EXCEL® 2019 PL

Programowanie w VBA

Michael Alexander
Dick Kusleika

Helion

Tytuł oryginału: Excel 2019 Power Programming with VBA

Tłumaczenie: Grzegorz Kowalczyk

ISBN: 978-83-283-6634-3

Helion SA
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/e19pvw
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Część III. Praca z formularzami UserForm 403

Dodatki 661

ROZDZIAŁ

11

Praca z danymi zewnętrznymi i plikami

W TYM ROZDZIALE:

- Praca z danymi zewnętrznymi
- Zastosowanie obiektów ActiveX do pobierania danych ze źródeł zewnętrznych
- Najczęściej wykonywane operacje na plikach
- Praca z plikami tekstowymi

Praca z danymi ze źródeł zewnętrznych

Dane zewnętrzne są dokładnie tym, co sugeruje ich nazwa, czyli danymi, które nie są zlokalizowane w skoroszycie Excela, z którym pracujesz. Przykładami zewnętrznych źródeł danych mogą być pliki tekstowe, tabele w bazach danych takich jak Access czy SQL Server lub nawet inne skoroszyty Excela.

Istnieje bardzo wiele sposobów na załadowanie danych zewnętrznych do Excela. Jeżeli weźmiemy pod uwagę możliwości zarówno interfejsu użytkownika, jak i języka VBA, to okaże się, że liczba dostępnych technik importowania danych zewnętrznych do skoroszytów Excela jest zdecydowanie zbyt duża, aby omówić je wszystkie w jednym czy nawet w kilku przeznaczonych jedynie im rozdziałach. Dlatego w tym rozdziale skoncentrujemy się na kilku starannie wybranych technikach, których możesz używać w większości standardowych scenariuszy importu danych i które nie kryją w sobie zbyt wielu pułapek i utrudnień.

Pierwsza z omawianych technik wykorzystuje ciekawy mechanizm Excela o nazwie Power Query.

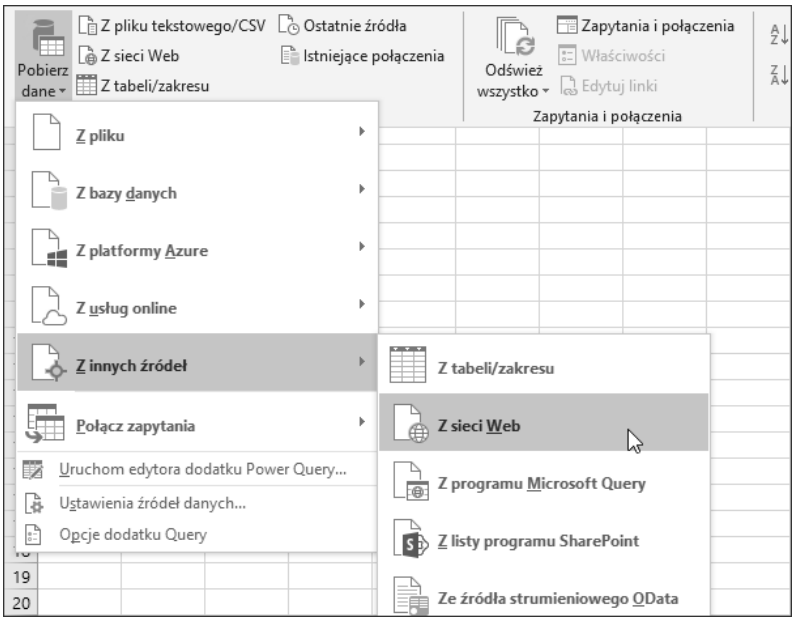
Wprowadzenie do zapytań Power Query

Zapytania Power Query to intuicyjny mechanizm pozwalający na pobieranie danych z wielu różnych źródeł, wykonywanie złożonych operacji na tych danych, a następnie ładowanie ich do skoroszytu.

Aby się przekonać, że zapytania Power Query nie są takie straszne, jak by się mogło na pierwszy rzut oka wydawać, przejdziemy od razu do prostego przykładu. Wyobraź sobie, że musisz zaimportować ceny akcji firmy Microsoft Corporation za pomocą usługi Yahoo Finance i załadować je do skoroszytu Excela. W takim scenariuszu musisz wykonać odpowiednie zapytanie internetowe, aby pobrać potrzebne dane z serwisu Yahoo Finance.

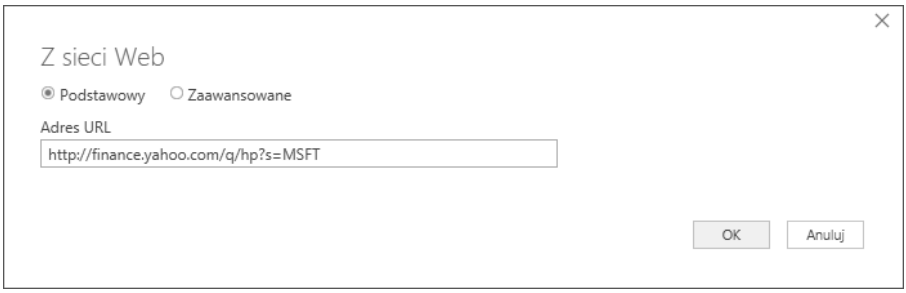
Aby utworzyć zapytanie, wykonaj następujące polecenia opisane poniżej:

1. W nowym skoroszycie programu Excel przejdź na kartę *Dane* i z grupy *Pobieranie i przekształcanie danych* wybierz polecenie *Pobierz dane/Z innych źródeł/Z sieci Web* (zobacz rysunek 11.1).



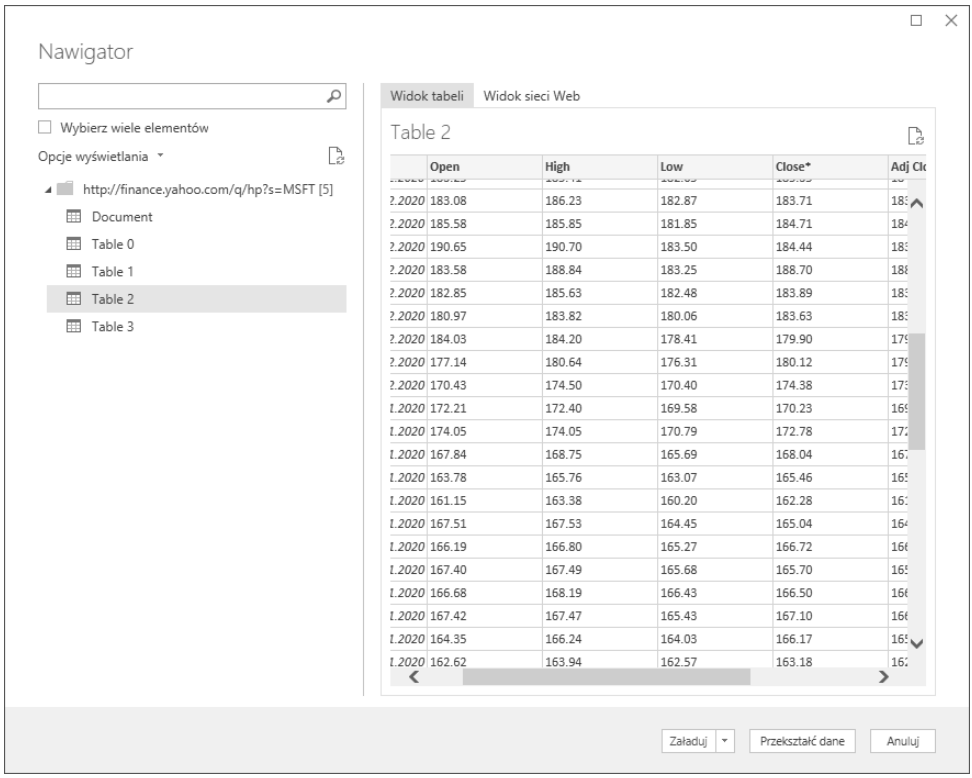
RYSUNEK 11.1. Tworzenie zapytania Power Query

2. W oknie dialogowym, które pojawi się na ekranie, wpisz adres URL potrzebnych danych i naciśnij przycisk *OK*; w naszym przypadku będzie to *http://finance.yahoo.com/q/hp?s=MSFT* (zobacz rysunek 11.2).



RYSUNEK 11.2. Wpisywanie adresu URL danych

3. Po krótkiej chwili na ekranie pojawi się okienko nawigatora pokazane na rysunku 11.3. Wybierz źródło danych, które chcesz pobrać. Aby zobaczyć podgląd danych, kliknij wybraną tabelę. W naszym przypadku potrzebne dane historyczne dotyczące notowań akcji zawiera tabela z etykietą *Table 2*, dlatego zaznacz tę tabelę, klikając jej nazwę lewym przyciskiem myszy, a następnie naciśnij przycisk *Przekształć dane*.



