

## 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

# MS Office 2000 i 2002/XP. Tworzenie własnych aplikacji w VBA

Autor: Maciej Łoś  
ISBN: 83-7197-878-2  
Format: B5, stron: 138



Pakiet Microsoft Office to nie tylko zaawansowany zestaw aplikacji biurowych, ale także platforma, na podstawie której programiści Visual Basic for Applications mogą pisać własne programy. Chociaż możliwości dostosowania poszczególnych aplikacji Office'a do specyficznych wymagań użytkownika bez konieczności pisania kodu są spore, dopiero użycie VBA pozwala na tworzenie własnych kreatorów, dodatków i asystentów.

Książka „MS Office 2000 i 2002/XP. Tworzenie własnych aplikacji VBA” opisuje sposób tworzenia specyficznych dla MS Office dodatków z użyciem języka Visual Basic for Applications. Przeznaczona jest dla osób znających ten język programowania w stopniu podstawowym, chcących poznać tajniki programowania Worda, Excela i Accessa.

Opisano:

- Różne rodzaje plików, używanych przez MS Office
- Wykorzystanie technologii ActiveX
- Tworzenie własnych pasków narzędziowych
- Tworzenie asystentów
- Pisanie kreatorów Worda
- Integrację poszczególnych składników pakietu Office
- Pisanie własnej aplikacji wyszukującej pliki Excela
- Tworzenie bazy danych opartej na Accessie jako samodzielnej aplikacji

Programiści VBA znajdują w tej książce także wiele cennych wskazówek dotyczących efektywnego wykorzystania dostępnych typów danych oraz optymalizacji kodu. Autor prezentuje wiele fragmentów kodu, które możesz zastosować we własnych aplikacjach.

Dostosuj MS Office do swoich wymagań :

- Pisz własne kreatory i dodatki do aplikacji Office
- Naucz się integrować poszczególne programy pakietu
- Poznaj wewnętrzne mechanizmy działania Office'a
- Skorzystaj z ćwiczeń i fragmentów kodu dostępnych na płycie CD



# Spis treści

<b>Podziękowania</b> .....	<b>5</b>
<b>Zanim rozpoczniesz</b> .....	<b>7</b>
Do kogo jest adresowana ta książka? .....	7
O czym jest ta książka? .....	8
<b>Rozdział 1. Programowanie — informacje ogólne</b> .....	<b>9</b>
Projekt to podstawa .....	9
Czemu ma służyć aplikacja?.....	9
Kto będzie jej używał? .....	11
Wizualizacja.....	11
Przelanie pomysłu na papier.....	11
Podsumowanie .....	12
Najważniejsze informacje o VBA.....	12
Efektywne stosowanie typów.....	12
Prawda i fałsz (Boolean).....	15
Numeryczne typy danych .....	16
Tablice.....	18
Optymalizacja kodu .....	20
Odwołania do funkcji Windows API.....	31
Pliki pakietu Office .....	34
<b>Rozdział 2. Wspólne cechy i obiekty programów Office</b> .....	<b>43</b>
Obiekty Office 97 — omówienie.....	43
Dobrodziejstwo ActiveX.....	43
Obiekt Application.....	47
Obiekt Commandbars.....	51
Obiekt Assistant.....	59
Obiekt FileSearch .....	63
Obiekt DocumentProperties .....	64
Środowisko aplikacji.....	65
Funkcja Environ .....	65
Funkcja Shell.....	67
Instrukcja AppActivate .....	68
Funkcja DoEvents.....	69
Funkcja Timer .....	69
Instrukcja SendKeys .....	70
Automakra .....	71

<b>Rozdział 3. Programowanie — MS Word</b> .....	<b>77</b>
Tworzenie szablonu z własnym paskiem narzędzi.....	77
Tworzenie własnego kreatora.....	80
Współpraca z innymi aplikacjami Office.....	87
Współpraca z MS Access.....	88
Współpraca z MS Excel.....	90
<b>Rozdział 4. Programowanie — MS Excel</b> .....	<b>93</b>
Własne okno dialogowe poszukiwania plików.....	93
Dodajemy element do menu kontekstowego — nowa funkcja.....	94
Tworzenie aplikacji „krok po kroku”.....	96
Własne menu podręczne dla formularzy MS Word i MS Excel.....	100
Wskaż katalog w formularzu MS Word i MS Excel.....	102
Współpraca z innymi aplikacjami Office.....	104
<b>Rozdział 5. Programowanie — MS Access</b> .....	<b>109</b>
SQL jako motor baz danych.....	109
Tworzenie formularza wyszukującego dane.....	110
Uwagi o konstruowaniu zapytań w kodzie.....	114
Podformularze i podraporty.....	115
MS Access i Internet.....	116
Eksportowanie tabeli, kwerendy, formularza lub raportu w formacie HTML.....	116
Eksportowanie danych do serwera FTP w sieci Internet.....	116
Współpraca z XML.....	117
Tworzenie bazy danych jako samodzielnej aplikacji.....	117
Ograniczenia bazy danych.....	117
Parametryzacja bazy danych.....	118
<b>Zakończenie</b> .....	<b>131</b>
<b>Skorowidz</b> .....	<b>133</b>

## Rozdział 2.

# Wspólne cechy i obiekty programów Office

Choć tak wiele różni aplikacje Office od siebie, bo każda służy do czego innego, to przyglądając się im od strony informatycznej widzimy, że mają wiele wspólnego. Jak wiele mają wspólnego, tak też wiele je różni. Rozdział 2 jest poświęcony właśnie tym podobieństwom i różnicom.

## Obiekty Office 97 — omówienie

W tym rozdziale zajmę się obiektami pakietu Office dostępnymi w każdej z omawianych aplikacji. Są nimi:

- ◆ obiekt *Application*,
- ◆ obiekt *CommandBars*,
- ◆ obiekt *Assistant*,
- ◆ obiekt *FileSearch*,
- ◆ obiekt *DocumentProperties*.

Zanim jednak to uczynię, opowiem Ci pewną historię, a nazwę ją „Dobrodziejstwo ActiveX”.

## Dobrodziejstwo ActiveX

Domyślam się, że nieraz spotkałeś się z terminem ActiveX. Być może nawet dokładnie wiesz, co on oznacza, ale tytułem przypomnienia:

ActiveX to nazwa zbioru technologii, które pozwalają budować i używać obiektów za pomocą technologii COM<sup>1</sup> i DCOM<sup>2</sup>.

Na czym polega to dobrodziejstwo w pakiecie Office? Aplikacje takie jak Word, Excel czy Access są właśnie obiektami ActiveX. To oznacza, że możesz manipulować jedną aplikacją Office z poziomu drugiej. Ba, mało tego, możesz również uzyskać dostęp do obiektów tej aplikacji, niezależnie od tego, czy będziesz to robił lokalnie na swoim komputerze, czy w sieci.

Jak tego dokonać?

Jeśli chcesz uruchomić kod w jednej aplikacji Office, który będzie pracował z obiektami w innej aplikacji, musisz wykonać następujące kroki:

1. Ustaw odwołanie do biblioteki tej innej aplikacji w oknie dialogowym *Odwołania* (menu *Narzędzia*). Jeśli tego dokonasz, uzyskasz dostęp do tej aplikacji oraz wszystkich jej obiektów, właściwości i metod. Chcesz odwołać się do Worda, wskaż *Microsoft Word 8.0 Object Library*. Chcesz odwołać się do Excela, wskaż *Microsoft Excel 8.0 Object Library*. Chcesz odwołać się do MS Access, wskaż *Microsoft Access 8.0 Object Library*.



Pamiętaj, nie musisz wskazywać biblioteki typu do aplikacji, w której pracujesz, a tylko do aplikacji, do której chcesz uzyskać dostęp!

2. Zadeklaruj zmienną obiektową, która wskaże na obiekty w innej aplikacji określonego typu. Deklaracja zmiennej obiektowej za pomocą klauzuli *As Object* spowoduje utworzenie zmiennej zawierającej odwołanie do dowolnego obiektu. Dostęp do tego obiektu będzie odbywał się w sposób opóźniony, tzn. przez dowiązanie w trakcie wykonywania programu. Dlatego lepiej jest utworzyć zmienną obiektową określonego typu, tj. zadeklarować zmienną obiektową z określonym identyfikatorem klasy. Spowoduje to tzw. dowiązanie wczesne, dokonywane w trakcie kompilacji programu. Taką zmienną możesz na przykład utworzyć tak: `Dim AppX1 as Excel.Application` lub `Dim docWrd as Word.Document`.



Używając odwołań bez numeru wersji utworzysz obiekt w wersji wywoływanej aplikacji, takiej, jaka jest aktualnie zainstalowana na komputerze. Jeśli jesteś pewien, że jest to Office 97, to zadeklaruj zmienną obiektową z numerem wersji, np. `Dim AppX1 as Excel.Application.8`. I jeszcze jedna istotna rzecz — nie wszystkie obiekty, ich właściwości i metody będą dostępne w poprzednich wersjach aplikacji Office (z czasów, kiedy nabywało się każdą aplikację oddzielnie!)

<sup>1</sup> COM (ang. *Component Object Model*) — technologia umożliwiająca tworzenie obiektów oraz dostęp do nich z programu Windows.

<sup>2</sup> DCOM (ang. *Distributed Component Object Model*) — technologia umożliwiająca dystrybucję obiektów w sieci lokalnej i Internecie.

3. Użyj funkcji `CreateObject` z identyfikatorem klasy aplikacji (*OLE Programmatic Identifier*), by uzyskać dostęp do tej aplikacji lub jej obiektów. Przypisanie do zmiennej obiektowej wartości będzie wyglądało tak: `Dim AppX1 as Excel.Application` — deklaracja zmiennej, `Set AppX1 = CreateObject("Excel.Application.8")` — przypisanie wartości.
4. Użyj właściwości i metod obiektu przechowywanego w zmiennej. Na przykład, następująca instrukcja spowoduje utworzenie nowego skoroszytu: `AppX1.Workbooks.Add`.
5. Po zakończeniu prac z inną aplikacją, użyj metody `Quit`, by ją zamknąć. Oto jak tego dokonać: `AppX1.Quit`.

Listing 2.1 demonstruje, jak z programu MS Excel przenieść dane do programu MS Word.

