helion.pl

Idź do

- Spis treści
- Przykładowy rozdział

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

Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2010

Excel. Programowanie dla profesjonalistów. Wydanie II



Autorzy: Rob Bovey, Dennis Wallentin, Stephen Bullen, John Green
Tłumaczenie: Robert Górczyński
ISBN: 978-83-246-2544-4
Tytuł oryginału: Professional Excel Development: The Definitive Guide to Developing Applications Using Microsoft Excel, VBA, and .NET (2nd Edition)
Format: 172×245, stron: 1096

Kompendium wiedzy dla programistów Excela!

- Jak wykorzystać nowy interfejs Excela 2007?
- Jak sterować innymi aplikacjami z pakietu Microsoft Office?
- Jak używać Windows API?

Dzięki książce „Excel. Programowanie dla profesjonalistów. Wydanie II" poznasz tajniki tworzenia aplikacji opartych o Microsoft Excel. Naukę rozpoczniesz od poznania architektury takich aplikacji. Poznasz też zestaw najlepszych praktyk w programowaniu w VBA. W kolejnych rozdziałach przyswoisz wiedzę na temat projektowania arkuszy, weryfikacji danych, sterowania paskami poleceń oraz współpracy z interfejsem typu wstążka (ang. Ribbon). Ponadto zdobędziesz informacje na temat wykorzystania Windows API, obsługi błędów, programowania baz danych oraz optymalizacji aplikacji. Wśród wielu innych zagadnień omawianych w tej książce warto także zwrócić uwagę na takie, jak współpraca z usługami sieciowymi, sterowanie innymi aplikacjami z pakietu Microsoft Office oraz techniki przetwarzania danych. Książka stanowi kompendium wiedzy na temat programowania w środowisku Microsoft Excel, a o jej jakości merytorycznej świadczy fakt, iż trójka jej autorów została wyróżniona przez firmę Microsoft tytułem MVP (Most Valuable Professional).

- Architektura aplikacji opartych o Microsoft Excel
- Najlepsze praktyki tworzenia aplikacji w Excelu i VBA
- Zasady projektowania arkuszy
- Używanie stylów, nazw zdefiniowanych, formatowań warunkowych
- Wykorzystanie kontrolek i zastosowanie dodatków funkcyjnych
- Użycie modułów klas do tworzenia obiektów
- Zarządzanie paskiem poleceń
- Praca z plikami XML
- Wykorzystanie możliwości nowego interfejsu Excela 2007
- Użycie Windows API
- Tworzenie formularzy
- Zastosowanie interfejsów oraz ponowne wykorzystanie kodu
- Sytuacje wyjątkowe – obsługa błędów
- Debugowanie i optymalizacja kodu VBA
- Zastosowanie asercji
- Współpraca z bazami danych i techniki przetwarzania danych
- Tworzenie zaawansowanych wykresów
- Sterowanie innymi aplikacjami z pakietu Microsoft Office
- Wykorzystanie języka VB.NET
- Możliwości narzędzia Visual Studio Tools for Office System

Wykorzystaj Excel jako fundament efektywnych rozwiązań!

SPIS TREŚCI

Podziękowania 14
O autorach 13

Rozdział 1. Wstęp

O książce 15
Dla kogo przeznaczona jest ta książka? 15
Twórca programowania excelowego 16
Excel jako platforma do tworzenia aplikacji 18
Struktura 21
Przykłady 22
Wersje obsługiwane 23
Rodzaje kroju pisma 24
Na płycie CD 24
Pomoc i wsparcie 25
Witryna internetowa 26
Komentarze czytelników 26

Rozdział 2. Architektura aplikacji

Koncepcje 27
Wnioski 38

Rozdział 3. Najlepsze praktyki programowania w Excelu i VBA

Konwencje nazw 41
Najlepsze praktyki organizacji i tworzenia struktury aplikacji 53
Najlepsze praktyki określające ogólne zasady tworzenia oprogramowania 58
Wnioski 78

Rozdział 4. Projektowanie arkusza

Zasady projektowania dobrego interfejsu użytkownika 79
Wiersze i kolumny programu: podstawowe techniki tworzenia interfejsu użytkownika 80
Nazwy definiowane 81
Style 88
Techniki kreślenia interfejsów użytkownika 94
Weryfikacja danych 99

Formatowanie warunkowe 103
Używanie kontrolek w arkuszu 108
Przykład praktyczny 111
Wnioski 116

Rozdział 5. Dodatki funkcyjne, ogólne i specjalizowane dla aplikacji

Cztery etapy rozwoju i działania aplikacji 117
Dodatki będące bibliotekami funkcji 120
Dodatki ogólne 127
Dodatki specjalizowane dla aplikacji 127
Przykład praktyczny 134
Wnioski 146

Rozdział 6. Aplikacja dyktatorska

Struktura aplikacji dyktatorskiej 147
Przykład praktyczny 161
Wnioski 168

Rozdział 7. Używanie modułów klas do tworzenia obiektów

Tworzenie obiektów 169
Tworzenie kolekcji 173
Wychwytywanie zdarzeń 179
Generowanie zdarzeń 182
Przykład praktyczny 189
Wnioski 195

Rozdział 8. Zaawansowane sterowanie paskami poleceń

Projektowanie paska poleceń 198
Tablicowe sterowanie paskami poleceń 199
Zbieranie wszystkiego razem 219
Ładowanie niestandardowych ikon z plików 227
Podczepianie obsługi zdarzeń do kontrolek paska poleceń 231
Przykład praktyczny 240
Wnioski 246

Rozdział 9. Wprowadzenie do formatu XML

XML 249
Wnioski 271

Rozdział 10. Wstążka w Office 2007

Podstawowe założenia RibbonX 273
Wprowadzenie do formatu pliku Open XML w Office 2007 274
Projektowanie Wstążki oraz najlepsze praktyki programowania 278

Dostosowanie opcji Wstążki bazującej na tabeli 288
Zaawansowane rozwiązywanie problemów 289
Dalsze pozycje do czytania 297
Portale internetowe 298
Wnioski 299

Rozdział 11. Tworzenie aplikacji niezależnych od wersji Excela

Pasek poleceń i Wstążka w pojedynczej aplikacji 301
Inne kwestie dotyczące programowania w Excelu 2007 315
Bezpieczeństwo w Windows Vista a struktura katalogów 322
Wnioski 326

Rozdział 12. Zrozumienie i używanie wywołań Windows API

Ogólny opis 327
Praca z ekranem 333
Praca z oknami 336
Praca z klawiaturą 344
Praca z systemem plików i siecią 349
Przykład praktyczny 361
Wnioski 365

Rozdział 13. Projektowanie formularzy UserForm i najlepsze praktyki

Zasady 367
Podstawy kontrolek 375
Efekty wizualne 382
Pozycjonowanie i rozmiary formularzy UserForm 390
Kreatory 396
Dynamiczne formularze UserForm 400
Niemodalne formularze UserForm 407
Wyszczególnienie kontrolek 412
Przykład praktyczny 418
Wnioski 419

Rozdział 14. Interfejsy

Co to jest interfejs? 421
Ponowne użycie kodu 422
Definiowanie własnych interfejsów 425
Implementacja własnego interfejsu 426
Używanie własnych interfejsów 427
Klasy polimorficzne 429
Polepszanie solidności 433
Upraszczanie rozwoju 434

Architektura modułów rozszerzających 443
Przykład praktyczny 445
Wnioski 446

Rozdział 15. Obsługa błędów VBA

Pojęcia obsługi błędów 447
Zasada pojedynczego punktu wyjścia 456
Prosta obsługa błędów 457
Złożone projekty obsługi błędów 458
Centralna obsługa błędów 462
Obsługa błędów w klasach i formularzach UserForm 469
Zbieranie wszystkiego razem 470
Przykład praktyczny 476
Wnioski 484

Rozdział 16. Debugowanie kodów VBA

Podstawowe techniki debugowania kodów VBA 485
Okno Immediate (Ctrl+G) 495
Call Stack — stos wywołań (Ctrl+L) 498
Okno Watch 500
Okno Locals 510
Object Browser — przeglądarka obiektowa (F2) 511
Tworzenie działającego otoczenia testowego 514
Stosowanie asercji 517
Debugerskie skróty klawiaturowe, które powinien znać każdy programista 518
Wnioski 520

Rozdział 17. Optymalizacja wydajności VBA

Mierzenie wydajności 523
Program narzędziowy PerfMon 524
Myślenie kreatywne 528
Makrooptymalizacja 534
Mikrooptymalizacja 543
Wnioski 550

Rozdział 18. Programowanie i bazy danych

Wprowadzenie do baz danych 551
Wprowadzenie do SQL 568
Dostęp do danych za pomocą ADO 572
Dalsze pozycje do czytania 586
Wnioski 587

Rozdział 19. Programowanie z użyciem bazy danych Access i SQL Server

Uwaga dotycząca przykładowej bazy danych Northwind 589
Projektowanie warstwy dostępu do danych 590
Praca z bazami danych Microsoft Access 594
Praca z bazami danych Microsoft SQL Server 603
Rozbudowa bazy danych Access do SQL Server 613
Dalsze pozycje do czytania 618
Przykład praktyczny 619
Wnioski 629

Rozdział 20. Techniki przetwarzania danych

Struktury danych Excela 631
Funkcje przetwarzania danych 637
Zaawansowane funkcje 648
Wnioski 656

Rozdział 21. Zaawansowane techniki tworzenia wykresów

Podstawowe techniki 657
Techniki VBA 674
Wnioski 679

Rozdział 22. Sterowanie innymi aplikacjami Office

Podstawy 681
Modele obiektowe głównych aplikacji Office 695
Dalsze pozycje do czytania 707
Przykład praktyczny 707
Wnioski 708

