykonywanych jest w dwóch pętlach For ... Next. Kod generuje odpowiednie znaczniki HTML i zapisuje informacje do pliku tekstowego. Jedynym złożonym fragmentem jest określenie sposobu wyrównania poziomego komórek (Excel nie podaje wprost takiej informacji). Na koniec następuje zamknięcie pliku i wyświetlenie odpowiedniego komunikatu dla użytkownika, zawierającego podsumowanie wykonanej operacji.

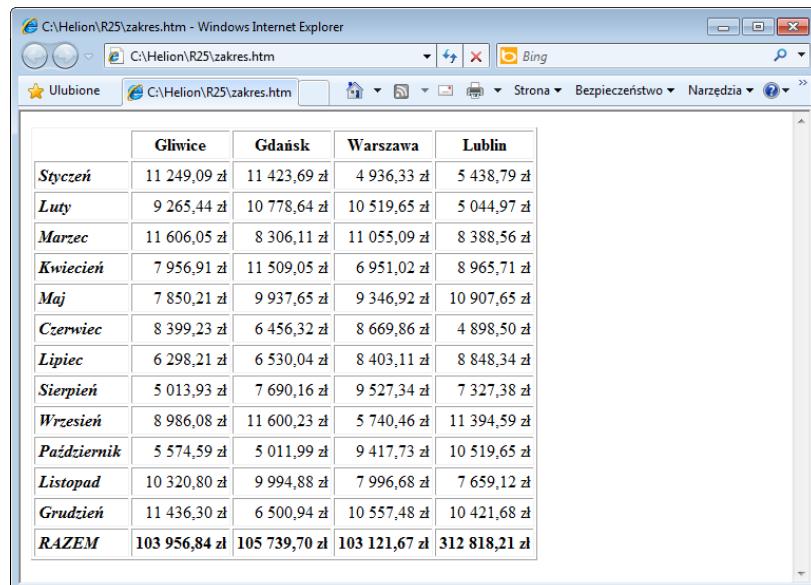
Na rysunku 25.6 pokazano zakres komórek w arkuszu, a na rysunku 25.7 wygląd tego samego zakresu komórek po przekształceniu na HTML i wyświetleniu w przeglądarce sieciowej.

Rysunek 25.6.
Zakres w arkuszu,
który zostanie
przekształcony
na plik HTML

	A	B	C	D	E	F
1		Gliwice	Gdańsk	Warszawa	Lublin	
2	<i>Styczeń</i>	11 249,09 zł	11 423,69 zł	4 936,33 zł	5 438,79 zł	
3	<i>Luty</i>	9 265,44 zł	10 778,64 zł	10 519,65 zł	5 044,97 zł	
4	<i>Marzec</i>	11 606,05 zł	8 306,11 zł	11 055,09 zł	8 388,56 zł	
5	<i>Kwiecień</i>	7 956,91 zł	11 509,05 zł	6 951,02 zł	8 965,71 zł	
6	<i>Maj</i>	7 850,21 zł	9 937,65 zł	9 346,92 zł	10 907,65 zł	
7	<i>Czerwiec</i>	8 399,23 zł	6 456,32 zł	8 669,86 zł	4 898,50 zł	
8	<i>Lipiec</i>	6 298,21 zł	6 530,04 zł	8 403,11 zł	8 848,34 zł	
9	<i>Sierpień</i>	5 013,93 zł	7 690,16 zł	9 527,34 zł	7 327,38 zł	
10	<i>Wrzesień</i>	8 986,08 zł	11 600,23 zł	5 740,46 zł	11 394,59 zł	
11	<i>Październik</i>	5 574,59 zł	5 011,99 zł	9 417,73 zł	10 519,65 zł	
12	<i>Listopad</i>	10 320,80 zł	9 994,88 zł	7 996,68 zł	7 659,12 zł	
13	<i>Grudzień</i>	11 436,30 zł	6 500,94 zł	10 557,48 zł	10 421,68 zł	
14	RAZEM	103 956,84 zł	105 739,70 zł	103 121,67 zł	312 818,21 zł	
15						
16						

Rysunek 25.7.

Dane z arkusza po przekształceniu na plik HTML



The screenshot shows a table of monthly sales data for four cities: Gliwice, Gdańsk, Warszawa, and Lublin. The data spans from January to December, ending with a summary row for the year. The table is displayed in a standard grid format with alternating row colors.

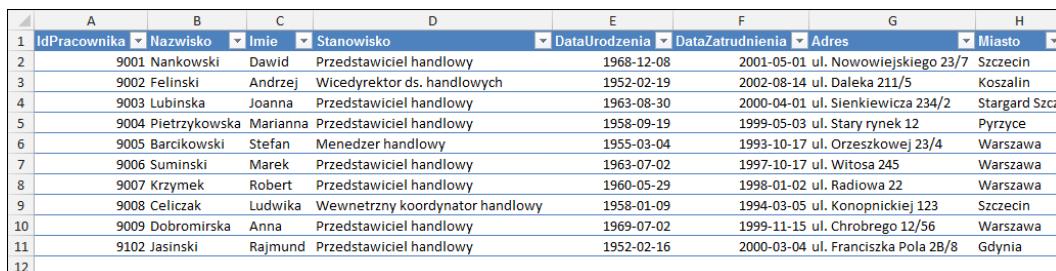
	Gliwice	Gdańsk	Warszawa	Lublin
Styczeń	11 249,09 zł	11 423,69 zł	4 936,33 zł	5 438,79 zł
Luty	9 265,44 zł	10 778,64 zł	10 519,65 zł	5 044,97 zł
Marzec	11 606,05 zł	8 306,11 zł	11 055,09 zł	8 388,56 zł
Kwiecień	7 956,91 zł	11 509,05 zł	6 951,02 zł	8 965,71 zł
Maj	7 850,21 zł	9 937,65 zł	9 346,92 zł	10 907,65 zł
Czerwiec	8 399,23 zł	6 456,32 zł	8 669,86 zł	4 898,50 zł
Lipiec	6 298,21 zł	6 530,04 zł	8 403,11 zł	8 848,34 zł
Sierpień	5 013,93 zł	7 690,16 zł	9 527,34 zł	7 327,38 zł
Wrzesień	8 986,08 zł	11 600,23 zł	5 740,46 zł	11 394,59 zł
Październik	5 574,59 zł	5 011,99 zł	9 417,73 zł	10 519,65 zł
Listopad	10 320,80 zł	9 994,88 zł	7 996,68 zł	7 659,12 zł
Grudzień	11 436,30 zł	6 500,94 zł	10 557,48 zł	10 421,68 zł
RAZEM	103 956,84 zł	105 739,70 zł	103 121,67 zł	312 818,21 zł



Skoroszyt z tym przykładem (*Export do HTML.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Eksportowanie zakresu komórek do pliku XML

Procedura, której kod zamieszczony został poniżej, eksportuje zakres komórek arkusza do prostego pliku XML. Jak wiadomo, w plikach w formacie XML znaczniki otaczają wszystkie elementy danych. Nasza procedura jako znaczniki XML wykorzystuje etykiety zapisane w pierwszym wierszu arkusza. Na rysunku 25.8 pokazano zakres komórek arkusza, natomiast na rysunku 25.9 zawartość pliku XML wyświetlzoną w przeglądarce sieciowej Internet Explorer.

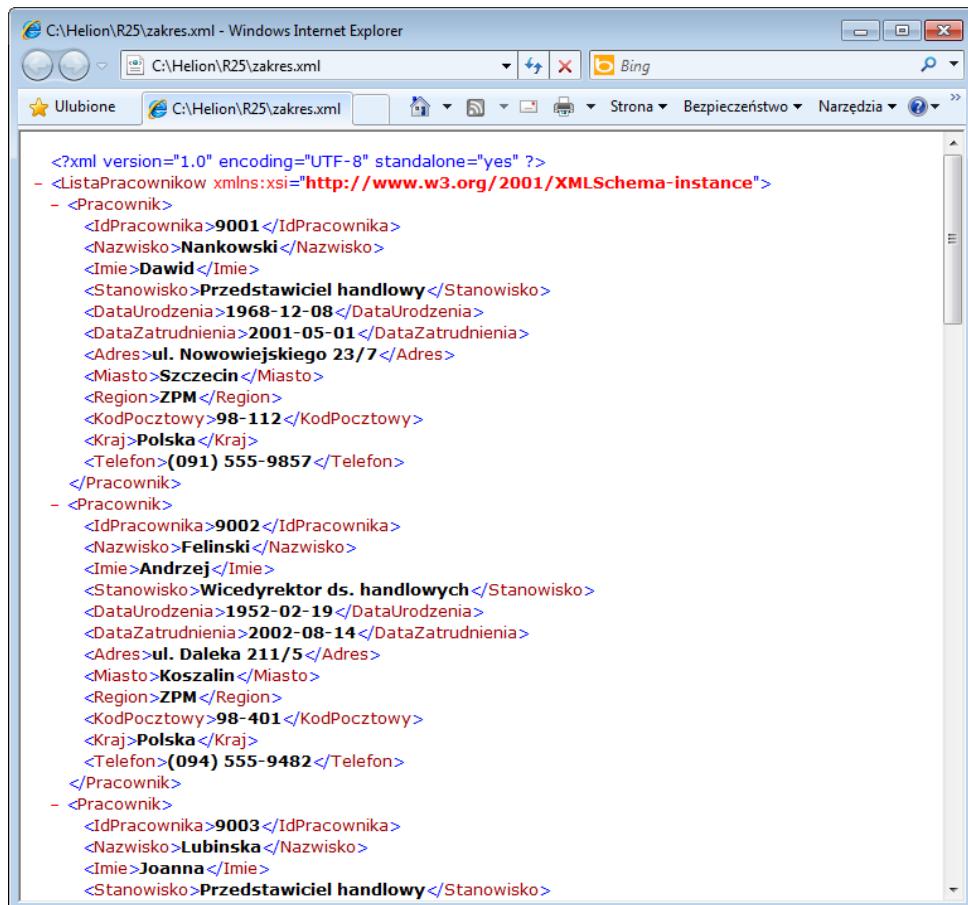


The screenshot shows a table of employee data in Excel. The columns are labeled: IdPracownika, Nazwisko, Imię, Stanowisko, DataUrodzenia, DataZatrudnienia, Adres, and Miasto. The data includes various employees with their names, job titles, birth dates, hire dates, addresses, and cities.

	A	B	C	D	E	F	G	H
1	IdPracownika	Nazwisko	Imię	Stanowisko	DataUrodzenia	DataZatrudnienia	Adres	Miasto
2	9001	Nankowski	Dawid	Przedstawiciel handlowy	1968-12-08	2001-05-01	ul. Nowowiejskiego 23/7	Szczecin
3	9002	Felinski	Andrzej	Wicedyrektor ds. handlowych	1952-02-19	2002-08-14	ul. Daleka 211/5	Koszalin
4	9003	Lubinska	Joanna	Przedstawiciel handlowy	1963-08-30	2000-04-01	ul. Sienkiewicza 234/2	Stargard Szcz
5	9004	Pietrzykowska	Marianna	Przedstawiciel handlowy	1958-09-19	1999-05-03	ul. Stary rynek 12	Pyrzycy
6	9005	Barcikowski	Stefan	Menedżer handlowy	1955-03-04	1993-10-17	ul. Orzeszkowej 23/4	Warszawa
7	9006	Sumiński	Marek	Przedstawiciel handlowy	1963-07-02	1997-10-17	ul. Wiltosa 245	Warszawa
8	9007	Krzymek	Robert	Przedstawiciel handlowy	1960-05-29	1998-01-02	ul. Radiona 22	Warszawa
9	9008	Celiczak	Ludwika	Weźwnetrzny koordynator handlowy	1958-01-09	1994-03-05	ul. Konopnickiej 123	Szczecin
10	9009	Dobromirska	Anna	Przedstawiciel handlowy	1969-07-02	1999-11-15	ul. Chrobrego 12/56	Warszawa
11	9102	Jasinski	Rajmund	Przedstawiciel handlowy	1952-02-16	2000-03-04	ul. Franciszka Pola 2B/8	Gdynia
12								

Rysunek 25.8. Dane z tego zakresu zostaną przekształcone na format XML

Pomimo iż w Excelu 2003 wprowadzono ulepszoną obsługę formatu XML, to jednak nawet Excel 2013 nie potrafi utworzyć pliku XML z dowolnego zakresu danych bez zdefiniowania pliku odwzorowań (schematu) dla tych danych.



Rysunek 25.9. Dane z arkusza po przekształceniu na format XML

Kod procedury ExportToXML został przedstawiony poniżej. Jak łatwo zauważyć, ma ona wiele wspólnego z procedurą ExportToHTML pokazaną w poprzednim podrozdziale.

```

Sub ExportToXML()
  Dim Filename As Variant
  Dim Rng As Range
  Dim r As Long, c As Long

  ' Ustaw zakres
  Set Rng = Range("Table1[#A11]")

  ' Pobierz nazwę pliku
  Filename = Application.GetSaveAsFilename( _
    InitialFileName:="zakres.xml", _
    fileFilter:="XML Files (*.xml), *.xml")
  If Filename = False Then Exit Sub

  ' Otwórz plik tekstowy
  Open Filename For Output As #1

```

```

' Zapisz znaczniki <xml>
Print #1, "<?xml version=""1.0"" encoding=""UTF-8"" standalone=""yes""?>" 
Print #1, "<ListaPracownikow xmlns:xsi=""http://www.w3.org/2001/
➥XMLSchema-instance"">" 

' Przetwarzanie komórek w pętli
For r = 2 To Rng.Rows.Count
    Print #1, "<Pracownik>" 
    For c = 1 To Rng.Columns.Count
        Print #1, "<" & Rng.Cells(1, c) & ">" 
        If IsDate(Rng.Cells(r, c)) Then
            Print #1, Format(Rng.Cells(r, c), "yyyy-mm-dd") 
        Else
            Print #1, Rng.Cells(r, c).Text 
        End If
        Print #1, "</>" & Rng.Cells(1, c) & ">" 
    Next c
    Print #1, "</Pracownik>" 
Next r
' Zamknij tabelę
Print #1, "</ListaPracownikow>" 
' Zamknij plik
Close #1

' Powiadom użytkownika
MsgBox Rng.Rows.Count - 1 & " wierszy wyeksportowano do pliku " & Filename
End Sub

```

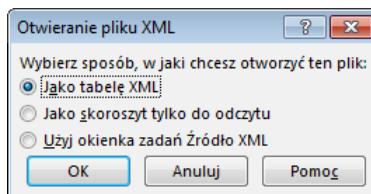


Skoroszyty z przykładami (*Export do XML.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Utworzone w powyższy sposób pliki w formacie XML mogą być otwierane w programie Excel. Kiedy otwierasz plik XML, na ekranie pojawia się okno dialogowe przedstawione na rysunku 25.10. Jeżeli wybierzesz opcję *Jako tabelę XML*, plik zostanie wyświetlony w postaci tabeli. Pamiętaj, że formuły zapisane w oryginalnej tabeli nie zostaną zachowane.

Rysunek 25.10.

W przypadku plików XML Excel oferuje trzy sposoby ich otwarcia



Pakowanie i rozpakowywanie plików

Prawdopodobnie najczęściej spotykanym formatem spakowanych plików jest popularny ZIP. Nawet dokumenty programu Excel 2010 są zapisywane w tym formacie (aczkolwiek nie używają rozszerzenia *.zip*). Pliki w formacie ZIP mogą zawierać dowolną liczbę plików, a nawet całe struktury katalogów. Zawartość plików ma bezpośredni wpływ na stopień kompresji. Na przykład pliki graficzne w formacie JPG są już skompresowane, stąd zapisanie ich dodatkowo w formacie ZIP w niewielkim stopniu wpłynie na ich rozmiar.



Skoroszyty z przykładami (*Pakowanie plików ZIP.xlsxm* oraz *Rozpakowywanie plików ZIP.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pww.htm>).

Pakowanie plików do formatu ZIP

Kod zamieszczony poniżej ilustruje sposób tworzenia spakowanego pliku w formacie ZIP zawierającego grupę plików wybranych przez użytkownika. Procedura ZipFiles wyświetla na ekranie okno dialogowe, za pomocą którego użytkownik może wybrać pliki przeznaczone do spakowania. Następnie w domyślnym katalogu programu Excel tworzony jest plik w formacie ZIP o nazwie *compressed.zip* zawierający spakowane pliki.

```
Sub ZipFiles()
    Dim ShellApp As Object
    Dim FileNameZip As Variant
    Dim FileNames As Variant
    Dim i As Long, FileCount As Long

    ' Pobierz nazwy plików
    FileNames = Application.GetOpenFilename _
        (FileFilter:="All Files (*.*) , *.*" , _
        FilterIndex:=1, _
        Title:="Zaznacz pliki przeznaczone do spakowania" , _
        MultiSelect:=True)

    ' Zakończ, jeżeli operacja została anulowana
    If Not IsArray(FileNames) Then Exit Sub

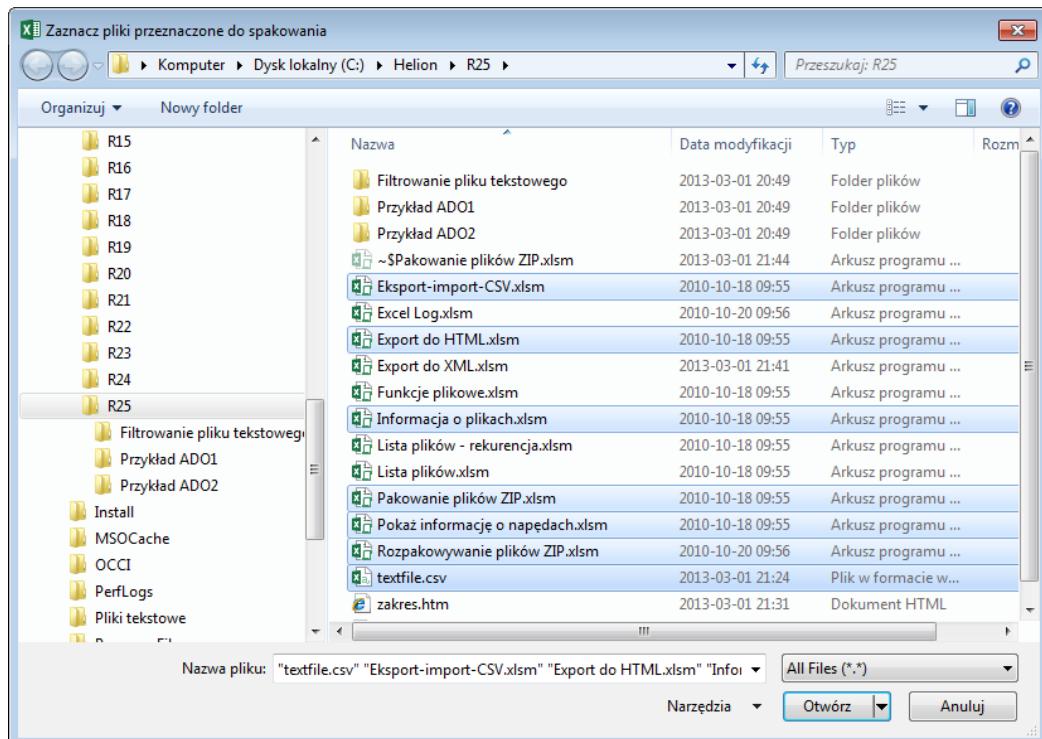
    FileCount = UBound(FileNames)
    FileNameZip = Application.DefaultFilePath & "\compressed.zip"

    ' Utwórz pusty plik ZIP z nagłówkiem
    Open FileNameZip For Output As #1
    Print #1, Chr$(80) & Chr$(75) & Chr$(5) & Chr$(6) & String(18, 0)
    Close #1

    Set ShellApp = CreateObject("Shell.Application")
    ' Kopiuje pliki do skompresowanego archiwum
    For i = LBound(FileNames) To UBound(FileNames)
        ShellApp.Namespace(FileNameZip).CopyHere FileNames(i)
        ' Czekaj, dopóki pakowanie nie zostanie zakończone
        On Error Resume Next
        Do Until ShellApp.Namespace(FileNameZip).items.Count = i
            Application.Wait (Now + TimeValue("0:00:01"))
        Loop
    Next i

    If MsgBox(FileCount & " plików zostało spakowanych do pliku ZIP: " & _
        vbCrLf & FileNameZip & vbCrLf & vbCrLf & _
        "Wyświetlić plik ZIP?", vbQuestion + vbYesNo) = vbYes Then _
        Shell "Explorer.exe /e," & FileNameZip, vbNormalFocus
End Sub
```

Na rysunku 25.11 przedstawiono okno wyboru plików do spakowania utworzone za pomocą metody `GetOpenFilename` obiektu `Application` (więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 10.). Okno dialogowe pozwala użytkownikowi na zaznaczenie wielu plików w danym katalogu.



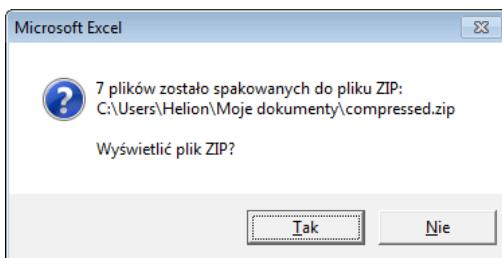
Rysunek 25.11. To okno dialogowe pozwala użytkownikowi na zaznaczenie plików wybranych do spakowania

Procedura `ZipFiles` tworzy plik o nazwie `compressed.zip` i zapisuje w nim ciąg znaków tworzący standardowy nagłówek pliku archiwum w formacie ZIP. Następnie tworzony jest obiekt `Shell.Application` i procedura wykorzystuje jego metodę `CopyHere` do skopiowania wybranych przez użytkownika plików do archiwum ZIP. W następnej sekcji kodu znajdziemy pętlę `Do Until`, która co sekundę sprawdza liczbę plików w archiwum ZIP. Taka operacja jest konieczna, ponieważ kopiowanie plików do archiwum może zająć sporo czasu, a jeżeli procedura zakończyłaby działanie przed zakończeniem kopирования plików, tworzone archiwum ZIP mogłoby być niekompletne (i prawdopodobnie uszkodzone).

Kiedy liczba plików w archiwum ZIP zgadza się z liczbą plików zaznaczonych przez użytkownika do spakowania, pętla kończy działanie i na ekranie zostaje wyświetlony komunikat, który został przedstawiony na rysunku 25.12. Naciśnięcie p do pliku przycisku *Tak* powoduje uruchomienie programu Windows Explorer, w którym zostanie wyświetlona zawartość archiwum.

Rysunek 25.12.

Użytkownik zostaje poinformowany o zakończeniu procesu tworzenia pliku ZIP



Procedura ZipFiles przedstawiona na powyższym przykładzie została maksymalnie uproszczona, aby ułatwić Czytelnikowi zrozumienie zasady jej działania. Kod procedury nie zawiera żadnych elementów sprawdzania błędów i nie jest zbyt uniwersalny. Na przykład nie ma tutaj opcji pozwalającej na wybranie nazwy i lokalizacji tworzonego archiwum ZIP, a domyślny plik *compressed.zip* jest za każdym razem po prostu nadpisywany bez żadnego ostrzeżenia.

Rozpakowywanie plików ZIP

Procedura, którą przedstawiamy w tym podrozdziale, spełnia dokładnie przeciwną funkcję do procedury omawianej w poprzednim przykładzie. Nasz program prosi użytkownika o wskazanie pliku archiwum ZIP i następnie wypakowuje pliki z archiwum i umieszcza je w katalogu *Rozpakowane* zlokalizowanym w domyślnym katalogu programu Excel.

```
Sub UnzipAFile()
    Dim ShellApp As Object
    Dim TargetFile
    Dim ZipFolder

    ' Plik docelowy i katalog roboczy
    TargetFile = Application.GetOpenFilename_
        (FileFilter:="Zip Files (*.zip), *.zip")
    If TargetFile = False Then Exit Sub

    ZipFolder = Application.DefaultFilePath & "\Rozpakowane\"

    ' Utwórz katalog roboczy
    On Error Resume Next
    RmDir ZipFolder
    MkDir ZipFolder
    On Error GoTo 0

    ' Wypakuj skompresowane pliki i umieść je w utworzonym katalogu roboczym
    Set ShellApp = CreateObject("Shell.Application")
    ShellApp.Namespace(ZipFolder).CopyHere _
        ShellApp.Namespace(TargetFile).items

    If MsgBox("Rozpakowane pliki zostały umieszczone w folderze:" & _
        vbNewLine & ZipFolder & vbNewLine & vbNewLine &_
        "Wyświetlić zawartość tego foldera?", vbQuestion + vbYesNo) = vbYes Then _
        Shell "Explorer.exe /e," & ZipFolder, vbNormalFocus
End Sub
```

Procedura UnzipAFile do pobrania nazwy archiwum ZIP wykorzystuje metodę GetOpenFilename, a następnie tworzy nowy folder i używa obiektu Shell.Application do skopiowania zawartości archiwum ZIP do utworzonego wcześniej foldera. Po zakończeniu wypakowywania plików procedura pyta użytkownika, czy wyświetlić zawartość foldera docelowego.

Działania z obiektami danych ActiveX (ADO)

ADO (ang. *ActiveX Data Objects*) jest modelem obiektów umożliwiającym uzyskanie dostępu do danych zapisanych w różnych formatach baz danych. Co bardzo istotne, metodologia ta umożliwia wykorzystanie jednego modelu obiektów dla wszystkich baz danych. Obecnie jest to preferowana metoda dostępu do danych, której nie należy mylić z *DAO* (ang. *Data Access Objects* — obiekty dostępu do danych).

W tym podrozdziale zaprezentowano prosty przykład, w którym wykorzystano model ADO w celu uzyskania danych z bazy danych Accessa.



Programowanie ADO jest zagadnieniem bardzo złożonym. Jeżeli chcesz w swoich aplikacjach Excela korzystać z zewnętrznych źródeł danych, powinieneś zaopatrzyć się w kilka książek poświęconych specjalnie temu tematowi.

W przykładzie przedstawionym poniżej następuje pobranie danych z bazy danych programu Access o nazwie *budżet.accdb*. Baza danych składa się z jednej tabeli (o nazwie Budżet) z siedmioma polami. Procedura pobiera dane rekordu, gdzie pole Pozycja zawiera tekst *Dzierżawa*, pole *Oddział* tekst *Ameryka Płn*, pole *Rok* wartość *2006*. Dane, na podstawie których są wybierane informacje, są zapisywane w obiekcie Recordset, a uzyskane wyniki przenoszone do arkusza (patrz rysunek 25.13).

	A	B	C	D	E	F	G	H	I	J	K	L
1	ID	SORT	REGION	WYDZIAŁ	KATEGORIA	POZYCJA	ROK	MIESIĄC	BUDŻET	KOSZT	RÓŻNICA	
2	58	58	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Styczeń	3898	2979	919	
3	1378	1378	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Luty	2860	3658	-798	
4	2698	2698	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Marzec	3796	4406	-610	
5	4018	4018	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Kwiecień	4039	2900	1139	
6	5338	5338	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Maj	3373	3894	-521	
7	6658	6658	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Czerwiec	4226	3706	520	
8	7978	7978	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Lipiec	2842	3345	-503	
9	9298	9298	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Sierpień	4346	2524	1822	
10	10618	10618	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Wrzesień	2938	3597	-659	
11	11938	11938	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Październik	4117	2549	1568	
12	13258	13258	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Listopad	2998	2656	342	
13	14578	14578	Północ	Księgowość	Zamówienia	Dzierżawa	2008	Grudzień	2783	2971	-188	
14	226	226	Północ	Marketing	Zamówienia	Dzierżawa	2008	Styczeń	3933	3580	353	
15	1546	1546	Północ	Marketing	Zamówienia	Dzierżawa	2008	Luty	3131	3864	-733	
16	2866	2866	Północ	Marketing	Zamówienia	Dzierżawa	2008	Marzec	3061	3327	-266	
17	4186	4186	Północ	Marketing	Zamówienia	Dzierżawa	2008	Kwiecień	3548	3408	140	
18	5506	5506	Północ	Marketing	Zamówienia	Dzierżawa	2008	Maj	4406	4341	65	
19	6826	6826	Północ	Marketing	Zamówienia	Dzierżawa	2008	Czerwiec	4476	3975	501	
20	8146	8146	Północ	Marketing	Zamówienia	Dzierżawa	2008	Lipiec	2799	3309	-510	
21	9466	9466	Północ	Marketing	Zamówienia	Dzierżawa	2008	Sierpień	2711	3003	-292	
22	10786	10786	Północ	Marketing	Zamówienia	Dzierżawa	2008	Wrzesień	2887	3904	-1017	
23	12106	12106	Północ	Marketing	Zamówienia	Dzierżawa	2008	Październik	3379	2992	387	
24	13426	13426	Północ	Marketing	Zamówienia	Dzierżawa	2008	Listopad	3749	2935	814	
25	14746	14746	Północ	Marketing	Zamówienia	Dzierżawa	2008	Grudzień	3812	4403	-591	
26	10	10	Północ	Przetwarzanie danych	Zamówienia	Dzierżawa	2008	Styczeń	3450	2631	819	
27	1330	1330	Północ	Przetwarzanie danych	Zamówienia	Dzierżawa	2008	Luty	4440	4357	83	

Rysunek 25.13. Te dane pobrano z bazy danych Accessa

```
Sub ADO_Demo()
    ' Procedura wymaga zdefiniowania odwołania do biblioteki Microsoft ActiveX Data Objects 2.x

    Dim DBFullName As String
    Dim Cnct As String, Src As String
    Dim Connection As ADODB.Connection
    Dim Recordset As ADODB.Recordset
    Dim Col As Integer
    Cells.Clear

    ' Informacje o bazie danych
    DBFullName = ThisWorkbook.Path & "\budżet.accdb"

    ' Otwarcie połączenia
    Set Connection = New ADODB.Connection
    Cnct = "Provider=Microsoft.ACE.OLEDB.12.0;" &
    Cnct = Cnct & "Data Source=" & DBFullName & ";"
    Connection.Open ConnectionString:=Cnct

    ' Utworzenie obiektu RecordSet
    Set Recordset = New ADODB.Recordset
    With Recordset
        ' Filtrowanie
        Src = "SELECT * FROM Budżet WHERE Pozycja = 'Dzierżawa' "
        Src = Src & "and Region = 'Północ'"
        Src = Src & "and Rok = '2006'"
        .Open Source:=Src, ActiveConnection:=Connection

        ' Zapisanie nazw pól
        For Col = 0 To Recordset.Fields.Count - 1
            Range("A1").Offset(0, Col).Value = _
                Recordset.Fields(Col).Name
        Next

        ' Zapisanie zestawu rekordów
        Range("A1").Offset(1, 0).CopyFromRecordset Recordset
    End With

    Set Recordset = Nothing
    Connection.Close
    Set Connection = Nothing
End Sub
```



Skoroszyt z tym przykładem (*Przykład ADO.xlsxm*) oraz bazę danych Access (*Budżet.accdb*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Dodatkowo zamieszczono tam również kolejny skoroszyt, *Przykład ADO2.xlsxm*, będący przykładem użycia zapytania ADO do przeszukiwania pliku w formacie CSV. W tym przypadku rolę bazy danych pełni duży plik o nazwie *ListaPlików.csv*.

Rozdział 26.

Operacje na składnikach

języka VBA

W tym rozdziale:

- Podstawowe informacje o środowisku IDE i jego modelu obiektowym
- Zastosowanie VBA do dodawania i usuwania modułów z projektu
- Zastosowanie VBA do generowania kodu VBA
- Zastosowanie kodu VBA do ułatwienia tworzenia formularzy *UserForm*
- Dynamiczne tworzenie formularzy *UserForm*

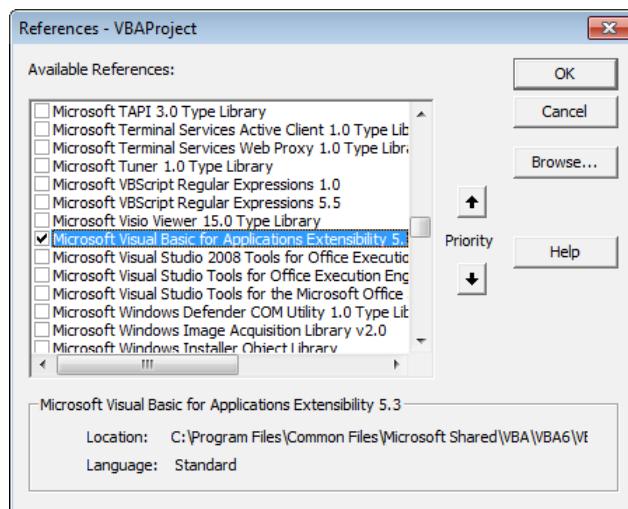
Podstawowe informacje o środowisku IDE

W tym rozdziale będziemy się zajmować bardzo interesującym zagadnieniem, które z pewnością dla wielu Czytelników będzie przydatne — tworzeniem kodu języka Visual Basic for Applications (VBA), który modyfikuje składniki projektu VBA. Środowisko IDE (ang. *Integrated Development Environment*) posiada model obiektowy, który udostępnia kluczowe elementy projektu VBA, włączając w to nawet edytor VBE (ang. *Visual Basic Editor*). Model obiektowy środowiska IDE pozwala na tworzenie kodu VBA dodającego nowe moduły do projektu, usuwającego moduły z projektu, generującego nowy kod VBA, a nawet dynamicznie tworzącego formularze *UserForm*.

IDE, czyli zintegrowane środowisko programisty (ang. *Integrated Development Environment*), jest w istocie interfejsem automatyzacji OLE (ang. *Object Linking and Embedding*) edytora *Visual Basic*, który zawiera model obiektów udostępniający kluczowe elementy projektów VBA. Po utworzeniu odwołania do tego obiektu będziesz miał pełny dostęp do wszystkich obiektów, właściwości i metod VBE, a nawet będziesz mógł deklarować nowe obiekty na bazie klas VBE.

Gdy za pomocą polecenia *Tools/References* edytora Visual Basic zdefiniujemy odwołanie do biblioteki *Microsoft Visual Basic for Application Extensibility* (patrz rysunek 26.1), uzyskamy dostęp do obiektu *VBIDE* (od tej chwili może deklarować jego obiekty składowe) oraz do wielu predefiniowanych stałych środowiska IDE. Dostęp do obiektów środowiska

Rysunek 26.1.
Tworzenie odwołania
do biblioteki Microsoft
Visual Basic for
Application Extensibility



IDE można właściwie uzyskać *bez* tworzenia odwołania, ale w takim przypadku nie jest możliwe wykorzystanie stałych w kodzie ani deklarowanie specyficznych obiektów, które odwołują się do komponentów środowiska IDE.

Po zapoznaniu się ze sposobem działania modelu obiektów środowiska IDE możesz tworzyć kod wykonujący różne operacje, takie jak na przykład:

- Dodawanie i usuwanie modułów VBA.
- Wstawianie kodu VBA.
- Tworzenie formularzy *UserForm*.
- Dodawanie formantów w formularzach *UserForm*.

Ważne informacje na temat bezpieczeństwa makr

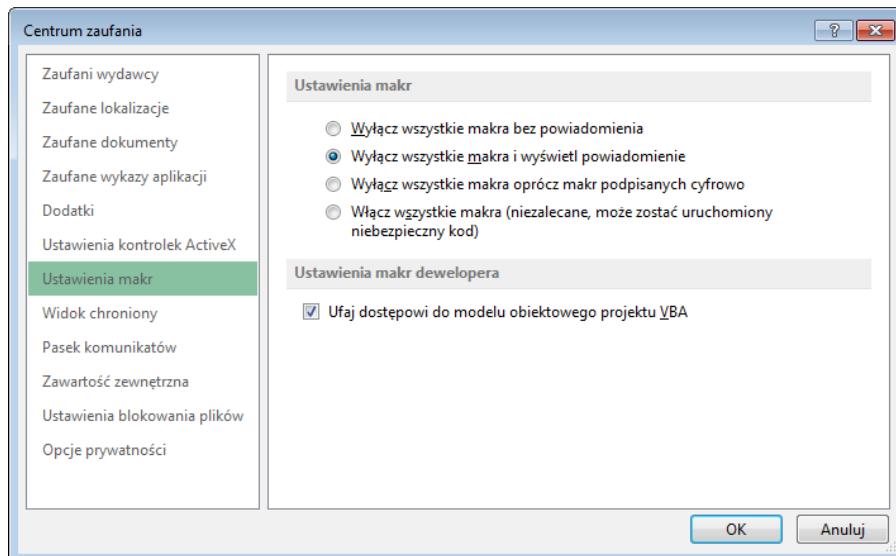
Jeżeli używasz Excela do tworzenia aplikacji, które będą wykorzystywane przez innych użytkowników, powinieneś pamiętać, że procedury opisywane w tym rozdziale mogą nie działać poprawnie. Dzieje się tak, ponieważ w celu zminimalizowania ryzyka rozprowadzania wirusów firma Microsoft (począwszy od wersji Excel 2002) znacznie utrudniła modyfikowanie komponentów projektu VBA za pomocą makr VBA, stąd próba wykonania procedur zaprezentowanych w tym rozdziale może zakończyć się wyświetleniem komunikatu o błędzie.

Wyświetlenie (badź nie) tego komunikatu zależy od ustawień odpowiednich opcji w Centrum zaufania programu Excel 2007. Aby wyświetlić lub zmienić domyślne ustawienia, powinieneś wykonać następujące polecenia:

1. Przejdz na kartę Plik i wybierz z menu polecenie Opcje.
2. W oknie dialogowym Opcje programu Excel, które pojawi się na ekranie, odszukaj i kliknij kategorię Centrum zaufania.
3. W prawej części okna odszukaj i naciśnij przycisk Ustawienia Centrum zaufania.
4. W oknie dialogowym Centrum zaufania, które pojawi się na ekranie, odszukaj i kliknij kategorię Ustawienia makr.

Zamiast tego możesz przejść na kartę *DEVELOPER* i nacisnąć przycisk *Bezpieczeństwo makr*, znajdujący się w grupie opcji *Kod*.

W sekcji *Ustawienia makr dewelopera* znajdziesz opcję *Ufaj dostępowi do modelu obiektowego projektu VBA*.



Domyślnie ta opcja jest wyłączona. Nawet jeżeli użytkownik całkowicie ufa *makrom* zapisanym w skoroszycie, to dopóki ta opcja jest wyłączona, modyfikowanie projektu VBA za pomocą makr będzie niemożliwe. Zwróć uwagę, że to ustawienie jest globalne, czyli dotyczy wszystkich skoroszytów i nie można go zmienić tylko dla określonego skoroszytu.

Korzystając z VBA, nie masz możliwości bezpośredniego sprawdzenia ustawienia tej opcji. Jedynym sposobem wykrycia tego ustawienia jest próba uzyskania dostępu do obiektu *VBProject*, a następnie sprawdzenie, czy wystąpił błąd. Przykładowy kod ilustrujący takie rozwiązanie został przedstawiony poniżej:

```
On Error Resume Next  
Set x = ActiveWorkbook.VBProject  
If Err <> 0 Then  
    MsgBox "Ustawienia zabezpieczeń w Twoim systemie uniemożliwiają uruchomienie tego makra."  
    Exit Sub  
End If
```

Na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) znajdziesz skoroszyt o nazwie *Sprawdź ustawienia zabezpieczeń.xlsxm*, który demonstruje, w jaki sposób sprawdzić, czy opcja *Ufaj dostępowi do modelu obiektowego projektu VBA* jest włączona, a jeżeli nie jest, podpowiada użytkownikowi, jak ją włączyć.

Nie wszystkie przykłady zaprezentowane w tym rozdziale przeznaczone są do uruchamiania przez zwykłych, szeregowych użytkowników. Wiele z nich zostało utworzonych w celu wspomagania tworzenia projektów przez programistów. Dla takich projektów powinieneś włączyć opcję *Ufaj dostępowi do modelu obiektowego projektu VBA*.

Model obiektowy środowiska IDE

Programowy dostęp do środowiska IDE wymaga zrozumienia jego modelu obiektowego. Obiektem, który znajduje się najwyżej w hierarchii, jest VBE (ang. *Visual Basic Environment* — środowisko Visual Basic). Podobnie jak jest w przypadku modelu obiektowego samego Excela, model środowiska IDE zawiera również inne obiekty. Uproszczoną wersję hierarchii obiektów środowiska IDE przedstawiono poniżej:

```
VBE
  VBProject
    VBComponent
      CodeModule
      Designer
      Property
      Reference
    Window
    CommandBar
```



Na potrzeby tego rozdziału zignorowano kolekcję Windows z biblioteki *Extensibility* oraz kolekcję CommandBars, które nie są zbyt przydatne dla programistów aplikacji Excela. Zamiast tego większość materiału w rozdziale poświęcono obiekowi VBProject, który jest kluczowym elementem dla dewelopera. Korzystając z tego obiektu, powinieneś jednak pamiętać o uwagach przedstawionych w ramce „Ważne informacje na temat bezpieczeństwa makr”.

Kolekcja VBProjects

Każdy otwarty skoroszyt lub dodatek jest reprezentowany przez obiekt VBProject. Aby uzyskać dostęp do obiektu VBProject ze skoroszytu, powinieneś się upewnić, że utworzyłeś odwołanie do biblioteki *Microsoft Visual Basic for Applications Extensibility Library* (patrz sekcja „Podstawowe informacje o środowisku IDE” we wcześniejszej części tego rozdziału).

Właściwość VBProject obiektu Workbook zwraca obiekt VBProject. Przykładowo wykonanie instrukcji przedstawionych poniżej spowoduje utworzenie zmiennej obiektowej reprezentującej obiekt VBProject aktywnego skoroszytu:

```
Dim VBP As VBProject
Set VBP = ActiveWorkbook.VBProject
```



Jeżeli podczas próby wykonania instrukcji Dim wystąpi błąd, powinieneś sprawdzić, czy dodałeś odwołanie do biblioteki *Microsoft Visual Basic for Applications Extensibility*.

Każdy obiekt VBProject zawiera kolekcję obiektów odpowiadających składnikom danego projektu VBA (formularze *UserForm*, moduły, moduły klas i moduły dokumentów). Nic więc dziwnego, że kolekcja tych obiektów nosi nazwę VBComponents. Obiekt VBProject zawiera także kolekcję References reprezentującą biblioteki, z których korzysta bieżący projekt.

Do kolekcji VBProjects nie możesz bezpośrednio dodawać nowych elementów — ale możesz to wykonać pośrednio, otwierając lub tworząc nowy skoroszyt w Excelu. Wykonanie takiej operacji automatycznie dodaje nowy element do kolekcji VBProjects. Analog-

gicznie nie można bezpośrednio usunąć obiektu VBProject — zamknięcie skoroszytu automatycznie usuwa obiekt VBProject z kolekcji.

Kolekcja VBComponents

Aby uzyskać dostęp do kolekcji VBComponents, należy wykorzystać właściwość `VBComponents`, podając jako argument wywołania numer indeksu bądź nazwę. Poniżej przedstawiono dwa sposoby uzyskania dostępu do składnika VBA i utworzenia zmiennej obiektowej:

```
Set VBC = ThisWorkbook.VBProject.VBComponents(1)  
Set VBC = ThisWorkbook.VBProject.VBComponents("Module1")
```

Kolekcja References

Każdy projekt VBA w Excelu zawiera wiele odwołań. Odwołania projektu można przeglądać, dodawać lub usuwać za pomocą polecenia *Tools/References* (okno dialogowe *References* zostało wcześniej przedstawione na rysunku 26.1). Każdy projekt zawiera jakieś odwołania (takie jak np. *VBA*, *Excel*, *OLE Automation* czy *Microsoft Office Object Library*). W miarę wzrastania potrzeb projektu możesz oczywiście dodawać nowe odwołania.

Operacje z odwołaniami projektu można także wykonywać za pomocą kodu VBA. Kolekcja *References* zawiera obiekty *Reference* posiadające odpowiednie właściwości i metody. Na przykład procedura, której kod przedstawiamy poniżej, wyświetla okno informacyjne z właściwościami *Name*, *Description* oraz *FullPath* dla każdego obiektu *Reference* aktywnego projektu:

```
Sub ListReferences()  
    Dim Ref As Reference  
    Msg = ""  
    For Each Ref In ActiveWorkbook.VBProject.References  
        Msg = Msg & Ref.Name & vbCrLf  
        Msg = Msg & Ref.Description & vbCrLf  
        Msg = Msg & Ref.FullPath & vbCrLf & vbCrLf  
    Next Ref  
    MsgBox Msg  
End Sub
```

Na rysunku 26.2 przedstawiono wyniki działania tej procedury dla skoroszytu z sześcioma odwołaniami.



Ponieważ w procedurze *ListReferences* została zadeklarowana zmienna obiektowa typu *Reference*, działanie procedury wymaga zdefiniowania odwołania do biblioteki *VBA Extensibility*. W przypadku zadeklarowania obiektu *Ref* jako zmiennej ogólnego typu *Object*, odwołanie do biblioteki *VBA Extensibility* nie byłoby potrzebne.

Odwołanie możesz także dodać za pomocą jednej z dwóch metod klasy *Reference*. Metoda *AddFromFile* pozwala dodać odwołanie, kiedy są znane nazwa pliku i katalog biblioteki. Jeżeli znasz *globalny identyfikator biblioteki* (ang. *GUID* — *Globally Unique Identifier*), możesz skorzystać z metody *AddFromGuid*. Więcej informacji na ten temat znajdziesz w pomocy systemowej programu Excel.

Rysunek 26.2.
*Okno dialogowe
 z informacją
 o odwołaniach projektu*



Wyświetlanie wszystkich składników projektu VBA

Procedura ShowComponents, której kod zamieszczamy poniżej, przetwarza w pętli wszystkie składniki VBA w aktywnym skoroszycie i zapisuje w arkuszu następujące informacje:

- Nazwa składnika.
- Typ składnika.
- Liczba wierszy kodu w module kodu składnika.

```
Sub ShowComponents()
    Dim VBP As VBIDE.VBProject
    Dim VBC As VBComponent
    Dim row As Long

    Set VBP = ActiveWorkbook.VBProject

    ' Zapisz nagłówki
    Cells.ClearContents
    Range("A1:C1") = Array("Nazwa", "Typ", "Liczba wierszy kodu")
    Range("A1:C1").Font.Bold = True
    row = 1
```

```

    Pętla przetwarzająca kolejne składniki VBA
    For Each VBC In VBP.VBComponents
        row = row + 1
        Nazwa składnika
        Cells(row, 1) = VBC.Name
        Typ składnika
        Select Case VBC.Type
            Case vbext_ct_StdModule
                Cells(row, 2) = "Moduł"
            Case vbext_ct_ClassModule
                Cells(row, 2) = "Moduł klas"
            Case vbext_ct_MSForm
                Cells(row, 2) = "Formularz UserForm"
            Case vbext_ct_Document
                Cells(row, 2) = "Moduł dokumentu"
        End Select
        Liczba wierszy kodu
        Cells(row, 3) = VBC.CodeModule.CountOfLines
    Next VBC
End Sub

```

Zwróć uwagę, że w naszym przykładzie do określenia typu składnika używamy wbudowanych stałych (na przykład `vbext_ct_StdModule`). Takie stałe nie są zdefiniowane, dopóki nie utworzysz odwołania do biblioteki *Microsoft Visual Basic for Applications Extensibility Library*.

Wynik działania procedury `ShowComponents` pokazano na rysunku 26.3. W tym przypadku projekt VBA zawierał sześć składników i tylko w jednym z nich był pusty moduł kodu.

Rysunek 26.3.
Wynik wykonania procedury `ShowComponents`

A	B	C	D
1 Nazwa	Typ	Liczba wierszy kodu	
2 Ten_skoroszyt	Moduł dokumentu	49	
3 Arkusz1	Moduł dokumentu	0	
4 Module1	Moduł dokumentu	157	
5 FormAbout	Formularz UserForm	133	
6 Narzędzia	Moduł dokumentu	11	
7 Zakładki	Moduł	196	
8			
9			



Skoroszyt z tym przykładem (*Wyświetl składniki VB.xls*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Zwróć uwagę, że w projekcie zostało zdefiniowane odwołanie do biblioteki *VBA Extensibility*.

Wyświetlanie wszystkich procedur VBA w arkuszu

Makro `ListProcedures`, którego kod zamieszczamy poniżej, tworzy w oknie dialogowym listę wszystkich procedur VBA aktywnego skoroszytu.

```

Sub ListProcedures()
    Dim VBP As VBIDE.VBProject
    Dim VBC As VBComponent
    Dim CM As CodeModule

```

```
Dim StartLine As Long
Dim Msg As String
Dim ProcName As String

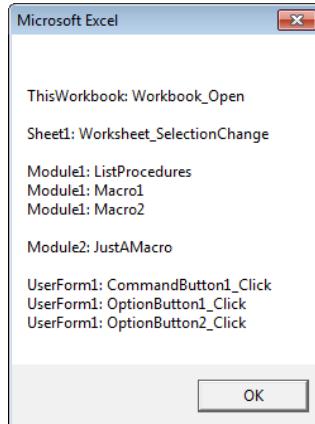
Użyj aktywnego skoroszytu
Set VBP = ActiveWorkbook.VBProject

Pętla przetwarzająca kolejne składniki VBA
For Each VBC In VBP.VBComponents
    Set CM = VBC.CodeModule
    Msg = Msg & vbCrLf
    StartLine = CM.CountOfDeclarationLines + 1
    Do Until StartLine >= CM.CountOfLines
        Msg = Msg & VBC.Name & ":" & _
            CM.ProcOfLine(StartLine, vbext_pk_Proc) & vbCrLf
        StartLine = StartLine + CM.ProcCountLines - _
            (CM.ProcOfLine(StartLine, vbext_pk_Proc). _
            vbext_pk_Proc)
    Loop
    Next VBC
    MsgBox Msg
End Sub
```

Wyniki działania procedury, przedstawione na rysunku 26.4, wskazują, że w naszym arkuszu przykładowym znajduje się dziewięć procedur.

Rysunek 26.4.

Okno dialogowe, w którym wyświetlona została lista wszystkich procedur VBA aktywnego skoroszytu



Przykładowy skoroszyt, o nazwie *Wyświetl listę wszystkich procedur.xlsm*, znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zastępowanie modułu uaktualnioną wersją

Przykład, który będzie omawiany w tym podrozdziale, ilustruje, w jaki sposób możesz zastąpić moduł VBA innym modułem VBA. Oprócz zademonstrowania sposobu działania trzech metod obiektu VBComponent (Export, Remove oraz Import) przedstawiona procedura ma również zastosowanie praktyczne. Na przykład wyobraź sobie sytuację, w której po instalacji skoroszytu przez grupę użytkowników okaże się, iż makra zawierają błędy

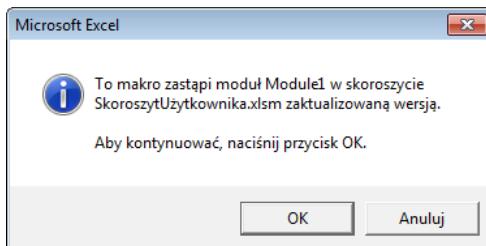
i trzeba je poprawić. Ponieważ użytkownicy pracowali już z aplikacją od pewnego czasu, zastąpienie całego skoroszytu nie jest zbyt dobrym rozwiązaniem. W takim przypadku najlepszym wyjściem będzie dostarczenie użytkownikom innego skoroszytu, zawierającego makro, które zastąpi moduł VBA zaktualizowaną wersją makra zapisanego w pliku.

Nasz przykład składa się z dwóch skoroszytów:

- *SkoroszytUżytkownika.xls* — zawiera moduł *Module1*, który trzeba zaktualizować;
- *AktualizacjaModulu.xls* — zawiera procedury VBA umożliwiające aktualizację modułu *Module1* w skoroszycie *SkoroszytUżytkownika.xls*.

Poniżej został zamieszczony kod procedury *BeginUpdate* zapisanej w skoroszycie *AktualizacjaModulu.xls*, który należy dostarczyć użytkownikom skoroszytu *SkoroszytUżytkownika.xls*. Procedura najpierw sprawdza, czy skoroszyt *SkoroszytUżytkownika.xls* został już otwarty, a potem wyświetla informację o planowanym działaniu procedury (patrz rysunek 26.5).