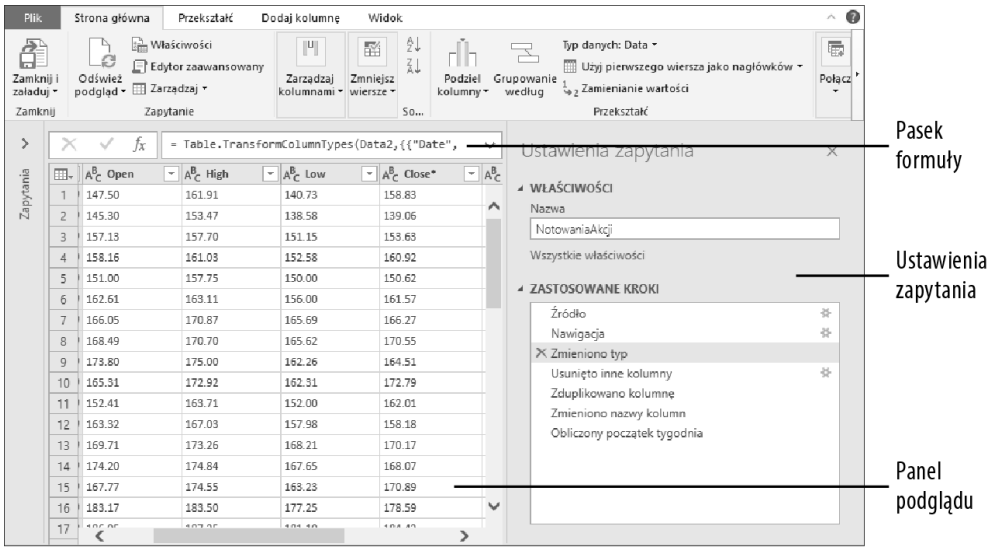
RYSUNEK 11.3. Wybierz żądane źródło danych i naciśnij przycisk Przekształć dane

Być może zauważyłeś, że w okienku *Nawigator*, pokazanym na rysunku 11.3, znajduje się również przycisk *Załaduj* (obok przycisku *Przekształć dane*). Przycisk *Załaduj* pozwala pominąć edycję i zaimportować dane docelowe bez żadnych zmian. Jeżeli masz pewność, że nie będziesz musiał w żaden sposób przekształcać ani modyfikować pobieranych danych, możesz po prostu nacisnąć przycisk *Załaduj* i zaimportować dane bezpośrednio do modelu danych lub arkusza w skoroszycie.

Excel posiada jeszcze inny przycisk polecenia *Z sieci Web*, który znajduje się na karcie *Dane* po prawej stronie przycisku polecenia *Pobierz dane*. To niefortunnie zduplikowane (i zlokalizowane) polecenie jest w rzeczywistości starszym narzędziem do pobierania danych z internetu, które można znaleźć już w Excelu 2000 i wszystkich nowszych wersjach tego programu.

Polecenie *Z sieci Web* w wersji Power Query (znajdujące się w menu rozwijanym polecenia *Pobierz dane*) znacznie wykracza swoją funkcjonalnością poza zwykłe pobieranie danych z internetu. Zapytanie Power Query jest w stanie pobierać dane z rozbudowanych stron internetowych i może je odpowiednio przekształcać w zależności od potrzeb użytkownika. Z tego względu, pobierając dane z internetu, powinieneś zawsze upewnić się, że używasz prawidłowego polecenia.

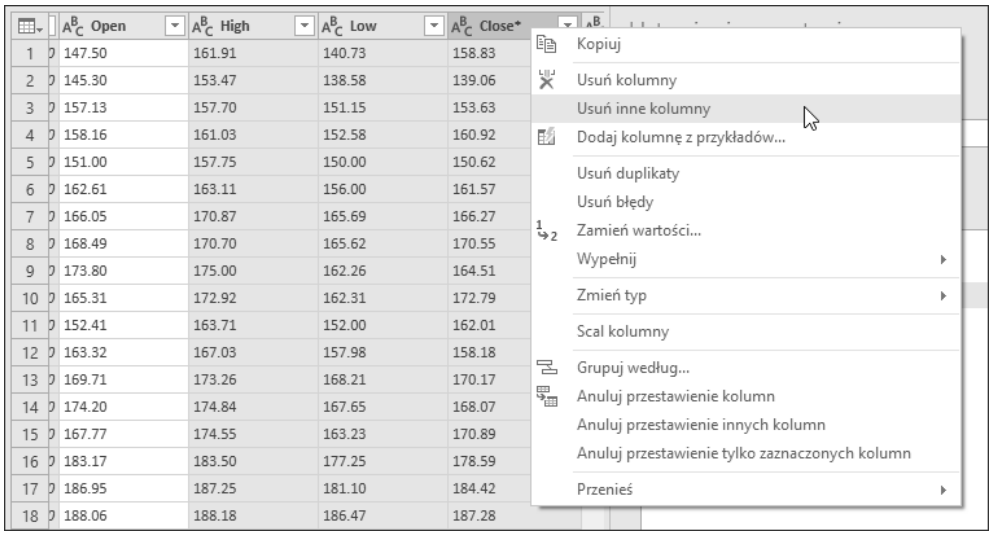
Po naciśnięciu przycisku *Przekształć dane* na ekranie pojawi się okno edytora zapytań Power Query, które posiada własną Wstążkę i panel pokazujący podgląd pobieranych danych (zobacz rysunek 11.4). Korzystając z tego okna, możesz zdefiniować określone działania w celu modyfikowania, czyszczenia i przekształcania danych przed zaimportowaniem do arkusza Excela.



RYSUNEK 11.4. Edytor Power Query pozwala na modyfikowanie, czyszczenia i przekształcanie danych

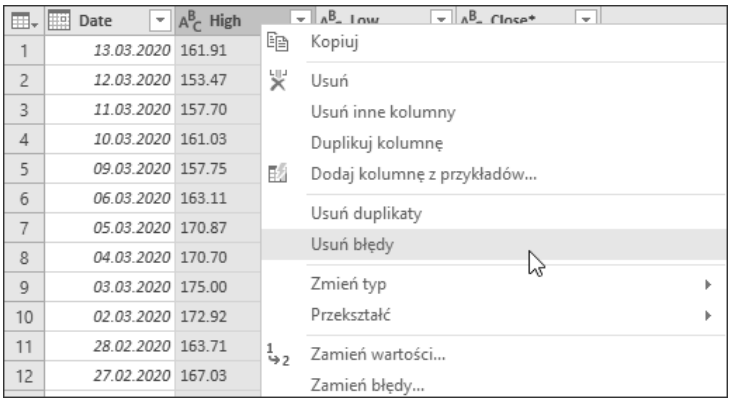
W edytorze Power Query możesz pracować z każdą kolumną danych osobno, definiując niezbędne działania, które zapewnią Ci potrzebne dane oraz ich strukturę. Więcej szczegółowych informacji na ten temat znajdziesz w dalszej części tego rozdziału, a teraz skoncentrujemy się na tym, jak pobrać notowania akcji firmy Microsoft Corporation z ostatnich 30 dni.

4. Usuń wszystkie niepotrzebne kolumny, klikając je po kolei prawym przyciskiem myszy i wybierając z menu podręcznego polecenie *Usuń* (oprócz pola *Date* będą Ci potrzebne jedynie kolumny *High*, *Low* i *Close*). Możesz też wcisnąć i przytrzymać klawisz *Ctrl*, zaznaczyć kolumny, które chcesz zachować, a następnie kliknąć prawym przyciskiem myszy dowolną z wybranych kolumn i z menu podręcznego wybrać polecenie *Usuń inne kolumny* (zobacz rysunek 11.5).



RYSUNEK 11.5. Wybierz kolumny, które chcesz zachować, i z menu podręcznego wybierz opcję Usuń inne kolumny

5. Upewnij się, że pola *High*, *Low* i *Close* są odpowiednio sformatowane jako liczby. Aby to zrobić, przytrzymaj klawisz *Ctrl* na klawiaturze, zaznacz trzy kolumny z danymi, kliknij prawym przyciskiem myszy jeden z nagłówków kolumn, a następnie z menu podręcznego wybierz polecenie *Zmień typ / Liczba dziesiętna*. Po wykonaniu tej czynności możesz zauważyć, że niektóre wiersze zawierają słowo *Error*. Są to wiersze zawierające wartości tekstowe, których nie można przekonwertować do postaci liczbowej.
6. W razie potrzeby usuń wiersze zawierające błędy, wybierając z menu podręcznego polecenie *Usuń błędy*, tak jak to zostało pokazane na rysunku 11.6.

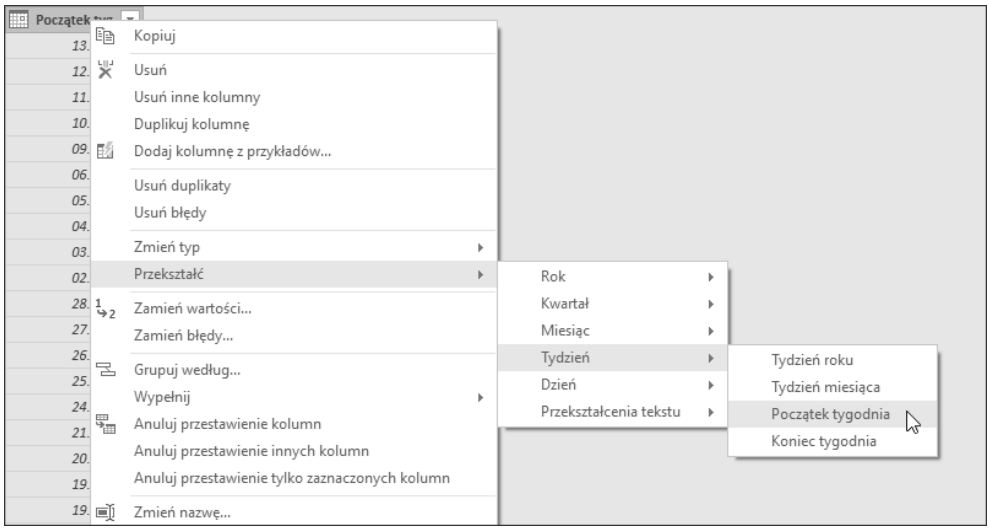


RYSUNEK 11.6. Z menu podręcznego możesz wybrać różne polecenia, które zostaną zastosowane do tabeli danych (na przykład polecenie Usuń błędy)