Rozdział 23. Połączenie Excela i Visual Basica 6

Witaj świecie ActiveX DLL 710
Dlaczego używać VB6 ActiveX DLL w projektach Excel VBA? 725
In-process kontra out-of-process 739
Automatyzacja Excela z VB6 EXE 740
Dodatki COM 748
Dodatek COM Witaj świecie 748
Projektant dodatków (Add-in Designer) 752
Instalacja 755
Zdarzenia AddinInstance 757
Obsługa paska poleceń 759
Dlaczego warto używać dodatku COM? 762
Automatyzacja dodatków 764
Przykłady praktyczne 767
Wnioski 779

Rozdział 24. Excel i VB.NET

Podstawy platformy .NET 782
Visual Basic.NET 783
Usuwanie błędów 807
Użyteczne narzędzia programistyczne 816
Automatyzacja Excela 819
Zasoby w rozwiązaniach .NET 826
Pobieranie danych za pomocą ADO.NET 827
Dalsze pozycje do czytania 832
Dodatkowe narzędzia programistyczne 833
Fora 834
Przykład praktyczny — narzędzie raportujące PETRAS w technologii .NET 834
Wnioski 848

Rozdział 25. Tworzenie zarządzanych dodatków COM za pomocą VB.NET

Wybór narzędzi programistycznych 852
Tworzenie dodatku zarządzanego COM 853
Budowanie interfejsu użytkownika 870
Tworzenie zarządzanych dodatków automatyzacji 888
Ręczne rejestrowanie i wyrejestrowanie dodatków COM 899
Używanie klas w VB.NET 900
Używanie klasycznego ADO w celu eksportu danych do Excela 906
Opakowywanie dodatków COM 910
Blogi związane z tematem 919
Dodatkowe narzędzia programistyczne 920
Przykład praktyczny — narzędzie raportujące PETRAS w technologii .NET 921
Wnioski 929

Rozdział 26. Tworzenie rozwiązań Excela za pomocą Visual Studio Tools for Office System (VSTO)

Co to jest VSTO? 932
Kiedy powinniśmy używać VSTO? 939
Praca z dodatkami VSTO 941
Praca z szablonami VSTO i rozwiązaniami w postaci skoroszytu 961
Implementacja i bezpieczeństwo 970
Inne pozycje do czytania 979
Różne portale i blogi 979
Dodatkowe narzędzia programistyczne 980
Wnioski 980

Rozdział 27. XLL i API C

Dlaczego warto tworzyć funkcje arkusza na bazie XLL? 983
Tworzenie projektu XLL w Visual Studio 984
Struktura XLL 989
Typy danych XLOPER i OPER 997
Funkcja Excel4 1002
Powszechnie używane funkcje API C 1004
XLOPER i zarządzanie pamięcią 1005
Rejestrowanie i wyrejestrowywanie własnych funkcji arkusza 1006
Przykładowa funkcja aplikacji 1009
Debugowanie funkcji arkusza 1011
Różne tematy 1012
Dodatkowe źródła informacji 1013
Wnioski 1014

Rozdział 28. Excel i usługi sieciowe

Usługi sieciowe 1015
Przykład praktyczny 1022
Wnioski 1031

Rozdział 29. Zapewnianie pomocy, bezpieczeństwa, pakowanie i rozpowszechnianie

Zapewnianie pomocy 1033
Bezpieczeństwo 1042
Pakowanie 1046
Rozpowszechnianie 1051
Wnioski 1052

Skorowidz 1055

ROZDZIAŁ 10.

WSTĄŻKA W OFFICE 2007

Od czasu wydania pakietu Office 2007 **RibbonX** i jego interfejs użytkownika (**Wstążka**) stały się tematem częstych dyskusji w społeczności programistów Office. Jak można się spodziewać po technologii w wersji 1.0, RibbonX nakłada pewne ograniczenia, które uniemożliwiają wykorzystanie w pełni możliwości drzemiących w tym interfejsie użytkownika. Jednak nawet na tak wczesnym etapie powinniśmy zrozumieć podstawowe założenia nowego interfejsu, aby budować własne, doskonale zaprojektowane interfejsy użytkownika.

Innym wyzwaniem jest równoczesna codzienna praca z paskiem poleceń oraz technologiami interfejsu użytkownika RibbonX. Zagadnienie to obejmuje tworzenie rozwiązań, które będą działały z obydwoma interfejsami użytkownika, czyli tzw. aplikacji niezależnych od wersji Excela. W tym rozdziale zostaną przedstawione najlepsze praktyki w zakresie projektowania Wstążki i programowania RibbonX, natomiast aplikacje działające niezależnie od wersji Excela omówimy w rozdziale 11. Tu zajmiemy się także pewnymi nieco bardziej skomplikowanymi problemami aplikacji dyktatorskich. Bazujący na tabelach proces budowania interfejsu użytkownika paska poleceń jest szeroko akceptowany i uznawany za standard. Oznacza to, że do tego celu powinniśmy również używać technologii RibbonX i ten temat też będzie poruszony w rozdziale.

W pakiecie Office 2007 wprowadzono nowy format zapisu plików o nazwie **Office Open XML** (**OOXML** lub po prostu **Open XML**). Oferuje on strukturę szkieletową dla RibbonX — format OOXML i RibbonX są ze sobą ściśle powiązane. Nowy format zapisu plików pozwala także na tworzenie dokumentów Office i manipulowanie nimi bez używania któregokolwiek z programów wchodzących w skład pakietu Office. W rozdziale przedstawimy także krótkie wprowadzenie do formatu plików Open XML.

Podstawowe założenia RibbonX

Przez ponad dekadę paski poleceń były jedyną technologią interfejsu użytkownika, której obsługą musieliśmy się zajmować. Wraz z wprowadzeniem RibbonX otrzymujemy nową technologię oraz nowe założenia stanowiące podstawę interfejsu użytkownika, które łącznie mają ogromny wpływ na

nasz sposób pracy. Na pierwszy rzut oka nauka i manipulowanie RibbonX może wydawać się wyzwaniem. Jednak z naszego doświadczenia wynika, że prawdziwym wyzwaniem jest równoczesna codzienna praca zarówno z paskami poleceń, jak i RibbonX.

RibbonX, w porównaniu do pasków poleceń, ma wady i zalety. Chociaż jest w swojej pierwszej wersji, jego podstawy zostały już ustalone i cechy charakterystyczne technologii RibbonX można podsumować w następujący sposób.

- Opcje pozwalające na dostosowanie do własnych potrzeb określamy w trakcie projektowania interfejsu. Są zdefiniowane za pomocą XML i przechowywane jako oddzielna część formatu pliku XML. Jednak większość atrybutów kontrolek może być zmodyfikowana w trakcie działania aplikacji przy użyciu VBA (np. włączony/wyłączony, etykieta, widoczność itd.).
- Po otworzeniu skoroszytu zawierającego dostosowane do własnych potrzeb opcje Wstążki Excel automatycznie odczytuje definicję XML Wstążki, a następnie przystępuje do tworzenia Wstążki określonej przez wymienione opcje. Do zainicjalizowania tego procesu nie jest wymagany żaden kod VBA. W rzeczywistości nawet nie ma sposobu, aby uniemożliwić inicjalizację tego procesu.
- Kiedy skoroszyt zostanie otwarty i jest aktywny, dostosowane do własnych potrzeb opcje Wstążki zostają zastosowane i są widoczne. Po zamknięciu skoroszytu dostosowane do własnych potrzeb opcje Wstążki są automatycznie usuwane.
- Po wczytaniu dodatku dowolnego rodzaju jego dostosowane do własnych potrzeb opcje Wstążki zostają zastosowane i widoczne są we wszystkich otwartych skoroszytach.
- Na własnej Wstążce wszystkie wbudowane kontrolki Wstążki mogą być włączone, nadpisane, wykonane lub zapytane o nagłówek, grafikę itd.

Wprowadzenie do formatu pliku Open XML w Office 2007

XML po raz pierwszy wprowadzony został w pakiecie Office 2002. Od tej chwili Microsoft nieustannie rozbudowuje i ulepsza moduł XML będący częścią pakietu Office. Wraz z wydaniem Office 2007 otrzymujemy nowy format pliku całkowicie bazujący na XML, czyli format Open XML. Format ten zapewnia możliwość pracy z dokumentami Office oraz ich treścią bez konieczności używania którejkolwiek z aplikacji wchodzących w skład pakietu Office. Dzięki temu zwiększa się liczba rozwiązań alternatywnych służących do generowania dokumentów Office po stronie serwera. Znacznemu ułatwieniu ulega również proces wymiany danych między różnymi systemami oraz dokumentami Office.

Każdy dokument Office bazujący na Open XML jest przechowywany w **archiwum ZIP**. Wspomniane archiwum stanowi rodzaj pojemnika zawierającego zarówno dane użytkownika w formacie XML, jak i inne pliki wraz z informacjami o stylu, obrazami itd. Kiedy dokument jest zapisany w formacie Open XML, jego zawartość zostaje skompresowana, co skutkuje mniejszą wielkością pliku, w porównaniu do pliku w formacie binarnym. Microsoft twierdzi, że wielkość pliku można zredukować nawet o 75%, choć według naszych obserwacji typowa wielkość redukcji mieści się w okolicach 60 – 65%.

Ponieważ dokument Open XML składa się z kilku elementów umieszczonych razem w archiwum ZIP, dokumenty Office w tym formacie są znacznie odporniejsze na uszkodzenia niż zapisane w formacie binarnym. W ten sposób zmniejsza się ryzyko utraty informacji ze względu na zniszczone bądź uszkodzone pliki.