**Listing 2.1.** *MS Word w MS Excel*

```
Const nazwanyZakres As String = "Dane_Sprzedaz"
Dim wdApp As Word.Application
Dim nazwaTabeli As String

Sub DoDziela()
    Dim tmpBoolean As Boolean
    nazwaTabeli = ThisWorkbook.Sheets(1).Range("A1")
    tmpBoolean = UtworzDokument()
    If tmpBoolean Then
        MsgBox "Dokument MS Word wraz z danymi z tabeli " & vbCrLf & _
            "" & nazwaTabeli & "" & vbCrLf & _
            "zostal pomyslnie utworzony!!!", vbInformation, "Sukces"
        wdApp.Visible = True
    Else
        MsgBox "Wystapily bledy podczas tworzenia dokumentu MS Word!!!" & vbCrLf & _
            "Dokument mogl:" & vbCrLf & _
            " - nie zostac utworzony," & vbCrLf & _
            " - zostac utworzony z bledami." & vbCrLf & _
            "Sprawdz, co moglo byc przyczyna bledu.", vbExclamation, "Niepowodzenie"
    End If
End Sub

Function UtworzDokument() As Boolean
    Dim wdDoc As Document
    Dim myRangeW As Word.Range
    Dim myRangeE As Range
    Dim myTable As Table
    Dim mySel As Word.Selection
    Dim ileWier As Long
    Dim ileKol As Long
    On Error GoTo Err_UtworzDokument
    UtworzDokument = True
    ileWier = ThisWorkbook.Sheets(1).Range(nazwanyZakres).Rows.Count
    ileKol = ThisWorkbook.Sheets(1).Range(nazwanyZakres).Columns.Count
    Set wdApp = CreateObject("Word.Application.8")
    wdApp.Visible = True
    Set wdDoc = wdApp.Documents.Add
```

```

With wdDoc
    Set myRangeW = .Range
    wdApp.Selection.TypeParagraph
    myRangeW.InsertBefore nazwaTabeli
    wdApp.Selection.EndKey Unit:=wdLine
    wdApp.Selection.TypeParagraph
    wdApp.Selection.Select
    Set myTable = .Tables.Add(wdApp.Selection.Range, ileWier, ileKol)
    If WstawDane_Tabeli(myTable, ileWier, ileKol) = False Then GoTo
        ↪Err_UtworzDokument
End With
Exit Function

Err_UtworzDokument:
    Err.Clear
    Set myTable = Nothing
    Set myRangeW = Nothing
    If Not wdDoc Is Nothing Then
        wdDoc.Close SaveChanges:=False
        Set wdDoc = Nothing
    End If
    If Not wdApp Is Nothing Then
        wdApp.Quit SaveChanges:=False
        Set wdApp = Nothing
    End If
    UtworzDokument = False
End Function

Function WstawDane_Tabeli(tabela As Table, ileWier As Long, ileKol As Long) As Boolean
    Dim kom As Cell
    Dim wier As Long, kol As Long
    Dim startWier As Long
    On Error GoTo Err_WstawDane_Tabeli
    WstawDane_Tabeli = True
    ThisWorkbook.Sheets(1).Range(nazwanyZakres).End(xlUp).Select
    ThisWorkbook.Sheets(1).Range(nazwanyZakres).End(xlToLeft).Select
    *pobierz nr wiersza początkowego
    startWier = CLng(Right(ActiveCell.Address, 1)) - 1
    For wier = 1 To ileWier
        For kol = 1 To ileKol
            tabela.Cell(wier, kol).Range.Text = ThisWorkbook.Sheets(1).Cells(wier +
                ↪startWier, kol)
        Next kol
    Next wier
Exit Function

Err_WstawDane_Tabeli:
    Err.Clear
    WstawDane_Tabeli = False
End Function

```

---

**Przyjrzyj się dokładnie temu listingowi. Przede wszystkim chodzi o to, byś zwrócił uwagę na deklarację obiektów Range. Czy widzisz różnice?**



Na dysku CD-ROM, w katalogu *Przykłady\Cz\_II*, w pliku *CreateDoc.xls* znajduje się pełny kod, opisy i odpowiedź na pytanie.

## Obiekt *Application*

Jest to bardzo interesujący obiekt i będzie on często przeze mnie używany w przykładach. Obiekt *Application* odwołuje się do aktywnej aplikacji Office. Oznacza to, że podczas pracy w edytorze tekstów Word, obiekt *Application* odwołuje się właśnie do tego programu oraz zawiera wszystkie obiekty i kolekcje obiektów tej aplikacji. Jak należy to rozumieć? Możesz użyć tego obiektu do zastosowania metod lub ustawień właściwości dla całej aplikacji: `Application.SetOption "Show Status Bar", True` lub do wywołania metod lub ustawień właściwości dla dowolnego obiektu (kolekcji obiektów) danej aplikacji, tak jakbyś osobiście tych obiektów używał: `Docmd.OpenForm "Zamówienia"` (otwieranie formularza „Zamówienia”).

W instrukcji pierwszej jawnie posłużyłem się obiektem *Application*. Przykład drugi już nie zawierał jawnego wywołania obiektu *Application*, a mimo to podany kod jest poprawny i zostanie prawidłowo zinterpretowany przez aplikację Access. Jak to możliwe? Jeśli nie wiesz, przeczytaj jeszcze raz to, co dotychczas przeczytałeś. W zdaniu drugim zapewniam Cię, że obiekt *Application* posiada wszystkie obiekty danej aplikacji. Dla VBA nie musisz jawnie posługiwać się tym kwalifikatorem obiektu.



Oczywiście przykład drugi będzie poprawny wyłącznie w aplikacji Access, ponieważ inne programy z pakietu Office nie posiadają obiektu `DoCmd`. O obiektach poszczególnych aplikacji dowiesz się w kolejnych częściach książki.

## Instancyjność aplikacji

Rozpatrując aplikacje Office pod kątem wykorzystania ich w kodzie, a uściślając — pod kątem tworzenia nowej instancji klasy aplikacji, wyróżniamy aplikacje jedno- lub wielokrotnego wystąpienia, użycia. Na przykład MS Word jest aplikacją jednokrotnego wystąpienia. Oznacza to, że kod:

```
Dim wdApp As Word.Application
Set wdApp = New Word.Application
```

niezależnie od tego, jak wiele w danej chwili jest uruchomionych instancji (kopii) MS Word, zawsze będzie tworzył nową instancję aplikacji. Zaś aplikacja wielokrotnego wystąpienia (jaką jest na przykład Outlook), niezależnie od tego, jak wiele w kodzie będzie odwołań do instancji tej aplikacji, gdy ta aplikacja jest otwarta, zawsze będzie wskazywać na aplikację już otwartą. Przykładowo, jeżeli Outlook będzie otwarty, wszystkie odwołania do nowej instancji klasy (aplikacji) w przedstawionym kodzie (Listing 2.2):

**Listing 2.2.** *Odwołania do instancji aplikacji*

```
Dim OuApp_1 As Outlook.Application
Dim OuApp_2 As Outlook.Application
Dim OuApp_3 As Outlook.Application
```

```
Set OuApp_1 = New Outlook.Application
Set OuApp_2 = CreateObject("Outlook.Application")
Set OuApp_3 = GetObject(, "Outlook.Application")
```

zawsze będą wskazywać na tę samą instancję MS Outlook, już otwartą.

Tabela 2.1 przedstawia typy aplikacji MS Office.

**Tabela 2.1.** Podział aplikacji MS Office ze względu na krotność instancji

Aplikacja	Krotność instancji	Tworzy za każdym razem...
MS Access	jedno...	nową instancję aplikacji niezależnie od liczby już uruchomionych
MS Excel	jedno...	nową instancję aplikacji niezależnie od liczby już uruchomionych
MS Word	jedno...	nową instancję aplikacji niezależnie od liczby już uruchomionych
MS FrontPage	jedno...	nową instancję aplikacji niezależnie od liczby już uruchomionych
MS Outlook	wielo...	zmienną obiektową zwracającą instancję aplikacji już uruchomionej
MS PowerPoint	wielo...	zmienną obiektową zwracającą instancję aplikacji już uruchomionej

Jak zauważyłeś, do tworzenia nowej instancji klasy służy funkcja `CreateObject`. Są dwa najważniejsze zastosowania tej funkcji:

- ◆ Sprawdzenie, czy na danym komputerze jest zainstalowana aplikacja, której stworzysz kopię. Przykładowy kod (Listing 2.3):

**Listing 2.3.** Użycie funkcji `CreateObject` do sprawdzenia, czy na danym komputerze jest zainstalowana aplikacja

```
Dim AppObj As Object

Sub CzyJestAccess()
    Dim bBool As Boolean
    bBool = TworzKopieAplikacji("Access.Application")

    If Not bBool Then
        MsgBox "MS Access nie jest zainstalowany na tym komputerze.",
        vbInformation, "Przykro mi..."
        Exit Sub
    End If

    If bBool Then
        MsgBox "MS Access jest zainstalowany na tym komputerze.", vbInformation,
        "Jest!!!"
        With AppObj
            'tu mozesz dolaczyc odpowiedni kod, np.:
        End With
    End If
End Sub
```

```
Function TworzKopieAplikacji(AppName As String) As Boolean

    TworzKopieAplikacji = True

    On Error Resume Next
    Set AppObj = CreateObject(AppName)
    If Err = 429 Then 'nie zainstalowana aplikacja
        TworzKopieAplikacji = False
    End If

End Function
```



Na dysku CD-ROM w katalogu \Przyklady\Cz\_II\ znajdziesz plik *CzyJest.xls*, z kodem listingu 2.3. Zwróć też uwagę na deklarację obiektu aplikacji, odbywa się on w sposób opóźniony.

- ♦ Odwołanie się do aplikacji, której nie na lokalnym komputerze ale jest na innym komputerze w sieci. Przykładem takiego zastosowania może być potrzeba wydrukowania raportu z bazy MS Access umiejscowionej na serwerze z pliku MS Excel, nawet jeśli MS Access nie jest zainstalowany na komputerze, z którego wyszło żądanie drukowania. Wtedy użycie funkcji `CreateObject` musi odbyć się z dwoma argumentami: nazwą aplikacji i nazwą serwera.

#### Listing 2.4. *Drukowanie z serwera*

```
Sub DrukujZSerwera()
    Dim AccApp As Access.Application

    On Error Resume Next

    Set AccApp = CreateObject("Access.Application", "MojSerwer")

    If Err <> 0 Then
        MsgBox "Bledna nazwa serwera lub serwer nie funkcjonuje!", vbExclamation, "Problem"
        Exit Sub
    End If
    With AccApp
        'tu instrukcje dzialajace na obiekcie, drukujace raport
    End With
End Sub
```



Aby kod listingu 2.4 mógł zadziałać muszą być spełnione następujące warunki: zainstalowany MS Office XP (funkcja `CreateObject` z dwoma argumentami jest dostępna dopiero od wersji XP) oraz odpowiednio skonfigurowany DCOM na serwerze i lokalnym komputerze (na przykład programem `Dcomcnfg.exe`).

W tekście opisującym krotkość instancji posłużyłem się również funkcją `GetObject`. Możesz jej używać do zwrócenia informacji o tym, czy:

- ♦ jest już uruchomiona aplikacja i utworzyć zmienną obiektową wskazującą na tę aplikację. Listing 2.5 ilustruje przykładowe zastosowanie funkcji `GetObject`.

**Listing 2.5.** *Czy jest uruchomiona aplikacja?*

```

Sub CzyOtwartyAccess()
    Dim bBool As Boolean
    bBool = CzyOtwartaApp("Access.Application")
    'jesli nie otwarta aplikacja, tworz nowa kopie
    If Not bBool Then bBool = TworzKopieAplikacji("Access.Application")

    If bBool Then
        With AppObj
            'tu mozesz dolaczyc odpowiedni kod, np.:
        End With
    End If
End Sub

Function CzyOtwartaApp(AppName As String) As Boolean

    CzyOtwartaApp = True
    On Error Resume Next
    'sprawdz, czy jest otwarta
    Set AppObj = GetObject(, AppName)
    If Err = 429 Then 'nie uruchomiona aplikacja
        CzyOtwartaApp = False
    End If
End Function

```



Jeżeli masz uruchomionych kilka kopii aplikacji, którą chcesz zautomatyzować, nie masz gwarancji, która instancję zwróci Ci funkcja `GetObject`, ponieważ funkcja ta tworzy zmienną obiektową wskazującą na aktualnie otwartą aplikację jednokrotnego wystąpienia.