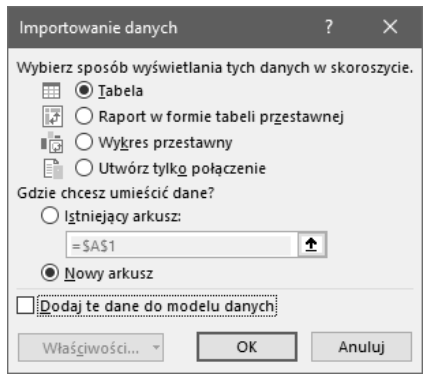
7. Po usunięciu wszystkich błędów dodaj nowe pole po nazwie *Początek tygodnia*, w którym będzie wyświetlany dzień rozpoczynający dany tydzień. Aby to zrobić, kliknij prawym przyciskiem myszy pole *Date* i wybierz opcję *Duplikuj kolumnę*. Do podglądu danych zostanie dodana nowa kolumna o nazwie *Date — Copy*).
8. Kliknij prawym przyciskiem myszy nowo dodaną kolumnę i z menu podręcznego wybierz polecenie *Zmień nazwę*, a następnie wpisz nową nazwę kolumny (*Początek tygodnia*).
9. Kliknij prawym przyciskiem myszy właśnie utworzoną kolumnę *Początek tygodnia* i z menu podręcznego wybierz polecenie *Przekształć/Tydzień/Początek tygodnia*, jak pokazano na rysunku 11.7. Excel przekształci daty tak, aby wyświetlić początek tygodnia dla kolejnych dat.
10. Po zakończeniu konfigurowania zapytania Power Query zapisz je i załaduj wyniki działania. Aby to zrobić, naciśnij przycisk *Zamknij i załaduj*, znajdujący się na karcie *Strona główna* wstążki edytora Power Query, i wybierz jedną z dwóch dostępnych opcji: *Zamknij i załaduj* lub *Zamknij i załaduj do...*

    Opcja *Zamknij i załaduj* powoduje zapisanie zapytania i załadowanie wyników jego działania w nowym arkuszu w skoroszycie w postaci tabeli programu Excel. Opcja *Zamknij i wczytaj do* powoduje wyświetlenie okna dialogowego *Importowanie danych*, które zostało pokazane na rysunku 11.8. Możesz w nim określić, czy pobierane dane zostaną załadowane do arkusza, czy do wewnętrznego modelu danych oraz w jaki sposób będą wyświetlane.

    Okno dialogowe *Importowanie danych* umożliwia również zapisanie zapytania jako połączenia ze źródłem danych, co oznacza, że będziesz mógł użyć zapytania bezpośrednio w różnych procesach w pamięci bez potrzeby wyświetlania wyników w arkuszu.
11. Wybierz opcję *Nowy arkusz*, aby wyświetlić wyniki jako tabelę w nowym arkuszu w aktywnym skoroszycie.



RYSUNEK 11.7. Za pomocą Edytora Power Query możesz dokonywać transformacji danych, takich jak np. wyświetlanie pierwszego dnia tygodnia dla określonej daty



RYSUNEK 11.8. Okno dialogowe Importowanie danych zapewnia pełną kontrolę nad sposobem wykorzystania wyników zapytań

Po wykonaniu zapytania dane zostaną pobrane i wyświetlone w postaci tabeli, tak jak to zostało pokazane na rysunku 11.9; takiej tabeli możemy teraz użyć na przykład do utworzenia potrzebnej tabeli przestawnej.

	A	B	C	D	E
1	Date	High	Low	Close*	Początek tyg.
2	13.03.2020	161.91	140.73	158.83	09.03.2020
3	12.03.2020	153.47	138.58	139.06	09.03.2020
4	11.03.2020	157.70	151.15	153.63	09.03.2020
5	10.03.2020	161.03	152.58	160.92	09.03.2020
6	09.03.2020	157.75	150.00	150.62	09.03.2020
7	06.03.2020	163.11	156.00	161.57	02.03.2020
8	05.03.2020	170.87	165.69	166.27	02.03.2020
9	04.03.2020	170.70	165.62	170.55	02.03.2020
10	03.03.2020	175.00	162.26	164.51	02.03.2020

RYSUNEK 11.9. Wyniki działania zapytania pobierającego dane z internetu zostały przekształcone, umieszczone w tabeli Excela; dane są gotowe do użycia w tabeli przestawnej

Teraz powinieneś poświęcić chwilę, aby dobrze zrozumieć i docenić to, co Power Query pozwoliło Ci przed chwilą zrobić. Za pomocą kilku kliknięć przeszukałeś internet, znalazłeś podstawowy zestaw danych, przetworzyłeś je, aby zachować tylko te kolumny, których potrzebujesz, a nawet zmodyfikowałeś je tak, aby dodać do podstawowych danych kolumnę zawierającą pierwszy dzień tygodnia. Właśnie o to chodzi w zapytaniach Power Query: umożliwienie łatwego pobierania, filtrowania i przekształcania danych bez konieczności posiadania jakiegokolwiek doświadczenia w programowaniu.

Najlepsze jest to, że zapytania Power Query mają możliwość łączenia się z szeroką gamą źródeł danych. Niezależnie od tego, czy chcesz pobierać dane z zewnętrznej strony internetowej, pliku tekstowego, systemu baz danych, Facebooka, czy serwisu internetowego, Power Query może zaspokoić większość, jeśli nie wszystkie Twoje potrzeby dotyczące źródeł danych. Wszystkie dostępne typy połączeń możesz wyświetlić, naciskając przycisk *Pobierz dane* znajdujący się na karcie *Dane*.

Zapytania Power Query oferują możliwość pobierania informacji z szerokiej gamy źródeł danych:

- *Z pliku* — pobiera dane z określonych plików Excela, plików tekstowych, plików CSV, plików XML lub całych folderów.
- *Z bazy danych* — pobiera dane z baz danych, takich jak Microsoft Access, SQL Server czy SQL Server Analysis Services.
- *Z platformy Azure* — pobiera dane z usługi Azure Cloud Services firmy Microsoft.
- *Z usług online* — pobiera dane z aplikacji w chmurze, takich jak Facebook, Salesforce czy Microsoft Dynamics online.
- *Z innych źródeł* — pobiera dane z szerokiej gamy innych źródeł, takich jak sieć WWW, chmury danych czy źródła danych ODBC.

Etapy tworzenia zapytania

Power Query używa własnego języka formuł, znanego jako język **M**, do kodowania Twoich zapytań. Podobnie jak w przypadku rejestrowania makr, każde działanie podejmowane podczas pracy z Power Query powoduje zapisywanie kodu w danym kroku zapytania. Poszczególne etapy zapytania są zarejestrowane w kodzie M, który umożliwia powtarzanie działań przy każdym odświeżeniu danych zapytania Power Query.

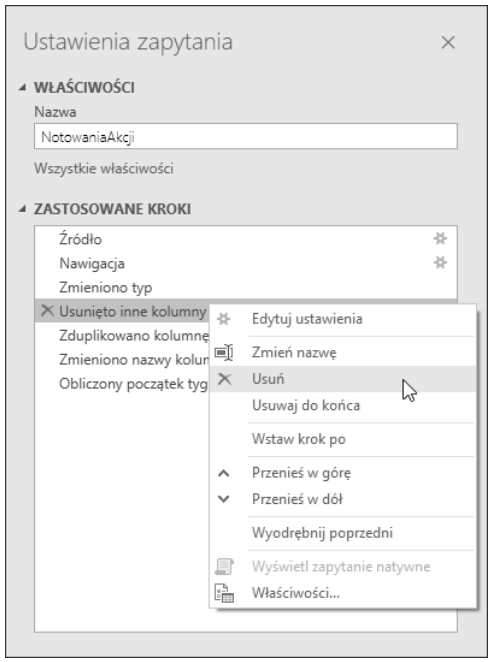
Możesz wyświetlić kolejne etapy zapytania, aktywując w oknie edytora Power Query panel *Ustawienia zapytania* (patrz rysunek 11.10). Aby to zrobić, wystarczy kliknąć polecenie *Ustawienia zapytania*, znajdujące się na karcie *Widok* Wstążki.

Zwróć uwagę na pokazane na rysunku 11.10 pole *Nazwa*, za pomocą którego możesz nadać danemu zapytaniu przyjazną nazwę. W naszym przykładzie nadaliśmy zapytaniu nazwę *NotowaniaAkcji*.

Możesz także włączyć wyświetlanie paska formuły, w którym zawarte są polecenia języka M dla wybranego kroku. Aby to zrobić, powinieneś w edytorze zapytań przejść na kartę *Widok* i włączyć opcję *Pasek formuły*, znajdującą się w grupie opcji *Układ*.

Każdy krok zapytania reprezentuje działanie podjęte w celu uzyskania dostępu do danych. Aby na pasku formuły Power Query wyświetlić kod w języku M, powinieneś po prostu kliknąć dowolny krok zapytania. Na przykład kliknięcie kroku o nazwie *Usuń błędy* spowoduje wyświetlenie na pasku formuły kodu tego kroku.

Po kliknięciu wybranego kroku zapytania dane wyświetlane w okienku podglądu są obrazem danych aż do kliknięcia kroku. Na przykład kliknięcie pokazanego na rysunku 11.10 kroku wykonanego przed wykonaniem polecenia *Usuń inne kolumny* pozwali zobaczyć, jak wyglądały dane przed usunięciem niepotrzebnych kolumn.