Powszechnie spotykanym wymaganiem jest, aby tworzone rozwiązania współdziałały z różnymi wersjami Excela. Rodzi to pytania o format pliku, który powinien zostać zastosowany. Rozwiązaniem zapewniającym największą elastyczność podczas wymiany danych pozostaje format *.xls* (w Excelu 2007 nosi nazwę *Skoroszyt programu Excel 97 – 2003*). Jednak możliwe jest użycie formatu pliku Open XML w starszych wersjach pakietu Office, o ile komputery docelowe mają zainstalowany *Pakiet zgodności formatu plików pakietu Microsoft Office dla programów Word, Excel i PowerPoint 2007*. Pakiet jest dostępny bezpłatnie na witrynie internetowej firmy Microsoft[1]. Warto zwrócić uwagę, że Office 2000 może konwertować pliki Open XML na format binarny tylko za pośrednictwem Eksploratora Windows.

Struktura formatu pliku Open XML

Format pliku Open XML może zostać opisany jako struktura składająca się z bloków stanowiących elementy składowe (czyli poszczególne części) oraz połączeń (związków) wykorzystywanych do skomponowania, zapakowania, rozpowszechniania i generowania zawartości dokumentu. Jądrem formatu pliku są schematy XML oraz archiwum ZIP. Podczas zapisu dokumentu Office w formacie Open XML następuje utworzenie archiwum ZIP zawierającego następujące komponenty.

- **Części** — większość elementów „części" to pliki XML opisujące dane aplikacji i metadane. Modułowość pozostaje najważniejszą cechą charakterystyczną formatu pliku, ponieważ pozwala na zlokalizowanie określonej części i bezpośrednią pracę z daną częścią.

[1] Pakiet można pobrać ze strony *http://www.microsoft.com/downloads/details.aspx?FamilyID=941b3470-3ae9-4aee-8f43-c6bb74cd1466&displaylang=pl — przyp. tłum.*

- **Elementy rodzaju treści** — opisują, jakiego rodzaju pliki są przechowywane w dokumencie. Przykładowo `image/png` oznacza plik graficzny w formacie PNG. Informacja ta pozwala aplikacjom na określenie zawartości dowolnej części pakietu i prawidłowe przetworzenie jej zawartości.
- **Związki** — opisują, jak zbiór elementów dokumentu razem tworzy postać dokumentu. Podczas gdy „części" tworzą treść pliku, związki opisują sposób, w jaki części ze sobą współpracują.

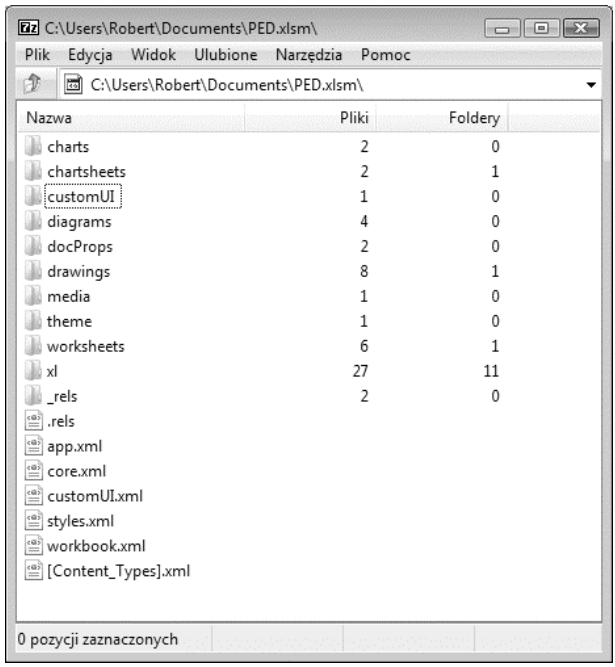
Przeanalizujmy samodzielnie zawartość pliku Excela. Do tej operacji potrzebny będzie program obsługujący archiwa ZIP, np. WinZip bądź 7-Zip. Pierwszym krokiem jest utworzenie pliku Excela, umieszczenie w nim wykresu, a następnie zapisanie pliku pod nazwą *PED.xlsm*. Do kolejnego kroku konieczne będzie użycie narzędzia **Office 2007 Custom UI Editor**, które można pobrać bezpłatnie z witryny Microsoft[2]. Otwórz plik w narzędziu Custom UI Editor i do skoroszytu dodaj własne opcje RibbonX przedstawione w listingu 10.1.

LISTING 10.1. Proste dostosowanie opcji w RibbonX

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
         <tab id="customTab" label="Własna karta">
            <group id="customGroup" label="Własna grupa">
               <button id="customButton" label="Własny przycisk"
               imageMso="HappyFace" size="large"
               onAction="Callback" />
            </group>
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

Na końcu otwórz plik *PED.xlsm* w programie obsługującym archiwa ZIP. Zawartość pliku powinna być podobna do pokazanej na rysunku 10.1. Niektóre programy obsługujące archiwa ZIP mogą wymagać tymczasowej zmiany rozszerzenia pliku skoroszytu na *.zip*, aby otworzenie archiwum było możliwe.

[2] Narzędzie jest dostępne na stronie *http://openxmldeveloper.org/articles/CustomUIeditor.aspx* — *przyp. tłum.*



RYSUNEK 10.1. Zawartość archiwum ZIP, pliki w formacie Open XML

Jak pokazano na rysunku 10.1, archiwum pliku Excela może składać się z dużej liczby katalogów i plików XML. Umożliwia to pracę z poszczególnymi częściami pliku, ale omówienie wszystkich elementów archiwum wykracza poza zakres tematyczny tego rozdziału. Plik Excela (*PED.xlsm*) oraz archiwum ZIP (*PED.zip*) umieszczono w katalogu *\Koncepcje\Rozdzial10* na płycie CD dołączonej do książki.

WSKAZÓWKA Jeżeli masz dostęp do oprogramowania Visual Studio 2008 Professional lub nowszej wersji oraz zainstalowany bezpłatny pakiet Microsoft Visual Tools for the Office System Power Tools, możesz przeglądać i edytować opcje Wstążki wewnątrz plików Excela, używając do tego celu narzędzia Open XML Editor.

Po dodaniu do skoroszytu lub dodatku opcji dostosowujących Wstążkę do własnych potrzeb wewnątrz struktury pliku Open XML zostaje utworzony katalog o nazwie *customUI*. W nim znajduje się plik XML o nazwie *customUI.xml* przechowujący wszystkie informacje potrzebne do dostosowania Wstążki do własnych potrzeb. Podczas uaktualniania opcji Wstążki plik *customUI.xml* jest

odpowiednio aktualizowany. Wszystkie opcje dostosowania Wstążki modyfikowane za pomocą narzędzia Custom UI Editor mają wpływ na odpowiedni fragment *customUI* w pliku Excela. W dalszej części rozdziału omówimy dostosowywanie przy użyciu VBA opcji Wstążki bazujących na tabelach.

Projektowanie Wstążki oraz najlepsze praktyki programowania

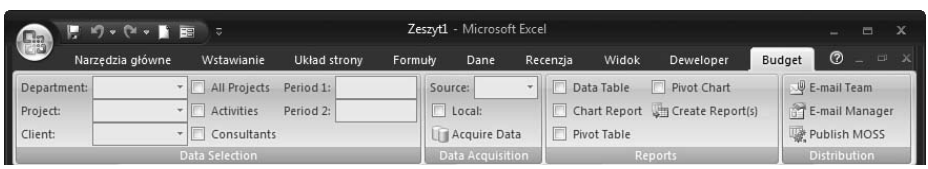
Wiele ogólnych zaleceń dotyczących podejść do programowania i przedstawionych w rozdziale 8. może być również zastosowanych podczas projektowania Wstążki. W czasie pisania tej książki Excel 2007 jest dostępny dopiero od dwóch lat i nadal ma stosunkowo niewielki udział w rynku. Wskutek tego w społeczności programistów Excela dopiero powstają najlepsze praktyki z zakresu pracy ze Wstążką.

Przedstawione poniżej zalecenia zostały sformułowane na podstawie naszych doświadczeń zdobytych podczas tworzenia rzeczywistych aplikacji i powinny być traktowane jedynie jako wstęp do najlepszych praktyk projektowania i programowania Wstążki. Zawsze wtedy, kiedy wykorzystywany jest graficzny interfejs użytkownika, najlepsze podejście to zachowanie prostoty i przejrzystości.

Projektowanie w celu obsługi procesów roboczych

Największym wyzwaniem podczas projektowania własnej Wstążki jest utworzenie takiego interfejsu, który faktycznie będzie wspomagał procesy robocze. Oznacza to, że musimy zidentyfikować wszystkie procesy robocze danego rozwiązania i zaimplementować je w projekcie Wstążki. Trzeba także określić elementy dla każdego procesu roboczego, w którym mają być one zaimplementowane w postaci hierarchicznej struktury.

W teorii osiągnięcie wymienionych celów projektowych może być proste, jednak w praktyce podczas projektowania Wstążki zwykle będziemy zmuszeni do stosowania kompromisów. Warto spojrzeć na przykładowy projekt Wstążki, który utworzyliśmy dla rozwiązania służącego do monitorowania wydatków budżetowych w różnych działach, projektach i dla odmiennych klientów (rysunek 10.2).



RYSUNEK 10.2. Projekt Wstążki

W powyższym projekcie zidentyfikowaliśmy cztery główne poziome procesy robocze, które następnie zostały podzielone na cztery oddzielne grupy na Wstążce. Oto one.