Rysunek 26.5.
Komunikat informujący użytkownika o planowanej aktualizacji modułu



```
Sub BeginUpdate()
    Dim FileName As String
    Dim Msg As String
    Filename = "SkoroszytUżytkownika.xls"

    ' Uaktywnienie skoroszytu
    On Error Resume Next
    Workbooks(Filename).Activate
    If Err <> 0 Then
        MsgBox Filename & " musi być otwarty!", vbCritical
        Exit Sub
    End If
    Msg = "To makro zastępuje moduł Module1 w skoroszycie SkoroszytUżytkownika.xls"
    Msg = Msg & "zaktualizowaną wersją." & vbCrLf & vbCrLf
    Msg = Msg & "Kliknij OK, aby kontynuować."
    If MsgBox(Msg, vbInformation + vbOKCancel) = vbOK Then
        Call ReplaceModule
    Else
        MsgBox "Moduł nie został zastąpiony!", vbCritical
    End If
End Sub
```

Kiedy użytkownik naciśnie przycisk *OK*, wywołana zostanie procedura *ReplaceModule*. Jej wykonanie powoduje zastąpienie istniejącego modułu *Module1* w skoroszycie *SkoroszytUżytkownika.xls* zaktualizowaną wersją zapisaną w skoroszycie *AktualizacjaModulu.xls*.

```

Sub ReplaceModule()
    Dim ModuleFile As String
    Dim VBP As VBIDE.VBProject

    ' Eksport modułu Module1 z tego skoroszytu
    ModuleFile = Application.DefaultFilePath & "\tempmodxxx.bas"
    ThisWorkbook.VBProject.VBComponents("Module1") _
        .Export ModuleFile

    ' Zastąpienie modułu Module1 w SkoroszycieUżytkownika.xlsxm
    Set VBP = Workbooks("UserBook.xlsxm").VBProject
    On Error GoTo ErrHandle
    With VBP.VBComponents
        .Remove VBP.VBComponents("Module1")
        .Import ModuleFile
    End With

    ' Usunięcie tymczasowego pliku modułu
    Kill ModuleFile
    MsgBox "Moduł został zastąpiony.", vbInformation
    Exit Sub

ErrHandle:
    ' Czy wystąpił błąd?
    MsgBox "BŁĄD. Moduł nie został zastąpiony.", _
        vbCritical
End Sub

```

Procedura przedstawiona powyżej wykonuje następujące operacje:

- Eksportuje moduł Module1 (zaktualizowany moduł) do pliku.
- Nazwa pliku została tak wybrana, aby zmniejszyć prawdopodobieństwo zastąpienia istniejącego pliku.
- Usuwa moduł Module1 (stary moduł) ze skoroszytu *SkoroszytUżytkownika.xlsxm* za pomocą metody Remove kolekcji VBComponents.
- Importuje moduł (zapisany w punkcie 1.) do skoroszytu *SkoroszytUżytkownika.xlsxm*.
- Usuwa plik zapisany w punkcie 1.
- Wyświetla informację dla użytkownika.

Do poinformowania użytkownika o powstałym błędzie wykorzystywany jest ogólny mechanizm obsługi błędów.



Skoroszyty z kodem przykładowego przedstawionego powyżej (pliki *SkoroszytUżytkownika.xlsxm* oraz *AktualizacjaModułu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Zastosowanie języka VBA do generowania kodu VBA

W tym podrozdziale przedstawimy przykład, który demonstruje, jak utworzyć kod VBA generujący inny kod VBA. Procedura AddSheetAndButton, której kod zamieszczamy poniżej, wykonuje następujące działania:

1. Wstawia nowy arkusz.
2. Umieszcza na arkuszu nowy przycisk (formant ActiveX CommandButton).
3. Ustawia pozycję, rozmiar i podpis przycisku.
4. W module kodu arkusza wstawia procedurę obsługi zdarzenia dla przycisku o nazwie CommandButton1_Click, która uaktywnia arkusz Arkusz1.

Poniżej przedstawiono kod procedury AddButtonAndCode.

```
Sub AddSheetAndButton()
    Dim NewSheet As Worksheet
    Dim NewButton As OLEObject

    ' Wstawienie nowego arkusza
    Set NewSheet = Sheets.Add

    ' Dodanie przycisku CommandButton
    Set NewButton = NewSheet.OLEObjects.Add _
        ("Forms.CommandButton.1")
    With NewButton
        .Left = 4
        .Top = 4
        .Width = 100
        .Height = 24
        .Object.Caption = "Powrót do arkusza Arkusz1"
    End With

    ' Dodanie procedury obsługi zdarzenia
    Code = "Sub CommandButton1_Click()" & vbCrLf
    Code = Code & "    On Error Resume Next" & vbCrLf
    Code = Code & "    Sheets(""Arkusz1""").Activate" & vbCrLf
    Code = Code & "    If Err <> 0 Then" & vbCrLf
    Code = Code & "        MsgBox ""Nie można uaktywnić arkusza Arkusz1.\"" " & vbCrLf
    Code = Code & "    End If" & vbCrLf
    Code = Code & "End Sub"

    With ActiveWorkbook.VBProject. _
        VBComponents(NewSheet.Name).CodeModule
        NextLine = .CountOfLines + 1
        .InsertLines NextLine, Code
    End With
End Sub
```

Na rysunku 26.6 przedstawiono arkusz oraz przycisk CommandButton dodany za pomocą procedury AddSheetAndButton.

Rysunek 26.6.

Ten arkusz wraz z przyciskiem oraz procedurą obsługi zdarzenia dodano za pomocą kodu VBA

A	B	C	D	E
1	Powrót na Arkusz1			
2				
3				
4				
5				
6				
7				
8				

Skoroszyt o nazwie *Dodaj przycisk i kod.xlsxm* znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Ciekawym elementem tej procedury jest wstawienie kodu VBA do modułu kodu nowego arkusza. Kod jest zapisany w zmiennej o nazwie *Code*, a każdą instrukcję kończy sekwencja znaków CR, LF. Metoda *InsertLines* wstawia kod do modułu kodu utworzonego arkusza.

Zmienna *NextLine* przechowuje numer ostatniego wiersza w module powiększony o wartość jeden. Dzięki temu procedura zostanie dodana na końcu modułu. Wstawienie kodu od pierwszego wiersza spowodowałoby błąd w przypadku, gdyby w systemie użytkownika ustawiono automatyczne wstawianie instrukcji *Option Explicit* do każdego modułu, ponieważ instrukcja ta musi być zawsze pierwszą instrukcją w module.

Na rysunku 26.7 przedstawiono wyświetlana w oknie kodu nową procedurę utworzoną za pomocą procedury *AddSheetAndButton*.

Rysunek 26.7.

Ta procedura obsługuje zdarzenia została wygenerowana za pomocą kodu VBA

```

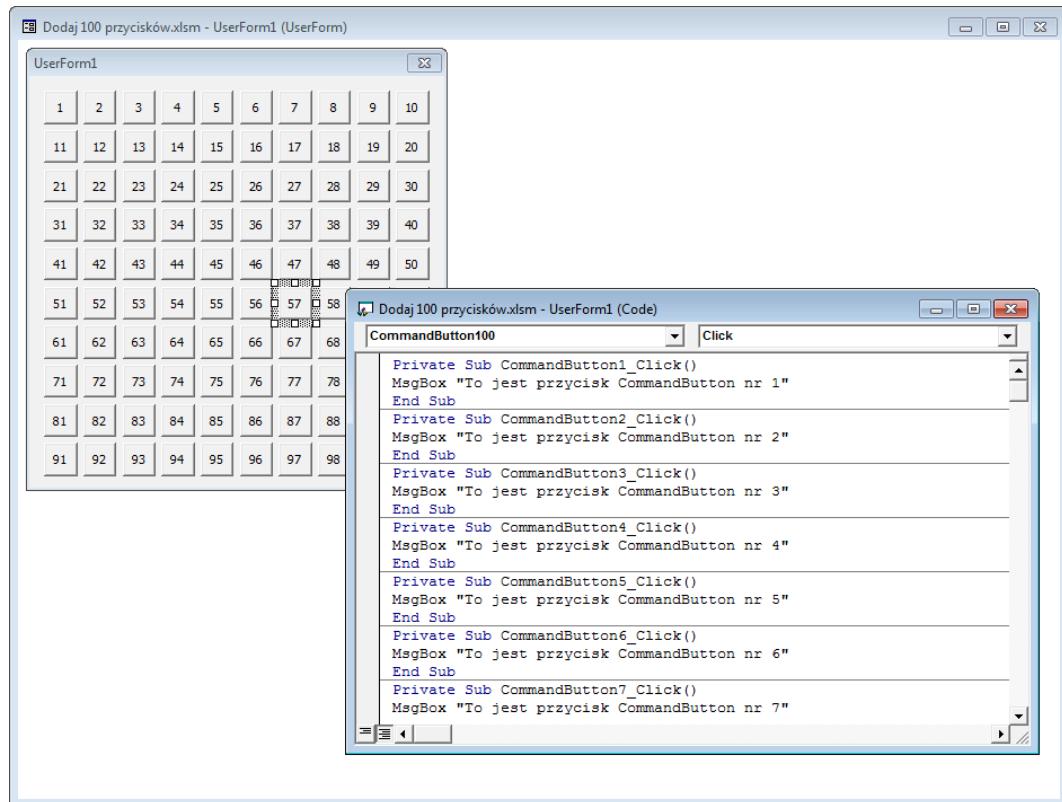
Sub CommandButton1_Click()
    On Error Resume Next
    Sheets("Arkusz1").Activate
    If Err <> 0 Then
        MsgBox "Nie można aktywować arkusza Arkusz1."
    End If
End Sub

```

Zastosowanie VBA do umieszczenia formantów na formularzu UserForm

Czytelnicy, którzy mają doświadczenie w projektowaniu formularzy *UserForm*, wiedzą, że umieszczanie, formatowanie i wyrównywanie formantów na formularzach *UserForm* może być bardzo nudnym zajęciem. Nawet w przypadku wykorzystania do maksimum poleceń formatowania edytora Visual Basic właściwe ustawienie formantów zajmuje dość dużo czasu.

Formularz *UserForm* pokazany na rysunku 26.8 zawiera 100 przycisków. Wszystkie mają identyczny rozmiar i są równomiernie rozmieszczone w oknie. Co więcej, z każdym przyciskiem związana jest procedura obsługi zdarzenia. Dodanie tych przycisków ręcznie i utworzenie dla nich procedur obsługi zdarzeń zajęłoby mnóstwo czasu, ale automatyczne dodanie przycisków za pomocą kodu VBA zajmuje około sekundy.



Rysunek 26.8. Przyciski na tym formularzu *UserForm* zostały dodane automatycznie za pomocą kodu VBA

Operacje z formularzami *UserForm* w fazie projektowania i wykonania

Operacje z formantami formularzy *UserForm* w fazie projektowania znacznie różnią się od podobnych operacji w fazie wykonania. Te ostatnie są widoczne podczas wyświetlania formularza *UserForm*, ale zmiany nie są trwałe. Można na przykład napisać kod zmieniający właściwość *Caption* formularza *UserForm* przed jego wyświetleniem. Nowy tytuł jest widoczny, kiedy okno wyświetla się na ekranie, ale po wejściu do edytora *Visual Basic* okaże się, że właściwość *Caption* nie zmieniła się. Wiele przykładów kodu wykonującego operacje z formularzami *UserForm* i ich formantami w fazie wykonania zaprezentowano w części IV naszej książki.

Z drugiej strony, operacje wykonywane w fazie projektowania są trwałe — efekt jest taki sam, jakby zmiany były wykonane ręcznie za pomocą narzędzi w edytorze Visual Basic. Zwykle przeprowadza się je, aby zautomatyzować żmudne czynności podczas projektowania formularzy *UserForm*. Aby wykonać operację w fazie projektowania, należy wykorzystać obiekt Designer.

W celu zademonstrowania różnic pomiędzy obydwooma rodzajami operacji opracowalem dwie proste procedury, które umieszczają formant *CommandButton* na formularzu *UserForm*. Pierwsza dodaje przycisk w fazie wykonania, natomiast druga w fazie projektowania.

Zamieszczona poniżej procedura *RunTimeButton* jest bardzo prosta. Jej uruchomienie z ogólnego modułu (w odróżnieniu od modułu formularza *UserForm*) powoduje dodanie przycisku, modyfikację jego kilku właściwości i wyświetlenie formularza. Przycisk wyświetla się na formularzu w czasie działania programu, ale kiedy otworzymy formularz *UserForm* w edytorze Visual Basic, okaże się, że przycisku tam nie ma.

```
Sub RunTimeButton()
    ' Dodanie przycisku w fazie wykonywania
    Dim Butn As CommandButton
    Set Butn = UserForm1.Controls.Add("Forms.CommandButton.1")
    With Butn
        .Caption = "Dodany w fazie wykonywania"
        .Width = 100
        .Top = 10
    End With
    UserForm1.Show
End Sub
```

Poniżej zamieszczono procedurę *DesignTimeButton*, która tym różni się od poprzedniej, że wykorzystano w niej obiekt *Designer* będący składową obiektu *VBComponent*. Mówiąc dokładniej, do dodania przycisku służy metoda *Add*. Dzięki wykorzystaniu obiektu *Designer* otrzymujemy dokładnie taki sam efekt, jaki uzyskalibyśmy, gdyby polecenia edytora Visual Basic zostały wybrane ręcznie.

```
Sub DesignTimeButton()
    ' Dodanie przycisku w fazie projektowania
    Dim Butn As CommandButton
    Set Butn = ThisWorkbook.VBProject.VBComponents("UserForm1") _
        .Designer.Controls.Add("Forms.CommandButton.1")
    With Butn
        .Caption = "Dodany w fazie projektowania"
        .Width = 120
        .Top = 40
    End With
    VBA.UserForms.Add("UserForm1").Show
End Sub
```

Dodawanie 100 przycisków *CommandButton* w fazie projektowania

Przykład z tego punktu demonstruje sposób wykorzystania obiektu *Designer* w celu ułatwienia projektowania formularzy *UserForm*. W tym przypadku kod dodaje 100 przycisków (idealnie rozmieszczonego i wyrównanych), ustawia tytuł dla każdego przycisku

oraz tworzy 100 procedur obsługi zdarzeń (po jednej dla każdego przycisku). Pełny kod procedury Add100Buttons przedstawiono poniżej:

```
Sub Add100Buttons()
    Dim UFvbc As VBComponent
    Dim CMod As CodeModule
    Dim ctl As Control
    Dim cb As CommandButton
    Dim n As Long, c As Long, r As Long
    Dim code As String

    Set UFvbc = ThisWorkbook.VBProject.VBComponents("UserForm1")

    ' Usunięcie wszystkich formantów, jeżeli jakieś istnieją
    For Each ctl In UFvbc.Designer.Controls
        UFvbc.Designer.Controls.Remove ctl.Name
    Next ctl

    ' Usunięcie całego kodu VBA
    UFvbc.CodeModule.DeleteLines 1, UFvbc.CodeModule.CountOfLines

    ' Dodanie 100 nowych obiektów CommandButtons
    n = 1
    For r = 1 To 10
        For c = 1 To 10
            Set cb = UFvbc.Designer. _
                Controls.Add("Forms.CommandButton.1")
            With cb
                .Width = 22
                .Height = 22
                .Left = (c * 26) - 16
                .Top = (r * 26) - 16
                .Caption = n
            End With

            ' Dodanie kodu procedury obsługi zdarzeń
            With UFvbc.CodeModule
                code = ""
                code = code & "Private Sub CommandButton" & n & "_Click" & vbCrLf
                code = code & "MsgBox ""To jest przycisk CommandButton nr " & n & _
                    """ & vbCrLf
                code = code & "End Sub"
                .InsertLines .CountOfLines + 1, code
            End With
            n = n + 1
        Next c
    Next r
End Sub
```

Skoroszyt z tym przykładem (*Dodaj 100 przycisków.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Procedura Add100Buttons wymaga zdefiniowania formularza *UserForm* o nazwie *UserForm1*. Należy nieco powiększyć rozmiar okna formularza *UserForm* w porównaniu z domyślnym, tak aby zmieściły się wszystkie przyciski. Procedura rozpoczęta działania od

usunięcia wszystkich formantów z formularza za pomocą metody Remove kolekcji Controls, a następnie usunięcia całego kodu z modułu kodu za pomocą metody DeleteLines obiektu CodeModule. Następnie w dwóch pętlach For ... Next są dodawane przyciski i bardzo proste procedury obsługi zdarzeń. Poniżej zamieszczono przykład kodu takiej procedury dla przycisku CommandButton1:

```
Private Sub CommandButton1_Click()
    MsgBox "To jest przycisk CommandButton nr 1"
End Sub
```

Aby wyświetlić formularz po dodaniu formantów w fazie projektowania, należy bezpośrednio przed instrukcją End Sub wprowadzić poniższą instrukcję:

```
VBA.UserForms.Add("UserForm1").Show
```

Zrozumienie sposobu wyświetlania formularza *UserForm* zajęło mi sporo czasu. Gdy formularz *UserForm* zawierający 100 przycisków jest generowany, w rzeczywistości istnieje w pamięci VBA, ale nie jest jeszcze oficjalną częścią projektu. Należy zatem wykorzystać metodę Add w celu formalnego dołączenia okna *UserForm1* do kolekcji UserForms. Wartość zwracana przez tę metodę jest odwołaniem do formularza. Właśnie dzięki temu wywołanie metody Show można umieścić na końcu metody Add. Trzeba jednak pamiętać, aby przed użyciem formularza *UserForm* dołączyć go do kolekcji UserForms.

Programowe tworzenie formularzy *UserForm*

Ostatnim zagadnieniem przedstawionym w tym rozdziale jest zastosowanie kodu VBA do tworzenia formularzy *UserForm* w fazie wykonywania. Zaprezentujemy tutaj dwa przykłady — jeden prosty, drugi dosyć złożony.

Prosty przykład formularza *UserForm*

Program, który zaprezentujemy za chwilę ilustruje kilka istotnych zagadnień. Procedura MakeForm realizuje następujące zadania:

1. Za pomocą metody Add kolekcji VBComponents tworzy w aktywnym skoroszycie tymczasowy formularz *UserForm*.
2. Za pomocą obiektu Designer dodaje przycisk poleceń (obiekt CommandButton).
3. Dodaje procedurę obsługi zdarzenia w module kodu formularza *UserForm* (CommandButton1_Click).

Wykonanie tej procedury powoduje wyświetlenie okna informacyjnego i usunięcie formularza z pamięci.

4. Wyświetla formularz *UserForm*.
5. Usuwa formularz *UserForm*.

Finalnym wynikiem działania procedury jest formularz *UserForm*, który jest tworzony „w locie”, wykorzystywany i następnie usuwany. Ten przykład oraz przykład z kolejnego

podrozdziału nieco zacierają teoretyczną linię podziału pomiędzy modyfikowaniem formularzy w fazie projektowania i fazie wykonania. Nasz formularz jest tworzony co prawda przy użyciu technik typowych dla fazy projektowania, ale wszystko dzieje się w czasie działania procedury (w fazie wykonania).

Poniżej zamieszczamy pełny kod procedury MakeForm:

```
Sub MakeForm()
    Dim TempForm As Object
    Dim NewButton As MSForms.CommandButton
    Dim Line As Integer

    Application.VBE.MainWindow.Visible = False

    ' Utworzenie formularza UserForm
    Set TempForm = ThisWorkbook.VBProject. _
        VBComponents.Add(3) ' stała vbext_ct_MSForm
    With TempForm
        .Properties("Caption") = "Formularz tymczasowy"
        .Properties("Width") = 200
        .Properties("Height") = 100
    End With

    ' Dodanie przycisku CommandButton
    Set NewButton = TempForm.Designer.Controls. _
        .Add("Forms.CommandButton.1")
    With NewButton
        .Caption = "Naciśnij mnie"
        .Left = 60
        .Top = 40
    End With

    ' Dodanie procedury obsługi zdarzenia dla przycisku
    With TempForm.CodeModule
        Line = .CountOfLines
        .InsertLines Line + 1, "Sub CommandButton1_Click()"
        .InsertLines Line + 2, "    MsgBox ""Witaj!"""
        .InsertLines Line + 3, "    Unload Me"
        .InsertLines Line + 4, "End Sub"
    End With

    ' Wyświetlenie formularza
    VBA.UserForms.Add(TempForm.Name).Show

    ' Usunięcie formularza
    ThisWorkbook.VBProject.VBComponents.Remove TempForm
End Sub
```



W sieci

Skoroszyt z tym przykładem (*Dodaj formularz UserForm.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Procedura MakeForm tworzy i wyświetla formularz *UserForm* przedstawiony na rysunku 26.9.

Rysunek 26.9.
 Ten formularz UserForm
 oraz związany z nim
 kod został utworzony
 dynamicznie („w locie”)



Skoroszyt zawierający procedurę MakeForm nie wymaga definiowania odwołania do biblioteki VBA Extensibility Library, ponieważ zmienna TempForm została zadeklarowana jako ogólny typ Object (a nie jako obiekt ObjectVBComponent). Dodatkowo procedura nie wykorzystuje żadnych wbudowanych stałych.

Zauważ, że jedna z pierwszych instrukcji ukrywa okno Visual Basic poprzez ustawienie właściwości Visible na wartość False. Wykonanie tej instrukcji eliminuje miganie ekranu, które może wystąpić podczas generowania formularza i kodu.

Użyteczny (ale już nie tak prosty) przykład dynamicznego formularza UserForm

Przykład zaprezentowany poniżej jest zarówno pouczający, jak i praktyczny. Zaprezentowano w nim funkcję GetOption, która wyświetla formularz UserForm. Na formularzu znajduje się kilka przycisków opcji, których tytuły są podawane jako argumenty funkcji. Funkcja zwraca wartość reprezentującą przycisk opcji, który został wybrany przez użytkownika.



Skoroszyt z tym przykładem (Funkcja GetOption.xlsm) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Pełny kod funkcji GetOption został zamieszczony poniżej:

```
Function GetOption(OpArray, Default, Title)
  Dim TempForm As Object
  Dim NewOptionButton As Msforms.OptionButton
  Dim NewCommandButton1 As Msforms.CommandButton
  Dim NewCommandButton2 As Msforms.CommandButton
  Dim i As Integer, TopPos As Integer
  Dim MaxWidth As Long
  Dim Code As String

  ' Ukrycie okna VBE w celu zapobieżenia miganiu ekranu
  Application.VBE.MainWindow.Visible = False

  ' Utworzenie formularza UserForm
  Set TempForm =
    ThisWorkbook.VBProject.VBComponents.Add(3)
  TempForm.Properties("Width") = 800

  ' Dodanie przycisków opcji
  TopPos = 4
  MaxWidth = 0 ' Zapamiętuje szerokość największego przycisku
  For i = LBound(OpArray) To UBound(OpArray)
```

```
Set NewOptionButton = TempForm.Designer.Controls. _
    Add("forms.OptionButton.1")
With NewOptionButton
    .Width = 800
    .Caption = OpArray(i)
    .Height = 15
    .Accelerator = Left(.Caption, 1)
    .Left = 8
    .Top = TopPos
    .Tag = i
    .AutoSize = True
    If Default = i Then .Value = True
    If .Width > MaxWidth Then MaxWidth = .Width
End With
TopPos = TopPos + 15
Next i

' Dodanie przycisku Anuluj
Set NewCommandButton1 = TempForm.Designer.Controls. _
    Add("forms.CommandButton.1")
With NewCommandButton1
    .Caption = "Anuluj"
    .Cancel = True
    .Height = 18
    .Width = 44
    .Left = MaxWidth + 12
    .Top = 6
End With

' Dodanie przycisku OK
Set NewCommandButton2 = TempForm.Designer.Controls. _
    Add("forms.CommandButton.1")
With NewCommandButton2
    .Caption = "OK"
    .Default = True
    .Height = 18
    .Width = 44
    .Left = MaxWidth + 12
    .Top = 28
End With

' Dodanie procedur obsługi zdarzeń dla przycisków
Code = ""
Code = Code & "Sub CommandButton1_Click()" & vbCrLf
Code = Code & "    GETOPTION_RET_VAL=False" & vbCrLf
Code = Code & "    Unload Me" & vbCrLf
Code = Code & "End Sub" & vbCrLf
Code = Code & "Sub CommandButton2_Click()" & vbCrLf
Code = Code & "    Dim ctl" & vbCrLf
Code = Code & "    GETOPTION_RET_VAL = False" & vbCrLf
Code = Code & "    For Each ctl In Me.Controls" & vbCrLf
Code = Code & "        If TypeName(ctl) = ""OptionButton""" - & " Then" & vbCrLf
Code = Code & "            If ctl Then GETOPTION_RET_VAL = " - & "ctl.Tag" & vbCrLf
Code = Code & "        End If" & vbCrLf
Code = Code & "    Next ctl" & vbCrLf
```

```
Code = Code & " Unload Me" & vbCrLf
Code = Code & "End Sub"

With TempForm.CodeModule
    .InsertLines .CountOfLines + 1, Code
End With

' Dostosowanie formularza
With TempForm
    .Properties("Caption") = Title
    .Properties("Width") = NewCommandButton1.Left + _
        NewCommandButton1.Width + 10
    If .Properties("Width") < 160 Then
        .Properties("Width") = 160
        NewCommandButton1.Left = 106
        NewCommandButton2.Left = 106
    End If
    .Properties("Height") = TopPos + 24
End With

' Wyświetlenie formularza
VBA.UserForms.Add(TempForm.Name).Show

' Usunięcie formularza
ThisWorkbook.VBProject.VBComponents.Remove VBComponent:=TempForm

' Przekazanie wybranej opcji do procedury wywołującej
GetOption = GETOPTION_RET_VAL
End Function
```

Zważywszy na liczbę działań wykonywanych „w tle”, funkcja GetOption działa stosunkowo szybko. Na moim komputerze formularz wyświetla się niemal natychmiast. Po wykonaniu zadania formularz *UserForm* jest usuwany.

Zastosowanie funkcji GetOption

Funkcja GetOption pobiera trzy argumenty:

- OpArray — tablica łańcuchów znaków zawierająca elementy, które mają być wyświetlane jako przyciski opcji;
- Default — liczba całkowita określająca domyślny przycisk opcji, wybierany w momencie wyświetlenia formularza *UserForm* (w przypadku wartości 0 nie jest wybierany żaden przycisk);
- Title — tekst wyświetlany w pasku tytułu formularza *UserForm*.

Jak działa funkcja GetOption?

Funkcja GetOption wykonuje następujące działania:

1. Ukrywa okno edytora Visual Basic w celu niedopuszczenia do migotania ekranu w czasie tworzenia formularza *UserForm* oraz dodawania kodów.

2. Tworzy formularz *UserForm* i przypisuje go do zmiennej obiektowej o nazwie *TempForm*.

3. Dodaje przyciski opcji za pomocą tablicy przekazywanej do funkcji poprzez argument *OpArray*.

Do zapisania numeru indeksu wykorzystano właściwość *Tag* kontrolki.

Ustawienie *Tag* wybranej opcji jest wartością, którą ostatecznie zwraca funkcja.

4. Dodaje dwa przyciski poleceń (*CommandButton*): *OK* oraz *Anuluj*.

5. Tworzy procedury obsługi zdarzenia dla każdego z przycisków poleceń.

6. Dostosowuje pozycje przycisków poleceń oraz rozmiar formularza *UserForm*.

7. Wyświetla formularz *UserForm*.

Naciśnięcie przycisku *OK* powoduje wykonanie procedury *CommandButton1_Click*.

Procedura sprawdza, który przycisk opcji został wybrany, oraz przypisuje odpowiednią wartość do zmiennej *GETOPTION_RET_VAL* (zmienna publiczna).

8. Usuwa formularz *UserForm* po jego zamknięciu.

9. Zwraca wartość zmiennej *GETOPTION_RET_VAL* jako wynik funkcji.



Istotną zaletą tworzenia formularzy *UserForm* „w locie” jest samowystarczalność funkcji — funkcja jest zapisana w jednym module i nie wymaga nawet odwołania do biblioteki *VBA Extensibility Library*. Dzięki temu można wyeksportować moduł (o nazwie *modOptionsForm*), a następnie zaimportować go do dowolnego skoroszytu, co pozwoli uzyskać dostęp do funkcji *GetOption*.

Sposób użycia funkcji *GetOption* zaprezentowano poniżej. W tym przypadku w formularzu *UserForm* wyświetla się pięć opcji (zapisanych w tablicy *Ops*).

```
Sub TestGetOption()
    Dim Ops(1 To 5)
    Dim UserOption
    Ops(1) = "Północ"
    Ops(2) = "Południe"
    Ops(3) = "Zachód"
    Ops(4) = "Wschód"
    Ops(5) = "Wszystkie regiony"
    UserOption = GetOption(Ops, 5, "Wybierz region")
    MsgBox Ops(UserOption)
End Sub
```

Zmienna *UserOption* zawiera numer indeksu opcji wybranej przez użytkownika. Jeżeli użytkownik naciśnie przycisk *Anuluj* (lub naciśnie klawisz *Escape*), zmienna *UserOption* zostanie ustawiona na wartość *False*.

Zwróć uwagę, że właściwość *Accelerator* jest ustawiana na pierwszy znak nazwy danej opcji, dzięki czemu użytkownik może do wybrania danej opcji użyć kombinacji klawiszy *Alt+<litera>*. Tworząc kod procedury, nawet nie próbowałem unikać zdublowanych klawiszy skrótu, dlatego aby wybrać określone opcje, czasami będziesz musiał kilkakrotnie nacisnąć klawisz skrótu.

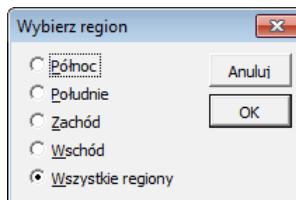
Formularz *UserForm* wygenerowany przez naszą funkcję pokazano na rysunku 26.10.

Rysunek 26.10.

Ten formularz UserForm został wygenerowany za pomocą funkcji GetOption



Uwaga



Formularz UserForm dostosowuje swój rozmiar do liczby elementów tablicy przekazanej jako argument funkcji. Teoretycznie funkcja GetOption może przyjąć tablicę dowolnych rozmiarów, jednak w praktyce należy ograniczyć liczbę opcji, tak aby utrzymać rozmiar okna formularza UserForm na rozsądny poziomie. Na rysunku 26.11 przedstawiono wygląd formularza w sytuacji, kiedy poszczególne opcje są bardziej rozbudowane.

Rysunek 26.11.

Rozmiary formularza UserForm są automatycznie dopasowywane tak, aby pomieścić odpowiednią liczbę opcji oraz ich opisy

**Kod obsługi zdarzeń funkcji GetOption**

Poniżej zaprezentowano procedury obsługi zdarzeń dla dwóch przycisków poleceń. Jest to kod generowany wewnątrz funkcji GetOption i umieszczony w module kodu tymczasowego formularza UserForm.

```
Sub CommandButton1_Click()
    GETOPTION_RET_VAL = False
    Unload Me
End Sub

Sub CommandButton2_Click()
    Dim ctl
    GETOPTION_RET_VAL = False
    For Each ctl In Me.Controls
        If TypeName(ctl) = "OptionButton" Then
            If ctl Then GETOPTION_RET_VAL = ctl.Tag
        End If
    Next ctl
    Unload Me
End Sub
```



Uwaga

Ponieważ formularz UserForm jest usuwany po wykorzystaniu, nie można obejrzeć go w edytorze Visual Basic. Aby obejrzeć okno UserForm w edytorze Visual Basic, należy przekształcić poniższą instrukcję na komentarz, wpisując apostrof na początku wiersza:

```
ThisWorkbook.VBProject.VBComponents.Remove _
VBComponent:=Temp Form
```

Rozdział 27.

Moduły klas

W tym rozdziale:

- Wprowadzenie do modułów klas
- Typowe zastosowania modułów klas
- Przykłady ilustrujące kluczowe zagadnienia związane z modułami klas

Czym jest moduł klasy?

Dla wielu programistów VBA pojęcie „moduły klas” nadal brzmi tajemniczo, mimo że mechanizm ten został wprowadzony w języku Visual Basic już ładnych kilkanaście lat temu. W tym rozdziale znajdziesz szereg przykładów, dzięki którym ten potężny i bardzo użyteczny mechanizm stanie się nieco mniej tajemniczy.

Moduł klasy jest specjalnym typem modułu VBA, który możesz umieścić w projekcie VBA. W uproszczeniu mówiąc, moduł klasy umożliwia programiście (czyli Tobie) tworzenie nowych klas obiektów. Jak wiadomo, programowanie w Excelu sprowadza się do wykonywania działań z obiektami. Moduły klas umożliwiają tworzenie nowych obiektów wraz z odpowiadającymi im właściwościami, metodami i zdarzeniami.



Niektóre przykłady w poprzednich rozdziałach wykorzystywały już moduły klas (patrz rozdziały 13., 16. i 17.).

W tym momencie możesz zapytać: Czy naprawdę muszę tworzyć nowe obiekty? Odpowiedź brzmi: *Nie musisz*, ale czasami warto to zrobić, aby uzyskać wymierne korzyści. W wielu przypadkach moduły klas zastępują istniejące funkcje lub procedury, z tym że są wygodniejsze i łatwiej się nimi zarządza. Innym razem zdefiniowanie klasy jest jedynym sposobem wykonania zadania.

Poniżej przedstawiamy listę typowych zastosowań modułów klas:

- **W celu obsługi zdarzeń związanych z wbudowanymi wykresami** (przykłady znajdziesz w rozdziale 16.).
- **W celu monitorowania zdarzeń poziomu aplikacji**, takich jak na przykład uaktywnienie arkusza (przykłady znajdziesz w rozdziale 17.).

- **W celu hermetyzacji funkcji interfejsu Windows API, co ułatwia stosowanie tych funkcji w kodzie** — możesz na przykład utworzyć klasę, która ułatwia wykrywanie lub ustawianie stanu klawiszy *NumLock* lub *CapsLock*, albo klasę, która ułatwia dostęp do rejestru Windows.
- **W celu umożliwienia wykonywania tej samej procedury przez wiele obiektów w formularzu *UserForm*** — w przykładzie z rozdziału 13. zademonstrowano sposób użycia modułu klasy w celu przypisania wielu przyciskom polecen jednej procedury obsługi zdarzenia *Click*.
- **W celu utworzenia komponentów wielokrotnego użytku, które można zaimportować do innych projektów** — moduł klasy ogólnego przeznaczenia można zaimportować do innych projektów.

Przykład: utworzenie klasy *NumLock*

W tym podrozdziale zaprezentuję krok po kroku, jak stworzyć przydatny, choć prosty moduł klasy. W module tym utworzono klasę *NumLock* zawierającą jedną właściwość (*Value*) oraz jedną metodę (*Toggle*).

Wykrycie zmiany stanu klawisza *NumLock* wymaga zastosowania kilku funkcji interfejsu Windows API, a konkretne procedury różnią się w zależności od wersji systemu Windows. Krótko mówiąc, jest to zadanie dość skomplikowane, a moduł klasy ma je uprościć. Wszystkie deklaracje API i kod umieszczono w module klasy (a nie w zwykłych modułach VBA). Korzyści? Praca z kodem stanie się łatwiejsza, a nowy moduł klasy będzie można wykorzystać w innych projektach.

Gdy moduł klasy zostanie utworzony, za pomocą kodu VBA, który wyświetla właściwość *Value* obiektu, można określić bieżący stan klawisza *NumLock*:

```
MsgBox NumLock.Value
```

Można też zmienić stan klawisza *NumLock* poprzez zmianę właściwości *Value*. Na przykład wykonanie poniższej instrukcji spowoduje włączenie klawisza *NumLock*:

```
NumLock.Value = True
```

Można również zdefiniować kod, który przełącza stan klawisza *NumLock*, wykorzystując metodę *Toggle*:

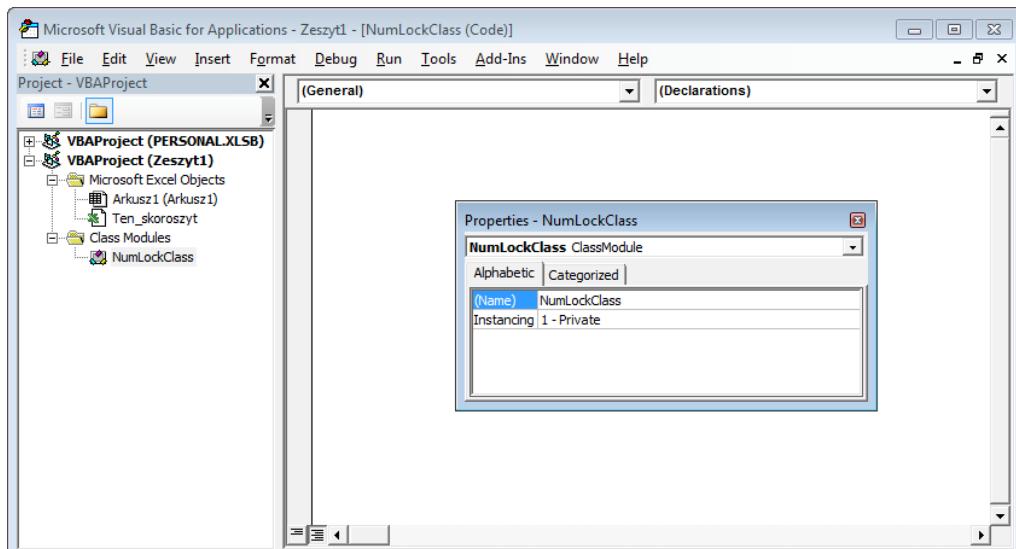
```
NumLock.Toggle
```

Pamiętaj, że moduł klasy zawiera kod, który *definiuje* obiekt wraz z jego właściwościami i metodami, dzięki czemu później można utworzyć egzemplarz tego obiektu w module kodu VBA i wykonywać działania z właściwościami i metodami obiektu.

Aby lepiej zrozumieć proces tworzenia modułu klasy, przeanalizujemy instrukcje zaprezentowane w kolejnych punktach. Rozpoczniemy od utworzenia pustego skoroszytu.

Wstawianie modułu klasy

Po uaktywnieniu edytora Visual Basic powinieneś wybrać polecenie *Insert/Class Module*. Do projektu zostanie dodany pusty moduł klasy o nazwie *Class1*. Jeżeli nie wyświetla się okno *Properties*, należy wcisnąć F4. Następnie można zmienić nazwę modułu klasy na *NumLockClass* (rysunek 27.1).



Rysunek 27.1. Pusty moduł klasy o nazwie *NumLockClass*

Dodawanie kodu VBA do modułu klasy

Teraz utworzymy kod dla właściwości *Value*. W celu wykrycia lub modyfikacji stanu klawisza *NumLock* należy w module klasy umieścić deklaracje funkcji interfejsu Windows API służących do wykrywania i ustawiania klawisza *NumLock*. Odpowiedni kod deklaracji zamieszczamy poniżej.



Kod VBA zaprezentowany w tym rozdziale utworzono na podstawie przykładu zamieszczonego na witrynie WWW firmy Microsoft. Przedstawiony kod działa tylko i wyłącznie z Exceliem 2010 i wersjami nowszymi. Kod w skoroszycie zamieszczonym na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>) działa również z poprzednimi wersjami Excela.

```
' Deklaracja typu
Private Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type
```

```

' Deklaracje API
Private Declare PtrSafe Function GetVersionEx Lib "Kernel32" _
    Alias "GetVersionExA" _
        (lpVersionInformation As OSVERSIONINFO) As Long

Private Declare PtrSafe Sub keybd_event Lib "user32" _
    (ByVal bVK As Byte, _
    ByVal bScan As Byte, _
    ByVal dwFlags As Long, ByVal dwExtraInfo As Long)

Private Declare PtrSafe Function GetKeyboardState Lib "user32" _
    (pbKeyState As Byte) As Long

Private Declare PtrSafe Function SetKeyboardState Lib "user32" _
    (lppbKeyState As Byte) As Long

' Deklaracje stałych
Const VK_NUMLOCK = &H90
Const VK_SCROLL = &H91
Const VK_CAPITAL = &H14
Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2

```

Następnie potrzebna jest procedura pobierająca bieżący stan klawisza *NumLock*. Procedurę tę nazwalem właściwością *Value* obiektu. Dla właściwości można zdefiniować dowolną nazwę, ale nazwa *Value* wydaje się odpowiednia. W celu uzyskania stanu klawisza *NumLock* należy wprowadzić następującą procedurę *Property Get*:

```

Property Get Value() As Boolean
    ' Pobranie bieżącego stanu
    Dim keys(0 To 255) As Byte
    GetKeyboardState keys(0)
    Value = keys(VK_NUMLOCK)
End Property

```



Szczegółowe informacje o procedurach *Property* można znaleźć w dalszej części rozdziału, w punkcie „Programowanie właściwości obiektów”.

Powyższa procedura, w której do sprawdzenia bieżącego stanu klawisza *NumLock* wykorzystano funkcję interfejsu Windows API *GetKeyboardState*, jest wywoływana za każdym razem, kiedy w kodzie VBA nastąpi odwołanie do właściwości *Value* obiektu. Przykładowo po utworzeniu obiektu wykonanie takiego polecenia VBA spowoduje wywołanie procedury *PropertyGet*:

```
MsgBox NumLock.Value
```

Teraz będzie nam potrzebna procedura pozwalająca na ustawienie odpowiedniego stanu klawisza *NumLock* (włączony lub wyłączony). Możemy to zrobić za pomocą procedury *Property Let*, której kod został zamieszczony poniżej:

```

Property Let Value(boolVal As Boolean)
    Dim o As OSVERSIONINFO
    Dim keys(0 To 255) As Byte
    o.dwOSVersionInfoSize = Len(o)
    GetVersionEx o
    GetKeyboardState keys(0)

```

```

    ' Czy bieżący stan jest odpowiedni?
    If boolVal = True And keys(VK_NUMLOCK) = 1 Then Exit Property
    If boolVal = False And keys(VK_NUMLOCK) = 0 Then Exit Property
    Przelączanie stanu
        Symulacja wcisnięcia klawisza
        keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
        Symulacja zwolnienia klawisza
        keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or _
            KEYEVENTF_KEYUP, 0
    End Property

```

Procedura `Property Let` pobiera jeden argument: `True` lub `False`. Polecenie VBA, takie jak przedstawione poniżej, ustawia właściwość `Value` obiektu `NumLock` na wartość `True` za pomocą wywołania procedury `PropertyLet`:

```
NumLock.Value = True
```

Na koniec zdefiniujemy procedurę przełączającą stan klawisza `NumLock`.

```

Sub Toggle()
    Przelączanie stanu
    Dim o As OSVERSIONINFO
    o.dwOSVersionInfoSize = Len(o)
    GetVersionEx o
    Dim keys(0 To 255) As Byte
    GetKeyboardState keys()
    Symulacja wcisnięcia klawisza
    keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or 0, 0
    Symulacja zwolnienia klawisza
    keybd_event VK_NUMLOCK, &H45, KEYEVENTF_EXTENDEDKEY Or _
        KEYEVENTF_KEYUP, 0
End Sub

```

Zauważ, że metoda `Toggle` jest standardową procedurą `Sub` (a nie procedurą `Property Let` lub `Property Get`). Zaprezentowana poniżej instrukcja VBA przełącza stan klawisza `NumLock` poprzez wykonanie procedury `Toggle`:

```
NumLock.Toggle
```

Wykorzystanie klasy `NumLock`

Przed użyciem modułu klasy `NumLock` musisz utworzyć egzemplarz obiektu. Czynność tę można wykonać za pomocą poniższej instrukcji, którą powinieneś umieścić w zwykłym module VBA, a nie w module klasy:

```
Dim NumLock As New NumLockClass
```

Nazwą typu obiektu (tzn. nazwą modułu klasy) jest `NumLockClass`. Nazwa zmiennej obiektowej może być dowolna, ale użycie nazwy `NumLock` w tym przypadku wydaje się być logicznym rozwiązaniem.

Poniższa procedura ustawia właściwość `Value` obiektu `NumLock` na wartość `True`. Wykonanie procedury powoduje włączenie klawisza `NumLock`:

```
Sub NumLockOn()
    Dim NumLock As New NumLockClass
    NumLock.Value = True
End Sub
```

Kolejna procedura wyświetla komunikat z informacją o bieżącym stanie klawisza *NumLock* (True oznacza klawisz włączony, False — wyłączony):

```
Sub GetNumLockState()
    Dim NumLock As New NumLockClass
    MsgBox NumLock.Value
End Sub
```

Poniższa procedura przełącza stan klawisza *NumLock*:

```
Sub ToggleNumLock()
    Dim NumLock As New NumLockClass
    NumLock.Toggle
End Sub
```

Można również przełączyć klawisz *NumLock* w inny sposób, nie używając metody *Toggle*:

```
Sub ToggleNumLock2()
    Dim NumLock As New NumLockClass
    NumLock.Value = Not NumLock.Value
End Sub
```

Jest chyba zupełnie oczywiste, że zastosowanie klasy *NumLock* jest znacznie prostsze niż używanie odpowiednich funkcji API. Po utworzeniu modułu klasy możesz ponownie używać go w innych projektach. Aby to zrobić, wystarczy po prostu zaimportować do nowego projektu odpowiedni moduł klasy.



Kompletny moduł klasy dla tego przykładu znajdziesz w skoroszycie *Klawiatura.xlsxm*, na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pw.htm>). Skoroszyt zawiera także moduły klas umożliwiające wykrywanie i ustawianie stanu klawiszy *CapsLock* oraz *ScrollLock*.

Dodatkowe informacje na temat modułów klas

Przykład zaprezentowany w poprzednim podrozdziale pokazuje sposób tworzenia nowej klasy z jedną właściwością *Value* oraz jedną metodą *Toggle*. Klasa może zawierać dowolną liczbę właściwości, metod i zdarzeń.

Nazwa modułu klasy, w którym definiuje się klase, jest również nazwą klasy obiektu. Domyślnie modułom klas są nadawane nazwy *Class1*, *Class2* itd. Zazwyczaj warto nadać im bardziej opisowe nazwy.

Programowanie właściwości obiektów

Większość obiektów posiada co najmniej jedną właściwość, aczkolwiek w razie potrzeby możesz tworzyć dowolną liczbę właściwości obiektów. Zdefiniowaną właściwość można zastosować w kodzie, wykorzystując standardową notację z kropką:

obiekt.właściwość

W edytorze Visual Basic dostępne jest automatyczne wyświetlanie składowych obiektów zdefiniowanych za pomocą modułów klas (mechanizm *Auto List Members*), co ułatwia wybór właściwości i metod podczas pisania kodu.

Definiowane właściwości obiektu mogą mieć status tylko do odczytu, tylko do zapisu lub do zapisu i odczytu. Właściwość *tylko do odczytu* definiuje się za pomocą jednej procedury z użyciem słowa kluczowego **Property Get**. Oto przykład:

```
Property Get FileNameOnly() As String
    Dim Sep As String, LastSep As Long
    Sep = Application.PathSeparator
    LastSep = InStrRev(FullName, Sep)
    FileNameOnly = Right(FullName, Len(FullName) - LastSep)
End Property
```

Z pewnością zauważyłeś, że procedura **Property Get** działa jak funkcja. Kod wykonuje obliczenia, a następnie zwraca wartość właściwości odpowiadającą nazwie procedury. W pokazanym przykładzie nazwa procedury to **FileNameOnly**. Wartość zwróconej właściwości to nazwa pliku wyznaczona na podstawie łańcucha znaków określającego pełną ścieżkę, który jest zapisany w zmiennej publicznej **FullName**. Jeżeli na przykład zmienna **FullName** ma wartość *c:\data\myfile.txt*, procedura zwróci wartość *myfile.txt*. Procedura **FileNameOnly** jest wywoływana w momencie odwołania do obiektu i właściwości w kodzie VBA.

Dla właściwości *do odczytu i zapisu* tworzy się dwie procedury: **Property Get** (do odczytywania wartości właściwości) oraz **Property Let** (do zapisywania wartości właściwości). Wartość przypisywana do właściwości jest przekazywana jako argument procedury **Property Get**.

Poniżej zaprezentowano dwa przykłady procedur:

```
Dim XLFile As Boolean

Property Get SaveAsExcelFile() As Boolean
    SaveAsExcelFile = XLfile
End Property

Property Let SaveAsExcelFile(boolVal As Boolean)
    XLfile = boolVal
End Property
```

Jeżeli dana właściwość jest typu obiektowego, zamiast procedury **Property Let** powinieneś używać procedury **Property Set**.



Jako właściwość obiektu można również wykorzystać zmienną publiczną zadeklarowaną w module klasy. W poprzednim przykładzie można wyeliminować procedury **Property Get** oraz **Property Let** i zastąpić je deklaracjami poziomu modułu:

```
Public SaveAsExcelFile As Boolean
```

Aby utworzyć właściwość *tylko do zapisu*, należy skorzystać wyłącznie z procedury **Property Let** (bez odpowiadającej jej procedury **Property Get**). Takie właściwości definiuje się jednak bardzo rzadko.

W poprzednich przykładach wykorzystywano zmienną poziomu modułu typu Boolean o nazwie `XLFile`. Procedura `Property Get` zwraca tę zmienną jako wartość właściwości. Jeżeli na przykład obiektowi nadano nazwę `FileSys`, poniższa instrukcja wyświetli bieżącą wartość właściwości `SaveAsExcelFile`:

```
MsgBox FileSys.SaveAsExcelFile
```

Z kolei instrukcja `Property Let` przyjmuje argument i wykorzystuje go do zmiany wartości właściwości. Na przykład w celu ustawienia właściwości `SaveAsExcelFile` na wartość `True` można wykorzystać następującą instrukcję:

```
FileSys.SaveAsExcelFile = True
```

W tym przypadku do procedury `Property Let` przekazano wartość `True`, powodując zmianę wartości właściwości.

W poprzednich przykładach wykorzystano zmienną poziomu modułu `XLFile`, w której była zapisana rzeczywista wartość właściwości. W module klasy należy zadeklarować zmienne reprezentujące wartości wszystkich zdefiniowanych właściwości.



Podczas nadawania nazw procedurom definiującym właściwości stosuje się zasady nadawania nazw obowiązujące dla zwykłych procedur. Użycie niektórych nazw (stów zastrzeżonych) jest niedozwolone. Zatem jeśli podczas tworzenia procedury definiującej właściwość wystąpi błąd składniowy, można spróbować zmienić nazwę procedury.

Programowanie metod obiektów

Obiekt zwykle zawiera metody, ale nie jest to warunek konieczny. Metody w klasach obiektów definiuje się za pomocą standardowych procedur `Sub` lub `Function`, które należy umieścić w module klasy. W kodzie wywołuje się je, korzystając ze standardowej notacji:

```
obiekt.metoda
```

Podobnie jak wszystkie metody w języku VBA, tak i metody definiowane przez programistę w module klasy wykonują pewne działania. Zaprezentowana poniżej procedura jest przykładem metody zapisującej skoroszyt w jednym z dwóch formatów plików w zależności od wartości zmiennej `XLFile`. Jak można się przekonać, w treści tej procedury nie ma niczego nadzwyczajnego.

```
Sub SaveFile()
    If XLFile Then
        ActiveWorkbook.SaveAs FileName:=FName, _
            FileFormat:=xlWorkbookNormal
    Else
        ActiveWorkbook.SaveAs FileName:=FName, _
            FileFormat:=xlCSV
    End If
End Sub
```

Przykład `CSVFileClass` zaprezentowany w następnym podrozdziale przybliża właściwości i metody klas definiowanych w modułach klas.

Zdarzenia definiowane w module klasy

W każdym module klasy znajdują się dwa zdarzenia: Initialize oraz Terminate. Zdarzenie Initialize jest generowane w momencie tworzenia nowego egzemplarza obiektu, natomiast zdarzenie Terminate w momencie niszczenia obiektu. Zdarzenie Initialize można wykorzystać do ustawienia domyślnych wartości właściwości.

Szablonowy procedur obsługi wymienionych zdarzeń zaprezentowano poniżej:

```
Private Sub Class_Initialize()  
    ' Tutaj należy wprowadzić kodinicjalizacji obiektu  
End Sub  
  
Private Sub Class_Terminate()  
    ' Tutaj należy wprowadzić kodwykonywany w przypadku niszczenia obiektu  
End Sub
```

Po zakończeniu wykonywania procedury lub modułu, w którym zadeklarowano obiekt, jest on *niszczony* (a zajmowana przez niego pamięć zwalniana). Obiekt można zniszczyć w dowolnym momencie, przypisując do niego wartość Nothing. Na przykład poniższa instrukcja niszczy obiekt o nazwie MyObject:

```
Set MyObject = Nothing
```

Przykład: klasa CSVFileClass

W przykładzie z tego podrozdziału zdefiniowano klasę CSVFileClass zawierającą dwie właściwości oraz dwie metody:

■ Właściwości:

- ExportRange — (odczyt-zapis) wyznacza zakres arkusza, który ma być wyeksportowany do pliku CSV.
- ImportRange — (odczyt-zapis) wskazuje zakres, do którego ma być zainportowany ten plik.