- ◆ oraz do otwarcia pliku przez odpowiednią aplikację MS Office. Przyjrzyj się listingowi 2.6.

**Listing 2.6.** *Otwarcie odpowiedniej aplikacji MS Office według rozszerzenia pliku.*

```

Sub Otworz()
    Dim bbool As Boolean
    Dim sciezkaDoPliku As String

    sciezkaDoPliku = "D:\test.doc"
    bbool = OtworzOdpowiedniaAplikacje(sciezkaDoPliku)
    If bbool Then
        MsgBox "Plik: '" & sciezkaDoPliku & "' został otwarty!" & vbCrLf & _
            "Za chwile zostanie zamknięty...", vbInformation, "Udalo sie"
        AppObj.Activate 'uaktywnij okno aplikacji przed zamknięciem
        AppObj.Close
        Set AppObj = Nothing
    End If
End Sub

Function OtworzOdpowiedniaAplikacje(sciezkaDoPliku As String) As Boolean

    OtworzOdpowiedniaAplikacje = True

```

```
On Error Resume Next

Set AppObj = GetObject(sciezkaDoPliku)
If Err <> 0 Then OtworzOdpowiedniaAplikacje = False

End Function
```

Instancyjność aplikacji nie jest łatwym zagadnieniem, ale mam nadzieję, że choć trochę przybliżyłem Ci tę tematykę.

## Obiekt Commandbars

Każda z aplikacji Office posiada własny zestaw pasków menu i narzędzi. Programując własne programy pod Office zapewne chciałbyś umieć programowo tworzyć własne paski narzędzi czy menu. W tej części tego rozdziału opiszę jak to zrobić.

Na początek słowo wyjaśnienia: jeśli używam słowa pasek bez kontekstu, to mam na myśli pasek menu lub pasek narzędzi lub pasek menu podręcznego. W każdym innym przypadku precyzuję, o jaki pasek mi chodzi.

### Podobieństwa i różnice

Wszystkie aplikacje Office używają tej samej technologii do tworzenia pasków menu i pasków narzędzi. Technologię tę firma Microsoft nazwała „Commandbars Object Model”. Są trzy rodzaje pasków: paski narzędzi, paski menu i menu podręczne. Menu podręczne są wyświetlane na trzy sposoby: jako przeciągnij-upuść z paska menu, jako podmenu i jako menu kontekstowe. Menu kontekstowe są wyświetlane, jeśli klikniemy prawym klawiszem myszy.

Ponieważ obiekty *CommandBars* są dostępne we wszystkich aplikacjach Office, możesz napisać kod, by manipulować paskami narzędzi czy menu we wszystkich aplikacjach Office. Każda aplikacja Office przechowuje informacje o swoich paskach w innym miejscu i, w niektórych przypadkach, w inny sposób. Podstawową różnicą jest to, jak i gdzie każdy program Office przechowuje paski utworzone przez użytkownika.



Kiedy poczynisz zmiany we wbudowanych paskach, informacje o tych zmianach zostaną zapisane w rejestrze Windows w profilu każdego użytkownika (jeśli Windows jest skonfigurowany na kilku użytkowników). Informacje o zakresie „widzialności” i „lokalizacji” są przechowywane w rejestrze Windows bez podziału na użytkowników. Ponadto każda aplikacja Office przechowuje swoje paski w dokumencie, w którym te paski zostały utworzone lub w konkretnym pliku. Oznacza to, że możesz kopiować paski tylko między dokumentami tego samego typu. Zatem, nie możesz skopiować paska z MS Word i używać go w MS Access.

### CommandBars aplikacji MS Access

Paski, które utworzysz w programie MS Access, są przechowywane w bazie danych, w której zostały utworzone. Jeśli chcesz utworzyć pasek, który będzie dostępny w innych

bazach danych, musisz utworzyć go w bibliotecznej bazie danych i ustawić odwołanie do tej bibliotecznej bazy danych w każdej bazie, w której dany pasek ma być dostępny. Informacje o wbudowanych paskach narzędzi przechowywane są w rejestrze.

## CommandBars aplikacji MS Excel

MS Excel umożliwia Ci przechowywanie pasków w danym skoroszycie lub w obszarze roboczym. Domyślnie paski Excel tworzone programowo są zapisywane w obszarze roboczym danego użytkownika. Jeśli Windows ma zdefiniowanych wielu użytkowników, jak na przykład w środowisku Windows NT, obszar roboczy użytkownika zapisywany jest w katalogu *C:\Winnt\* w pliku utworzonym od nazwy użytkownika — jak podano poniżej.

Jeśli nie zostały utworzone profile użytkowników, to obszar roboczy użytkownika zapisywany jest w katalogu *C:\Windows* w pliku o nazwie składającej się z następujących elementów:

- ◆ nazwy użytkownika podanej w oknie dialogowym rejestrującym produkty Office (na przykład: *Dorcia*),
- ◆ numeru wersji Office (na przykład: *8*),
- ◆ rozszerzenia tego pliku — *\*.xlb*.

W wyniku tego otrzymujemy nazwę pliku: *Dorcia8.xlb*.

Aby zapisać pasek w danym skoroszycie, musisz otworzyć okno dialogowe *Dostosuj...*, a następnie nacisnąć przycisk *Dołącz*. W polu po lewej stronie wybierz swój pasek i kliknij przycisk *Kopiuuj*.



Niestety, możesz kopiować paski do skoroszytu tylko „ręcznie”, niemożliwe jest skopiowanie paska z poziomu VBA. Podobnie rzecz ma się, jeśli chodzi o usuwanie.

Jeśli chcesz usunąć pasek z aktywnego skoroszytu, wybierz z okna po prawej stronie swój pasek, a następnie kliknij przycisk *Usuń*. Nie bój się dodawać i usuwać paski z aktywnego skoroszytu. Usuwasz tylko ich kopie z tego skoroszytu, a nie z obszaru roboczego.

Wszystkie paski zapisane w obszarze roboczym będą dostępne w każdym z otwieranych skoroszytów. Jeśli chcesz, by dany pasek nie był dostępny w innych skoroszytach, musisz programowo go usunąć lub ukryć. Paski z danego skoroszytu będą dostępne tylko w tym skoroszycie.



Jeżeli wybierzesz polecenie z paska zapisanego w obszarze roboczym, utworzonego w skoroszycie *X* (tu jest zapisany kod, który polecenie ma wykonać), to skoroszyt *X* zostanie automatycznie otwarty i będzie widoczny. Jeśli otworzysz skoroszyt zawierający pasek, który nie istnieje w Twoim obszarze roboczym, to zostanie on skopiowany do obszaru roboczego. Ta kopia nie jest usuwana po zamknięciu tego skoroszytu.

## CommandBars aplikacji MS Word

MS Word może przechowywać paski w kilku miejscach: w pliku *Normal.dot* (szablone globalnym), innym szablonie (\*.dot) lub w danym dokumencie (\*.doc). Różnica polega na tym, że paski przechowywane w pliku *Normal.dot* są dostępne dla każdego dokumentu, który tworzysz, nawet jeśli jest oparty na innym szablonie.



Nie jest dobrą praktyką przechowywać własne paski narzędzi w pliku szablonu globalnego, ponieważ może to spowodować rozrost tego pliku do dużych rozmiarów i MS Word będzie ładował się wolno. Ponadto, będąc administratorem sam dbałbyś o to, by plik *Normal.dot* był chroniony przed zapisem. Uchroniłoby Cię to przed groźnymi wirusami. Zatem przechowuj własne paski narzędzi w pliku zwykłego szablonu.

Pasek utworzony w szablonie będzie dostępny w każdym dokumencie opartym na tym szablonie. Jeśli zaś pasek został utworzony w dokumencie, to będzie on dostępny tylko po otwarciu tego dokumentu.

Domyślnie wszystkie paski, które utworzysz, będą zapisane w pliku *Normal.dot.*, chyba że zapiszesz je dla danego szablonu lub dokumentu w oknie *Dostosuj\Zapisz w...* Jeśli utworzysz własny pasek za pomocą VBA, musisz określić, gdzie ma zostać zapisany za pomocą właściwości *CustomizationContext* obiektu *Application*.

*CustomizationContext* — zwraca lub ustawia obiekt, szablon lub dokument, w którym zmieniasz paski. Właściwość ta jest zarówno do odczytu, jak i zapisu. Poniższy przykład (listing 2.7) dodaje kombinację klawiszy do polecenia *Zamknij* i zapisuje w pliku szablonu globalnego.

**Listing 2.7.** Dodawanie kombinacji klawiszy do polecenia w pasku

```
Sub AddKeyToControl()
    CustomizationContext = NormalTemplate
    KeyBindings.Add KeyCode:=BuildKeyCode(wdKeyControl, wdKeyAlt, wdKeyW), _
        KeyCategory:=wdKeyCategoryCommand, Command:="FileClose"
End Sub
```

Poniższy przykład (listing 2.8) dodaje przycisk *Informacje o wersji pliku* do paska *Standardowy* i zapisuje w pliku szablonu, na którym oparty jest aktywny dokument.

**Listing 2.8.** Dodawanie przycisku do paska

```
CustomizationContext = ActiveDocument.AttachedTemplate
Application.CommandBars("Standard").Controls.Add Type:=msoControlButton, _
    ID:=2522, Before:=8
```

Obiekt *CommandBars* reprezentuje kolekcję obiektów *CommandBar* (pasków narzędzi, menu) w aplikacji.

## Kolekcja pasków

Każda z aplikacji Office posiada nie jeden, a wiele pasków. Jeśli chciałbyś wyliczyć wszystkie paski dostępne w aplikacji, posłuż się obiektem *CommandBars*, który reprezentuje kolekcję pasków, by zwrócić pojedynczy obiekt *CommandBar*. Listing 2.9 przedstawia przykładową procedurę, która wyświetla w oknie *Instrukcje bezpośrednie* zarówno nazwę, jak i nazwę lokalną paska menu i paska narzędzi oraz wartość, która identyfikuje, czy pasek menu lub pasek narzędzi jest widoczny.

**Listing 2.9.** *Wyliczanie pasków*

```
Sub WyliczPaki()
    Dim cbar as CommandBar
    For Each cbar in Application.CommandBars
        Debug.Print cbar.Name, cbar.NameLocal, cbar.Visible
    Next
End Sub
```

## Dodawanie paska do kolekcji

Dodawanie paska do kolekcji jest bardzo proste. Wystarczy posłużyć się metodą *Add*. Listing przedstawia procedurę, która tworzy pasek narzędzi i nadaje mu nazwę *MojPasek*. Następnie wyświetla go jako pasek narzędzi pływający (nie mający stałej pozycji wśród pasków narzędzi). Czytaj uwagi w dalszej części książki, by zrozumieć, dlaczego wyliczam zarówno nazwę, jak i nazwę lokalną.