1. ***Data Selection***, który określa parametry identyfikujące żądane dane.
2. ***Data Acquisition*** pozwala na wskazanie źródła danych, które będzie użyte do pobierania danych.
3. ***Reports*** określa żądane raporty i pozwala na ich utworzenie.
4. ***Distribution and Publishing*** wysyła raport do ustalonej liczby odbiorców lub publikuje raport za pomocą usługi Microsoft Office SharePoint Service (MOSS).

Na rysunku 10.2 pokazano również, że wewnątrz każdego procesu roboczego zidentyfikowaliśmy elementy niezbędne do zaimplementowania w pionowej hierarchicznej strukturze.

Używanie karty Dodatki

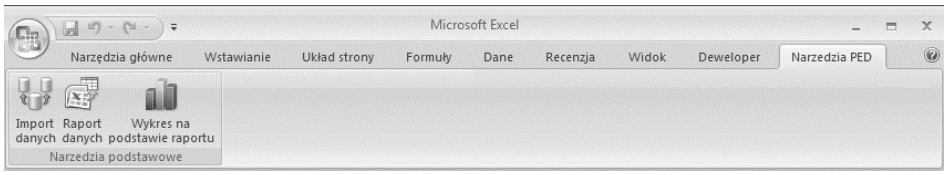
Z naszego doświadczenia wynika, że karta *Dodatki* jest lokalizacją zalecaną dla interfejsu użytkownika dodatków ogólnych. Stosowanie karty *Dodatki* dla prostych interfejsów narzędzi pozwala na ograniczenie liczby wyświetlanych kart i ogólne oczyszczenie interfejsu użytkownika Excela.

Karta *Dodatki* powinna być używana, gdy dodatek wymaga wyświetlenia tylko kilku kontrolek. Ponieważ dodatki tworzące jedynie pasek narzędziowy na podstawie interfejsu użytkownika są automatycznie umieszczane pod kartą *Dodatki*, zachowujemy kontrolę nad tym, czy poszczególne dodatki mogą być umieszczone w wymienionym miejscu. Wiele narzędzi firm zewnętrznych, np. SnagIt, przeznaczonych do działania w różnych wersjach Excela, celowo umieszcza pasek narzędziowy pod kartą *Dodatki*.

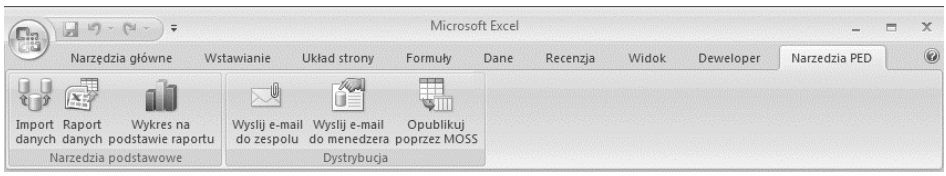
Współdzielenie własnych kart i grup przez wiele dodatków

W większości przypadków tworzymy konkretne rozwiązania, z których każde dodaje na Wstążce własne karty, grupy i kontrolki. Jednak w pewnych sytuacjach chcemy sprawić, by elementy te były współdzielone przez różne dodatki. Przykładowo możemy mieć dodatek główny tworzący podstawową kartę, grupę i strukturę menu, do których inne powiązane dodatki będą dodawały własne elementy. Na rysunku 10.3 pokazano dodatek zawierający kartę, która ma być współdzielona przez różne dodatki.

Kiedy zostanie wczytany powiązany z nim dodatek, wykorzysta współdzieloną kartę do umieszczenia własnej grupy kontrolek, co pokazano na rysunku 10.4.



RYSUNEK 10.3. Współdzielona karta



RYSUNEK 10.4. Kontrolki innego dodatku dodane do współdzielonej karty

Taka możliwość może być użytecznym podejściem podczas tworzenia rozwiązań o dużym stopniu skalowalności. Kluczowym atrybutem wymaganym do utworzenia współdzielonej karty i zarządzania nią jest atrybut **namespace**. Za każdym razem gdy tworzymy własną Wstążkę, używamy przestrzeni nazw. Pierwszy wiersz we wszystkich plikach XML Wstążki zawiera odniesienie do przestrzeni nazw *http://schemas.microsoft.com/office/2006/01/customui*, tak jak przedstawiono poniżej:

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/
➥01/customui">
```

Prefiks `xmlns` jest skrótem od *XML NameSpace* (przestrzeń nazw XML). Firma Microsoft zdecydowała się na używanie adresów URL jako identyfikatorów przestrzeni nazw we wszystkich definicjach XML Wstążki. Przestrzeń nazw pozwala na dostosowanie opcji Wstążki we wszystkich programach Office. W kolejnym przykładzie utworzymy własną przestrzeń nazw pozwalającą na identyfikację i dodawanie kontrolek z wielu plików do własnej karty. Ten rodzaj rozwiązania nosi nazwę **karty współdzielonej**.

Rozwiązanie współdzielące kartę wymaga przynajmniej dwóch plików Excela. W omawianym przykładzie użyjemy dwóch dodatków. Pierwszy będzie działał w charakterze gospodarza dla karty współdzielonej, natomiast drugi umieści na karcie grupę własnych kontrolek. Na początek musimy utworzyć dodatek o nazwie *Shared Tab.xlam* i umieścić w nim definicję XML Wstążki przedstawioną w listingu 10.2.

Na początku kodu przedstawionego w listingu 10.2 tworzymy nową przestrzeń nazw `nsPED`. Do własnej karty dodaliśmy również nowy atrybut `idQ`, który oznacza **Qualified ID**. Wartość atrybutu `idQ` karty poprzedzimy

LISTING 10.2. Definicja XML Wstążki w pierwszym dodatku dla Shared Tab

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
      xmlns:nsPED="PED Namespace">
   <ribbon startFromScratch="false">
   <tabs>
      <tab idQ="nsPED:rxtabPED" label="Narzędzia PED" visible="1">
      <group id="rxgrpBasic" label="Narzędzia podstawowe" visible="1">
         <button id="rxbtnImportData"
            label="Import danych"
            screentip="Import danych."
            imageMso="DatabaseAccessBackEnd"
            size="large"
            tag="Import_Data"
            onAction="Shared_Basic_Tools_Click"/>
         <button id="rxbtnDataReport"
            label="Raport danych"
            screentip="Utworzenie raportu danych."
            imageMso="ImportExcel"
            size="large"
            tag="Data_Report"
            onAction="Shared_Basic_Tools_Click"/>
         <button id="rxbtnChartReport"
            label="Wykres na podstawie raportu"
            screentip="Utworzenie wykresu na podstawie raportu."
            imageMso="PivotChartType"
            size="large"
            tag="Chart_Report"
            onAction="Shared_Basic_Tools_Click"/>
      </group>
      </tab>
   </tabs>
   </ribbon>
</customUI>
```

prefiksem z naszej przestrzeni nazw `nsPed`. Poprzez określenie w ten sposób wartości atrybutu `idQ` umożliwiamy odwołanie się do danego obiektu karty z poziomu dowolnego pliku używającego naszej przestrzeni nazw. Pokazane tutaj podejście podczas tworzenia karty współdzielonej może być również zastosowane na współdzielonych grupach bądź kontrolkach.

Następnym krokiem jest dodanie do modułu kodu w dodatku globalnej procedury wywołania zwrotnego, którą przedstawiono w listingu 10.3. Procedura wywołania zwrotnego `Shared_Basic_Tools_Click` używa atrybutu `tag` wywołującej kontrolki w celu określenia przycisku, który spowodował wywołanie zwrotne.

LISTING 10.3. Procedura wywołania zwrotnego w pierwszym dodatku

```
Option Explicit
Sub Shared_Basic_Tools_click(control As IRibbonControl)
Const sMESSAGE As String = "Chcesz "
Dim sActivity As String
```

```
Select Case control.Tag
   Case "Import_Data": sActivity = "zaimportować dane."
   Case "Data_Report": sActivity = "utworzyć raport danych."
   Case "Chart_Report": sActivity = "utworzyć wykres na podstawie
   ↪raportu."
End Select
MsgBox sMESSAGE & sActivity
End Sub
```

Po wczytaniu dodatku *Shared Tab.xlam* nastąpi utworzenie karty *Narzędzia PED* pokazanej na rysunku 10.3. Aby zilustrować współdzieloną naturę tej karty, utworzymy drugi dodatek i nadamy mu nazwę *Add Group Shared Tab.xlam*. Następnie do drugiego dodatku dodamy plik definicji XML Wstążki przedstawiony w listingu 10.4.