■ Metody:

- Import — importuje plik CSV reprezentowany przez argument CSVFileName do zakresu reprezentowanego przez właściwość ImportRange.
- Export eksportuje zakres reprezentowany przez właściwość ExportRange do pliku CSV reprezentowanego przez argument CSVFileName.



Skoroszyt z tym przykładem (*Klasa CSV.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W sieci

Zmienne poziomu modułu dla klasy CSVFileClass

W module klasy muszą być zadeklarowane zmienne prywatne, które odwzorowują ustawienia właściwości klasy. W module klasy CSVFileClass wykorzystano dwie zmienne służące do zapisania dwóch ustawień właściwości. Zmienne te zadeklarowano na początku modułu klasy:

```
Private RangeToExport As Range  
Private ImportToCell As Range
```

RangeToExport jest obiektem typu Range, który reprezentuje zakres do wyeksportowania. ImportToCell jest obiektem typu Range reprezentującym lewą górną komórkę zakresu, do którego zostanie zimportowany plik. Tym zmiennym są nadawane wartości za pomocą procedur Property Get oraz Property Let, które przedstawiono w następnym punkcie.

Definicje właściwości klasy CSVFileClass

Procedury definicji właściwości dla modułu klasy CSVFileClass prezentuje poniższy listing. Procedury Property Get zwracają wartość zmiennej, natomiast procedury Property Let ustawiają wartość zmiennej.

```
Property Get ExportRange() As Range  
    Set ExportRange = RangeToExport  
End Property  
  
Property Let ExportRange(rng As Range)  
    Set RangeToExport = rng  
End Property  
  
Property Get ImportRange() As Range  
    Set ImportRange = ImportToCell  
End Property  
  
Property Let ImportRange(rng As Range)  
    Set ImportToCell = rng  
End Property
```

Definicje metod klasy CSVFileClass

Moduł klasy CSVFileClass zawiera dwie procedury reprezentujące dwie metody. Omówiono je w kolejnych podpunktach.

Procedura Export

Procedura Export jest wywoływaną w momencie wykonywania metody Export. Pobiera jeden argument: pełną nazwę pliku, do którego zostanie zapisany wyeksportowany zakres. Procedura zawiera prostą obsługę błędów. Między innymi wymusza ustawienie właściwości ExportRange poprzez sprawdzenie zmiennej RangeToExport. W celu śledzenia innych błędów w procedurze zdefiniowano blok obsługi błędów.

```
Sub Export(CSVFileName)
    ' Eksportuje zakres do pliku CSV
    If RangeToExport Is Nothing Then
        MsgBox "Nie zdefiniowano obszaru ExportRange"
        Exit Sub
    End If

    On Error GoTo ErrHandle
    Application.ScreenUpdating = False
    Set ExpBook = Workbooks.Add(xlWorksheet)
    RangeToExport.Copy
    Application.DisplayAlerts = False

    With ExpBook
        .Sheets(1).Paste
        .SaveAs FileName:=CSVFileName, FileFormat:=xlCSV
        .Close SaveChanges:=False
    End With
    Application.CutCopyMode = False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    Exit Sub
ErrHandle:
    ExpBook.Close SaveChanges:=False
    Application.CutCopyMode = False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "Error " & Err & vbCrLf & vbCrLf & Error(Err), _
           vbCritical, "Błąd metody Export"
End Sub
```

Procedura kopiuje zakres określony przez zmienną RangeToExport do nowego, tymczasowego skoroszytu, zapisuje skoroszyt jako plik tekstowy *CSV*, a następnie zamyka plik. Ponieważ aktualizowanie ekranu jest wyłączone, użytkownik nie widzi tych działań. Jeżeli wystąpi błąd — na przykład zostanie wprowadzona niepoprawna nazwa pliku — następuje przekazanie sterowania do sekcji ErrHandle i wyświetlenie komunikatu o błędzie, zawierającego numer błędu oraz jego opis.

Procedura Import

Procedura Import importuje plik *CSV* określony przez argument CSVFileName i kopiuje jego zawartość do zakresu określonego zmienną ImportToCell, której wartość odpowiada właściwości ImportRange. Po wykonaniu tej operacji plik jest zamknięty. Ponieważ i tym razem aktualizowanie ekranu jest wyłączone, użytkownik nie widzi otwartego pliku. Podobnie jak procedura Export, tak i procedura Import zawiera prostą obsługę błędów.

```
Sub Import(CSVFileName)
    ' Importuje plik CSV do zakresu
    If ImportToCell Is Nothing Then
        MsgBox "Nie zdefiniowano zakresu ImportRange"
        Exit Sub
    End If
```

```
If CSVFileName = "" Then
    MsgBox "Nie zdefiniowano pliku do zimportowania"
    Exit Sub
End If

On Error GoTo ErrHandle
Application.ScreenUpdating = False
Application.DisplayAlerts = False
Workbooks.Open CSVFileName
Set CSVFile = ActiveWorkbook
ActiveSheet.UsedRange.Copy Destination:=ImportToCell
CSVFile.Close SaveChanges:=False
Application.ScreenUpdating = True
Application.DisplayAlerts = True
Exit Sub
ErrHandle:
    CSVFile.Close SaveChanges:=False
    Application.ScreenUpdating = True
    Application.DisplayAlerts = True
    MsgBox "Error " & Err & vbCrLf & vbCrLf & Error(Err), _
        vbCritical, "Błąd metody Import"
End Sub
```

Wykorzystanie obiektów CSVFileClass

W celu utworzenia egzemplarza klasy CSVFileClass należy najpierw zadeklarować zmienną typu CSVFileClass. Oto przykład:

```
Dim CSVFile As New CSVFileClass
```

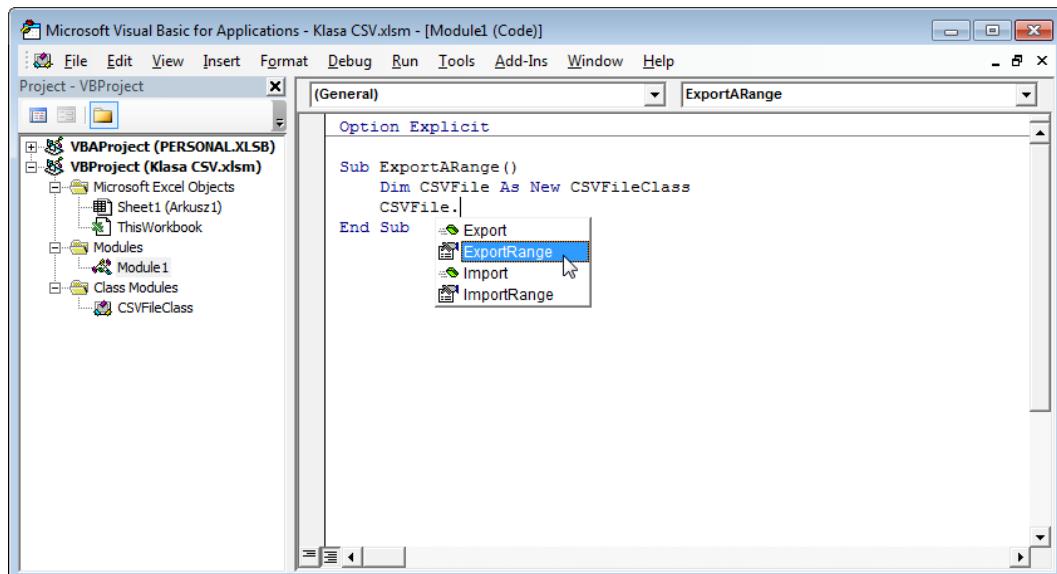
Można też najpierw zadeklarować zmienną obiektową, a obiekt utworzyć wtedy, kiedy będzie potrzebny. Taka operacja wymaga użycia instrukcji Dim i Set:

```
Dim CSVFile As CSVFileClass
    ' tutaj można umieścić inny kod
Set CSVFile = New CSVFileClass
```

Zaletą zastosowania instrukcji Dim i Set jest fakt, że obiekt nie istnieje do czasu wykonania instrukcji Set. W ten sposób można zaoszczędzić pamięć, ponieważ obiekt jest tworzony dopiero wtedy, kiedy jest rzeczywiście potrzebny. W kodzie można umieścić instrukcje warunkowe, które zadecydują, czy obiekt będzie faktycznie utworzony. Dodatkowo zastosowanie instrukcji Set umożliwia utworzenie wielu egzemplarzy obiektu.

Po utworzeniu egzemplarza obiektu można wykorzystywać właściwości i metody zdefiniowane w module klasy.

Jak można zobaczyć na rysunku 27.2, edytor Visual Basic automatycznie wyświetla składowe zdefiniowanego obiektu klasy. Po wpisaniu nazwy zmiennej i kropki wyświetla się lista właściwości i metod obiektu.



Rysunek 27.2. Mechanizm Auto List Members automatycznie wyświetla listę właściwości i metod

Poniższa procedura pokazuje sposób zapisania zaznaczonego zakresu do pliku CSV o nazwie *temp.csv*, znajdującego się w tym samym katalogu, co bieżący skoroszyt:

```
Sub ExportARange()
    Dim CSVFile As New CSVfileClass
    With CSVFile
        .ExportRange = ActiveWindow.RangeToExport
        .Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
    End With
End Sub
```

Zastosowanie struktury *With ... End With* nie jest obowiązkowe. Procedurę równie dobrze można zapisać w następujący sposób:

```
Sub ExportARange()
    Dim CSVFile As New CSVfileClass
    CSVFile.ExportRange = ActiveWindow.RangeToExport
    CSVFile.Export CSVFileName:=ThisWorkbook.Path & "\temp.csv"
End Sub
```

Poniższa procedura pokazuje sposób importowania pliku CSV, począwszy od aktywnej komórki:

```
Sub ImportAFile()
    Dim CSVFile As New CSVfileClass
    With CSVFile
        On Error Resume Next
            .ImportRange = ActiveCell
            .Import CSVFileName:=ThisWorkbook.Path & "\temp.csv"
    End With
    If Err <> 0 Then
        MsgBox "Nie można zaimportować " & ThisWorkbook.Path & "\temp.csv"
    End Sub
```

Nie ma przeskódu, by zdefiniować więcej niż jeden egzemplarz obiektu. Na przykład poniższy kod tworzy tablice trzech obiektów CSVFileClass:

```
Sub Export3Files()
    Dim CSVFile(1 To 3) As New CSVFileClass
    CSVFile(1).ExportRange = Range("A1:A20")
    CSVFile(2).ExportRange = Range("B1:B20")
    CSVFile(3).ExportRange = Range("C1:C20")

    For i = 1 To 3
        CSVFile(i).Export CSVFileName:="Plik" & i & ".csv"
    Next i
End Sub
```

Rozdział 28.

Praca z kolorami

W tym rozdziale:

- Definiowanie kolorów w kodzie VBA
- Zastosowanie funkcji VBA do konwersji różnych modeli kolorów
- Konwersja kolorów na skalę szarości
- Praca z motywami dokumentów
- Modyfikacja kolorów kształtów (obiekty Shape)
- Modyfikacja kolorów wykresów

Definiowanie kolorów

Praca z kolorami w Excelu wcale nie jest taka prosta, jak mogłoby się na pierwszy rzut oka wydawać. Co gorsza, rejestrowanie makr podczas zmiany koloru komórek i obiektów wcale nie pomaga, a wręcz przeciwnie, może wywoływać zakłopotanie.

Jedną z najbardziej znaczących zmian wprowadzonych dwie wersje temu, w Excelu 2007, było porzucenie starej, 56-elementowej palety kolorów. Przed pojawiением się Excela 2007, kolory z tej palety były jedynymi, jakich mogłeś używać do zmiany barwy tła komórek, zmiany koloru czcionek i formatowania wykresów. Oczywiście mogłeś zmieniać kolory tych elementów, ale nie było żadnego sposobu na użycie kolorów spoza palety 56 kolorów przechowywanej w skoroszycie.

Wraz z wprowadzeniem Excela 2007 wszystko się zmieniło. Teraz możesz korzystać z niespalnie nieograniczonej liczby kolorów — a dokładniej mówiąc, ilość kolorów jest „ograniczona” do astronomicznej liczby 16 777 216 kolorów, co jednak w praktyce pozwala przyjąć założenie, że liczba dostępnych kolorów w skoroszycie jest po prostu nielimitowana.

W języku VBA możesz zdefiniować kolor jako wartość dziesiętną, reprezentowaną przez liczbę z zakresu od 0 do 16 777 215. Na przykład polecenie VBA przedstawione poniżej zmienia kolor tła aktywnej komórki na ciemny brąz:

```
ActiveCell.Interior.Color = 5911168
```

W języku VBA istnieje zestaw predefiniowanych stałych reprezentujących często używane kolory. Na przykład stała vbRed ma wartość 255 (wartość dziesiętna, reprezentująca kolor czerwony), a stała vbGreen ma wartość 65280.

Oczywiście nikt nie jest w stanie zapamiętać wartości niemal 17 milionów kolorów i siłą rzeczy liczba predefiniowanych stałych również jest dosyć ograniczona. Lepszym sposobem na zmianę koloru jest zdefiniowanie koloru za pomocą wartości reprezentujących jego kolory składowe — czerwony, zielony i niebieski, czyli dobrze wszystkim znane kolory modelu RGB.

Model kolorów RGB

Model RGB pozwala na zdefiniowanie koloru za pomocą poziomów jego kolorów składowych: czerwonego, zielonego i niebieskiego. Zakres wartości kolorów składowych może się zmieniać od 0 do 255, stąd całkowitą liczbę dostępnych kolorów można wyliczyć z prostego działania $255 \times 255 \times 255 = 16\ 777\ 216$. Kiedy wszystkie trzy kolory składowe mają wartość 0, w rezultacie otrzymujemy kolor czarny. Kiedy wszystkie trzy kolory składowe mają wartość 255, w rezultacie otrzymujemy kolor biały. Kiedy wszystkie trzy kolory składowe mają wartość 128 (połowa zakresu), wynikiem będzie pośredni kolor szary. Cała reszta, czyli — bagatela — 16 777 213 kolorów, jest reprezentowana przez pozostałe kombinacje trzech wspomnianych kolorów składowych.

Aby w języku VBA zdefiniować kolor reprezentowany przez wartości RGB, powinieneś użyć funkcji RGB, która pobiera trzy argumenty reprezentujące składową czerwoną, zieloną i niebieską i zwraca wartość dziesiętną reprezentującą kolor wyjściowy.

Polecenie przedstawione poniżej używa funkcji RGB do ustawienia dokładnie takiego samego koloru tła komórki, jak w poprzednim przykładzie (ciemny brąz, o wartości 5911168):

```
ActiveCell.Interior.Color = RGB(128, 50, 90)
```

W tabeli 28.1 przedstawiono wartości RGB i odpowiadające im wartości dziesiętne wybranych kolorów.

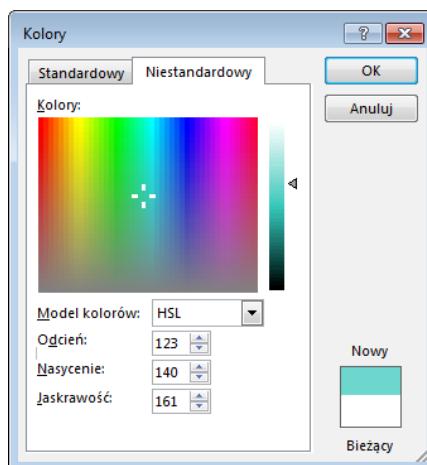
Tabela 28.1. Wartości składników RGB wybranych kolorów

Nazwa	Składnik czerwony (R)	Składnik zielony (G)	Składnik niebieski (B)	Wartość koloru
Czarny	0	0	0	0
Biały	255	255	255	16777215
Czerwony	255	0	0	255
Zielony	0	255	0	65280
Niebieski	0	0	255	16711680
Żółty	255	255	0	65535
Różowy	255	0	255	16711935
Turkusowy	0	255	255	16776960
Brązowy	153	51	0	13209
Błękitny	51	51	153	10040115
Szary 80%	51	51	51	3355443

Model kolorów HSL

Jeżeli podczas wybierania kolorów w Excelu wybierzesz opcję *Więcej kolorów*, na ekranie pojawi się okno dialogowe *Kolory*. Przejdź na kartę *Niestandardowy*, gdzie z listy rozwijanej będziesz mógł wybrać żądany model kolorów: RGB lub HSL. Na rysunku 28.1 przedstawiono wygląd okna dialogowego *Kolory* po wybraniu modelu HSL.

Rysunek 28.1.
Wybieranie koloru
przy użyciu
modelu HSL



W modelu kolorów HSL kolor jest definiowany przy użyciu trzech parametrów: *Odcienia* (inaczej barwy), *Nasycenia* i *Jaskrawości*. Podobnie jak w przypadku modelu RGB, tak i tu każdy z tych parametrów może przyjmować wartości z zakresu od 0 do 255. Każdy kolor modelu RGB ma swój odpowiednik w modelu HSL, a każdy kolor modelu HSL posiada reprezentującą go wartość dziesiętną. Innymi słowy, dowolny z 16 777 216 kolorów może być zdefiniowany za pomocą dowolnego z trzech modeli: RGB, HSL lub wartości dziesiętnej.

Okno dialogowe *Kolory* pozwala na zdefiniowanie koloru za pomocą modelu HSL, ale warto zauważyć, że jest to *jedynie* miejsce, gdzie Excel obsługuje model HSL. Jeśli na przykład definiujesz kolor przy użyciu VBA, musisz użyć wartości dziesiętnej lub użyć funkcji RGB, która zamienia składowe koloru RGB na wartość dziesiętną. W języku VBA nie istnieje funkcja pozwalająca na zdefiniowanie koloru za pomocą parametrów odcienia, nasycenia i jaskrawości, czyli inaczej mówiąc, za pomocą modelu kolorów HSL.

Konwersja kolorów

Jeżeli znasz wartości trzech kolorów składowych, zamiana koloru RGB na wartość dziesiętną jest zadaniem prostym — wystarczy użyć funkcji RGB języka VBA. Założmy, że w procedurze zostały zadeklarowane trzy zmienne, *r*, *g* oraz *b*, reprezentujące poszczególne wartości składowe koloru (zakres wartości od 0 do 255). Aby wyznaczyć wartość dziesiętną reprezentującą dany kolor, możesz użyć na przykład następującego polecenia:

```
DecimalColor = RGB(r, g, b)
```

Funkcja, która zwraca wartość koloru

Jeżeli chcesz umożliwić użytkownikowi wybieranie koloru, powinieneś rzucić okiem na bardzo przydatną funkcję, której kod prezentujemy poniżej. Funkcja GetAColor wyświetla na ekranie okno dialogowe *Kolory* (patrz rysunek 28.1) i zwraca wartość dziesiętną, reprezentującą kod danego koloru (lub wartość *False*, jeżeli użytkownik naciśnie przycisk *Anuluj*).

```
Function GetAColor() As Variant
    Dim OldColor As Double
    OldColor = ActiveWorkbook.Colors(1)
    If Application.Dialogs(xlDialogEditColor).Show(1) = True Then
        GetAColor = ActiveWorkbook.Colors(1)
    Else
        GetAColor = False
    End If
    ActiveWorkbook.Colors(1) = OldColor
End Function
```

Funkcja domyślnie korzysta ze starej palety 56 kolorów. Jest to to samo okno dialogowe, które pojawia się, kiedy użytkownik chce zmienić pierwszy kolor w domyślnej palecie kolorów. Oczywiście sama paleta kolorów nie jest w żaden sposób modyfikowana, a funkcja zwraca po prostu wartość koloru wybranego przez użytkownika.

Skoroszyt z tym przykładem (*Pobieranie wartości koloru.xlsm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

Aby przeprowadzić taką konwersję w formule arkuszowej, możesz utworzyć prostą funkcję osłonową VBA:

```
Function RGB2DECIMAL(R, G, B)
    ' Zamienia kolor z modelu RGB na wartość dziesiętną
    RGB2DECIMAL = RGB(R, G, B)
End Function
```

Przedstawiona poniżej formuła arkuszowa przyjmuje, że wartości trzech kolorów składowych znajdują się w zakresie komórek A1:C1:

=RGB2DECIMAL(A1,B1,C1)

Zamiana wartości dziesiętnej na wartości kolorów składowych RGB jest już nieco bardziej złożona. Poniżej przedstawiono funkcję, która zwraca trzyelementową tablicę zawierającą wartości składowe koloru RGB:

```
Function DECIMAL2RGB(ColorVal) As Variant
    ' Zamienia wartość dziesiętną koloru na składowe RGB
    ' Zwraca 3-elementową tablicę typu variant
    DECIMAL2RGB = Array(ColorVal \ 256 ^ 0 And 255,
                         ColorVal \ 256 ^ 1 And 255, ColorVal \ 256 ^ 2 And 255)
End Function
```

Aby użyć funkcji DECIMAL2RGB w formule arkuszowej, formuła musi zostać utworzona jako trójkomórkowa formuła tablicowa. Na przykład założmy, że w komórce A1 znajduje się wartość dziesiętna reprezentująca kolor. Aby zamienić tę wartość na elementy składowe RGB, zaznacz poziomy zakres obejmujący trzy komórki i wprowadź formułę przedstawioną poniżej (nie wpisuj nawiasów klamrowych). Po zakończeniu naciśnij kombinację klawiszy *Ctrl+Shift+Enter*, aby utworzyć formułę tablicową.

{=DECIMAL2RGB(A1)}

Jeżeli zaznaczony zakres trzech komórek jest pionowy, musisz dokonać transpozycji tablicy, na przykład:

{=TRANSPOSE(DECIMAL2RGB(A1))}

Na rysunku 28.2 przedstawiono sposób użycia funkcji DECIMAL2RGB w arkuszu.

Rysunek 28.2.

Funkcja DECIMAL2RGB zamienia wartości dziesiętne kolorów na elementy składowe RGB

		Decimal			Decimal-2-RGB			Decimal-2-HSL			
1	2	A	B	C	D	E	F	G	H	I	J
	Wartość koloru			R	G	B		H	S	L	
2	0			0	0	0		0	0	0	
3	167 772			92	143	2		58	248	72	
4	335 544			184	30	5		6	242	94	
5	503 316			20	174	7		82	235	90	
6	671 088			112	61	10		21	213	61	
7	838 860			204	204	12		42	227	108	
8	1 006 632			40	92	15		71	184	54	
9	1 174 404			132	235	17		63	221	126	
10	1 342 176			224	122	20		21	213	122	
11	1 509 948			60	10	23		244	182	35	
12	1 677 720			152	153	25		43	183	89	
13	1 845 492			244	40	28		2	231	136	
14	2 013 264			80	184	30		71	184	107	
15	2 181 036			172	71	33		12	173	102	
16	2 348 808			8	215	35		91	237	112	
17	2 516 580			100	102	38		44	117	70	
18	2 684 352			192	245	40		53	232	142	
19	2 852 124			28	133	43		91	166	80	
20	3 019 896			120	20	46		244	182	70	
21	3 187 668			212	163	48		30	167	130	
22	3 355 440			48	51	51		128	8	50	
23	3 523 212			140	194	53		59	146	124	
24	3 690 984			232	81	56		6	202	144	



Skoroszyt z tym przykładem (*Funkcje konwersji kolorów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W skoroszycie zostały zdefiniowane następujące funkcje: DECIMAL2RGB, DECIMAL2HSL, HSL2RGB, RGB2DECIMAL, RGB2HSL oraz HSL2DECIMAL.

Skala szarości

Kiedy tworzysz skoroszyty i wykresy, powinieneś pamiętać, że nie wszyscy użytkownicy posiadają drukarki kolorowe. Nawet jeżeli wykres zostanie wydrukowany w kolorze, może później zostać powielony na kserokopiarce, przesłany faksem lub oglądany przez osoby mające zaburzenia widzenia kolorów (co dotyczy nawet ponad 8% populacji dorosłych ludzi).

Kilka dodatkowych słów o wartościach dziesiętnych kolorów

Być może jesteś ciekawy, w jaki sposób zorganizowane jest tych 16 777 216 wartości dziesiętnych kolorów. Kolor o wartości 0 to czarny, kolor o wartości 16777215 to biały, ale jak są poukładane kolory pomiędzy tymi wartościami?

Aby łatwiej wyobrazić sobie strukturę wartości dziesiętnych kolorów, możesz sobie wyobrazić, że kolejne wartości zostały wygenerowane przez pętlę For ... Next w procedurze przedstawionej poniżej:

```
Sub GenerateColorValues()
    Dim Red As Long, Blue As Long, Green As Long
    Dim AllColors(0 To 16777215) As Long
    Dim ColorNum As Long
    ColorNum = 0
    For Blue = 0 To 255
        For Green = 0 To 255
            For Red = 0 To 255
                AllColors(ColorNum) = RGB(Red, Blue, Green)
                ColorNum = ColorNum + 1
            Next Red
        Next Green
    Next Blue
End Sub
```

Po uruchomieniu tej procedury kolejne wartości w tabeli AllColors odpowiadają kolejnym wartościom dziesiętnych kolorów używanych w Excelu.

Kiedy pełnokolorowy dokument jest drukowany na urządzeniu czarno-białym, kolory są zamieniane na skalę szarości. Przy odrobinie szczęścia po takiej zamianie Twój dokument może wyglądać całkiem przyzwoicie, ale nie zawsze się tak dzieje. Na przykład może się zdarzyć, że po zamianie na skalę szarości poszczególne serie danych na wykresie będą trudne do rozróżnienia.

Każdy kolor w skali szarości ma swój odpowiednik w modelu kolorów RGB. Kolor czarny jest reprezentowany przez RGB(0, 0, 0). Kolor biały jest reprezentowany przez RGB(255, 255, 255). Pośredni kolor szary jest reprezentowany przez RGB(128, 128, 128). Skala szarości pozwala na uzyskanie 256 odrębnych poziomów szarości.

Aby utworzyć 256-kolorową skalę szarości w zakresie komórek, powinieneś wykonać procedurę, której kod zamieszczono poniżej. Uruchomienie procedury zmienia kolor tła komórek w zakresie A1:A256, począwszy od koloru czarnego aż do pełnej bieli. Aby zobaczyć cały zakres komórek, możesz użyć funkcji skalowania podglądu arkusza.

```
Sub GenerateGrayScale()
    Dim r As Long
    For r = 0 To 255
        Cells(r + 1, 1).Interior.Color = RGB(r, r, r)
    Next r
End Sub
```

Na rysunku 28.3 przedstawiono wynik działania tej procedury, po zmniejszeniu wysokości wierszy i zwiększeniu szerokości kolumny A.



Rysunek 28.3. Komórki arkusza wyświetlające 256 odcieni szarości

Zamiana kolorów na skalę szarości

Jednym ze sposobów konwersji kolorów na skalę szarości jest uśrednienie wartości elementów składowych RGB koloru i użycie uzyskanej w ten sposób pojedynczej wartości dla składników RGB odpowiadającego poziomu szarości. Takie podejście nie uwzględnia jednak faktu, że różne kolory są postrzegane jako mające różne poziomy jaskrawości. Na przykład kolor zielony jest odbierany jako jaśniejszy niż czerwony, a czerwony jako jaśniejszy niż niebieski.

Doświadczenia z użytkownikami przyniosły w rezultacie swego rodzaju „empiryczną” recepturę pozwalającą na efektywną zamianę kolorów RGB na odpowiadające im poziomy szarości:

- 29,9% składnika czerwonego;
- 58,7% składnika zielonego;
- 11,4% składnika niebieskiego.

Na przykład rozważmy kolor o wartości 16751001, czyli odcień fioletu, odpowiadający kolorowi RGB(153, 153, 255). Podstawiając te wartości do naszego równania konwersji, otrzymujemy odpowiednio następujące wartości kolorów składowych RGB:

- **R (czerwony):** $29,9\% \times 153 = 46$
- **G (zielony):** $58,7\% \times 153 = 90$
- **B (niebieski):** $11,4\% \times 255 = 29$

Suma tych wartości wynosi 165, stąd odpowiadający kolorowi 16751001 poziom szarości w modelu RGB będzie reprezentowany przez wartość RGB(165, 165, 165).

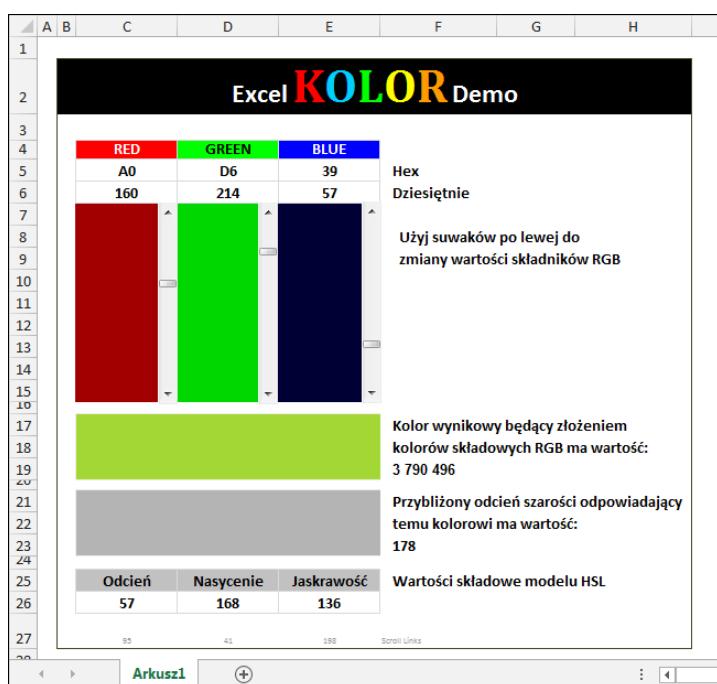
Poniżej zamieszczono kod funkcji VBA, która pobiera jako argument dziesiętną wartość koloru i zwraca odpowiadającą jej wartość dziesiętną reprezentującą kolor w skali szarości.

```
Function Grayscale(color)
    Dim r As Long, g As Long, b As Long
    r = (color \ 256 ^ 0 And 255) * 0.299
    g = (color \ 256 ^ 1 And 255) * 0.587
    b = (color \ 256 ^ 2 And 255) * 0.114
    Grayscale = RGB(r + g + b, r + g + b, r + g + b)
End Function
```

Eksperymenty z kolorami

Na rysunku 28.4 przedstawiono skoroszyt, który został utworzony w celu eksperymentowania z kolorami. Jeżeli nie jesteś pewny, jak działa model kolorów RGB, poświęcenie chwili czasu na zabawę z tym skoroszytem z pewnością wyjaśni Twoje wątpliwości.

Rysunek 28.4.
Skoroszyt ilustruje
zasady tworzenia
kolorów
w modelu RGB



Skoroszyt z tym przykładem (*Model RGB.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).

W skoroszycie znajdziesz trzy pionowe suwaki sterujące kolorami tła zakresu komórek. Za ich pomocą możesz zmieniać wartości składowych RGB w zakresie od 0 do 255. Zmiana położenia suwaków wprowadza zmiany w kilku obszarach arkusza:

- Komórki znajdujące się powyżej suwaków wyświetlają wartości kolorów składowych w formacie szesnastkowym (heksadecymalnym), w zakresie od 00 do FF. Szesnastkowe wartości kolorów składowych RGB są często wykorzystywane do definiowania kolorów w dokumentach HTML.
- Zakresy komórek znajdujące się obok poszczególnych suwaków zmieniają nasycenie koloru odpowiadające aktualnemu położeniu suwaka (czyli inaczej mówiąc, aktualnej wartości koloru składowego).
- Zakres komórek znajdujący się poniżej suwaków wyświetla kolor wynikowy, reprezentujący złożenie kolorów składowych RGB.
- W komórce obok wyświetlana jest wartość dziesiętna reprezentująca kolor wynikowy.
- Kolejny zakres komórek poniżej wyświetla wygląd koloru wynikowego po zamianie na odpowiadający mu poziom skali szarości.
- Zakres komórek poniżej wyświetla odpowiadające wybranemu kolorowi wartości składowe koloru w modelu HSL.

Praca z motywami dokumentów

Jedną ze znaczących funkcji wprowadzonych już w Excelu 2007 są motywy dokumentów. Dzięki zastosowaniu motywów użytkownik może za pomocą jednego kliknięcia myszy zmienić wygląd całego dokumentu. Motyw dokumentu składa się z trzech głównych elementów: kolorów, czcionek i efektów (dla elementów graficznych). Racjonalne użycie motywów może ułatwić użytkownikowi tworzenie efektownych i spójnie sformatowanych dokumentów. Motywy są nakładane na cały skoroszyt, a nie tylko w aktywnym arkuszu.

Kilka słów o motywach dokumentów

W pakiecie Microsoft Office 2013 masz do dyspozycji bardzo wiele gotowych, predefiniowanych motywów dokumentów, a w razie potrzeby możesz również tworzyć swoje własne motywы. Na Wstążce znajdziesz szereg gotowych galerii stylów formatowania (na przykład galerię stylów wykresów). Style dostępne w galeriach zmieniają się w zależności od motywu przypisanego do dokumentu. Jeżeli zmienisz motyw dokumentu na inny, zmieni się kolorystyką dokumentu, lista używanych czcionek oraz efekty graficzne.



W sieci

Jeżeli nie korzystałeś do tej pory z motywów dokumentów, otwórz skoroszyt o nazwie *Motyw dokumentów.xlsx*, który znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). W skoroszycie znajdziesz zdefiniowane zakresy komórek ilustrujące kolorystkę motywu, dwa kształty, elementy tekstowe (korzystające z różnych czcionek) oraz wykres. Aby zobaczyć, jak zmienia się wygląd dokumentu po zmianie motywu, przejdź na kartę **UKŁAD STRONY** i wybierz inny motyw z galerii w grupie **Motyw**.

Użytkownicy mogą zmieniać i mieszać elementy motywów. Na przykład możesz użyć kolorystyki dokumentu z jednego motywu, czcionek z innego motywu, a efektów graficznych z jeszcze innego. Oprócz tego możesz tworzyć własne zestawy kolorów i czcionek. Zmodyfikowane motywów mogą być zapisywane i następnie wykorzystywane do formatowania innych skoroszytów.



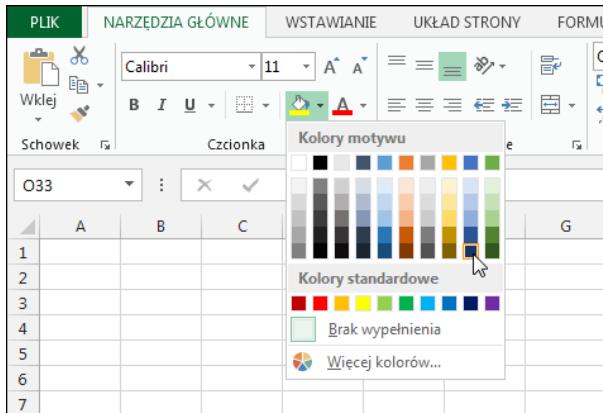
Uwaga

Koncepcja motywów dokumentu wzięła się z założenia, że użytkownicy nie będą zmieniali wyglądu dokumentów sformatowanych przy użyciu motywów. Jeżeli użytkownik używa swojej kolorystyki lub zestawu czcionek, które nie są częścią aktywnego motywu, takie elementy nie są zmieniane po wybraniu innego motywu, stąd utworzenie dokumentu przeładowanego efektami i epatującego niedopasowanymi zestawami kolorów i czcionek nadal wcale nie jest takie trudne.

Kolory motywów dokumentów

Kiedy użytkownik zmienia kolor komórki lub obiektu, może wybrać żądany kolor z galerii kolorów przedstawionej na rysunku 28.5. W galerii wyświetlane jest 60 kolorów motywu (10 kolumn po 6 kolorów), plus dodatkowych 10 kolorów standardowych. Wybranie polecenia *Więcej kolorów* wyświetla na ekranie okno dialogowe *Kolory*, za pomocą którego użytkownik może wybrać dowolny spośród 16 777 216 dostępnych kolorów.

Rysunek 28.5.
Galeria kolorów



Uwaga

Obiekty Excela (takie jak zakresy, kształty i elementy wykresów) mogą być kolorowane na dwa sposoby: poprzez nadanie jednego z kolorów motywu lub poprzez nadanie standardowego, arbitralnie wybranego koloru. Kiedy korzystasz z formantu wyboru koloru, możesz nadać danemu obiektemu żądany kolor, wybierając go z palety w sekcji *Kolory motywu*. Jeżeli wybierzesz jakiś kolor z sekcji *Kolory standardowe* lub naciśniesz przycisk *Więcej kolorów*, taki kolor nie będzie kolorem motywu i dlatego nie zmieni się po wybraniu innego motywu.

Kolory dostępne w galerii kolorów motywu są identyfikowane za pomocą podpowiedzi ekranowych. Na przykład kolor znajdujący się w szóstej kolumnie drugiego wiersza galerii jest określany jako *Akcent 2, jaśniejszy — 80%*.

Pierwszy wiersz każdej kolumny zawiera „czysty” kolor. Poniżej tego koloru znajduje się 6 wariantów koloru bazowego, różniących się odcieniem i jaskrawością. W tabeli 28.2 przedstawiono nazwy wszystkich kolorów motywu.

Tabela 28.2. Nazwy kolorów motywów

Pamiętaj, że nazwy kolorów pozostają takie same, nawet jeżeli użyjesz innego motywu dokumentu. Paleta kolorów motywu składa się z dziesięciu kolorów wyświetlanego w górnym wierszu galerii (cztery kolory tła i tekstu oraz sześć kolorów akcentowanych) i każdy z tych dziesięciu kolorów bazowych posiada po pięć dodatkowych wariantów różniących się odcieniem i jaskrawością. Jeżeli wybierzesz polecenie *UKŁAD STRONY/Motyw/Kolory/Utwórz nowe kolory motywu*, w galerii kolorów motywu znajdziesz dwa dodatkowe kolory: *Hiperłącze* oraz *Użyte hiperłącze*. Są to kolory nadawane podczas tworzenia hiperłącza; nie są wyświetlane w podstawowej galerii kolorów motywu.

Perspektywa zarejestrowania makra podczas zmiany koloru wypełnienia i koloru czcionki zakresu komórek może wydawać się interesującym doświadczeniem. Poniżej przedstawiono kod makra, które zostało zarejestrowane podczas modyfikacji kolorów zaznaczonego wcześniej zakresu komórek. Jako kolor wypełnienia wybrałem *Akcent 2, ciemniejszy 25%*, a jako kolor tekstu *Tekst/Tło 2, jaśniejszy 80%*.

```
Sub ChangeColors()
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .ThemeColor = xlThemeColorAccent2
        .TintAndShade = -0.249977111117893
        .PatternTintAndShade = 0
    End With
    With Selection.Font
        .ThemeColor = xlThemeColorLight2
        .TintAndShade = 0.799981688894314
    End With
End Sub
```

Możesz bezpiecznie zignorować trzy właściwości związane z deseniami wypełnienia (Pattern, PatternColorIndex oraz PatternTintAndShade). Wymienione właściwości odnoszą się do przestarzałych (ale nadal obsługiwanych) deseni wypełnienia komórek, które możesz zdefiniować na karcie *Wypełnienie* okna dialogowego *Formatowanie komórek*. Właściwości te odpowiadają za ewentualne desenie wypełnienia komórek, które mogą ewentualnie istnieć w zaznaczonym zakresie komórek.

Po usunięciu wspomnianych wcześniej właściwości związanych z deseniami kod naszego makra wygląda następująco:

```
Sub ChangeColors()
    With Selection.Interior
        '(Akcent 2, Ciemniejszy 25%)
        .ThemeColor = xlThemeColorAccent2
        .TintAndShade = -0.249977111117893
    End With
    With Selection.Font
        '(Tekst 2, Jaśniejszy 80%)
        .ThemeColor = xlThemeColorLight2
        .TintAndShade = 0.799981688894314
    End With
End Sub
```

Jak widać, poszczególne kolory są definiowane za pomocą właściwości `ThemeColor` oraz `TintAndShade`. Wartość właściwości `ThemeColor` jest łatwa do rozszyfrowania — to po prostu numer kolumny w tabeli 10×6 kolorów motywów. Wartości właściwości są przypisywane przy użyciu wbudowanych stałych, których wartości odpowiadają poszczególnym pozycjom tabeli 10×6 kolorów motywów. Na przykład `x1ThemeColorAccent2` ma wartość 6. Ale co w takim razie z wartością właściwości `TintAndShade`?

Właściwość `TintAndShade` może przyjmować wartości z zakresu od -1 do 1. Wartość -1 reprezentuje kolor czarny, natomiast wartość 1 reprezentuje kolor biały. Wartość 0 właściwości `TintAndShade` daje w rezultacie *czysty* kolor bazowy. Innymi słowy, jeżeli wartość właściwości `TintAndShade` przyjmuje wartości ujemne, kolor wynikowy jest coraz ciemniejszy (w porównaniu z czystym kolorem bazowym), aż do osiągnięcia barwy czarnej. Jeżeli wartość właściwości `TintAndShade` przyjmuje wartości dodatnie, kolor wynikowy jest coraz jaśniejszy (w porównaniu z czystym kolorem bazowym), aż do osiągnięcia barwy białej. Wartości właściwości `TintAndShade` odpowiadają nazwie koloru wyświetlanego w tabeli kolorów.

Jeżeli dany wariant koloru jest określany jako *Ciemniejszy*, właściwość `TintAndShade` przyjmuje wartość ujemną. Jeżeli dany wariant koloru jest określany jako *Jaśniejszy*, właściwość `TintAndShade` ma wartość dodatnią.

Zwrót uwagi na fakt, że same kolory nie są tutaj określone. Wybór konkretnego koloru jest uzależniony od bieżącego motywów dokumentu.



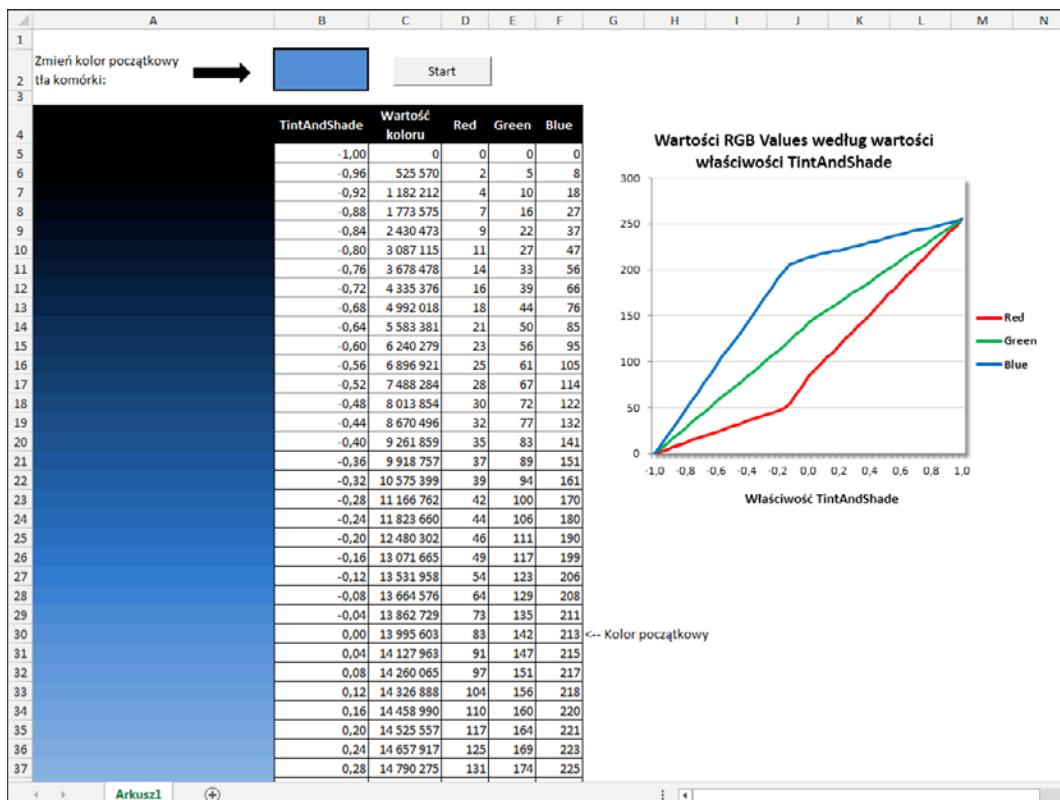
Trudno powiedzieć, dlaczego wartości właściwości `TintAndShade` w rejestrach makrach są podawane z tak wielką dokładnością (liczbą miejsc po przecinku), gdyż w praktyce tak wielka dokładność z pewnością nie jest wymagana. Na przykład wartość -0.249977111117893 właściwości `TintAndShade` wygląda dokładnie tak samo, jak wartość -0.25.



Aby przekonać się, jak zmiany wartości właściwości `TintAndShade` wpływają na kolory, otwórz skoroszyt o nazwie *Właściwość TintAndShade.xlsxm* (patrz rysunek 28.6), znajdujący się na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>). Wybierz kolor początkowy, naciśnij przycisk *Start* i makro wyświetli kolory dla 50 poziomów wartości właściwości `TintAndShade`, zmieniających się w zakresie od -1 do 1. Oprócz tego w skoroszycie wyświetlana jest wartość dziesiętna, reprezentująca dany kolor oraz wartości składowych RGB koloru (całość jest prezentowana na wykresie).

Wyświetlanie wszystkich kolorów motywów

Napisałem makro, które wyświetla w zakresie komórek wszystkie 60 kolorów aktywnego motywów. Są to te same kolory, które możesz znaleźć w tabeli wyboru kolorów motywów.



Rysunek 28.6. Skoroszyt ilustrujący wpływ wartości właściwości TintAndShade na kolory

```

Sub ShowThemeColors()
    Dim r As Long, c As Long
    For r = 1 To 6
        For c = 1 To 10
            With Cells(r, c).Interior
                .ThemeColor = c
                Select Case c
                    Case 1 'Tekst / Tło 1
                    Select Case r
                        Case 1: .TintAndShade = 0
                        Case 2: .TintAndShade = -0.05
                        Case 3: .TintAndShade = -0.15
                        Case 4: .TintAndShade = -0.25
                        Case 5: .TintAndShade = -0.35
                        Case 6: .TintAndShade = -0.5
                    End Select
                    Case 2 'Tekst / Tło 2
                    Select Case r
                        Case 1: .TintAndShade = 0
                        Case 2: .TintAndShade = 0.5
                        Case 3: .TintAndShade = 0.35
                        Case 4: .TintAndShade = 0.25
                        Case 5: .TintAndShade = 0.15
                        Case 6: .TintAndShade = 0.05
                    End Select
                End Select
            End With
        Next c
    Next r
End Sub

```

```

Case 3 ' Tekst / Tlo 3
Select Case r
    Case 1: .TintAndShade = 0
    Case 2: .TintAndShade = -0.1
    Case 3: .TintAndShade = -0.25
    Case 4: .TintAndShade = -0.5
    Case 5: .TintAndShade = -0.75
    Case 6: .TintAndShade = -0.9
End Select
Case Else ' Tekst / Tlo 4 plus akcenty 1 – 6
Select Case r
    Case 1: .TintAndShade = 0
    Case 2: .TintAndShade = 0.8
    Case 3: .TintAndShade = 0.6
    Case 4: .TintAndShade = 0.4
    Case 5: .TintAndShade = -0.25
    Case 6: .TintAndShade = -0.5
End Select
End Select
Cells(r, c) = .TintAndShade
End With
Next c
Next r
End Sub

```

Na rysunku 28.7 przedstawiono wyniki działania procedury ShowThemeColors (nie da się ukryć, że zdecydowanie lepiej wygląda to w kolorze...). Jeżeli zmienisz motyw dokumentu na inny, zestaw kolorów zostanie odpowiednio zmodyfikowany, tak aby odpowiadał bieżącemu motywowi.

	A	B	C	D	E	F	G	H	I	J	K
1	0										
2	-0,04999	0,499985	-0,09998	0,799982	0,799982	0,799982	0,799982	0,799982	0,799982	0,799982	0,799982
3	-0,15	0,349986	-0,24998	0,599994	0,599994	0,599994	0,599994	0,599994	0,599994	0,599994	0,599994
4	-0,24998	0,249977	-0,49998	0,399976	0,399976	0,399976	0,399976	0,399976	0,399976	0,399976	0,399976
5	-0,34999	0,149998	-0,74999	-0,24998	-0,24998	-0,24998	-0,24998	-0,24998	-0,24998	-0,24998	-0,24998
6	-0,49998		-0,89999	-0,49998	-0,49998	-0,49998	-0,49998	-0,49998	-0,49998	-0,49998	-0,49998
7											
8											

Rysunek 28.7. Kolory motywu wygenerowane za pomocą makra VBA



Skoroszyt z tym przykładem (*Wyświetlanie kolorów motywu.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pwm.htm>).

We wcześniejszej części rozdziału pokazaliśmy, jak zmieniać kolor tła zakresu komórek poprzez ustawianie wartości właściwości `Color` obiektu `Interior`. Jak wspominaliśmy wcześniej, użycie funkcji RGB znakomicie ułatwia takie zadanie. Polecenia przedstawione poniżej pokazują, jak zmienić kolor wypełnienia zakresu komórek (oba polecenia dają taki sam rezultat):

```
Range("A1:F24").Interior.Color = 5913728
Range("A1:F24").Interior.Color = RGB(128, 60, 90)
```

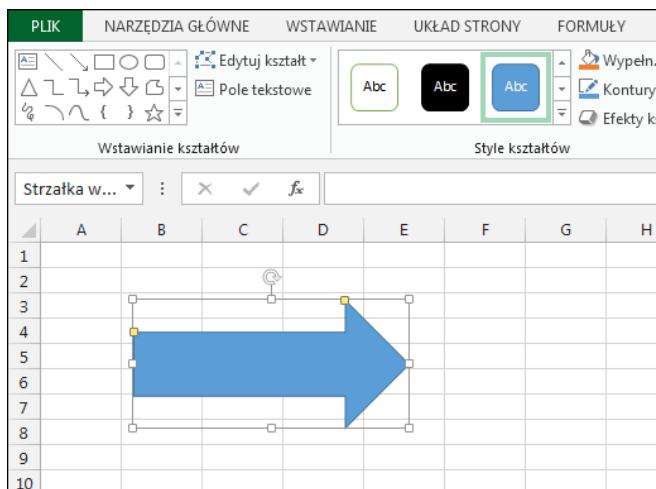
Pamiętaj, że przypisanie koloru w taki sposób nie powoduje, że dany kolor staje się kolorem motywu. Innymi słowy, jeżeli po wykonaniu jednego z tych poleceń użytkownik wybierze inny motyw dokumentu, to komórki w zakresie A1:F24 nie zmienią koloru. Aby zmieniać kolory w sposób spójny z motywem, musisz użyć właściwości `ThemeColor` oraz (opcjonalnie) właściwości `TintAndShade`.

Praca z obiektami Shape

Do tej pory w tym rozdziale koncentrowaliśmy się wyłącznie na modyfikacji kolorów zakresu komórek. W tym podrozdziale omówiono zagadnienia związane ze zmianą kolorów obiektów Shape (kształtów). Aby umieścić na arkuszu wybrany kształt, przejdź na kartę *WSTAWIANIE* i wybierz polecenie *Kształty*, znajdujące się w grupie poleceń *Ilustracje*.

Na rysunku 28.8 przedstawiono wygląd arkusza z umieszczonym na nim kształtem. Domyślana nazwa tego kształtu to *Right Arrow 1*. Liczba na końcu nazwy zmienia się w zależności od liczby kształtów umieszczonych na arkuszu. Na przykład: jeżeli wcześniej umieściłeś na arkuszu dwa inne, dowolne kształty, nazwa trzeciego będzie brzmiała *Right Arrow 3*.

