

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

KATALOG KSIĄŻEK

KATALOG ONLINE

ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK

DODAJ DO KOSZYKA

CENNIK I INFORMACJE

ZAMÓW INFORMACJE
O NOWOŚCIACH

ZAMÓW CENNIK

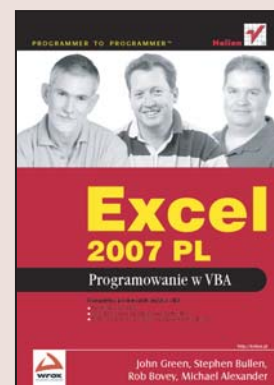
CZYTELNIA

FRAGMENTY KSIĄŻEK ONLINE

Excel 2007 PL. Programowanie w VBA

Autor: John Green, Stephen Bullen,
Rob Bovey, Michael Alexander
Tłumaczenie: Maria Chaniewska
(wstęp, rozdz. 1 - 27), Krzysztof Bąbol
(dodatek A), Paweł Dyl (dodatki B - C)
ISBN: 978-83-246-1357-1

Tytuł oryginału: [Excel 2007 VBA Programmer](#)



Kompletny podręcznik języka VBA

- Praktyczny opis języka
- Współpraca z innymi aplikacjami pakietu Office
- Pełny opis metod, właściwości i zdarzeń obiektów Excela

Visual Basic for Applications, w skrócie VBA, to język programowania, który pozwala tworzyć programy wykorzystujące olbrzymie możliwości Excela. Dzięki niemu można zarówno zbudować makro polecenia czy zautomatyzować żmudne prace wykonywane w arkuszu kalkulacyjnym, jak i zaprojektować oraz wykonać zaawansowane, komunikujące się z użytkownikiem aplikacje, współpracujące z zewnętrznymi danymi. Książka „Excel 2007 PL. Programowanie w VBA” to znakomite kompendium wiedzy o tym, jak za pomocą potęgi języka VBA rozbudować arkusze Excela. Korzystając z niej, nauczysz się wykorzystywać nazwy, zakresy i listy danych, a także tworzyć tabele i wykresy. Poznasz opracowane na podstawie wieloletnich doświadczeń utalentowanych programistów techniki projektowania interaktywnych aplikacji wykorzystujących formularze i Windows API. Dowiesz się, jak zarządzać zewnętrznymi danymi w różnych formatach oraz publikować arkusze w internecie, nie popełniając błędów związanych z niewłaściwymi ustawieniami lokalizacyjnymi. W książce tej znajdziesz także niezbędne wskazówki i gruntownie omówione przykłady, pozwalające szybko zdobyć wiedzę potrzebną do tworzenia nowoczesnych i profesjonalnych aplikacji w Excelu.

- Podstawy Excel VBA
- Nazwy, zakresy i listy danych
- Tabele i wykresy
- Obsługa zdarzeń
- Formanty
- XML i OpenXML
- Formularze
- RibbonX
- Źródła danych OLAP
- Visual Basic Editor - obsługa i narzędzia
- Windows API
- Lokalizacja
- Model obiektowy Excela 2007

Poznaj pełnię możliwości, zaawansowane oblicze Excela!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl



Spis treści

O autorach	19
Podziękowania	21
Wstęp	23
Rozdział 1. Excel VBA — elementarz	33
Używanie narzędzia do rejestrowania makr	34
Rejestrowanie makr	35
Uruchamianie makr	38
Visual Basic Editor	41
Inne sposoby uruchamiania makr	44
Funkcje zdefiniowane przez użytkownika	50
Tworzenie UDF	50
Czego funkcje UDF nie potrafią	54
Model obiektowy programu Excel	54
Obiekty	55
Uzyskiwanie pomocy	60
Eksperymenty w oknie Immediate	62
Język VBA	63
Podstawy wejścia i wyjścia	64
Wywoływanie funkcji i procedur sub	69
Nawiasy i listy parametrów	70
Deklaracje zmiennych	72
Zakres i czas życia zmiennych	74
Typy zmiennych	76
Zmienne obiektowe	79
Podejmowanie decyzji	82
Pętle	85
Tablice	91
Obsługa błędów czasu wykonania	94
Podsumowanie	98

Rozdział 2. Obiekt Application	99
Metody i właściwości globalne	99
Aktywne właściwości	100
Wyświetlanie ostrzeżeń	101
Aktualizowanie ekranu	102
Metoda Evaluate	102
Funkcja InputBox	104
Właściwość StatusBar	106
Funkcja SendKeys	106
Metoda OnTime	107
Metoda OnKey	109
Funkcje arkusza	110
Właściwość Caller	111
Podsumowanie	112
Rozdział 3. Skoroszyty i arkusze	113
Kolekcja Workbooks	113
Pobieranie nazwy pliku ze ścieżki	114
Pliki w tym samym katalogu	117
Nadpisywanie istniejących skoroszytów	117
Zapisywanie zmian	119
Kolekcja Sheets	120
Arkusze kalkulacyjne	120
Kopiowanie i przenoszenie	122
Grupowanie arkuszy	124
Obiekt Window	125
Synchronizowanie arkuszy	127
Podsumowanie	129
Rozdział 4. Używanie zakresów	131
Metody Activate i Select	131
Właściwość Range	132
Skrócone wskazania zakresów	133
Zakresy w nieaktywnych arkuszach	134
Właściwość Range obiektu Range	134
Właściwość Cells	135
Cells używane w Range	136
Zakresy w nieaktywnych arkuszach	136
Więcej na temat właściwości Cells obiektu Range	137
Wskaźnik jednoparametrowy do Range	139
Właściwość Offset	140
Właściwość Resize	141
Metoda SpecialCells	143
Ostatnia komórka	143
Usuwanie liczb	145
Właściwość CurrentRegion	146
Właściwość End	148
Wskazywanie zakresów z wykorzystaniem End	148
Sumowanie zakresów	149
Właściwości Columns i Rows	149
Obszary	151

Metody Union i Intersect	152
Puste komórki	153
Przekazywanie wartości między tablicami i zakresami	155
Usuwanie wierszy	158
Podsumowanie	161
Rozdział 5. Używanie nazw	163
Nazywanie zakresów	165
Używanie właściwości Name obiektu Range	165
Nazwy specjalne	166
Przechowywanie wartości w nazwach	167
Przechowywanie tablic	168
Ukrywanie nazw	169
Praca z nazwanymi zakresami	170
Szukanie nazwy	171
Szukanie nazwy zakresu	172
Wyznaczanie, które nazwy nakładają się na zakres	174
Podsumowanie	177
Rozdział 6. Listy danych	179
Strukturalizowanie danych	179
Sortowanie zakresu	180
Starsze wersje Excela	182
Tworzenie tabeli	182
Sortowanie tabeli	183
Autofiltr	184
Obiekt Autofiltr	185
Obiekt Filter	186
Własny filtr daty	186
Dodawanie pól kombi	187
Kopiowanie widocznych wierszy	191
Znajdowanie widocznych wierszy	192
Zaawansowany filtr	194
Formularz danych	196
Podsumowanie	198
Rozdział 7. Tabele przestawne	199
Tworzenie raportu tabeli przestawnej	200
Obiekty PivotCache	203
Kolekcja PivotTables	204
Kolekcja PivotFields	204
Pola wyliczane	207
Kolekcja PivotItems	209
Grupowanie	210
Właściwość Visible	213
Elementy wyliczane	214
Wykresy przestawne	215
Zewnętrzne źródła danych	216
Podsumowanie	218

Rozdział 8. Wykresy	219
Arkusze wykresów	220
Nagrane makro	220
Dodawanie arkusza wykresu z wykorzystaniem kodu VBA	222
Zagnieżdżone wykresy	222
Użycie rejestratora makr	223
Dodawanie zagnieżdżonego wykresu za pomocą kodu VBA	224
Edycja serii danych	225
Definiowanie serii wykresów za pomocą tablic	228
Konwertowanie wykresu, aby używał tablic	231
Wyznaczanie zakresów użytych w wykresie	231
Etykiety wykresów	233
Podsumowanie	234
Rozdział 9. Procedury obsługi zdarzeń	237
Zdarzenia arkusza	238
Włączanie zdarzeń	239
Przeliczenie arkusza	239
Zdarzenia wykresów	240
Przed dwukrotnym kliknięciem	241
Zdarzenia skoroszytu	243
Zapisywanie zmian	245
Nagłówki i stopki	246
Podsumowanie	246
Rozdział 10. Dodawanie formantów	249
Formanty formularza i ActiveX	249
Formanty ActiveX	250
Pasek przewijania	251
Przycisk pokrętła	251
Pole wyboru	252
Przyciski opcji	253
Formanty formularza	254
Dynamiczne formanty ActiveX	257
Formanty na wykresach	260
Podsumowanie	260
Rozdział 11. Pliki tekstowe i okno FileDialog	263
Otwieranie plików tekstowych	263
Pisanie do plików tekstowych	264
Czytanie plików tekstowych	265
Pisanie do plików tekstowych za pomocą metody Print	267
Czytanie łańcuchów znaków	268
Elastyczne separatory i ograniczniki	270
FileDialog	272
FileDialogFilters	274
FileDialogSelectedItems	274
Typy okien dialogowych	275
Metoda Execute	275
MultiSelect	275
Podsumowanie	275