LISTING 10.4. Definicja XML Wstążki w drugim dodatku dla Shared Tab

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
      xmlns:nsPED="PED Namespace">
   <ribbon startFromScratch="false">
   <tabs>
      <tab idQ="nsPED:rxtabPED" label="Narzędzia PED" visible="1">
      <group id="rxgrpDistribution" label="Dystrybucja">
         <button id="rxbtnEmailTeam"
            label="Wyślij e-mail do zespołu"
            screentip="Wyślij e-mail do zespołu."
            imageMso="AttachItem"
            size="large"
            tag="Email_Team"
            onAction="Shared_Distribution_Click"/>
         <button id="rxbtnEmailManager"
            label="Wyślij e-mail do menedżera"
            screentip="Wyślij e-mail do menedżera."
            imageMso="FileManageMenu"
            size="large"
            tag="Email_Manager"
            onAction="Shared_Distribution_Click"/>
         <button id="rxbtnPublishMoss"
            label="Opublikuj poprzez MOSS"
            screentip="Opublikuj poprzez MOSS."
            imageMso="ExportSharePointList"
            size="large"
            tag="Publish_MOSS"
            onAction="Shared_Distribution_Click"/>
      </group>
      </tab>
   </tabs>
   </ribbon>
</customUI>
```

Porównując definicję XML Wstążki przedstawioną w listingu 10.4 z definicją przedstawioną w listingu 10.3, zauważamy, że współdzielą tę samą przestrzeń nazw `nsPED` oraz używają tego samego atrybutu `idQ` i nazw etykiety

karty współdzielonej. Jest to wymagane do współdzielenia karty przez wiele plików. Na końcu drugi dodatek uzupełnimy globalną procedurą wywołania zwrotnego, którą przedstawiono w listingu 10.5.

LISTING 10.5. Procedura wywołania zwrotnego w drugim dodatku

```
Option Explicit
Sub Shared_Distribution_Click(control As IRibbonControl)
Const sMESSAGE As String = "Chcesz "
Dim sActivity As String
Select Case control.Tag
   Case "Email_Team": sActivity = "wysłać raport do członków zespołu."
   Case "Email_Manager": sActivity = "wysłać raport menedżerowi."
   Case "Publish_MOSS": sActivity = "opublikować raport poprzez MOSS."
End Select
MsgBox sMESSAGE & sActivity
End Sub
```

Po wczytaniu dodatków kontrolki Wstążki utworzone przez obydwa dodatki zostaną wyświetlone na tej samej karcie współdzielonej *Narzędzia PED*, którą pokazano na rysunku 10.4. Oba utworzone dodatki można także wczytać niezależnie. Każdy z nich ma możliwość utworzenia karty współdzielonej i umieszczenia na niej kontrolek. Jednak to prowadzi do małego błędu.

Ponieważ na naszej karcie współdzielonej brakuje rodzimego mechanizmu kontroli sekwencji grup, kolejność wyświetlania na niej grup będzie zależała od kolejności otwierania dodatków korzystających z tej karty współdzielonej. Najlepszym rozwiązaniem tego problemu będzie użycie pliku głównego kontrolującego kolejność wczytywania powiązanych z nim plików.

Dodatek utworzony w tym przykładzie został zamieszczony w katalogu *\Koncepcje\Rozdzial10* na płycie CD dołączonej do książki.

Atrybut keytip

Spotkaliśmy się już z pewnymi imponującymi rozwiązaniami Wstążki, w których nie dostarczono żadnych atrybutów **keytip**. To poważne ograniczenie. Naciśnięcie klawisza *Alt* powoduje przejście klawiatury do **trybu nawigacji**. W tym trybie użytkownicy mogą bardzo łatwo poruszać się po Wstążce, po prostu naciskając klawisze na klawiaturze.

Największym problemem z atrybutem `keytip` nie jest proces jego dodawania do definicji XML Wstążki — to bardzo łatwe zadanie. Problemem jest określenie odpowiednich klawiszy do wykorzystania. Tak wiele kombinacji klawiszy już wykorzystano we wbudowanych funkcjach Excela, że niezbyt dużo pozostało do wyboru. W aplikacjach dyktatorskich będzie to znacznie mniejszy problem, ale w dodatkach ogólnych i skoroszytach znalezienie dobrego rozwiązania może być dość trudne.

Zarządzanie własnymi obrazami kontrolek

Pakiet Office 2007 jest dostarczany z dużą ilością wbudowanych obrazów, które można wykorzystać we własnych kontrolkach. Dlatego też przed podjęciem decyzji o użyciu własnych obrazów warto przejrzeć te, które dostarczono w standardzie. Stosowanie wbudowanych obrazów pomaga również w zachowaniu znanego wyglądu interfejsu użytkownika. Zamiast np. używać własnego obrazu dla przycisku *Drukuj*, należy skorzystać z dostarczonego standardowo przez aplikację. Stosowanie wielu własnych obrazów może mieć negatywny wpływ na wydajność aplikacji.

Microsoft udostępnia do pobrania plik Excela o nazwie *Office2007Icons Gallery.xlsm*. Powoduje on umieszczenie na karcie *Deweloper* zestawu kolekcji obrazów, które ułatwiają odszukanie nazw obrazów wbudowanych w pakiet w celu ich użycia we własnych interfejsach użytkownika RibbonX. Inny użyteczny plik do pobrania to *2007OfficeControlIDsExcel2007.exe*; w nim znajdziemy listę identyfikatorów dla wszystkich wbudowanych kontrolek.

Jeżeli postanowisz, że konieczne jest zastosowanie własnych obrazów, wówczas zalecanym formatem pliku jest **PNG**. Silnik graficzny Wstążki został zaprojektowany do pracy z obrazami w pełnej palecie kolorów (24 bity), która mają również kanał alfa określający przezroczystość każdego piksela. Ponieważ format PNG obsługuje kanał alfa i pozwala na zachowanie względnie małej wielkości plików, będzie najlepszym wyborem. Zalecana wielkość własnych ikon to 16 na 16 pikseli (małe) lub 32 na 32 piksele (duże).

Za pomocą narzędzia Custom UI Editor można dodać własne obrazy do skoroszytów, ale nie można się do nich odwołać w pliku definicji XML Wstążki. Oznacza to konieczność użycia oddzielnych plików obrazów. W listingu 10.6 przedstawiono definicję XML Wstążki korzystającą z obrazów w formacie PNG za pomocą atrybutu `getImage`.

LISTING 10.6. Definicja XML Wstążki dla grafiki w formacie PNG

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
         <tab id="rxtabPED" label="Narzędzia PED" visible="1">
         <group id="rxgrpPED" label="Zestaw narzędzi PED" visible="1">
            <button id="rxbtnReport"
               label="Raport"
               screentip="Utwórz raport."
               getImage="GetImage"
               size="large"
               onAction="rxbtnReport_Click"/>
         </group>
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

Procedurę wywołania zwrotnego dla atrybutu `getImage` przedstawiono w listingu 10.7. Jedyny problem z formatem pliku PNG polega na tym, że musimy mieć własne rozwiązanie, aby wczytać obrazy w tym formacie. Wynika to z faktu, że wbudowana w Excel 2007 funkcja `LoadPicture` nie obsługuje formatu PNG. W listingu 10.7 użyliśmy własnej funkcji `Load ↪PictureGDI`, która pobiera jeden argument — pełną ścieżkę dostępu i nazwę pliku PNG przeznaczonego do wczytania.

LISTING 10.7. Wywołanie zwrotne dla atrybutu getImage

```
Sub GetImage(control As IRibbonControl, ByRef returnedVal)

   Set returnedVal = _
      LoadPictureGDI(ThisWorkbook.Path & "\Report.png")

End Sub
```

Przedstawiona powyżej funkcja używa pewnych API **GDI+** (***Graphics Device Interface***) do konwersji obrazu PNG na obiekt `IPicture`. Szczegółowe omówienie procesu tej konwersji wykracza poza zakres tematyczny rozdziału. Jednak kod własnej funkcji wczytującej obrazy PNG umieściliśmy w skoroszycie *Load PNG pictures.xlsm* znajdującym się w katalogu *\Koncepcje\Rozdzial10* na płycie CD dołączonej do książki.

Używanie globalnych procedur wywołań zwrotnych

Użycie globalnych procedur wywołań zwrotnych pozwala na jednoczesne obsłużenie wielu obiektów kontrolek. Gdy pojedyncza procedura wywołań zwrotnych obsługuje wiele powiązanych obiektów kontrolek, kod jest bardziej strukturalny, a jego obsługa wymaga mniejszej ilości czasu. W definicji XML Wstążki przedstawionej w listingu 10.8 atrybutom `onAction` wszystkich trzech przycisków przypisano tę samą procedurę wywołań zwrotnych o nazwie `PED_Click`.

LISTING 10.8. Używanie jednej procedury wywołania zwrotnego do obsługi kilku obiektów kontrolek

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
   <tabs>
      <tab id="rxtabPED" label="Narzędzia PED" visible="1">
         <group id="rxgrpPED" label="Zestaw narzędzi PED" visible="1">
            <button id="rxbtnImportData"
               label="Import danych"
               screentip="Import danych."
               imageMso="DatabaseAccessBackEnd"
               size="large"
```

```
            tag="ImportData"
            onAction="PED_click" />
         <button id="rxbtnDataReport"
            label="Utwórz raport"
            screentip="Utworzenie raportu na podstawie danych."
            imageMso="ImportExcel"
            size="large"
            tag="DataReport"
            onAction="PED_click" />
         <button id="rxbtnChartReport"
            label="Utwórz wykres"
            screentip="Utworzenie wykresu na podstawie raportu."
            imageMso="PivotChartType"
            size="large"
            tag="ChartReport"
            onAction="PED_click" />
      </group>
   </tab>
</tabs>
</ribbon>
</customUI>
```

Procedura wywołania zwrotnego `PED_Click` używa atrybutu `tag` wywołującej kontrolki w celu określenia przycisku, który spowodował wywołanie procedury. Procedurę obsługi przedstawiono w listingu 10.9.

LISTING 10.9. Wspólna procedura wywołania zwrotnego

```
Sub PED_click(control As IRibbonControl)
   Select Case control.Tag
      Case "ImportData": Import_Data
      Case "DataReport": Create_Data_Report
      Case "ChartReport": Create_Chart_Report
   End Select
End Sub
```

Unieważnianie

Unieważnianie jest procesem intensywnie wykorzystującym zasoby i podczas jego używania należy zachować ostrożność. Gdy tylko będzie to możliwe, warto unieważniać określone kontrolki zamiast całej Wstążki, gdyż to drugie podejście ma bardzo negatywny wpływ na wydajność aplikacji. Unieważnianie kontrolek nie powoduje ich ponownego wczytania, a jedynie odświeżenie.