RYSUNEK 11.10. Kolejnymi krokami zapytania można zarządzać w sekcji Zastosowane kroki w panelu Ustawienia zapytania

Odświeżanie danych z zapytania Power Query

Powinieneś zawsze pamiętać, że dane pobierane za pomocą zapytań Power Query nie są w żaden sposób powiązane z danymi źródłowymi użytymi do ich wyodrębnienia. Innymi słowy, tabela danych uzyskana za pomocą zapytania Power Query jest jedynie migawką danych dostępnych w chwili wykonania zapytania. Innymi słowy, gdy zmieniają się dane źródłowe, zapytania Power Query nie będą automatycznie nadążać za zmianami; musisz ręcznie odświeżać swoje dane.

Jeżeli zdecydujesz się załadować wyniki Power Query do tabeli Excela w istniejącym skoroszycie, możesz je ręcznie odświeżyć, klikając tabelę prawym przyciskiem myszy i wybierając opcję *Odśwież*.

Jeżeli zdecydujesz się załadować dane z zapytania Power Query do wewnętrznego modelu danych, to aby odświeżyć dane, musisz przejść na kartę *Dane*, wybrać polecenie *Zapytania i połączenia* znajdujące się w grupie opcji *Zapytania i połączenia*, a następnie kliknąć prawym przyciskiem myszy wybrane zapytanie docelowe w panelu zadań i z menu podręcznego wybrać opcję *Odśwież*.

Aby nieco bardziej zautomatyzować odświeżanie zapytań, możesz skonfigurować źródła danych, aby automatycznie odświeżały dane z zapytań Power Query. Aby to zrobić, wykonaj następujące kroki:

1. Przejdź na kartę *Dane* na wstążce programu Excel i wybierz polecenie *Zapytania i połączenia*. Na ekranie pojawi się panel *Zapytania i połączenia*.
2. Kliknij prawym przyciskiem myszy zapytanie Power Query, które chcesz odświeżyć, a następnie wybierz opcję *Właściwości*.
3. Na ekranie pojawi się okno dialogowe *Właściwości zapytania*. Przejdź na kartę *Użycie*.
4. Ustaw opcje odświeżania wybranego zapytania zgodnie z potrzebami:
   - ***Odśwież co n minut*** — zaznaczenie tej opcji powoduje, że program Excel automatycznie odświeża dane wybranego zapytania co określoną liczbę minut. Excel odświeży dane ze wszystkich tabel związanych z tym zapytaniem.

- ***Odśwież dane podczas otwierania pliku*** — zaznaczenie tej opcji informuje Excela o konieczności odświeżenia danych z wybranego zapytania automatycznie po każdorazowym otwarciu skoroszytu. Excel odświeży dane ze wszystkich tabel powiązanych z tym zapytaniem za każdym razem, kiedy skoroszyt zostanie otwarty.

Wspomniane wyżej opcje odświeżania są bardzo przydatne, gdy chcesz mieć pewność, że zawsze korzystasz z najnowszych, najbardziej aktualnych danych. Oczywiście ustawienie tych opcji nie wyklucza możliwości ręcznego odświeżenia danych.

Zarządzanie istniejącymi zapytaniami

Gdy dodajesz różne zapytania do skoroszytu, będziesz potrzebować sposobu na zarządzanie nimi. Excel rozwiązuje ten problem za pomocą panelu *Zapytania i połączenia*, który umożliwia edycję, duplikowanie, odświeżanie i ogólne zarządzanie wszystkimi zapytaniami utworzonymi w skoroszycie. Aby aktywować panel *Zapytania i połączenia*, powinieneś przejść na kartę *Dane* wstążki Excela i wybrać polecenie *Zapytania i połączenia*.

Kiedy panel pojawi się na ekranie, odszukaj zapytanie, z którym chcesz pracować, a następnie kliknij je prawym przyciskiem myszy i z menu podręcznego wybierz jedno z następujących poleceń:

- *Edytuj* — otwiera edytor Power Query, w którym możesz modyfikować ustawienia zapytania.
- *Usuń* — usuwa wybrane zapytanie.
- *Odśwież* — odświeża dane w wybranym zapytaniu.
- *Załaduj do* — aktywuje okno dialogowe *Importowanie danych*, w którym możesz określić, gdzie będą używane wyniki wybranego zapytania.
- *Duplikuj* — tworzy kopię zapytania.
- *Odwołanie* — tworzy nowe zapytanie, które odwołuje się do wyników działania oryginalnego zapytania.
- *Scalanie* — scala wybrane zapytanie z innym zapytaniem w skoroszycie, dopasowując określone kolumny.
- *Dołączanie* — dołącza wyniki innego zapytania w skoroszycie do wybranego zapytania.
- *Eksportuj plik połączenia* — tworzy plik *.odc*, pozwalający na przeniesienie lub udostępnienie wybranego zapytania.
- *Przenieś do grupy* — przenosi wybrane zapytanie do grupy logicznej; takie grupy pozwalają na lepszą organizację zapytań.
- *Przenieś w górę* — przesuwa wybrane zapytanie w górę listy zapytań w panelu *Zapytania i połączenia*.
- *Przenieś w dół* — przesuwa wybrane zapytanie w dół listy zapytań w panelu *Zapytania i połączenia*.
- *Pokaż wartość szczytową* — wyświetla podgląd wyników działania wybranego zapytania.
- *Właściwości* — pozwala na zmianę nazwy zapytania i ustawienie jego wielu różnych właściwości.

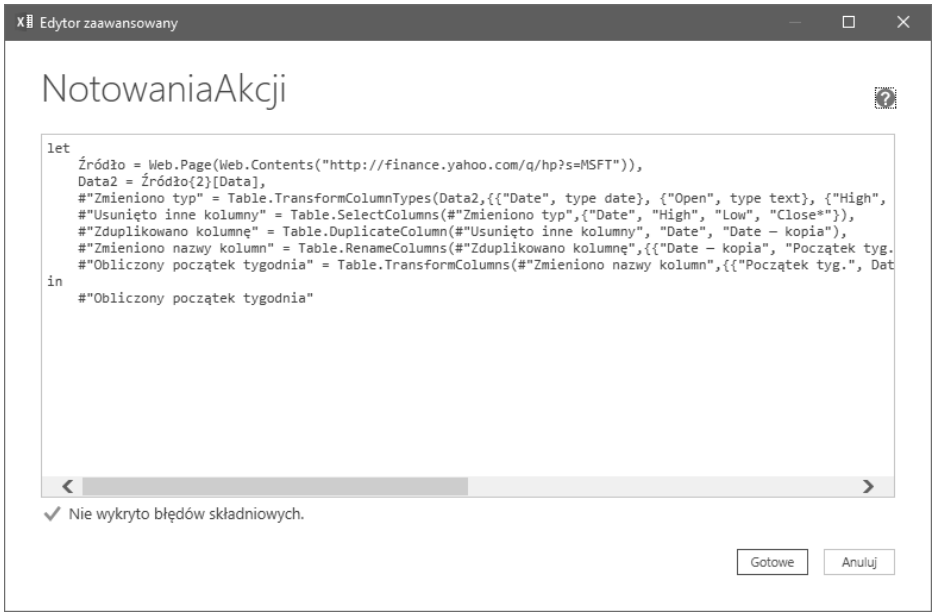
Panel *Zapytania i połączenia* jest szczególnie przydatny, gdy skoroszyt zawiera kilka zapytań. Możemy go traktować jak coś w rodzaju spisu treści, który pozwala łatwo znaleźć i wchodzić w interakcje z poszczególnymi zapytaniami w skoroszycie.

Używanie języka VBA do tworzenia dynamicznych połączeń

Budując niestandardowe zapytanie Power Query, zasadniczo nie robisz nic poza rejestrowaniem składni potrzebnej do zwrócenia pożądanego wyniku. Kod dowolnego zapytania Power Query może zostać skopiowany z okna zaawansowanego edytora zapytań, a następnie użyty w języku VBA.

Aby przywołać na ekran zaawansowany edytor zapytań, powinieneś w oknie edytora Power Query przejść na kartę *Widok* i wybrać polecenie *Edytor zaawansowany*.

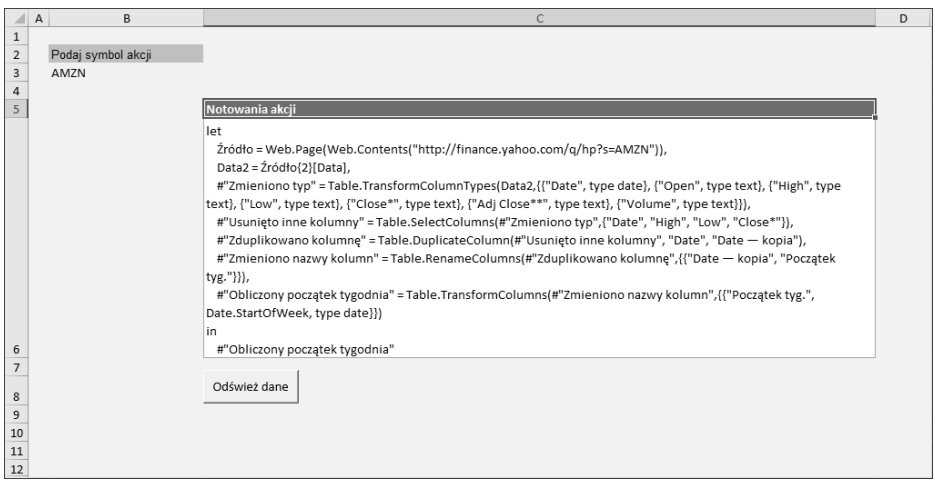
Jeżeli poprawnie wykonałeś pierwsze ćwiczenie, okno zaawansowanego edytora zapytań powinno wyglądać mniej więcej tak, jak to zostało pokazane na rysunku 11.11.