Rysunek 28.8.
Obiekt typu *Shape*
umieszczony na arkuszu



Kolor tła kształtu

Kolor tła kształtu jest definiowany za pomocą właściwości RGB. Aby wyświetlić wartość dziesiętną reprezentującą kolor kształtu, możesz użyć następującego polecenia:

```
MsgBox ActiveSheet.Shapes("Right Arrow 1").Fill.ForeColor.RGB
```

Składnia tego polecenia może być niezbyt oczywista, więc postaram się ją objaśnić. Właściwość Fill obiektu Shape zwraca obiekt FillFormat. Właściwość ForeColor obiektu FillFormat zwraca obiekt ColorFormat, zatem właściwość RGB w zasadzie odnosi się do obiektu ColorFormat i zwraca wartość dziesiętną reprezentującą kolor kształtu.



Jeżeli sposób użycia właściwości ForeColor nie jest dla Ciebie zrozumiały, to nie przejmuj się — nie jesteś pierwszy. Większość użytkowników, łącznie ze mną, mogłaby oczekiwać, że za kolor tła kształtu jest odpowiedzialna właściwość BackColor obiektu FillFormat, ale okazuje się, że właściwość BackColor reprezentuje drugi kolor kształtu używany w sytuacji, kiedy kształt posiada cieniowanie bądź jest wypełniony deseniem. W przypadku obiektów, które nie są wypełnione deseniem, za kolor tła odpowiada właściwość ForeColor.

Kiedy pracujesz z obiektami Shape, niemal zawsze kod procedury musi wykonywać wiele różnych operacji, stąd zawsze warto w takiej sytuacji utworzyć zmienną obiektową. Fragment kodu przedstawiony poniżej tworzy zmienną obiektową o nazwie Shp:

```
Dim Shp As Shape  
Set Shp = ActiveSheet.Shapes("Right Arrow 1")  
MsgBox Shp.Fill.ForeColor.RGB
```



Dodatkową zaletą utworzenia zmiennej obiektowej jest możliwość skorzystania z mechanizmu *Auto List Members*, który podczas tworzenia kodu VBA automatycznie wyświetla listę wszystkich dostępnych właściwości i obiektów (patrz rysunek 28.9). Jest to szczególnie przydatne podczas pracy z obiektami klasy Shape, ponieważ takie obiekty nie są rejestrowane przez rejestrator makr programu Excel.

Rysunek 28.9.

Tworzenie nowego wiersza kodu przy użyciu mechanizmu *Auto List Members*

```
Sub ModifyShape()  
    Dim Shp As Shape  
    Set Shp = ActiveSheet.Shapes("Right Arrow 1")  
    Shp.Fill.Fo|  
End Sub
```

The screenshot shows the Microsoft Visual Basic Editor window with the title bar "Zeszyt2 - Module1 (Code)". The code editor displays a subroutine named "ModifyShape". Inside the subroutine, there is a line of code: "Shp.Fill.Fo|". The word "ForeColor" is highlighted in blue, indicating it was selected from an auto-listed member suggestion. A dropdown menu is open over the highlighted text, showing several other members: "GradientAngle", "GradientColorType", "GradientDegree", "GradientStops", "GradientStyle", and "GradientVariant". The "ForeColor" option is the first item in the list.

Jeżeli pracujesz tylko z kolorami kształtu, możesz utworzyć zmienną obiektową dla obiektu ColorFormat. Aby to zrobić, możesz użyć następującego fragmentu kodu:

```
Dim ShpCF As ColorFormat  
Set ShpCF = ActiveSheet.Shapes("Right Arrow 1").Fill.ForeColor  
MsgBox ShpCF.RGB
```

Właściwość RGB obiektu ColorFormat odpowiada za kolor kształtu. Poniżej przedstawiono kilka dodatkowych właściwości tego obiektu. Jeżeli nie korzystałeś jeszcze z motywów dokumentów, powinieneś zajrzeć do podrozdziału „Praca z motywami dokumentów” we wcześniejszej części tego rozdziału.

- Brightness — liczba z zakresu od -1 do 1, reprezentująca jaskrawość koloru.
Wartość -1 odpowiada kolorowi czarnemu, a wartość 1 kolorowi białemu.
- ObjectThemeColor — liczba z zakresu od 1 do 15, reprezentująca kolor motywu.
- SchemeColor — liczba z zakresu od 0 do 80, reprezentująca indeks koloru ze starej 56-elementowej palety kolorów. Ponieważ właściwość ta odnosi się do przestarzałej palety kolorów, nie ma żadnej potrzeby korzystania z tej właściwości.
- TintAndShade — liczba z zakresu od -1 do 1, reprezentująca odcień i jaskrawość koloru motywu.
- Type — liczba reprezentująca typ obiektu ColorFormat. O ile mi wiadomo, jest to właściwość przeznaczona tylko do odczytu, której wartość zawsze wynosi 1 (co reprezentuje model kolorów RGB).

Zmiana koloru tła kształtu nie ma żadnego wpływu na kolor obramowania kształtu. Aby zmienić kolor obramowania kształtu, powinieneś użyć obiektu ColorFormat zawartego w obiekcie LineFormat. Fragment kodu przedstawiony poniżej zmienia kolor tła i obrysu kształtu na czerwony:

```
Dim Shp As Shape  
Set Shp = ActiveSheet.Shapes("Right Arrow 1")  
Shp.Fill.ForeColor.RGB = RGB(255, 0, 0)  
Shp.Line.ForeColor.RGB = RGB(255, 0, 0)
```

Poniżej przedstawiono alternatywny sposób osiągnięcia tego samego efektu, tym razem przy użyciu zmiennych obiektowych:

```
Dim Shp As Shape  
Dim FillCF As ColorFormat  
Dim LineCF As ColorFormat  
Set Shp = ActiveSheet.Shapes("Right Arrow 1")  
Set FillCF = Shp.Fill.ForeColor  
Set LineCF = Shp.Line.ForeColor  
FillCF.RGB = RGB(255, 0, 0)  
LineCF.RGB = RGB(255, 0, 0)
```

Pamiętaj, że kod zaprezentowany w powyższym przykładzie generuje kolory, które nie są zgodne z kolorami motywu dokumentu. Aby określić kolory zgodne z motywem, powinieneś użyć właściwości SchemeColor oraz (opcjonalnie) właściwości TintAndShade.

Kształty i kolory motywów

Aby przypisać wybranemu kształtowi kolor motwu, powinieneś użyć właściwości ObjectThemeColor oraz TintAndShade obiektu Forecolor. Żeby się przekonać, jak można to zrobić, włączylem rejestrator makr i zmieniłem kolor kształtu na *Akcent 4, jaśniejszy 40%*. W rezultacie otrzymałem następujący fragment kodu:

```
ActiveSheet.Shapes.Range(Array("Right Arrow 1")).Select
With Selection.ShapeRange.Fill
    .Visible = msoTrue
    .ForeColor.ObjectThemeColor = msoThemeColorAccent4
    .ForeColor.TintAndShade = 0
    .ForeColor.Brightness = 0.400000006
    .Transparency = 0
    .Solid
End With
```

Zwróć uwagę, że makro przedstawione powyżej modyfikuje właściwość Brightness, a nie właściwość TintAndShade. Odkryłem również, że właściwość Brightness kształtu odpowiada właściwości TintAndShade w komórce. Właściwość Brightness została wprowadzona w Excelu 2010, stąd próba wykonania takiej procedury w Excelu 2007 spowoduje wystąpienie błędu.

Niestety implementacja motywów, jakiej dokonała firma Microsoft, nie jest doskonała. Na przykład zestaw kolorów motwu dla zakresu komórek nie do końca odpowiada zestawowi kolorów motwu dla kształtów. Kolory motywów komórek można wybierać z zakresu od 1 do 12, podczas gdy dla obiektów zakres dostępnych kolorów motwu rozcina się od 1 do 15. Pierwsze cztery wartości kolorów się nie pokrywają.

Po kilku próbach napisałem makro, które nadaje wybranemu kształtowi taki sam kolor, jaki został użyty do wypełnienia zakresu komórek — jak widać poniżej, takie makro nie jest wcale takie proste, jak mogłyby się na pierwszy rzut oka wydawać.

```
Sub ColorShapeLikeCell()
    ' Nadaje kształtowi kolor wypełnienia odpowiadający kolorowi komórki A1
    Dim Cell As Interior
    Dim Shape As ColorFormat
    Set Cell = Range("A1").Interior
    Set Shape = ActiveSheet.Shapes(1).Fill.ForeColor

    If Cell.ThemeColor = 0 Then
        Shape.RGB = Cell.Color
    Else
        Select Case Cell.ThemeColor
            Case 1: Shape.ObjectThemeColor = 2
            Case 2: Shape.ObjectThemeColor = 1
            Case 3: Shape.ObjectThemeColor = 4
            Case 4: Shape.ObjectThemeColor = 3
            Case Else
                Shape.ObjectThemeColor = Cell.ThemeColor
        End Select
        Shape.Brightness = Cell.TintAndShade
    End If
End Sub
```

Jeżeli wartość właściwości `ThemeColor` jest równa 0, oznacza to, że kolor komórki nie jest kolorem motywu. W takim wypadku kształt otrzymuje po prostu takie same ustawienia koloru. Jeżeli komórka została sformatowana przy użyciu koloru motywu, kod procedury musi dokonać odpowiednich modyfikacji ustawień dla kolorów 1, 2, 3 oraz 4. Dla pozostałych kolorów właściwość `Brightness` kształtu jest ustawiana na wartość właściwości `TintAndShadow` komórki.

Poniżej przedstawiamy kolejne makro, `ColorCellLikeShape`, będące swego rodzaju komplementarnym dopełnieniem makra `ColorShapeLikeCell`, które nadaje komórce A1 taki sam kolor, jaki ma kształt.

```
Sub ColorCellLikeShape()
    ' Nadaje komórce A1 kolor wypełnienia odpowiadający kolorowi kształtu
    Dim Cell As Interior
    Dim Shape As ColorFormat
    Set Cell = Range("A1").Interior
    Set Shape = ActiveSheet.Shapes(1).Fill.ForeColor

    If Shape.ObjectThemeColor = 0 Then
        Cell.Color = Shape.RGB
    Else
        Select Case Shape.ObjectThemeColor
            Case 1: Cell.ThemeColor = 2
            Case 2: Cell.ThemeColor = 1
            Case 3: Cell.ThemeColor = 4
            Case 4: Cell.ThemeColor = 3
            Case Else
                Cell.ThemeColor = Shape.ObjectThemeColor
        End Select
        Cell.TintAndShade = Shape.Brightness
    End If
End Sub
```

Skoroszyt z tym przykładem (*Dopasowywanie kolorów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Modyfikacja kolorów wykresów

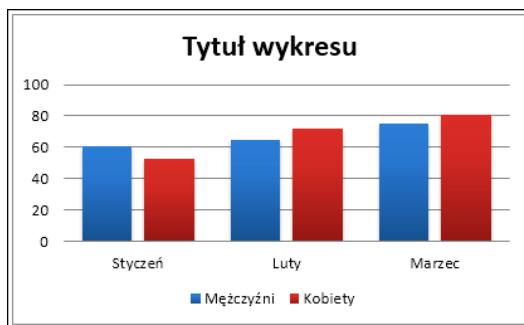
W tym podrozdziale omówię zagadnienia związane ze zmianą kolorów wykresów. Kluczowym elementem takiej operacji jest poprawne zidentyfikowanie elementu, którego kolorystykę chcesz zmodyfikować. Innymi słowy, aby zmienić kolorystykę wykresu, najpierw musisz zidentyfikować żądany obiekt, a potem zmodyfikować odpowiednie właściwości.

Na rysunku 28.10 przedstawiono prosty wykres kolumnowy o nazwie *Wykres 1*. Na wykresie znajdują się dwie serie danych, legenda i tytuł wykresu.

Poniżej przedstawiono polecenie VBA, które zmienia kolor pierwszej serii danych wykresu na czerwony.

```
ActiveSheet.ChartObjects("Wykres 1").Chart.
    SeriesCollection(1).Format.Fill.ForeColor.RGB = vbRed
```

Rysunek 28.10.
Prosty wykres
kolumnowy



Na pierwszy rzut oka polecenie robi wrażenie bardzo skomplikowanego, ponieważ używa bardzo złożonej struktury obiektów. Hierarchia obiektów użyta w tym poleceniu jest następująca:

Aktywny skoroszyt zawiera kolekcję obiektów ChartObjects. Jednym z obiektów tej kolekcji jest ChartObject o nazwie *Wykres 1*. Właściwość Chart obiektu ChartrObject zwraca obiekt Chart, który posiada kolekcję SerieCollection, zawierającą obiekt Series o indeksie równym 1. Właściwość Format obiektu Series zwraca obiekt ChartFormat. Właściwość Fill obiektu ChartFormat zwraca obiekt FillFormat. Właściwość ForeColor obiektu FillFormat zwraca obiekt ColorFormat. Właściwość RGB obiektu ColorFormat jest ustalana na wartość reprezentującą kolor czerwony.



Więcej szczegółowych informacji na temat pracy z wykresami przy użyciu VBA znajdziesz w rozdziale 16.

Innym sposobem zapisania polecenia przedstawionego poniżej jest użycie zmiennych obiektowych reprezentujących poszczególne obiekty (i być może bardziej przejrzyście ilustrujących relacje pomiędzy obiekttami):

```
Sub ChangeSeries1Color
    Dim MyChartObject As ChartObject
    Dim MyChart As Chart
    Dim MySeries As Series
    Dim MyChartFormat As ChartFormat
    Dim MyFillFormat As FillFormat
    Dim MyColorFormat As ColorFormat

    ' Tworzenie zmiennych obiektowych
    Set MyChartObject = ActiveSheet.ChartObjects("Wykres 1")
    Set MyChart = MyChartObject.Chart
    Set MySeries = MyChart.SeriesCollection(1)
    Set MyChartFormat = MySeries.Format
    Set MyFillFormat = MyChartFormat.Fill
    Set MyColorFormat = MyFillFormat.ForeColor

    ' Zmiana koloru
    MyColorFormat.RGB = vbRed
End Sub
```

Właściwość RGB akceptuje dziesiętną wartość koloru, która została określona przy użyciu wbudowanej stałej VBA.

Inne właściwości obiektu `ColorFormat` związane z kolorami są takie same, jak w przypadku kształtów (mówiliśmy o nich nieco wcześniej w tym rozdziale). Poniżej zamieszczamy krótkie zestawienie takich właściwości:

- `Brightness` — liczba z zakresu od -1 do 1, reprezentująca jaskrawość koloru. Wartość -1 odpowiada kolorowi czarnemu, a wartość 1 kolorowi białemu.
- `ObjectThemeColor` — liczba z zakresu od 1 do 15, reprezentująca kolor motywu.
- `TintAndShade` — liczba z zakresu od -1 do 1, reprezentująca jaskrawość bazowego koloru motywu.



W sieci

Skoroszyt z omawianymi wyżej przykładami (*Kolory wykresów.xlsxm*) znajdziesz na stronie internetowej naszej książki (patrz <http://www.helion.pl/ksiazki/e23pvw.htm>).



Jeżeli używasz języka VBA do ustawiania koloru dowolnego elementu wykresu, taki kolor zostanie nadany, ale nie zmieni się, kiedy użytkownik wybierze inny motyw dokumentu. Kiedy kod VBA próbuje zmodyfikować wartość właściwości `ObjectThemeColor`, to zawsze właściwość ta automatycznie przyjmuje ponownie wartość 0, co oznacza, że taki kolor nie jest sterowany poprzez motyw dokumentu. Jest to dobrze znany błąd w Excelu, który jak widać nie został poprawiony również w wersji 2013.

W języku VBA możesz również definiować gradienty kolorów. Poniżej przedstawiono kod procedury, która nadaje predefiniowane wypełnienie gradientowe drugiej serii danych wykresu. Zwróć uwagę na fakt, że gradient jest nadawany przy użyciu obiektu `FillFormat`:

```
Sub AddPresetGradient()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.ChartObjects("Wykres 1").Chart
    With MyChart.SeriesCollection(1).Format.Fill
        .PresetGradient
        Style:=msoGradientHorizontal, _
        Variant:=1, _
        PresetGradientType:=msoGradientFire
    End With
End Sub
```

Praca z innymi elementami wykresów jest bardzo podobna. Procedura, której kod przedstawiono poniżej, zmienia kolory obszaru wykresu i obszaru kreślenia na kolory aktywnego motywu.

```
Sub RecolorChartAndPlotArea()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.ChartObjects("Wykres 1").Chart
    With MyChart
        .ChartArea.Format.Fill.ForeColor.ObjectThemeColor = _
            msoThemeColorAccent6
        .ChartArea.Format.Fill.ForeColor.TintAndShade = 0.9
        .PlotArea.Format.Fill.ForeColor.ObjectThemeColor = _
            msoThemeColorAccent6
        .PlotArea.Format.Fill.ForeColor.TintAndShade = 0.5
    End With
End Sub
```

W ostatnim przykładzie tego podrozdziału przedstawiamy procedurę, która nadaje poszczególnym elementom wykresu losowe kolory. Użycie tej procedury praktycznie gwarantuje, że otrzymany wykres będzie wyglądał naprawdę paskudnie, ale zadaniem tej procedury jest po prostu ilustracja sposobu zmiany kolorów innych elementów wykresu. Procedura UseRandomColors do wybrania losowego koloru wykorzystuje funkcję RandomColor:

```
Sub UseRandomColors()
    Dim MyChart As Chart
    Set MyChart = ActiveSheet.ChartObjects("Wykres 4").Chart
    With MyChart
        .ChartArea.Format.Fill.ForeColor.RGB = RandomColor
        .PlotArea.Format.Fill.ForeColor.RGB = RandomColor
        .SeriesCollection(1).Format.Fill.ForeColor.RGB = RandomColor
        .SeriesCollection(2).Format.Fill.ForeColor.RGB = RandomColor
        .Legend.Font.Color = RandomColor
        .ChartTitle.Font.Color = RandomColor
        .Axes(xlValue).MajorGridlines.Border.Color = RandomColor
        .Axes(xlValue).TickLabels.Font.Color = RandomColor
        .Axes(xlValue).Border.Color = RandomColor
        .Axes(xlCategory).TickLabels.Font.Color = RandomColor
        .Axes(xlCategory).Border.Color = RandomColor
    End With
End Sub

Function RandomColor()
    RandomColor = Application.RandBetween(0, RGB(255, 255, 255))
End Function
```


Rozdział 29.

Często zadawane pytania na temat programowania w Excelu

W tym rozdziale:

- Dziwne problemy z Excelem, których można i których nie można uniknąć
- Często zadawane pytania na temat zagadnień związanych z programowaniem
- Wskazówki dotyczące pracy z edytorem VBE

FAQ — czyli często zadawane pytania

Czytelnicy, którzy często korzystają z sieci Internet, niewątpliwie wiedzą, co to są listy często zadawanych pytań (ang. *FAQ — Frequently Asked Questions*). Zauważylem, że użytkownicy Excela zadają podobne pytania dotyczące programowania, utworzyłem więc swoją listę FAQ.

Z pewnością lista nie zawiera odpowiedzi na wszystkie pytania, ale mimo wszystko zawiera zestaw najczęściej zadawanych pytań i być może pomoże Ci w wyjaśnieniu wielu napotkanych problemów.

Aby nieco uporządkować listę, podzieliłem pytania na następujące kategorie tematyczne:

- Ogólne pytania dotyczące programu Excel
- Edytor VBE
- Procedury Sub
- Procedury Function
- Obiekty, właściwości, metody i zdarzenia
- Zagadnienia związane z bezpieczeństwem
- Formularze *UserForm*

- Dodatki
- Modyfikacje interfejsu programu Excel

W wielu przypadkach umieszczenie pytania w określonej kategorii jest czysto umowne i równie dobrze można by je przypisać do innej kategorii. Pytania w obrębie poszczególnych kategorii nie są wyszczególnione w jakimś konkretnym porządku.

Większość zagadnień poruszanych w tym rozdziale została szczegółowo omówiona w innych rozdziałach tej książki.

Ogólne pytania dotyczące programu Excel

Jak zarejestrować makro?

Kliknij lewym przyciskiem myszy małą, kwadratową ikonę znajdującą się w lewej części paska stanu.

Jak uruchomić makro?

Przejdź na kartę *WIDOK* i naciśnij przycisk *Makra*, znajdujący się w grupie opcji *Makra*. Zamiast tego możesz również nacisnąć przycisk *Makra* znajdujący się na karcie *DEVELOPER* w grupie opcji *Kod* lub skorzystać z kombinacji klawiszy *lewy Alt+F8*.

Co mam zrobić, jeżeli karta **DEVELOPER** nie jest widoczna?

Kliknij Wstążkę prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Dostosuj Wstążkę*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel* otwarte na karcie *Dostosowywanie Wstążki*. Na liście *Karty główne* odszukaj i zaznacz opcję *DEVELOPER*.

Zarejestrowałem nowe makro i zapisałem skoroszyt na dysku.

Po ponownym otwarciu skoroszytu okazało się, że wszystkie makra zniknęły! Co się z nimi stało?

Domyślnie podczas pierwszego zapisywania pliku Excel proponuje usunięcie wszystkich makr i zapisanie samego skoroszytu. Podczas zapisywania skoroszytów powinieneś dokładnie czytać treść pojawiających się na ekranie komunikatów i automatycznie nie naciskać domyślnego przycisku *Tak*. Jeżeli Twój skoroszyt zawiera makra, musisz zapisać go w formacie XLSM, a nie w formacie XLSX.

Co zrobić, jeżeli nie znajdziesz tutaj odpowiedzi na swoje pytanie?

Większość zagadnień poruszonych w tym rozdziale zastała opisana szczegółowo w innych rozdziałach. Jeżeli więc tu nie znajdziesz odpowiedzi na pytanie, zajrzyj do indeksu. Książka zawiera mnóstwo informacji, które nie zostały sklasyfikowane jako pytania FAQ. Jeżeli mimo wszystko nadal nie znajdziesz odpowiedzi, powinieneś spróbować poszukać rozwiązań w sieci Internet.

Jak ukryć Wstążkę, aby nie zajmowała na ekranie tyle miejsca?

Jednym ze sposobów może być użycie polecenia *Opcje wyświetlania Wstążki*, którego przycisk znajduje się po prawej stronie paska tytułowego okna Excela 2013, zaraz za ikoną przycisku pomocy. Naciśnięcie tego przycisku daje Ci do wyboru trzy możliwe ustawienia Wstążki.

Jeżeli chcesz szybko ukryć Wstążkę, powinieneś po prostu dwukrotnie kliknąć lewym przyciskiem myszy dowolną kartę Wstążki. Aby wyświetlić Wstążkę, kliknij wybraną kartę lewym przyciskiem myszy. By przywrócić na stałe wyświetlanie Wstążki, ponownie dwukrotnie kliknij dowolną kartę Wstążki lewym przyciskiem myszy.

Zamiast tego możesz również nacisnąć kombinację klawiszy *Ctrl+F1*, która działa jak przełącznik — na przemian włącza i wyłącza wyświetlanie Wstążki.

Jeżeli chcesz całkowicie usunąć Wstążkę z ekranu, powinieneś skorzystać z następującego makra XLM:

```
ExecuteExcel14Macro "SHOW.TOOLBAR(""Ribbon"",False)"
```

Po wykonaniu tego polecenia użytkownik nie może włączyć wyświetlania Wstążki. Jedynym sposobem na przywrócenie wyświetlania Wstążki jest ponowne wykonanie powyższego makra XLM, tym razem z ostatnim argumentem ustawnionym na wartość TRUE.

Gdzie podziały się moje stare paski narzędzi?

Jeżeli paski narzędzi są wyświetlane automatycznie, przejdź na kartę *DODATKI*, gdzie znajdziesz je w grupie opcji *Paski narzędzi*.

Czy mogę nadal korzystać z moich starych, „pływających” pasków narzędzi?

Niestety nie. Wszystkie stare paski narzędzi zostały na stałe „uziemione” na karcie *DODATKI* w grupie opcji *Paski narzędzi*.

Jak mogę ukryć pasek stanu?

Aby ukryć pasek stanu, musisz użyć VBA. Polecenie przedstawione poniżej powinno załatwić sprawę:

```
Application.DisplayStatusBar = False
```

Czy istnieje jakieś narzędzie, które potrafi zamienić moją aplikację programu Excel w samodzielny, wykonywalny plik EXE?

Nie.

Dlaczego naciśnięcie kombinacji klawiszy Ctrl+A nie zaznacza wszystkich komórek mojego arkusza?

Najprawdopodobniej dzieje się tak dlatego, że wskaźnik komórki znajduje się w obrębie tabeli. Kiedy aktywna komórka znajduje się w tabeli, to aby zaznaczyć wszystkie komórki arkusza, musisz nacisnąć kombinację klawiszy *Ctrl+A* aż trzykrotnie. Pierwsze naciśnięcie zaznacza komórkę danych, drugie komórkę danych oraz wiersz nagłówków, a dopiero trzecie naciśnięcie tej kombinacji zaznacza wszystkie komórki arkusza.

Jeżeli aktywna komórka znajduje się w bloku komórek, naciśnięcie kombinacji klawiszy *Ctrl+A* zaznacza cały zakres. Ponowne naciśnięcie kombinacji klawiszy *Ctrl+A* zaznaczy wszystkie komórki w całym arkuszu.

Dlaczego polecenie Widoki niestand. nie jest aktywne?

Prawdopodobnie dzieje się tak dlatego, że w Twoim arkuszu znajduje się tabela. Po zamianie tabeli na zakres komórek będziesz mógł ponownie używać polecenia *Widoki niestand.* (karta *WIDOK*, grupa opcji *Widok skoroszytu*). Nikt (z wyjątkiem oczywiście programistów firmy Microsoft) nie wie, dlaczego tak się dzieje.

Jak umieścić listę rozwijaną w komórce, tak aby użytkownik mógł wybierać odpowiednie wartości z listy?

Takie rozwiązanie nie wymaga tworzenia żadnego makra. Aby to zrobić, powinieneś umieścić listę elementów w osobnej kolumnie (w razie potrzeby możesz tę kolumnę później ukryć). Zaznacz komórkę (lub komórki), w której chcesz umieścić listę, i wybierz polecenie *Poprawność danych*, znajdujące się na karcie *DANE* w grupie opcji *Narzędzia danych*. Na ekranie pojawi się okno dialogowe *Sprawdzanie poprawności danych*. Przejdź na kartę *Ustawienia* i z listy rozwijanej *Dozwolone* wybierz opcję *Lista*. W polu *Źródło* wpisz adres zakresu komórek zawierającego elementy listy. Upewnij się, że opcja *Rozwinienia w komórce* jest zaznaczona. Jeżeli lista dozwolonych wartości jest krótka, możesz ją po prostu wpisać bezpośrednio w polu *Źródło*, oddzielając od siebie poszczególne elementy średnikami.

Używam właściwości Application.Calculation do ustawienia trybu przeliczania arkusza na ręczny. Wygląda jednak na to, że zmiana ma zastosowanie do wszystkich skoroszytów, a nie tylko do skoroszytu aktywnego.

Właściwość *Calculation* jest właściwością obiektu *Application*, stąd zmiany trybu przeliczania arkusza odnoszą się do wszystkich skoroszytów. Nie możesz zmienić trybu przeliczania tylko dla jednego, wybranego skoroszytu. W Excelu 2000 i wersjach późniejszych masz do dyspozycji właściwość *EnableCalculation* obiektu *Property*. Kiedy właściwość ta jest ustalona na wartość *False*, skoroszyt nie będzie przeliczany, nawet jeżeli użytkownik będzie próbował wymusić przeliczenie ręcznie. Aby przywrócić możliwość przeliczania, musisz ustawić tę właściwość na wartość *True*.

Co się stało z możliwością odczytywania na głos zawartości komórek?

Aby skorzystać z tego mechanizmu, musisz odpowiednio dostosować pasek narzędzi *Szybki dostęp* lub *Wstążkę*. Najlepiej dokonać tej operacji w oknie opcji programu Excel. Polecenia odczytujące na głos zawartość komórek znajdują się w grupie opcji *Polecenia, których nie ma na Wstążce* (wszystkie polecenia „głosowe” rozpoczynają się od słowa *Czytaj*).

Otworzyłem skoroszyt i okazało się, że arkusze mają tylko po 65 546 wierszy. Co się stało?

Domyślnie skoroszyty programu Excel 2007 i wszystkich nowszych wersji zawierają 1 048 576 wierszy i 16 384 kolumny. Jeżeli nie widzisz takiej ilości wierszy i kolumn, to znaczy, że skoroszyt został otwarty w trybie zgodności. Kiedy Excel otwiera skoroszyt zapisany w formacie poprzednich wersji Excela, nie dokonuje automatycznej konwersji na format skoroszytu nowych wersji Excela (2007 i nowszych). Musisz to zrobić ręcznie. Aby tego dokonać, zapisz skoroszyt w nowym formacie (*.xlsx lub *.xlsm), zamknij go i następnie ponownie otwórz. Od tego momentu będziesz miał do dyspozycji całe mnóstwo dodatkowych wierszy i kolumn.

Jak mogę w starych skoroszytach używać nowych czcionek?

Począwszy od wersji Excel 2007, nowa czcionka domyślna jest zdecydowanie bardziej czytelna niż w poprzednich wersjach. Aby wymusić używanie nowej czcionki w starych skoroszytach, utwórz nowy skoroszyt, naciskając kombinację klawiszy *Ctrl+N*. Aktywuj stary skoroszyt i przejdź na kartę *NARZĘDZIA GŁÓWNE*. Kliknij polecenie *Style komórki* i z menu podręcznego wybierz polecenie *Scal style*. Na ekranie pojawi się okno dialogowe *Scalanie stylów*. Na liście *Scalaj style* z dwukrotnie kliknij lewym przyciskiem myszy nowy skoroszyt (ten, który utworzyłeś, naciskając kombinację klawiszy *Ctrl+N*). Style starego skoroszytu zostaną zastąpione nowymi. Opisana metoda działa jedynie z komórkami, które nie zostały sformatowane przy użyciu innych atrybutów czcionek (na przykład komórki, w których czcionka została wcześniej pogrubiona, pozostaną nienaruszone). W takich komórkach będziesz musiał zmienić formatowanie ręcznie.

Jak wyświetlić podgląd wydruku?

W Excelu podgląd wydruku włączany jest automatycznie po wybraniu polecenia *PLIK/Drukuj*. Innym rozwiązaniem jest wykorzystanie widoku *Układ strony* (przycisk tego polecenia znajdziesz w szeregu ikon w prawej części paska stanu).

Aby włączyć starą wersję podglądu wydruku, musisz skorzystać z VBA. Polecenie przedstawione poniżej wyświetla podgląd wydruku aktywnego arkusza:

```
ActiveSheet.PrintPreview
```

Kiedy włączam nowy motyw dokumentu, mój arkusz nie mieści się już dłużej na jednej stronie. Dlaczego tak się dzieje?

Dzieje się tak prawdopodobnie dlatego, że różne motywy arkuszy korzystają z różnych czcionek. Po nadaniu nowego motywu możesz wybrać polecenie *Czcionki* (karta *UKŁAD STRONY*, grupa opcji *Motyw*) i przywrócić oryginalne czcionki, tak aby były używane

przez ten motyw. Zamiast tego możesz również zmienić rozmiar czcionki dla stylu *Normalny*. Jeżeli dopasowanie arkusza do strony jest sprawą krytyczną, powinieneś wybrać docelowy motyw jeszcze przed rozpoczęciem projektowania arkusza.

Jak pozbyć się irytującego, kropkowanego oznaczenia podziałów strony po przełączeniu do widoku normalnego?

Otwórz okno dialogowe *Opcje programu Excel*, kliknij kategorię *Zaawansowane*, przewiń prawy panel w dół, aż do kategorii *Opcje wyświetlania dla tego arkusza* i usuń zaznaczenie opcji *Pokaż podziały stron*.

Czy mogę dodać polecenie Pokaż podziały stron do paska narzędzi Szybki dostęp lub Wstążki?

Nie. Z jakiegoś powodu ikony tego użytkelnego polecenia nie można umieścić na pasku narzędzi *Szybki dostęp* ani na *Wstążce*. Aby wyłączyć wyświetlanie podziału stron, możesz użyć następującego polecenia VBA:

```
ActiveSheet.DisplayPageBreaks = False
```

Próbuje nadać tabeli nowy styl formatowania, ale nie przynosi to żadnego widocznego efektu. Co mogę zrobić w takiej sytuacji?

Dzieje się tak prawdopodobnie dlatego, że komórki tabeli zostały wcześniej sformatowane ręcznie. Zaznacz komórki i ustaw kolor wypełnienia na *Brak wypełnienia* oraz kolor czcionki na *Automatyczny*. Od tej chwili formatowanie przy użyciu stylów tabeli powinno znowu działać poprawnie.

Czy mogę zmienić kolor karty arkusza?

Kliknij kartę arkusza prawym przyciskiem myszy i z menu podręcznego wybierz polecenie *Kolor karty*. Pamiętaj, że po wybraniu innego motywu dokumentu kolory kart arkuszy również ulegną zmianie.

Czy mogę utworzyć makro VBA, które odtwarzają dźwięki?

Tak, za pomocą procedur VBA możesz odtwarzać pliki WAV i MIDI, ale musisz do tego celu użyć odpowiednich wywołań funkcji Windows API (ang. *Windows Application Programming Interface*). Innym rozwiązaniem jest skorzystanie z metod obiektu Speech. Na przykład polecenie przedstawione poniżej pozwoli użytkownika po imieniu:

```
Application.Speech.Speak ("Hello" & Application.UserName)
```

Kiedy otwieram skoroszyt, Excel pyta, czy chcę zaktualizować łącza. Przeszukałem wszystkie formuły i nie mogę znaleźć żadnego łącza w tym skoroszycie. Czy to błąd Excela?

Prawdopodobnie jednak nie. Spróbuj wykonać polecenie *PLIK/Informacje/Edytuj łącza do plików*. Na ekranie pojawi się okno dialogowe *Edytowanie łączy*. Wybierz opcję *Przerwij łącze*. Pamiętaj, że łącza mogą występować nie tylko w formułach, ale również w innych

miejscach. Jeżeli w skoroszycie masz wykres, kliknij kolejno wszystkie serie danych i sprawdź na pasku formuły, jak wygląda funkcja SERIE. Jeżeli okaże się, że któraś seria danych odwołuje się do innego skoroszytu, znalazłeś zagubione łącze. Aby wyeliminować takie łącza, przenieś dane do bieżącego skoroszytu i utwórz nowy wykres.

Jeżeli skoroszyt zawiera arkusze dialogowe Excela 5/95, zaznacz kolejno poszczególne obiekty na arkuszu i sprawdź powiązane z nimi formuły na pasku formuł. Jeżeli którykolwiek obiekt zawiera odwołanie do innego skoroszytu, zmodyfikuj lub usuń to odwołanie.

Przejdz na kartę *FORMUŁY* i naciśnij przycisk *Menedżer nazw*, znajdujący się w grupie opcji *Nazwy zdefiniowane*. Na ekranie pojawi się okno dialogowe menedżera nazw. Przyjrzyj się liście zdefiniowanych nazw i sprawdź kolumnę *Odwołuje się do*. Usuń nazwy, które odwołują się do innych skoroszytów lub które zawierają nieprawidłowe odwołania (takie jak błąd #ADR!). To najczęściej występujące przyczyny powstawania takich „zagubionych” łączy.

Gdzie mogę znaleźć przykłady procedur VBA?

W sieci Internet można znaleźć tysiące stron z przykładami programowania w języku VBA. Na dobry początek możesz rozpocząć od strony internetowej autora tej książki, <http://spreadsheetpage.com/>. Możesz również skorzystać z jednej z wielu wyszukiwarek sieciowych, na przykład popularnego serwisu Google: <http://www.google.pl>.

Pytania dotyczące edytora Visual Basic

Czy można wykorzystać rejestrator makr VBA do zarejestrowania wszystkich rodzajów makr?

Nie. Właściwie rejestrator makr nadaje się tylko do bardzo prostych makr. Takich makr, w których wykorzystuje się zmienne, pętle oraz inne mechanizmy modyfikacji przepływu sterowania, nie można zarejestrować. Nie możesz również rejestrować funkcji. Można jednak wykorzystać rejestrator makr w celu zapisania elementów kodu lub w celu zapoznania się z odpowiednimi właściwościami i metodami obiektów.

Zdefiniowałem kilka makr ogólnego przeznaczenia. Chciałbym, aby były dostępne przez cały czas. W jaki sposób najlepiej to zrobić?

Spróbuj zapisać te makra w *osobistym arkuszu makr* (ang. *Personal Macro Workbook*). W normalnych warunkach jest to ukryty skoroszyt ładowany automatycznie przez Excela. W czasie rejestrowania makra istnieje opcja jego zarejestrowania w osobistym arkuszu makr. Plik *Personal.xlsb* jest domyślnie przechowywany w katalogu *|XLStart*.

Nie mogę odnaleźć osobistego arkusza makr. Gdzie on się podzielił?

Plik *Personal.xlsb* nie istnieje, dopóki nie zarejestrujemy w nim dowolnego makra i nie zamkniesz programu Excel.

Kiedy wstawiam nowy moduł, zawsze rozpoczyna się on od wiersza Option Explicit. Co to oznacza?

Jeżeli na początku modułu znajduje się instrukcja Option Explicit, przed użyciem dowolnej zmiennej najpierw trzeba ją zadeklarować (co jest dobrą praktyką). Aby wyłączyć wstawianie tego wiersza w nowych modułach, należy uaktywnić edytor *Visual Basic*, wybrać polecenie *Tools/Options*, kliknąć zakładkę *Editor* i wyczyścić pole wyboru *Require Variable Declaration*. Po wykonaniu tej czynności można zadeklarować zmienne ręcznie lub pozostawić obsługę typów danych systemowi VBA.

Dlaczego kod VBA jest wyświetlany w różnych kolorach? Czy mogę je zmienić?

W systemie VBA kolory służą do wyróżnienia różnego typu tekstów: komentarzy, słów kluczowych, identyfikatorów, instrukcji zawierających błędy itd. Kolory można zmienić poprzez wybranie polecenia *Tools/Options* (zakładka *Editor Format*) w edytorze Visual Basic.

Chcę usunąć moduł VBA za pomocą kodu. Czy mogę to zrobić?

Tak. Poniższy kod powoduje usunięcie modułu *Module1* z aktywnego skoroszytu:

```
With ActiveWorkbook.VBProject  
    .VBComponents.Remove .VBComponents("Module1")  
End With
```

Czasami jednak mogą wystąpić problemy z wykonaniem takiego polecenia. Zobacz odpowiedź na następne pytanie.

Napisałem makro, które dodaje kod VBA do projektu VB. Kiedy kolega próbuje je uruchomić, wyświetla się komunikat o błędzie. W czym problem?

Excel ma opcję, która odpowiada za to, czy kod VBA może modyfikować zawartość projektu VBA: *Ufaj dostępowi do programu Visual Basic Project*. Domyslnie to ustawienie jest wyłączone. Aby zmienić to ustawienie, wybierz polecenie *PLIK/Opcje/Centrum zaufania*. Naciśnij przycisk *Ustawienia Centrum zaufania*. Na ekranie pojawi się okno dialogowe *Centrum zaufania*. Kliknij kategorię *Ustawienia makr* i zaznacz opcję *Ufaj dostępowi do modelu obiektowego projektu VBA*.

W jaki sposób napisać makro zmieniające zabezpieczenia makr ustawione przez użytkownika? Chciałbym uniknąć wyświetlania komunikatu „Ten skoroszyt zawiera makra” w momencie otwierania mojej aplikacji.

Nie ma takiej możliwości. Gdyby była, cały system zabezpieczania makr byłby po prostu bezużyteczny. Pomyśl o tym!

Nie rozumiem, jak działa opcja UserInterfaceOnly w kontekście zabezpieczania arkusza

Zabezpieczając arkusz, można wykorzystać następującą instrukcję:

```
ActiveSheet.Protect UserInterfaceOnly:=True
```

W ten sposób arkusz zostanie zabezpieczony, ale w dalszym ciągu będzie można wprowadzać w nim zmiany. Należy pamiętać, że to ustawienie nie jest zapisywane w skoroszycie. Po ponownym otwarciu skoroszytu, aby włączyć zabezpieczenie UserInterfaceOnly, należy jeszcze raz wykonać tę instrukcję.

W jaki sposób można stwierdzić, czy skoroszyt zawiera wirusa w makrze?

Należy uaktywnić w edytorze Visual Basic projekt związany ze skoroszytem, sprawdzić wszystkie moduły kodu (również moduł kodu obiektu `ThisWorkbook`) i przekonać się, czy nie zawierają nieznanego kodu VBA. Zazwyczaj kod wirusa nie jest dobrze sformowany i zawiera mnóstwo dziwnych nazw zmiennych. Można też wykorzystać komercyjny skaner antywirusowy.

Mam problem z operatorem konkatenacji (&) w VBA.

Podczas próby połączenia dwóch łańcuchów znaków wyświetlany jest komunikat o błędzie.

Prawdopodobnie system VBA interpretuje znak & jako znak deklaracji typu. Upewnij się, czy wstawiłeś spację przed i po operatorze konkatenacji.

Wydaje mi się, że operator kontynuacji wiersza w języku VBA (znak podkreślenia) nie działa.

Operator kontynuacji wiersza składa się tak naprawdę z dwóch znaków: spacji i znaku podkreślenia. Jeżeli pominiesz tę spację, całość po prostu nie będzie działać.

Z mojej aplikacji Excela korzysta wielu użytkowników. Na niektórych komputerach nie działają procedury obsługi błędów VBA. Dlaczego tak się dzieje?

Procedury obsługi błędów nie będą działać, jeżeli w edytorze Visual Basic użytkownik włączy opcję *Break on All Errors* (w zakładce *General* okna dialogowego *Options*). Nieustety, nie można zmienić tego ustawienia za pomocą kodu VBA.

Inną możliwą przyczyną może być to, że użytkownik zainstalował dodatek *Narzędzie do waluty euro* (ang. *Euro Currency Tools add-in*). Starsze wersje tego dodatku są związane z tego, że powodowały problemy i niepożądane interakcje z innymi dodatkami.

Pytania dotyczące procedur

Jaka jest różnica pomiędzy procedurą VBA a makrem?

Nie ma żadnej różnicy. Pojęcie „makro” pochodzi z czasów poprzednich wersji. Obecnie tych pojęć można używać zamiennie.

Czym jest procedura?

Procedura jest grupą instrukcji VBA, które można wywołać za pomocą nazwy. Jeżeli instrukcje mają zwracać jawne wyniki do bloku instrukcji, w którym nastąpiło ich wywołanie, należy utworzyć z nich funkcję (procedurę typu Function). W innym przypadku można wykorzystać procedurę Sub.

Czym jest typ danych Variant?

Zmiennym, które nie są zadeklarowane jawnie, domyślnie przypisuje się typ danych Variant, a system VBA automatycznie przekształca dane na właściwy typ w momencie ich użycia. Szczególnie ten typ danych nadaje się do pobierania wartości z arkusza, kiedy nie wiadomo, jaka jest zawartość komórki. Lepiej jednak, jeżeli przed użyciem zadeklarujemy zmienne jawnie za pomocą instrukcji Dim, Public lub Private, ponieważ kod ze zmiennymi typu Variant jest nieco wolniejszy i zużywa więcej pamięci.

Jaka jest różnica pomiędzy tablicą typu Variant a tablicą zawierającą dane typu Variant?

Dane typu Variant są jednostkami pamięci specjalnego typu, które mogą zawierać dowolne dane: pojedynczą wartość lub tablicę wartości (tzn. tablicę typu Variant). Poniższy kod tworzy zmienną typu Variant zawierającą tablicę składającą się z trzech elementów:

```
Dim X As Variant  
X = Array(30, 40, 50)
```

Zwykła tablica może zawierać elementy określonego typu danych, w tym zmienne typu Variant, dla których nie określono typu. Poniższa instrukcja tworzy tablicę z 3 wartościami typu Variant:

```
Dim X (0 To 2) As Variant
```

Chociaż zmienna typu Variant zawierająca tablicę pojęciowo różni się od tablicy, której elementy są danymi typu Variant, dostęp do elementów tablic uzyskuje się w taki sam sposób.

Co to jest znak definicji typu?

W języku VBA można dołączyć do nazwy zmiennej znak, który wskazuje na jej typ. Na przykład można zadeklarować zmienną MyVar jako dane typu Integer poprzez dołączenie znaku % do nazwy zmiennej:

```
Dim MyVar%
```

Oto nazwy typów wraz z podanymi w nawiasie znakami deklaracji, które są obsługiwane w języku VBA:

- INTEGER (%)
- LONG (&)
- SINGLE (!)
- DOUBLE (#)
- CURRENCY (@)
- STRING (\$)

Znaki definicji typu są obsługiwane głównie ze względu na konieczność zachowania kompatybilności ze starszymi wersjami VBA. Obecnie ogólnie przyjętym standardem jest deklarowanie typu zmiennych za pomocą odpowiednich słów kluczowych.

**Chciałbym utworzyć procedurę, która automatycznie modyfikuje formatowanie komórki na podstawie wprowadzonych danych.
Jeżeli na przykład wprowadzę wartość większą od zera,
to komórki ma być czerwone. Czy to możliwe?**

Tak, i w dodatku nie trzeba pisać żadnego kodu. Należy wykorzystać właściwość formatowania warunkowego dostępną w Excelu po wybraniu polecenia *NARZĘDZIA GŁÓWNE/Style/Formatowanie warunkowe*.

**Funkcja formatowania warunkowego jest przydatna,
ale chciałbym również wykonać inne działania w czasie,
kiedy w komórce są wprowadzane dane.**

W takim przypadku można wykorzystać zdarzenie Change obiektu Worksheet, które jest generowane przy każdej modyfikacji zawartości komórki. Jeżeli moduł kodu obiektu Sheet zawiera procedurę Worksheet_Change, będzie ona wykonana automatycznie przy zmianie zawartości komórki.

Jakie inne rodzaje zdarzeń można monitorować?

Mnóstwo! Pełną listę zdarzeń można znaleźć w systemie pomocy.

**Próbowałem zdefiniować procedurę obsługi zdarzenia
(Sub Workbook_Open), ale procedura nie wykonuje się
podczas otwierania skoroszytu. W czym problem?**

Prawdopodobnie została umieszczona w złym miejscu. Procedury obsługi zdarzeń poziomu skoroszytu należy zapisywać w module kodu obiektu ThisWorkbook. Procedury obsługi zdarzeń poziomu arkusza należy umieścić w module kodu odpowiedniego obiektu Sheet zgodnie z tym, co wyświetla się w oknie Project edytora Visual Basic.

Inną przyczyną może być fakt, że wykonywanie makr zostało zablokowane. Sprawdź ustawienia makr w *Centrum zaufania* (możesz do niego przejść za pomocą okna *Opcje programu Excel*).

Wiem, że można napisać procedurę obsługi zdarzenia dla określonego skoroszytu, ale czy można napisać procedurę obsługi zdarzenia działającą dla dowolnego otwartego skoroszytu?

Tak, ale należy w tym celu wykorzystać moduł klasy. Zagadnienie to zostało szczegółowo opisane w rozdziale 17.

Potrafię tworzyć formuły w Excelu. Czy w języku VBA są wykorzystywane te same operatory matematyczne i logiczne?

Tak, oraz dodatkowo inne operatory, których nie można stosować w formułach arkuszy. Te dodatkowe operatory VBA wyszczególniono w poniższej tabeli.

Operator	Działanie
\	Dzielenie dające w wyniku liczbę całkowitą
Eqv	Zwraca wartość True, jeżeli oba wyrażenia mają wartość True lub oba mają wartość False
Imp	Logiczna implikacja dwóch wyrażeń
Is	Porównanie dwóch zmiennych obiektowych
Like	Porównanie dwóch łańcuchów znaków z wykorzystaniem znaków wzorca
Xor	Zwraca wartość True, jeżeli tylko jedno z wyrażeń ma wartość True

W jaki sposób można uruchomić procedurę znajdującą się w innym skoroszycie?

Należy wykorzystać metodę Run obiektu Application. Poniższa instrukcja wykonuje procedurę Macro1 umieszczoną w skoroszycie Personal.xlsx:

```
Run "Personal.xlsx!Macro1"
```

Innym rozwiązaniem jest dodanie odwołania do skoroszytu. Aby to zrobić, wybierz z menu głównego edytora VBE polecenie *Tools/References*. Po dodaniu odwołania będziesz mógł uruchamiać procedure ze skoroszytu, do którego zdefiniowano odwołanie, bez konieczności podawania nazwy tego skoroszytu.

Za pomocą języka VBA utworzyłem kilka funkcji. Chciałbym wykorzystać je w formułach arkusza, ale poprzedzanie nazwy funkcji nazwą arkusza jest dla mnie niewygodne. Czy jest jakiś sposób obejścia tego problemu?

Tak. Należy przekształcić skoroszyt zawierający definicję funkcji na dodatek XLAM. Po otwarciu dodatku można korzystać z funkcji w dowolnym arkuszu bez konieczności odwoływania się do nazwy pliku, w którym zdefiniowano funkcje.

Oprócz tego można zdefiniować odwołanie do skoroszytu z funkcjami użytkownika, dzięki czemu nie będzie trzeba poprzedzać nazwy funkcji nazwą skoroszytu. Aby utworzyć odwołanie, użyj polecenia *Tools/References* w edytorze Visual Basic.

Chciałbym, aby pewien skoroszyt był ładowany za każdym razem, kiedy uruchamiam Excela. Chciałbym też, aby makro zapisane w tym skoroszycie było wykonywane automatycznie. Czy wymagam zbyt wiele?

Nie. Aby otwierać skoroszyt automatycznie, wystarczy zapisać go w katalogu *\XLStart*. Aby makro wykonywało się automatycznie, należy utworzyć procedurę *Workbook_Open* w module kodu obiektu *ThisWorkbook* skoroszytu, która będzie wywoływała żądane makro.

Mam skoroszyt, w którym zdefiniowałem procedurę *Workbook_Open*. Czy jest sposób zabezpieczenia się przed wykonywaniem tej procedury w momencie otwierania skoroszytu?

Tak. Należy przytrzymać wciśnięty klawisz *Shift* w czasie otwierania skoroszytu. Aby zabezpieczyć się przed wykonywaniem procedury *Workbook_BeforeClose*, należy wcisnąć klawisz *Shift* w czasie zamknięcia skoroszytu. Wciśnięcie klawisza *Shift* nie zabezpiecza przed wykonywaniem procedur w przypadku otwierania dodatku.

Czy z poziomu procedury VBA można uzyskać dostęp do wartości komórki arkusza, który nie jest otwarty?

W języku VBA nie da się tego zrobić, ale taka operacja jest możliwa w starym języku makr Excela — XLM. Na szczęście z poziomu VBA można wykonywać kod XLM. Poniżej pokazano prosty przykład pobrania wartości z komórki A1 arkusza Arkusz1 w skoroszycie *myfile.xlsx* umieszczonego w katalogu *c:\files*:

```
MsgBox ExecuteExcel4Macro("'c:\files\[myfile.xlsx]Arkusz1'!W1K1")
```

Adres komórki musi być zapisany w notacji *WIK1*.

Jeżeli plik nie ma ogromnych rozmiarów, może się okazać, że łatwiej będzie po prostu otworzyć taki skoroszyt, pobrać potrzebne wartości i go zamknąć. Jeżeli na czas tej operacji wyłączysz odświeżanie ekranu, użytkownik nawet nie zauważycie, że taka operacja miała miejsce.

Jak uniknąć wyświetlania pytania „Czy chcesz zapisać plik...” w momencie zamknięcia skoroszytu z poziomu VBA?

Można wykorzystać następującą instrukcję:

```
ActiveWorkbook.Close SaveChanges:=False
```

Można też ustawić właściwość *Saved* skoroszytu na wartość *True* za pomocą następującej instrukcji:

```
ActiveWorkbook.Saved = True
```

Wykonanie tej instrukcji nie powoduje rzeczywistego zapisania pliku, a zatem po zamknięciu skoroszytu wszystkie niezapisane zmiany będą utracone.

Ogólnym sposobem zabezpieczenia się przed wyświetlaniem komunikatów Excela jest użycie następującej instrukcji:

```
Application.DisplayAlerts = False
```

Po zamknięciu pliku, aby nadal normalnie korzystać z Excela, należy przywrócić wartość True właściwości DisplayAlerts.

Co zrobić, aby makro uruchamiało się co godzinę?