Rozdział 12. Praca z plikami w formatach XML i Open XML	279
Podstawy używania danych XML w Excelu	280
Podstawy XML	280
Bezpośrednie konsumowanie danych XML	286
Tworzenie własnych map XML i zarządzanie nimi	289
Używanie VBA do programowania procesów XML	293
Programowanie map XML	293
Korzystanie z DOM i XPath do manipulowania plikami XML	298
Używanie VBA do programowania plików Open XML	306
Programowanie plików Open XML w VBA	306
Podsumowanie	313
Rozdział 13. Formularze	315
Wyświetlanie formularzy	315
Tworzenie formularza	316
Bezpośredni dostęp do formantów w UserForm	319
Wyłączenie przycisku zamykającego	323
Utrzymywanie listy danych	324
Formularze niemodalne	331
Wskaźnik postępu	331
Zmienna jako nazwa formularza UserForm	334
Podsumowanie	334
Rozdział 14. RibbonX	335
Przegląd	335
Wymagania wstępne	336
Dodawanie modyfikacji	337
Struktura XML	337
RibbonX i VBA	340
Typy kontroltek	341
Podstawowe kontrolki	341
Kontrolki kontenera	341
Atrybuty kontroltek	343
Wywołania zwrotne kontroltek	345
Zarządzanie obrazami kontroltek	346
Inne elementy, atrybuty i wywołania zwrotne RibbonX	349
Współdzielenie kontroltek między wieloma skoroszytami	350
Uaktualnienie kontroltek w czasie działania	351
Przechwytywanie wbudowanych kontroltek	354
RibbonX w dedykowanych aplikacjach	354
Dostosowywanie menu Office	355
Dostosowywanie paska narzędzi Szybki dostęp	355
Kontrolowanie kart, zbiorów kart i grup	356
Kontrolki dynamiczne	357
dropDown, comboBox i gallery	358
dynamicMenu	358
Rozszerzenia CommandBar dla Wstążki	360
Ograniczenia RibbonX	360
Podsumowanie	361

Rozdział 15. Paski poleceń	363
Paski narzędzi, paski menu i menu podręczne	364
Wbudowane paski poleceń Excela	366
Kontrolki na wszystkich poziomach	369
Artybuty Faceld	372
Tworzenie nowych menu	374
Makra OnAction	376
Przekazywanie wartości parametrów	377
Usuwanie menu	378
Tworzenie paska narzędzi	379
Menu podręczne	382
Ukazywanie wyskakujących pasków poleceń	386
Wykorzystywanie tabel do tworzenia pasków narzędzi	388
Podsumowanie	397
Rozdział 16. Moduły klas	399
Tworzenie własnych obiektów	400
Procedury właściwości	401
Tworzenie kolekcji	403
Kolekcja modułu klasy	405
Hermetyzacja	407
Przechwytywanie zdarzeń aplikacji	407
Zdarzenia zagnieżdżonego wykresu	410
Kolekcja formantów obiektu UserForm	412
Wskazywanie klas między projektami	414
Podsumowanie	415
Rozdział 17. Dodatki	417
Ukrywanie kodu	418
Tworzenie dodatku	419
Zamykanie dodatków	419
Zmiany kodu	420
Zapisywanie zmian	421
Zmiany interfejsu	421
Instalacja dodatków	423
Zdarzenie AddinInstall	425
Usuwanie dodatku z listy dodatków	426
Podsumowanie	426
Rozdział 18. Dodatki automatyzacji i dodatki COM	429
Dodatki automatyzacji	429
Prosty dodatek — Sekwencja	430
Rejestrowanie dodatków automatyzacji w Excelu	431
Używanie dodatków automatyzacji	432
Wprowadzenie do interfejsu IDTExtensibility2	434
Dodatki COM	441
Interfejs IDTExtensibility2 (kontynuacja)	441
Rejestracja dodatku COM z Excelem	441
COM Add-In Designer	442
Podsumowanie	457

Rozdział 19. Współpraca z innymi aplikacjami Office	459
Nawiązywanie połączenia	459
Późne wiązanie	460
Wczesne wiązanie	462
Otwieranie dokumentu w programie Word	464
Dostęp do aktywnego dokumentu programu Word	465
Tworzenie nowego dokumentu Worda	466
Access i ADO	467
Access, Excel i Outlook	469
Lepsze niż korespondencja seryjna	472
Czytelne zmienne dokumentów	477
Podsumowanie	478
Rozdział 20. Dostęp do danych za pomocą ADO	481
Wprowadzenie do SQL (ang. Structured Query Language)	481
Polecenie SELECT	482
Polecenie INSERT	484
Polecenie UPDATE	484
Polecenie DELETE	485
Przegląd ADO	485
Obiekt Connection	487
Obiekt Recordset	491
Obiekt Command	496
Używanie ADO w aplikacjach Microsoft Excel	499
Używanie ADO z Microsoft Access	499
Używanie ADO z Microsoft SQL Server	506
Używanie ADO z niestandardowymi źródłami danych	516
Podsumowanie	521
Rozdział 21. Zarządzanie zewnętrznymi danymi	523
Interfejs użytkownika dla danych zewnętrznych	523
Dane zewnętrzne	524
Połączenia	524
QueryTable i ListObject	526
Tabela QueryTable bazująca na relacyjnej bazie danych	526
Tabela QueryTable związana z ListObject	529
Tabele QueryTable i zapytania parametryzowane	530
Tabela QueryTable bazująca na kwerendzie sieci Web	533
Tabela QueryTable bazująca na pliku tekstowym	536
Tworzenie i używanie plików połączeń	537
Obiekt WorkbookConnection i kolekcja Connections	541
Ustawienia zabezpieczeń danych zewnętrznych	543
Podsumowanie	544
Rozdział 22. Centrum zaufania i bezpieczeństwo dokumentów	545
Centrum zaufania	545
Zaufani wydawcy	546
Zaufane lokalizacje	546
Dodatki	548
Ustawienia formantów ActiveX	549
Ustawienia makr	551

Pasek komunikatów	553
Zawartość zewnętrzna	554
Opcje prywatności	555
Automatyzacja inspekcji dokumentu	557
Metoda RemoveDocumentInformation	557
Kolekcja DocumentInspectors	559
Podsumowanie	561
Rozdział 23. Przeglądanie źródeł danych OLAP za pomocą Excela	563
Analizowanie danych OLAP za pomocą tabel przestawnych	564
Łączenie ze źródłem danych OLAP	564
Przeglądanie źródła danych OLAP	565
MDX związany z tabelami przestawnymi bazującymi na OLAP	567
Podstawy MDX	569
Przeglądanie źródeł danych OLAP bez tabel przestawnych	573
Używanie ADO do zwracania płaskich zbiorów danych	573
Używanie ADO MD do pobierania informacji o schemacie kostki	574
Tworzenie inwentaryzacji wymiarów, hierarchii i poziomów	575
Tworzenie kostek offline	576
Ręczne tworzenie kostek offline	576
Używanie metody CreateCubeFile	577
Tworzenie kostki offline za pomocą ADO MD i VBA	577
Podsumowanie	579
Rozdział 24. Excel i internet	581
Co internet może zrobić dla Ciebie?	582
Używanie internetu do przechowywania skoroszytów	582
Używanie internetu jako źródła danych	583
Otwieranie stron WWW jako skoroszytów	583
Używanie kwerend sieciowych	585
Przetwarzanie stron WWW w celu uzyskania konkretnych informacji	587
Używanie internetu do publikowania wyników	588
Konfiguracja serwera WWW	588
Zapisywanie arkuszy jako stron WWW	589
Tworzenie interaktywnych stron WWW	590
Używanie internetu jako kanału komunikacyjnego	590
Komunikowanie z serwerem WWW	591
Podsumowanie	593
Rozdział 25. Problemy lokalizacyjne	595
Zmienianie ustawień regionalnych Windows i języka interfejsu użytkownika Office 2007	596
Ustawienia regionalne i wersja językowa Windows	596
Identyfikacja ustawień regionalnych użytkownika i wersji językowej Windows	597
Funkcje konwersji VBA z perspektywy międzynarodowej	597
Interakcje z Excelem	604
Wysyłanie danych do Excela	604
Czytanie danych z Excela	607
Reguły pracy z Excelem	608
Interakcja z użytkownikami	608
Rozmiary papieru	609
Wyświetlanie danych	609
Interpretacja danych	609

