helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

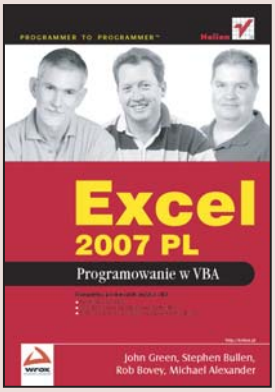
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

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 makropolecenia 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

2

Obiekt Application

Ten rozdział przedstawia zakres funkcjonalności programu Excel. Opisane w nim funkcjonalności nie muszą być koniecznie 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 `Appli ↪cation` 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 kglobalnych `<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
```

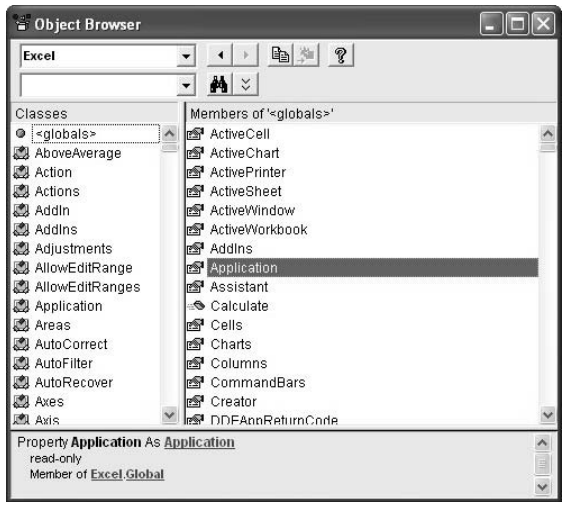
Jednak trzeba zachować ostrożność. Łatwo założyć, że często używane właściwości `Applica ↪tion`, takie jak `ScreenUpdating`, należą do grupy `<globals>`, podczas gdy tak nie jest. Następujący kod jest poprawny:

```
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`. Zwykła składnia 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` skoroszytu 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

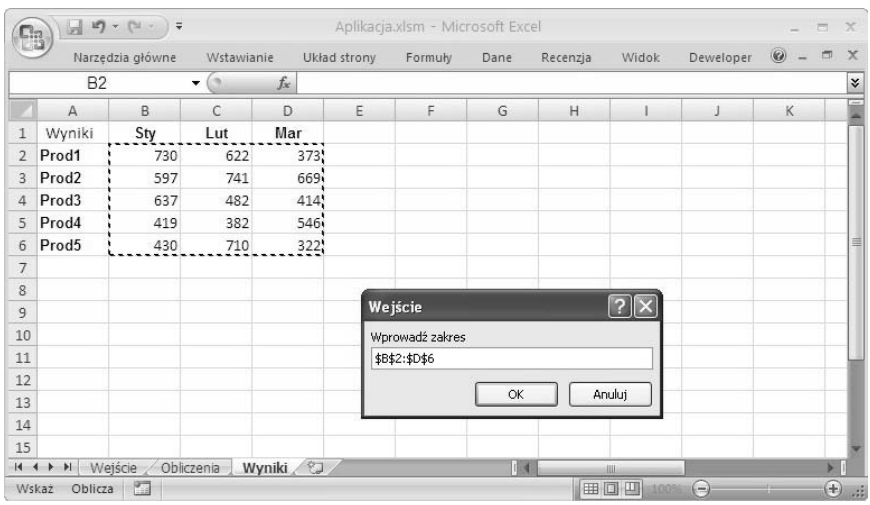
Wartość typu	Znaczenie
0	Formuła
1	Liczba
2	Tekst (`String`)
4	Wartość logiczna (`True` lub `False`)
8	Wskaźnik do komórki, jako obiekt `Range`
16	Wartość błędu, taka jak `#N/A`
64	Tablica wartości

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 przypisana 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 uruchamiać 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 `DolDziesiec` do klawisza *strzałka w dół*. Gdy `Przy` `↪piszDol` zostanie uruchomione, klawisz strzałki w dół będzie uruchamiać makro `Dol` `↪Dziesiec` i przenosić wskaźnik komórki o dziesięć wierszy w dół, a nie o jeden.

Listing 2.6

```
Sub PrzypiszDol()
   Application.OnKey "{Down}", "DolDziesiec"
End Sub

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

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

`WyczyscDol` 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 wariancie 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 (chociaż 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 PrzykładKonkatenacji1()
   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 PrzykładKonkatenacji2()
   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:

```
DataPoczatkowa = 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"))
```

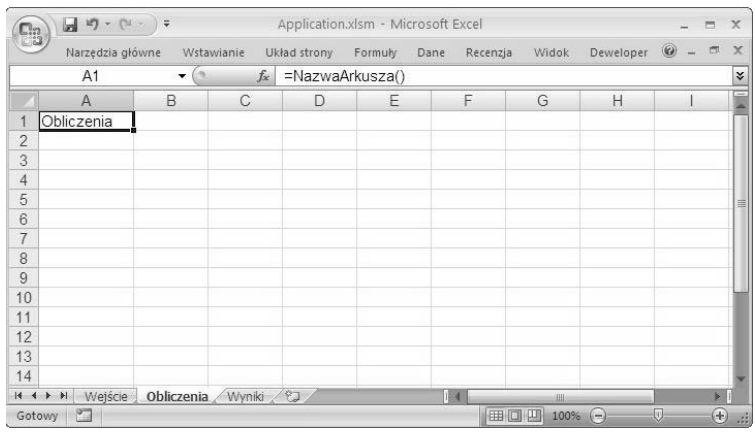
Kompletna lista funkcji arkusza bezpośrednio dostępnych w VBA znajduje się w opisie obiektu WorksheetFunction w dodatku A.

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.