RYSUNEK 11.11. Okno zaawansowanego edytora zapytań

Ogromną zaletą takiego rozwiązania jest to, że nie musisz być ekspertem programowania w języku M, aby za pomocą VBA dynamicznie tworzyć i budować złożone zapytania pobierające dane ze źródeł zewnętrznych.

Na przykład na rysunku 11.12 pokazano arkusz, w którym w komórce C6 można wybrać symbol akcji, aby zmienić składnię zapytania Power Query pobierającego notowania giełdowe wybranej spółki. Kliknięcie przycisku *Odśwież dane* spowoduje przebudowanie zapytania Power Query z uwzględnieniem nowej składni.



RYSUNEK 11.12. Wyznaczona komórka przechowuje kryterium wyboru zapytania

Poniższe makro używa obiektów `Workbook.Query` i `Workbook.Connection` do przebudowania zapytania na podstawie podanego kryterium:

```
Sub RefreshPowerQuery()
    Dim Qry As WorkbookQuery
    Dim QryName As String
    Dim QrySyntax As String
    Dim QryDesc As String
    Dim OutputSheet As Worksheet
    Dim ws As Worksheet

'   Ustawianie zmiennych
    QryName = ThisWorkbook.Sheets("Query Changer").Range("C5").Value
    QrySyntax = ThisWorkbook.Sheets("Query Changer").Range("C6").Value
    QryDesc = ThisWorkbook.Sheets("Query Changer").Range("C5").Value

'   Usuwanie istniejącego zapytania
    For Each Qry In ThisWorkbook.Queries
        If Qry.Name = QryName Then
            Set Qry = ThisWorkbook.Queries(QryName)
            Qry.Delete
        End If
    Next Qry

'   Dodawanie nowego zapytania
    Set Qry = ThisWorkbook.Queries.Add(QryName, QrySyntax, QryDesc)

'   Usuwanie starego arkusza
    Application.DisplayAlerts = False
    For Each ws In ThisWorkbook.Worksheets
        If ws.Name = QryName Then ws.Delete
    Next ws
    Application.DisplayAlerts = True

'   Dodawanie do nowego arkusza
    Set OutputSheet = Sheets.Add(After:=ActiveSheet)
    OutputSheet.Name = QryName

    With OutputSheet.ListObjects.Add(SourceType:=0, Source:= _
        "OLEDB;Provider=Microsoft.Mashup.OleDb.1;Data " & _
        "Source=$Workbook$;Location=" & Qry.Name _
        , Destination:=Range("$A$1")).QueryTable
        .CommandType = xlCmdDefault
        .CommandText = Array("SELECT * FROM [" & Qry.Name & "]")
        .RefreshOnFileOpen = False
        .BackgroundQuery = True
    End With
End Sub
```

Jeżeli wszystko pójdzie gładko i zgodnie z oczekiwaniami, po zakończeniu będziesz dysponować sprytnym mechanizmem, który pozwala na dynamiczne edytowanie składni zapytań Power Query, pozwalającym na tworzenie bardziej elastycznych raportów.

FTP Przykładowy skoroszyt o nazwie *PowerQuery.xlsx* znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

Przechodzenie w pętli przez wszystkie połączenia skoroszytu

Kolekcji `Workbook.Connections` możesz również używać do przechodzenia w pętli przez wszystkie połączenia skoroszytu i sprawdzania lub modyfikowania ich właściwości. Na przykład makro przedstawione poniżej wypełnia arkusz listą wszystkich połączeń aktywnego skoroszytu razem z ich ciągami połączenia oraz treścią zapytań SQL zapisanych we właściwościach `CommandText`:

```
Sub ListConnections()

   Dim i As Long
   Dim Cn As WorkbookConnection

   Worksheets.Add
   With ActiveSheet.Range("A1:C1")
      .Value = Array("Cn Name", "Connection String", "Command Text")
      .EntireColumn.AutoFit
   End With

   For Each Cn In ThisWorkbook.Connections
      i = i + 1
      Select Case Cn.Type
         Case Is = xlConnectionTypeODBC
            With ActiveSheet
               .Range("A1").Offset(i, 0).Value = Cn.Name
               .Range("A1").Offset(i, 1).Value = Cn.ODBCConnection.
               ↪Connection
               .Range("A1").Offset(i, 2).Value = Cn.ODBCConnection.
               ↪CommandText
            End With
         Case Is = xlConnectionTypeOLEDB
            With ActiveSheet
               .Range("A1").Offset(i, 0).Value = Cn.Name
               .Range("A1").Offset(i, 1).Value = Cn.OLEDBConnection.
               ↪Connection
               .Range("A1").Offset(i, 2).Value = Cn.OLEDBConnection.
               ↪CommandText
            End With
      End Select
      Next Cn
End Sub
```

Zastosowanie ADO i VBA do pobierania danych ze źródeł zewnętrznych

Kolejną techniką pozwalającą na pracę z danymi zewnętrznymi jest zastosowanie procedur VBA i obiektów ADO (ang. *ActiveX Data Objects*). Zastosowanie kombinacji ADO z procedurami języka VBA umożliwia pracę z danymi zewnętrznymi przechowywanymi w buforach w pamięci operacyjnej. Takie rozwiązanie jest bardzo wygodne zwłaszcza w sytuacji, kiedy musisz wykonywać złożone, wielopoziomowe procedury przetwarzające dane zewnętrzne, ale jednocześnie nie chcesz tworzyć połączeń danych w skoroszycie ani zapisywać danych zewnętrznych bezpośrednio w skoroszycie.

Pracując ze złożonymi skoroszytami Excela, które pobierają i przetwarzają dane ze źródeł zewnętrznych, z pewnością wcześniej czy później spotkasz się z procedurami VBA wykorzystującymi obiekty ADO. Z tego powodu powinieneś dobrze poznać i zrozumieć zasady pracy z takimi obiektami, tak abyś mógł swobodnie korzystać z takiego kodu.

W kilku kolejnych podrozdziałach omówimy szereg najważniejszych koncepcji i zagadnień związanych z obiektami ADO oraz pokażemy, w jaki sposób możesz tworzyć swoje własne procedury wykorzystujące ADO do pobierania danych z zewnętrznych źródeł danych. Pamiętaj jednak, że obiekty ADO to bardzo szerokie zagadnienie, którego z oczywistych powodów nie jesteśmy tutaj w stanie wyczerpująco omówić. Jeżeli planujesz wykorzystywanie obiektów ADO do pobierania i przetwarzania danych ze źródeł zewnętrznych na większą skalę w swoich projektach, to prawdopodobnie powinieneś jeszcze dodatkowo zainwestować w kolejną książkę, która będzie w bardziej szczegółowy sposób przedstawiała takie zagadnienia.

Jeżeli chcesz szybko „załapać" podstawową koncepcję obiektów ADO, powinieneś spróbować sobie wyobrazić ADO jako narzędzie, które pozwala na zrealizowanie dwóch zadań: utworzenie połączenia z zewnętrznym źródłem danych oraz wybranie zestawu danych, z którymi chcesz pracować. W kolejnym podrozdziale przedstawimy składnię podstawowych poleceń i wyrażeń umożliwiających pracę z obiektami ADO.

Ciąg połączenia

Pierwszą operacją, jaką musisz wykonać, jest nawiązanie połączenia ze źródłem danych. Aby to zrobić, musisz podać językowi VBA kilka podstawowych informacji, które są przekazywane do interpretera w formie tak zwanego *ciągu połączenia* (ang. *connection string*). Poniżej przedstawiamy przykład ciągu połączenia wykorzystywanego do nawiązania połączenia z bazą danych Access:

```
"Provider=Microsoft.ACE.OLEDB.12.0;" & _
"Data Source= C:\MyDatabase.accdb;" & _
"User ID=Administrator;" & _
"Password=AdminPassword"
```

Nie bądź onieśmielony pozornie złożoną składnią przedstawionego przykładu. Ciąg połączenia to w zasadzie nic innego jak tylko ciąg tekstowy zawierający szereg zmiennych (nazywanych również *argumentami*) i ich wartości, których VBA używa do identyfikacji i utworzenia połączenia ze źródłem danych. Choć niektóre ciągi połączenia mogą być naprawdę imponujące i zawierać dziesiątki argumentów i opcji, to jednak do tworzenia połączeń z bazami danych Access czy skoroszytami Excela będziemy wykorzystywać tylko niektóre z nich.

Jeżeli nigdy wcześniej nie miałeś okazji pracować z obiektami ADO, to możesz skoncentrować się tylko na kilku podstawowych argumentach ciągu połączenia, których znajomość jest absolutnie niezbędna: `Provider` (nazwa dostawcy), `Data Source` (źródło danych), `Extended Properties` (rozszerzone właściwości), `User ID` (nazwa konta użytkownika) oraz `Password` (hasło dostępu):

- **`Provider`** (nazwa dostawcy) — argument ten informuje VBA o tym, z jakim typem źródła danych będziesz pracować. Jeżeli jako źródła danych używasz bazy danych Access lub skoroszytów Excela, to argument ten powinien wyglądać następująco:

  ```
  Provider=Microsoft.ACE.OLEDB.12.0
  ```

- **`Data Source`** (źródło danych) — ten argument informuje VBA o tym, gdzie można znaleźć bazę danych lub skoroszyt zawierający dane, z którymi będziesz pracować. Wartością tego argumentu jest zazwyczaj pełna ścieżka wskazująca lokalizację bazy danych lub skoroszytu, na przykład:

  ```
  Data Source=C:\Moje dokumenty\MojaBazaDanych.accdb
  ```

- **`Extended Properties`** (rozszerzone właściwości) — tego argumentu używamy zazwyczaj, kiedy tworzymy połączenie ze skoroszytem Excela. Argument informuje VBA o tym, że źródłem danych jest obiekt inny niż baza danych. Jeżeli pracujesz ze skoroszytami Excela, to argument ten powinien wyglądać następująco:

  ```
  Extended Properties=Excel 12.0
  ```

- **`User ID`** (nazwa konta użytkownika) — jest to argument opcjonalny, używany wyłącznie w sytuacjach, kiedy podanie nazwy konta użytkownika jest niezbędne do nawiązania połączenia ze źródłem danych, na przykład:

  ```
  User Id=JanKowalski
  ```

- **`Password`** (hasło dostępu) — jest to argument opcjonalny, używany wyłącznie w sytuacjach, kiedy podanie hasła dostępu (łącznie z nazwą konta użytkownika) jest niezbędne do nawiązania połączenia ze źródłem danych, na przykład:

  ```
  Password=MojeTajneHasło
  ```

Przyjrzymy się teraz kilku wybranym przykładom zastosowania opisanych wyżej argumentów w różnych ciągach połączeń.