Właściwości xxxLocal	610
Reguły pracy z użytkownikami	610
Opcje międzynarodowe Excela 2007	612
Funkcje niezgodne z regułami	614
Funkcja OpenText	614
Funkcja SaveAs	616
Procedura ShowDataForm	616
Wklejanie tekstu	617
Pola obliczeniowe i elementy tabel przestawnych, format warunkowy i formuły sprawdzania danych	617
Kwerendy sieciowe	618
Funkcja arkusza =TEKST()	619
Właściwości Range.Value, Range.Formula i Range.FormulaArray	619
Metoda Range.AutoFilter	619
Metoda Range.AdvancedFilter	620
Funkcje Application.Evaluate, Application.ConvertFormula i Application.ExecuteExcel4Macro	621
Reakcja na ustawienia językowe Office 2007	621
Skąd ten tekst pochodzi?	621
Rozpoznawanie ustawień języka interfejsu użytkownika Office	623
Tworzenie aplikacji wielojęzycznej	623
Praca w środowisku wielojęzycznym	626
Reguły wytwarzania wielojęzycznej aplikacji	627
Kilka pomocnych funkcji	627
Funkcja bWinDoLiczby	627
Funkcje bWinDoDaty	628
Funkcja sFormatujDate	629
Funkcja ZastapRezerwacje	630
Podsumowanie	630
Rozdział 26. Programowanie VBE	633
Identyfikacja obiektów VBE w kodzie	634
Obiekt VBE	634
Obiekt VBProject	634
Obiekt VBComponent	635
Obiekt CodeModule	636
Obiekt CodePane	636
Obiekt Designer	637
Rozpoczynamy	637
Dodawanie elementów menu do VBE	638
Praca ze skoroszytami	642
Praca z kodem	651
Praca z formularzami UserForm	656
Praca z referencjami	661
Dodatki COM	662
Podsumowanie	663
Rozdział 27. Programowanie z wykorzystaniem Windows API	665
Budowa wywołania API	666
Interpretacja deklaracji w stylu C	667
Stałe, struktury, uchwyty i klasy	670
Co zrobić, jeśli coś się nie uda?	673

Umieszczanie wywołań API w modułach klas	675
Kilka przykładowych klas	680
Czasomierz wysokiej częstotliwości	680
Moduł klasy CCzasomierz	681
Zamrożenie formularza	682
Informacje o systemie	684
Modyfikacje stylów formularza	687
Style okien	687
Klasa CZmianaFormularza	689
Formularze o zmiennych rozmiarach	690
Zmiany bezwzględne	691
Zmiany względne	692
Klasa CRozmiaryFormularza	693
Podsumowanie	699

Dodatek A Model obiektowy programu Excel 2007: Ogólnie dostępne właściwości i metody701

Właściwości wspólne dla kolekcji oraz związanych z nimi obiektów	701
Wspólne właściwości kolekcji	701
Wspólne metody kolekcji	702
Wspólne właściwości obiektów	702
Obiekty aplikacji Excel, ich właściwości, metody i zdarzenia	702
Obiekt AboveAverage	703
Obiekt Action i kolekcja Actions	704
Obiekt AddIn i kolekcja AddIns	705
Obiekt Adjustments	706
Obiekt AllowEditRange i kolekcja AllowEditRanges	707
Obiekt Application	709
Kolekcja Areas	730
Obiekt AutoCorrect	731
Obiekt AutoFilter	733
Obiekt AutoRecover	735
Obiekt Axis i kolekcja Axes	736
Obiekt AxisTitle	740
Obiekt Border i kolekcja Borders	742
Kolekcja CalculatedFields	744
Kolekcja CalculatedItems	744
Obiekt CalculatedMember i kolekcja CalculatedMembers	744
Obiekt CalloutFormat	746
Obiekt CellFormat	748
Obiekt Characters	751
Obiekt Chart i kolekcja Charts	752
Obiekt ChartArea	762
Obiekt ChartColorFormat	764
Obiekt ChartFillFormat	764
Obiekt ChartFormat	766
Obiekt ChartGroup i kolekcja ChartGroups	767
Obiekt ChartObject i kolekcja ChartObjects	770
Obiekt ChartTitle	775
Obiekt ChartView	776
Obiekt ColorFormat	777
Obiekt ColorScale	778

Obiekt ColorScaleCriterion i kolekcja ColorScaleCriteria	779
Obiekt ColorStop i kolekcja ColorStops	780
Obiekt Comment i kolekcja Comments	781
Obiekt ConditionValue	783
Obiekt Connections	783
Obiekt ConnectorFormat	784
Obiekt ControlFormat	786
Obiekt CubeField i kolekcja CubeFields	789
Obiekt CustomProperty i kolekcja CustomProperties	791
Obiekt CustomView i kolekcja CustomViews	794
Obiekt Databar	795
Obiekt DataLabel i kolekcja DataLabels	797
Obiekt DataTable	801
Obiekt DefaultWebOptions	802
Obiekt Dialog i kolekcja Dialogs	804
Obiekt DisplayUnitLabel	805
Obiekt DownBars	806
Obiekt DropLines	807
Obiekt Error i kolekcja Errors	808
Obiekt ErrorBars	809
Kolekcja ErrorCheckingOptions	810
Obiekt FillFormat	811
Obiekt Filter i kolekcja Filters	814
Obiekt Floor	815
Obiekt Font	816
Obiekt FormatColor	817
Obiekt FormatCondition i kolekcja FormatConditions	818
Obiekt FreeformBuilder	821
Obiekt Graphic	822
Obiekt Gridlines	824
Kolekcja GroupShapes	825
Obiekt HeaderFooter	825
Obiekt HiLoLines	826
Obiekt HPageBreak i kolekcja HPageBreaks	826
Obiekt Hyperlink i kolekcja Hyperlinks	827
Obiekt Icon	829
Obiekt IconCriterion i kolekcja IconCriteria	830
Obiekt IconSet i kolekcja IconSets	830
Obiekt IconSetCondition	831
Obiekt Interior	833
Obiekt IRtdServer	835
Obiekt IRTDUpdateEvent	836
Obiekt LeaderLines	836
Obiekt Legend	837
Obiekt LegendEntry i kolekcja LegendEntries	839
Obiekt LegendKey	840
Obiekt LinearGradient	842
Obiekt LineFormat	842
Obiekt LinkFormat	844
Obiekt ListColumn i kolekcja ListColumns	845
Obiekt ListDataFormat	846

Obiekt ListObject i kolekcja ListObjects	847
Obiekt ListRow i kolekcja ListRows	850
Obiekt Mailer	850
Obiekt MultiThreadedCalculation	851
Obiekt Name i kolekcja Names	852
Obiekt ODBCConnection	854
Obiekt ODBCError i kolekcja ODBCErrors	856
Obiekt OLEDBConnection	857
Obiekt OLEDBError i kolekcja OLEDBErrors	859
Obiekt OLEFormat	860
Obiekt OLEObject i kolekcja OLEObjects	861
Obiekt Outline	866
Obiekt Page i kolekcja Pages	867
Obiekt PageSetup	868
Obiekt Pane i kolekcja Panes	871
Obiekt Parameter i kolekcja Parameters	873
Obiekt Phonetic i kolekcja Phonetics	874
Obiekt PictureFormat	876
Obiekt PivotAxis	877
Obiekt PivotCache i kolekcja PivotCaches	877
Obiekt PivotCell	881
Obiekt PivotField oraz kolekcje PivotFields i CalculatedFields	882
Obiekt PivotFilter i kolekcja PivotFilters	888
Obiekt PivotFormula i kolekcja PivotFormulas	890
Obiekt PivotItem oraz kolekcje PivotItems i CalculatedItems	891
Kolekcja PivotItemList	892
Obiekt PivotLayout	893
Obiekt PivotLine oraz kolekcje PivotLines i PivotLinesCells	894
Obiekt PivotTable i kolekcja PivotTables	894
Obiekt PlotArea	904
Obiekt Point i kolekcja Points	907
Obiekt Protection	909
Obiekt PublishObject i kolekcja PublishObjects	911
Obiekt QueryTable i kolekcja QueryTables	913
Obiekt Range i kolekcja Ranges	919
Obiekt RecentFile i kolekcja RecentFiles	934
Obiekt RectangularGradient	935
Obiekt RoutingSlip	936
Obiekt RTD	937
Obiekt Scenario i kolekcja Scenarios	937
Obiekt Series i kolekcja SeriesCollection	939
Obiekt SeriesLines	943
Kolekcja ServerViewableItems	944
Obiekt ShadowFormat	945
Obiekt Shape i kolekcja Shapes	947
Obiekt ShapeNode i kolekcja ShapeNodes	952
Kolekcja ShapeRange	954
Kolekcja Sheets	958
Obiekt SheetViews	959
Obiekt SmartTag i kolekcja SmartTags	960
Obiekt SmartTagAction i kolekcja SmartTagActions	961