Wydaje się, że w wywołaniu `getEnabled` bieżącej wersji Wstążki istnieje błąd. Jeżeli do dynamicznego włączania lub wyłączania określonych kontrolek Wstążki użyjesz wywołania zwrotnego `getEnabled`, wywołanie zwrotne może nie zostać prawidłowo wywołane, o ile nie nastąpi unieważnienie całej Wstążki zamiast poszczególnych kontrolek.

Spójrzmy bliżej na proces unieważniania w definicji XML Wstążki i jego wymagania w VBA. Przede wszystkim potrzebujemy zmiennej VBA, która przedstawia obiekt `IRibbonUI`. Aby ją otrzymać, musimy określić procedurę wywołania zwrotnego dla atrybutu `onLoad`, co przedstawiono w listingu 10.10.

LISTING 10.10. Definicja XML Wstążki służąca do unieważniania przycisku

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
      onLoad="rxRibbonUI_onLoad">
   <ribbon>
      <tabs>
         <tab id="rxtabInvalidate" label="PED">
            <group id="rxgrpPed" label="Invalidate">
               <button id="rxbtnPED"
                  getLabel ="rxbtn_GetTime"
                  screentip="Wyświetl godzinę."
                  imageMso="DateAndTimeInsert"
                  size="large"
                  onAction="rxbtn_Invalidate_Click"/>
            </group>
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

W powyższym przykładzie podczas każdego kliknięcia przycisku odświeżamy datę i godzinę, które są wyświetlane przez kontrolkę przycisku. Dlatego też potrzebujemy procedury wywołania zwrotnego unieważniającej kontrolkę przycisku, tak aby było możliwe odświeżenie jej zawartości. Ponadto potrzebna jest procedura wywołania zwrotnego służąca do wyświetlenia przez kontrolkę przycisku bieżącej daty i godziny. W listingu 10.10 użyliśmy atrybutu `getLabel` do pobrania daty i godziny. Natomiast atrybut `onAction` jest wykorzystywany do unieważniania kontrolki przycisku. Wywołanie zwrotne `onLoad` i dwa wywołania zwrotne przycisku zostały przedstawione w listingu 10.11.

LISTING 10.11. Wywołania zwrotne służące do unieważniania przycisku

```
Private m_rxRibbonUI As IRibbonUI

' Wywołanie zwrotne dla customUI.onLoad.
Sub rxRibbonUI_onLoad(Ribbon As IRibbonUI)
   Set m_rxRibbonUI = Ribbon
End Sub

' Wywołanie zwrotne dla rxbtnPED onAction.
Sub rxbtn_Invalidate_Click(control As IRibbonControl)
   m_rxRibbonUI.InvalidateControl control.ID
End Sub

' Wywołanie zwrotne dla rxbtnPED getLabel.
```

```
Sub rxbtn_GetTime(control As IRibbonControl, ByRef returnedValue)
   returnedValue = CStr(Now())
End Sub
```

Przykładowy plik zawierający ten kod znajduje się w katalogu *\Koncepcje\Rozdzial10* na płycie CD dołączonej do książki. Microsoft opublikował także dokument poruszający temat projektowania Wstążki. Można go znaleźć po przeszukaniu witryny Microsoft pod kątem wyrażenia *2007 Office System Document: UI Style Guide for Solutions and Add-Ins*.

Dostosowanie opcji Wstążki bazującej na tabeli

Podczas budowania interfejsów użytkownika paska poleceń proces bazujący na tabeli jest faktycznie standardem, zwłaszcza w przypadku rozwiązań o dużym poziomie skalowalności. Moglibyśmy założyć, że nie ma powodu, aby takie samo podejście zastosować w trakcie dostosowywania opcji Wstążki.

Jednak dwa główne ograniczenia techniczne uniemożliwiają tworzenie Wstążki bazującej na tabeli. Po pierwsze, w VBA nie mamy dostępu do modelu obiektowego w celu manipulowania Wstążką. Po drugie, środowisko uruchomieniowe VBA nie może dostarczyć XML dla definicji Wstążki. Bardzo użyteczne byłoby zobaczenie, że standardowe zdarzenie `GetCustomUI` zostało dodane do obiektu `Workbook`, a metoda `CreateCustomerUI` — do obiektu `CommandBars`.

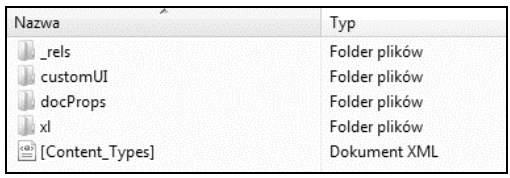
Użycie Visual Studio Tools for Office System (VSTO) pozwala na uzyskanie dostępu do narzędzia Visual Designer dla Wstążki, które w tworzonym rozwiązaniu umożliwia wizualną manipulację tym interfejsem. Narzędzie to na podstawie projektu graficznego automatycznie generuje plik definicji XML Wstążki, wywołania zwrotne oraz inne komponenty interfejsu użytkownika. Innymi słowy, VSTO oferuje pełną obsługę dostosowywania opcji Wstążki, podczas gdy VBA w ogóle nie dostarcza takiego wsparcia.

Wobec takiego stanu rzeczy możemy zakładać, że Microsoft zepchnął VBA do roli drugorzędnej. W kolejnym rozdziale przeanalizujemy możliwości, jakie obecnie mamy do wykorzystania w poszczególnych plikach Excela podczas pracy z **customUI XML** przy użyciu VBA.

Uzyskanie dostępu do elementu customUI XML

Aby uzyskać dostęp do elementu `customUI` XML w pliku Excela, konieczna jest tymczasowa zmiana rozszerzenia na *.zip*, dostosowanie wybranych opcji, a następnie przywrócenie rozszerzeniu jego wartości początkowej. Tymczasowa zmiana rozszerzenia pliku na *.zip* powoduje, że składniki XML nowego formatu pliku stają się widoczne i dostępne z poziomu Eksploratora Windows, co pokazano na rysunku 10.5. W trakcie ręcznego dodawania własnych opcji

na Wstążce za pomocą narzędzia Custom UI Editor automatycznie tworzony jest katalog *customUI*. Musi on istnieć, jeśli chcemy, aby wprowadzone modyfikacje opcji działały prawidłowo. Jeżeli katalog nie będzie istniał, musi zostać zbudowany przed utworzeniem i zapisaniem w nim pliku *customUI.xml*.



RYSUNEK 10.5. Zawartość archiwum ZIP

Wszystkie definicje XML Wstążki muszą znajdować się w pliku *customUI.xml*. Ponadto XML Wstążki musi spełniać kryteria określone w schemacie definicji XML (**XSD — XML Schema Definition**) opisanym w pliku *customUI.xsd*. Plik XML to zwykły plik tekstowy z rozszerzeniem *.xml* zamiast *.txt*. Z tego powodu pliki *customUI.xml* możemy tworzyć w taki sam sposób jak zwykłe pliki tekstowe. Jeżeli masz wiedzę wystarczającą do pracy z analizatorem składni XML, np. Microsoft XML, może to być lepszym rozwiązaniem. Podczas uaktualniania pliku *customUI.xml* znacznie prostsze i szybsze jest nadpisanie bieżącego pliku zupełnie nową wersją zamiast modyfikacji zawartego w nim kodu.

Z technicznego punktu widzenia możliwe jest utworzenie w Excelu narzędzia służącego do generowania definicji XML Wstążki na podstawie wpisów w tabeli skoroszytu, podobnie jak podczas budowania bazujących na tabeli interfejsów użytkownika paska poleceń. Znacznie lepszym rozwiązaniem będzie opracowanie tego rodzaju narzędzia dla platformy .NET, ponieważ oferuje ona daleko idącą obsługę pracy z XML oraz ze Wstążką.

Zaawansowane rozwiązywanie problemów

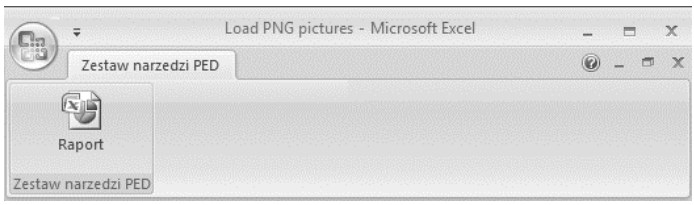
Wstążka jest krytykowana za brak modelu obiektowego w VBA. Jednak zdecydowanie częściej wskazuje się na fakt, że oferuje elastyczność znacznie mniejszą niż stary interfejs użytkownika paska poleceń. Wiele poleceń jest zbyt głęboko zaszytych w interfejsie użytkownika, zatem nie są łatwo dostępne. Z punktu widzenia procesu, polecenia nie są dostępne w logiczny sposób.

Własne rozwiązania Wstążki mogą pomóc w pokonaniu niektórych ograniczeń. Natomiast inne ograniczenia, takie jak związane z używaniem kontrolek dodatkowych, mogą być rozwiązane tylko przy użyciu platformy .NET oraz narzędzi firm trzecich. Pozostałe ograniczenia, np. brak możliwości

tworzenia własnych bądź „pływających" pasków narzędziowych, wynikają z budowy architektury Wstążki i nie mogą być pokonane. Poniżej przedstawiono omówienie niektórych kwestii, z którymi programiści borykają się najczęściej, oraz sposoby ich rozwiązania.

Tworzenie Wstążki dla aplikacji dyktatorskiej

W aplikacji dyktatorskiej całkowicie usuwamy domyślną Wstążkę i zastępujemy ją własną. Standardowym podejściem jest ustawienie w pliku definicji XML Wstążki atrybutu `startFromScratch="true"`, co daje interfejs pokazany na rysunku 10.6.