Należy wykorzystać metodę OnTime obiektu Application. Umożliwia ona określenie procedury, która będzie inicjowana o wybranej porze dnia. W ostatniej instrukcji procedury należy ponownie skorzystać z metody OnTime, aby zaplanować kolejne zdarzenie za godzinę.

W jaki sposób nie dopuścić do wyświetlania makra na liście makr?

Aby dane makro nie było widoczne na liście makr w oknie dialogowym *Makro* (by przywołać je na ekran, przejdź na kartę *WIDOK* i naciśnij przycisk *Makra* znajdujący się w grupie poleceń *Makra*), powinieneś zadeklarować taką procedurę ze słowem kluczowym Private:

```
Private Sub MyMacro()
```

Można też zastosować opcjonalny sztuczny argument zadeklarowany z konkretnym typem danych:

```
Sub MyMacro(Optional FakeArg as Integer)
```

Czy można zapisać wykres w formacie GIF?

Tak. Poniższy kod spowoduje zapisanie pierwszego wbudowanego wykresu w arkuszu Arkusz1 w pliku *GIF* o nazwie *Mychart.gif*:

```
Set CurrentChart = Sheets("Arkusz1").ChartObjects(1).Chart  
Fname = ThisWorkbook.Path & "\Mychart.gif"  
CurrentChart.Export Filename:=Fname, FilterName:="GIF"
```

Czy zmienne zadeklarowane w procedurze VBA są dostępne w innych procedurach VBA?

To pytanie dotyczące *zasięgu* zmiennych. Są trzy poziomy zasięgu: lokalny, modułu oraz publiczny. Zmienne lokalne mają najmniejszy zasięg i deklaruje się je wewnątrz procedury. Zmienna lokalna jest widoczna tylko wewnątrz procedury, w której została zadeklarowana. Zmienne poziomu modułu deklaruje się na początku modułu, przed pierwszą procedurą. Zmienne poziomu modułu są widoczne we wszystkich procedurach w module. Zmienne publiczne mają najszerzy zasięg i deklaruje się je za pomocą słowa kluczowego *Public*.

Pytania dotyczące funkcji

Utworzyłem funkcję VBA, która będzie używana w formułach jako funkcja arkuszowa. Niestety próba jej użycia zawsze kończy się błędem #NAZWA? Co jest nie tak?

Prawdopodobnie umieściłeś kod funkcji w module kodu arkusza (na przykład Arkusz1) lub w module kodu obiektu ThisWorkbook. Własne funkcje arkuszowe muszą być definiowane w standardowych modułach kodu VBA.

Napisałem funkcję VBA, która działa bez zarzutu, jeżeli wywołam ją z innej procedury. Funkcja nie działa jednak, gdy zostanie użyta w formule. W czym problem?

Istnieją pewne ograniczenia dotyczące funkcji VBA wywoływanych z formuł arkusza. Ogólnie rzecz biorąc, takie funkcje muszą być ścisłe „pasywne”. Oznacza to, że nie mogą modyfikować aktywnej komórki, stosować formatowania, otwierać skoroszytu lub modyfikować aktywnego arkusza. Jeżeli funkcja spróbuje wykonać takie działania, formuła zwróci błąd.

Utworzyłem funkcję arkusza. Kiedy próbuję z niej skorzystać za pomocą okna dialogowego Wstawianie funkcji, wyświetla się komunikat „Pomoc niedostępna”. Co zrobić, aby w oknie Wstawianie funkcji wyświetlał się opis mojej funkcji?

Aby wprowadzić opis funkcji użytkownika, należy uaktywnić skoroszyt zawierający funkcję i wybrać polecenie WIDOK/Makra/Makra. Na ekranie pojawi się okno dialogowe Makra. Funkcja nie jest wyświetlana na liście, a zatem trzeba wpisać jej nazwę w polu Nazwa makra. Po wpisaniu nazwy makra należy kliknąć przycisk Opcje, aby wyświetlić okno dialogowe Opcje makra, i w polu Opis wprowadzić opisowy tekst.

Czy w oknie dialogowym Wstawianie funkcji można również wyświetlić pomoc na temat argumentów zdefiniowanej funkcji?

Tak. Już w Excelu 2010 został dodany nowy argument metody MacroOptions, który pozwala na zdefiniowanie opisów poszczególnych argumentów funkcji. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 8.

Zdefiniowana przez mnie funkcja wyświetla się w kategorii Użytkownika okna dialogowego Wstawianie funkcji. W jaki sposób spowodować, aby wyświetlała się w innej kategorii?

W tym celu należy skorzystać z kodu VBA. Poniższa instrukcja przypisuje funkcję MyFunc do kategorii 1 (Finansowe):

```
Application.MacroOptions Macro:="MyFunc", Category:=1
```

P pełną listę numerów poszczególnych kategorii funkcji znajdziesz w rozdziale 8.

Jak utworzyć nową kategorię funkcji?

Powinieneś zdefiniować łańcuch tekstu będący argumentem Category metody MacroOptions. Oto przykład:

```
Application.MacroOptions Macro:="MyFunc", Category:="Moje funkcje"
```

**Zdefiniowałem funkcję do wykorzystania w formułach arkusza.
Co należy zrobić, aby funkcja zwracała wartość błędu (#ARG!),
jeżeli użytkownik wprowadzi niepoprawne argumenty?**

Jeżeli funkcja ma nazwę MyFunction, w celu zwrócenia wartości błędu do komórki zawierającej funkcję można wykorzystać poniższą instrukcję:

```
MyFunction = CVErr(xlErrValue)
```

W tym przykładzie xlErrValue jest predefiniowaną stałą. Stałe odpowiadające innym błędom można znaleźć w systemie pomocy.

**W moim programie wykorzystuję wywołania funkcji Windows API
i wszystko działa poprawnie. Przesłałem skoroszyt do kolegi
i u niego występują błędy komplikacji. W czym jest problem?**

Prawdopodobnie Twój kolega korzysta z 64-bitowej wersji Excela. Aby wywołania funkcji API działały poprawnie w 64-bitowej wersji Excela, ich deklaracje muszą zostać uzupełnione klauzulą PtrSafe. Na przykład deklaracja funkcji przedstawiona poniżej będzie działała poprawnie z 32-bitową wersją Excela, ale podczas próby uruchomienia na 64-bitowej wersji Excela 2010 lub Excela 2013 spowoduje wystąpienie błędu komplikacji:

```
Declare Function GetWindowsDirectoryA Lib "kernel32"  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

W wielu przypadkach zapewnienie kompatybilności z 64-bitową wersją Excela jest bardzo proste i wymaga tylko dodania w deklaracji funkcji po słowie kluczowym Declare klauzuli PtrSafe. Dodanie klauzuli PtrSafe działa poprawnie dla zdecydowanej większości powszechnie używanych funkcji API, aczkolwiek niektóre funkcje mogą również wymagać zmiany typu danych argumentów.

Deklaracja przedstawiona poniżej jest kompatybilna zarówno z 32-, jak i 64-bitową wersją Excela 2010 i 2013:

```
Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32"  
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long
```

Z drugiej jednak strony, próba wykonania takiego kodu w Excelu 2007 (i wersjach wcześniejszych) zakończy się wystąpieniem błędu, ponieważ słowo kluczowe PtrSafe nie zostało rozpoznane. Poniżej zamieszczamy przykład użycia odpowiednich dyrektyw kompilatora do zadeklarowania funkcji API, która będzie kompatybilna zarówno z 32-bitowymi wersjami Excela (włączając wersje wcześniejsze niż 2010), jak i z najnowszymi, 64-bitowymi wersjami Excela:

```
#If VBA7 And Win64 Then
    Declare PtrSafe Function GetWindowsDirectoryA Lib "kernel32" -
        (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#Else
    Declare Function GetWindowsDirectoryA Lib "kernel32" -
        (ByVal lpBuffer As String, ByVal nSize As Long) As Long
#End If
```

Pierwsze polecenie `Declare` jest używane tylko wtedy, kiedy zarówno VBA7, jak i Win64 mają wartość True, co ma miejsce tylko i wyłącznie w przypadku 64-bitowej wersji Excela. We wszystkich pozostałych przypadkach używane jest drugie polecenie `Declare`.

Jak wymusić przeliczenie formuł, w których wykorzystano funkcję użytkownika?

Aby wymusić przeliczenie pojedynczej formuły, powinieneś zaznaczyć komórkę zawierającą taką formułę, nacisnąć klawisz *F2* i następnie nacisnąć klawisz *Enter*. Aby wymusić przeliczenie wszystkich formuł i funkcji, powinieneś nacisnąć kombinację klawiszy *Ctrl+Alt+F9*.

Czy można użyć wbudowanych funkcji Excela w kodzie VBA?

W większości przypadków tak. Dostęp do funkcji Excela można uzyskać za pomocą metody `WorksheetFunction` obiektu `Application`. Na przykład funkcję SUMA (ang. SUM) można wykorzystać za pomocą następującej instrukcji:

```
Ans = Application.WorksheetFunction.Sum(Range("A1:A3"))
```

Wykonanie tego przykładu powoduje przypisanie sumy wartości z zakresu A1:A3 (w aktywnym arkuszu) do zmiennej `Ans`.

Ogólnie rzecz biorąc, obowiązuje zasada, że jeżeli w języku VBA jest dostępna funkcja będąca odpowiednikiem funkcji Excela, to nie można wykorzystać wersji funkcji z Excela. Oto przykład: ponieważ w języku VBA jest dostępna funkcja obliczająca pierwiastek kwadratowy z liczby (`Sqr`), w kodzie VBA nie można wykorzystać funkcji Excela `PIERWIATEK`.

Czy jest sposób, aby wymusić koniec wiersza w wyświetlanym oknie informacyjnym MsgBox?

Aby wymusić przejście do nowego wiersza, należy użyć znaku powrotu karetki albo wysuwu wiersza. Poniższa instrukcja wyświetla tekst komunikatu w dwóch wierszach. `vbNewLine` jest wbudowaną stałą reprezentującą znak powrotu karetki.

```
MsgBox "Witaj" & vbNewLine & Application.UserName
```

Pytania dotyczące obiektów, właściwości, metod i zdarzeń

Czy jest gdzieś lista obiektów Excela, z których można korzystać?

Tak. W systemie pomocy znajdziesz diagram ukazujący hierarchię dostępnych obiektów.

Przeraża mnie liczba dostępnych właściwości i metod.

Jak mogę się dowiedzieć, jakie metody i właściwości są dostępne dla określonego obiektu?

Jest kilka sposobów. Można skorzystać z przeglądarki obiektów dostępnej w edytorze *Visual Basic*. Aby ją wyświetlić, należy wcisnąć *F2*, a następnie na rozwijanej liście bibliotek wybrać pozycję *Excel*. Lista *Classes* (z lewej strony ekranu) wyświetla wszystkie obiekty dostępne w Excelu. Zaznaczenie obiektu powoduje, że odpowiadające mu właściwości i metody wyświetlają się na liście *Member* po prawej stronie ekranu.

Można także uzyskać informację o właściwościach i metodach w czasie wpisywania kodu. Na przykład spróbujmy wpisać następujący łańcuch znaków:

```
Range("A1").
```

Wpisanie kropki spowoduje wyświetlenie listy wszystkich właściwości i metod obiektu *Range*. Jeżeli to nie działa, należy wybrać polecenie *Tools/Options*, kliknąć zakładkę *Editor* i zaznaczyć pole wyboru obok pozycji *Auto List Members*. Niestety, mechanizm *Auto List Members* działa nie dla wszystkich obiektów. Na przykład po wpisaniu takiego polecenia lista dostępnych metod i właściwości nie zostanie wyświetlona:

```
ActiveSheet.Shapes(1).
```

Jeżeli jednak wcześniej zadeklarujesz zmienną obiektową, mechanizm *Auto List Members* będzie działał zgodnie z oczekiwaniami. Poniżej przedstawiono przykład deklaracji zmiennej obiektowej:

```
Dim S as Shapes
```

Oczywiście istnieje jeszcze obszerny system pomocy języka VBA. Znajdują się w nim informacje na temat właściwości i metod większości istotnych obiektów. Najprostszym sposobem uzyskania tych informacji jest wpisanie nazwy obiektu w oknie *Immediate* u dołu ekranu edytora Visual Basic i przemieszczenie kurSORA w dowolne miejsce nazwy obiektu. Po wcisnięciu *F1* uzyskamy pomoc na temat określonego obiektu.

**O co chodzi z tymi kolekcjami? Czym właściwie są kolekcje?
Czy są to obiekty?**

Kolekcje to obiekty zawierające grupę powiązanych z sobą obiektów. Nazwa kolekcji ma formę liczby mnogiej utworzonej od nazwy obiektów, które zawiera. Na przykład kolekcja *Worksheets* jest obiektem zawierającym wszystkie obiekty *Worksheet* w skoroszytcie. Z kolekcji można korzystać tak jak z tablicy: *Worksheets(1)* odnosi się do pierwszego obiektu *Worksheet* w skoroszytcie. Zamiast numerów indeksów można też używać

nazw arkuszy, na przykład Worksheets("Arkusz1"). Zastosowanie kolekcji ułatwia pracę z wszystkimi powiązanymi ze sobą obiektami oraz przetwarzanie w pętli wszystkich obiektów w kolekcji za pomocą konstrukcji For Each ... Next.

Kiedy próbuję odwołać się do arkusza w kodzie VBA, uzyskuję komunikat o błędzie „subscript out of range” (indeks poza zakresem). Ja przecież nie używam żadnych indeksów. O co tu chodzi?

Ten błąd powstaje podczas próby dostępu do nieistniejącego obiektu kolekcji. Na przykład wykonanie poniższej instrukcji spowoduje powstanie błędu w przypadku, gdy aktywny skoroszyt nie zawiera arkusza o nazwie MójArkusz:

```
Set X = ActiveWorkbook.Worksheets("MójArkusz")
```

Jak zablokować możliwość przewijania arkusza przez użytkownika?

Można ukryć nieużywane wiersze i kolumny albo skorzystać z instrukcji VBA w celu ustawienia obszaru przewijania dla arkusza. Na przykład poniższa instrukcja spowoduje ustawienie obszaru przewijania w arkuszu Arkusz1 w taki sposób, że użytkownik nie będzie mógł uaktywnić żadnych komórek poza zakresem B2:D50:

```
Worksheets("Arkusz1").ScrollArea = "B2:D50"
```

Aby przywrócić zwykły tryb przewijania, należy wykorzystać następującą instrukcję:

```
Worksheets("Arkusz1").ScrollArea = ""
```

Pamiętaj, że ustawienie ScrollArea nie jest zapisywane w skoroszycie. Z tego powodu instrukcję przypisania właściwości ScrollArea należy wykonywać po każdym otwarciu skoroszytu. Instrukcję tę można umieścić w procedurze obsługi zdarzenia Workbook_Open.

Czym się różni metoda Select obiektu Range od metody Goto obiektu Application?

Metoda Select obiektu Range powoduje zaznaczenie zakresu tylko w aktywnym skoroszycie. Metoda Application.Goto umożliwia zaznaczenie zakresu w dowolnym arkuszu skoroszytu. Za pomocą metody Application.Goto można też uaktywnić arkusz oraz przenieść go w taki sposób, aby zaznaczony zakres znalazł się w lewym górnym rogu.

Jaka jest różnica pomiędzy uaktywnieniem zakresu a jego zaznaczeniem?

W niektórych przypadkach wykonanie metody Activate niczym nie różni się od wykonania metody Select. Jednak czasami wyniki są zupełnie inne. Założmy, że zaznaczono zakres A1:C3. Wykonanie poniższej instrukcji spowoduje uaktywnienie komórki C3. W dalszym ciągu pozostanie zaznaczony ten sam zakres, ale aktywną komórką, tzn. tą, w której znajduje się wskaźnik, stanie się komórka C3.

```
Range("C3").Activate
```

Wykonanie kolejnej instrukcji spowoduje natomiast zaznaczenie pojedynczej komórki, która równocześnie stanie się komórką aktywną (przy założeniu, że zaznaczono ten sam zakres A1:C3):

```
Range("C3").Select
```

Czy istnieje łatwy sposób usunięcia z arkusza wszystkich wartości i pozostawienie formuł bez zmian?

Tak. Poniższy kod powoduje usunięcie wszystkich wartości z tych komórek aktywnego arkusza, które nie zawierają formuł (instrukcja nie modyfikuje formatowania komórek).

```
On Error Resume Next
Cells.Special Cells(xlCellTypeConstants, 23).ClearContents
```

Drugi argument, 23, to suma wartości następujących wbudowanych stałych: xlErrors (16), xlLogical (4), xlNumbers (1) oraz xlTextValues (2).

Użycie instrukcji On Error Resume Next zabezpiecza przed powstaniem błędu, jeżeli nie ma komórek do usunięcia.

Potrafię napisać kod VBA, który zaznacza zakres na podstawie podanego adresu komórki, ale w jaki sposób napisać kod, który zaznacza zakres tylko na podstawie numeru wiersza i kolumny?

Należy skorzystać z metody Cells. Na przykład poniższa instrukcja powoduje zaznaczenie zakresu w 5. wierszu i 12. kolumnie (chodzi o komórkę L5):

```
Cells(5, 12).Select
```

Jak wyłączyć aktualizowanie ekranu w czasie działania makra?

Poniższa instrukcja wyłącza aktualizowanie ekranu i przyspiesza działanie makr modyfikujących ekran:

```
Application.ScreenUpdating = False
```

Po zakończeniu działania procedury właściwość ScreenUpdating jest automatycznie ustawiana na wartość True. W razie potrzeby jednak możesz przywrócić aktualizację ekranu w dowolnym momencie poprzez wykonanie następującego polecenia:

```
Application.ScreenUpdating = True
```

Napisałem makro, które wykorzystuje pętlę do animowania wykresu, ale niestety animacja nie chce działać.

Spróbuj wstawić w pętli następujące polecenie:

```
DoEvents
```

Jak najłatwiej zdefiniować nazwę zakresu za pomocą kodu VBA?

Włączenie rejestratora makr w czasie nadawania nazwy zakresowi spowoduje wygenerowanie kodu o następującej postaci:

```
Range("D14:G20").Select
ActiveWorkbook.Names.AddName := "InputArea", _
RefersToR1C1 := "=Arkusz1!R14C4:R20C7"
```

Jednak znacznie prostszą metodą jest użycie następującej instrukcji:

```
Sheets("Arkusz1").Range("D14:G20").Name = "InputArea"
```

Jak sprawdzić, czy komórce lub zakresowi nadano nazwę?

Należy sprawdzić właściwość Name obiektu Name będącego składową obiektu Range. Poniższa funkcja pobiera zakres jako argument i zwraca nazwę zakresu (jeżeli taką nazwę przypisano). Jeżeli zakresowi nie nadano nazwy, zwraca wartość False.

```
Function RangeName(rng) As Variant  
On Error Resume Next  
RangeName = rng.Name.Name  
If Err <> 0 Then RangeName = False  
End Function
```

Czy można wyświetlać komunikaty na pasku stanu w czasie działania makra? Napisałem makro, które dłużej działa. Byłoby dobrze, gdyby postęp wykonania wyświetlał się na pasku stanu.

Tak. Należy przypisać tekst do właściwości StatusBar obiektu Application. Oto przykład:

```
Application.StatusBar = "Przetwarzanie pliku " & FileNum
```

Po zakończeniu działania procedury należy przywrócić stan początkowy za pomocą następującej instrukcji:

```
Application.StatusBar = False  
Application.StatusBar = ""
```

Zarejestrowałem makro VBA kopiujące zakres i wklejające go do innego obszaru. Wykorzystałem w nim metodę Select. Czy istnieje lepszy sposób od wycinania i wklejania?

Tak. Chociaż rejestrator makr zwykle zaznacza komórki przed wykonaniem z nimi działań, zaznaczanie nie jest konieczne, a wręcz spowalnia działanie makra. Zarejestrowanie bardziej prostej operacji kopiowania i wklejania powoduje wygenerowanie czterech wierszy kodu VBA; w dwóch spośród nich wykorzystywana jest metoda Select. Oto przykład:

```
Range("A1").Select  
Selection.Copy  
Range("B1").Select  
ActiveSheet.Paste
```

Te cztery wiersze można zastąpić pojedynczą instrukcją, na przykład:

```
Range("A1").Copy Range("B1")
```

Zwróć uwagę, że w tej instrukcji nie wykorzystano metody Select.

Nie udało mi się znaleźć metody pozwalającej na sortowanie tablicy VBA. Czy to oznacza, że muszę skopiować wartości do arkusza i zastosować metodę Range.Sort?

Nie ma wbudowanej metody sortowania tablicy w języku VBA. Jednym ze sposobów jest skopiowanie tablicy do arkusza, ale lepiej napisać własną procedurę sortowania. Istnieje wiele algorytmów sortowania, a niektóre z nich są łatwe do zaprogramowania w języku VBA. W tej książce przedstawiono kod VBA dla kilku technik sortowania.

Moje makro działa z zaznaczonymi komórkami, ale nie działa w przypadku zaznaczenia innych elementów (np. wykresu). W jaki sposób sprawdzić, czy zaznaczono zakres komórek?

Można skorzystać z funkcji VBA TypeName w celu sprawdzenia obiektu Selection. Oto przykład:

```
If TypeName(Selection) <> "Range" Then  
    MsgBox "Zaznacz zakres komórek!"  
    Exit Sub  
End If
```

Innym sposobem jest wykorzystanie właściwości RangeSelection zwracającej obiekt Range, który reprezentuje zaznaczone komórki w arkuszu wyświetlonym w określonym oknie nawet wtedy, kiedy jest aktywny obiekt graficzny. Właściwość RangeSelection jest składową obiektu Window, a nie Workbook. Na przykład wykonanie poniższej instrukcji powoduje wyświetlenie adresu zaznaczonego zakresu:

```
MsgBox ActiveWindow.RangeSelection.Address
```

Jak sprawdzić, czy wykres został uaktywniony?

Należy skorzystać z następującego fragmentu kodu:

```
If ActiveChart Is Nothing Then  
    MsgBox "Zaznacz wykres"  
    Exit Sub  
End If
```

Komunikat wyświetli się tylko wtedy, kiedy wykres nie został uaktywniony (dotyczy to zarówno wykresów wbudowanych, jak i wykresów w osobnych arkuszach).

Chcę, by makro VBA zliczało liczbę wierszy zaznaczonych przez użytkownika. Zastosowanie metody Selection.Rows.Count nie działa, jeżeli zostaną zaznaczone wiersze, które ze sobą nie sąsiadują. Czy to jest błąd?

W zasadzie metoda Selection.Rows.Count tak właśnie powinna działać — zwrócić liczbę elementów w pierwszym zaznaczonym obszarze (nieciągły zakres składa się z kilku obszarów). Aby uzyskać poprawną liczbę wierszy, kod VBA powinien najpierw sprawdzić liczbę zaznaczonych obszarów, a następnie zliczyć liczbę wierszy w każdym z nich. Do zliczenia liczby obszarów można wykorzystać metodę Selection.Areas.Count. Poniższy kod zapisuje całkowitą liczbę zaznaczonych wierszy w zmiennej NumRows:

```
NumRows = 0  
For Each areaCounter In Selection.Areas  
    NumRows = NumRows + areaCounter.Rows.Count  
Next areaCounter
```

Podobną procedurę można także zastosować do zliczania kolumn i komórek.

Korzystam z Excela do tworzenia faktur.**Czy istnieją sposoby generowania niepowtarzalnych numerów faktur?**

Jednym ze sposobów jest wykorzystanie rejestru Windows. Pokazano to w poniższym przykładzie:

```
Licznik = GetSetting("Firma XYZ", "NrFaktury", "Numer", 0)
Licznik = Licznik + 1
SaveSetting "Firma XYZ", "NrFaktury", "Numer", Licznik
```

Wykonanie tych instrukcji spowoduje pobranie bieżącej wartości z rejestru, dodanie do niej jedynki i przypisanie do zmiennej **Licznik**. Następnie zaktualizowana w ten sposób wartość zostanie ponownie zapisana do rejestru. Wartość zmiennej **Licznik** można wykorzystać jako niepowtarzalny numer faktury.

Opisaną powyżej technikę możesz zaadaptować również do innych celów. Na przykład możesz użyć takiego rozwiązania do zliczania liczby otwarć skoroszytu. Aby to zrobić, powinieneś umieścić podobny kod w procedurze **Workbook_Open**.

**Czy istnieje właściwość skoroszytu,
która uniemożliwia ukrycie okna Excela przez okno innej aplikacji?**

Nie.

**Czy istnieje instrukcja VBA umożliwiająca zaznaczenie ostatniej pozycji
w kolumnie lub wierszu? Ręcznie można to zrobić za pomocą klawiszy
Ctrl+Shift+strzałka w dół lub **Ctrl+Shift+strzałka w prawo**,
ale jak zrobić to samo za pomocą makra?**

Odpowiednikiem wciśnięcia klawiszy **Ctrl+Shift+strzałka w dół** w języku VBA jest następująca instrukcja:

```
Selection.End(xlDown).Select
```

Stałe, które można wykorzystać dla innych kierunków strzałek, są następujące: **xlToLeft** (w lewo), **xlToRight** (w prawo) oraz **xlUp** (w góre).

**Jak uzyskać informację o ostatniej niepustej komórce
w podanej kolumnie?**

Adres ostatniej niepustej komórki w kolumnie A wyświetla następująca instrukcja:

```
MsgBox ActiveSheet.Cells(Rows.Count, 1).End(xlUp).Address
```

Niestety powyższe polecenie nie będzie działać poprawnie, jeżeli ostatnia komórka kolumny nie będzie pusta. Aby obsłużyć taką jednak mało prawdopodobną sytuację, możesz użyć następującego kodu:

```
With ActiveSheet.Cells(Rows.Count, 1)
If IsEmpty(.Value) Then
    MsgBox .End(xlUp).Address
End With
```

```
Else  
    MsgBox .Address  
End If  
End With
```

Odwołania w języku VBA bywają bardzo długie, zwłaszcza jeżeli trzeba użyć pełnej specyfikacji arkusza i skoroszytu. Czy istnieje sposób skrócenia tych odwołań?

Tak. Należy skorzystać z instrukcji Set w celu utworzenia zmiennej obiektowej. Oto przykład:

```
Dim MyRange as Range  
Set MyRange = ThisWorkbook.Worksheets("Arkusz1").Range("A1")
```

Po wykonaniu instrukcji Set można odwoływać się do jednokomórkowego obiektu Range za pomocą nazwy MyRange. Na przykład za pomocą poniższego kodu można przypisać wartość do komórki:

```
MyRange.Value = 10
```

Co więcej, zastosowanie zmiennych obiektowych przyspiesza także wykonywanie kodu.

Czy można zadeklarować tablicę, jeżeli nie wiadomo, ile elementów ma ona zawierać?

Tak. Można zadeklarować tablicę dynamiczną za pomocą instrukcji Dim z pustymi nawiasami, a następnie, kiedy już wiadomo, ile elementów będzie zawierać, przydzielić dla niej pamięć za pomocą instrukcji ReDim. Aby nie stracić bieżącej zawartości tablicy w momencie ponownego przydziału pamięci, należy zastosować instrukcję ReDim Preserve.

Jak mogę wycofać operacje wykonane przed uruchomieniem makra?

Niestety nie możesz — uruchomienie makra w Excelu kasuje zawartość stosu mechanizmu wycofywania poleceń.

Czy mogę umożliwić użytkownikowi cofnięcie skutków działania makra?

W większości przypadków tak, ale wycofanie skutków działania makra nie jest czymś, co mogłoby być zrobione automatycznie.

Aby umożliwić cofanie skutków działania makra, należy śledzić zmiany wykonywane przez makro, a następnie zapewnić odtworzenie stanu początkowego, jeżeli użytkownik wybierze polecenie *Cofnij*.

Aby uaktywnić to polecenie, należy użyć metody OnUndo jako ostatniego polecenia makra. Metoda ta umożliwia zdefiniowanie tekstu, który wyświetli się w miejscu polecenia *Cofnij*, a także procedury, która będzie wykonana po wybraniu tego polecenia. Oto przykład:

```
Application.OnUndo "Ostatnie makro", "MyUndoMacro"
```

Więcej szczegółowych informacji na temat wycofywania zmian wprowadzonych przez makro znajdziesz w rozdziale 14.

Czy mogę zatrzymać działanie makra, tak aby użytkownik mógł wprowadzić dane do wybranej komórki?

W Excelu można wykorzystać instrukcję `InputBox` do pobrania wartości od użytkownika i umieszczenia jej w określonej komórce. Na przykład pierwsza z instrukcji pokazanych poniżej wyświetla okno `InputBox`. Kiedy użytkownik wprowadzi wartość, zostanie ona umieszczona w komórce A1.

```
UserVal = Application.InputBox(prompt:="Podaj wartość:", Type:=1)  
If TypeName(UserVal)<>"Boolean" Then Range("A1") = UserVal
```

W języku VBA istnieje funkcja `InputBox`, ale jest także metoda `InputBox` obiektu `Application`. Czy to jest to samo?

Nie, metoda `InputBox` obiektu `Application` jest bardziej uniwersalna, ponieważ umożliwia sprawdzanie poprawności danych wprowadzanych przez użytkownika. W poprzednim przykładzie użyto wartości 1 dla argumentu `Type` metody `InputBox`. Użycie tej wartości zapewnia wprowadzenie liczb.

Próbuję napisać instrukcję VBA tworzącą formułę. Aby to zrobić, muszę użyć znaku cudzysłowu ("") wewnętrz łańcucha znaków ujętego w cudzysłów. Jak to zrobić?

Załóżmy, że za pomocą kodu VBA chcemy wprowadzić poniższą formułę do komórki B1:

```
= IF(A1="Tak", PRAWDA, FAŁSZ)
```

Próba wykonania poniższej instrukcji kończy się powstaniem błędu składniowego:

```
Range("B1").Formula = "=IF(A1=""Tak"", PRAWDA, FAŁSZ)"      'powstanie błędu
```

Rozwiązywanie polega na wykorzystaniu podwójnych cudzysłowów z każdej strony. Pożądany efekt uzyskamy za pomocą następującej instrukcji:

```
Range("B1").Formula = "=IF(A1=""Tak"", PRAWDA, FAŁSZ)"
```

Innym sposobem jest wykorzystanie funkcji VBA `Chr` z argumentem 34, która zwraca znak cudzysłowu. Pokazano to w poniższym przykładzie:

```
Range("B1").Formula =  
    "=IF(A1=" & Chr(34) & "Tak" & Chr(34) & ", PRAWDA, FAŁSZ)"
```

Jeszcze innym podejściem do rozwiązywania problemu jest użycie w formule apostrofów zamiast znaków cudzysłowu i następnie użycie funkcji `Replace` języka VBA do zamiany apostrofów na znaki cudzysłowu.

```
TheFormula = "=IF(A1='Yes',TRUE,FALSE)"  
Range("B1").Formula = Replace(TheFormula, "'", Chr(34))
```

Utworzyłem tablicę, ale pierwszy jej element jest traktowany jak drugi. W czym problem?

W przypadku braku innych ustaleń w języku VBA indeksy tablic zaczynają się od zera. Aby indeksy wszystkich tablic zawsze zaczynały się od 1, w module VBA należy wstawić następującą instrukcję:

Option Base 1

Można również określić dolny i górny indeks w instrukcji deklaracji tablicy. Oto przykład:

```
Dim Months(1 To 12) As String
```

Chciałbym, aby kod VBA wykonywał się tak szybko, jak to tylko możliwe. Czy są na to jakieś sposoby?

Poniżej znajdziesz kilka ogólnych wskazówek:

- Sprawdź, czy zadeklarowałeś wszystkie użyte zmienne. Aby wymusić konieczność deklaracji, na początku modułu umieść instrukcję Option Explicit.
- Jeżeli wykorzystujesz dowolny obiekt Excela więcej niż raz, utwórz zmienną obiektywną dla tego obiektu.
- Tam, gdzie to możliwe, korzystaj z konstrukcji With ... End With.
- Jeżeli makro zapisuje informacje do arkusza, wyłącz aktualizowanie ekranu za pomocą instrukcji Application.ScreenUpdating =False.
- Jeżeli aplikacja wprowadza w komórkach dane, do których odwołania występują w jednym lub kilku formułach, ustaw tryb przeliczania na ręczny, aby zapobiec wykonywaniu zbędnych obliczeń.

Pytania dotyczące zagadnień związanych z bezpieczeństwem

Dlaczego podczas otwierania pliku Excel wypisuje komunikat, że makra zostały zablokowane, mimo że w skoroszycie nie ma żadnego makra?

Taki komunikat pojawia się nawet wtedy, kiedy skoroszyt zawiera całkowicie pusty moduł kodu VBA. Kiedy usuniesz taki moduł, komunikat przestanie się pojawiać.

Jak mogę się upewnić, że każdy użytkownik, który otwiera mój skoroszyt, włączy makra?

Nie ma żadnego sposobu, aby to zrobić, ale jednym z możliwych rozwiązań jest takie przygotowanie skoroszytu, aby stał się on całkowicie bezużyteczny, jeżeli makra nie zostaną włączone. Na przykład możesz ukryć krytyczne arkusze skoroszytu i wyświetlać je dopiero za pomocą makra. Niestety taka metoda nie jest całkowicie skuteczna i odporna na działania użytkownika.

W jaki sposób mogę zabezpieczyć kod mojego dodatku przed przeglądaniem go przez innych użytkowników?

Przejdź do edytora VBE i z menu głównego wybierz polecenie Tools/xxxx Properties (gdzie xxxx to nazwa Twojego projektu). Przejdź na kartę Protection, zaznacz opcję Lock Project for Viewing i następnie wpisz hasło. Po zakończeniu zapisz skoroszyt.

Czy moje dodatki są bezpieczne? Innymi słowy, czy jeżeli udostępnię plik XLAM innym użytkownikom, to mogę być pewny, że nikt nie będzie w stanie dostać się do mojego kodu?

Przede wszystkim zabezpiecz dodatek przy użyciu hasła. Takie rozwiązanie skutecznie uniemożliwi dostęp do kodu większości użytkowników. Najnowsze wersje Excela mają znaczco poprawione i ulepszone mechanizmy zabezpieczeń, ale pamiętaj, że hasło zawsze można złamać za pomocą wielu różnych narzędzi. Wniosek? Nigdy nie powinieneś traktować dodatków XLAM jako plików, w których Twoje tajemnice są bezpieczne.

Zablokowałem dostęp do projektu VBA za pomocą hasła, a potem je zapomniałem. Czy istnieje sposób odblokowania projektu?

Niezależni producenci oferują różne narzędzia do łamania haseł. Najlepiej skorzystać z wyszukiwarki i poszukać frazy Excel hasło (lub Excel password w celu wyszukania stron w języku angielskim). Skoro istnieją takie narzędzia, to powinieneś się domyślić, że hasła Excela nie są zbyt bezpieczne.

Jak mogę napisać makro zmieniające hasło projektu?

Nie można napisać takiego makra. Elementy zabezpieczeń projektu VBA nie są ujawnione w modelu obiektów. Najprawdopodobniej w ten sposób twórcy Excela chcieli utrudnić tworzenie narzędzi do łamania haseł.

Napisałem makro tworzące inne makra, które znakomicie działa na moim komputerze, ale nie chce działać na komputerach innych użytkowników.

Najprawdopodobniej na komputerach innych użytkowników opcja *Ufaj dostępowi do programu Visual Basic Project* jest wyłączona. Znajdziesz ją w kategorii *Ustawienia makr*, w oknie dialogowym *Centrum zaufania*. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 26.

Pytania dotyczące formularzy UserForm

Chcę pobrać od użytkownika zaledwie kilka informacji. Wydaje mi się, że zastosowanie formularza UserForm to zbyt dużo. Czy istnieje jakieś alternatywne rozwiązanie?

Tak, można skorzystać z funkcji MsgBox i InputBox języka VBA. Można też użyć metody InputBox obiektu Application. Więcej szczegółowych informacji na temat tych i innych funkcji znajdziesz w rozdziale 10.

W formularzu UserForm zdefiniowałem 12 przycisków.

Jak przypisać pojedyncze makro, które będzie wykonywane w momencie kliknięcia dowolnego z nich?

Nie ma łatwego sposobu wykonania tej czynności, ponieważ każdy przycisk posiada własną procedurę obsługi zdarzenia `Click`. Jednym z rozwiązań jest wywoływanie tej samej procedury z każdej procedury `CommandButton_Click`. Innym sposobem jest wykorzystanie modułu klasy w celu zdefiniowania nowej klasy. Taka technika została opisana w rozdziale 13.

Jak wyświetlić wykres w formularzu UserForm?

Nie możesz tego zrobić bezpośrednio. Jednym z możliwych rozwiązań jest napisanie makra, które zapisuje wykres do pliku GIF, a następnie ładuje ten plik do formantu `Image` w formularzu `UserForm`. Odpowiedni przykład takiego rozwiązania znajdziesz w rozdziale 13.

Jak usunąć przycisk „X” z paska tytułu formularza UserForm?

Nie chcę, aby użytkownik mógł zamykać okno w ten sposób.

Usunięcie przycisku *Zamknij* z paska tytułu formularza `UserForm` wymaga wywołania kilku skomplikowanych funkcji API. Prostszym rozwiązaniem jest przechwytcie wszystkich prób zamknięcia formularza `UserForm` za pomocą procedury obsługi zdarzenia `UserForm_QueryClose` umieszczonej w module kodu formularza `UserForm`. Poniższy przykład pokazuje sposób zablokowania możliwości zamknięcia okna poprzez kliknięcie przycisku *Zamknij*:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = vbFormControlMenu Then
        MsgBox "Nie można zamykać formularza w ten sposób!"
        Cancel = True
    End If
End Sub
```

Utworzyłem formularz UserForm, którego formanty są powiązane z arkuszem za pomocą właściwości ControlSource. Czy jest to najlepszy sposób rozwiązania tego problemu?

Prawdopodobnie nie. Ogólnie rzecz biorąc, należy unikać korzystania z łączy do komórek arkusza, chyba że jest to bezwzględnie konieczne. Wykorzystanie łączy powoduje zwolnienie działania aplikacji, ponieważ arkusz jest przeliczany za każdym razem, kiedy kontrolka modyfikuje komórkę. Dodatkowo, jeżeli w formularzu `UserForm` znajduje się przycisk *Anuluj*, istnieje obawa, że w momencie jego naciśnięcia, część komórek arkusza została już wcześniej zmodyfikowana.

Czy można utworzyć tablicę formantów dla formularza UserForm? Można to zrobić w Visual Basicu, ale nie wiem, jak się to robi w języku VBA.

Nie można utworzyć tablicy formantów, ale można utworzyć tablicę obiektów `Control`. Poniższy kod tworzy tablicę składającą się z formantów `CommandButton`:

```
Private Sub UserForm_Initialize()
    Dim Buttons() As CommandButton
    Cnt = 0
    For Each Ctl In UserForm1.Controls
        If TypeName(Ctl) = "CommandButton" Then
            Cnt = Cnt + 1
            ReDim Preserve Buttons(1 To Cnt)
            Set Buttons(Cnt) = Ctl
        End If
    Next Ctl
End Sub
```

Czy jest różnica pomiędzy ukryciem formularza UserForm a usunięciem go z pamięci?

Tak, użycie metody `Hide` powoduje tylko, że formularz `UserForm` jest niewidoczny. Dalej jednak jest przechowywany w pamięci. Metoda `UnLoad` rozpoczyna proces rzeczywistego „niszczenia” formularza (generuje zdarzenie `Terminate` dla formularza `UserForm`) i usuwa go z pamięci.

Co zrobić, aby formularz UserForm pozostał otwarty podczas wykonywania innych działań?

Domyślnie formularze `UserForm` są modalne, co oznacza, że należy je zamknąć przed wykonaniem innych działań. Jednak w razie potrzeby można zdefiniować niemonalne formularze `UserForm` poprzez użycie stałej `vbModeless` jako argumentu metody `Show`. Oto przykład:

```
UserForm1.Show vbModeless
```

Chciałbym wyświetlić wskaźnik postępu zadania dla dugo wykonujących się procesów, podobny do tych, które wyświetlają się w programach instalujących oprogramowanie. Jak to zrobić?

Można wykorzystać formularz `UserForm`. W rozdziale 13. zostało opisanych kilka różnych technik tworzenia wskaźników postępu. Jeden z nich polega na stopniowym rozciąganiu paska w formularzu `UserForm` podczas długotrwałego procesu.

Czy można używać kształtów na formularzu UserForm?

Kształtów nie możesz bezpośrednio wstawić do formularza `UserForm`, ale możesz to zrobić pośrednio. Najpierw zaznacz żądaną kształt i naciśnij kombinację klawiszy `Ctrl+C`. Następnie przejdź do formularza `UserForm` i wstaw formant `Image`. Naciśnij klawisz `F4`, aby wyświetlić okno dialogowe `Properties`. Zaznacz właściwość `Picture` i naciśnij kombinację klawiszy `Ctrl+V`, by wstawić zawartość schowka systemowego do formantu `Image`. W razie potrzeby możesz również ustawić właściwość `AutoSize` na wartość `True`.

Jak wygenerować listę plików i katalogów w formularzu UserForm, aby użytkownik mógł wybrać plik z listy?

Nie ma takiej potrzeby. Aby zrealizować tę funkcję, wystarczy skorzystać z metody `GetOpenFilename` języka VBA. Jej wykonanie spowoduje wyświetlenie okna dialogowego *Otwórz*, w którym użytkownik może wybrać napęd, katalog i plik. Sama metoda nie otwiera wybranego pliku, więc aby tego dokonać będziesz musiał użyć dodatkowego kodu.

Chciałbym połączyć łańcuchy znaków i wyświetlić je za pomocą kontrolki ListBox. Kiedy jednak to robię, nie są one właściwie wyrównane. Co zrobić, aby pomiędzy łańcuchami znaków były wyświetlane równe odstępy?

Można wykorzystać czcionkę o stałej szerokości znaków, na przykład *Courier New*. Jednak lepszym sposobem jest zdefiniowanie dwóch kolumn w formancie *ListBox* (więcej informacji znajduje się w rozdziale 12.).

Czy istnieje prosty sposób wypełnienia pola listy lub pola kombi?

Tak, można wykorzystać tablicę. Wykonanie poniższej instrukcji spowoduje dodanie trzech pozycji do pola listy *ListBox1*:

```
ListBox1.List = Array("Sty", "Lut", "Mar")
```

Czy można wyświetlać wbudowane okna dialogowe Excela za pomocą kodu VBA?

Większość okien dialogowych Excela, choć nie wszystkie, można wyświetlić za pomocą metody `Application.Dialogs`. Na przykład wykonanie poniższej instrukcji spowoduje wyświetlenie okna dialogowego pozwalającego na formatowanie liczb w komórkach:

```
Application.Dialogs(xlDialogFormatNumber).Show
```

Pamiętaj jednak, że taka metoda nie zawsze działa poprawnie, a poza tym nie wszystkie okna Excela są w ten sposób dostępne.

Znacznie lepszym rozwiązaniem będzie wykonanie polecenia znajdującego się na Wstążce (włączając w to wyświetlanie okien dialogowych), używając metody `ExecuteMso` razem z nazwą formantu. Na przykład polecenie zamieszczone poniżej wyświetla okno dialogowe pozwalające na formatowanie wartości liczbowych w komórkach:

```
Application.CommandBars.ExecuteMso("NumberFormatsDialog")
```

Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 20.

Wypróbowałem technikę opisaną w odpowiedzi na poprzednie pytanie i uzyskałem komunikat o błędzie. Dlaczego?

Wywołanie metody `Dialogs` (jak i metody `ExecuteMso`) zakończy się błędem, jeżeli próba zostanie wykonana w niewłaściwym kontekście. Tak się stanie na przykład podczas próby wyświetlenia okna dialogowego *Typ wykresu* (`xlDialogChartType`) w momencie, kiedy wykres nie jest aktywny.

```
Application.CommandBars.ExecuteMso ("CellsInsertDialog")
```

Za każdym razem, kiedy tworzę formularze UserForm, dodaję przyciski OK i Anuluj. Czy jest jakiś sposób, aby takie formanty wyświetlały się automatycznie?

Tak. Należy zdefiniować formularz *UserForm* z najczęściej używanymi formantami. Następnie wybrać polecenie *File/Export File*, aby je zapisać. Aby dodać nowy formularz do innego projektu, należy skorzystać z polecenia *File/Import File*.

Czy można utworzyć formularz UserForm bez paska tytułu?:?

Tak, ale takie rozwiązanie wymaga użycia kilku złożonych funkcji Windows API. Więcej szczegółowych informacji na ten temat znajdziesz w rozdziale 13.

Kiedy kliknę przycisk w zdefiniowanym przeze mnie formularzu UserForm, nic się nie dzieje. Czy robię coś źle?

Formanty dodane w formularzach *UserForm* nie wykonują żadnych działań, dopóki nie zostaną dla nich zdefiniowane procedury obsługi zdarzeń. Procedury te muszą znajdować się w module kodu formularza *UserForm* i posiadać odpowiednie nazwy.

Czy można utworzyć formularz UserForm, którego rozmiar jest taki sam, niezależnie od rozdzielczości ekranu?

Można, ale chyba nie ma sensu. Należy napisać kod sprawdzający rozdzielcość ekranu, a następnie wykorzystać właściwość *Zoom* formularza *UserForm* do przeskalowania formularza. Zwykle jednak projektuje się formularze *UserForm* dla najniższej rozdzielczości, która będzie używana.

Czy można utworzyć pole w formularzu UserForm umożliwiające określenie zakresu arkusza poprzez jego wskazanie?

Tak. Do tego celu należy wykorzystać formant *RefEdit*. Przykład zaprezentowano w rozdziale 12.

Czy można zmienić początkową pozycję formularza UserForm?

Tak, można ustawić właściwości *Left* i *Top* okna. Aby jednak ustawienia te przyniosły efekt, należy ustawić właściwość *StartUpPosition* formularza *UserForm* na wartość 0.

Używam komputera wyposażonego w dwa monitory, a formularze UserForm nie wyświetlają się w centralnej części okna Excela. Czy można wymusić taką lokalizację okien formularzy?

Tak. Do wyświetlania okien formularzy powinieneś w takiej sytuacji użyć następującego kodu:

```
With UserForm1  
    .StartPosition = 0  
    .Left = Application.Left + (0.5 * Application.Width) - (0.5 * .Width)
```

```
.Top = Application.Top + (0.5 * Application.Height) - (0.5 * .Height)
.Show 0
End With
```

Czy można utworzyć formularz UserForm tak, aby użytkownik mógł zmieniać jego rozmiar?

Tak. Przykład takiego rozwiązania znajdziesz w rozdziale 13.

Pytania dotyczące dodatków

Gdzie można znaleźć dodatki Excela?

Dodatki Excel możesz znaleźć w wielu różnych miejscach:

- Excel posiada kilka swoich dodatków, dostarczanych razem z programem. Do ich zainstalowania powinieneś użyć okna dialogowego *Dodatki*.
- Więcej dodatków możesz pobrać ze strony internetowej *Microsoft Office Update*.
- Specjalizowane dodatki są również tworzone, sprzedawane i udostępniane przez firmy trzecie.
- Wielu programistów tworzy własne dodatki, które udostępnia bezpłatnie w sieci Internet.
- Możesz tworzyć swoje własne dodatki.

W jaki sposób zainstalować dodatek?

Najbardziej rozpowszechnionym sposobem instalowania dodatków jest użycie okna dialogowego *Dodatki*. Aby przywołać je na ekran, przejdź na kartę *PLIK* i z menu wybierz polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Kliknij kategorię *Dodatki*, z listy rozwijanej *Zarządzaj* wybierz opcję *Dodatki programu Excel* i naciśnij przycisk *Przejdź*.

Dodatek możesz również zainstalować, wybierając polecenie *PLIK/Otwórz*, ale preferowanym rozwiązaniem jest użycie okna dialogowego *Dodatki*. Dodatku otwartego za pomocą polecenia *PLIK/Otwórz* nie można zamknąć bez użycia VBA.

Kiedy instaluję dodatek z okna dialogowego Dodatki, wyświetla się on bez nazwy i opisu. Jak dodać opis dodatku?

Przed utworzeniem dodatku powinieneś wykonać polecenie *PLIK/Informacje/Właściwości/Właściwości zaawansowane*, co spowoduje wyświetlenie panelu właściwości dokumentu. Następnie powinieneś w polu *Tytuł* wprowadzić tekst, który ma się wyświetlić w oknie dialogowym *Dodatki*, a w polu *Komentarze* wprowadzić opis dodatku. Po wykonaniu tych działań można utworzyć dodatek w zwykły sposób.

Mam kilka dodatków, których nie używam, ale nie wiem, w jaki sposób usunąć je z listy Dostępne dodatki w oknie dialogowym Dodatki.

Niestety, nie ma sposobu usunięcia niepotrzebnych dodatków z listy bezpośrednio z poziomu Excela. W tym celu należy otworzyć rejestr Windows i usunąć odwołania do plików dodatków, które chcemy usunąć. Innym sposobem jest przeniesienie lub usunięcie pliku dodatku. Przy próbie otwarcia dodatku w oknie dialogowym *Dodatki* Excel wyświetli pytanie, czy chcemy usunąć dodatek z listy. Odpowiedz twierdząco.

Jak tworzy się dodatki?

Należy uaktywnić dowolny arkusz i wybrać polecenie *PLIK/Zapisz jako*. Następnie na rozwijanej liście *Zapisz jako typ* wybrać opcję *Dodatek programu Excel (*.xlam)*. Excel utworzy plik dodatku, a oryginalny skoroszyt nadal pozostanie otwarty.

Próbowałem utworzyć dodatek, ale w polu Zapisz jako typ nie ma możliwości wyboru dodatku.

Najprawdopodobniej aktywny arkusz nie jest arkuszem danych (np. jest to arkusz wykresu). Dodatek musi posiadać co najmniej jeden arkusz danych, który powinien być aktywny w momencie zapisywania skoroszytu.

Czy kiedy utworzę dodatek, będzie on działał z pakietem Excel Web App (sieciowa wersja Excela)?

Nie, sieciowa wersja Excel Web App nie obsługuje dodatków, a nawet makr.

Czy moje dodatki będą działać z wersjami Excela dla urządzeń mobilnych pracujących pod kontrolą systemu Windows RT?

Nie. Takie wersje Excela nie obsługują ani dodatków, ani makr.

Czy powiniensem przekształcić wszystkie moje ważne skoroszyty na dodatki?

Nie! Chociaż można utworzyć dodatek z dowolnego skoroszytu, to jednak nie wszystkie skoroszyty się do tego nadają. Skoroszyt przekształcony na dodatek stanie się niewidoczny. Dla większości skoroszytów nie jest to dobre rozwiązanie.

Czy powiniensem zachować dwie kopie skoroszytu: wersję XLSM i XLAM?

Nie, w razie potrzeby możesz bezpośrednio edytować dodatek, a nawet przekształcić go ponownie do postaci normalnego skoroszytu.

Jak modyfikować dodatek po jego utworzeniu?

Jeżeli chcesz modyfikować kod VBA, to nie musisz wykonywać żadnych dodatkowych operacji — kod dodatku możesz modyfikować i zapisywać bezpośrednio z poziomu edytora VBE. Jeżeli musisz wprowadzić modyfikacje arkuszy dodatku, uaktywnij edytor Visual

Basic (naciśnij kombinację klawiszy *Alt+F11*) i ustaw właściwość `IsAddIn` obiektu `ThisWorkbook` na wartość `False`. Następnie wprowadź zmiany, ustaw właściwość `IsAddIn` na wartość `True` i ponownie zapisz plik.

Czym się różni plik XLSM od pliku XLAM utworzonego na podstawie tego pliku XLSM? Czy plik XLAM to wersja skompilowana? Czy działa szybciej?

Nie ma wielkich różnic pomiędzy tymi dwoma rodzajami plików i w zasadzie nie ma różnic w szybkości działania. Kod VBA zawsze jest komplikowany przed wykonaniem, niezależnie od tego, czy jest zapisany w pliku *XLSM*, czy w pliku *XLAM*. Kod VBA w pliku *XLAM* nie jest skompilowany. Istotna różnica polega na tym, że w pliku *XLAM* skoroszyt nie jest widoczny.

Pytania dotyczące interfejsu użytkownika

Jak użyć VBA do umieszczenia nowego przycisku polecenia na Wstążce?