Kolekcja SmartTagOptions	962
Obiekt SmartTagRecognizer i kolekcja SmartTagRecognizers	962
Obiekt Sort	963
Obiekt SortField i kolekcja SortFields	963
Obiekt SoundNote	965
Obiekt Speech	965
Kolekcja SpellingOptions	966
Obiekt Style i kolekcja Styles	968
Obiekt Tab	970
Obiekt TableStyle i kolekcja TableStyles	971
Obiekt TableStyleElement i kolekcja TableStyleElements	973
Obiekt TextEffectFormat	974
Obiekt TextFrame	975
Obiekt TextFrame2	976
Obiekt ThreeDFormat	978
Obiekt TickLabels	980
Obiekt Top10	982
Obiekt TreeviewControl	984
Obiekt Trendline i kolekcja Trendlines	984
Obiekt UniqueValues	986
Obiekt UpBars	988
Kolekcja UsedObjects	989
Obiekt UserAccess	989
Kolekcja UserAccessList	989
Obiekt Validation	990
Obiekt VPageBreak i kolekcja VPageBreaks	993
Obiekt Walls	994
Obiekt Watch i kolekcja Watches	995
Obiekt WebOptions	996
Obiekt Window i kolekcja Windows	998
Obiekt Workbook i kolekcja Workbooks	1003
Obiekt WorkbookConnection	1020
Obiekt Worksheet i kolekcja Worksheets	1020
Obiekt WorksheetFunction	1030
Obiekt WorksheetView	1033
Obiekt XmlDataBinding	1034
Obiekt XmlMap i kolekcja XmlMaps	1034
Obiekt XmlNamespace i kolekcja XmlNamespaces	1036
Obiekt XmlSchema i kolekcja XmlSchemas	1037
Obiekt XPath	1037
Dodatek B VBE Object Model	1039
Powiązania pomiędzy modelami obiektowymi Excel i VBE	1039
Wspólne właściwości i metody	1040
Obiekt AddIn i kolekcja Add-Ins	1041
Obiekt CodeModule	1043
Obiekt CodePane i kolekcja CodePanels	1047
Obiekt CommandBarEvents	1049
Obiekt Events	1051
Kolekcja LinkedWindows	1051
Obiekt Property oraz kolekcja Properties	1052

Obiekt Reference oraz kolekcja References	1053
Obiekt ReferencesEvents	1055
Obiekt VBComponent oraz kolekcja VBComponents	1056
Obiekt VBE	1059
Obiekt VBProject oraz kolekcja VBProjects	1060
Wspólne właściwości VBProject	1060
Obiekt Window oraz kolekcja Windows	1062
Wspólne właściwości Window	1063
Dodatek C Office 2007 Object Model	1065
Wspólne właściwości z kolekcjami i skojarzonymi obiektami	1065
Wspólne właściwości Collection	1065
Wspólne właściwości Object	1066
Obiekty Office oraz ich właściwości i zdarzenia	1066
BulletFormat2	1066
Obiekt COMAddIn oraz kolekcja COMAddIns	1067
Obiekt CommandBar oraz kolekcja CommandBars	1069
Obiekt CommandBarButton	1074
Obiekt CommandBarComboBox	1077
Obiekt CommandBarControl oraz kolekcja CommandBarControls	1081
Obiekt CommandBarPopup	1085
Obiekt CustomTaskPane	1088
Obiekt CustomXMLNode oraz kolekcja CustomXMLNodes	1089
Obiekt CustomXMLPart oraz kolekcja CustomXMLParts	1092
Obiekt CustomXMLPrefixMapping oraz kolekcja CustomXMLPrefixMappings	1095
Obiekt CustomXMLSchema oraz kolekcja CustomXMLSchemaCollection	1096
Obiekt CustomXMLValidationError oraz kolekcja CustomXMLValidationErrors	1097
Obiekt DocumentInspector oraz kolekcja DocumentInspectors	1098
Obiekt DocumentLibraryVersion oraz kolekcja DocumentLibraryVersions	1100
Obiekt DocumentProperty oraz kolekcja DocumentProperties	1101
Obiekt EncryptionProvider	1104
Obiekt FileDialog	1105
Obiekt FileDialogFilter oraz kolekcja FileDialogFilters	1107
Kolekcja FileDialogSelectedItems	1108
Obiekt FileTypes	1108
Obiekt Font2	1109
Obiekt GlowFormat	1111
Obiekt GradientStop oraz kolekcja GradientStops	1111
Obiekt IAssistance	1112
Obiekty IBlogExtensibility oraz IBlogPictureExtensibility	1113
Obiekt ICTPFactory	1115
Obiekt ICustomTaskPaneConsumer	1115
Obiekt IDocumentInspector	1116
Obiekt IRibbonControl	1116
Obiekt IRibbonExtensibility	1117
Obiekt IRibbonUI	1117
Obiekt LanguageSetting	1118
Obiekt MetaProperty oraz kolekcja MetaProperties	1118
Obiekt MsoEnvelope	1120
Obiekt NewFile	1121
Obiekt ODSOColumn oraz kolekcja ODSOColumns	1121

Obiekt ODSOFilter oraz kolekcja ODSOFilters	1122
Obiekt OfficeDataSourceObject	1122
Obiekt OfficeTheme	1122
Obiekt ParagraphFormat2	1122
Obiekt Permission	1124
Obiekt PolicyItem oraz kolekcja ServerPolicy	1125
Obiekt ReflectionFormat	1126
Obiekt Ruler2	1127
Obiekt RulerLevel2 oraz kolekcja RulerLevels2	1127
Obiekt ScopeFolder oraz kolekcja ScopeFolders	1128
Kolekcja SearchFolders	1129
Obiekt SearchScope oraz kolekcja SearchScopes	1129
Obiekt SharedWorkspace	1130
Obiekt SharedWorkspaceFile oraz kolekcja SharedWorkspaceFiles	1131
Obiekt SharedWorkspaceFolder oraz kolekcja SharedWorkspaceFolders	1133
Obiekt SharedWorkspaceLink oraz kolekcja SharedWorkspaceLinks	1134
Obiekt SharedWorkspaceMember oraz kolekcja SharedWorkspaceMembers	1135
Obiekt SharedWorkspaceTask oraz kolekcja SharedWorkspaceTasks	1136
Obiekt Signature oraz kolekcja SignatureSet	1138
Obiekt SignatureInfo	1140
Obiekt SignatureProvider	1141
Obiekt SignatureSetup	1142
SmartDocument	1143
Obiekt SoftEdgeFormat	1144
Obiekt Sync	1144
Obiekt TabStop2 oraz kolekcja TabStops2	1145
Obiekt TextColumn2 oraz kolekcja TextColumns2	1146
Obiekt TextRange2	1147
Obiekt ThemeColor	1149
Obiekt ThemeColorsScheme	1149
Obiekt ThemeEffectScheme	1150
Obiekt ThemeFont oraz kolekcja ThemeFonts	1150
Obiekt ThemeFontScheme	1151
UserPermission	1152
Obiekt WebPageFont oraz kolekcja WebPageFonts	1152
Obiekt WorkflowTask oraz kolekcja WorkflowTasks	1154
Obiekt WorkflowTemplate oraz kolekcja WorkflowTemplates	1155

Skorowidz	1157
------------------------	-------------

2

Obiekt Application

Ten rozdział przedstawia zakres funkcjonalności programu Excel. Opisane w nim funkcjonalności nie muszą być konieczne powiązane z sobą. Generalnie model obiektowy Excela zawiera obiekty przeznaczone do wykonywania różnych zadań. Obiekt `Application` znajduje się na szczycie hierarchii modelu obiektowego Excela i zawiera wszystkie inne obiekty tego programu. Skupia także wszystkie właściwości i metody, które nie podlegają żadnym innym obiektom, ale są niezbędne do kontroli programistycznej aplikacji Excel. Na przykład właściwości `Application` kontrolują uaktualnienia ekranu i wyświetlanie wiadomości alarmowych, a metoda `Application` oblicza formuły we wszystkich otwartych skoroszytach.

Metody i właściwości globalne

Większość metod i właściwości obiektu `Application` należy także do grupy elementów globalnych `<globals>`, która znajduje się na górze listy klas w narzędziu *Object Browser*, poazanym na rysunku 2.1.