**Listing 2.10.** *Tworzenie paska*

```
Sub UtworzPasek()
    Set cbar1 = CommandBars.Add(Name:="MojPasek", Position:=msoBarFloating)
    cbar1.Visible = True
End Sub
```

Do utworzonego przed chwilą paska możesz dodać polecenia lub zmienić miejsce jego dokowania. Aby odwołać się do istniejącego paska, musisz użyć składni *CommandBars(index)*, gdzie *index* jest nazwą lub indeksem paska narzędzi. Następująca instrukcja dokuje pasek narzędzi nazwany *MojPasek* w dolnej części okna aplikacji: *CommandBars("MojPasek").Position = msoBarBottom*.



Możesz używać nazwy lub indeksu, by określić pasek menu lub pasek narzędzi z listy dostępnych pasków menu lub pasków narzędzi w kolekcji aplikacji. Aczkolwiek, musisz posłużyć się nazwą, by określić menu, menu podręczne lub podmenu. Jeśli dwa lub więcej z pasków użytkownika będą miały tę samą nazwę, obiekt *CommandBars(index)* zwróci pierwszą z nich. By upewnić się, że zwracasz poprawne menu lub podmenu, zlokalizuj kontrolkę, która przechowuje to menu.

## Pasek

Reprezentuje pojedynczy pasek narzędzi lub menu w kolekcji pasków narzędzi lub menu aplikacji.

Listing 2.11 przedstawia funkcję, która sprawdza, czy w kolekcji pasek znajduje się pasek o nazwie określonej przez procedurę, która wywołuje funkcję `Pasek`. Jeśli znajdzie, czyni go widocznym i chroni jego miejsce dokowania. Funkcja zwraca prawdę, jeżeli pasek został znaleziony i fałsz, jeżeli nie udało się znaleźć określonego paska w kolekcji pasek aplikacji.

**Listing 2.11.** Szukanie paska w kolekcji pasek

```
Function Pasek(cbName as String) As Boolean
Dim cb As CommandBar
znaleziony = False
For Each cb In CommandBars
    If cb.Name = cbName Then
        cb.Protection = msoBarNoChangeDock
        cb.Visible = True
        znaleziony = True
    End If
Next cb
Pasek = znaleziony
End Function
```

## Kopiowanie paska

Kopiowanie paska może być wykonane na dwa sposoby. Pierwszy polega na „ręcznym” skopiowaniu, drugi na skopiowaniu programowym. Aby programowo skopiować pasek, należy najpierw utworzyć nowy pasek, a następnie posłużyć się metodą `Copy`, by skopiować każde z poleceń z oryginalnego paska do nowego. Przykładową procedurę kopiowania paska *Formatowanie* ilustruje listing 2.12.

**Listing 2.12.** Kopiowanie paska

```
Sub Kopiuj()
If KopiujPasek("Formatting", "NewFormatting", True) Then
    MsgBox "Pasek został skopiowany", vbInformation, "Powodzenie"
End If
End Sub

Function KopiujPasek(orygNazwa As String, nowaNazwa As String, _
    Optional pokaz As Boolean = True) As Boolean
Dim cbOryg As CommandBar
Dim cbKopia As CommandBar
Dim cbcontrol As CommandBarControl
Dim cbType As Long
On Error GoTo BładKopiowania
Set cbOryg = CommandBars(orygNazwa)
cbType = cbOryg.Type
Select Case cbType
Case msoBarTypeMenuBar
    Set cbKopia = CommandBars.Add(Name:=nowaNazwa, Position:=msoBarMenuBar)
Case msoBarTypePopUp
    Set cbKopia = CommandBars.Add(Name:=nowaNazwa, Position:=msoBarTypePopUp)
Case Else
    Set cbKopia = CommandBars.Add(Name:=nowaNazwa)
End Select
```

```

For Each cbControl in cbOryg.Controls
    cbControl.Copy
Next cbControl

If pokaz Then
If cbKopia.Type = msoBarTypePopup Then
    cbKopia.ShowPopup
Else
    cbKopia.Visible = pokaz
End If
End If

Kopiuj Pasek = True
Exit Function

BładKopowania:
    Kopiuj Pasek = False
End Function

```

## Usuwanie paska

Usuwanie paska, tak jak dodawanie czy kopiowanie, możesz wykonać „ręcznie” lub napisać odpowiednią funkcję, procedurę. Jest jednak jedno małe „ale”. Paska typu *Pop-up* nie możesz usunąć „ręcznie” w oknie dialogowym *Dostosuj*. Jediną możliwością jest usunięcie takiego paska z kodu. Przykładową procedurę usuwającą pasek, który kopiowaliśmy w poprzednim rozdziale, przedstawia listing 2.13.

**Listing 2.13.** *Usuwanie paska*

```

Sub Usun()
    CommandBars("NewFormatting").Delete
End Sub

```

Jednak bardziej uniwersalnym rozwiązaniem będzie kod przedstawiony w listingu 2.14.

**Listing 2.14.** *Usuwanie paska z programem obsługi błędów*

```

Function UsunPasek(nazwaPaska As String) As Boolean
On Error Resume Next
Application.CommandBars(nazwaPaska).Delete
UsunPasek = Iif(Err<>0, False, True)
End Function

```

Jeżeli dobrze przyjrzałeś się listingowi, to zapewne zauważyłeś, że umieściłem tam instrukcję `On Error Resume Next`, która ma za zadanie przechwycić i obsłużyć błąd, kiedy podana nazwa nie jest nazwą paska lub pasek nie ma w kolekcji pasek. Taka sytuacja może wystąpić również, jeżeli próbujesz usunąć któryś z wbudowanych pasek narzędzi, na przykład *Standardowy*.

## Blokowanie paska przed modyfikacjami

Czasami trzeba zablokować pasek przed modyfikacjami ze strony innych użytkowników. Zadanie to nie jest takie proste, bo użytkownicy mogą dotrzeć do pasek na trzy sposoby:

1. używając polecenia *Widok\Paski Narzędzi\Dostosuj*,
2. klikając prawym klawiszem myszy, gdy wskaźnik znajdzie się na szarym polu pasków,
3. klikając prawym klawiszem myszy, gdy wskaźnik znajdzie się na którymś z pasków.

Na szczęście jest rozwiązanie.

**Po pierwsze:** ukryj pasek menu lub wyłącz polecenie *Widok\Paski Narzędzi*, wtedy będziesz miał pewność, że użytkownik nie będzie miał dostępu do modyfikacji pasków pierwszą z wymienionych metod.

**Po drugie:** użyj poniższego polecenia, by zabrać prawa do przeglądania i modyfikowania niestandardowych pasków narzędzi za pomocą 2. i 3. metody: `CommandBars ("Toolbar List").Enabled = False`.

### Dodawanie poleceń do paska

Poniższy przykład (listing 2.15) dodaje nowy przycisk w menu *Narzędzia*. W procedurze kliknięcia na tym przycisku umieszcza funkcję, która wyświetli komunikat.

**Listing 2.15.** Dodanie polecenia z kodu i przypisanie mu procedury

```
Sub UtworzPolecenie()
    Set newItem = CommandBars("Tools").Controls.Add(Type:=msoControlButton)
With newItem
    .BeginGroup = True
    .Caption = "Komunikat"
    .FaceID = 52 'swinka
    .OnAction = "Komunikat"
End With
End Sub

Sub Komunikat()
    MsgBox "To jest komunikat spod znaku swinki",vbCritical,"Komunikat"
End Sub
```



Używaj nazwy, a nie nazwy lokalnej, by mieć pewność, że zwracasz prawidłowe menu.

### Pokazywanie i ukrywanie poleceń

W zależności od tego, jak bardzo rozbudowana będzie Twoja aplikacja, będziesz musiał utworzyć jakiś uniwersalny pasek, w którym umieścisz tak wiele poleceń, jak będzie konieczne. Jednakże nie zawsze wszystkie polecenia muszą być w danym momencie widoczne. Przykładowo, na jednym formularzu użytkownik będzie miał dostępnych tylko pięć elementów paska, a na drugim już siedem. Taka sytuacja zaistnieje tylko wtedy, gdy każdy z formularzy będzie służyć do czego innego. Czy jednak zawsze trzeba ukrywać polecenia? Czasami wystarczy uniemożliwić dostęp do nich w inny sposób, ale o tym już w dalszej części książki.

Oto prosta instrukcja zmieniająca stan widzialności polecenia na pasku:

```
CommandBars("Worksheet Menu Bar").Controls("Plik").Visible = True
```

Jeśli często w aplikacji będziesz pokazywał lub ukrywał jakieś polecenie, lepiej jest napisać procedurę, która będzie pobierała 2 argumenty: etykietę polecenia i stan. Przykładowa procedura dla aplikacji Excel mogłaby wyglądać tak, jak przedstawia listing 2.16.

**Listing 2.16.** Procedura pokazywania i ukrywania poleceń paska

```
Sub PokazPoleceniePaska(etykieta As String, Optional stan As Boolean = True)
    Dim cntrl As CommandBarControl
    For Each cntrl In Application.CommandBars("Worksheet Menu Bar").Controls
        If cntrl.Caption = etykieta Then
            cntrl.Visible = stan
        Exit For
    End If
Next
End Sub
```



Jeśli zechcesz zmienić pasek menu w innej aplikacji Office, użyj Menu Bar zamiast Worksheet Menu Bar.

Jeśli powyższa procedura (listing 2.16) zostanie wywołana tylko z argumentem etykieta, to domyślnie pokaże dane polecenie. Parametr stan nie jest wymagany. Aby ukryć jakieś polecenie, po etykiecie podaj również stan = False. Następujący kod: PokazPoleceniePaska "&Plik", False ukrywa polecenie *Plik*, a kolejny: PokazPoleceniePaska "&Plik" pokazuje wcześniej ukryte polecenie.



Do ustawiania domyślnych parametrów wywoływanej procedury lub funkcji służy słowo kluczowe *Optional*. Przyjrzyj się dokładnie procedurze PokazPoleceniePaska, a zauważysz, że stan zadeklarowałem jako wartość typu Boolean i od razu ustawiłem na True. Jeżeli w parametrach, które pobiera funkcja, znajdzie się wartość typu Variant, to nie możesz jej przypisać wartości domyślnej (ze względu na to, że nie wiesz jak konkretnie wartość zostanie przekazana do funkcji). Aby sprawdzić, czy procedura została wywołana z wartością, która nie jest wymagalna, musisz użyć funkcji *IsMissing*('tutaj wartosc typu variant'), by zwrócić prawdę (podany argument) lub fałsz. Następnie będziesz mógł przypisać jakąś wartość.