Niestety nie możesz tego zrobić. Aby tego dokonać, musisz napisać specjalny kod XML (znany jako kod RibbonX) i wstawić dokument XML zawierający ten kod do skoroszytu (do tego celu musisz użyć narzędzi tworzonych przez firmy trzecie). Jeżeli jednak nie boisz się wyzwań, możesz samodzielnie rozpakować plik skoroszytu i ręcznie dokonać odpowiednich modyfikacji.

Jakie możliwości modyfikacji interfejsu pod kątem ułatwienia uruchamiania makr oferuje Excel?

W Excelu 2010 i nowszych masz do dyspozycji następujące rozwiązania:

- Modyfikacja Wstążki (choć wcale nie jest takie proste zadanie).
- Dodanie nowych elementów do menu podręcznego, wyświetlanego po kliknięciu prawym przyciskiem myszy, za pomocą kodu RibbonX (to też nie jest wcale takie proste).
- Umieszczenie przycisku uruchamiającego makro na pasku narzędzi *Szybki dostęp* (zadanie manualne, nie możesz go wykonać przy użyciu VBA).
- Dodanie makra do Wstążki (również zadanie manualne, którego nie można wykonać przy użyciu VBA).
- Przypisanie do makra odpowiedniego skrótu klawiszowego.
- Dodanie za pomocą VBA nowego polecenia do menu podręcznego, aktywowanego prawym przyciskiem myszy. W Excelu 2013 takie rozwiązanie ma pewne ograniczenia.
- Użycie VBA do utworzenie paska narzędzi lub menu w starym stylu — takie elementy będą wyświetlane na karcie *Dodatki*.

Jak dodać makro do paska narzędzi Szybki dostęp?

Kliknij pasek narzędzi *Szybki dostęp* prawym przyciskiem myszy i wybierz z menu podręcznego polecenie *Dostosuj pasek narzędzi Szybki dostęp*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Z listy rozwijanej *Wybierz polecenia z* wybierz opcję *Makra*. Zaznacz wybrane makro na liście i naciśnij przycisk *Dodaj*. Aby zmienić ikonę przycisku lub jego opis, naciśnij przycisk *Modyfikuj*.

Jak dodać makro do Wstążki?

Kliknij Wstążkę prawym przyciskiem myszy i wybierz z menu podręcznego polecenie *Dostosuj Wstążkę*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*. Na karcie *Dostosowywanie Wstążki* z listy rozwijanej po lewej stronie wybierz opcję *Makra*, następnie odszukaj i zaznacz żąданie makro i naciśnij przycisk *Dodaj*. Pamiętaj, że nie możesz dodawać nowych makr do istniejących, wbudowanych grup poleceń. Zamiast tego musisz przed dodaniem makra utworzyć nową, własną grupę poleceń (możesz tego dokonać, naciskając przycisk *Nowa grupa*).

Jak użyć VBA do aktywowania określonej karty Wstążki?

Jedynym wyjściem jest użycie metody *SendKeys*. Aby odszukać żądaną sekwencję klawiszy, naciśnij klawisz *Alt*. Na przykład: aby przejść na kartę *UKLAD STRONY*, powinieneś wykonać następujące polecenie:

```
Application.SendKeys "%A"
```

Takie polecenie będzie działać poprawnie tylko wtedy, kiedy aktywne jest główne okno programu Excel. Jeżeli na przykład oknem aktywnym będzie okno VBE, to nie będziesz mógł wykonać takiego polecenia.

Moje niestandardowe menu podręczne nie działa poprawnie w Excelu 2013. Dlaczego tak się dzieje?

W Excelu 2013 wprowadzono nowy, jednodokumentowy interfejs użytkownika, gdzie każdy ze skoroszytów ma swoje własne okno i z tego powodu niestandardowe menu podręczne działają w nieco inny sposób niż w poprzednich wersjach Excela. Kiedy za pomocą kodu VBA modyfikujesz menu podręczne, to takie zmiany mają miejsce tylko i wyłącznie w aktywnym skoroszytcie. Jeżeli chcesz, aby wprowadzane zmiany działały we wszystkich skoroszytach, będziesz musiał napisać nieco dodatkowego kodu, który za pomocą pętli dokona zmian we wszystkich otwartych skoroszytach. Niestety nawet wtedy nie możesz mieć pewności, że modyfikacje menu podręcznego będą poprawnie działać we wszystkich skoroszytach.

Czy oznacza to, że w Excelu 2013 nie można wykorzystywać niestandardowego menu podręcznego do uruchamiania makr przechowywanych w dodatkach?

Menu podręczne są nadal dobrym rozwiązaniem, z tym że powinieneś je modyfikować za pomocą kodu RibbonX, a nie procedur VBA.

Dodatki

W tej części:

Dodatek A „Instrukcje i funkcje VBA”

Dodatek B „Kody błędów VBA”

Dodatek C „Strona internetowa książki”

Skorowidz

Dodatek A

Instrukcje i funkcje VBA

W tym dodatku zamieszczono listę wszystkich wbudowanych instrukcji (tabela A.1) i funkcji (tabela A.2) języka VBA. Szczegółowe informacje na ich temat możesz znaleźć w pomocy systemowej programu Excel.



W programie Excel 2013 nie zostały wprowadzone żadne nowe instrukcje języka Visual Basic for Applications (VBA).

Tabela A.1. Zestawienie instrukcji języka VBA

Instrukcja	Działanie
AppActivate	Uaktywnia okno aplikacji
Beep	Generuje dźwięk poprzez głośnik systemowy
Call	Przekazuje sterowanie do innej procedury
ChDir	Zmienia bieżący katalog
ChDrive	Zmienia bieżący napęd
Close	Zamyka plik tekstowy
Const	Deklaruje wartość stałą
Date	Ustawia bieżącą datę systemową
Declare	Deklaruje odwołania do zewnętrznej procedury w bibliotece DLL (ang. <i>Dynamic Link Library</i>)
DefBool	Ustawia domyślny typ danych na Boolean dla zmiennych rozpoczynających się od określonej litery
DefByte	Ustawia domyślny typ danych na Byte dla zmiennych rozpoczynających się od określonej litery
DefCur	Ustawia domyślny typ danych na Currency dla zmiennych rozpoczynających się od określonej litery
DefDate	Ustawia domyślny typ danych na Date dla zmiennych rozpoczynających się od określonej litery
DefDec	Ustawia domyślny typ danych na Decimal dla zmiennych rozpoczynających się od określonej litery

Tabela A.1. Zestawienie instrukcji języka VBA — ciąg dalszy

Instrukcja	Działanie
DefDb1	Ustawia domyślny typ danych na Double dla zmiennych rozpoczynających się od określonej litery
DefInt	Ustawia domyślny typ danych na Integer dla zmiennych rozpoczynających się od określonej litery
DefLng	Ustawia domyślny typ danych na Long dla zmiennych rozpoczynających się od określonej litery
DefObj	Ustawia domyślny typ danych na Object dla zmiennych rozpoczynających się od określonej litery
DefSng	Ustawia domyślny typ danych na Single dla zmiennych rozpoczynających się od określonej litery
DefStr	Ustawia domyślny typ danych na String dla zmiennych rozpoczynających się od określonej litery
DefVar	Ustawia domyślny typ danych na Variant dla zmiennych rozpoczynających się od określonej litery
DeleteSetting	Usuwa sekcję lub ustawienia klucza w rejestrze Windows
Dim	Deklaruje zmienne i (opcjonalnie) ich typy danych
Do-Loop	Blok pętli programowej
End	Instrukcja wykorzystana samodzielnie powoduje zakończenie działania programu. Używa się jej także do zakończenia bloku instrukcji rozpoczynających się od słów kluczowych If, With, Sub, Function, Property, Type oraz Select
Enum	Deklaruje typy wyliczeniowe
Erase	Dokonuje ponownego zainicjowania tablicy
Error	Symuluje wystąpienie określonego błędu
Event	Deklaruje zdarzenia użytkownika
Exit Do	Wychodzi z bloku pętli Do ... Loop
Exit For	Wychodzi z bloku pętli For ... Next
Exit Function	Wychodzi z funkcji
Exit Property	Wychodzi z procedury Property
Exit Sub	Wychodzi z procedury Sub
FileCopy	Kopiuje plik
For Each ... Next	Blok pętli programowej wykonywany dla każdego elementu serii
For ... Next	Blok pętli programowej wykonywany określoną liczbę iteracji
Function	Deklaruje nazwy i argumenty funkcji
Get	Odczytuje dane z pliku tekstowego
GoSub ... Return	Wywołanie i powrót z procedury

Tabela A.1. Zestawienie instrukcji języka VBA — ciąg dalszy

Instrukcja	Działanie
GoTo	Przekazuje sterowanie do określonego miejsca procedury
If-Then-Else	Blok instrukcji warunkowej
Implements	Okręsła interfejs lub klasę, która ma być zaimplementowana w module klasy
Input #	Odczytuje dane z sekwencyjnego pliku tekstowego
Kill	Usuwa plik z dysku
Let	Przypisuje wartość wyrażenia do zmiennej lub właściwości
Line Input #	Odczytuje wiersz danych z sekwencyjnego pliku tekstowego
Load	Laduje obiekt, ale go nie wyświetla
Lock ... Unlock	Steruje dostępem do pliku tekstowego
Lset	Wyrównuje do lewej ciąg znaków w zmiennej tekstowej
Mid	Zastępuje ciąg znaków w łańcuchu innym ciągiem znaków
MkDir	Tworzy nowy katalog na dysku
Name	Zmienia nazwę pliku lub katalogu
On Error	Przekazuje sterowanie do bloku obsługi błędu
On ... GoSub	Warunkowe przekazanie sterowania do procedury
On ... GoTo	Warunkowe przekazanie sterowania do określonego miejsca procedury
Open	Otwiera plik tekstowy
Option Base	Zmienia domyślną wartość dolnego ograniczenia indeksu tablic
Option Compare	Deklaruje domyślny tryb porównywania ciągów znaków
Option Explicit	Wymusza deklarację wszystkich zmiennych w module
Option Private	Deklaruje moduł jako prywatny
Print #	Zapisuje dane do pliku sekwencyjnego
Private	Deklaruje lokalną tablicę lub zmienną
Property Get	Deklaruje nazwy i argumenty procedury Property Get
Property Let	Deklaruje nazwy i argumenty procedury Property Let
Property Set	Deklaruje nazwy i argumenty procedury Property Set
Public	Deklaruje publiczne tablice lub zmienne
Put	Zapisuje zmienną do pliku tekstowego
RaiseEvent	Generuje zdarzenie zdefiniowane przez użytkownika
Randomize	Inicjuje generator liczb losowych
ReDim	Zmienia wymiary tablicy
Rem	Deklaruje dany wiersz jako komentarz (to samo, co znak apostrofu ['])

Tabela A.1. Zestawienie instrukcji języka VBA — ciąg dalszy

Instrukcja	Działanie
Reset	Zamyka wszystkie otwarte pliki tekstowe
Resume	Wznawia wykonywanie programu po zakończeniu działania procedury obsługi błędu
RmDir	Usuwa pusty katalog
RSet	Wyrównuje ciąg znaków w zmiennej tekstowej do prawej
SaveSetting	Zapisuje lub tworzy wpis w rejestrze Windows
Seek	Ustawia pozycję kolejnego dostępu do pliku tekstowego
Select Case	Blok warunkowy
SendKeys	Wysyła ciąg klawiszy do aktywnego okna
Set	Przypisuje adres obiektu do zmiennej lub właściwości
SetAttr	Zmienia atrybuty pliku
Static	Deklaruje zmienne na poziomie procedury w taki sposób, że zmienne zachowują swoje wartości przez cały okres działania programu
Stop	Zatrzymuje działanie programu
Sub	Deklaruje nazwę i argumenty procedury Sub
Time	Ustawia czas systemowy
Type	Definicja niestandardowego typu danych
Unload	Usuwa obiekt z pamięci
While ... Wend	Blok pętli programowej wykonywany, dopóki warunek jest spełniony
Width #	Ustawia wyjściową szerokość wiersza pliku tekstowego
With	Ustawia szereg właściwości obiektu
Write #	Zapisuje dane do sekwencyjnego pliku tekstowego

Wywoływanie funkcji Excela w instrukcjach VBA

Jeżeli funkcja VBA będąca odpowiednikiem funkcji Excela nie jest dostępna, możesz użyć funkcji Excela bezpośrednio w kodzie VBA. Aby to zrobić, powinieneś poprzedzić wywołanie żądanej funkcji odwołaniem do obiektu WorksheetFunction. Na przykład w języku VBA nie jest dostępna funkcja przekształcająca radiany na stopnie. Ponieważ w Excelu jedna z funkcji arkusza wykonuje taką operację, możesz wykorzystać następującą instrukcję VBA:

```
Deg = Application.WorksheetFunction.Degrees(3.14)
```



W programie Excel 2013 nie zostały wprowadzone żadne nowe funkcje języka Visual Basic for Applications (VBA).

Tabela A.2. Zestawienie funkcji języka VBA

Funkcja	Działanie
Abs	Zwraca wartość bezwzględną liczby
Array	Zwraca zmienną typu Variant zawierającą tablicę
Asc	Przekształca pierwszy znak łańcucha znaków na wartość kodu ASCII
Atn	Zwraca wartość arcus tangens liczby
CallByName	Wykonuje metodę albo ustawia lub zwraca wartość właściwości
CBool	Przekształca wyrażenie na typ danych Boolean
CByte	Przekształca wyrażenie na typ danych Byte
CCur	Przekształca wyrażenie na typ danych Currency
CDate	Przekształca wyrażenie na typ danych Date
CDbl	Przekształca wyrażenie na typ danych Double
CDec	Przekształca wyrażenie na typ danych Decimal
Choose	Wybiera i zwraca wartość z listy argumentów
Chr	Przekształca kod znaku na znak
CInt	Przekształca wyrażenie na typ danych Integer
CLng	Przekształca wyrażenie na typ danych Long
Cos	Zwraca wartość cosinusa liczby
CreateObject	Tworzy obiekt automatyzacji OLE
CSng	Przekształca wyrażenie na typ danych Single
CStr	Przekształca wyrażenie na typ danych String
CurDir	Zwraca ścieżkę bieżącego katalogu
CVar	Przekształca wyrażenie na typ danych Variant
CVDate	Przekształca wyrażenie na typ danych Date (polecenie niezalecane)
CVErr	Zwraca błąd zdefiniowany przez użytkownika, odpowiadający podanemu kodowi
Date	Zwraca bieżącą datę systemową
DateAdd	Dodaje przedział czasu do danej typu Date
DateDiff	Zwraca przedział czasu pomiędzy dwiema datami
DatePart	Zwraca określoną część daty
DateSerial	Przekształca datę na liczbę seryjną
DateValue	Przekształca ciąg znaków na datę
Day	Zwraca dzień miesiąca na podstawie danej typu Date

Tabela A.2. Zestawienie funkcji języka VBA — ciąg dalszy

Funkcja	Działanie
DDB	Zwraca spadek wartości środka trwałego
Dir	Zwraca nazwę pliku lub katalogu odpowiadającą wzorcowi
DoEvents	Przekazuje sterowanie do systemu operacyjnego w celu obsługi zdarzeń
Environ	Zwraca zmienną środowiskową systemu operacyjnego
EOF	Zwraca wartość True w przypadku osiągnięcia końca pliku tekstowego
Error	Zwraca komunikat o błędzie odpowiadający podanemu kodowi
Exp	Zwraca podstawę logarytmu naturalnego (<i>e</i>) podniesioną do potęgi
FileAttr	Zwraca tryb pliku tekstowego
FileDateTime	Zwraca datę i godzinę ostatniej modyfikacji pliku
FileLen	Zwraca liczbę bajtów w pliku
Filter	Zwraca podzbiór tablicy łańcuchów znaków z uwzględnieniem filtra
Fix	Zwraca część całkowitą liczby
Format	Wyświetla wyrażenie w określonym formacie
FormatCurrency	Zwraca sformatowane wyrażenie z systemowym znakiem waluty
FormatDateTime	Zwraca wyrażenie sformatowane jako data i godzina
FormatNumber	Zwraca wyrażenie sformatowane jako liczba
FormatPercent	Zwraca wyrażenie sformatowane jako procent
FreeFile	Zwraca kolejny dostępny numer pliku
FV	Zwraca przyszłą wartość rocznej spłaty
GetAllSettings	Zwraca listę ustawień i wartości z rejestru Windows
GetAttr	Zwraca kod reprezentujący atrybut pliku
GetObject	Pobiera obiekt automatyzacji OLE z pliku
GetSetting	Zwraca określone ustawienie z rejestru Windows
Hex	Przekształca wartość dziesiętną na szesnastkową
Hour	Zwraca wartość godziny na podstawie danej określającej czas
IIf	Wylicza wartość wyrażenia i zwraca zależnie od tego wyrażenia jeden z dwóch argumentów
Input	Zwraca znaki z sekwencyjnego pliku tekstowego
InputBox	Wyświetla okno umożliwiające wprowadzanie danych użytkownika
InStr	Zwraca pozycję łańcucha znaków w innym łańcuchu znaków
InStrRev	Zwraca pozycję łańcucha znaków w innym łańcuchu znaków, zaczynając sprawdzanie od jego końca
Int	Zwraca część całkowitą liczby

Tabela A.2. Zestawienie funkcji języka VBA — ciąg dalszy

Funkcja	Działanie
IPmt	Zwraca kwotę odsetek dla określonego okresu spłaty
IRR	Zwraca wewnętrzną stopę zwrotu dla ciągu przepływów gotówkowych
IsArray	Zwraca wartość True, jeśli zmienna jest tablicą
IsDate	Zwraca wartość True, jeśli zmienna jest danymi typu Date
IsEmpty	Zwraca wartość True, jeśli zmienna nie została zainicjowana
IsError	Zwraca wartość True, jeśli wyrażenie jest danymi typu Error
IsMissing	Zwraca wartość True, jeżeli do procedury nie przekazano opcjonalnego argumentu
IsNull	Zwraca wartość True, jeśli wyrażenie zawiera wartość Null
IsNumeric	Zwraca wartość True, jeśli wyrażeniu można nadać wartość liczby
IsObject	Zwraca wartość True, jeśli w wyrażeniu występuje odwołanie do obiektu automatyzacji OLE
Join	Łączy łańcuchy znaków zapisane w tablicy
LBound	Zwraca najmniejszy indeks tablicy
LCase	Zwraca łańcuch znaków przekształcony na małe litery
Left	Zwraca określoną liczbę znaków z lewej strony łańcucha znaków
Len	Zwraca liczbę znaków w łańcuchu znaków
Loc	Zwraca bieżącą pozycję do odczytu lub zapisu w pliku tekstowym
LOF	Zwraca liczbę bajtów otwartego pliku tekstowego
Log	Zwraca logarytm naturalny z liczby
LTrim	Zwraca kopię łańcucha znaków niezawierającą wiodących spacji
Mid	Zwraca określoną liczbę znaków z łańcucha
Minute	Zwraca minutę na podstawie danych reprezentujących godzinę
MIRR	Zwraca zmodyfikowaną wewnętrzną stopę zwrotu dla ciągu okresowych przepływów gotówkowych
Month	Zwraca miesiąc na podstawie daty
MonthName	Zwraca miesiąc w postaci ciągu znaków
MsgBox	Wyświetla modalne okno informacyjne
Now	Zwraca bieżącą datę i godzinę systemową
NPer	Zwraca liczbę okresów spłaty
NPV	Zwraca obecną wartość netto inwestycji
Oct	Przekształca liczbę z postaci dziesiętnej na ósemkową
Partition	Zwraca łańcuch znaków reprezentujący zakres, do którego należy wartość
Pmt	Zwraca kwotę spłaty pożyczki

Tabela A.2. Zestawienie funkcji języka VBA — ciąg dalszy

Funkcja	Działanie
Pmt	Zwraca wartość kapitału dla raty pożyczki
PV	Zwraca obecną wartość inwestycji
QBColor	Zwraca kod koloru RGB
Rate	Zwraca oprocentowanie dla okresu spłaty pożyczki
Replace	Zwraca łańcuch znaków, w którym określony łańcuch jest zastępowany innym łańcuchem znaków
RGB	Zwraca liczbę reprezentującą wartość RGB koloru
Right	Zwraca określoną liczbę znaków z prawej strony łańcucha
Rnd	Zwraca liczbę losową z zakresu od 0 do 1
Round	Zwraca liczbę po zaokrągleniu
RTrim	Zwraca kopię łańcucha znaków pozbawioną końcowych spacji
Second	Zwraca część sekundową na podstawie danych reprezentujących godzinę
Seek	Zwraca bieżącą pozycję w pliku tekstowym
Sgn	Zwraca liczbę całkowitą reprezentującą znak liczby
Shell	Uruchamia program wykonywalny
Sin	Zwraca sinus liczby
SLN	Zwraca amortyzację środka trwałego za dany okres
Space	Zwraca łańcuch znaków zawierający określoną liczbę spacji
Spc	Ustawia pozycję podczas wydruku do pliku
Split	Zwraca jednowymiarową tablicę zawierającą łańcuchy znaków powstałe wskutek podzielenia wejściowego łańcucha znaków na mniejsze części (na podstawie określonego znaku rozdzielającego)
Sqr	Zwraca pierwiastek kwadratowy z liczby
Str	Zwraca znakową reprezentację liczby
StrComp	Zwraca wartość reprezentującą wynik porównywania łańcuchów znaków
StrConv	Zwraca łańcuch znaków po konwersji
String	Zwraca łańcuch znaków składający się z powtarzających się znaków
StrReverse	Zwraca odwrócony łańcuch znaków
Switch	Ocenia wartość ciągu wyrażeń typu Boolean i zwraca wartość związaną z pierwszym wyrażeniem, które ma wartość True
SYD	Zwraca amortyzację środka trwałego za dany okres metodą sumy cyfr wszystkich lat amortyzacji
Tab	Ustawia pozycję wydruku podczas drukowania do pliku
Tan	Zwraca tangens liczby

Tabela A.2. Zestawienie funkcji języka VBA — ciąg dalszy

Funkcja	Działanie
Time	Zwraca bieżącą godzinę systemową
Timer	Zwraca liczbę sekund, jakie uplynęły od północy
TimeSerial	Zwraca liczbę całkowitą reprezentującą czas dla podanej godziny, minuty i sekundy
TimeValue	Przekształca łańcuch znaków na liczbę całkowitą, reprezentującą czas
Trim	Zwraca łańcuch znaków bez wiodących i (lub) końcowych spacji
TypeName	Zwraca łańcuch znaków opisujący typ danych zmiennej
UBound	Zwraca największy indeks tablicy
UCase	Przekształca łańcuch znaków na wielkie litery
Val	Zwraca wartości liczb tworzących ciąg w postaci wartości numerycznej odpowiedniego typu.
VarType	Zwraca wartość reprezentującą podtyp zmiennej
Weekday	Zwraca liczbę reprezentującą dzień tygodnia
WeekdayName	Zwraca łańcuch znaków reprezentujący dzień tygodnia
Year	Zwraca rok na podstawie danej typu Date

Dodatek B

Kody błędów VBA

W tym dodatku zamieszczono pełną listę kodów błędów języka Visual Basic for Applications (VBA) przydatnych podczas śledzenia i obsługi błędów. Szczegółowe informacje o poszczególnych błędach można znaleźć w pomocy systemowej programu Excel.

Kod błędu	Opis
3	Użycie instrukcji Return bez instrukcji GoSub
5	Niepoprawne wywołanie lub niepoprawny argument procedury
6	Przepelnienie (np. zbyt duża wartość dla liczby typu Integer)
7	Brak pamięci. Ten błąd rzadko jest związany z ilością pamięci fizycznie zainstalowanej w systemie i dotyczy raczej stałego obszaru pamięci wykorzystywanego przez Excela lub system Windows (np. obszaru do przechowywania grafiki lub formatów użytkownika)
9	Indeks poza zakresem. Ten komunikat o błędzie uzyskamy również w przypadku problemów ze znalezieniem w zbiorze obiektów elementu o określonej nazwie (np. jeżeli w kodzie występuje odwołanie do Sheets("Arkusz2"), podczas gdy Arkusz2 nie istnieje)
10	Tablica stała lub tymczasowo zablokowana
11	Dzielenie przez zero
13	Niezgodne typy
14	Brak miejsca na łańcuchy znaków
16	Wyrażenie zbyt złożone
17	Nie można wykonać żądanej operacji
18	Przerwanie użytkownika. Ten błąd powstaje w sytuacji, kiedy użytkownik przerwie makro, klikając przycisk <i>Anuluj</i>
20	Próba wykonania instrukcji Resume bez wystąpienia błędu. Ten kod oznacza prawdopodobnie, że programista zapomniał wstawić instrukcję Exit Sub przed wywołaniem obsługi błędu
28	Brak miejsca na stosie
35	Nie zdefiniowano procedury Sub lub Function
47	Zbyt wiele klientów aplikacji DLL

Kod błędu	Opis
48	Błąd ładowania biblioteki DLL
49	Błędna konwencja wywołania DLL
51	Błąd wewnętrzny
52	Błędna nazwa pliku lub liczba
53	Nie znaleziono pliku
54	Niepoprawny tryb pliku
55	Plik jest już otwarty
57	Błąd wejścia-wyjścia urządzenia
58	Plik już istnieje
59	Błędna długość rekordu
61	Dysk jest pełny
62	Próba wprowadzenia danych poza koniec pliku
63	Niepoprawny numer rekordu
67	Zbyt wiele plików
68	Urządzenie niedostępne
70	Brak uprawnień
71	Dysk nie jest gotowy
74	Nie można zmienić nazwy z użyciem innego napędu
75	Błąd dostępu do pliku lub katalogu
76	Nie znaleziono katalogu
91	Nie ustalono zmiennej obiektowej lub zmiennej bloku With. Ten kod błędu pojawia się wtedy, gdy nie wprowadzimy instrukcji Set na początku instrukcji tworzącej zmienną obiektową, albo w przypadku próby odwołania do obiektu arkusza (np. ActiveCell) w momencie, kiedy jest aktywny arkusz wykresu
92	Nie zainicjowano pętli For
93	Niepoprawny łańcuch wzorca
94	Nieprawidłowe użycie wartości Null
96	Nie można generować nowych zdarzeń, ponieważ obiekt obecnie generuje zdarzenia dla maksymalnej liczby odbiorców
97	Nie można wywołać funkcji zaprzyjaźnionej obiektu, który nie jest egzemplarzem klasy definiującej
98	Wywołanie właściwości lub metody nie może zawierać odwołania do prywatnego obiektu, ani w postaci argumentu, ani w postaci zwracanej wartości
321	Niepoprawny format pliku
322	Nie można utworzyć potrzebnego pliku tymczasowego

Kod błędu	Opis
325	Niepoprawny format w pliku zasobów
380	Niepoprawna wartość właściwości
381	Niepoprawny indeks tablicy właściwości
382	Ustawianie wartości nie jest obsługiwane w fazie wykonywania programu
383	Ustawianie wartości nie jest obsługiwane (właściwość tylko do odczytu)
385	Wymagany indeks tablicy właściwości
387	Instrukcja Set niedozwolona
393	Pobieranie wartości nie jest obsługiwane w fazie wykonywania programu
394	Pobieranie wartości nieobsługiwane (właściwość tylko do zapisu)
422	Nie znaleziono właściwości
423	Nie znaleziono właściwości lub metody
424	Wymagany obiekt. Ten błąd występuje wtedy, kiedy tekst poprzedzający kropkę nie został rozpoznany jako obiekt
429	Nieudana próba utworzenia obiektu przez komponent ActiveX (być może problem z rejestracją biblioteki, do której użyto odwołania)
430	Klasa nie obsługuje automatyzacji lub nie implementuje określonego interfejsu
432	Nie znaleziono nazwy pliku lub klasy podczas operacji automatyzacji
438	Obiekt nie obsługuje tej właściwości lub metody
440	Błąd automatyzacji
442	Ultracono połaczenie z biblioteką typów procesu zdalnego. Aby usunąć odwołanie, należy kliknąć <i>OK</i>
443	Obiekt automatyzacji nie ma wartości domyślnej
445	Obiekt nie obsługuje tego działania
446	Obiekt nie obsługuje nazwanych argumentów
447	Obiekt nie obsługuje bieżącego ustawienia lokalizacji
448	Nie znaleziono nazwanego argumentu
449	Argument nie jest opcjonalny
450	Niewłaściwa liczba argumentów lub nieprawidłowe przypisanie właściwości
451	Nie zdefiniowano procedury Property Let, a procedura Property Get nie zwróciła obiektu
452	Nieprawidłowa liczba porządkowa
453	Nie znaleziono określonej funkcji DLL
454	Nie znaleziono zasobu kodu
455	Blokada zasobu kodu
457	Klucz jest już powiązany z elementem tego zbioru

Kod błędu	Opis
458	Dla zmiennej wykorzystano typ automatyzacji nieobsługiwany w języku Visual Basic
459	Obiekt lub klasa nie obsługuje zestawu zdarzeń
460	Nieprawidłowy format schowka
461	Nie znaleziono metody lub składowej
462	Zdalny serwer nie istnieje lub jest niedostępny
463	Na lokalnym komputerze nie zarejestrowano klasy
481	Nieprawidłowy obraz
482	Błąd wydruku
735	Nie można zapisać pliku do katalogu <i>TEMP</i>
744	Nie znaleziono wyszukiwanego tekstu
746	Zbyt długi zastępowany tekst
1004	Błąd aplikacji lub obiektu. Jest to bardzo popularny komunikat o błędzie generowany wówczas, gdy powstały błąd nie został zdefiniowany w języku VBA. Mówiąc inaczej, błąd został zdefiniowany w Excelu (lub przez inny obiekt) i nastąpiła jego propagacja do podsystemu VBA

Dodatek C

Strona internetowa książki

W tym dodatku znajdziesz opis skoroszytów przykładowych, które zostały utworzone na potrzeby tej książki. Aby pobrać pliki przykładów, powinieneś uruchomić swoją ulubioną przeglądarkę sieciową i zajrzeć na stronę internetową <http://www.helion.pl/ksiazki/e23pvw.htm>.

Archiwum zawiera ponad 300 plików, których używaliśmy jako przykładów w poszczególnych podrozdziałach. Pliki są pogrupowane według kolejnych rozdziałów. Z wyjątkiem kilku plików wszystkie pliki przykładowe są plikami programu Excel 2013 i posiadają jedno z następujących rozszerzeń:

- *.xlsx* — skoroszyt programu Excel.
- *.xlsm* — skoroszyt programu Excel zawierający makra VBA.
- *.xlam* — dodatek programu Excel zawierający makra VBA.

Kiedy otwierasz plik XLSM, Excel może wyświetlić ostrzeżenie, że obsługa makr została wyłączona. Aby włączyć obsługę makr, powinieneś w oknie ostrzeżenia nacisnąć przycisk *Opcje* i wybrać opcję *Włącz tą zawartość*.

Ponieważ pliki pochodzą z pewnego, zaufanego źródła, możesz zapisać je na dysk twardego i umieścić w bezpiecznej lokalizacji, w której przechowujesz zaufane pliki. Aby to zrobić, powinieneś wykonać polecenia opisane poniżej:

1. Uruchom Excela, przejdź na kartę *PLIK* i z menu wybierz polecenie *Opcje*. Na ekranie pojawi się okno dialogowe *Opcje programu Excel*.
2. W oknie dialogowym *Opcje programu Excel* odszukaj i kliknij kategorię *Centrum zaufania*.
3. Naciśnij przycisk *Ustawienia Centrum zaufania*.
4. W oknie dialogowym *Centrum zaufania*, które pojawi się na ekranie, kliknij kategorię *Zaufane lokalizacje*.
5. Naciśnij przycisk *Dodaj nową lokalizację*. Na ekranie pojawi się okno dialogowe *Zaufana lokalizacja pakietu Microsoft Office*.
6. Naciśnij przycisk *Przeglądaj* i odszukaj folder zawierający pliki przykładów, pobrane z naszej strony internetowej.

7. Upewnij się, że zaznaczyłeś opcję *Podfoldery tej lokalizacji są także zaufane* i naciśnij przycisk *OK*.

Po wykonaniu polecień opisanych powyżej będziesz mógł z tej lokalizacji otwierać sko-roszty zawierające makra, a Excel nie będzie wyświetlał na ekranie żadnych ostrzeżeń.

Poniżej przedstawiono pełną listę przykładowych plików, wraz z krótkimi opisami.



W niektórych rozdziałach nie używano żadnych plików przykładowych.

Rozdział 2.

- *Funkcje wyszukiwania i adresu.xlsx* — skoroszyt zawierający przykłady zastosowania podstawowych funkcji wyszukiwania i adresu.
- *Kalendarz na cały rok.xlsx* — skoroszyt zawierający kalendarz na cały rok, generowany przy użyciu formuł tablicowych.
- *Megaformuła.xlsm* — skoroszyt zawierający przykład formuły pośredniej, megaformuły oraz funkcji VBA zdefiniowanej przez użytkownika.
- *Nazwane formuły.xlsx* — przykłady użycia nazwanych formuł.
- *Przykłady formuł tablicowych.xlsx* — skoroszyt zawierający różne przykłady formuł tablicowych.
- *Zaawansowane funkcje wyszukiwania i adresu.xlsx* — skoroszyt zawierający przykłady zastosowania zaawansowanych funkcji wyszukiwania i adresu.
- *Zliczanie i sumowanie.xlsx* — skoroszyt zawierający przykłady formuł używanych do zliczania i sumowania.

Rozdział 3.

- *Przykład.xlsm* — przykład ilustrujący strukturę pliku skoroszytu programu Excel.

Rozdział 4.

- *Formanty arkusza.xlsx* — skoroszyt zawierający przykłady zastosowania formantów ActiveX na arkuszu (bez użycia makr).

Rozdział 5.

- *Obiekt Comment.xlsm* — skoroszyt zawierający przykłady kodu VBA wykorzystującego obiekty typu *Comment*.

Rozdział 6.

- *Pomiar czasu.xlsm* — skoroszyt zawierający procedurę VBA ilustrującą różnice w czasie wykonania kodu dla danych typu *Variant* i danych o ściśle zadeklarowanym typie danych.

Rozdział 7.

- *Sortowanie arkuszy.xlsxm* — skoroszyt zawierający makro sortujące arkusze skoroszytu.

Rozdział 8.

- *Argumenty tablicowe.xlsxm* — skoroszyt zawierający przykłady funkcji wykorzystującej argumenty tablicowe.
- *Funkcja MySum.xlsxm* — skoroszyt zawierający funkcję MySum, która emuluje funkcję SUMA Excela.
- *Funkcja RemoveVowels.xlsxm* — skoroszyt zawierający funkcję usuwającą samogłoski z łańcucha tekstów będącego jej argumentem.
- *Funkcje bez argumentów.xlsxm* — skoroszyt zawierający przykłady zastosowania funkcji bezargumentowych.
- *Katalog Windows.xlsxm* — skoroszyt zawierający funkcję VBA wykorzystującą funkcję API do określenia katalogu domowego systemu Windows.
- *Klawisze.xlsxm* — skoroszyt ilustrujący sposób użycia funkcji interfejsu API w celu sprawdzenia, czy wciśnięto klawisze *Shift*, *Ctrl* lub *Alt*.
- *Losowanie.xlsxm* — skoroszyt zawierający funkcję, która wybiera losowo jedną komórkę z zakresu.
- *Nazwy miesięcy.xlsxm* — skoroszyt ilustrujący sposób zwracania tablicy przez funkcję.
- *Obliczanie prowizji.xlsxm* — skoroszyt zawierający przykład funkcji wykorzystującej argumenty wywołania.
- *Rozszerzone funkcje daty.pdf* — dokument PDF opisujący rozszerzone funkcje daty.
- *Rozszerzone funkcje daty.xlsxm* — skoroszyt ilustrujący zastosowanie rozszerzonych funkcji daty do przetwarzania dat wcześniejszych niż rok 1900.
- *Wielkie litery.xlsxm* — skoroszyt zawierający funkcję, która zamienia tekst na wielkie litery.

Rozdział 9.

- *\Pobieranie wartości z zamkniętego pliku* — folder ze skorosztami ilustrującymi pobieranie danych z zamkniętego skoroszytu.
- *\Przetwarzanie wsadowe* — folder zawierający pliki danych i makro ilustrujące wsadowe przetwarzanie plików.
- *Data i czas.xlsxm* — skoroszyt zawierający procedurę, która wyświetla bieżącą datę i godzinę.
- *Funkcja CellType.xlsxm* — skoroszyt zawierający funkcję opisującą typ danych znajdujących się w komórce, której adres jest argumentem funkcji.

- *Funkcja InputBox.xlsm* — skoroszyt ilustrujący sposób pobierania danych od użytkownika.
- *Funkcja InRange.xlsm* — skoroszyt zawierający funkcję określającą, czy dany zakres znajduje się wewnątrz innego zakresu komórek.
- *Funkcja SPELDOLLARS.xlsm* — skoroszyt zawierający funkcję, która zwraca wartość numerycznego argumentu wywołania opisaną słowami.
- *Funkcje arkuszowe.xlsm* — skoroszyt zawierający kilka użytecznych funkcji arkuszowych utworzonych przy użyciu VBA.
- *Funkcje narzędziowe VBA.xlsm* — skoroszyt zawierający kilka użytecznych funkcji VBA.
- *Informacja o drukarce.xlsm* — skoroszyt zawierający wywołania funkcji API zwracających informacje o aktywnej drukarce.
- *Informacja o dyskach.xlsm* — skoroszyt zawierający makro wyświetlające informacje o wszystkich dyskach podłączonych do komputera.
- *Informacja o rozdzielcości karty graficznej.xlsm* — skoroszyt zawierający wywołania funkcji API zwracających informacje o aktualnie używanym trybie karty graficznej.
- *Kopiowanie wielu zakresów.xlsm* — skoroszyt zawierający makro kopiące dane z nieciągłych obszarów komórek.
- *Lista czcionek.xlsm* — skoroszyt zawierający makro wyświetlające listę wszystkich zainstalowanych czcionek.
- *NastępnaPustaKomórka.xlsm* — skoroszyt zawierający makro określające następną pustą komórkę w kolumnie.
- *Powielanie wierszy.xlsm* — skoroszyt zawierający makro powielające wiersze w zależności od zawartości danej komórki.
- *Przenoszenie tablicy typu Variant.xlsm* — skoroszyt zawierający makro, które przenosi dane z zakresu komórek do tablicy typu Variant.
- *Przyjazny czas.xlsm* — skoroszyt zawierający funkcję, która opisuje różnice w czasie w formacie przyjaznym dla użytkownika.
- *Rejestr systemu Windows.xlsm* — skoroszyt zawierający makra odczytujące i zapisujące dane w rejestrze systemu Windows.
- *Skojarzenia plików.xlsm* — skoroszyt zawierający wywołania funkcji API zwracających nazwy aplikacji skojarzonych z plikami danego typu.
- *Sortowanie.xlsm* — skoroszyt zawierający makra ilustrujące cztery sposoby sortowania tablicy.
- *Synchronizacja arkuszy.xlsm* — skoroszyt zawierający procedurę synchronizującą arkusze.
- *Tworzenie hiperłączy.xlsm* — skoroszyt z makrem tworzącym nowy arkusz zawierający listę hiperłączy do wszystkich arkuszy danego skoroszytu.

- *Tworzenie wydajnych pętli.xlsm* — skoroszyt ilustrujący sposób tworzenia wydajnych pętli przetwarzających zakresy komórek.
- *Ukrywanie wierszy i kolumn.xlsm* — skoroszyt zawierający makro ukrywające wszystkie wiersze i kolumny znajdujące się poza aktualnym obszarem zaznaczenia.
- *Usuń puste wiersze.xlsm* — skoroszyt zawierający procedurę usuwającą puste wiersze w skoroszycie.
- *Wypełnianie zakresu przy użyciu pętli i tablicy.xlsm* — skoroszyt zawierający makro ilustrujące różne sposoby wypełniania zakresu komórek.
- *Zaznacz zakres.xlsm* — skoroszyt zawierający makro ilustrujące sposób wprowadzania zakresów zaznaczonych przez użytkownika.
- *Zaznaczanie według wartości.xlsm* — skoroszyt zawierający makro ilustrujące sposób zaznaczania komórek w zależności od ich wartości.
- *Zaznaczanie zakresów.xlsm* — skoroszyt zawierający makra ilustrujące różne metody zaznaczania zakresów.
- *Zaznaczanie.xlsm* — skoroszyt zawierający procedurę opisującą bieżący zaznaczony zakres.

Rozdział 10.

- *Formularz wprowadzania danych.xlsm* — skoroszyt zawierający makro wyświetlające wbudowany formularz pobierania danych programu Excel.
- *Metoda InputBox.xlsm* — skoroszyt zawierający makra ilustrujące sposób użycia metody InputBox programu Excel.
- *Okno wprowadzania danych VBA.xlsm* — skoroszyt zawierający makra ilustrujące sposób użycia funkcji InputBox języka VBA.
- *Pobieranie nazwy katalogu.xlsm* — skoroszyt zawierający makro ilustrujące sposób wskazywania wybranego katalogu przez użytkownika.
- *Pobieranie nazwy pliku.xlsm* — skoroszyt zawierający makra ilustrujące sposób wskazywania wybranego pliku (lub plików) przez użytkownika.
- *Przykłady MsgBox.xlsm* — skoroszyt zawierający przykłady zastosowania funkcji MsgBox.

Rozdział 11.

- *Formanty ActiveX arkusza.xlsx* — skoroszyt ilustrujący sposób użycia formantów ActiveX na arkuszu (bez użycia makr).
- *Formanty SpinButton i TextBox.xlsm* — skoroszyt ilustrujący sposób powiązania formantu SpinButton z formantem TextBox.
- *NoweFormanty.pag* — plik zawierający niestandardowe formanty, które mogą być zaimportowane do przybornika formantów formularza w edytorze VBE i wykorzystywane na formularzach UserForm.

- *Przykład formularza UserForm.xlsx* — skoroszyt zawierający przykład prostego formularza *UserForm*.
- *Wszystkie formanty formularza.xlsx* — skoroszyt zawierający formularz *UserForm*, na którym umieszczone zostały dostępne formanty formularza.
- *Zdarzenia formantu SpinButton.xlsx* — skoroszyt zawierający ilustrujący zdarzenia formantu *SpinButton*.
- *Zdarzenia formularza UserForm.xlsx* — skoroszyt zawierający ilustrujący zdarzenia formularzy *UserForm*.

Rozdział 12.

- *\MediaPlayer* — folder zawierający skoroszyt *MediaPlayer.xlsx* (ilustrujący sposób użycia formantu *MediaPlayer*) oraz kilka przykładowych plików MP3.
- *Aktywacja arkusza.xlsx* — skoroszyt ilustrujący sposób, w jaki użytkownik może zaznaczyć arkusz przy użyciu formantu *ListBox*.
- *Elementy listy formantu ListBox.xlsx* — skoroszyt ilustrujący sposoby wypełniania listy elementów formantu *ListBox* na formularzach *UserForm*.
- *Elementy unikatowe1.xlsx* — skoroszyt ilustrujący sposób wypełniania listy formantu *ListBox* elementami unikatowymi.
- *Elementy unikatowe2.xlsx* — modyfikacja skoroszytu *Elementy unikatowe1.xlsx*, podczas której dodatkowo unikatowe elementy listy są sortowane.
- *Formant DateAndTimePicker.xlsx* — skoroszyt ilustrujący sposób użycia formantu *DateAndTimePicker*.
- *Formant MultiPage.xlsx* — skoroszyt ilustrujący sposób użycia formantu *MultiPage* na formularzach *UserForm*.
- *Generator liczb losowych.xlsx* — skoroszyt ilustrujący sposób tworzenia prostych animacji na formularzach *UserForm* Excela.
- *Menu formularzy UserForm.xlsx* — skoroszyt ilustrujący sposób użycia formularza *UserForm* do utworzenia prostego menu makr.
- *Okno powitalne.xlsx* — skoroszyt prezentujący sposób użycia formularza *UserForm* do utworzenia okna powitalnego wyświetlanego w momencie otwarcia skoroszytu.
- *Powiększanie formularza.xlsx* — skoroszyt ilustrujący metodę umożliwienia użytkownikowi zmiany rozmiaru formularza *UserForm*.
- *Powiększanie i przewijanie arkusza.xlsx* — skoroszyt prezentujący sposoby powiększania i przewijania arkusza podczas wyświetlania formularza *UserForm*.
- *Przenoszenie elementów listy formantu ListBox.xlsx* — skoroszyt ilustrujący możliwości przenoszenia elementów pomiędzy dwoma formantami *ListBox*.
- *Skalowalny formularz UserForm — API.xlsx* — skoroszyt ilustrujący zastosowanie funkcji Windows API do zmiany rozmiaru formularza *UserForm*.

- *Wiele list formantu ListBox.xlsm* — skoroszyt ilustrujący sposób wyświetlania wielu list elementów w pojedynczym formancie `ListBox`.
- *Wielokolumnowe formanty ListBox1.xlsm* — skoroszyt ilustrujący zastosowanie wielokolumnowych formantów `ListBox`, wykorzystujących zakres komórek.
- *Wielokolumnowe formanty ListBox2.xlsm* — skoroszyt ilustrujący zastosowanie wielokolumnowych formantów `ListBox`, wykorzystujących tablice danych.
- *Wybieranie wierszy.xlsm* — skoroszyt ilustrujący sposób wykorzystania wielokolumnowego formantu `ListBox` do zaznaczania wierszy w arkuszu.
- *Zaznaczanie elementów listy.xlsm* — skoroszyt ilustrujący sposób identyfikacji zaznaczonych elementów listy formantu `ListBox`.
- *Zaznaczanie zakresu.xlsm* — skoroszyt ilustrujący zastosowanie na formularzach `UserForm` formantu `RefEdit`.
- *Zdarzenie QueryClose.xlsm* — skoroszyt ilustrujący technikę uniemożliwiającą zamknięcie formularza `UserForm` poprzez kliknięcie przycisku *Zamknij*.
- *Zmiana kolejności elementów listy.xlsm* — skoroszyt ilustrujący sposób przenoszenia elementów w górę lub w dół na liście formantu `ListBox`.
- *Zmiana rozmiaru formularza.xlsm* — skoroszyt ilustrujący sposób wykorzystania VBA do zmiany rozmiaru formularza `UserForm`.

Rozdział 13.

- *\dataform* — podkatalog zawierający dodatek *Enhanced DataForm* utworzony przez autora tej książki.
- *Demo kreatora.xlsm* — skoroszyt wykorzystujący formant `MultiPage` do wyświetlania na formularzu `UserForm` prostego kreatora.
- *Efekt podświetlenia.xlsm* — skoroszyt ilustrujący sposób „przygaszenia” głównego okna Excela podczas wyświetlania innego okna dialogowego.
- *Emulacja funkcji MsgBox.xlsm* — skoroszyt zawierający makro emulujące działanie funkcji `MsgBox` języka VBA.
- *Emulacja panelu zadań.xlsm* — skoroszyt demonstrujący, jak można zasymulować panel zadań Excela 2013 za pomocą formularza `UserForm`.
- *Formularz bez paska tytułowego.xlsm* — skoroszyt zawierający makra wykorzystujące funkcje API do wyświetlania formularza `UserForm` bez paska tytuловego.
- *Formularz niemodalny—SDI.xlsm* — skoroszyt demonstrujący, jak sprawić, aby niemodalny formularz był zawsze wyświetlany w oknie aktywnego skoroszytu.
- *Funkcja GetAColor.xlsm* — skoroszyt zawierający funkcję, która pozwala użytkownikowi na wybranie określonego koloru za pomocą formantów na formularzu `UserForm`.

- *Niemodalny formularz UserForm1.xls* — skoroszyt ilustrujący sposób wyświetlania niemodalnego formularza *UserForm*, prezentującego informacje o aktywnej komórce.
- *Niemodalny formularz UserForm2.xls* — bardziej rozbudowana wersja skoroszytu *Niemodalny formularz UserForm1.xls*.
- *Okno powitalne2.xls* — zmodyfikowana wersja skoroszytu *Okno powitalne.xls* (patrz rozdział 12.), wyświetlająca formularz *UserForm* (okno powitalne) pozbawiony paska tytułowego.
- *Półprzezroczysty formularz UserForm.xls* — skoroszyt ilustrujący sposób wyświetlania półprzezroczystych formularzy *UserForm*.
- *Przesuwanie formantów.xls* — skoroszyt ilustrujący możliwość przesuwania przez użytkownika formantów na formularzu *UserForm*.
- *Puzzle.xls* — skoroszyt zawierający formularz *UserForm* spełniający rolę puzzli.
- *Skalowalny formularz UserForm.xls* — skoroszyt zawierający formularz *UserForm*, którego rozmiary mogą być zmieniane przez użytkownika.
- *Symulacja paska narzędzi.xls* — skoroszyt wykorzystujący formularz *UserForm* do symulacji paska narzędzi.
- *Video poker.xls* — skoroszyt zawierający grę w pokera, zrealizowaną za pomocą formularza *UserForm*.
- *Wiele przycisków.xls* — skoroszyt ilustrujący sposób użycia modułu klasy i jednej procedury obsługi zdarzeń dla wielu formantów.
- *Wskaźnik postępu1.xls* — skoroszyt wyświetlający wskaźnik postępu na formularzu *UserForm*.
- *Wskaźnik postępu2.xls* — skoroszyt używający formantu *MultiPage* do wyświetlania wskaźnika postępu na formularzu *UserForm*.
- *Wskaźnik postępu3.xls* — skoroszyt wyświetlający wskaźnik postępu na formularzu *UserForm*, wykorzystujący zmianę rozmiaru formularza.
- *Wykres na formularzu UserForm.xls* — skoroszyt ilustrujący sposób wyświetlania wykresu na formularzu *UserForm*.

Rozdział 14.

- *\HelpSource* — katalog zawierający źródłowe pliki pomocy HTML dla dodatku *Operacje tekstowe*.
- *Operacje tekstowe.xlam* — dodatek zawierający szereg narzędzi do przetwarzania tekstu.
- *OperacjeTekstowe.chm* — skompilowany plik pomocy dla dodatku *Operacje tekstowe.xlam*.
- *Prosty przykład procedury Cofnij.xls* — skoroszyt ilustrujący metodę wycofywania efektów działania makr VBA.

Rozdział 15.

- *Analiza danych z ankiety.xlsxm* — skoroszyt zawierający makro generujące na podstawie zakresu danych 28 tabel przestawnych.
- *Dane znormalizowane.xlsx* — skoroszyt ilustrujący różnicę pomiędzy danymi zsumowanymi a znormalizowanymi.
- *Odwrócona tabela przestawna.xlsxm* — skoroszyt zawierający makro zamieniające tabelę podsumowującą w 3-kolumnową tabelę danych.
- *Prosta tabela przestawna.xlsxm* — skoroszyt zawierający dane dla prostej tabeli przestawnej.
- *Złożona tabela przestawna.xlsxm* — skoroszyt zawierający dane dla złożonej tabeli przestawnej.

Rozdział 16.

- *Arkusz wykresu — zdarzenie MouseOver.xlsxm* — skoroszyt ilustrujący zastosowanie zdarzenia MouseOver dla wykresu utworzonego na osobnym arkuszu wykresu.
- *Dane klimatyczne.xlsx* — skoroszyt zawierający wykres interaktywny, utworzony bez użycia makr.
- *Eksport wszystkich obiektów graficznych.xlsxm* — skoroszyt zawierający makro, które eksportuje wszystkie obiekty graficzne znajdujące się na arkuszu.
- *Etykiety danych.xlsxm* — skoroszyt zawierający makro, które tworzy i umieszcza na wykresie etykiety danych przechowywane w zakresie komórek.
- *Formatowanie wszystkich wykresów.xlsxm* — skoroszyt zawierający makro, które formatuje wszystkie wykresy znajdujące się na arkuszu.
- *Krzywe hipocykloidalne.xlsxm* — skoroszyt zawierający makra wyświetlające animowany wykres krzywych hipocykloidalnych.
- *Pobieranie zakresów serii.xlsxm* — skoroszyt zawierający funkcje identyfikujące zakresy danych prezentowane na wykresie.
- *Przewijanie wykresu.xlsxm* — skoroszyt ilustrujący sposób tworzenia animowanego, przewijanego wykresu.
- *PUP — Tworzenie etykiet danych.xlsxm* — skoroszyt zawierający narzędzie przeznaczone do modyfikowania etykiet na wykresach, opracowane na bazie dodatku *Power Utility Pak* (którego twórcą jest autor niniejszej książki).
- *Ukrywanie i wyświetlanie serii danych.xlsxm* — skoroszyt zawierający pola wyboru pozwalające na wskazywanie serii danych, które będą prezentowane na wykresie.
- *Wykres — zegar VBA.xlsxm* — skoroszyt zawierający przykład wykresu, którego wygląd przypomina zegar.
- *Wykres aktywnej komórki.xlsxm* — skoroszyt ilustrujący sposób modyfikacji serii danych wykresu na podstawie aktywnej komórki.