- Połączenie z bazą danych Access:

  ```
  "Provider=Microsoft.ACE.OLEDB.12.0;" & _
  "Data Source= C:\MyDatabase.accdb"
  ```

- Połączenie z bazą danych Access z wykorzystaniem uwierzytelnienia (nazwa konta użytkownika i hasło dostępu):

  ```
  "Provider=Microsoft.ACE.OLEDB.12.0;" & _
  "Data Source= C:\MyDatabase.accdb;" & _
  "User ID=Administrator;" & _
  "Password=AdminPassword"
  ```

- Połączenie ze skoroszytem Excela:

  ```
  "Provider=Microsoft.ACE.OLEDB.12.0;" & _
  "Data Source=C:\MyExcelWorkbook.xlsx;" & _
  "Extended Properties=Excel 12.0"
  ```

Deklarowanie zestawu rekordów

Oprócz utworzenia połączenia ze źródłem danych konieczne jest również zdefiniowanie zestawu danych, z którym będziesz pracować. W terminologii obiektów ADO taki zestaw rekordów nosi nazwę `Recordset`. Obiekt `Recordset` to inaczej mówiąc kontener dla rekordów pobieranych ze źródła danych. Najpopularniejszą metodą definiowania zestawu danych jest otwarcie istniejącej tabeli lub widoku z użyciem następujących argumentów:

```
Recordset.Open Source, ConnectString, CursorType, LockType
```

Argument `Source` (źródło) wskazuje dane, które mają być pobierane. Zazwyczaj będzie to tabela, widok lub po prostu zapytanie SQL pobierające odpowiednie rekordy. Argument `ConnectString` definiuje ciąg połączenia wykorzystywany do utworzenia połączenia ze źródłem danych. Argument `CursorType` (typ kursora) określa, w jaki sposób obiekt `Recordset` będzie pozwalał poruszać się po pobieranych danych. Najczęściej spotykane wartości tego argumentu są następujące:

- **`adOpenForwardOnly`** — jest to ustawienie domyślne, jeżeli w definicji zestawu danych nie podasz żadnego typu kursora, to domyślnie zostanie przyjęta wartość `adOpenForwardOnly`. Taki rodzaj kursora jest najbardziej efektywny, ponieważ pozwala na poruszanie się w zestawie danych tylko w jednym kierunku: od początku do końca. Jest to idealne rozwiązanie do tworzenia raportów, gdzie dane muszą być po prostu sekwencyjnie odczytywane i nie ma

potrzeby odczytywania wybranych rekordów z różnych obszarów zestawu danych. Pamiętaj, że korzystając z tego typu kursora, nie możesz wprowadzać żadnych modyfikacji do danych w takim zestawie.

- **`adOpenDynamic`** — taki rodzaj kursora jest wykorzystywany w procesach, gdzie wymagane jest przechodzenie przez kolejne rekordy w pętli, poruszanie się w górę i w dół zestawu danych czy możliwość dynamicznego obserwowania zmian wprowadzanych do rekordów w zestawie danych. Zastosowanie takiego kursora wymaga zazwyczaj zaangażowania znacznych ilości pamięci operacyjnej i innych zasobów systemu, dlatego powinien być używany tylko wtedy, kiedy jest to niezbędne.
- **`adOpenStatic`** — taki kursor jest idealnym rozwiązaniem do szybkiego zwracania wyników i zwraca statyczny zbiór rekordów ze źródła danych. Działa bardzo podobnie jak kursory `adOpenForwardOnly`, ale pozwala na swobodne poruszanie się pośród zwracanych rekordów. Dodatkowo po ustawieniu odpowiedniej blokady `LockType` (innej niż `adLockReadOnly`) kursory `adOpenStatic` umożliwiają wprowadzanie zmian w zwracanych danych.

Argument `LockType` pozwala na określenie, czy dane umieszczane w zestawie danych `Recordset` mogą być modyfikowane. Domyślnie argument ten jest ustawiany na wartość `adLockReadOnly`, która wskazuje, że nie ma potrzeby (ani możliwości) modyfikowania zwracanych danych. Jeśli jednak zechcesz, możesz ustawić ten argument na wartość `adLockOptimistic`, dzięki czemu będziesz mógł swobodnie modyfikować zwracane dane.

Odwołania do biblioteki obiektów ADO

Poznałeś już podstawowe zagadnienia i pojęcia związane z obiektami ADO, możesz więc zacząć pisanie swojej pierwszej procedury, wykorzystującej takie obiekty. Zanim jednak rozpoczniesz, musisz utworzyć w edytorze VBE odwołanie do biblioteki obiektów ADO. Podobnie jak każda aplikacja pakietu Microsoft Office posiada swój zestaw obiektów, właściwości i metod, również ADO posiada swój zestaw takich obiektów. Ponieważ Excel domyślnie nie zna modelu obiektowego ADO, musimy wskazać Excelowi odpowiednie odwołanie do biblioteki obiektów ADO.

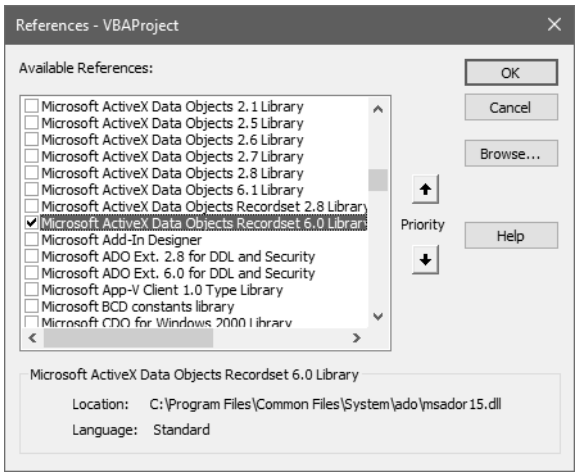
Aby to zrobić, otwórz nowy skoroszyt Excela, a następnie uruchom edytor VBE.

Po przejściu do okna edytora VBE wybierz z menu głównego polecenie *Tools/References* (narzędzia/odwołania). Na ekranie pojawi się okno dialogowe *References*, przedstawione na rysunku 11.13. Na liście *Available References* (dostępne odwołania) odszukaj i zaznacz najnowszą wersję biblioteki *Microsoft ActiveX Data Objects Library*, a następnie naciśnij przycisk *OK*.

To całkowicie normalne, że na liście dostępnych bibliotek pojawia się po kilka wersji tej samej biblioteki. Zazwyczaj najlepszym rozwiązaniem jest zaznaczenie najnowszej wersji. Zwróć uwagę, że począwszy od wersji 2.8, biblioteka ta nosi nazwę *Microsoft ActiveX Data Objects Recordset Library*.

Po naciśnięciu przycisku *OK* możesz ponownie otworzyć okno dialogowe *References* i upewnić się, czy odwołanie do nowej biblioteki zostało poprawnie ustawione. Możesz się o tym łatwo przekonać, ponieważ zaznaczone odwołania pojawiają się na szczycie listy.

Pamiętaj, że odwołania, które ustawisz w danym skoroszycie lub bazie danych, nie są ustawiane na poziomie aplikacji. Oznacza to, że ustawianie odwołań do wybranych bibliotek musisz powtarzać dla każdego nowego skoroszytu lub bazy danych, które mają z nich korzystać.



RYSUNEK 11.13. Zaznacz najnowszą wersję biblioteki Microsoft ActiveX Data Objects Library

Łączenie wszystkiego razem w kodzie procedury

Omówiliśmy już podstawowe zagadnienia związane z obiektami ADO, nadszedł więc czas, aby połączyć wszystkie zdobyte wiadomości w kodzie VBA. Przedstawiona poniżej przykładowa procedura wykorzystuje obiekty ADO do nawiązania połączenia z bazą Access i pobrania danych z tabeli `Products`.

```
Sub GetAccessData()
   Dim MyConnect As String
   Dim MyRecordset As ADODB.Recordset

   MyConnect = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
               "Data Source= C:\MyDir\MyDatabaseName.accdb"
   Set MyRecordset = New ADODB.Recordset
   MyRecordset.Open "Products", _
      MyConnect, adOpenStatic, adLockReadOnly

   Sheets("MySheetName").Range("A2").CopyFromRecordset MyRecordset

   With ActiveSheet.Range("A1:C1")
      .Value = Array("Product", "Description", "Segment")
      .EntireColumn.AutoFit
   End With
End Sub
```

Teraz poświęcimy chwilę na omówienie sposobu działania naszego makra.