Jeśli właściwość lub metoda są globalne, można wskazać tę właściwość lub metodę bez poprzedzania jej referencją do obiektu. Na przykład następujące dwa odwołania są równoważne:

```
Application.ActiveCell  
ActiveCell
```

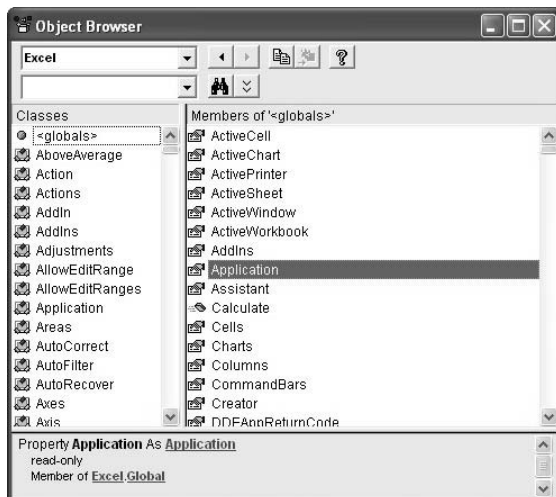
Jednak trzeba zachować ostrożność. Łatwo założyć, że często używane właściwości `Application`, takie jak `ScreenUpdating`, należą do grupy `<globals>`, podczas gdy tak nie jest. Następujący kod jest poprawny:

```
Application.ScreenUpdating = False
```

Natomiast poniższy kod może dawać nieoczekiwane wyniki:

```
ScreenUpdating = False
```

Rysunek 2.1



Tworzy on nową zmienną i przypisuje do niej wartość False. Można łatwo uniknąć tego błędu, wstawiając linię kodu `Option Explicit` na górze każdego modułu. Takie odwołania zostaną oznaczone jako niezdefiniowane zmienne, gdy kod będzie kompilowany.

Pamiętajmy, że można spowodować automatyczne wstawianie `Option Explicit` w nowych modułach, jeśli użyjemy *Tools*, a potem *Options* w oknie VBE, a w zakładce *Editor* zaznaczymy pole wyboru *Require Variable Declaration*.

Aktywne właściwości

Obiekt `Application` dostarcza wielu skrótów, które pozwalają wskazać aktywne obiekty bez jawnego ich nazywania. Dzięki temu podczas działania makra można odkryć, które obiekty są aktywne. Łatwiejsze jest także pisanie uogólnionego kodu, który może być zastosowany do obiektów tego samego typu z różnymi nazwami.

Następujące właściwości obiektu `Application` są globalnymi właściwościami, które pozwalają wskazywać aktywne obiekty:

- `ActiveCell`,
- `ActiveChart`,
- `ActivePrinter`,
- `ActiveSheet`,
- `ActiveWindow`,
- `ActiveWorkbook`,
- `Selection`.

Jeśli właśnie utworzyliśmy nowy skoroszyt i chcemy zapisać go z konkretną nazwą pliku, użyjemy właściwości `ActiveWorkbook`, aby zwrócić wskaźnik do nowego obiektu `Workbook`:

```
Workbooks.Add
ActiveWorkbook.SaveAs Filename:="C:\Dane.xls"
```

Aby napisać makro, które stosuje pogrubienie do aktualnie zaznaczonych komórek, możemy użyć właściwości `Selection`. Zwrócony zostanie wówczas wskaźnik do obiektu typu `Range` zawierający zaznaczone komórki:

```
Selection.Font.Bold = True
```

Należy uważać, ponieważ `Selection` nie odnosi się do obiektu `Range`, jeśli na przykład inny typ obiektu, taki jak `Shape`, jest aktualnie zaznaczony lub gdy aktywny arkusz nie jest arkuszem roboczym. Warto wyposażyć makro w funkcję sprawdzania, aby mieć pewność, że arkusz kalkulacyjny jest zaznaczony, zanim rozpocznie się wprowadzanie do niego danych.

```
If TypeName(ActiveSheet) <> "Worksheet" Or _
    TypeName(Selection) <> "Range" Then
    MsgBox "Można uruchamiać to makro tylko na obiektach typu range", vbCritical
    Exit Sub
End If
```

Wyświetlanie ostrzeżeń

Odpowiadanie na ostrzeżenia systemu podczas działania makra może być denerwujące. Jeśli na przykład makro kasuje arkusz, pojawia się komunikat ostrzegawczy i trzeba kliknąć przycisk *OK*, aby kontynuować. Jednak użytkownik może także kliknąć przycisk *Anuluj*, przerywając w ten sposób operację usuwania. Może to mieć wpływ na dalsze wykonywanie kodu, w którym zakłada się, że operacja usuwania została wykonana.

Można zlikwidować większość ostrzeżeń, ustawiając właściwość `DisplayAlerts` na `False`. Gdy kusuniemy okno dialogowe, akcja, która jest powiązana z domyślnym przyciskiem tego ona, jest automatycznie wykonywana, tak jak w poniższym fragmencie kodu:

```
Application.DisplayAlerts = False
ActiveSheet.Delete
Application.DisplayAlerts = True
```

Nie trzeba koniecznie przywracać `DisplayAlerts` wartości `True` na końcu makra, ponieważ VBA robi to automatycznie. Jednak zwykle dobrze jest po wyłączeniu konkretnego komunikatu przywrócić alarmy tak, żeby nieoczekiwane ostrzeżenia pojawiały się na ekranie.

`DisplayAlerts` jest zazwyczaj używane do wyłączania ostrzeżeń o tym, że za chwilę nastąpi nadpisanie istniejącego pliku podczas operacji *Zapisz jako*. Gdy wyłączymy to ostrzeżenie, podejmowana jest domyślna akcja, tzn. plik jest nadpisywany bez przerywania działania makra.

Aktualizowanie ekranu

Oglądanie zmian ekranu i migotania podczas działania makra może być denerwujące. Dzieje się tak w przypadku makr, które zaznaczają lub aktywują obiekty, i jest typowe dla kodu generowanego przez narzędzie nagrywania makr.

Lepiej pominąć zaznaczanie obiektów w VBA. Rzadko jest to konieczne, a kod będzie działał szybciej, jeżeli zrezygnujemy z zaznaczania lub aktywowania obiektów. Większość kodu w tej książce unika zaznaczania, jeśli jest to możliwe.

Aby zamrozić ekran na czas działania makra, należy użyć następującej linii kodu:

```
Application.ScreenUpdating = False
```

Ekran pozostanie zamrożony aż do czasu przypisania tej właściwości wartości True lub zakończenia działania przez makro i powrotu kontroli do interfejsu użytkownika. Nie ma potrzeby przywracania ScreenUpdating wartości True, jeśli nie chcemy wyświetlać zmian ekranu podczas dalszego działania makra.

Istnieje jedna sytuacja, gdy warto ustawić ScreenUpdating na True, podczas gdy makro działa. Jeśli podczas działania makra wyświetla się formularz użytkownika lub wbudowane okno dialogowe, użytkownik powinien się upewnić, że uaktualnienie ekranu jest włączone, przed pokazaniem obiektu. Jeśli uaktualnienie ekranu jest wyłączone i użytkownik przeciąga formularz po ekranie, formularz ten działa jak gumka dla zawartości ekranu w tle. Można wyłączyć uaktualnianie ekranu ponownie po pokazaniu obiektu.

Pozytywnym efektem wyłączenia uaktualnienia ekranu jest to, że kod działa szybciej. Może to nawet przyspieszyć kod, który unika zaznaczania obiektów, gdzie tylko niewielkie uaktualnienia ekranu są wymagane. Kod działa z maksymalną prędkością, gdy zaznaczanie jest dokonywane sporadycznie, a uaktualnianie ekranu zostanie wyłączone.

Metoda Evaluate

Metoda Evaluate może być używana do obliczania formuł arkusza i generowania odnośników do obiektu Range. Zwykłą składnią dla metody Evaluate jest następująca:

```
Evaluate("Expression")
```

Można także użyć formatu skróconego, gdzie omija się cudzysłowy i umieszcza kwadratowe nawiasy wokół wyrażenia, jak w następującym przykładzie:

```
[Expression]
```

Expression może być albo dowolnym prawidłowym wyrażeniem arkusza (ze znakiem równości po lewej lub bez niego), albo wskazaniem na zakres komórek. Obliczenia arkusza mogą zawierać funkcje arkusza, które nie są dostępne dla VBA przez obiekt WorksheetFunction, mogą też

być formułami tablic arkusza. Więcej informacji na temat obiektu `WorksheetFunction` znajduje się dalej w tym rozdziale.

Na przykład funkcja `ISBLANK`, której można używać w formułach arkusza, nie jest dostępna dla VBA przez obiekt `WorksheetFunction`, ponieważ równoważna funkcja VBA `IsEmpty` dostarcza tej samej funkcjonalności. Oczywiście można użyć `ISBLANK`, jeśli się jej potrzebuje. Następujące dwa przykłady są równoważne i zwracają `True`, jeśli `A1` jest puste, lub `False`, jeśli `A1` nie jest puste:

```
MsgBox Evaluate("=ISBLANK(A1)")
MsgBox [ISBLANK(A1)]
```

Zaletą pierwszej techniki jest to, że możemy generować wartość łańcucha, używając kodu, co czyni go bardziej elastycznym. Druga technika jest krótsza, ale możemy zmienić wyrażenie, tylko edytując swój kod. Następujące procedury wyświetlają `True` lub `False`, aby wskazać, czy aktywna komórka jest pusta, i ilustrują elastyczność pierwszej techniki:

Listing 2.1

```
Sub CzyAktywnaKomorkaJestPusta()
    Dim sNazwaFunkcji As String, sWskaznikKomorki As String
    sNazwaFunkcji = "ISBLANK"
    sWskaznikKomorki = ActiveCell.Address
    MsgBox Evaluate(sNazwaFunkcji & "(" & sWskaznikKomorki & ")")
End Sub
```

Należy zauważyć, że za pomocą drugiej techniki nie można wyliczać wyrażenia zawierającego zmienne.