- *Wykres na formularzu UserForm.xlsx* — skoroszyt ilustrujący sposób dynamicznego tworzenia wykresu w oparciu o pozycję aktywnej komórki i wyświetlanego na formularzu *UserForm*.
- *Wykres statyczny.xlsx* — skoroszyt zawierający makra ilustrujące dwa sposoby „zerwania” dynamicznego połączenia pomiędzy danymi na arkuszu a danymi prezentowanymi na wykresie.
- *Wykresy — mapa obrazkowa.xlsx* — skoroszyt ilustrujący sposób tworzenia wykresu służącego jako rodzaj mapy obrazkowej.
- *Wykresy animowane.xlsx* — skoroszyt zawierający makra ilustrujące sposób tworzenia animowanych wykresów przy użyciu VBA.
- *Wykresy osadzone — zdarzenie MouseOver.xlsx* — skoroszyt ilustrujący zastosowanie zdarzenia *MouseOver* dla wykresu osadzonego.
- *Wykresy przebiegu w czasie.xlsx* — skoroszyt ilustrujący zastosowanie wykresów przebiegu w czasie oraz generujący raport opisujący wykresy.
- *Zdarzenia — arkusz wykresu.xlsx* — skoroszyt ilustrujący zdarzenia wykresu umieszczonego na osobnym arkuszu wykresu.
- *Zdarzenia — wykres osadzony.xlsx* — skoroszyt ilustrujący zdarzenia wykresu osadzonego na arkuszu danych.
- *Zmiana rozmiaru i wyrównania wykresów.xlsx* — skoroszyt zawierający makro zmieniające rozmiary i sposób wyrównania wszystkich wykresów na arkuszu.

Rozdział 17.

- *Cieniowanie aktywnego wiersza i kolumny.xlsx* — skoroszyt ilustrujący sposób użycia zdarzenia *Worksheet_SelectionChange* do celu podświetlenia aktywnego wiersza i kolumny.
- *Monitorowanie otwartych skoroszytów.xlsx* — skoroszyt ilustrujący sposób śledzenia otwierania skoroszytów poprzez zastosowanie modułu klasy.
- *Pogrubianie formuł.xlsx* — skoroszyt ilustrujący sposób obsługi zdarzenia *Worksheet_Change*.
- *Sprawdzanie poprawności1.xlsx* — skoroszyt ilustrujący sposób sprawdzania poprawności danych wprowadzonych do komórki za pomocą języka VBA (przy użyciu właściwości *EnableEvents*).
- *Sprawdzanie poprawności2.xlsx* — skoroszyt ilustrujący sposób sprawdzania poprawności danych wprowadzonych do komórki za pomocą języka VBA (przy użyciu zmiennej statycznej).
- *Sprawdzanie poprawności3.xlsx* — skoroszyt ilustrujący sposób sprawdzania poprawności danych wprowadzonych do komórki za pomocą wbudowanego mechanizmu sprawdzania poprawności danych programu Excel (łącznie z upewnieniem się, że kryteria poprawności danych nie zostaną skasowane).

- *Ukrywanie kolumn przed drukowaniem.xlsxm* — skoroszyt wykorzystujący zdarzenia do ukrywania kolumn przed drukowaniem i przywracania ich wyświetlania po zakończeniu drukowania.
- *Zablokuj menu podręczne.xlsxm* — skoroszyt zawierający makra wykorzystujące zdarzenie Worksheet_Open do zablokowania wyświetlania menu podręcznego oraz zdarzenia Worksheet_BeforeClose do przywracania wyświetlania menu.
- *Zdarzenia poziomu aplikacji.xlsxm* — skoroszyt ilustrujący sposób monitorowania zdarzeń poziomu aplikacji.
- *Zdarzenie OnKey.xlsxm* — skoroszyt ilustrujący sposób obsługi zdarzenia OnKey.
- *Zdarzenie OnTime.xlsxm* — skoroszyt ilustrujący sposób obsługi zdarzenia OnTime.
- *Zdarzenie Workbook_BeforeClose.xlsxm* — skoroszyt ilustrujący sposób obejścia problemu ze zdarzeniem Workbook_BeforeClose.

Rozdział 18.

- *\Automatyzacja Excela* — folder zawierający dokument Worda z makrami automatyzacji Excela.
- *\Funkcja ShellExecute* — folder zawierający skoroszyt z makrami wykorzystującymi funkcję API o nazwie ShellExecute oraz kilka dodatkowych plików.
- *Obliczanie pożyczki.xlsx* — skoroszyt wykorzystywany przez dokument *Automatyzacja Excela.docm*.
- *Okna Panelu sterowania.xlsxm* — skoroszyt zawierający makra pozwalające na wyświetlanie okien dialogowych Panelu sterowania.
- *Outlook — późne wiązanie.xlsxm* — skoroszyt zawierający makra umożliwiające wysyłanie spersonalizowanych wiadomości poczty elektronicznej z Excela za pomocą programu Outlook Express (z użyciem późnego wiązania).
- *Tworzenie notatek.xlsxm* — skoroszyt wykorzystujący automatyzację Worda w celu wygenerowania notatek na podstawie danych zapisanych w arkuszu.
- *Uruchamianie kalkulatora.xlsxm* — skoroszyt zawierający makro uruchamiające popularny kalkulator systemu Windows.
- *Wysyłanie PDF z Outlooka.xlsxm* — skoroszyt zawierający makro wysyłające przy użyciu Outlooka wiadomość poczty elektronicznej z załączonym plikiem PDF.
- *Wysyłanie wiadomości — Outlook.xlsxm* — skoroszyt zawierający makra, umożliwiające wysyłanie spersonalizowanych wiadomości poczty elektronicznej z Excela za pomocą programu Outlook Express (z użyciem wcześniego wiązania).

Rozdział 19.

- *\Eksport wykresów — pomoc* — podkatalog zawierający pliki źródłowe, na podstawie których został utworzony skompilowany plik pomocy *Eksport wykresów.chm*.
- *Eksport wykresów.chm* — plik pomocy dla skoroszytu *Eksport wykresów.xlsxm*.

- *Eksport wykresów.xlsxm* — skoroszyt zawierający narzędzie do eksportowania wykresów, które może zostać przekształcone w dodatek.
- *Informacje o dodatkach.xlsxm* — skoroszyt zawierający makro wyświetlające informacje o wszystkich zainstalowanych dodatkach.
- *Sprawdzanie dodatków.xlam* — skoroszyt zawierający makro sprawdzające, czy dodatek jest zainstalowany poprawnie.

Rozdział 20.

- *Demo formantów Wstępki.xlsxm* — skoroszyt ilustrujący kilka rodzajów formantów Wstępki.
- *Dodatek — wyświetlanie znaczników podziału stron.xlam* — dodatek umieszczający na Wstępce nowe, przydatne polecenie.
- *Menu dynamiczne.xlsxm* — skoroszyt ilustrujący zastosowanie formantu dynamicMenu.
- *Modyfikacja Wstępki.xlsxm* — skoroszyt ilustrujący prosty przykład modyfikacji Wstępki.
- *Nazwy formantów Wstępki.xlsx* — skoroszyt zawierający nazwy wszystkich formantów Wstępki.
- *Przeglądanie obrazów imageMso.xlsxm* — skoroszyt zawierający makro wyświetlające obrazy powiązane ze Wstępką.
- *Tradycyjny pasek narzędzi.xlsxm* — skoroszyt ilustrujący sposób tworzenia pasków narzędzi stosowanych w poprzednich wersjach programu Excel.
- *Wyświetlanie znaczników podziału stron.xlsxm* — skoroszyt, na podstawie którego utworzony został dodatek.

Rozdział 21.

- *Nowe menu podręczne.xlsxm* — skoroszyt zawierający makro, które tworzy nowe menu podręczne.
- *Pokaż elementy menu podręcznych.xlsxm* — skoroszyt zawierający makro, które wyświetla listę wszystkich elementów dla wszystkich menu podręcznych.
- *Pokaż nazwy menu podręcznych.xlsxm* — skoroszyt zawierający makro, które wyświetla listę nazw wszystkich menu podręcznych.
- *Pokaż obrazy FaceID.xlsxm* — skoroszyt zawierający makro, które wyświetla obrazy FaceID.
- *Resetowanie menu podręcznego.xlsxm* — skoroszyt zawiera makro, które resetuje niestandardowe menu podręczne we wszystkich otwartych oknach Excela.
- *Tworzenie menu Excel 2003.xlsxm* — skoroszyt zawierający makro, które tworzy pasek narzędzi odpowiadający menu Excela 2003.
- *Tworzenie nowego podmenu.xlsxm* — skoroszyt zawierający makro, które dodaje nowe podmenu do menu podręcznego.

- *Tworzenie nowych elementów menu podręcznego.xlsm* — skoroszyt zawierający makro, które dodaje nowy element do menu podręcznego.

Rozdział 22.

- *\Arkusz* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie arkusza do wyświetlania treści pomocy.
- *\Funkcje* — katalog, w którym znajdują się pliki ilustrujące sposób wyświetlania pomocy dla własnych funkcji arkuszowych VBA.
- *\HTML help* — katalog, w którym znajdują się pliki ilustrujące sposób tworzenia skompilowanego pliku pomocy.
- *\Komentarze* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie komentarzy komórek do utworzenia prostego systemu pomocy.
- *\MHTML* — katalog, w którym znajdują się pliki ilustrujące zastosowanie MHTML do wyświetlania treści pomocy w przeglądarce sieciowej.
- *\Pole tekstowe* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie pola tekstowego do wyświetlania pomocy.
- *\Przeglądarka sieciowa* — katalog, w którym znajdują się pliki ilustrujące zastosowanie formularza *UserForm* do wyświetlania treści pomocy.
- *\Userform1* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie formantu *SpinButton* do wyświetlania treści pomocy.
- *\Userform2* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie przewijanego formantu *Label* do wyświetlania treści pomocy.
- *\Userform3* — katalog, w którym znajduje się skoroszyt ilustrujący zastosowanie formantu *ComboBox* do wyświetlania treści pomocy.

Rozdział 23.

- *Kreator amortyzacji pożyczki.xlam* — dodatek tworzący harmonogram amortyzacji pożyczki dla kredytu o stałym oprocentowaniu.

Rozdział 24.

- *Kreator międzynarodowy.xlsm* — przykład prostego skoroszytu, w którym utworzono kreatora działającego w trzech językach.

Rozdział 25.

- *\Filtrowanie pliku tekowego* — folder zawierający skoroszyt z makrem, które odczytuje z pliku tekowego tylko wybrane informacje.
- *\Przykład ADO1* — folder zawierający przykład zastosowania ADO do pobierania danych z pliku bazy danych Access.

- \Przykład ADO2 — folder zawierający przykład zastosowania ADO do pobierania danych z pliku w formacie *CSV*.
- Eksport-import-CSV.xlsx — skoroszyt zawierający procedury umożliwiające wyeksportowanie zakresu do pliku w formacie *CSV*.
- Excel Log.xlsx — skoroszyt ilustrujący sposób logowania operacji wykonywanych przez Excela
- Export do HTML.xlsx — skoroszyt zawierający makro pozwalające na eksport danych do pliku w formacie *HTML*.
- Export do XML.xlsx — skoroszyt zawierający makro pozwalające na eksport danych do pliku w formacie *XML*.
- Funkcje plikowe.xlsx — skoroszyt zawierający funkcje `FileExists` oraz `PathExists`.
- Informacja o plikach.xlsx — skoroszyt zawierający makro tworzące listę plików wraz ze szczegółowymi informacjami o tych plikach.
- Lista plików — rekurencja.xlsx — skoroszyt zawierający makro tworzące listę wszystkich plików w danym katalogu i jego wszystkich podkatalogach.
- Lista plików.xlsx — skoroszyt zawierający makro, które tworzy listę plików znajdujących się w danym katalogu.
- Pakowanie plików ZIP.xlsx — skoroszyt zawierający makro, które pakuje pliki do archiwum ZIP.
- Pokaż informację o napędach.xlsx — skoroszyt zawierający makro wyświetlające informacje o wszystkich dyskach podłączonych do komputera.
- Rozpakowywanie plików ZIP.xlsx — skoroszyt zawierający makro, które rozpakowuje pliki z archiwum ZIP.

Rozdział 26.

- \AktualizacjaModulu — folder ze skoroszytami zawierającymi makra, które zamieniają istniejący moduł VBA na nowy.
- Dodaj 100 przycisków.xlsx — skoroszyt zawierający procedurę umieszczającą na formularzu *UserForm* 100 przycisków polecen (obiektów `CommandButton`) i tworzącą dla każdego z nich procedurę obsługi zdarzeń.
- Dodaj formularz UserForm.xlsx — skoroszyt zawierający makro, które tworzy formularz *UserForm*.
- Dodaj przycisk i kod.xlsx — skoroszyt zawierający procedurę, która umieszcza przycisk na arkuszu oraz tworzy procedurę VBA, która będzie wykonywana po naciśnięciu tego przycisku.
- Funkcja GetOption.xlsx — skoroszyt zawierający funkcję, która „w locie” tworzy formularz *UserForm* (z formantami `OptionButton`) i zwraca liczbę całkowitą, odpowiadającą opcji wybranej przez użytkownika.

- *Sprawdź ustawienia zabezpieczeń.xlsm* — skoroszyt zawiera makro, które sprawdza, czy użytkownik włączył dostęp do modelu obiektowego projektu VBA.
- *Wyświetl listę wszystkich procedur.xlsm* — skoroszyt zawierający makro, które wyświetla listę wszystkich procedur VBA znajdujących się w aktywnym skoroszycie.
- *Wyświetl składniki VB.xlsm* — skoroszyt zawierający makro, które wyświetla listę wszystkich procedur VBA znajdujących się w aktywnym skoroszycie.

Rozdział 27.

- *Klasa CSV.xlsm* — skoroszyt zawierający moduł klasy umożliwiający import i eksport plików *CSV*.
- *Klawiatura.xlsm* — skoroszyt zawierający moduł klasy, w którym zdefiniowano klasy NumLock, CapsLock oraz ScrollLock.

Rozdział 28.

- *Dopasowywanie kolorów.xlsm* — skoroszyt zawierający makra dopasowujące kolory komórek do kształtów i odwrotnie.
- *Funkcje konwersji kolorów.xlsm* — skoroszyt zawierający funkcje dokonujące konwersji kolorów pomiędzy różnymi modelami kolorów.
- *Kolory wykresów.xlsm* — skoroszyt zawierający makra, które modyfikują kolory wykresów.
- *Model RGB.xlsm* — skoroszyt zawierający interaktywną demonstrację modelu kolorów RGB.
- *Motywy dokumentów.xlsx* — skoroszyt zawierający różne elementy ilustrujące wpływ zmiany motywów na wygląd dokumentu.
- *Wybieranie wartości koloru.xlsm* — skoroszyt demonstrujący łatwy sposób wybierania kolorów przez użytkownika.
- *Właściwość TintAndShade* — skoroszyt ilustrujący sposób działania właściwości *TintAndShade*.
- *Wyświetlanie kolorów motywów.xlsm* — skoroszyt zawierający makro demonstrujące kolory motywów.

Skorowidz

.NET, 149, 699

A

- A1, 66
- ACCDB, 103
- ACCDE, 103
- Access, 855
- Activate, 446, 447, 611, 644, 646, 651, 660, 666, 935
- ActiveCell, 336
- ActiveChart, 589
- ActivePrinter, 395
- ActiveX, 258, 259, 429, 456
- ActiveX Data Objects, *Patrz:* ADO
- AddControl, 666
- AddFromFile, 861
- AddFromGuid, 861
- add-in, *Patrz:* dodatek
- AddIn, 180, 698, 706, 709, 714
 - dodawanie elementów, 714
 - właściwości, 716, 717
 - zdarzenia, 719
- AddInInstall, 644, 719, 721
- AddInUninstall, 644, 719
- AddItem, 472
- Additional Controls, 455
- Address, 184
- ADO, 855
- AfterCalculate, 662
- AfterSave, 644
 - algorytm sortowania, *Patrz:* sortowanie
 - ambiguously named procedure, *Patrz:* procedura nazwa niejednoznaczna
- Analysis ToolPak, 59, 697, 701
- animacja, 624, 628, 936
- API, 143, 326, 676, 815, 932
 - deklaracja, 327
 - funkcje, 327, 329
 - hermetyzacja, 880
 - MacOS, 818
- aplikacja, 31, 123, *Patrz też:* Application
 - aktualizacja, 140
 - arkusz kalkulacyjnego, *Patrz:* aplikacja bazodanowa, 838
 - bezpieczeństwo, 127
 - błąd, *Patrz:* błąd
 - dokumentacja, 139, 140
 - interakcje, 673, 676, 680
 - klawisze skrótu, 129
 - kliencka, 680
 - menu podręczne, *Patrz:* menu podręczne Microsoft Office, 678
 - miedzynarodowa, 821, 823
 - obsługa błędów, *Patrz:* błąd
 - projektowanie, 124, 125, 128, 133, 134, 137, 138, 141, 809, 810
 - wersje językowe, 141, 809, 815, 819, 821, 822, 823
 - przekazanie użytkownikom, 140
 - przyjazna dla użytkownika, 799, 809, 810
 - struktura, 127
 - system pomocy, 139, 781, 784, 786, 796
 - testowanie, 134, 135, 136
 - uaktywnienie, 677
 - uniwersalność, 128
 - uruchamianie, 673, 676, 677
 - użytkownik, 125, 126, 128, 135
 - wersja beta, 135
 - wstrzymywanie, 674
 - wydajność, 127, 142
 - wygląd, 137
 - zamiana w plik wykonywalny, 919
 - zdarzenie, 638, 660, 662, 664
- AppActivate, 677
- Application, 150, 180, 181, 191, 227, 259, 688
 - Dialogs, 946
 - International, 823
 - OperatingSystem, 817
 - Transpose, 473
 - właściwości, 192
- Apps for Office, 36
- Areas, 344
- arkusz, 32, 33, 148, 150, 180, 181
 - aktywny, 32, 192
 - blokowanie obiektów, 137
 - dialogowy, 32, 35

- arkusz
drukowanie, 648
funkcja, *Patrz*: funkcja arkuszowa
kolor karty, 922
liczba wierszy, 382
makr
 MS Excel 4, 34
 osobisty, *Patrz*: makro arkusz osobisty
 XLM, 32, 34
nazwa, 32, 274, 374, 588
obszar roboczy, 729
ochrona, *Patrz*: ochrona
przeliczanie, 662, 920
przetwarzanie, 360
przewijanie, 786
 blokowanie, 935
synchronizacja, 363
system pomocy, 786
ukrywanie, 34, 137
 komórek, 361
ustawienia domyślne, 110
warstwa rysunkowa, 34, 131
współczynnik powiększenia, 77
wykres, 32, 34, 55, 579
 aktywny, 192
 osadzony, 582, 584
wymiary, 33
zabezpieczanie hasłem, 127
zdarzenie, 637, 651, 652, 653, 657, 658, 659
- Array, 308
- array formula, *Patrz*: formuła tablicowa
- As, 291
- Auto Data Tips, 168
- Auto Indent, 168
- Auto List Members, 167, 682, 885, 891, 909, 934
- Auto Quick Info, 168
- Auto Syntax Check, 166, 205
- automatyzacja, 680
 aplikacja kliencka, 680
 odwołanie do obiektu, 684
 serwer, 680
sterowanie Wordem, 685
- wiązanie, *Patrz*: wiązanie
- Word, 680
- Word.Application, 683
- zarządzanie Exceliem z poziomu innej aplikacji, 688
- autoodzyskiwanie, 107, 108
- B**
- BackColor, 909
- Backstage, 36
- BASIC, 147
- baza danych, 56, *Patrz też*: dane
 Access, 855
 ADO, 855
 arkuszowa, 57, 59
 dBASE, 103
 dostęp, 56
 format pliku, 103
 zewnętrzna, 57, 58, 59
- BeforeClose, 644, 649
- BeforeDoubleClick, 611, 651, 658, 660
- BeforeDragOver, 666
- BeforeDropOrPaste, 666
- BeforePrint, 644, 648
- BeforeRightClick, 651, 659, 660
- BeforeSave, 637, 644, 647
- bezpieczeństwo, 127, 136, 138, 698, 703, 942
- makra, 858
- biblioteka
 DLL, 326, 392
 Extensibility, 860
 funkcji, 323
 identyfikator globalny, *Patrz*: GUID
 Microsoft Excel 14.0 Object Library, 258
 Microsoft Forms 2.0 Object Library, 258
 Microsoft Office 14.0 Object Library, 258
 Microsoft Scripting Library, 828
 Microsoft Visual Basic for Applications
 Extensibility Library, 857, 860, 863
 Microsoft Word Object Library, 681
 MSForms, 661
 obiektów, 258, 681
 Excela, 822
 VBA, 822
 OLE Automation, 258
 Visual Basic for Applications, 258
 Windows Scripting Host, 833, 838
- BIFF, 102
- binarny format plików szablonów, 102
- blokada, 839
- błąd, 76, 121, 127, 134, 135, 136, 141, 171, 268, 282, 298, 319, 342, 737
 #####, 76
 #ADR!, 76, 923
 #ARG!, 76, 298, 319, 932
 #DZIEL/0!, 76
 #LICZBA!, 76
 #N/D!, 76
 #NAZWA, 76, 289, 931
 #ZERO!, 76
- Ambiguous name detected, 257
- daty, 219
- indeks poza zakresem, 935
- kod, 311, 965
- komunikat, 267, 283

On Error, 266
 out of memory, 588
 przechwytywanie, 266
 runtime, 200
 składni, 205, 266
 sprawdzanie w tle, 82
 subscript out of range, 935
 w formule, 75
 wartość, 311, 965
 wykonania, 266
 wykrywanie, 228, 319
 wyszukiwanie, 77
 book.xlsx, 110, 111
 BoundColumn, 481
 Break on All Errors, 266
 Break on Unhandled Errors, 266
 BUBBLESIZE_FROM_SERIES, 603
 Built-in Menus, 768
 Button, 133
 Button_Click, 328
 ByRef, 266
 ByVal, 265

C

Calculate, 611, 637, 651, 660
 calculated column, *Patrz*: kolumna obliczeniowa
 Calculation, 920
 Call, 256, 259, 264
 callback procedure, *Patrz*: procedura zwrotna
 Cell, 762, 772
 Cells, 194, 936
 Change, 446, 450, 512, 651, 652, 653, 654, 927
 Chapman Noyes, 538
 Chart, 150, 180, 533, 581, 688, 913
 publiczny, 615
 ChartObject, 180, 192, 581, 913
 ChartObjects, 180, 913
 rozmiar, 593
 usuwanie elementów, 589
 wyrównywanie, 593
 Charts, 180, 584, 589
 ChartTitle, 581
 ChDir, 828
 ChDrive, 828
 CheckBox, 131, 133, 426, 745
 CHM, 783
 Clear, 183
 ClearContents, 183, 197
 Click, 510, 666
 Code, 154, 157, 158
 kod źródłowy, 159, 160, 162, 163
 maksymalizacja, 158
 minimalizacja, 158

CodeModule, 860
 Cofnij, 940
 Collection, *Patrz*: kolekcja
 Color, 908
 ColorFormat, 189, 913
 ColumnCount, 470
 ColumnHeads, 470
 COM, 118, 699
 ComboBox, 133, 426, 790
 CommandBar, 150, 418, 419, 728, 729, 730, 761, 763, 860
 CommandButton, 129, 132, 133, 261, 427, 453, 460, 465, 870
 Comment, 185, 187, 188, 189, 190
 dodawanie, 190
 właściwości, 185, 187
 Component Object Model, *Patrz*: COM
 Const, 217
 Controls, 452
 ControlSource, 470, 944
 ControlTipText, 789
 CopyRange2, 333
 COUNTA, 349, 445
 COUNTIFS, 381
 CreateObject, 683, 684
 CreatePivotTable, 566, 571
 CSV, 103, 663, 840, 842, 843, 887
 CSVFileClass, 887, 888
 Currency, 197
 CurrentRegion, 335, 336
 Custom UI Editor for Microsoft Office, 560, 740, 742
 CustomUI, 757
 CVErr, 311, 932
 czas, 93, 94, 365, 367, 628, 667
 wersje językowe, 821, 825
 wprowadzanie, 94
 czeionki, 368, 901, 921
 o stałej szerokości, 946
 CZY.BŁĄD, 351
 CZY.LOGICZNA, 351
 CZY.TEKST, 351

D

dane
 analiza statystyczna, 59
 arkusz, *Patrz*: arkusz
 baza, *Patrz*: baza danych
 etykieta, 605
 filtrowanie, 57
 formularz, 420, 422
 hierarchiczne, 59
 kanał, 838

- dane
konspekt, *Patrz*: konspekt
mapowanie, 46
na wykresie
 zmiana, 599
seria, 601
 ukrywanie, 618
sortowanie, 57
sprawdzanie poprawności, 445, 654
szybka analiza, *Patrz*: Quick Analysis
tabela, 420, *Patrz*: tabela
typ, 206, 208, 209, 211, 351, *Patrz też*: wartość definiowana przez użytkownika, 225
deklaracja, 215
konwersja, 211
 wbudowany, 208
wprowadzanie, 24, 48, 403, 536, 537, 538
wyszukiwanie, 46
zapisywanie
 do pliku tekstowego, 839
DANE, 58, 96, 728, 736
DAO, 855
data, 93, 218, 244, 317, 365
 format, 219, 825
 MacOS, 817
 separatator, 824
 sprzed roku 1900, 94
 wersje językowe, 821, 825
wprowadzanie, 94
wydrukowania pliku, 379
zapisania pliku, 379
DATA, 219
Data Access Objects, *Patrz*: DAO
DataPivotField.Orientation, 571
Date, 244, 365
DateSerial, 219, 244, 825
dBASE, 103
DBF, 103
DblClick, 666
DDE, 128
de Bruin Ron, 818
Deactivate, 447, 611, 644, 647, 651, 660, 666
Debug.Print, 319
debugging, *Patrz*: VBE błąd
DECIMAL2RGB, 897
Declare, 326
Default to Full Module View, 169
Definiowanie nazw, 73
Definiuj nazwę, 69, 71, 73
deklaracja, 159
Delete, 589
DeleteHiddenNames(), 70
deseń wypełnienia, 904
Designer, 860, 870
DEWELOPER, 132, 152, 172, 253, 259, 728, 736, 918
Dialogs, 418, 946
DIF, 103
Dim, 213, 214, 215, 224, 225, 264, 890
Dir, 828, 830
DisplayAlerts, 930
DisplayGridlines, 221
DisplayVideoInfo, 396
DL, 86
Do Until, 245
Do While, 244
Docking, 171
dodatek, 60, 76, 102, 118, 121, 127, 180, 254, 543, 547, 662, 697, 802, 948
Analysis ToolPak, *Patrz*: Analysis ToolPak
COM, 118, 699
dostęp do procedur, 711
dystrybucja, 707
formularz danych, 537
instalacja, 548, 948
instalowanie, 705
jako skoroszyt, 719
kompatybilność, 723
menedżer, 700, 706
modyfikacja, 707, 949
nazwa, 716
nazwa pliku, 716
ochrona, *Patrz*: ochrona dodatku
odwołania do innych plików, 723
opis, 704
otwieranie, 929
problemy, 721, 723
przechowywanie funkcji niestandardowych, 325
przekształcenie w skoroszyt, 562
przetwarzanie VBA, 714
skoroszyt zawierający funkcje, 296
Solver, *Patrz*: Solver
SP, *Patrz*: SP
SR, *Patrz*: SR
ścieżka, 716
testowanie, 707
tworzenie, 702, 703, 704, 949
 lista kontrolna, 708
wiadomość powitalna, 803
właściwości, 716, 717
wydajność, 720
XLAM, 118, 928
XLL, 118
zabezpieczony, 712
zastosowanie, 698
zdarzenie, 719
dodatki, 60

- DODATKI, 728, 736
DoEvents, 625
dokument
 motyw, 901, 911, 921
 deseń wypełnienia, 904
 kolor, 902
 kolory, 905
 ochrona, *Patrz*: ochrona dokumentu
 ukrywanie, 137
dokumentacja, 139, 140
Drag-and-Drop Text Editing, 169
DrawMenuBar, 519
Drives, 835
DropDown, 133
drukarka domyślna, 394
drukowanie
 nagłówek, 648
 podgląd, 921
 stopka, 648
 ukrywanie kolumn, 649
Dynamic Data Exchange, *Patrz*: DDE
Dynamic Link Library, 392, *Patrz*: biblioteka DLL
DynamicMenu, 753
DZIŚ, 299
dźwięk, 922
- E**
- early binding, *Patrz*: wiązanie wcześnie
Editor, 166
Editor Format, 169
edytor VBE, *Patrz*: VBE
efekt falowy, 75
ekran
 aktualizacja, 936
 odświeżanie, 376
 oświeżanie, 282
 rozdzielcość, 947
 zrzut, 596
Else, 233
e-mail, 690
 załącznik, 693
EnableCalculation, 920
EnableCancelKey, 283
Enabled, 809
End, 214
End Function, 150
End Sub, 149, 250, 264
End With, 177
Err, 267
Error, 267, 448, 666
- etykieta, 133, 427
animacja, 490
 system pomocy, 788
event handler procedure, *Patrz*: procedura obsługi
 zdarzenia
Excel
 błąd, 135
 historia, 43
 kompatybilność, 61, 105, 368, 419, 568, 640,
 723, 730, 765, 813, 814, 815, 816
 czas, 821
 data, 821
 Macintosh, 813, 817
 konfiguracja, 119
 konieczność, 762
 MacOS, 817
 obszar roboczy, *Patrz*: obszar roboczy
 plik, *Patrz*: plik
 rejestrowanie wykorzystania, 845
 tryb projektowania, 429
 uruchamianie, 99
 wersja, 127, 141
 32-bitowa, 815, 818, 932
 64-bitowa, 326, 815, 818, 932
 numer, 815
 wersje językowe, 141, 809, 815, 819, 821, 823
 identyfikator języka, 821
 kod języka, 820
 zarządzanie z poziomu innej aplikacji, 688
Excel 2000, 813
Excel 2002, 813
Excel 2003, 22, 60, 730, 813
 menu, 768
Excel 2007, 22, 23, 33, 58, 88, 101, 326, 727, 762,
 813
 Centrum zaufania, 858
Excel 2008, 813
Excel 2010, 818
Excel 2013, 21, 22, 33, 44, 60, 767, 813, 818, 951
 nowości, 36
 rejestr systemu, 120, 121
Excel MacOS, 813, 817
Excel Web App, 949
Excel XP, 60
Excel.officeUI, 117
ExecuteMso, 418, 733, 946
Exit For, 231, 243
Exit Sub, 270
Extensibility, 860
eXtensible Markup Language, *Patrz*: XML
extension hardening, *Patrz*: utwardzanie rozszerzeń
EXTRACTELEMENT, 384

F

FaceID, 776
faktura generowanie numeru, 939
False, 364
FAQ, 917
File Block Policy, *Patrz*: reguła blokowania plików
FileCopy, 828
FileDateTime, 828
FileDialog, 417
FileExists, 372, 829, 834
FileLen, 828, 830, 960
FileNameOnly, 372
FileSearch, 828
FileSystemObject, 828, 833, 834, 835, 838
 Drive, 833
 Drives, 835
 File, 833
 Folder, 833
 TextStream, 833
Fill, 909
Fill Flash, 36
FillFormat, 189, 909, 913
filtrowanie, 90
 osi czasu, *Patrz*: Timeline Slicer
FindControl, 765
FindWindowA, 519
Flash Fill, 50, 96
fmMultiSelectExtended, 475
fmMultiSelectMulti, 475
fmMultiSelectSingle, 475
folder
 _rels, 113, 743
 charts, 115
 chartsheets, 115
 customXml, 113
 diagrams, 115
 docProps, 115
 drawings, 115
 media, 115
 tables, 115
 theme, 115
 worksheets, 115
 xl, 115
FollowHyperlink, 651
For ... Next, 240
For Each, 227, 229
For Each ... Next, 270, 274
ForeColor, 909, 911, 913
formant, 34, 40, 56, 131, 429, 733
 ActiveX, 258, 259, 429, 442, 455
 MacOS, 817
 testowanie, 456
CheckBox, 426, 745

ComboBox, 426, 790
CommandButton, 427, 460, 465,
 Patrz: CommandButton
Controls, 452
dane XML, 753
DynamicMenu, 753
etykieta, 427, 733
formularza, 132, 259, 429
Frame, 427
identyfikacja, 454
identyfikator, 765
Image, 427, 518, 608, 744
klawisz skrótu, 435
 kolejność, 434
kontener, 427
Label, 427, 490, 525
ListBox, *Patrz*: ListBox
menu podręczne, 764, 765
MHTML, *Patrz*: MHTML
modyfikowanie, 430, 431, 432
MultiPage, 427, 487, 504, 505, 506, 507, 512,
 801
 kreator, 509
o zmiennym położeniu, 518
obraz, 427, 733
OptionButton, 428, 433
pasek przewijania, 428
podmenu, 765
pokrętło, 428
pole
 tekstowe, 428
 listy, 427
przycisk, 765
 opcji, 428
 przelącznika, 429
RefEdit, 428, 462
 błędy, 462
ScrollBar, 428, 468, 469, 531
SpinButton, 428, 449
 zdarzenie, 447, 448
superetykieta, 734
szablon, 455
TabStrip, 428, 487
testowanie, 429
TextBox, 428, 449, 490
ToggleButton, 429
Toolbox, 425
tworzenie, 748
umieszczanie na formularzu UserForm, 868
umieszczany w arkuszu, 129
ViewCustomViews, 731
właściwość, 432, 433
wstawianie do arkusza, 132
wyboru koloru, 902
zewnętrzny, 488

- format
 CSV, *Patrz*: CSV
 GIF, *Patrz*: GIF
 HTML, *Patrz*: HTML, *Patrz*: HTML
 MHTML, *Patrz*: MHTML
 MP3, *Patrz*: MP3
 pliku, 100, 103, 115
 ZIP, *Patrz*: ZIP
formatowanie, 51
 komórek arkusza, 52
 styl, 901
 wartości numerycznych, 51
 warunkowe, 51, 52, 296, 346, 658, 927
 wyglądu arkusza, 51
Formatowanie, 38
Formula, 183, 197, 198, 602
FormulaArray, 197
FormulaLocal, 197
FormulaR1C1, 197, 198
formularz
 bez paska tytułowego, 519
 danych, 420, 422, 536, 537, 538
 etykieta, 490
 inicjalizacja, 804
 ładowanie, 437
 niemodalny, 100, 499, 521, 523
 położenie na ekranie, 436
 półprzezroczysty, 534
 procedura obsługi zdarzeń, 444
 przewijanie, 468
 przycisk, 528
 puzzle, 538
 rozmiar, 524
 sprawdzanie poprawności danych, 445
 testowanie, 443
 tworzenie, 439
 ukrywanie, 438
UserForm, *Patrz*: UserForm
wideo Poker, 540
wielkość, 466, 468
wybór koloru, 531
wykres, 532, 608
zamykanie, 438
zdarzenie, 445, 447, 638, 665
formula, 34, 48, 63, 64
 błędy, 75
 nazwa, 73, 74
obliczanie
 automatyczne, 64
 ręczne, 64
opcje obliczania, 64
pośrednia, 96
sumująca, 89
szacowanie, 84
tablicowa, 63, 85, 311, 388
kalendarz, 86
wady, 87
zalety, 87
ukrywanie, 53, 136
wyszukiwania i adresu, 91, 92
zliczająca, 88
FORMUŁY, 42, 48, 64, 69, 71, 72, 76, 78, 82, 728, 735
fragmentator, 106, *Patrz*: Slicer
Frame, 427, 463
FreeFile, 840
Frequently Asked Questions, *Patrz*: FAQ
Function, 158, 293, 321
funkcja, 24, 48, 158, 220, 250, 254, 287, 931, 955, 958
 API, 327, 329, 392, 815
 GetKeyState, 328
argumenty, 49, 168, 293, 298, 304, 932
nieokreślona liczba, 312
opcjonalne, 306, 307, 308
opis, 321
tablica, 305
arkuszowa, 36, 226, 227, 931
Array, 308
BD.ILE.REKORDÓW, 90
BD.SUMA, 90
bezargumentowa, 299
biblioteki DLL, 392
biznesowa, 148
błędy, *Patrz*: błąd
BRAK, 76
BUBBLESIZE_FROM_SERIES, 603
CDec, 208
celu, 60
COUNTA, 349, 445
COUNTIFS, 381
CreateObject, 683, 684
CVErr, 311
CZY.BŁĄD, 96, 351
CZY.LOGICZNA, 351
CZY.TEKST, 351
data, 317
DATA, 219
Date, 244, 365
DateSerial, 219, 244, 825
Dir, 830
DŁ, 86
DrawMenuBar, 519
Error, 267
EXTRACTELEMENT, 384
FileDialog, 417
FileExists, 372, 829
FileLen, 830, 960

funkcja
 FileNameOnly, 372
 finansowa, 148
 FindWindowA, 519
 FreeFile, 840
 GetExecutable, 393
 GetExitCodeProcess, 675
 GetKeyboardState, 882
 GetObject, 683, 684
 GetRegistry, 397
 GetSetting, 399, 532, 555, 807
 GetSystemDirectory, 723
 GetWindowLong, 519
 IIf, 236
 ILE.NIEPUSTYCH, 445
 InputBox, 341, 404, 413, 943
 inspekcji, 48
 internetowa, 58
 inżynierska, 148
 IsDate, 351
 IsEmpty, 351
 ISLIKE, 383
 IsMissing, 308, 310
 IsNumeric, 351
 IsText, 312
 jednoargumentowa, 302
 JEŻELI, 96
 kategoria, 321, 323, 324, 931
 tworzenie, 932
 LASTPRINTED, 379
 LASTSAVED, 379
 Left, 366
 LEWY, 96, 97
 LICZ.JEŻELI, 88, 89
 LICZ.WARUNKI, 381
 lista, 226
 LITERY.WIELKIE, 546
 LoadPicture, 533
 LOS, 301
 matematyczna, 148
 MAXALLSHEETS, 387
 Month, 244
 MsgBox, 204, 222, 228, 276, 319, 409, 943
 emulacja, 514, 515, 516, 517
 nazwa, 293, 294, 307
 niestandardowa, 288, 291, 292, 293, 296, 321,
 325, 372, 376
 deklaracja, 293, 294
 w arkuszu, 289, 292
 w procedurze, 290, 292
 opis, 321, 324, 931
 osłonowa, 300, 327
 PathExists, 373, 829
 PIERWIASTEK, 76
 PRAWY, 96
 przeliczanie, 301
 przykłady, 372, 376
 przypisanie tematów pomocy, 797
 RANDOMINTEGERS, 388
 RangeNameExists, 373
 RGB, 894
 SaveSetting, 399, 532, 555, 808
 SAYIT, 378
 Seek, 840
 SERIE, 599, 601, 602, 626
 argumenty, 601
 SERIESNAME_FROM_SERIES, 603
 SetWindowLong, 519
 SheetExists, 374
 SheetOffset, 386
 Shell, 673, 674
 ShellExecute, 676, 677
 SPELLDOLLARS, 385
 Split, 373
 Sqr, 226
 SUMA, 86, 89, 90, 313
 SUMA.JEŻELI, 88, 90
 testowanie, 296, 297, 319
 Timer, 353
 TRANSPONUJ, 309, 356
 TRANSPOSE, 356
 TypeName, 211
 UCase, 226
 USUN.ZBĘDNE.ODSTĘPY, 96, 546
 użytkownika, 933
 VALUES_FROM_SERIES, 603, 604
 wbudowana, 220, 226, 287, 306, 933
 WeekDay, 237
 wieloargumentowa, 305
 wielofunkcyjna, 386
 WorkbookIsOpen, 375
 WriteRegistry, 398
 wstawianie, 48
 WYSZUKAJ, 76
 wyszukiwanie i adresu, 91, 92
 wywoływanie, 294, 295, 296, 297, 931, 958
 XDATE, 317
 XDATEADD, 317
 XDATEDAY, 317
 XDATEDIF, 317
 XDATEMONTH, 317
 XDATEYEAR, 317
 XDATEYEARIF, 317
 XVALUE_FROM_SERIES, 604
 XVALUES_FROM_SERIES, 603
 zasieg, 294
 ZDATEDOW, 317
 ZŁĄCZ.TEKSTY, 546

ZNAJDŹ, 96
 zwracanie tablic VBA, 308
 zwracanie wartości błędu, 311

G

galeria stylów, 901
 generator mowy, 378
GetAttr, 828
GetEnabledMso, 733
GetExecutable, 393
GetExitCodeProcess, 675
GetImageMso, 733, 735
GetKeyboardState, 882
GetKeyState, 328
GetLabelMso, 733
GetObject, 683, 684
GetOpenFilename, 413, 946
GetPressedMso, 733
GetRegistry, 397
GetSaveAsFilename, 416
GetScreentipMso, 733
GetSetting, 399, 532, 555, 807
GetSupertipMso, 734
GetSystemDirectory, 723
GetValue2, 338
GetWindowLong, 519
GIF, 533, 597, 930
Globally Unique Identifier, *Patrz:* GUID
Goto, 935
GoTo, 232, 240
GroupBox, 133
 grupa
 pole, 133
GUID, 861

H

harmonogram amortyzacji, 800
hasło, 53, 54, 127, 137, 138, 184, 698, 703, 710,
 942, 943
Height, 507, 527
Help, 795
HelpFile, 797
hidden name, *Patrz:* nazwa ukryta
Hide, 438, 439
 hiperłącze, 58, 362, 651, 662
HKEY_CLASSES_ROOT, 120
HKEY_CURRENT_CONFIG, 120
HKEY_CURRENT_USER, 120
HKEY_LOCAL_MACHINE, 120
HKEY_USERS, 120
HLP, 793

HTML, 104
HTML, 58, 104, 597, 791, 827, 846
HTML Help, 793, 794
HTML Help Viewer, 794

I

IDE, 857, 858
 hierarchia, 860
identyfikator, 924
If, 230, 236
If ... Then, 232
If ... Then ... Else, 234, 236
ILE.NIEPUSTYCH, 445
Image, 133, 427, 608, 744
imageMso, 744
Immediate, 155, 158, 202, 252, 262, 274, 297
Immediate Window, 153
implikacja logiczna, 221
INI, 119
InitialFilename, 417
Initialize, 446, 447, 452, 460, 638, 666, 887
Input, 841
InputBox, 341, 404, 406, 413, 941, 943
Inquire, 36
Insert/Class Module, 881
Inspekcja formuł, 48, 76, 78, 82, 84
instrukcja, 955
 AppActivate, 677
 Debug.Print, 319
 Dim, 890
 For...Next, 204
 Input, 841
 InputBox, 941
 Line Input, 841
 On Error, 269
 On Error GoTo, 342
 On Error Resume Next, 342
 Open, 838, 839
 Option Explicit, 924, 942
 Print, 841
 przypisania, 204, 220
 ReDim, 940
 ReDim Preserve, 940
 Set, 890, 940
 Write, 841
Integrated Development Environment, *Patrz:* IDE
interfejs
 automatyzacji, 857
 użytkownika, 23, 35, 37, 128, 950
 kontekstowy, 23
 modyfikacja, 803
 projektowanie, 138
 wersja zminimalizowana, 748

International, 823
 Internet, 58
 intersection operator, *Patrz*: operator przecięcia
 intranet, 58
 IRibbonControl, 742
 IsAddin, 698, 712
 IsDate, 351
 IsEmpty, 351
 ISLIKE, 383
 IsMissing, 308, 310
 IsNumeric, 351
 IsText, 312

J

język
 BASIC, *Patrz*: BASIC
 interpretowany, 147
 komplikowany, 148
 skryptowy, 148
 strukturalny, 241
 ściśle deklarowany, 208
 VBA, *Patrz*: VBA
 XLM, *Patrz*: XLM
 JPEG, 597

K

kalendarz, 86, 317
 kalkulator, 673
 karta, 23, 37
 aktywacja, 735
 DANE, *Patrz*: DANE
 DEWELOPER, *Patrz*: DEWEOPER
 Docking, 171
 DODATKI, *Patrz*: DODATKI
 dodawanie, 453
 FORMUŁY, *Patrz*: FORMUŁY
 graficzna, 142
 rozdzielcość, 142, 395, 411
 grupa, 748
 kolor, 32
 kontekstowa, 38
 Formatowanie, 38
 Projektowanie, 38
 NARZĘDZIA GŁÓWNE, *Patrz*: NARZĘDZIA GŁÓWNE
 PLIK, *Patrz*: PLIK
 RECENZJA, *Patrz*: RECENZJA
 tworzenie, 748
 UKŁAD STRONY, *Patrz*: UKŁAD STRONY
 WIDOK, *Patrz*: WIDOK
 WSTAWIANIE, *Patrz*: WSTAWIANIE

katalog
 domowy, 327
 lista plików, 829, 831
 nazwa, 417
 startowy, 417
 XLStart, 923
 KeyDown, 666
 KeyPress, 666
 KeyUp, 666
 Kill, 828
 klasa, 879
 CSVFileClass, 887, 888
 IRibbonControl, 742
 moduł, 614
 obiektów, 150
 tworzenie, 879
 Reference, 861
 klawiatura, 23, 434
 zdarzenie, 448
 klawisz, 328, 668
 kod, 328, 670
 NumLock, 880
 Shift, 328
 skrótu, *Patrz*: skrót klawiaturowy
 kod
 RibbonX, 548, 560, 730, 736, 737, 803, 951
 błędy, 737
 menu podręczne, 769
 modyfikacja, 740
 tworzenie, 743
 spaghetti, 240, 241
 kolekcja, 150, 180, 181, 199, 227, 374, 934
 AddIns, 698, 706, 709, 714, 715
 ChartObjects, 913
 usuwanie elementów, 589
 Charts, 584
 usuwanie elementów, 589
 CommandBars, 763
 Comments, 187
 Controls, 452
 Dialogs, 418
 PivotFields, 567
 przetwarzanie, 229
 References, 860, 861
 Shapes, 582
 SparklineGroups, 633
 VBComponents, 860, 861
 VBProjects, 860
 Workbooks, 181, 709
 Worksheets, 181
 kolor, 189, 531, 901
 definiowanie, 893, 905
 gradient, 914
 HSL, *Patrz*: HSL

- konwersja, 895
ksztaltów, 908, 909, 910
motyw dokumentu, 902
nazwa, 902
paleta, 893
RGB, *Patrz*: RGB
skala szarości, 897
VBA, 895
wartość dziesiętna, 893, 894, 895, 898
wykres, 912
 losowy, 915
 zamiana na skalę szarości, 899
kolumna
 nazwa, 69, 71
 obliczeniowa, 68
 ostatnia komórka, 939
 szerokość, 34
 ukrywanie, 34, 137
 przed wydrukiem, 649
komentarz, 185, 187, 190, 204, 205, 206, 924
 do zawartości komórek, 784
 wyświetlanie obrazów, 785
 zmiana, 292
komórka
 aktywna, 191
 modyfikacja wykresu, 599
 blokowanie, 127, 136
 dostęp gdy arkusz nie jest otwarty, 929
 formatowanie, 296, 377, 927
 komentarz, 190, 784
 modyfikacja, 652, 653
 nazwa, 69
 nazwana, 75
 odczytywanie na głos, 921
 ostatnia niepusta, 381
 pobieranie informacji, 197
poprzednik, 79
 bezpośredni, 79
 identyfikowanie, 80
 pośredni, 79
scalona, 195
tło, 658
typ danych, 351
ukrywanie, 361
wartość maksymalna, 387
wprowadzanie wartości, 338, 339
z wykresem, 633
zakres, 342
 eksportowanie do pliku HTML, 846
 eksportowanie do pliku XML, 849
 nazwa, 69
 nieciągły, 358
 przetwarzanie, 346
zależna, 79
 identyfikowanie, 81
zawierająca formuły, 653
zaznaczanie na podstawie wartości, 357
zliczanie, 342, 381
kompatybilność, *Patrz*: Excel kompatybilność
komunikat, 228
 błędu, 267
 ikony, 409
 na pasku stanu, 937
 powitalny, 645
 przycisk, 409
 wyświetlanie, 409, 930
koniunkcja logiczna, 221
konspekt, 59
kreator, 508, 509
 importu tekstu, 840
 konwersji tekstu na kolumny, 840, 842

L

- Label, 133, 427, 490, 525
 system pomocy, 788
LastInRow, 382
LASTPRINTED, 379
LASTSAVED, 379
late binding, *Patrz*: wiązanie późne
Layout, 666
Left, 366
LEWY, 96, 97
LICZ.WARUNKI, 381
liczba seryjna, 93
licznik, 499
Line Input, 841
LinkedCell, 429
lista, *Patrz też*: ListBox
 danych, 56, 420
 pole, 133
 rozwijana, 39, 57, 426, 631, 920
 separatator, 823
Lista pól tabeli przestawnej, 572

ListBox, 129, 133, 427, 460, 461, 469, 946
 arkusz uaktywnianie, 484
 element, 475
 lista elementów, 470, 471, 473, 475, 476, 477,
 479, 482
 lista wielokolumnowa, 480
 uaktywnianie arkusza, 484
 zawartość zmienna, 476
 ListIndex, 461
 literał, 264, 298
 tablicowy, 824
 LITERY.WIELKIE, 546
 Load, 437
 LoadPicture, 533
 LOS, 299, 301, 302

Ł

łańcuch tekstowy, 218, 275, 462
 element, 384
 wzorzec, 383
 łącza, 922, 944

M

Macintosh, 817
 MacroOptions, 321, 323, 798
 makro, 31, 50, 60, 101, 106, 133, 159, 162, 175, 926
 arkusz osobisty, 178, 845, 923
 bezpieczeństwo, 858
 cofanie skutków działania, 940
 dodawanie
 do paska narzędzi Szybki dostęp, 951
 do Wstążki, 951
 interaktywne, 341
 klawisz skrótu, 129, 130, 177
 kompatybilność, 568
 miejsce przechowywania, 177
 modyfikacja, 178
 nazwa, 133, 177
 opis, 178
 osobiste, 178
 przerwanie wykonywania, 466
 rejestrator, 163, 172, 173, 177, 200, 217, 252,
 273, 332, 923, 936
 wykresy, 580
 rejestrowanie, 163, 273, 918
 podczas zmiany koloru, 904
 skrót klawiszowy, 254
 uruchamianie, 918
 okresowe, 930
 wskaźnik postępu zadania, 499
 wykonywanie
 automatycznie, 929
 wyświetlanie, 930
 XLM, 34, 149
 arkusz, *Patrz:* arkusz makr XLM
 rejestrowanie, 60, 148
 Maksymalizuj, 158
 Malware, *Patrz:* oprogramowanie złośliwe
 mapa, 56
 MAXALLSHEETS, 387
 MDB, 103
 MDE, 103
 Me, 438
 megaformuła, 48, 97
 tworzenie, 95
 Menedżer nazw, 69, 72
 menu, 459, 729, 764, 765
 Backstage, 23
 Excel 2003, 768
 kontekstowe, 44
 podręczne, 37, 255, 645, 729
 Built-in Menus, 768
 Cell, 762, 764, 770, 772
 dodawanie elementów, 772
 dosłosowywanie, 730
 dziedziczenie, 769
 elementy, 766, 772, 774, 778
 Excel 2013, 769
 FaceID, 776
 formant, 764, 765
 właściwość, 765
 wyszukiwanie, 765
 ikona polecenia, 776
 kod RibbonX, 769
 kontekstowe, 778
 modyfikacja, 44, 767, 777
 modyfikacja automatyczna, 777
 nazwa, 763
 niestandardowe, 761, 951
 podmenu, 774, 776
 resetowanie, 770
 tworzenie, 778
 tworzenie automatyczne, 777
 ukrywanie elementów, 778
 usuwanie automatyczne, 777
 usuwanie elementów, 774
 VBA, 767
 wyłączanie, 671, 771
 wyłączanie elementów, 772, 778
 wyświetlanie, 762
 zdarzenie, 777
 Menu bar, *Patrz:* pasek menu
 menu podręczne, 44
 Cell, 763
 CommandBar, 762
 dosłosowywanie, 129