Najpierw deklarujemy dwie zmienne: pierwsza to zmienna tekstowa, która będzie przechowywała ciąg połączenia, a druga to zmienna obiektowa typu `Recordset`, która będzie przechowywała zestaw danych pobieranych ze źródła. W naszym przykładzie zmienna `MyConnect` to zmienna tekstowa, w której zapiszemy ciąg połączenia identyfikujący źródło danych. Zmienna `MyRecordset` będzie przechowywała dane zwracane przez procedurę.

Następnie definiujemy ciąg połączenia dla procedury ADO. W tym scenariuszu łączymy się z bazą danych Access o nazwie *MyDatabaseName.accdb*, znajdującą się w katalogu *C:\MyDir*. Po zdefiniowaniu źródła danych możesz otworzyć zestaw danych i użyć zmiennej `MyConnect` do pobrania statycznego zestawu danych, który można tylko odczytywać.

Teraz możemy użyć metody `CopyFromRecordset` Excela do pobrania danych z zestawu danych i zapisania ich w skoroszycie. Metoda ta wymaga podania dwóch argumentów: docelowej lokalizacji danych oraz nazwy zestawu danych, z którego mają zostać pobrane rekordy. W naszym przykładzie będziemy kopiować dane z zestawu o nazwie `MyRecordset` do arkusza o nazwie `MySheetName` (rozpoczynając od komórki `A2`).

Co ciekawe, metoda `CopyFromRecordset` nie zwraca nagłówków kolumn ani nazw pól rekordów. Taki sposób działania wymusza wykonanie jeszcze jednej operacji, w której dodamy do arkusza nagłówki kolumn. Aby to zrobić, po prostu zdefiniujemy odpowiednie nagłówki w tablicy i zapiszemy je w aktywnym skoroszycie.

Korzystając z obiektów ADO i języka VBA, możesz utworzyć wszystkie niezbędne elementy połączenia w jednej, zgrabnej procedurze, o której możesz później spokojnie zapomnieć. Dopóki wartości zmiennych zdefiniowanych w kodzie (takie jak ścieżka do źródła danych, nazwa zestawu danych czy lokalizacja docelowa) nie będą się zmieniać, Twoja procedura wykorzystująca obiekty ADO nie będzie wymagała praktycznie żadnej aktualizacji.

Zastosowanie obiektów ADO w aktywnym skoroszycie

Wiedzę, którą nabyłeś w tym rozdziale, możesz wykorzystywać na wiele różnych sposobów. Z oczywistych powodów nie jesteśmy tutaj w stanie opisać wszystkich możliwości, ale istnieje jednak kilka najczęściej powtarzających się scenariuszy, w których VBA może znakomicie ułatwić integrację Excela z bazami danych Access.

Pobieranie danych ze skoroszytów Excela

Skoroszytów Excela możesz używać jako źródeł danych dla procedur ADO. Aby to zrobić, powinieneś po prostu utworzyć odpowiednie zapytania SQL, które będą się odwoływać do danych znajdujących się w skoroszycie Excela. Cała idea polega na tym, aby przypisać zestaw danych ze skoroszytu Excela do zapytania poprzez przekazanie nazwy arkusza, adresów zakresu komórek lub nazwy zakresu do takiego zapytania SQL.

Aby pobrać wszystkie dane z określonego arkusza, musimy przekazać do zapytania nazwę arkusza zakończoną znakiem dolara ($) jako nazwę tabeli. Pamiętaj, aby nazwę arkusza umieścić w nawiasach kwadratowych. Przykładowe zapytanie może wyglądać tak:

```
SELECT FROM [MySheet$]
```

Jeżeli nazwa arkusza zawiera spacje lub znaki specjalne, to musimy ją ująć w znaki apostrofu, na przykład:

```
SELECT FROM ['January; Forecast vs. Budget$']
```

Aby pobierać dane z wybranego zakresu komórek określonego arkusza, najpierw musimy zdefiniować arkusz w sposób opisany powyżej, a następnie dodać do niego adresy odpowiedniego zakresu komórek, na przykład:

```
SELECT * FROM [MySheet$A1:G17]
```

Aby pobierać dane z nazwanego zakresu komórek, wystarczy po prostu użyć w zapytaniu SQL nazwy zakresu zamiast nazwy tabeli, na przykład:

```
SELECT * FROM MyNamedRange
```

W przykładzie przedstawionym poniżej wykonujemy zapytanie na arkuszu SampleData i zwracamy tylko takie rekordy, dla których nazwa regionu to North.

```
Sub GetData_From_Excel_Sheet()

   Dim MyConnect As String
   Dim MyRecordset As ADODB.Recordset
   Dim MySQL As String

   MyConnect = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
               "Data Source=" & ThisWorkbook.FullName & ";" & _
               "Extended Properties=Excel 12.0"

   MySQL = " SELECT * FROM [SampleData$]" & _
           " WHERE Region ='NORTH'"

   Set MyRecordset = New ADODB.Recordset
   MyRecordset.Open MySQL, MyConnect, adOpenStatic, adLockReadOnly

   ThisWorkbook.Sheets.Add
   ActiveSheet.Range("A2").CopyFromRecordset MyRecordset

   With ActiveSheet.Range("A1:F1")
      .Value = Array("Region", "Market", "Branch_Number", _
      "Invoice_Number", "Sales_Amount", "Contracted Hours")
      .EntireColumn.AutoFit
   End With
End Sub
```

FTP Skoroszyt z tym przykładem (*Pobieranie danych z Excela.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

Dołączanie rekordów do istniejącej tabeli Excela

Bardzo często zdarza się, że nie chcesz w skoroszycie Excela nadpisywać istniejących danych nowymi rekordami. Wtedy lepszym rozwiązaniem będzie dołączanie nowych rekordów na końcu istniejącej tabeli. W typowym scenariuszu początkowa lokalizacja czy zakres komórek arkusza, do których chcesz skopiować nowy zestaw rekordów, są zakodowane na stałe w procedurze. W takich sytuacjach procedura musi dynamicznie modyfikować lokalizację docelową, aby dostosować ją do adresu pierwszej wolnej komórki arkusza. Poniżej przedstawiamy przykład procedury VBA wykorzystującej takie rozwiązanie:

```
Sub Append_Results()
   Dim MyConnect As String
   Dim MyRecordset As ADODB.Recordset
   Dim MyRange As String

   MyConnect = "Provider=Microsoft.ACE.OLEDB.12.0;" & _
               "Data Source= C:\MyDir\MyDatabase.accdb"

   Set MyRecordset = New ADODB.Recordset

   MyRecordset.Open "Products", MyConnect, adOpenStatic

   Sheets("AppendData").Select
   MyRange = "A" & _
```

```
    ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Row + 1

    ActiveSheet.Range(MyRange).CopyFromRecordset MyRecordset
End Sub
```

Ponieważ chcemy dołączyć nowe rekordy do istniejącej tabeli, musimy dynamicznie określić adres pierwszej wolnej komórki, której możemy użyć jako docelowej lokalizacji nowych rekordów. Pierwszym krokiem, jaki musimy wykonać, jest znalezienie pierwszego pustego wiersza. Dzięki zastosowaniu metody `SpecialCells` Excela nie jest to na szczęście trudnym zadaniem.

Korzystając z metody `SpecialCells`, możemy znaleźć adres ostatniej używanej komórki arkusza, a następnie odczytać numer wiersza, w którym ta komórka się znajduje. Taka operacja da nam numer ostatniego używanego wiersza arkusza. Aby uzyskać numer pierwszego pustego wiersza, wystarczy po prostu do numeru ostatniego używanego wiersza dodać `1` (wiersz następujący po ostatnim używanym wierszu musi być z definicji pusty).

Cały pomysł opiera się na połączeniu polecenia wykorzystującego metodę `SpecialCells` z literą reprezentującą kolumnę (w naszym przypadku `A`) do utworzenia ciągu znaków reprezentującego zakres komórek. Jeśli na przykład pierwszy pusty wiersz ma numer `10`, to kod przedstawiony poniżej zwróci adres `A10`.

```
"A" & ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Row + 1
```

Jeżeli teraz przypiszemy wynik działania powyższego polecenia do zmiennej tekstowej, np. `MyRange`, będziemy ją mogli następnie przekazać jako argument wywołania metody `CopyFromRecordset`.

Operacje na plikach tekstowych

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 swobodny** — 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 `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 rodzaj_dostępu] [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*.
- *rodzaj_dostępu* (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

Podstawowa 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

Podstawowa 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 "inny.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 wewnątrz 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 często nazywane plikami z kolumnami o stałej szerokości (ang. *fixed-width files*).
- **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 wyświetlania kreatora importowania. 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 określoną 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 tekstowego

Procedura przedstawiona poniżej odczytuje dane z pliku tekstowego, 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*.

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
   MsgBox ExpRng.Count & " komórek zostało wyeksportowanych do  " _
   & Filename, vbInformation
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`.

FTP Skoroszyt z tym przykładem (*Eksport-import-CSV.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

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
   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 zaimportowane daty będą otoczone znakami #, na przykład #2013-05-12#.

Logowanie 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 2019-11-16 21:27:43
```

Pokazana poniżej procedura wykonuje się podczas zamykania skoroszytu. Jej działanie polega na dodaniu do pliku tekstowego wiersza zawierającego frazę *Zakończenie pracy programu Excel* 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
```

FTP Skoroszyt z tym przykładem (*Excel Log.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

Więcej szczegółowych informacji na temat procedur obsługi zdarzeń `Workbook_Open` oraz `Workbook_BeforeClose` znajdziesz w rozdziale 6., „Obsługa zdarzeń”.