## Włączanie i wyłączanie poleceń

Czy pamiętasz jak pisałem, że nie zawsze trzeba ukrywać polecenia i czasami wystarczy wyłączyć to polecenie? Do włączania lub wyłączania poleceń paska służy właściwość *Enabled*. Możesz ustawić dowolny stan dla danego polecenia na przykład tak:

```
CommandBars("Worksheet Menu Bar").Controls("Plik").Enabled = True
```

lub posłuż się procedurą podobną do PokazPoleceniePaska w poprzednim rozdziale, tylko zamiast właściwości *Visible*, użyj *Enabled*. Proste? Mam nadzieję, że tak. Zapewne zauważyłeś, że zajmowałem się dotychczas tylko poleceniami głównymi. Jak dotrzeć do poleceń, umieszczonych wewnątrz tych poleceń? Zastanów się i spróbuj napisać odpowiedni kod.

## Obiekt Assistant

Obiekt *Assistant* to popularny w pakiecie Asystent. Wiadomo, że generalnie podpowiada, co mamy zrobić, przy okazji posługując się ciekawymi efektami graficznymi. Asystent to doskonałe narzędzie do Twoich aplikacji. Można go tak zaprogramować, by nie tylko wyświetlał podpowiedzi, ale również współdziałał z elementami interfejsu aplikacji. Możesz, na przykład, za pomocą Asystenta uruchamiać polecenia z paska lub wstawiać grafikę do dokumentu. Ciekawą sprawą jest to, że cały proces zachodzi bez zamykania okna aplikacji.

Asystenta możesz zaprogramować tak, by wykonywał wszystkie czynności jakie chcesz. Do tego celu musisz ustawić zmienną obiektową, która pozwoli na manipulowanie wszystkimi właściwościami i metodami Asystenta. Możesz na przykład uczynić Asystenta widocznym, przenieść go w inne miejsce na ekranie, określić animację, którą chcesz uruchomić i wyświetlić *Assistant balloons* (słowa tego nie tłumaczymy, ale dla ułatwienia będą posługiwał się określeniem podpowiedź, podpowiedzi) zawierające określony przez Ciebie tekst i kontrolki (np. pola opcji).

Obiekt *Assistant* reprezentuje Asystenta pakietu Office. Obiekt *Assistant* nie jest kolekcją obiektów *Assistant* i tylko jeden *Assistant* może być aktywny w danym czasie.

Popularny Asystent może przyjmować jedną z wielu zaprogramowanych postaci. Postaci te ukryte są w plikach o rozszerzeniu *\*.act* i najczęściej znajdują się w katalogu *C:\Program Files\Microsoft Office\Office\Actors*. Tabela 2.2 przedstawia nazwy poszczególnych plików dla każdej z postaci Asystenta.

**Tabela 2.2.** Nazwy plików Asystenta

Postać (nazwa oryginalna)	Postać (polski odpowiednik)	Nazwa pliku
Office Logo	Logo	<i>Logo.act</i>
PowerPup	Super Pies	<i>Powerpup.act</i>
The Genius	Geniusz	<i>Genius.act</i>
Hoverbot	Poduszkobot	<i>Hoverbot.act</i>
Scribble	Świstek	<i>Scribble.act</i>
The Dot	Punkt	<i>Dot.act</i>
Clippit	Spinacz	<i>Clippit.act</i>
Mother Nature	Matka Natura	<i>MNature.act</i>
Will	Will	<i>Will.act</i>

Posługując się nazwą pliku możesz określić, jakiego Asystenta chcesz w danej chwili wyświetlić. Służy do tego właściwość *FileName* obiektu *Assistant*. Musisz być jednak pewien, że na komputerze użytkownika zainstalowane są składniki odpowiadające za wyświetlanie odpowiedniej postaci Asystenta. W innym przypadku będzie występował błąd. Ponadto, każdy z użytkowników personalizuje Office, co oznacza, że tylko dana postać Asystenta może mu odpowiadać.

Aby wyświetlić aktualną (używaną przez użytkownika) postać Asystenta, wykonaj ten kod:

```
With Assistant
    .Visible = True
End With
```

Samo wyświetlenie Asystenta jeszcze nic nie oznacza. Nic nie będzie podpowiadał. Do komunikacji z użytkownikiem poprzez Asystenta będziesz potrzebował obiektu *Balloon*.



Różnice w programowaniu Asystenta dla pakietów Office: do pakietu Office XP Asystent jest albo widoczny i dostępny, albo niewidoczny i niedostępny, ale nigdy nie można go całkowicie wyłączyć. Od pakietu Office XP Asystent ma nową właściwość *On* i może być albo włączony, albo wyłączony. Ta właściwość decyduje o tym, czy Asystent jest dostępny w całym pakiecie. Dlatego, podczas programowania Asystenta w pakiecie Office XP, zwróć szczególną uwagę na to, w jakim stanie znajdował się Asystent zanim zacząłeś nim manipulować w kodzie. Najlepiej przypisz do zmiennej jego stan przed wywołaniem, a po zakończeniu prac z Asystentem, ustaw stan Asystenta na tę zmienną.

Przyjrzyj się listingowi 2.17.

#### Listing 2.17. Korzystanie z Asystenta

```
Sub Asys()
    Dim czyWlaczony As Balloon
    Dim czyWidoczny As Balloon
    With Assistant
        czyWlaczony = .On
        czyWidoczny = .Visible
    End With

    'przywrac ustawienia Asystenta
    With Assistant
        .On = czyWlaczony
        .Visible = czyWidoczny
    End With
```

## Balloon

Jak już wspomniałem, obiekt *Balloon* Asystenta służy do komunikacji z użytkownikiem Twojej aplikacji. Jest on zaprojektowany tak, byś mógł łatwo go wywoływać, ale nie ma na celu zastąpienia okien dialogowych. Powiedzmy sobie, że jest tylko środkiem pomocniczym.

Obiekt *Balloon* reprezentuje podpowiedź wyświetlaną za pomocą Asystenta, z nagłówkiem i wiadomością tekstową. Może on zawierać kontrolki takie jak: pola wyboru (*CheckBoxes*) lub etykiety (*Labels*). *Balloon* nie jest kolekcją *Balloons*, ponieważ jest pojedynczym obiektem. W jednym czasie może być widoczny tylko jeden *Balloon*, aczkolwiek możliwe jest zdefiniowanie wielu *Balloons* i wywołanie dowolnego z nich wtedy, kiedy będzie potrzebny. Do utworzenia nowego obiektu *Balloon* służy właściwość *NewBalloon*.

Listing 2.18 przedstawia przykład tworzenia pojedynczego obiektu *Balloon*.

**Listing 2.18.** Tworzenie pojedynczego obiektu *Balloon*

```
Sub PojedPodpow()
Dim podpowiedz As Balloon

Set podpowiedz = Assistant.NewBalloon
With podpowiedz
    .Heading = "Oto najprostszy przyklad tworzenia podpowiedzi."
    .Text = "Dim podpowiedz As Balloon" & vbCrLf & _
        "Set podpowiedz = Assistant.NewBalloon" & vbCrLf & _
        "With podpowiedz" & vbCrLf & _
        ".Heading = ""Oto najprostszy przyklad tworzenia podpowiedzi.""
        & vbCrLf & _
        ".Text = ""Jest to najprostszy przyklad tworzenia podpowiedzi.""
        & vbCrLf & _
        ".Button = msoButtonSetOK" & vbCrLf & _
        ".Show" & vbCrLf & _
        "End With"
    .Button = msoButtonSetOK
    .Show
End With

End Sub
```

Listing 2.19 przedstawia przykład tworzenia wielu obiektów *Balloon*.

**Listing 2.19.** Przykład tworzenia wielu obiektów *Balloon*

```
Sub KilkaPodpowiedziAsystenta()
Dim podpowiedz1 As Balloon
Dim podpowiedz2 As Balloon
Dim podpowiedz3 As Balloon
Dim odp As Long

Set podpowiedz1 = Assistant.NewBalloon
With podpowiedz1
    .Heading = "Podpowiedz pierwsza."
    .Text = "Czy chcesz zobaczyc kolejna podpowiedz?"
    .Button = msoButtonSetOkCancel
End With

Set podpowiedz2 = Assistant.NewBalloon
With podpowiedz2
    .Heading = "Podpowiedz druga."
    .Text = "Czy chcesz zobaczyc kolejna podpowiedz?"
    .Button = msoButtonSetOkCancel
End With

Set podpowiedz3 = Assistant.NewBalloon
With podpowiedz3
    .Heading = "Podpowiedz trzecia."
    .Text = "To juz koniec podpowiedzi."
    .Button = msoButtonSetOK
End With
```

```

odp = odpowiedz1.Show
If odp = msoBalloonButtonOK Then
    odp = odpowiedz2.Show
    If odp = msoBalloonButtonOK Then
        odpowiedz3.Show
    End If
End If

End Sub

```

Listing 2.20 przedstawia trzy procedury. Pierwsza wywołuje procedurę tworzącą nową odpowiedź z kilkoma opcjami wyboru i wyświetla odpowiedź. Druga jest powołana do tego, by dowiedzieć się, jakiego wyboru dokonał użytkownik, a trzecia wyświetla, również przez Asystenta, odpowiedź.

**Listing 2.20.** *Kompleksowa obsługa Asystenta*

```

Sub WywolajOpcje()
    AsystentDajOpcje "tytul", "tekst", 1, "wybor1", "wybor2", "wybor3"
End Sub

Sub AsystentDajOpcje(tytul As String, tekst As String, opcjaPrawidlowa As Long,
    ►ParamArray wybory() As Variant)
    Dim odpowiedz As Balloon
    Dim i As Long, x As Long
    Dim wybor As String

    Set odpowiedz = Assistant.NewBalloon
    With odpowiedz
        .BalloonType = msoBalloonTypeBullets
        .Button = msoButtonSetOkCancel
        .Heading = tytul
        .Text = tekst
        'pobierz ile przekazanych parametrow
        For i = LBound(wybory) To UBound(wybory)
            wybor = wybory(i)
            .CheckBoxes(i + 1).Text = wybor
        Next i
        x = .Show
    End With

    JakaOpcja odpowiedz, x, opcjaPrawidlowa

End Sub

Sub JakaOpcja(bal As Balloon, wyb As Long, praw As Long)
    Dim i As Long
    Dim tmpBool As Boolean

    If wyb = -1 Then
        tmpBool = bal.CheckBoxes(praw).Checked
        If tmpBool = True Then
            AsystentDajOdpowiedz "Super", "Brawo, prawidlowa odpowiedz"
        End If
    End If
End Sub

```

```
Else
    AsystentDajOdpowiedz "Uuupps...", "Przykro mi, to nie jest prawidłowa
    ↳ odpowiedz"
End If
End If
End Sub
```

## Obiekt FileSearch

Obiekt *FileSearch* jest programowym odzwierciedleniem okna dialogowego *Otwórz* dostępnego z menu *Plik*. Oznacza to, że masz do dyspozycji wszystkie metody i właściwości, które są normalnie dostępne w powołanym oknie. Co ciekawe, wcale nie musisz używać tego okna, a wręcz możesz skonstruować własne.

Obiekt *FileSearch* posiada dwie metody i kilka właściwości, których możesz użyć do zbudowania własnego okna do poszukiwania plików. Musisz użyć właściwości *FileSearch*, by zwrócić odwołanie do tego obiektu. Następnie posłuż się metodą *NewSearch*, by zresetować (wyczyścić) wszystkie kryteria poprzedniego poszukiwania.