Następujące dwie linie kodu pokazują dwa sposoby, jak można użyć `Evaluate`, aby wygenerować wskaźnik do obiektu `Range` i przypisać wartość do tego obiektu:

```
Evaluate("A1").Value = 10
[A1].Value = 10
```

Pierwsze wyrażenie jest niepraktyczne i rzadko używane. Drugie to wygodny sposób wskazywania obiektu `Range`, chociaż niezbyt elastyczny. Można skrócić te wyrażenia, omijając właściwość `Value`, ponieważ jest to domyślna właściwość obiektu `Range`:

```
[A1] = 10
```

Bardziej interesującym zastosowaniem `Evaluate` jest zwracanie zawartości kolekcji `Names` skroszty i wydajne generowanie tablic wartości. Poniższy kod tworzy ukrytą nazwę do przechowywania hasła. Ukryte nazwy nie mogą być widoczne w oknie dialogowym *Menedżer nazw* dostępnym z zakładki *Formuły* Wstążki, więc stanowią wygodny sposób przechowywania informacji w skoroszycie, nie powodując bałaganu w interfejsie użytkownika.

```
Names.Add Name:="Hasło", RefersTo:="Bazonkas", Visible:=False
```

Można następnie używać ukrytych danych w wyrażeniach podobnych do następujących:

```
sDaneUzytkownika = InputBox("Podaj hasło")
If sDaneUzytkownika = [Hasło] Then
    ...
```

Wykorzystywanie nazw do przechowywania danych jest opisane bardziej szczegółowo w rozdziale 5.

Metoda `Evaluate` może być także używana z tablicami. Kolejne wyrażenie generuje dwuwymiarową tablicę typu `Variant`, zawierającą 100 wierszy i jedną kolumnę, z wartościami od 101 do 200. Ten proces jest wykonywany wydajniej niż za pomocą pętli `For...Next`:

```
vTablicaWierszy = [ROW(101:200)]
```

I jeszcze jeden kod, który przypisuje wartości od 101 do 200 do zakresu `B1:B100` i ponownie robi to wydajniej niż pętla `For...Next`:

```
[B1:B100] = [ROW(101:200)]
```

Funkcja InputBox

Funkcja `InputBox` dostarcza łatwego sposobu pytania o dane wejściowe. Istnieje też metoda `InputBox` obiektu `Application`, która tworzy bardzo podobne okno dialogowe pobierające dane, ale jest wydajniejsza. Pozwala na kontrolę typu danych, które mają być podane przez użytkownika, i umożliwia wykrycie, kiedy kliknięty jest przycisk *Anuluj*.

Jeśli odwołanie do `InputBox` jest niekwalifikowane, co pokazano niżej, używa się funkcji VBA `InputBox`:

```
sOdpowiedz = InputBox(prompt:="Wprowadź zakres")
```

Użytkownik może jedynie wpisywać dane do okna dialogowego. Nie jest możliwe wskazanie komórki za pomocą myszy. Zwracana wartość z funkcji `InputBox` jest zawsze typu `String`. Nie jest przeprowadzane sprawdzanie, co ten łańcuch znaków zawiera. Jeśli użytkownik nic nie wpisze, zwracany jest łańcuch o długości zerowej. Jeśli użytkownik kliknie przycisk *Anuluj*, także jest zwracany łańcuch o wartości zerowej. Kod nie rozróżnia między brakiem wpisu i wynikiem kliknięcia *Anuluj*.

W następnym przykładzie użyto metody `InputBox` obiektu `Application` do pytania o zakres:

```
vOdpowiedz = Application.InputBox(Prompt:="Wprowadź zakres", Type:=8)
```

Parametr `Type` może przyjmować różne wartości lub sumę tych wartości, jeśli stosowanych jest wiele typów (tabela 2.1).

Użytkownik może wskazywać komórki za pomocą myszy lub wpisywać dane. Jeśli wprowadzone dane są błędnego typu, metoda `InputBox` wyświetla komunikat o błędzie i prosi ponownie o dane. Jeśli użytkownik kliknie przycisk *Anuluj*, metoda `InputBox` zwróci wartość `False`.

Jeśli przypisze się zwracaną wartość do zmiennej typu `Variant`, dla większości zwracanych typów można sprawdzić, czy wartość jest równa `False`, aby wykryć, czy naciśnięto przycisk *Anuluj*. Jeśli jest się proszonym o zakres, sytuacja nie jest tak prosta, trzeba użyć kodu podobnego do przedstawionego w listingu 2.2.

Tabela 2.1

Wartość typu	Znaczenie
0	Formuła
1	Liczba
2	Tekst (String)
4	Wartość logiczna (True lub False)
8	Wskaźnik do komórki, jako obiekt Range
16	Wartość błędu, taka jak #N/A
64	Tablica wartości

Listing 2.2

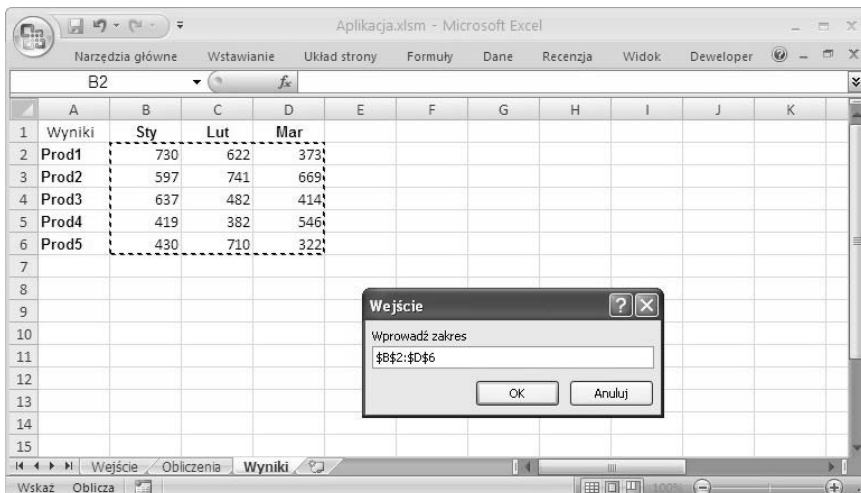
```

Sub PobierzZakres()
    Dim rng As Range

    On Error Resume Next
    Set rng = Application.InputBox(Prompt:="Wprowadź zakres:", Type:=8)
    If rng Is Nothing Then
        MsgBox "Operacja anulowana"
    Else
        rng.Select
    End If
End Sub

```

Gdy uruchomi się ten kod, należy użyć myszy do zaznaczenia zakresu. Efekt tej akcji powinien wyglądać podobnie jak na rysunku 2.2.



Rysunek 2.2

Problem polega na tym, że najpierw trzeba użyć polecenia Set do przypisania obiektu typu range do zmiennej obiektowej. Jeśli użytkownik kliknie *Anuluj* i zwrócona jest wartość False, polecenie Set się nie powiedzie i zostanie wygenerowany błąd czasu wykonania. Używając polecenia On Error Resume Next, unikniemy błędu czasu wykonania i możemy sprawdzić, czy został wygenerowany prawidłowy zakres. Wbudowane sprawdzanie typów metody InputBox zapewnia, że prawidłowy zakres został zwrócony, jeśli użytkownik kliknął *OK*, a zatem pusty zakres wskazuje, że kliknięto *Anuluj*.

Właściwość StatusBar

Właściwość StatusBar pozwala wyświetlać łańcuch tekstu po lewej stronie paska stanu na dole ekranu. Dzięki temu użytkownicy mogą być informowani o postępie długich operacji makra. Jest to dobry pomysł, szczególnie gdy uaktualnianie ekranu jest wyłączone, a na ekranie nie ma żadnego znaku aktywności. Nawet jeśli wyłączyliśmy uaktualnienia ekranu, możemy nadal wyświetlać komunikaty na pasku stanu.

Następujący kod pokazuje, jak można użyć tej techniki w procedurze pętli:

Listing 2.3

```
Sub PokazKomunikat()  
    Dim lLicznik As Long  
    For lLicznik = 0 To 100000000  
        If lLicznik Mod 1000000 = 0 Then  
            Application.StatusBar = "Przetwarzanie rekordu " & lLicznik  
        End If  
    Next lLicznik  
    Application.StatusBar = False  
End Sub
```

Na koniec przetwarzania trzeba przypisać właściwości StatusBar wartość False, aby powrócić do normalnego działania. W przeciwnym razie na ekranie pozostanie ostatni komunikat.