RYSUNEK 10.6. Podstawowa Wstążka dla aplikacji dyktatorskiej

Główną zaletą takiego podejścia pozostaje brak konieczności używania kodu VBA do ustawienia Wstążki po otworzeniu skoroszytu. Kod VBA nie będzie także wymagany do przywrócenia ustawień domyślnych interfejsu po zamknięciu aplikacji. Wszystko jest wykonywane automatycznie podczas otwierania i zamykania skoroszytu.

Na rysunku 10.6 widać, że ***Przycisk pakietu Office*** jest nadal dostępny, podobnie jak część podstawowych poleceń łącznie z ***Paskiem narzędzi Szybki dostęp*** (**QAT**, czyli **Quick Access Toolbar**). Dwa elementy pozostające w przycisku Office po użyciu atrybutu `startFromScratch="true"` to przyciski *Opcje programu Excel* oraz *Zakończ program Excel*. Być może w aplikacji dyktatorskiej trzeba będzie uniemożliwić dostęp do wymienionych funkcji.

Niestety, ukrycie dwóch omawianych kontrolek jest niemożliwe, więc najlepszym rozwiązaniem pozostaje ich zablokowanie. Chociaż fizycznie są umieszczone w menu Office, RibbonX nie pozwala na manipulację nimi z poziomu elementu `officeMenu` w definicji XML Wstążki. W zamian trzeba wykorzystać element `command`, co przedstawiono w listingu 10.12.

LISTING 10.12. Definicja XML Wstążki służąca do wyłączenia przycisków Opcje programu Excel oraz Zakończ program Excel

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
...
   <commands>
```

```
      <command idMso="FileExit" enabled="false"/>
      <command idMso="ApplicationOptionsDialog" enabled="false"/>
   </commands>
...
</customUI>
```

W przycisku Office nadal pozostają dostępne opcje *Nowy*, *Otwórz* i *Zapisz*. Jeżeli i one mają zostać ukryte, musimy użyć elementu `officeMenu`, tak jak przedstawiono w listingu 10.13.

LISTING 10.13. Definicja XML Wstążki służąca do ukrywania poleceń Nowy, Otwórz i Zapisz

```
<officeMenu>
   <button idMso="FileNew" visible="false" />
   <button idMso="FileOpen" visible="false" />
   <button idMso="FileSave" visible="false" />
</officeMenu>
```

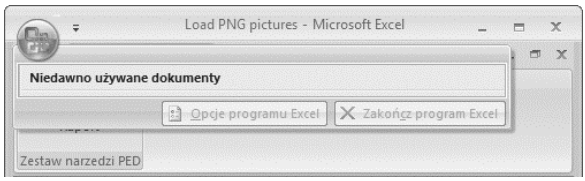
Do ukrycia w menu Office pozostał już tylko element — ***Niedawno używane dokumenty*** — znany również jako lista niedawno używanych plików (**MRU**, czyli **Most Recently Used**). Ukrycia nie możemy przeprowadzić za pomocą definicji XML Wstążki, natomiast w VBA można tylko wyczyścić listę, a nie usunąć zupełnie. Aby wyczyścić listę podczas uruchomienia aplikacji, należy użyć pierwszego bloku kodu przedstawionego w listingu 10.14. Podczas zamykania aplikacji trzeba przywrócić wartość maksymalnej liczby dostępnych plików. Do tego celu wykorzystamy drugi fragment kodu przedstawiony w listingu 10.14.

LISTING 10.14. Kod służący do czyszczenia listy ostatnio używanych plików

```
' Zmienna modułu przechowująca maksymalną liczbę ostatnio używanych plików.
Dim miNumberOfFiles As Integer
   '...
   With Application.RecentFiles
      ' Pobranie maksymalnej liczby dostępnych plików.
      miNumberOfFiles = .Maximum
      ' Wyczyszczenie listy.
      .Maximum = 0
   End With

' ...
' Przywrócenie maksymalnej liczby ostatnio używanych plików.
Application.RecentFiles.Maximum = miNumberOfFiles
```

Po uruchomieniu kodu zmodyfikowany *Przycisk pakietu Office* będzie wyglądał, tak jak pokazano na rysunku 10.7.



RYSUNEK 10.7. Menu Office po modyfikacji

Należy pamiętać, że ustawienie wartości zero maksymalnej liczbie ostatnio używanych plików powoduje trwałe wyczyszczenie listy MRU. Wielu użytkowników polega w swojej pracy na tej liście i mogą być niezadowoleni, jeśli lista nie zostanie przywrócona. Mamy zatem problem. Lista MRU (jak również maksymalna liczba plików znajdujących się na liście) jest przechowywania w następującym kluczu rejestru Windows:

```
HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\Excel\File MRU
```

Z technicznego punktu widzenia, możliwe jest zachowanie, a następnie przywrócenie klucza rejestru, aby nie spowodować zniszczenia listy MRU użytkownika. Jednak dość powszechną praktyką, zwłaszcza w środowiskach korporacyjnych, jest uniemożliwianie programom dostępu do rejestru. Takie ograniczenie nie pozwala na zbudowanie naprawdę solidnego rozwiązania problemu trwałego czyszczenia listy MRU.

Zazwyczaj chcemy również usunąć domyślne menu kontekstowe wyświetlane po kliknięciu prawym przyciskiem myszy w skoroszycie. To zadanie jest bardzo łatwe do wykonania za pomocą VBA, co przedstawiono w listingu 10.15.

LISTING 10.15. Usunięcie w Excelu 2007 menu kontekstowego wyświetlanego po kliknięciu prawym przyciskiem myszy

```
Private Sub Workbook_SheetBeforeRightClick(ByVal Sh As Object, _
      ByVal Target As Range, _
      Cancel As Boolean)

   Cancel = True

End Sub
```

Następnie domyślne menu kontekstowe zastępujemy własnym paskiem poleceń. W poprzednim zdaniu nie popełniliśmy pomyłki, mówiąc „o pasku poleceń". Nawet w Excelu 2007 menu kontekstowe nadal jest takim samym paskiem poleceń jak we wcześniejszych wersjach Excela.

Wspomniane menu kontekstowe zostało wprawdzie rozbudowane o pewne funkcje dodatkowe, do których nie mamy dostępu, ale nadal pozostaje pas-

kiem poleceń. W tworzonej aplikacji zastępujemy je więc własnym paskiem poleceń, używając poniższego wiersza kodu:

```
Application.CommandBars("CustomMenu").ShowPopup
```

Możemy jedynie zgadywać, że Microsoft ma zamiar połączyć tę część interfejsu użytkownika Excela z nowym modelem Wstążki. Prawdopodobnie zabrakło czasu, aby zrobić to przed wprowadzeniem Excela 2007, ale wszystko może się zmienić w przyszłych wersjach aplikacji. Po wprowadzeniu powyższych zmian pozostało tylko dostosowanie Excela w ten sam sposób, który przedstawiono w rozdziale 6.

Ukrycie Wstążki

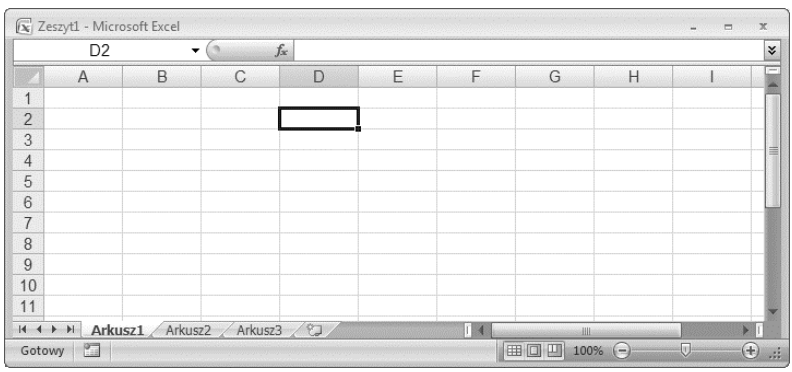
W pewnych sytuacjach trzeba całkowicie ukryć Wstążkę. Zadanie to można przeprowadzić za pomocą VBA poprzez wykonanie w pierwszej kolejności makra przedstawionego w listingu 10.16. Aby przywrócić Wstążkę, należy skorzystać z VBA do wykonania drugiego makra XLM przedstawionego w listingu 10.16.

LISTING 10.16. Ukrywanie i odkrywanie Wstążki

```
' Ukrycie Wstążki.
Application.ExecuteExcel4Macro "Show.Toolbar(""Ribbon"", False)"