Metoda *NewSearch* nie czyści kryterium *LookIn*, czyli ścieżki, która jest przeszukiwana. Aby ją wyczyścić, a raczej ustawić na nowo, po prostu przypisz jej inną wartość.

Listing 2.21 przedstawia przykład kodu, który powoduje wyświetlenie wszystkich plików, które w nazwie mają słowo „office”, znalezionych w określonej lokalizacji. W przypadku niepowodzenia wyświetlony zostanie komunikat, że nie znaleziono określonych plików.

**Listing 2.21.** Poszukiwanie plików z kodu

```
Sub SzukajPlikow()
    Dim kom As String
    With Application.FileSearch
        .NewSearch
        .LookIn = "C:\\"
        .SearchSubFolders = True
        .FileName = "office"
        .FileType = msoFileTypeAllFiles
    End With
    If .Execute() > 0 Then
        Select Case .FoundFiles.Count

            Case 1
                kom = "Znalazlem " & .FoundFiles.Count & _
                    " plik wedlug zadanych kryteriow poszukiwania"
                MsgBox kom, vbInformation, "Komunikat"

            Case 2, 3, 4
                kom = "Znalazlem " & .FoundFiles.Count & _
                    " pliki wedlug zadanych kryteriow poszukiwania"
                MsgBox kom, vbInformation, "Komunikat"
```

```

Case Else
    kom = "Znalazlem " & .FoundFiles.Count & _
        " plikow wedlug zadanych kryteriow poszukiwania"
    MsgBox kom, vbInformation, "Komunikat"

End Select

For i = 1 To .FoundFiles.Count
    MsgBox .FoundFiles(i), vbInformation, i
Next i
Else
    MsgBox "Nie znalazlem plikow wedlug zadanych" & _
        " kryteriow poszukiwania", vbExclamation, "Brak"

End If
End With
End Sub

```



Na dysku CD-ROM w katalogu *Przyklady\Cz. II* znajdziesz przykład własnego okienka do poszukiwania plików, plik *FileSearch.xls*.

## Obiekt DocumentProperties

Każdy plik pakietu Office ma wbudowane pewne właściwości. Możesz dodawać również własne właściwości, czy to manualnie, czy to z kodu. Używając tych właściwości możesz tworzyć, przechowywać i śledzić informacje o dokumentach pakietu Office. Są to między innymi: data utworzenia, autor, dane o osobie, która dokonała ostatnich zmian i wiele, wiele innych.



MS Access nie posiada obiektu *DocumentProperties* do przechowywania wbudowanych i własnych właściwości wyświetlanych w oknie dialogowym *Właściwości bazy danych*.

Aby zwrócić kolekcję *DocumentProperties*, użyj dwóch podstawowych właściwości:

- ◆ *BuiltInDocumentProperties*, zawierającej wszystkie wbudowane właściwości dokumentu,
- ◆ *CustomDocumentProperties*, zawierającej wszystkie dodane właściwości dokumentu.

Listing 2.22 przedstawia przykładową procedurę wyliczającą po nazwie wbudowane właściwości aktywnego skoroszytu i ich wartości.

### Listing 2.22. Wyliczanie wbudowanych właściwości aktywnego skoroszytu i ich wartości

```

Sub WyliczWbudowaneWlasciwosciExcel()
On Error GoTo Err_WyliczWbudowaneWlasciwosciExcel

    rw = 1
    Worksheets(1).Activate

```

```
For Each p In ActiveWorkbook.BuiltinDocumentProperties
    Cells(rw, 1).Value = p.Name
    Cells(rw, 2).Value = p.Value
    rw = rw + 1
Next

Exit Sub

Err_WyliczWbudowaneWlasciowosciExcel:
    Select Case Err.Number
        Case -2147467259
            Cells(rw, 2).Value = "brak danych"
            Resume Next

        Case Else
            MsgBox "Przyczyna błędu: " & Err.Description, vbCritical, "Bład nr "
            & Err.Number
    End Select
    Err.Clear
End Sub
```

---

Identyczną procedurę należałoby napisać do wyliczenia własnych, dodanych właściwości danego dokumentu. Oczywiście, jeśli dany kod miałby być uruchomiony dla aplikacji MS Word, zamiast wyrażenia `ActiveWorkbook` należałoby użyć `ActiveDocument`.

Posługując się tymi właściwościami można bardzo dużo się dowiedzieć o historii dokumentu. Mianowicie, kiedy i przez kogo był ostatnio zapisany, czy został wydrukowany i kto go drukował oraz uzyskać wiele innych ciekawych informacji. Zastanów się, jak mógłbyś je wykorzystać.

## Środowisko aplikacji

Dlaczego środowisko aplikacji? Nazwałem ten rozdział tak, ponieważ wszystkie podane funkcje są związane ze środowiskiem, otoczeniem, sprawami związanymi z aplikacją, którą projektujesz. Bardzo często oprócz typowych informacji zawartych w dokumencie, skoroszycie czy bazie danych będziesz potrzebował innych, np.

- ♦ jaki użytkownik jest w tej chwili zalogowany?
- ♦ gdzie jest katalog systemowy, tymczasowy?
- ♦ jak wywołać inny program?
- ♦ i wiele, wiele innych.

## Funkcja Environ

Pierwsza z kilku funkcji związanych ze środowiskiem aplikacji, bardzo interesująca. Służy do zwracania wartości typu `String` odpowiadającej wartości zmiennej otoczenia systemu operacyjnego. Ja osobiście używałem jej w kilku sytuacjach:

- ◆ Dla aplikacji pracującej w Windows NT<sup>®</sup> odczytywałem użytkownika. W ten sposób pomijałem proces logowania uprawnionych użytkowników. Oczywiście aplikacja kontrolowała, kto miał nadane uprawnienia.
- ◆ Do zwrócenia wartości ścieżki katalogu TEMP.

Składnia tej funkcji jest następująca: Environ({envstring | number}).

envstring — jest to wyrażenie znakowe zawierające nazwę zmiennej otoczenia. Argument nieobowiązkowy.

number — jest to wyrażenie numeryczne odpowiadające liczbie porządkowej ciągu otoczenia w tablicy ciągów otoczenia. Argument number może być dowolnym wyrażeniem numerycznym, ale przed użyciem jego wartość jest zaokrąglana do najbliższej liczby całkowitej. Argument nieobowiązkowy.

Możesz posłużyć się tą funkcją na dwa sposoby:

1. Użyć argumentu envstring, na przykład tak:

```
zmienna = Environ("PATH")
```

Jeżeli argument envstring zostanie znaleziony w tablicy ciągów otoczenia, zwracana jest wartość *Zmiennej środowiskowej*.

2. Użyć argumentu number:

```
zmienna = Environ(1)
```

Jeżeli argument number zostanie znaleziony w tablicy ciągów otoczenia, zwracana jest nazwa zmiennej i jej wartość w następującej postaci:

```
zmienna=wartosc_zmiennej
```

Kiedy nie zostanie znaleziony żaden z argumentów, funkcja Environ zwróci ciąg o zerowej długości ("").

W listingu 2.23 funkcja Environ zwraca nazwy i wartości wszystkich zmiennych środowiska.

**Listing 2.23.** *Wyliczenie zmiennych środowiskowych*

```
Sub WyliczZmienne()
Dim Zmienna As String, Komunikat As String
Dim i As Integer

i = 1
Do
    Zmienna = Environ(i)
    Komunikat = Komunikat & vbCrLf & Zmienna
    i = i + 1
Loop Until Zmienna = ""

MsgBox Komunikat
End Sub
```

## Funkcja Shell

Funkcja ta uruchamia określony program. W przypadku powodzenia zwraca wartość typu Variant (Double) reprezentującą identyfikator zadania odpowiadającego uruchomionemu programowi. W przypadku niepowodzenia zwraca zero.

**Składnia tej funkcji:** Shell (pathname[, windowstyle]).

pathname — jest to nazwa programu, który ma być uruchomiony. Użycie wraz ze wszystkimi niezbędnymi argumentami lub opcjami podawanymi w wierszu poleceń; może zawierać nazwę katalogu lub folderu oraz nazwę dysku. Argument obowiązkowy typu Variant (String).

windowstyle — podając ten argument określasz styl okna, w którym ma być wykonywany program. Pomijając go, uruchomisz program w postaci zminimalizowanej z fokusem. Argument nieobowiązkowy typu Variant (Integer). Argument ten może przyjąć jedną z wartości przedstawionych w tabeli 2.3.

**Tabela 2.3.** Wartości parametru windowstyle

Wartość	Opis działania
vbHide	0 Okno jest ukrywane, a fokus jest przekazywany do tego ukrytego okna.
vbNormalFocus	1 Okno otrzymuje fokus, a rozmiar i położenie okna są przywracane do stanu początkowego.
vbMinimizedFocus	2 Okno jest wyświetlane w postaci zminimalizowanej z fokusem.
vbMaximizedFocus	3 Okno jest maksymalizowane i ma fokus.
vbNormalNoFocus	4 Otworzony jest ostatni rozmiar i położenie okna. Aktualnie aktywne okno pozostaje aktywne.
vbMinimizedNoFocus	6 Okno jest wyświetlane jako ikona. Aktualnie aktywne okno pozostaje aktywne.



Funkcja Shell uruchamia programy w sposób asynchroniczny. Oznacza to, że program uruchomiony przez funkcję Shell może się nie zakończyć przed rozpoczęciem wykonywania instrukcji występujących po funkcji Shell.

Przykład wykorzystania funkcji Shell przedstawia listing 2.24.

**Listing 2.24.** Przykład wykorzystania funkcji Shell

```
Dim MajProgram
MajProgram = Shell("C:\MajProgram.EXE", 1)
If MajProgram <> 0 Then
    'tutaj okresl, co chcesz zrobic z uruchomionym programem
Else
    MsgBox "Uruchomienie programu nie powiodlo sie!", vbCritical, "UWAGA"
End If
```

Spotkałem się z wykorzystaniem funkcji Shell do pakowania plików, które były następnie wysyłane do zainteresowanych osób. Muszę przyznać, że jest to bardzo ciekawe wykorzystanie tej funkcji.

## Instrukcja AppActivate

Instrukcja AppActivate uaktywnia okno aplikacji.

Składnia: AppActivate title[, wait].

title — wyrażenie znakowe określające tekst umieszczony na pasku tytułu uaktywnianego okna aplikacji. W celu uaktywnienia aplikacji zamiast argumentu title można również użyć identyfikatora zadania zwracanego przez funkcję Shell. Argument obowiązkowy.

wait — wartość typu Boolean, która określa, czy aplikacja wołająca inną aplikację ma fokus. Wartość False (domyślnie) powoduje, że wywoływana aplikacja jest uaktywniana niezależnie od tego, czy aplikacja miała fokus. Wartość True powoduje, że aplikacja wołająca czeka, aż zostanie do niej przekazany fokus, a następnie uaktywnia podaną aplikację. Argument nieobowiązkowy.