- metoda, 152, 182, 934
Activate, 935
Add, 759
AddChart, 580, 582
AddComment, 191
AddFromFile, 861
AddFromGuid, 861
AddItem, 472
argumenty, 168, 184
 nazwane, 184
Cells, 936
Clear, 183
ClearContents, 152, 183
ExecuteMso, 418, 733, 946
FindControl, 765
GetEnabledMso, 733
GetImageMso, 733, 735
GetLabelMso, 733
GetOpenFilename, 413, 946
GetPressedMso, 733
GetSaveAsFilename, 416
GetScreentipMso, 733
GetSupertipMso, 734
Goto, 935
Help, 795
Hide, 438
InputBox, 406, 941, 943
MacroOptions, *Patrz:* MacroOptions
nazwa, 24
obiektywia, 183
obiektu Comment, 185, 187
OnTime, 930
OnUndo, 940
Open, 838
Print, 277
Protect, 184
SearchHelp, 796
Select, 935, 937
Selection.Areas.Count, 938
Selection.Rows.Count, 938
SendKeys, 735, 951
Show, 436
ShowDataForm, 422
Workbooks, 709
MHT, 104
MHTML, 104, 791, 792
Microsoft HTML Help Workshop, 783
Microsoft Office, 678
Microsoft Office Code Compatibility Inspector, 816
Microsoft Office Compatibility Pack, *Patrz:* Pakiet zgodności formatu plików pakietu Microsoft Office
Microsoft Scripting Library, 828
Microsoft SkyView, *Patrz:* SkyView
Microsoft Visual Basic for Application Extensibility, 858
Microsoft Visual Studio Tools for Office, *Patrz:* VSTO
MIDI, 922
MIME HyperText Markup Language, *Patrz:* MHTML
minipasek narzędzi, 44
MkDir, 828
Mod, 220
modal dialog box, *Patrz:* okno dialogowe modalne
model
 ADO, 855
 HSL, *Patrz:* HSL
 obiektowy, 32, 148, 150, 180, 857, 860
 RGB, *Patrz:* RGB
modeless dialog box, *Patrz:* okno dialogowe niemodalne
moduł
 klasy, 614, 639, 660, 662, 879
 dodawanie kodu VBA, 881
 tworzenie, 880
 wstawianie, 881
 zdarzenie, 887
 klasy, 884
 usuwanie
 za pomocą kodu, 924
 VBA
 aktualnianie, 864
modyfikacja zarejestrowanych makr, 178
monitor
 podwójny, 947
Month, 244
MouseDown, 611, 660, 666
MouseMove, 611, 660, 661, 666
MouseOver, 621, 622
MouseUp, 611, 660, 666
MP3, 490
MSForms, 661
MsgBox, 204, 222, 228, 276, 319, 409, 943
 emulacja, 514, 515, 516, 517
 koniec wiersza, 933
 stała, 409, 410
MultiPage, 427, 463, 487, 505, 508, 801
 kreator, 509
 wskaźnik postępu zadania, 504, 505, 506, 507
Multiplan, 103
MultiSelect, 470, 475
mysz, 24
 zdarzenie, 448

N

nagłówek, 648
 Name, 180, 588, 828, 937
 napęd dyskowy, 394, 835
 Narzędzia do rysowania, 38
NARZĘDZIA GŁÓWNE, 37, 39, 40, 52, 77, 728, 735
 Narzędzia równań, 38
 narzędzie, 543, 545
 operacje tekstowe, 545, 547
 Operacje tekstowe, *Patrz*: Operacje tekstowe wymagania, 547
 nazwa, 49, 69, 180
 formuły, 73, 74
 kolumny, 71
 komórki, 75
 obiektu, 75
 stałej, 73
 ukryta, 70
 wiersza, 71
 wycofywanie, 71
 zakresu, 75
 zasięg, 72
 Nazwy zdefiniowane, 72
 negacja logiczna, 221
 NewSheet, 637, 644, 647
 NewWorkbook, 638, 662
 nierównoważność logiczna, 221
 niestandardowe menu podręczne, 130
 Not, 364, 365
 notacja
 A1, *Patrz*: A1
 W1K1, *Patrz*: W1K1
 notacja W1K1, 66
 Nowa nazwa, 73

O

OASIS, 104
 obiekt, 31, 150, 180, 199, 224, 227, 262, 374, 934
 AddIn
 właściwości, 716, 717
 AddIns, 180
 aktywny, 150
 Application, 180, 191
 tworzenie, 688
 właściwości, 192
 zdarzenie, 662
 Chart, 581, 639, 913
 publiczny, 615
 tworzenie, 688
 ChartObject, 192, 581, 588
 ChartObjects, 180

Charts, 180
 ChartTitle, 581
 ClipArt, *Patrz*: ClipArt
 ColorFormat, 189, 910
 CommandBar, 418, 419, 728, 729, 761, 762
 Comment, 185, 187, 188, 189
 dodawanie, 190
 metody, 185, 187
 właściwości, 185, 187
 danych ActiveX, 855
 DataPivotField, 571
 Debug, 277
 Designer, 870
 Err, 267
 FileDialog, 417
 FileSearch, 828
 FileSystemObject, 828, 833, 834, 835, 838
 FillFormat, 189, 909
 graficzny, 597
 hierarchia, 32, 150, 180, 380, 934
 klasa, *Patrz*: klasa
 kolor, 902
 Label, 788
 metoda, 152, 183, 199, 934
 programowanie, 886
 model obiektowy, 32
 nadzędny, 380
 Name, 937
 Names, 180
 nazwa, 24, 75
 odwołanie, 150
 OLE, *Patrz*: OLE
 PageSetup, 180
 PivotTables, 180
 Range, 180, 182, 183, 193, 194, 196, 197
 właściwości, 193
 Reference, 861
 Selection, 178
 Shape, 189, 192, 622
 kolor, 908
 kolor tła, 909
 Sheet, 639
 Sparkline, 633
 SparklineGroup, 633
 Speech, 922
 TextStream, 838
 ThisWorkbook, 639
 tworzenie, 688
 UserForm, 639
 VBComponent, 864
 VBE, 860
 VBIDE, 857
 VBProject, 858, 860, 861
 Windows, 180

-
- właściwość, 150, 182, 199, 934
 - domyślna, 183
 - programowanie, 884
 - Workbook, 181, 581
 - tworzenie, 688
 - Workbooks, 180
 - Worksheet, 181, 193, 194
 - WorksheetFunction, 227, 958
 - Worksheets, 180
 - zaznaczony, 50, 192
 - Object Browser, *Patrz*: przeglądarka obiektów
 - Object Linking and Embedding, *Patrz*: OLE
 - ObjectThemeColor, 911
 - obraz, 133, 427
 - FaceID, 776
 - imageMso, 744
 - obsługa błędów, *Patrz*: błąd
 - obszar roboczy, 104
 - ochrona, 52, 53, 54, 137, 184
 - dodatku, 137
 - dokumentu, 137
 - formuł, 52
 - hasło, 53
 - kodu VBA, 54
 - skoroszytu, 283
 - struktury skoroszytu, 53
 - ODS, 104
 - odwołanie, 65, 200, 940
 - bezwzględne, 65, 67, 173
 - do arkusza, 67
 - do biblioteki obiektów, 681
 - do formantu, 451
 - do kolumny, 65
 - do komórek, 304, 307
 - bezwzględne, 65, 68
 - nazwy, 71
 - notacja W1K1, 66
 - scalonych, 195
 - względne, 65
 - do modelu obiektowego
 - Windows Script Host, 834
 - do obiektu, 181, 684
 - do projektu VBA, 861, 862
 - do skoroszytu, 67, 257, 929
 - do tabeli, 68
 - do wiersza, 65
 - do zakresu, 194, 337
 - formuł łączy, 68
 - mieszane, 65
 - nadawanie nazw, 71
 - strukturalne, 68
 - wyczywanie nazw, 71
 - względne, 65, 66, 67, 173, 175, 335
 - odwołanie do modelu obiektowego
 - Windows Script Host, 834
 - OfficeUI, 116
 - Offset, 198
 - okno, 180
 - aktywne, 192
 - dialogowe, 37, 45, 138, 228, 403, 668
 - data i godzina, 678
 - efekt podświetlenia, 535
 - emulacja, 456
 - karta, 45, 427
 - modalne, 45, 495
 - niemodalne, 45, 495
 - niestandardowe, 130, 131, 423
 - Panelu sterowania, 678
 - testowanie, 443
 - tworzenie, 46
 - uaktualnienie, 496
 - wbudowane, 946
 - wielkość, 466
 - wyswietlanie, 417
 - zmiana rozmiaru, 524
 - Immediate, 262, 274
 - linie siatki, 221
 - Makro, 253
 - niemodalne
 - wyswietlanie, 436, 437
 - o stałym położeniu, 53
 - o stałym rozmiarze, 53
 - powitalne, 463
 - Properties, 432
 - Przechodzenie do — specjalnie, 77, 80, 81
 - przezroczystość, 534
 - skalowanie, 142
 - Toolbox, 453, *Patrz*: Toolbox
 - dodawanie formantów, 453
 - wprowadzania danych, 403
 - Wstawianie funkcji, 320
 - OLE, 128, 680, 857
 - On Error, 266, 269
 - On Error GoTo, 342
 - On Error Resume Next, 282, 342
 - OnKey, 638, 666, 668, 669
 - kody klawiszy, 670
 - OnTime, 638, 666, 667, 930
 - OnUndo, 940
 - Open, 637, 644, 645, 838, 839
 - OpenOffice, 104
 - Operacje tekstowe, 546, 547
 - ApplyButton_Click, 553
 - CloseButton_Click, 554
 - Cofnij, 557
 - ComboBoxOperation_Change, 553
 - CreateWorkRange, 554
 - HelpButton_Click, 554
 - Module1, 550

- Operacje tekstowe
pomoc, 559
PROGRESSTHRESHOLD, 551
ShowTextToolsDialog, 551
umieszczanie polecen na Wstążce, 560
UndoTextTools, 552
UserChoices, 551, 556
UserForm, 549, 552
UserForm_Initialize, 552
ustawienia, 555
wydajność, 554
OperatingSystem, 817
operator, 205, 220
", 928
And, 222
Eqv, 928
Imp, 928
Is, 928
kolejność, 221
konkatenacji, 925
kontynuacji, 925
kropki, 181
Like, 383, 928
logiczny, 221, 928
matematyczny, 928
Mod, 220
negacji, 221
nierówności, 220
Not, 221, 364, 365
porównania, 220
potęgowania, 221
przecięcia, 71, 72
przecięcia zakresów, 194
przypisania, 206, 220
równości, 220
Xor, 928
oprogramowanie złośliwe, 106
Option Base, 311
Option Explicit, 924, 942
Option Private Module, 250
OptionButton, 131, 133, 428, 432, 433, 439
optymalizacja, 60
Outlook, 690, 693
- P**
- PageSetup, 165, 180
Pakiet zgodności formatu plików pakietu
Microsoft Office, 60, 102, 723, 814
panel zadań, 36, 37
emulacja, 523
Panel sterowania, 678
Panel zadań, 46
ParamArray, 313
- Parent, 380
pasek
menu, 729
narzędzi, 154, 729, 919, 922
dostosowywanie, 41
niestandardowy, 761
pływający, 919
symulacja, 520
Szybki dostęp, 37, 41, 951
tworzenie, 730, 757, 758
przewijania, 133, 428
separatorka, 169
stanu, 500
ukrywanie, 919
PathExists, 373, 829
PDF, 104, 694
Personal Macro Workbook,
Patrz: makro arkusz osobisty
pętla, 204, 240
Do Until, 245
Do While, 244
For ... Next, 240
While Wend, 247
zła, 240
piaskownica, 106
Picture, 533, 534, 776
PivotCache, 571
PivotCaches, 565
PivotFields, 566, 567
PivotItems, 566
PivotTable, 150
PivotTables, 180, 566
PivotTableUpdate, 651
plik, 99, 104
aplikacji, 127
arkusza, 123
binarny, 112
blokada, 839
CHM, 783
CSV, *Patrz:* CSV
dodatków, 118
dostęp, 838
binarny, 838
losowy, 838
sekwencyjny, 838
Excel.OfficeUI, *Patrz:* OfficeUI
EXE, 919
format, 100, 103, 115
ACCDB, *Patrz:* ACCDB
ACCDE, *Patrz:* ACCDE
baz danych, 103
binarny, 102
CSV, *Patrz:* CSV
DBF, *Patrz:* DBF

- DIF, *Patrz*: DIF
HTM, *Patrz*: HTM
HTML, *Patrz*: HTML
MDB, *Patrz*: MDB
MDE, *Patrz*: MDE
MHT, *Patrz*: MHT
MHTML, *Patrz*: MHTML
ODS, *Patrz*: ODS
otwarty, 112
PDF, *Patrz*: PDF
PRN, *Patrz*: PRN
SLK, *Patrz*: SLK
TXT, *Patrz*: TXT
wymiany danych, 103
XLA, *Patrz*: XLA
XLAM, *Patrz*: XLAM
XLS, *Patrz*: XLS, *Patrz*: XLS
XLSB, *Patrz*: XLSB
XLSM, *Patrz*: XLSM
XLSX, *Patrz*: XLSX
XLT, *Patrz*: XLT
XLTX, *Patrz*: XLTX
XLTXM, *Patrz*: XLTXM
XLW, *Patrz*: XLW
XML, *Patrz*: XML
XPS, *Patrz*: XPS
zgodność, 813, 814, 818
ZIP, 112
GIF, 533
graficzny, 433, 533, 596
HLP, 793
HTML, *Patrz*: HTML
INI, 119
kompatybilność, 60, 105
konfiguracyjny, 116
konwerter, 121
lista, 829, 831
MHTML, *Patrz*: MHTML
MP3, 490
nazwa, 24
 MacOS, 818
 pobieranie, 413, 416
numer, 840
o dostępie sekwencyjnym, 840
 odczytywanie, 841
 zapisywanie, 841
obszaru roboczego, 104
ODS, *Patrz*: ODS
OfficeUI, *Patrz*: OfficeUI
operacje, 827, 843
 wejścia-wyjścia, 838
operacje VBA, 828, 829, 841
pakowanie, 851, 852
PDF, *Patrz*: PDF
Personal.xlsb, 178, 923
podpisany cyfrowo, 107
PRN, *Patrz*: PRN
przetwarzanie grup, 370
rels, 743
rozpakowywanie, 851, 854
skojarzenia, 393
szablonu, *Patrz*: szablon
ścieżka, 372, 373
tekstowy, 102, 103, 663, 827, 838
 dostęp sekwencyjny, 840
eksport, 840
eksportowanie zakresu, 843
filtrowanie zawartości, 846
import, 840
importowanie do zakresu, 844
odczyt danych, 841
odczytywanie, 839
otwieranie, 838
zapisywanie danych, 839
TXT, *Patrz*: TXT
właściwości, 836
współdzielenie, 839
XLB, *Patrz*: XLB
XLS, *Patrz*: XLS
XLSM, *Patrz*: XLSM
XLSX, *Patrz*: XLSX
 zgodność wersji, 60
PLIK, 101
PNG, 597
podmenu, 774, 776
podprogram, 159
pogrubienie zawartości komórek
 zawierających formuły, 653
pokrętło, 40, 133, 428, 807, 809
pole
 grupy, 133, 427
 kombi, 133, 426, 807, 946
 system pomocy, 790
 listy, 133, 427, 946
 tabele przestawnej, 564
 tekstowe, 428, 622, 784, 785, 807, 809
 wyboru, 40, 133, 426, 807
polecamie, 23, 38, *Patrz też*: słowo kluczowe
Cofnij, 41
długie, 204
Do Until, 245
Do While, 244
DoEvents, 492, 625
Exit For, 231
For ... Next, 240
GoTo, 232, 240
If ... Then, 232
If ... Then... Else, 234, 236

- polecenie
 Inspekcja formuł, 48
 kontynuacja, 204
 Load, 437
 On Error Resume Next, 282
 Option Explicit, 212
 przełącznik, 39
 przycisk, 39
 References, 257
 Run Sub/UserForm, 252, 253, 296
 Select Case, 236, 239
 While Wend, 247
 Wstążki, 731
 Wykonaj ponownie, 41
 Zapisz, 41
 pomoc, 61
 Pope Andy, 524
 Portabel Document Format, 104
 Power Utility Pak, *Patrz*: PUP
 PowerPivot, 36
 PowerView, 36
 PRAWY, 96,
 Private, 213, 250, 293, 327
 PRN, 103, 840
 procedura, 149, 159, 162, 249, 926
 argumenty, 249, 262, 264, 265, 266
 Function, 149, 158, 159, 250, 254, 266, 287,
 291, 293, 321, *Patrz też*: funkcja
 nazwa, 24, 250, 251
 niejednoznaczna, 257
 obsługi zdarzeń, 159, 261, 328, 437, 444,
 637, 927, 928, 944
 argumenty, 642
 lokalizacja, 638
 przycisk, 528
 tworzenie, 641
 wskaźnik postępu zadania, 502, 503
 wykres, 615
 pasek separatora, 169
 Property, 159, 882, 885
 Property Get, 882, 885
 Property Let, 882
 prywatna, 251, 254, 257
 publiczna, 251
 SendMail, 690
 ShowShortcutMenuNames, 762
 sortująca, 277
 Sub, 149, 158, 159, 162, 250, 266, 271, 287, 293
 wywoływanie, 252, 253, 254, 255
 testowanie, 277, 281
 tworzenie, 277
 uruchamianie, *Patrz*: wywoływanie
 wycofywanie, 558
 wykonywanie przez wiele obiektów, 880
 wyświetlanie, 863
 wywoływanie, 161, 249, 255, 256, 257, 259,
 260, 261, 262, 928
 zakończenie, 250
 zasięg, 251
 zwrotna, 736, 742, 745
 Procedure Separator, 169
 programowanie
 ADO, 855
 strukturalne, 241
 Project Explorer, 154, 155
 Export File, 157
 Import File, 157
 Insert, 156
 Modules, 156
 moduł, 156, 157
 References, 157
 Project Properties, 54
 projekt, 154, 155, 862
 Properties, 432
 Property, 159, 860, 882, 885
 Protect, 184
 ProtectStructure, 283
 przeglądarka
 internetowa, 58
 Microsoft Internet Explorer, 104
 obiektów, 201, 661, 833, 934
 Opera, 104
 system pomocy, 791
 przekształcanie dodatku w skoroszyt, 708
 przycisk, 56, 133, 944
 Anuluj, 947
 dodawanie, 870
 kreatora, *Patrz*: kreator przyciski
 OK, 947
 opcji, 133, 428, 631, 807
 pokrętła, 133
 polecenia, 133, 427
 przełącznika, 133, 429
 Zamknij, 465
 Przywróć okno, 158
 PtrSafe, 326, 932
 Public, 213, 215, 224, 250, 265, 293
 pułapka, 319
 PUP, 27, 71, 544, 545, 562
 puzzle, 538

Q

- QueryClose, 438, 447, 465, 666
 Quick Analysis, 36
 quick-sort, *Patrz*: sortowanie szybkie

R

R1C1 notation, *Patrz:* W1K1
 RANDOMINTEGERS, 388
 Range, 150, 180, 182, 183, 187, 193, 194, 196, 197, 351
 właściwości, 193
 RangeNameExists, 373
 RangeSelection, 938
 RECENZJA, 53, 136, 185, 728, 736
 ReDim, 940
 RefEdit, 428, 462, 947
 błędy, 462
 Reference, 860, 861
 References, 257, 258, 860, 861
 regedit.exe, 120
 reguła blokowania plików, 107
 rejestr systemu, 119, 121, 804, 807, 808, 939
 dostęp, 399
 klucz, 120, 532
 odczytywanie, 397, 399
 zapisywanie, 397, 398, 399
 rejestrator makr, *Patrz:* makro rejestrator
 rekurencja, 831
 RemoveControl, 666
 Require Variable Declaration, 167
 Resize, 611, 660, 666
 RGB, 189, 894, 895, 898
 zamiana na skalę szarości, 899
 Ribbon, *Patrz:* Wstążka
 RibbonX, *Patrz:* kod RibbonX
 RmDir, 828
 RowSource, 471, 472, 477, 481
 równanie
 liniowe, 60
 nielinijowe, 60
 równoważność logiczna, 221
 Run, 256, 259, 711
 Run Sub/UserForm, 161, 252, 253, 296
 rundll32.exe, 679

S

Safe Mode, *Patrz:* tryb awaryjny
 Sandbox, *Patrz:* piaskownica
 SaveSetting, 399, 532, 555, 808
 SAYIT, 378
 scenariusz, 148
 warunkowy, 60
 Schowek, 40
 ScreenUpdating, 610
 Scroll, 666
 ScrollBar, 131, 133, 428, 468, 469, 531
 ScrollColumns, 469

ScrollRow, 469
 SearchHelp, 796
 Seek, 840
 Select, 611, 637, 660, 935, 937
 Select Case, 236, 239, 366
 Selected, 475
 Selection, 178
 Selection.Areas.Count, 938
 Selection.Rows.Count, 938
 SelectionChange, 600, 637, 651, 657
 SendKeys, 951
 SendMail, 690
 SERIE, 599, 601, 602, 626
 SeriesChange, 611, 637, 660
 SERIESNAME_FROM_SERIES, 603
 Service Pack, *Patrz:* SP
 Service Release, *Patrz:* SR
 serwer, 680
 Set, 224, 890, 940
 SetAttr, 828
 SetWindowLong, 519
 Shape, 189, 192, 586, 622
 kolor, 908
 tła, 909
 sheet.xlsx, 111
 SheetActivate, 496, 498, 638, 642, 645, 646, 662
 SheetBeforeDoubleClick, 645, 662
 SheetBeforeRightClick, 645, 662
 SheetCalculate, 645, 662
 SheetChange, 638, 645, 662
 SheetDeactivate, 638, 645, 662
 SheetExists, 374
 SheetFollowHyperlink, 645, 662
 SheetOffset, 386
 SheetPivotTableUpdate, 645, 662
 SheetSelectionChange, 496, 498, 645, 662
 Shell, 673, 674
 ShellExecute, 676, 677
 Shortcut menu, *Patrz:* menu podręczne
 Show, 436
 ShowDataForm, 422
 ShowInstalledFonts, 368
 skala szarości, 897, 899
 skoroszyt, 32, 150, 180, 181, 783
 aktywny, 32, 192
 dezaktywacja, 647
 dodawanie arkusza, 647
 makr osobistych, *Patrz:* makro arkusz osobisty
 ochrona, *Patrz:* ochrona
 odwołanie, 257
 okno
 dodawanie, 33
 ukrywanie, 32
 otwarty, 375, 663

skoroszyt
 otwieranie, 662
 automatyczne, 929
 w trybie zgodności, 921
przetwarzanie, 360
szablon, 111, 112
trójwymiarowy, 386
tworzenie, 662
uaktywnianie, 646
udostępnienie, 137
wersja
 niezapisana, 108
 poprzednia, 107
zamknięty, 375
zamykanie, 360, 649
zapisywanie, 360, 647
zdarzenie, 644, 645, 647, 649
skróty klawiatury, 42, 47, 129, 130, 254, 435
SkyView, 36
Slicer, 36
SLK, 103
słowo kluczowe, 203, 205, 207, 924, *Patrz też:*
 polecenie,
 ByRef, 266
 ByVal, 265
 Call, 256, 259
 Const, 217
 Declare, 326
 Dim, 213, 225
 Do Until, 245
 Do While, 244
 Else, 233
 For ... Next, 240
 For Each, 227, 229
 Function, 293
 GoTo, 232, 240
 If ... Then, 230, 232
 If ... Then... Else, 234, 236
 Me, 438
 Optional, 307
 ParamArray, 313
 Private, 250, 293, 327, 930
 Property Get, 885
 PtrSafe, 326, 932
 Public, 215, 250, 265, 293
 Rem, 206
 Select Case, 236, 239
 Set, 224
 Static, 213, 216, 250, 293
 Sub, 250
 vbModeless, 496
 While Wend, 247
 With, 224, 227, 229
 WithEvents, 615, 663
SmartArt, 34, 38, 56, 259, 261
SmartTag, 68
Solver, 60, 70
sortowanie, 271, 277, 368
 arkuszowe, 369, 370
 bąbelkowe, 277, 369, 370
 szybkie, 369, 370
 zliczające, 369, 370
sortowanie metoda quick-sort, 369
SortSheets, 284, 285
SP, 141
spaghetti, 240, 241
Sparkline, 55, 633, *Patrz:* wykres przebiegu
 w czasie
SparklineGroup, 633
SpecialCells, 268
Speech, 922
SPELLDOLLARS, 385
SpinButton, 133, 428, 446, 449, 809
 TextBox, 449
 zdarzenie, 447, 448
Spinner, 133
spinners, *Patrz:* pokrętło
SpinUp, 446
spis treści, 362
splash screen, *Patrz:* okno powitalne
Split, 373
Sprawdzanie błędów, 48
Sprawdzanie zgodności, 568
Sqr, 226
SR, 141
stała, 73, 206, 216, 262, 298
 deklaracja, 217
 nazwa, 73
 predefiniowana, 217
PROGESSTHRESHOLD, 551
reprezentująca kolor, 893
vbAbort, 410
vbAbortRetryIgnore, 409
vbCancel, 410
vbCritical, 409
vbCrLf, 366, 412
vbDefaultButton, 409
vbExclamation, 409
vbext_ct_StdModule, 863
vbFormControlMenu, 466
vbGreen, 893
vbInformation, 409
vbModeless, 945
vbMsgBoxHelpButton, 409
vbNewLine, 412
vbNo, 228, 410
vbOK, 410
vbOKCancel, 409

- vbOKOnly, 366, 409
 vbQuestion, 228, 409
 vbRed, 893
 vbRetry, 410
 vbRetryCancel, 409
 vbSystemModal, 409
 vbYes, 228, 410
 vbYesNo, 228, 409
 vbYesNoCancel, 409
 xlMicrosoftWord, 678
 xl24HourClock, 824
 xl4DigitYears, 825
 xlAlternateArraySeparator, 824
 xlColumnSeparator, 824
 xlCountryCode, 823
 xlCountrySetting, 823
 xlCurrencyBefore, 825
 xlCurrencyCode, 824
 xlCurrencyDigits, 824
 xlCurrencyLeadingZeros, 825
 xlCurrencyMinusSign, 825
 xlCurrencyNegative, 824
 xlCurrencySpaceBefore, 825
 xlCurrencyTrailingZeros, 825
 xlDateOrder, 824
 xlDateSeparator, 824
 xlDayCode, 824
 xlDayLeadingZero, 825
 xlDecimalSeparator, 823
 xlDown, 336, 939
 xlErrDiv0, 312
 xlErrNA, 312
 xlErrName, 312
 xlErrNull, 312
 xlErrNum, 312
 xlErrRef, 312
 xlErrValue, 312, 932
 xlGeneralFormatName, 824
 xlHourCode, 824
 xlLandscape, 217
 xlLeftBrace, 824
 xlLeftBracket, 824
 xlListSeparator, 823
 xlLowerCaseColumnLetter, 824
 xlMDY, 825
 xlMetric, 825
 xlMicrosoftAccess, 678
 xlMicrosoftFoxPro, 678
 xlMicrosoftMail, 678
 xlMicrosoftPowerPoint, 678
 xlMicrosoftProject, 678
 xlMinuteCode, 824
 xlMonthCode, 824
 xlMonthLeadingZero, 825
 xlMonthNameChars, 824
 xlNoncurrencyDigits, 824
 xlNonEnglishFunctions, 825
 xlPortrait, 217
 xlRightBrace, 824
 xlRightBracket, 824
 xlRowSeparator, 824
 xlSecondCode, 824
 xlThousandsSeparator, 823
 xlTimeLeadingZero, 825
 xlTimeSeparator, 824
 xlToLeft, 336, 939
 xlToRight, 336, 939
 xlUp, 336, 939
 xlUpperCaseColumnLetter, 824
 xlUpperCaseRowLetter, 823
 xlWeekdayNameChars, 824
 xlYearCode, 824
 zasięg, 217
 StartUpPosition, 947
 Static, 213, 216, 250, 293
 StatusBar, 937
 stopka, 648
 Stosowanie nazw, 71
 String, 218
 structured referencing, *Patrz:* odwołanie strukturalne
 struktura sterująca, 203
 Style komórki, 110
 style tabeli, 922
 Sub, 149, 158, 233, 250, 264, 287, 293
 SUMA, 313
 suma logiczna, 221
 sumowanie warunkowe, 88, 89, 90
 Sun Microsystems, 104
 symbol, *Patrz też:* znak
 dnia, 824
 waluty, 51, 824
 system
 metryczny, 825
 miar angielski, 825
 operacyjny, 492
 plików, 833
 pomocy, 61, 139, 185, 186, 201, 781, 784
 arkusz, 784, 786
 etykieta, 788
 etykieta przewijana, 788
 HTML Help, 793, 794
 komentarze do zawartości komórek, 784
 kontekstowy, 805
 kształt, 785
 łączenie z aplikacją, 796
 metoda Help, 795
 MHTML, 792
 nieoficjalny, 783

system

- oficjalny, 783
- online, 781, 783
- pole kombi, 790
- pole tekstowe, 784, 785
- poprawność danych, 785
- przeglądarka sieciowa, 791
- przypisanie tematów do funkcji, 797
- tematy, 796
- UserForm, 784, 788
- rejestr, *Patrz*: rejestr systemu
- szablon, 108, 110, 783, 800
- arkusza domyślny, 109
- galeria, 109
- skoroszytu, 111, 112
 - domyślny, 109
 - własny, 109
- tworzenie, 109
- ustawienia domyślne, 109

Szacuj formułę, 84

Szyfrowanie dokumentu, 54

Ś

środowisko

- IDE, *Patrz*: IDE
- Visual Basic, *Patrz*: VBE

T

Tab, 434

tabela, 57, 68, 335, 732, 920

kolumna, 57

przestawna, 46, 55, 59, 127, 148, 180, 563, 651

bufor, 565

CreatePivotTable, 566, 571

dane źródłowe, 566

działanie, 571

kompatybilność, 568

lista pól, 564

nazwa, 75

odwrócona, 576

optymalizacj, 566

PivotCache, 571

PivotCaches, 565

PivotFields, 566, 567

PivotItems, 566

PivotTables, 566

pole danych, 566

pole kategorii, 566

tworzenie, 564, 569, 573

przestawne, 90

styl formatowania, 658

znormalizowana, 566

tabele przestawne, 563

Lista pól tabeli przestawnej, 572

tablica, 85, 222, 262, 298, 941

deklaracja, 222

deklarowanie, 940

dynamiczna, 223, 940

jako argument, 305

jednowymiarowa, 355

liczb losowych, 388

pionowa, 309, 356

pozioma, 309, 356

sortowanie, 277, 368, 937

Variant, 926

VBA, 308

wielowymiarowa, 223

zmiennych typu Variant, 309

TabStrip, 428, 487

tag, *Patrz*: SmartTag

Tag, 451

tekst

łańcuch, *Patrz*: łańcuch znakowy

przetwarzanie, 545, 547

TERAZ, 299

Terminate, 438, 447, 666, 887, 945

Text, 197

TextBox, 133, 428, 490, 809

SpinBox, 449

TextStream, 838

ThemeColor, 905

ThisWorkbook, 159, 639, 698

Timeline Slicer, 36

Timer, 353

TintAndShade, 905, 911

ToggleButton, 133, 429

ToggleWrapText, 364

Toolbar, *Patrz*: pasek narzędzi

Toolbox, 425, 453, 488

dodawanie formantów, 453, 455

dodawanie kart, 453

Topic ID, *Patrz*: system pomocy tematy

TRANSPONUJ, 309, 356

Transpose, 473

TRANSPOSE, 356

True, 364

tryb awaryjny, 100

TXT, 103, 840

typy danych, *Patrz*: wartość

U

UCase, 226
UKŁAD STRONY, 37, 40, 728, 735
urządzenie mobilne, 36
UsedRange, 348
UserForm, 34, 46, 129, 131, 159, 403, 423, 429, 869, 943
 bez paska tytułowego, 519
 ładowanie, 437
 menu, 459
 niemodalny, 945
 odwołania, 451
 okno
 niemodalne, 436, 437
 pasek tytułu, 947
 pozycja początkowa, 947
 półprzezroczysty, 534
 procedura obsługi zdarzeń, 444
 przewijanie, 468
 przykłady, 459
 rozmiar
 zmiana, 524
 sprawdzanie poprawności danych, 445
 system pomocy, 788
 szablon, 455
 tablica formantów, 944
 testowanie, 435
 tworzenie, 423, 439
 programowe, 872, 874, 876, 878
 ukrywanie, 438, 945
 umieszczanie formantów, 868
 usuwanie z pamięci, 945
 wielkość, 466, 468
 wskaźnik postępu zadania, 499, 501
 wstawianie, 424
 wybór koloru, 531
 wykres, 532, 608, 944
 wyświetlanie, 436
 na podstawie zmiennej, 437
 z wieloma opcjami, 487
 zakres, 461
 zamykanie, 438
 zdarzenie, 445, 447, 638, 665
UserForms, 100, 170
UserInterfaceOnly, 925
ustawienia
 domyślne, 807
 międzynarodowe, *Patrz:* Excel wersje językowe
USUŃ.ZBĘDNE.ODSTĘPY, 546
utwardzanie rozszerzeń, 698
użytkownik, 366

V

Value, 152, 182, 197, 433, 506
Value2, 197
Values, 602
VALUES_FROM_SERIES, 603, 604
Variant, 235, 308, 356, 926
VBA, 21, 31, 59, 60, 70, 100, 147, 148, 203, 544
 błędy, 155, 266
 data, 95
 Do While, 244
 edytor, 23
 kod
 generowanie, 867
 programu, 149
 przetwarzanie dodatków, 714
 wersja językowa, 822
 źródłowy, 24, 154, 158, 159, 160, 162, 163, 204, 275
 kolekcja, *Patrz:* kolekcja
 kolor, 895, 924
 Macintosh, 149
 model obiektowy, 148
 moduł, 149, 862, 639
 dodawanie, 156
 eksportowanie, 157
 importowanie, 157
 usuwanie, 156
 obiekt, *Patrz:* obiekt
 odwołanie, 181
 optymalizacja, 942
 procedura, *Patrz:* procedura
 przykłady, 331
 technika programowania, 364
 testowanie, 155
 wprowadzenie, 149
 zdarzenie, *Patrz:* zdarzenie
 zmienna, *Patrz:* zmienna
vbAbort, 410
vbAbortRetryIgnore, 409
vbCancel, 410
VBComponent, 860, 864
VBCOMPONENTS, 860, 861
vbCritical, 409
vbCrLf, 366, 412
vbDefaultButton, 409
vbDirectory, 831
VBE, 23, 149, 152, 155, 166, 205, 262, 857, 860, 923
 Auto Data Tips, 168
 Auto Indent, 168
 Auto List Members, 167
 Auto Quick Info, 168

VBE

Auto Syntax Check, 166
 błąd, 168, 171
 składni, 166, 205
 Code, 154, 157
 czcionki, 170
 Default to Full Module View, 169
 Drag-and-Drop Text Editing, 169
 Editor Format, 169
 edytor, 47
 Immediate, 155
 kod źródłowy, 154, 157, 158, 159, 163
 komplikacja, 171
 lista funkcji, 226
 menu podręczne, 153
 okna, 153
 pasek
 menu, 153
 narzędzi, 154
 podpowiedzi, 171
 pomoc, 167
 Procedure Separator, 169
 Project Explorer, *Patrz:* Project Explorer
 projekt, *Patrz:* projekt
 Require Variable Declaration, 167
 skoroszyt, *Patrz:* projekt
 tekstu kopiowanie, 169
 uruchamianie, 153
 wcięcia, 168
 wygląd, 169
 vbExclamation, 409
 vbext_ct_StdModule, 863
 vbFormControlMenu, 466
 vbHidden, 831
 vbIgnore, 410
 vbInformation, 409
 vbModeless, 496, 945
 vbMsgBoxHelpButton, 409
 vbNewLine, 412
 vbNo, 228, 410
 vbNormal, 831
 vbOK, 410
 vbOKCancel, 409
 vbOKOnly, 366, 409
 VBProject, 858, 860, 861
 VBProjects, 860
 vbQuestion, 228, 409
 vbReadOnly, 831
 vbRetry, 410
 vbRetryCancel, 409
 vbSystem, 831
 vbSystemModal, 409
 vbVolume, 831
 vbYes, 228, 410
 vbYesNo, 228, 409

vbYesNoCancel, 409
 ViewCustomViews, 732
 VisiCalc, 103
 Visual Basic Editor, *Patrz:* VBE
 Visual Basic Environment, *Patrz:* VBE
 Visual Basic for Applications, 21
 Visual Basic for Windows, 148
 VSTO, 21

W

W1K1, 66, 67
 wartość, 215
 ActiveChart, 589
 Boolean, *Patrz:* wartość logiczna
 Byte, 209
 Currency, 209, 215, 216, 927
 Date, 209, 216, 218
 Decimal, 208, 209
 Double, 206, 208, 209, 215, 216, 927
 funkcji, 291, 293
 Integer, 208, 209, 215, 216, 927
 liczbową, 48
 logiczna, 34, 48, 206, 209, 216, 364
 Long, 208, 209, 215, 216, 927
 maksymalna, 387
 numeryczną, 51
 Object, 209, 216
 przełączanie, 364
 Single, 209, 215, 216, 927
 String, 209, 215, 216, 927
 Time, 216
 Variant, 208, 209, 216, 235, 291, 926
 wydajność, 210
 xlRowField, 571
 zamiana na słowo, 385
 WAV, 922
 wczesne wiązanie, 681
 Weekday, 237
 Wend, 958
 wersja beta, 135
 While, 958
 While Wend, 247
 wiadomość e-mail, 690
 załącznik, 693
 wiązanie
 późne, 681, 683, 684
 wczesne, 681, 837
 wideo Poker, 540
 widok chroniony, 106
 WIDOK, 32, 33, 40, 78, 104, 728, 734, 736
 Width, 527
 wiersz
 nazwa, 71
 ostatnia komórka, 939

-
- powielanie, 349
pusty, 348
ukrywanie, 34, 137
usuwanie, 348
wysokość, 34
Window, 180, 860
WindowActivate, 645, 662
WindowDeactivate, 645, 662
WindowResize, 645, 662
Windows, *Patrz:* rejestr systemu
Windows API, *Patrz:* API
Windows Explorer, 676
Windows Help System, 793
Windows Media Player, 488
Windows RT, 949
Windows Scripting Host, 833, 834, 838
WinHelp, 793
wirus, 858, 925
With, 177, 224, 227, 229
WithEvents, 663
właściwość, 182, 200, 433, 884, 934
 ActiveCell, 191, 336
 ActivePrinter, 395
 Application, 187
 argumenty, 168, 184
 Author, 187
 BackColor, 909
 BoundColumn, 481
 Calculation, 920
 Color, 908
 ColumnCount, 470
 ColumnHeads, 470
 Comment, 188
 ControlSource, 470, 944
 ControlTipText, 789
 Count, 188
 Creator, 187
 Dependents, 499
 DisplayAlerts, 930
 do odczytu i zapisu, 885
 domyślna, 183
 EnableCalculation, 920
 Enabled, 809
 Fill, 189, 909
 ForeColor, 189, 909, 913
 Formula, 602
 International, 823
 IsAddin, 698
 ListIndex, 461
 logiczna, 364
 MultiSelect, 470, 475
 Name, 937
 nazwa, 24
ObjectThemeColor, 911
OperatingSystem, 817
Parent, 187, 380
Picture, 533, 534
Precedents, 499
przelączanie, 364
RangeSelection, 938
RGB, 189
rng.Column, 382
rng.Parent, 382
Rows.Count, 382
RowSource, 471, 472, 477, 481
ScreenUpdating, 610
ScrollColumns, 469
ScrollRow, 469
Selected, 475
Selection, 191
Shape, 187
StartUpPosition, 947
Tag, 451
Text, 187
ThemeColor, 905
TintAndShade, 905, 911
tylko do odczytu, 885
tylko do zapisu, 885
Values, 602
Visible, 187
wersja lokalna, 822
Xvalues, 599
XValues, 602
Zoom, 469
Word, 680
 sterowanie z Excela, 685
Word.Application, 683
WordArt, 259, 261
Workbook, 150, 180, 184, 581, 688
Workbook_BeforeClose, 929
Workbook_Open, 777, 927
WorkbookActivate, 662
WorkbookAddinInstall, 662
WorkbookAddinUninstall, 662
WorkbookBeforeClose, 638, 662
WorkbookBeforePrint, 662
WorkbookBeforeSave, 662
WorkbookDeactivate, 662
WorkbookIsOpen, 375
WorkbookNewSheet, 638, 662
WorkbookOpen, 662
Workbooks, 118, 180, 181, 709
Worksheet, 150, 180, 181, 193, 194
WorksheetFunction, 227, 958
Worksheets, 150, 180, 181
workspace file, *Patrz:* plik obszaru roboczego
wrapper function, 300, *Patrz:* funkcja osłonowa

- Write, 841
 WriteRegistry, 398
 wskaźnik postępu, 499, 507, 945
 MultiPage, 504, 505, 506, 507
 na pasku stanu, 500
 procedura obsługi zdarzeń, 502, 503
 samodzielny, 500, 501, 503
 WSTAWIANIE, 37, 56, 57, 728, 735
 Wstawianie funkcji, 49, 320
 Wstążka, 23, 37, 255, 727
 aktywacja karty, 951
 dodawanie makra, 951
 dodawanie poleceń, 286
 dostosowywanie, 128, 129, 736
 formant, 731, 747
 karty, 23, 728
 karty kontekstowe, 38
 konfiguracja, 736
 lista rozwijana, *Patrz*: lista rozwijana
 modyfikacja, 745, 756, 809
 modyfikowanie
 dynamiczne, 753
 pasek poleceń, 950
 pokrętło, *Patrz*: pokrętło
 pole wyboru, *Patrz*: pole wyboru
 przycisk, *Patrz*: polecenie przycisk
 skróty klawiszowe, 41
 ukrywanie, 38, 919
 umieszczanie poleceń, 560
 VBA, 731
 wydajność systemu, 142
 wydruk, *Patrz*: drukowanie
 wykres, 34, 55, 148, 150, 180, 259, 579
 aktywacja, 586, 588, 938
 aktywny, 588
 animacja, 936
 animowany, 624, 627
 zegar, 628
 arkusz, *Patrz*: arkusz wykresu
 Chart, 581
 ChartObject, 581
 dane
 etykieta, 605
 identyfikacja zakresu, 600
 seria, 601
 ukrywanie, 618
 dezaktywacja, 587, 660
 eksportowanie, 596
 filtry, 620
 formatowanie, 36, 46
 GIF, 597
 interaktywny, 629, 630, 633
 przycisk opcji, 631
 zakres danych, 632
 kolor, 912
 losowy, 915
 krzywych hipocykloidalnych, 627
 lista rozwijana, 631
 lokalizacja, 579
 modyfikacja
 na podstawie aktywnej komórki, 599
 modyfikowanie, 584
 nazwa, 69, 588
 osadzony
 drukowanie, 618
 zdarzenie, 614, 616
 osadzony na arkuszu danych, 582
 plik GIF, 533
 procedura obsługi zdarzeń, 615
 przebieg w czasie, 55, 633
 przenoszenie, 587
 przestawny, 55
 przetwarzanie, 590
 przewijanie, 625
 rejestrator makr, 580
 rozmiar, 660
 zmiana, 593
 statyczny, 620
 tworzenie, 584
 UserForm, 532, 608, 944
 w komórce, 633
 wyrównywanie, 593
 wyświetlanie tekstu, 621
 zapisywanie
 jako GIF, 930
 zdarzenie, 611, 637, 660
 zmiana danych, 599
 wypełnianie błyskawiczne, *Patrz*: Flash Fill
 wyrażenie, 220, 262, 298
 wzorzec, 383

X

- x1MicrosoftWord, 678
 XDATE, 317
 XDATEADD, 317
 XDATEDAY, 317
 XDATEDIF, 317
 XDATEMONTH, 317
 XDATEYEAR, 317
 XDATEYEARDIF, 317
 xl4DigitYears, 825
 XLA, 102, 548, 698, 706
 xlAlternateArraySeparator, 824
 XLAM, 101, 118, 547, 548, 560, 698, 706, 709,
 928, 943, 949, 950
 arkusz, 710
 widoczność, 709
 wykres, 710

- XLB, 117
xlColumnSeparator, 824
xlCountryCode, 823
xlCountrySetting, 823
xlCurrencyBefore, 825
xlCurrencyLeadingZeros, 825
xlCurrencyMinusSign, 825
xlCurrencySpaceBefore, 825
xlCurrencyTrailingZeros, 825
xlDateSeparator, 824
xlDayCode, 824
xlDayLeadingZero, 825
xlDecimalSeparator, 823
xlDown, 336, 939
xlErrDiv0, 312
xlErrNA, 312
xlErrName, 312
xlErrNull, 312
xlErrNum, 312
xlErrRef, 312
xlErrValue, 312, 932
xlHourCode, 824
XLL, 118, 706
xLLandscape, 217
xlLeftBrace, 824
xlLeftBracket, 824
xLListSeparator, 823
xlLowerCaseColumnLetter, 824
xlLowerCaseRowLetter, 824
XLM, 34, 60, 148, 544, 929
makro, 148, 149
xlMDY, 825
xlMetric, 825
xlMicrosoftAccess, 678
xlMicrosoftFoxPro, 678
xlMicrosoftMail, 678
xlMicrosoftPowerPoint, 678
xlMicrosoftProject, 678
xlMinuteCode, 824
xlMonthCode, 824
xlMonthLeadingZero, 825
xlNonEnglishFunctions, 825
xIPortrait, 217
xlRightBrace, 824
xlRightBracket, 824
xIRowField, 571
xIRowSeparator, 824
XLS, 33, 102, 105
XLSB, 101, 108
xlSecondCode, 824
XLSM, 33, 101, 112, 698, 705, 709, 918, 949, 950
arkusz, 710
widoczność, 709
wykres, 710
XLSX, 33, 100, 101, 918
XLT, 102
xlThousandsSeparator, 823
xlTimeLeadingZero, 825
xlTimeSeparator, 824
xlToLeft, 336, 939
xlToRight, 336, 939
XLTX, 101, 109
XLTXM, 101
xlUp, 336, 939
xlUpperCaseColumnLetter, 824
xlUpperCaseRowLetter, 823
XLW, 104
xlYearCode, 824
XML, 46, 102, 112, 115, 116, 149, 736, 849
XML Paper Specification, 104
XMLSS, 102
XPS, 104
XVALUE_FROM_SERIES, 604
XValues, 599, 602
XVALUES_FROM_SERIES, 603

Z

zadanie

- kreatora, *Patrz*: kreator wykonywanie zadań panel, 36, 37 emulacja, 523
- zakres, 76 aktywacja, 935 danych na wykresie, 600 eksportowanie do pliku, 888 identyfikacja, 335 kopianie, 332, 334 nazwa, 24, 936 nazwany, 75 nieciągły, 358 odczytywanie, 352 określanie typu, 344 podzakres, 351 porządkowanie w sposób losowy, 389 przenoszenie, 334 przetwarzanie, 332, 335, 346 sortowanie, 391 zapisywanie, 352, 354 zaznaczanie, 335, 461, 935, 936 zmiana rozmiaru, 337
- załącznik, 693
- Zapisz jako, 69
- Zapisz pliki w następującym formacie, 101
- zapytanie sieciowe, 58
- Zastosuj nazwy, 71

- zdarzenie, 152, 159, 261, 437, 637, 805, 927, 944
Activate, 446, 611, 644, 646, 651, 660, 666
AddControl, 666
AddInInstall, 644, 719, 721
AddInUninstall, 644, 719
AfterCalculate, 662
AfterSave, 644
AfterUpdate, 448
Activate, 447
aplikacja, 638, 660, 662, 664
arkusz, 637, 651, 652, 653, 657, 658, 659
BeforeClose, 644, 649
BeforeDoubleClick, 611, 651, 658, 660
BeforeDragOver, 448, 666
BeforeDropOrPaste, 448, 666
BeforePrint, 644, 648
BeforeRightClick, 651, 659, 660
BeforeSave, 637, 644, 647
BeforeUpdate, 448
Button_Click, 328
Calculate, 611, 637, 651, 660
Change, 446, 448, 450, 512, 651, 652, 653, 654, 927
Click, 510, 666
DblClick, 666
Deactivate, 447, 611, 644, 647, 651, 660, 666
Enter, 448
Error, 448, 666
Exit, 448
FollowHyperlink, 651
Initialize, 446, 447, 452, 638, 666, 887
KeyDown, 448, 666
KeyPress, 448, 666
KeyUp, 448, 666
klawiatura, 448
kompatybilność, 640
Layout, 666
menu podręczne, 777
modułu klasy, 887
MouseDown, 611, 660, 666
MouseMove, 611, 660, 661, 666
MouseOver, 621, 622
MouseUp, 611, 660, 666
mysz, 448
NewSheet, 637, 644, 647
NewWorkbook, 638, 662
obiektu Application, 662
obsługa, 261
OnKey, 638, 666, 668, 669
OnTime, 638, 666, 667
Open, 637, 644, 645
PivotTableUpdate, 651
planowanie, 667
QueryClose, 438, 447, 465, 666
- RemoveControl, 666
Resize, 611, 660, 666
Scroll, 666
sekwencja, 638
Select, 611, 637, 660
SelectionChange, 600, 637, 651, 657
SeriesChange, 611, 637, 660
SheetActivate, 496, 498, 638, 642, 645, 646, 662
SheetBeforeDoubleClick, 645, 662
SheetBeforeRightClick, 645, 662
SheetCalculate, 645, 662
SheetChange, 638, 645, 662
SheetDeactivate, 638, 645, 662
SheetFollowHyperlink, 645, 662
SheetPivotTableUpdate, 645, 662
SheetSelectionChange, 496, 498, 645, 662
skoroszyt, 637, 644, 645, 647, 649
SpinButton, 447, 448
SpinDown, 448
SpinUp, 446, 448
Terminate, 438, 666, 887, 945
UserForm, 445, 447, 638, 665
WindowActivate, 645, 662
WindowDeactivate, 645, 662
WindowResize, 645, 662
Workbook_Open, 152
WorkbookAddinInstall, 662
WorkbookAddinUninstall, 662
WorkbookBeforeClose, 638, 662
WorkbookBeforePrint, 662
WorkbookBeforeSave, 662
WorkbookDeactivate, 662
WorkbookNewSheet, 638, 662
WorkbookOpen, 662
Worksheet_Change, 152
wykres, 611, 637, 660
 osadzony, 614, 616
wyłączanie obsługi, 640
wyszukiwanie, 661
Zoom, 666
- ZDATEDOW, 317
zegar, 628
ZIP, 112, 115, 851, 852, 854
zliczanie warunkowe, 88, 90
ZŁĄCZ.TEKSTY, 546
zmienna kolejności tabulacji formantów, 434
zmienna, 152, 167, 204, 206, 262, 298
 deklaracja, 204, 209, 211, 212
 wymuszona, 212
globalna, 215
lokalna, 213, 930
łańcuchowa, 218
modułowa, 214
nazwa, 205, 207, 212, 216

- obiektowa, 224, 940
- przypisanie, 224
- poziomu modułu, 930
- prywatna, 888
- publiczna, 265, 516, 930
- statyczna, 215
- typ, *Patrz*: wartość
- zasięg, 213, 930
- zawierająca tablicę, 309
- znak
 - !, 215, 927
 - ", 928, 941
 - #, 215, 927
 - //, 525
 - \$, 65, 215, 927
 - %, 215, 670, 926, 927
 - &, 215, 764, 925, 927
- ?, 297
- @, 68, 215, 927
- ^, 670
- =, 220
- definicji typu, 926
- deklaracji typu, 215
- kropki szybkość przetwarzania, 224
- łańcuch, *Patrz*: łańcuch znakowy
- plus, 670
- podkreślenia, 24, 925
- podziału strony, 544
- podziału wiersza, 412
- powrotu karetki, 933
- tabulacji, 412
- wysuwu wiersza, 933
- Zoom, 469, 666

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

- 
1. ZAREJESTRUJ SIĘ
 2. PREZENTUJ KSIĄŻKI
 3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>