Funkcja SendKeys

SendKeys pozwala na wysyłanie wciśnięć klawiatury do aktualnie aktywnego okna. Opcja ta jest używana do kontrolowania aplikacji, które nie wspierają żadnych innych form komunikacji, takich jak DDE (ang. *Dynamic Data Exchange*) czy OLE, i zasadniczo uważana jest za technikę ostatniej szansy.

Kod z następującego przykładu otwiera aplikację Notatnik, która nie wspiera DDE ani OLE, i zapisuje linie danych do dokumentu Notatnika.

Listing 2.4

```

Sub SleKlawisze()
    Dim dWartoscZwracana As Double
    dWartoscZwracana = Shell("NOTEPAD.EXE", vbNormalFocus)
    AppActivate dWartoscZwracana
    Application.SendKeys "Copy Dane.xlsx c:\", True
    Application.SendKeys "~", True
    Application.SendKeys "%PABATCH%Z", True
End Sub

```

Ten przykład może nie działać prawidłowo z VBA. Należy go uruchomić z okna programu Excel.

SleKlawisze używa *Alt+P+A* do wykonania polecenia *Zapisz jako* z menu *Plik*. Wpisuje nazwę pliku BATCH, a następnie używa *Alt+Z*, aby zapisać plik tekstowy. Symbol procenta (%) reprezentuje *Alt*, a tylda (~) reprezentuje *Enter*. Symbol karetki (^) jest używany do reprezentacji *Ctrl*. Inne klawisze specjalne są określane przez umieszczanie ich nazw w nawiasach klamrowych. Na przykład klawisz *Delete* jest reprezentowany przez {Del}, co pokazano w następnym przykładzie.

Można także wysłać wciśnięte klawisze bezpośrednio do programu Excel. Poniższa procedura czyści okno *Immediate* VBE. Jeśli w oknie *Immediate* wydawane były polecenia lub jeśli do pisania było używane `Debug.Print`, okno *Immediate* może być zabałaganione starymi informacjami. Ta procedura przełącza fokus do okna *Immediate* i wysyła *Ctrl+A*, aby zaznaczyć cały tekst w oknie. Tekst jest kasowany za pomocą *Del*.

Listing 2.5

```

Sub CzyszczenieOknaImmediate()
    Application.VBE.Windows.Item("Immediate").SetFocus
    Application.SendKeys "^a"
    Application.SendKeys "{Del}"
End Sub

```

Aby to makro działało, konieczny jest programowy dostęp do projektu Visual Basic. Można go ustawić z poziomu Wstążki. Należy wybrać zakładkę *Deweloper*, wybrać *Bezpieczeństwo makr* i zaznaczyć pole wyboru obok *Ufaj dostępowi do modelu obiektowego projektu VBA*.

Metoda OnTime

Można użyć metody `OnTime`, aby zaplanować uruchomienie makra w przyszłości. Trzeba określić datę i czas uruchomienia makra i jego nazwę. Jeżeli używa się metody `Wait` dla obiektu `Application` do wstrzymania makra, cała aktywność programu Excel, w tym ręczna interakcja, zostaje wstrzymana. Zaletą `OnTime` jest pozwolenie na powrót do normalnej interakcji

z programem Excel, włączając w to wykonywanie innych makr, podczas oczekiwania na uruchomienie zaplanowanego makra.

Powiedzmy, że mamy otwarty skoroszyt z łączami do pliku *Dane.xlsx*, który istnieje na serwerze sieciowym, ale nie jest aktualnie otwarty. O godzinie 15 chcemy uaktualnić łącza do *Dane.xlsx*. Kod z przykładu uruchamia makro *OdswiezDane* o godzinie 15 bieżącego dnia. *Date* zwraca bieżącą datę, a funkcja *TimeSerial* zawiera wybraną godzinę.

```
Sub UruchomOCzasie()  
    Application.OnTime Date + TimeSerial(15, 0, 0), "OdswiezDane"  
End Sub
```

Warto zauważyć, że jeśli przystąpimy do uruchomienia tego makra po godzinie 15, otrzymamy komunikat o błędzie, ponieważ nie można zaplanować zadania do wykonania w przeszłości. Jeśli to konieczne, należy przesunąć w przyszłość godzinę uruchomienia.

Makro *OdswiezDane* używa metody *UpdateLink*, aby uaktualnić łącza do *Dane.xlsx*, które istnieją w *ThisWorkbook*. *ThisWorkbook* jest wygodnym sposobem wskazywania skoroszytu zawierającego makro:

```
Sub OdswiezDane()  
    ThisWorkbook.UpdateLink Name:="C:\Dane.xlsx", Type:=xlExcelLinks  
End Sub
```

Aby kontynuować regularne odświeżanie danych, można wykonać makro wywoływane w następujący sposób:

```
Dim mdteCzasHarmonogramu As Date  
  
Sub OdswiezDane()  
    ThisWorkbook.UpdateLink Name:="C:\Dane.xlsx", Type:=xlExcelLinks  
    mdteCzasHarmonogramu = Now + TimeSerial(0, 1, 0)  
    Application.OnTime mdteCzasHarmonogramu, "OdswiezDane"  
End Sub  
  
Sub ZatrzymajOdswiezanie()  
    Application.OnTime mdteCzasHarmonogramu, "OdswiezDane", , False  
End Sub
```

Procedura *OdswiezDane* po każdym wywołaniu planuje kolejne uruchamianie po upływie minuty. Aby zatrzymać makro, trzeba znać czas harmonogramu. Do przechowywania ostatniego czasu harmonogramu używana jest zmienna poziomu modułu *mdteCzasHarmonogramu*. Procedura *ZatrzymajOdswiezanie* ustawia czwarty parametr *OnTime* na *False*, aby anulować zaplanowane wykonanie procedury *OdswiezDane*.

Gdy planujemy, że makro będzie uruchamiał się w przyszłości przy użyciu metody *OnTime*, musimy upewnić się, że Excel będzie działał w pamięci do zaplanowanego czasu uruchomienia. Nie jest konieczne, by skoroszyt, który zawiera makro korzystające z metody *OnTime*, pozostawał otwarty. Excel otworzy go, jeśli zaistnieje taka potrzeba.

Metoda *OnTime* jest użyteczna, gdy chcemy wprowadzić opóźnienie w przetwarzaniu makr, aby pozwolić na wystąpienie zdarzenia, które znajduje się poza kontrolą. Na przykład można chcieć wysłać dane do innej aplikacji przez łącza DDE i czekać na odpowiedź od tej aplikacji

przed kontynuowaniem przetwarzania. W tym celu należy utworzyć dwa makra. Pierwsze wysyła dane i planuje, że drugie makro, które przetwarza odpowiedź, zostanie wykonane po upływie odpowiedniego czasu. Drugie makro powinno uruchamiać się wielokrotnie, aż do wykrycia zmiany w arkuszu lub środowisku spowodowanej odpowiedzią z zewnętrznej aplikacji.

Metoda OnKey

Metody OnKey można użyć do przypisania procedury makra do pojedynczego klawisza lub kombinacji *Ctrl*, *Shift*, *Alt* z innym klawiszem. Można jej także użyć do wyłączenia kombinacji klawiszy.

Przykład pokazuje, jak przypisać makro Do1Dziesiec do klawisza *strzałka w dół*. Gdy Przy piszDo1 zostanie uruchomione, klawisz strzałki w dół będzie uruchamiać makro Do1Dziesiec i przenosić wskaźnik komórki o dziesięć wierszy w dół, a nie o jeden.

Listing 2.6

```
Sub PrzypiszDo1()
    Application.OnKey "{Down}", "Do1Dziesiec"
End Sub

Sub Do1Dziesiec()
    ActiveCell.Offset(10, 0).Select
End Sub

Sub WyczyscDo1()
    Application.OnKey "{Down}"
End Sub
```

WyczyscDo1 przywraca normalne działanie klawiszowi *strzałka w dół*.

Metoda OnKey jest stosowana również do wyłączania istniejących skrótów klawiaturowych. Można na przykład wyłączyć skrót *Ctrl+C*, normalnie używany do kopiowania, za pomocą następującego kodu przypisującego procedurę null do tej kombinacji klawiszy:

```
Sub StopSkrotKopiowanie()
    Application.OnKey "^c", ""
End Sub
```

Zwróćmy uwagę, że użyte jest małe *c*. Jeśli użyjemy wielkiego *C*, dotyczyć to będzie kombinacji *Ctrl+Shift+c*. Aby przywrócić normalne działanie *Ctrl+c*, trzeba posłużyć się następującym kodem:

```
Sub CzyscSkrotKopiowanie()
    Application.OnKey "^c"
End Sub
```

Przypisanie klawiszy metodą OnKey dotyczy wszystkich otwartych skoroszytów i działa tylko podczas bieżącej sesji programu Excel.

Funkcje arkusza

Bezpośrednio w kodzie Excel VBA można użyć wbudowanych funkcji pochodzących z dwóch źródeł. Jedną grupę stanowią funkcje będące częścią języka VBA, drugą — funkcji arkusza.