Instrukcja AppActivate przenosi fokus na podaną aplikację lub okno, ale nie wpływa na to, czy aplikacja jest minimalizowana czy maksymalizowana. Fokus jest przenoszony z okna uaktywnionej aplikacji, gdy użytkownik wykonuje czynność związaną ze zmianą fokusu lub zamyka okno dialogowe. Aby uruchomić aplikację lub ustawić styl okna, należy użyć funkcji Shell.



Podczas określania, która aplikacja ma zostać uaktywniona, argument title jest porównywany z tytułami wszystkich aktualnie działających aplikacji. Jeśli żaden z tytułów okien nie jest zgodny z porównywanym wzorcem, uruchamiana jest ta aplikacja, dla której początkowa część tytułu odpowiada wartości argumentu title. Jeśli zostało uruchomionych kilka kopii tej samej aplikacji o podanym tytule, zostanie wybrana jedna w nich.

Gdy podany program jest już uruchomiony, uaktywniasz jego okno wywołując go w podany sposób:

```
AppActivate "Microsoft Word"
```

Ponieważ instrukcja AppActivate może używać wartości zwróconej przez funkcję Shell, możesz uaktywnić okno aplikacji poprzednio uruchomionej funkcją Shell. Przykłady wykorzystania funkcji Shell i AppActivate przedstawia listing 2.25.

**Listing 2.25.** Przykłady wykorzystania funkcji Shell i AppActivate

```
WordID = Shell("C:\Program Files\Microsoft Office\Office\WINWORD.EXE", 1)
AppActivate WordID

ExcelID = Shell("C:\Program Files\Microsoft Office\Office\EXCEL.EXE", 1)
AppActivate ExcelID
```

## Funkcja DoEvents

Funkcja ta przekazuje sterowanie do systemu operacyjnego, umożliwiając przetwarzanie zdarzeń. Sterowanie to zwracane jest po obsłużeniu przez system operacyjny wszystkich zdarzeń z systemowej kolejki oraz po wysłaniu z kolejki *SendKeys* wszystkich kluczy.

We wszystkich aplikacjach Office funkcja *DoEvents* zwraca wartość zero. Z kolei w autonomicznych wersjach systemu Visual Basic, jak np. Visual Basic, Standard Edition, funkcja *DoEvents* zwraca wartość typu *Integer* reprezentującą liczbę otwartych formularzy.

Funkcja *DoEvents* jest bardzo przydatna, gdy wykonywany program zajmuje dużo czasu procesora. Korzystając z tej funkcji (tylko w razie takiej potrzeby), pozwalasz systemowi operacyjnemu na obsługę bez zbędnego opóźnienia zdarzeń takich, jak naciśnięcia klawiszy czy kliknięcia myszą.



Przekazując sterowanie za pomocą funkcji *DoEvents*, upewnij się, czy procedura oddająca sterowanie NIE ZOSTANIE ponownie wywołana z innej części programu przed powrotem sterowania po pierwszym wywołaniu funkcji *DoEvents*. Grozi to trudnymi do przewidzenia skutkami, uszkodzeniami plików, a nawet systemu. Pamiętaj też, że funkcji *DoEvents* nie należy stosować, jeśli inne aplikacje mogą w tym czasie odwoływać się do procedury, która oddała sterowanie.

Jak używać tej funkcji, pokażę przy opisywaniu kolejnej.

## Funkcja Timer

Zwraca wartość typu *Single* określającą liczbę sekund, które minęły od północy.

W tym przykładzie (listing 2.26) funkcja *Timer* przerywa wykonywanie aplikacji na określony czas i pozwala, by w trakcie tej przerwy mogły działać inne procesy.

**Listing 2.26.** *Przykład przerywania wykonywania aplikacji*

```
Dim DlugoscPrzerwy, Start, Koniec, CzasCalk
If (MsgBox("Nacisnij Tak, aby przerwac program na 5 sekund", 4)) = vbYes Then
    DlugoscPrzerwy = 5 ' Ustaw dlugosc przerwy.
    Start = Timer ' Pobierz czas poczatku przerwy.
    Do While Timer < Start + DlugoscPrzerwy
        DoEvents ' Pozwol dzialac innym procesom.
    Loop
    Koniec = Timer ' Pobierz czas zakonczenia przerwy.
    CzasCalk = Koniec - Start ' Oblicz czas calkowity.
    MsgBox "Przerwa trwala " & CzasCalk & " sekund"
Else
    End
End If
```



Aplikacja MS Excel posiada oddzielną funkcję. W aplikacji MS Excel do zatrzymania wykonywania kodu na określony czas lub do określonej godziny służy funkcja `Wait`. Listing 2.27 ilustruje procedurę `Czekaj`, która w parametrach wywołania pobiera czas w sekundach.

### Listing 2.27. Procedura `Czekaj`

```
Sub Zatrzymaj()
    MsgBox "Po naciśnięciu OK, aplikacja zostanie wstrzymana na 5 sekund"
    Czekaj 5
    MsgBox "Minelo 5 sekund"
End Sub

Sub Czekaj(ileSekund As Integer)
    Dim aktGodz As Integer
    Dim aktMin As Integer
    Dim aktSek As Integer
    Dim czasOczek As String

    aktGodz = Hour(Now)
    aktMin = Minute(Now)
    aktSek = Second(Now)

    czasOczek = TimeSerial(aktGodz, aktMin, aktSek + ileSekund)
    Application.Wait czasOczek

End Sub
```

## Instrukcja `SendKeys`

Wysyła do aktywnego okna dane generowane przez naciśnięcie klawiszy, tak jakby zostały one wprowadzone z klawiatury.

**Składnia:** `SendKeys string[, wait]`.

`string` — wyrażenie znakowe określające naciśnięcia klawiszy. Argument obowiązkowy. Aby użyć klawiszy specjalnych, musisz wpisać ich kod (tabela 4.2).

`wait` — wartość typu `Boolean` określająca typ czekania. Wartość `False` (domyślnie) oznacza, że sterowanie powróci do procedury zaraz po wysłaniu sekwencji naciśnięć klawiszy. Wartość `True` oznacza, że sterowanie powróci do procedury po przetworzeniu wysłanych naciśnięć klawiszy. Argument nieobowiązkowy.

Każdy klawisz jest reprezentowany przez jeden lub więcej znaków. Aby przesłać jeden znak umieszczony na klawiaturze, jako wartość należy podać ten znak. Na przykład, aby przesłać literę `A`, argumentowi `String` należy nadać wartość `A`. Aby przesłać więcej niż jeden znak, należy dołączać kolejne znaki do ciągu. I tak, aby przesłać literę `A`, `B` i `C`, argumentowi `String` należy nadać wartość `ABC`.

Znaki plus (+), daszek (^), procent (%), tylda (~) i nawiasy okrągłe ( ) mają w instrukcji `SendKeys` specjalne znaczenie. Aby przesłać dowolny z tych znaków, należy umieścić je w nawiasach klamrowych {}. Na przykład, aby przesłać znak plus, należy napisać {+}.

Nawiasy kwadratowe [ ] nie mają specjalnego znaczenia w instrukcji `SendKeys`, jednakże muszą być one umieszczane w nawiasach okrągłych. Mogą one mieć znaczenie w innych aplikacjach, np. przy dynamicznej wymianie danych (DDE). Aby przesłać nawiasy klamrowe, należy wpisać `{ }` i `{ }`.

Znaki, które nie są wyświetlane po naciśnięciu klawisza, takie jak *Enter* czy *Tab*, oraz klawisze oznaczające akcje, a nie znaki, są reprezentowane przez następujące kody (tabela 2.4):

Aby przesłać kombinację klawisza z klawiszami *Shift*, *Ctrl*, lub *Alt*, należy poprzedzić kod klawisza odpowiednimi kodami z tabeli 2.5

Aby określić, że klawisze *Shift*, *Ctrl* lub *Alt* mają być wciśnięte w momencie naciskania kilku innych klawiszy, należy kody tych klawiszy ująć w nawias. Na przykład, aby określić, że ma być wciśnięty klawisz *Shift*, podczas naciskania klawiszy E i C, należy napisać "+(EC)". Aby określić, że klawisz *Shift* ma być wciśnięty tylko podczas naciskania klawisza E, należy napisać "+EC".

Aby określić kolejne naciśnięcia tego samego klawisza, należy napisać {klawisz liczba}. Parametry klawisz i liczba muszą być rozdzielone spacją. Na przykład, {LEFT 42} oznacza, że kod klawisza STRZAŁKA W LEWO ma zostać wysłany 42 razy; {h 10} oznacza, że kod klawisza H zostanie wysłany 10 razy.



Za pomocą funkcji `SendKeys` nie można wysłać naciśnień klawiszy do aplikacji, która nie została zaprojektowana dla systemu Microsoft Windows. `SendKeys` nie umożliwia również wysłania klawisza `PRINT SCREEN` {PRTSC} do żadnej aplikacji.

Listing 2.28 ilustruje wykorzystanie funkcji `Shell` do uruchomienia aplikacji *Kalkulator* systemu Microsoft Windows. Za pomocą funkcji `SendKeys` wysyłane są naciśnięcia klawiszy w celu pomnożenia liczb i zakończenia pracy *Kalkulatora*.

**Listing 2.28.** Użycie funkcji `Shell` i `SendKeys` w praktyce

```
Sub Mnozenie()
  Dim ZwracWart, I
  ZwracWart = Shell("CALC.EXE", 1)
  AppActivate ZwracWart
  For I = 1 To 5
    SendKeys I & "{*}", True
  Next I
  SendKeys "=", True
  SendKeys "%{F4}", True
End Sub
```

## Automakra

Po serii ciekawych instrukcji i funkcji przyszedł czas na automakra. Są one nierozdzielnie związane ze środowiskiem aplikacji, to znaczy, że są wykonywane zaraz po uruchomieniu aplikacji czy dokumentu.

**Tabela 2.4.** Kody klawiszy funkcyjnych

<b>Klawisz</b>	<b>Kod</b>
<i>BACKSPACE</i>	{BACKSPACE}, {BS} <b>lub</b> {BKSP}
<i>BREAK</i>	{BREAK}
<i>CAPS LOCK</i>	{CAPSLOCK}
<i>DEL lub DELETE</i>	{DELETE} <b>lub</b> {DEL}
<b>STRZAŁKA W DÓŁ</b>	{DOWN}
<i>END</i>	{END}
<i>ENTER</i>	{ENTER} <b>lub</b> ~
<i>ESC</i>	{ESC}
<i>HELP</i>	{HELP}
<i>HOME</i>	{HOME}
<i>INS lub INSERT</i>	{INSERT} <b>lub</b> {INS}
<b>STRZAŁKA W LEWO</b>	{LEFT}
<i>NUM LOCK</i>	{NUMLOCK}
<i>PAGE DOWN</i>	{PGDN}
<i>PAGE UP</i>	{PGUP}
<i>PRINT SCREEN</i>	{PRTSC}
<b>STRZAŁKA W PRAWO</b>	{RIGHT}
<i>SCROLL LOCK</i>	{SCROLLLOCK}
<i>TAB</i>	{TAB}
<b>STRZAŁKA W GÓRĘ</b>	{UP}
<i>F1</i>	{F1}
<i>F2</i>	{F2}
<i>F3</i>	{F3}
<i>F4</i>	{F4}
<i>F5</i>	{F5}
<i>F6</i>	{F6}
<i>F7</i>	{F7}
<i>F8</i>	{F8}
<i>F9</i>	{F9}
<i>F10</i>	{F10}
<i>F11</i>	{F11}
<i>F12</i>	{F12}
<i>F13</i>	{F13}
<i>F14</i>	{F14}
<i>F15</i>	{F15}
<i>F16</i>	{F16}

**Tabela 2.5.** *Kody klawiszy specjalnych*

Klawisz	Kod
SHIFT	+
CTRL	^
ALT	%

Automakra to takie makra, które noszą specjalną nazwę, przez co program automatycznie je uruchamia, gdy znajdzie określona sytuacja, na przykład, kiedy uruchomisz lub zamkniesz dany program lub dokument.