Filtrowanie zawartości pliku tekstowego

W przykładzie zaprezentowanym poniżej zademonstrujemy metodę jednoczesnego przetwarzania dwóch plików tekstowych. Procedura `FilterFile` odczytuje dane z pliku tekstowego (*infile.txt*) i kopiuje wiersze zawierające określony ciąg znaków (na przykład *"Styczeń"*) do drugiego pliku tekstowego (*output.txt*).

```
Sub FilterFile()

    Dim TextToFind As String
    Dim Filtered As Long
    Dim data As String

    Open ThisWorkbook.Path & "\infile.txt" For Input As #1
    Open Application.DefaultFilePath & "\output.txt" For Output As #2
    If Err <> 0 Then
      MsgBox "Błąd odczytu lub zapisu pliku."
      Exit Sub
```

```
    End If
    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
    MsgBox Filtered & " wierszy zostało zapisanych do pliku:" & vbNewLine & _
      Application.DefaultFilePath & "\output.txt"
End Sub
```

FTP Skoroszyt z tym przykładem (*Filtrowanie pliku tekstowego.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

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 (ang. *Hypertext Markup Language*).

W tym podrozdziale dowiesz się, jak używać języka Visual Basic for Applications (VBA) do realizacji zarówno tych najczęściej wykonywanych, jak i tych mniej popularnych operacji na plikach.

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 sekcje 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 11.1. Większość poleceń nie wymaga specjalnego komentarza, a wszystkie są opisane w systemie pomocy Excela.

W dalszej części tego rozdziału znajdziesz szereg przykładów ilustrujących zastosowanie niektórych poleceń.

TABELA 11.1. 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

Funkcja VBA sprawdzająca, czy istnieje dany plik

Poniższa funkcja zwraca wartość `True`, jeżeli określony plik istnieje, a wartość `False`, jeżeli plik nie zostanie odnaleziony. Jeżeli funkcja `Dir` zwraca pusty ciąg znaków, oznacza to, że nie można odnaleźć żądanego 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, a 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 zamykający ścieżkę jest opcjonalny. Poniżej przedstawiamy przykład wywołania takiej funkcji.

```
MyFolder = "c:\użytkownicy\jan\pulpit\pobieranie\"
MsgBox PathExists(MyFolder)
```

FTP Skoroszyt zawierający funkcje `FileExists` oraz `PathExists` (*Funkcje plikowe.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

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.ClearContents
    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 2 GB
         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
```

Funkcja `FileLen` języka VBA wykorzystuje dane typu `Long`, dlatego w przypadku plików większych niż 2 GB będzie zwracała niepoprawny rozmiar pliku (liczbę ujemną). Kod procedury sprawdza, czy funkcja `FileLen` zwróciła wartość ujemną, a 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.

FTP Skoroszyt z bardziej zaawansowaną wersją tej procedury, umożliwiającą wybranie katalogu za pomocą okna dialogowego (*Lista plików.xlsm*), znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

Pierwszym argumentem funkcji `Dir` może być nazwa pliku 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 zwracane są nazwy plików posiadających czteroznakowe rozszerzenie zaczynające się od liter XL. Mogą to być na przykład pliki o rozszerzeniach *.xlsx*, *.xltx* 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 11.2 zamieszczono zestawienie stałych, które mogą być argumentami funkcji `Dir`.

TABELA 11.2. Zestawienie stałych, które mogą być argumentami funkcji Dir

Nazwa stałej	Wartość	Opis
`vbNormal`	0	Plik bez atrybutów, jest to domyślne ustawienie atrybutów dla tej funkcji
`vbReadOnly`	1	Pliki tylko do odczytu
`vbHidden`	2	Pliki ukryte
`vbSystem`	4	Pliki systemowe
`vbVolume`	8	Etykieta woluminu. Jeżeli w wywołaniu funkcji został użyty jakikolwiek inny atrybut, ten atrybut będzie ignorowany
`vbDirectory`	16	Katalogi. Co ciekawe, ten atrybut… po prostu nie działa. Wywołanie funkcji `Dir` z atrybutem `vbDirectory` nie zwraca nazw podkatalogów

Jeżeli używasz funkcji `Dir` do przechodzenia w pętli przez kolejne pliki i wywoływania 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 2 GB
                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\")
```

FTP Skoroszyt z tym przykładem (*Lista plików - rekurencja.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

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, ściśle 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* (narzędzia/odwołania), a następnie w oknie dialogowym *References* zaznaczyć odpowiednią opcję. 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* (automatyczne wyświetlanie składowych obiektu) edytora VBE, który znakomicie ułatwia wpisywanie oraz identyfikację odpowiednich właściwości i metod obiektów. Co więcej, dzięki temu możesz również skorzystać z przeglądarki obiektów i sprawdzić informacje na temat danego obiektu. Aby to zrobić, wystarczy po wejściu do VBE nacisnąć klawisz *F2*.

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(pname) 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 dysków

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.

FTP Skoroszyt z tym przykładem (*Pokaż informację o napędach.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

```
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
      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 7., „Przykłady i techniki programowania w języku VBA", znajdziesz opis innej metody pobierania informacji o napędach dyskowych, wykorzystującej funkcje API systemu Windows.

Pakowanie i rozpakowywanie plików

Prawdopodobnie najczęściej spotykanym formatem spakowanych plików jest popularny ZIP. Nawet dokumenty programu Excel 2007 (i wersji późniejszych) są zapisywane w tym formacie (choć nie używają rozszerzenia *.zip*). Archiwa 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, więc zapisanie ich dodatkowo w formacie ZIP w niewielkim stopniu wpłynie na ich rozmiar.

FTP Skoroszyty z przykładami (*Pakowanie plików ZIP.xlsm* oraz *Rozpakowywanie plików ZIP.xlsm*) znajdziesz na serwerze FTP wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/e19pvw.zip*).

Pakowanie plików do formatu ZIP

Kod zamieszczony poniżej ilustruje sposób tworzenia spakowanego archiwum 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 tworzone jest archiwum w formacie ZIP o nazwie *compressed.zip* zawierające 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")

   'Kopiuj 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
           DoEvents
           Application.Wait (Now + TimeValue("0:00:01"))
        Loop
        Application.StatusBar = "Plik " & i & " z " & Ubound(Filenames)
    Next i
    If MsgBox(FileCount & " plików zostało spakowanych do pliku ZIP: " & _
        vbNewLine & FileNameZip & vbNewLine & vbNewLine & _
        "Wyświetlić plik ZIP?", vbQuestion + vbYesNo) = vbYes Then _
        Shell "Explorer.exe /e," & FileNameZip, vbNormalFocus
End Sub
```

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 sekunda 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 kopiowania 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 odpowiedni komunikat. Naciśnięcie przycisku *Tak* powoduje uruchomienie programu Windows Explorer, w którym zostanie wyświetlona zawartość archiwum.

Procedura `ZipFiles` przedstawiona na przykładzie na poprzedniej stronie 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. Z pewnością nie zastąpi to wbudowanych narzędzi do kompresowania, dostępnych w systemie Windows, ale jest ciekawą demonstracją tego, co możesz zrobić za pomocą języka VBA.

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.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

V

W

X

Y

Z

VBA dla Excela: niemożliwe staje się proste!

Możliwości arkusza MS Excel są imponujące. Uważa się, że nikt na świecie nie zna jego wszystkich funkcji i narzędzi. Mimo to wiele osób odczuwa potrzebę wykroczenia poza standardowo dostępne opcje Excela. Często jest to automatyzacja jakiegoś żmudnego zadania albo utworzenie narzędzia do specjalnych celów. I właśnie dla takich użytkowników przygotowano VBA — język, w którym można napisać prosty program do przetwarzania danych, własny dodatek do Excela albo nawet profesjonalną aplikację. A do tego programowania w VBA można nauczyć się błyskawicznie!

Ta książka jest jedynym w swoim rodzaju źródłem wiedzy o VBA i jego wykorzystywaniu do najróżniejszych zadań. Bardzo złożone zagadnienia zaprezentowano tu w prosty i przystępny sposób, koncentrując się na praktycznym wykorzystywaniu kodu VBA. Znalazło się tutaj wiele wskazówek, porad i ciekawych rozwiązań, które w połączeniu z pełnymi kodami programów i przykładowymi skoroszytami umożliwiają szybkie zrozumienie poszczególnych kwestii. Pokazano krok po kroku, jak pisać programy do automatyzacji wielu zadań w Excelu. Książka zawiera wszystkie wiadomości potrzebne do nauki rejestrowania prostych makr, pisania kodu, a także do tworzenia wyrafinowanych narzędzi i aplikacji.

W tej książce między innymi:

- **dynamiczna praca ze skoroszytami i z arkuszami**
- **automatyzacja operacji na tabelach przestawnych i wykresach**
- **przetwarzanie danych z plików i innych źródeł**
- **seryjna korespondencja elektroniczna bezpośrednio z Excela**
- **projektowanie elementów interfejsu i korzystanie ze Wstążki**
- **tworzenie i udostępnianie własnych dodatków do Excela**

MICHAEL ALEXANDER specjalizuje się w zaawansowanej analizie biznesowej z użyciem MS Access i MS Excel. Napisał kilka książek o tej tematyce. Uzyskał certyfikat MCAD (Microsoft Certified Application Developer) oraz tytuł MVP (Most Valuable Professional).

DICK KUSLEIKA jest wielokrotnym zdobywcą tytułu MVP. Od ponad 20 lat zajmuje się pakietem Microsoft Office, a swoją ogromną wiedzą dzieli się na forach internetowych, konferencjach i w książkach. Opracowuje systemy bazujące na programach Access i Excel oraz szkoli w zakresie zaawansowanej obsługi Office.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-6634-3

INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 99,00 zł
WILEY