Excel i język Visual Basic, w wariantcie VBA, nie były łączone do czasu Excela 5. Każdy system niezależnie tworzył własne funkcje, co nieuchronnie powodowało konflikty między obiema grupami. Na przykład Excel ma funkcję DATA (DATE) i VBA ma funkcję Date. Funkcja Excela DATA przyjmuje trzy parametry wejściowe (rok, miesiąc i dzień), aby wygenerować datę. Funkcja VBA Date nie przyjmuje parametrów wejściowych i zwraca bieżącą datę z zegara systemowego. Dodatkowo VBA ma funkcję DateSerial, która przyjmuje te same parametry wejściowe i zwraca ten sam wynik, co funkcja DATA programu Excel. W końcu funkcja programu Excel DZIŚ (TODAY) nie przyjmuje parametrów i zwraca ten sam wynik, co funkcja VBA Date.

W zasadzie jeśli funkcja VBA służy temu samemu celowi, co funkcja programu Excel, to funkcja programu Excel nie jest bezpośrednio dostępna dla makr VBA (choć można użyć metody Evaluate, aby skorzystać z dowolnej funkcji Excela, o czym wspomniano już w tym rozdziale). Istnieje także specjalny przypadek dotyczący funkcji programu Excel MOD. Funkcja ta nie jest bezpośrednio dostępna w VBA, ale VBA ma operator Mod, który służy do tych samych celów. Następujący kod używa skróconej metody Evaluate i wyświetla dzień tygodnia jako numer, używając funkcji programu Excel MOD i TODAY:

```
MsgBox [MOD(TODAY(),7)]
```

Ten sam wynik można osiągnąć znacznie prościej dzięki funkcji VBA Date i operatorowi Mod:

```
MsgBox Date Mod 7
```

Funkcja programu Excel ZŁĄCZ.TEKSTY (CONCATENATE) także nie jest dostępna w VBA. Można użyć operatora konkatenacji (&) jako substytutu, tak samo jak w formułach arkusza Excel. Jeśli chcemy koniecznie użyć funkcji CONCATENATE w VBA, możemy napisać kod podobny do poniższego:

```
Sub PrzykladKonkatenacji1()
    Dim s1 As String, s2 As String
    s1 = "Jan "
    s2 = "Kowalski"
    MsgBox Evaluate("CONCATENATE("" & s1 & """, "" & s2 & """)")
End Sub
```

Z drugiej strony możemy uniknąć absurdów i otrzymać ten sam wynik, stosując następujący kod:

```
Sub PrzykladKonkatenacji2()
    Dim s1 As String, s2 As String
    s1 = "Jan "
    s2 = "Kowalski"
    MsgBox s1 & s2
End Sub
```

Funkcje VBA, takie jak Date, DateSerial i IsEmpty, mogą być używane bez kwalifikacji, ponieważ są elementami grupy <globals>. Na przykład można użyć następujących wyrażeń:

```
DataPoczątkowa = DateSerial(1999, 6, 1)
```

Funkcje Excela, takie jak WYSZUKAJ.PIONOWO (VLOOKUP) i SUMA (SUM), stanowią metody obiektu WorksheetFunction i są używane z następującą składnią:

```
Suma = WorksheetFunction.Sum(Range("A1:A10"))
```

W celu zachowania kompatybilności z Excelem 5 i 95 można użyć Application zamiast WorksheetFunction:

```
Suma = Application.Sum(Range("A1:A10"))
```

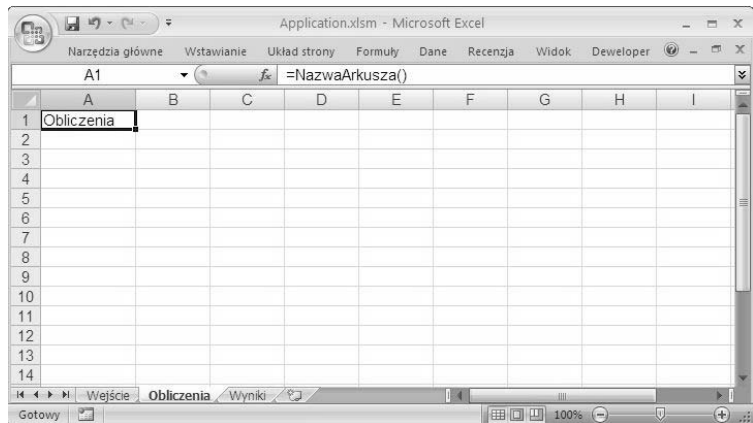
Kompletna lista funkcji arkusza bezpośrednio dostępnych w VBA znajduje się w opisie obiektu WorksheetFunction w dodatku A.

Właściwość Caller

Właściwość Caller obiektu Application zwraca wskaźnik do obiektu, który wywołał lub wykonał procedurę makra. Miała ona szeroki zakres zastosowań w Excelu 5 i 95, gdzie była używana z menu i formantami na arkuszach dialogowych. W Excelu 97 paski poleceń i formanty ActiveX na formularzach użytkowników zastąpiły menu i formanty na arkuszach dialogowych, a teraz Wstążka i pasek narzędzi *Szybki dostęp* wyparły paski poleceń. Właściwość Caller nie dotyczy tych nowych funkcjonalności.

Caller nadal stosuje się do formantów paska narzędzi *Forms*, rysowanych obiektów, do których przyłączono makro, i funkcji zdefiniowanych przez użytkownika. Jest to szczególnie użyteczne w przypadku wyznaczania komórki, która wywołała zdefiniowaną przez użytkownika funkcję. Arkusz z rysunku 2.3 używa funkcji NazwaArkusza, aby wyświetlić nazwę arkusza w A1.

Rysunek 2.3



Wyrażenie Application.Caller, gdy zostanie użyte w funkcji, zwraca wskaźnik do komórki, która wywołała funkcję, jako obiekt Range. W kolejnym przykładzie funkcja NazwaArkusza używa właściwości Parent obiektu Range do generowania wskaźnika do obiektu Worksheet zawierającego obiekt Range. Przypisuje ona właściwość Name obiektu Worksheet do zwracanej wartości funkcji. Metoda Volatile obiektu Application wymusza na programie Excel przeliczenie funkcji za każdym razem, gdy arkusz jest wyliczany, więc jeśli zmieni się nazwę arkusza, nowa nazwa jest wyświetlana przez tę funkcję:

```
Function NazwaArkusza()  
    Application.Volatile  
    NazwaArkusza = Application.Caller.Parent.Name  
End Function
```

Błędem może być użycie następującego kodu w funkcji NazwaArkusza:

```
NazwaArkusza = ActiveSheet.Name
```

Jeśli ponowne wyliczenie odbywa się, gdy aktywny jest inny arkusz niż ten zawierający formułę, błędna nazwa zostanie wpisana do komórki.

Podsumowanie

Ten rozdział omawia niektóre użyteczne właściwości i metody obiektu `Application`. Ponieważ obiekt `Application` jest używany do przechowywania funkcjonalności ogólnego przeznaczenia, które nie podlegają innym obiektom, łatwo jest pominąć niektóre z tych bardzo użytecznych możliwości.

Opisane zostały następujące właściwości i metody:

- `ActiveCell`: zawiera wskaźnik do aktywnej komórki;
- `ActiveChart`: zawiera wskaźnik do aktywnego wykresu;
- `ActivePrinter`: zawiera wskaźnik do aktywnej drukarki;
- `ActiveSheet`: zawiera wskaźnik do aktywnego arkusza;
- `ActiveWindow`: zawiera wskaźnik do aktywnego okna;
- `ActiveWorkbook`: zawiera wskaźnik do aktywnego skoroszytu;
- `Caller`: zawiera wskaźnik do obiektu, który wywołał makro;
- `DisplayAlerts`: określa, czy mają być wyświetlane ostrzegawcze okna dialogowe;
- `Evaluate`: używana do obliczania funkcji Excela i generowania obiektów `Range`;
- `InputBox`: używana do proszenia użytkownika o wprowadzenie danych;
- `OnKey`: przypisuje makro do pojedynczego klawisza lub kombinacji (z *Ctrl*, *Alt* itp.);
- `OnTime`: używana do określania terminu uruchomienia makra;
- `ScreenUpdating`: określa, czy uaktualnianie ekranu jest włączone, czy wyłączone;
- `Selection`: zawiera wskaźnik do zaznaczonego zakresu;
- `SendKeys`: wysyła wciśnięte klawisze do aktywnego okna;
- `StatusBar`: pozwala na wyświetlanie komunikatów na pasku stanu;
- `WorksheetFunction`: zawiera funkcje Excela dostępne w VBA.

Jest to tylko niewielki wybór właściwości i metod obiektu `Application` — w Excelu 2007 istnieje ich ponad 200. Pełna lista znajduje się w dodatku A.