Tabela 2.6 przedstawia nazwy makr rozpoznawane przez poszczególne aplikacje Office.

**Tabela 2.6.** *Nazwy automakr*

Program	Nazwa	Kiedy zostaje uruchomione?
Word	AutoExec	Podczas otwierania (ładowania) szablonu globalnego Word.
	AutoNew	Za każdym razem, kiedy stworzysz nowy dokument.
	AutoOpen	Za każdym razem, kiedy otwierasz istniejący dokument (szablon).
	AutoClose	Za każdym razem, kiedy zamykasz dokument (szablon).
	AutoExit	Podczas zamykania szablonu globalnego lub zamykania aplikacji Word.
Excel	Auto_Open	Za każdym razem, kiedy otwierasz istniejący skoroszyt (szablon).
	Auto_Activate	Za każdym razem, kiedy uaktywniasz obiekt.
	Auto_Deactivate	Za każdym razem, kiedy deaktywujesz obiekt.
Access	Auto_Close	Za każdym razem, kiedy zamykasz skoroszyt (szablon).
	AutoExec	Za każdym razem, kiedy otwierasz bazę danych.

Oczywiście automakra nie są jedynymi procedurami uruchamianymi podczas otwierania określonego rodzaju dokumentu. Większość dokumentów pakietu Office posiada własne procedury uruchamiane wraz z otwarciem czy zamknięciem dokumentu. Podstawowe procedury przedstawione są w tabeli 2.7. Jeżeli chcesz poznać inne, zajrzyj do przeglądarki obiektów.

**Tabela 2.7.** *Inne makra uruchamiane automatycznie*

Aplikacja	Nazwa procedury	Kiedy zostaje uruchomione?
Word	Document_New	Za każdym razem, kiedy stworzysz nowy dokument.
	Document_Open	Za każdym razem, kiedy otwierasz istniejący dokument (szablon).
	Document_Close	Za każdym razem, kiedy zamykasz istniejący dokument (szablon).
Excel	Workbook_Open	Za każdym razem, kiedy otwierasz skoroszyt (szablon).
	Workbook_BeforeClose	Za każdym razem, kiedy zamykasz skoroszyt (szablon).

W literaturze spotkałem się z określeniem, iż automakra pozostały dla zachowania zgodności z poprzednimi wersjami VBA dla Office. Ale czy tak jest naprawdę? Nie wiem. Wielokrotnie spotkałem się nawet w najnowszych szablonach, kreatorach z autoprocedurami, zamiast z nowymi procedurami uruchomieniowymi dla poszczególnych dokumentów.

### Jak utworzyć automakro w Wordzie?

- ◆ Jeżeli automakro ma być umieszczone w szablonie globalnym...

Wstaw moduł do normal i nadaj mu dowolną nazwę. W module tym umieść na przykład taką procedurę:

```
Sub AutoExec()  
    'tu instrukcje, które mają być wykonywane  
End Sub
```

- ◆ Jeżeli automakro ma być umieszczone w szablonie i (lub) dokumencie...

Wstaw moduł do tego szablonu (dokumentu) i nadaj mu dowolną nazwę. W module tym umieść na przykład taką procedurę:

```
Sub AutoOpen()  
    'tu instrukcje, które mają być wykonywane  
End Sub
```



Automakra mogą być przechowywane w szablonie globalnym, innym szablonie lub dokumencie. W przypadku wystąpienia konfliktu nazw (wiele makr będzie miało tę samą nazwę), Word uruchomi makro w najbliższym kontekście. Na przykład, jeżeli utworzysz makro `AutoClose` w szablonie i dokumencie, Word uruchomi tylko makro z dokumentu. Jeśli utworzysz makro `AutoNew` w szablonie globalnym, makro to zostanie uruchomione pod warunkiem, że nie będzie zawierał takiego żaden z otwartych dokumentów lub szablonów.

### Jak utworzyć automakro w Excelu?

Podobnie jak w Wordzie, do aktywnego skoroszytu musisz wstawić moduł i nadać mu dowolną nazwę. Następnie umieść w nim procedurę, jak pokazuje przykład poniżej:

```
Sub Auto_Open()  
    'tu instrukcje, które mają być wykonywane  
End Sub
```



Aby uruchomić makra `Auto_Activate` lub `Auto_Deactivate`, musisz posłużyć się metodą `RunAutoMacros`.

Metoda `RunAutoMacros` dla MS Excela ma następującą składnię:

wyrażenie `.RunAutomacros(które)`

wyrażenie — określa skoroszyt. Argument obowiązkowy.

które — argument obowiązkowy, określający makro do uruchomienia. Może przyjąć jedną z następujących postaci (tabela 2.8):

**Tabela 2.8.** Wartości argumentu które dla metody *RunAutoMacros*

Wartość	Uruchamia...
x!AutoOpen	procedurę Auto_Open.
x!AutoClose	procedurę Auto_Close.
x!AutoActivate	procedurę Auto_Activate.
x!AutoDeactivate	procedurę Auto_Deactivate.

## Jak utworzyć automakro w programie Access?

Tu zagadnienie nieco się komplikuje, a to ze względu na kilka czynników. Postaram się jednak Ci to wszystko wytłumaczyć.

Możesz utworzyć makro, które będzie automatycznie się uruchamiało podczas otwierania bazy danych na dwa sposoby:

1. Dodać nowe makro (oczywiście odpowiednio je nazwać) i tam określić wszystkie polecenia do wykonania.

Zawartość makra AutoExec:

Pole Akcja	Pole Nazwa formularza
Otworz formularz.	Okresl nazwe formularza, ktory ma byc uruchamiany na starcie.

2. Wstawić moduł i umieścić w nim funkcję, którą każdorazowo będzie wykonywała baza danych podczas otwierania. Następnie dodać nowe makro, a w nim określić, jaki kod ma wykonać. W polu *Akcja* umieść *Uruchom kod*, a w polu *Nazwa funkcji* nazwę funkcji, która ma zostać wykonana.

Zawartość makra AutoExec:

Pole Akcja	Pole Nazwa funkcji
Uruchom kod.	Otwarcie()

Zawartość modułu:

```
Function Otwarcie()
'tu instrukcje, ktore maja byc wykonywane
End Functions
```

Oczywiście, sposób drugi jest bardziej wydajny, bo masz praktycznie nieograniczone możliwości (no, chyba że ograniczone do możliwości języka VBA i Twoich). Zaś sposób pierwszy pozwala Ci wykonać tylko akcje dostępne w polu *Akcja*.



Zapewne zwróciłeś uwagę, że w aplikacjach Word i Excel używaliśmy słowa kluczowego *Sub*, a w Access *Function*. I to jest właśnie ta podstawowa różnica w konstrukcji modułu.



Aby wyłączyć działanie automakr, wystarczy podczas otwierania określonego dokumentu Office przytrzymać klawisz *Shift*. Nie da się jednak zatrzymać automakr zapisanych w dodatkach.



Directory. Przejście przez dotychczasowe rozdziały powinno dać biegłość we wszystkich głównych elementach Active Directory i doprowadzić do rozpoczęcia pierwszego rzeczywistego

## Ostrzeżenie

Wiesz już jak umieszczać automakra w dokumentach Office. Chcę Cię jednak ostrzec, że automakra są często wykorzystywane w wirusach. W dokumentach Word najczęściej dochodzi do zarażenia szablonu globalnego, po czym w tym szablonie umieszczana jest procedura AutoExec z instrukcjami destrukcyjnymi. Jeżeli otwierasz jakiegokolwiek dokument Word, który nie powinien mieć żadnych makr czy kodu, a pojawia się ostrzeżenie o makrowirusach, oznacza to, że najprawdopodobniej masz zarażony szablon globalny. Oczywiście nic tak nie pomoże jak dobry program antywirusowy, ale możesz spróbować się bronić sam. Utwórz skrót do programu Word, po czym wejdź do właściwości tego skrótu. W polu *Obiekt docelowy* dodaj na końcu ścieżki do programu po spacji parametr /m. Spowoduje to wywoływanie aplikacji Word bez opcji uruchamiania automakra. Oczywiście makro AutoExec, które sam napisałeś, również nie zostanie uruchomione. Jeśli jednak Twoja aplikacja wymaga, by jakieś makro było uruchamiane wraz z programem Word, musisz utworzyć makro i nadać mu na przykład nazwę WłasnyAutoExec, po czym otwierać aplikację przez skrót utworzony w następujący sposób:

1. Utwórz skrót do programu Word, po czym wejdź do właściwości tego skrótu.
2. W polu *Obiekt docelowy* dodaj na końcu ścieżki do programu po spacji parametr /m oraz nazwę Twego makra (bez spacji): /mWłasnyAutoExec.

Excel posiada także pewne właściwości startowe. Jeśli chcesz, by jakiś skoroszyt był uruchamiany za każdym razem, kiedy jest uruchamiany Excel, wystarczy, że zapiszesz go w katalogu: `\Program Files\Microsoft Office\Office\XlStart`. Możesz w ten sposób monitorować kto, kiedy i jaki otwierał skoroszyt.

Taką metodę wykorzystywał słynny wirus o nazwie „Laroux”. Umieszczał on we wspomnianym katalogu skoroszyt o nazwie „Personal”, który automatycznie uruchamiał się wraz z programem Excel. Po otwarciu skoroszytu uaktywniał się wirus, który sprawdzał, czy pozostałe skoroszyty (otwarte przez użytkownika lub nowo utworzone) zawierają arkusz o nazwie takiej, jak wirus. Jeśli nie, tworzył taki arkusz, ukrywał i wstawiał moduł z kodem wirusa. Otwarcie tak zarażonego pliku na dowolnym, innym komputerze powodowało utworzenie pliku *Personal.xls* (jeśli nie istniał) itd., itd. Istnieje kilkanaście odmian tego wirusa. Najłagodniejsza utrudnia życie tylko wtedy, jeśli wykonujesz jakieś czynności automatycznie (z kodu), szczególnie kiedy w odwołaniach do arkuszy nie posługiwałeś się nazwą, a indeksem. W najgroźniejszej swej odmianie wirus formatuje dyski.



Przykłady skrótów z opcjami uruchamiania lub wyłączania automakr dla poszczególnych aplikacji znajdują się na dysku CD-ROM w katalogu *Przykłady\Cz\_II, pliki \*.lnk*.