' Przywrócenie Wstążki.
Application.ExecuteExcel4Macro "Show.Toolbar(""Ribbon"", True)"
```

Na rysunku 10.8 pokazano wynik użycia powyższej techniki do ukrycia Wstążki.



RYSUNEK 10.8. Interfejs użytkownika Excela z całkowicie ukrytą Wstążką

Dwie implikacje powyższej techniki wymagają dalszych objaśnień. Po pierwsze, wykonanie wskazanego makra XLM w celu ukrycia Wstążki ma wpływ na wszystkie otwarte skoroszyty w bieżącej sesji Excela. Po drugie, technika opiera się na obsłudze makr XLM w Excelu. XLM to język makr używany do tworzenia makr i aplikacji Excela przed wprowadzeniem VBA. Minęło już ponad piętnaście lat do oficjalnego zastąpienia XLM przez VBA i zachodzi duże prawdopodobieństwo, że Microsoft zakończy obsługę XLM w jednej z kolejnych wersji pakietu Office. Dlatego też, stosując powyższą technikę, nie można mieć pewności, że będzie działała w kolejnych wersjach Excela.

Określenie wielkości kontrolek comboBox, dropDown i editBox

W pewnych sytuacjach trzeba kontrolować wielkość kontrolki `comboBox` oraz powiązanych z nią kontrolek `dropDown` i `editBox`. W tym celu używamy atrybutu `sizeString`, ale zamiast liczby, takiej jak `10` lub `20`, podajemy ciąg tekstowy zawierający maksymalną liczbę znaków, które chcemy wyświetlić. Znaki użyte w tym ciągu tekstowym nie są ważne, natomiast czcionka zastosowana w kontrolce wpływa na jej rzeczywistą szerokość.

Jeśli chcemy ograniczyć liczbę znaków wprowadzonych w kontrolce, możemy użyć atrybutu `maxLength` wraz z liczbą, np. `5` lub `8`. W listingu 10.17 ustawiono szerokość kontrolki `comboBox` na siedem znaków, a maksymalną liczbę znaków na osiem.

LISTING 10.17. Definicja XML Wstążki służąca do ustawienia wielkości kontrolki listy rozwijanej

```
<comboBox id="rxcboName"
   label="Name:"
   sizeString="xxxxxxx"
   maxLength="8"
   screentip="Proszę podać imię."
   onChange="rxcboName_OnChange"/>
```

Nawigacja po kartach

W rozwiązaniach z dużą ilością kart w interfejsie użytkownika skoroszytu lub z ukrytym paskiem kart może wystąpić konieczność dostarczenia własnej funkcji służącej do nawigacji po kartach. W omawianym przykładzie mamy sześć kart, z których jedna pozostaje ukryta. Dlatego też musimy utworzyć listę arkuszy zawierającą pięć widocznych arkuszy.

W omawianym przykładzie użyjemy kontrolki `dropDown` zamiast `combo`
`↪Box` głównie dlatego, że kontrolka `comboBox` pozwala użytkownikom na wprowadzanie nowych wpisów w kontrolce, a na to nie możemy się zgodzić.

Jednak problem z kontrolką `dropDown` polega na tym, że — w przeciwieństwie do `comboBox` — nie podaje nazwy zaznaczonego elementu, a jedynie jego identyfikator i numer indeksu.

Chociaż powinniśmy dążyć do tworzenia w pełni dynamicznych aplikacji, wiąże się to z kosztem, jakim jest konieczność napisania kodu dla wszelkich możliwych sytuacji. W rzeczywistych aplikacjach skłaniamy się więc do tworzenia półstałych aplikacji, gdzie pewne elementy pozostają stałe, natomiast inne są dynamiczne. W ten sposób zbudowaliśmy także rozwiązanie w omawianym przykładzie. Ponieważ dysponujemy ustaloną listą arkuszy, których nazwy nie ulegają zmianie podczas działania aplikacji, nie trzeba unieważniać kontrolki `dropDown` w trakcie działania programu. Natomiast nazwy skoroszytów nie zostały zapisane na stałe, co ma ułatwić ich przyszłą obsługę.

Arkusz o nazwie *Hidden* pozostaje ukryty i dlatego zawsze powinien być wykluczony z listy. Definicja XML Wstążki dla omawianego przykładu została przedstawiona w listingu 10.18. Atrybutu `getItemCount` użyliśmy w celu pobrania liczby nazw arkuszy, podczas gdy atrybut `getItemLabel` wypełnia kontrolkę `dropDown` listą nazw. Stosując oba atrybuty, zyskujemy nieco bardziej dynamiczne rozwiązanie.

LISTING 10.18. Definicja XML Wstążki służąca do nawigacji między arkuszami

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui">
   <ribbon startFromScratch="false">
      <tabs>
         <tab id="rxtabPED" label="Nawigacja po arkuszach">
            <group id="rxgrpDropDowns" label="Nawigacja">
               <dropDown id="rxddSheetNavigation"
                  label="Przejdź do:"
                  getItemCount="rxdd_ItemCount"
                  getItemLabel="rxdd_ListItem"
                  onAction="rxdd_Item_Selected"/>
            </group>
         </tab>
      </tabs>
   </ribbon>
</customUI>
```

Wymagane procedury wywołań zwrotnych zostały przedstawione w listingu 10.19. Wywołanie zwrotne `rxdd_ItemCount` zwraca liczbę arkuszy widocznych w skoroszycie. W ten sposób Excel zostaje poinformowany, ile razy powinien wykonać wywołanie zwrotne `rxdd_ListItem`. Wywołanie to dodaje do kontrolki `dropDown` nazwę każdego widocznego arkusza. Podczas pobierania nazwy zaznaczonego arkusza dodajemy `1` do indeksu listy przekazywanego do VBA. Wynika to z faktu, że wartości indeksu tablicy kolekcji arkuszy rozpoczynają się od jednego, podczas gdy VBA operuje na tablicach, których numeracja rozpoczyna się od zera.

LISTING 10.19. Wywołania zwrotne do nawigacji między arkuszami

```
Option Explicit

' Wywołanie zwrotne dla rxddSheetNavigation getItemCount.
Sub rxdd_ItemCount(control As IRibbonControl, ByRef returnedVal)

   Dim lCount As Long
   Dim wksSheet As Worksheet

   Set mwkbNavigation = ThisWorkbook

   ' Pobranie liczby widocznych arkuszy.
   For Each wksSheet In mwkbNavigation.Worksheets
      If wksSheet.Visible = xlSheetVisible Then
         lCount = lCount + 1
      End If
   Next wksSheet

   ' Wymiary tablicy arkuszy.
   returnedVal = lCount

End Sub

' Wywołanie zwrotne dla rxddSheetNavigation getItemLabel.
Sub rxdd_ListItem(control As IRibbonControl, index As Integer, _
      ByRef returnedVal)

   ' Wypełnienie kontrolki dropDown nazwami arkuszy.
   If mwkbNavigation.Worksheets(index + 1).Visible = _
         xlSheetVisible Then

      returnedVal = mwkbNavigation.Worksheets(index + 1).Name

   End If

End Sub

' Wywołanie zwrotne dla rxddSheetNavigation onAction.
Sub rxdd_Item_Selected(control As IRibbonControl, id As String, _
      index As Integer)

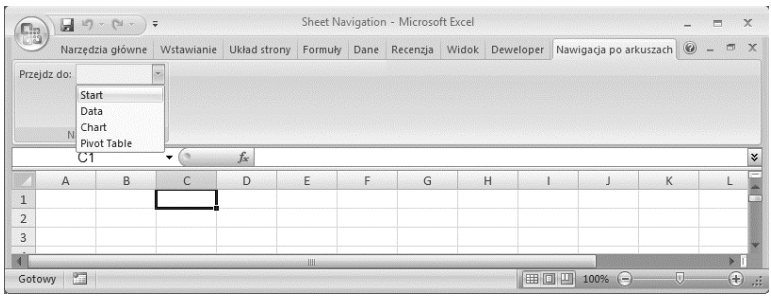
   Dim sSheetName As String

   ' Pobranie nazwy aktualnie zaznaczonego arkusza.
   sSheetName = mwkbNavigation.Worksheets(index + 1).Name

   ' Aktywacja zaznaczonej nazwy.
   mwkbNavigation.Worksheets(sSheetName).Activate

End Sub
```

Po uruchomieniu skoroszytu następuje utworzenie nowej karty *Nawigacja po arkuszach*. Użytkownik może przejść do dowolnego arkusza poprzez wybranie jego nazwy z listy wyświetlanej przez kontrolkę `dropDown`, co pokazano na rysunku 10.9.



RYSUNEK 10.9. Nawigacja po arkuszach

Skoroszyt zawierający omówiony przykład znajduje się w katalogu *\Koncepcje\Rozdzial10* na płycie CD dołączonej do książki.

Używanie szablonów

Stosując szablony, można na etapie tworzenia aplikacji zaoszczędzić sporo czasu. Narzędzie Custom UI Editor znacznie ułatwia tworzenie i używanie dostosowanych do własnych potrzeb szablonów XML Wstążki.

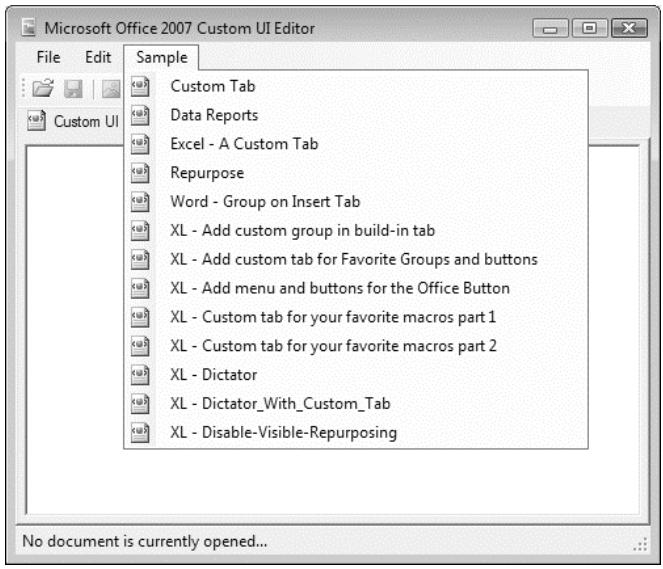
1. W narzędziu Custom UI Editor należy utworzyć definicję XML Wstążki i upewnić się, że jest prawidłowo zbudowana.
2. Następnie zawartość XML trzeba skopiować do edytora tekstowego, takiego jak Notatnik, i zapisać w katalogu *\Program Files\Custom UIEditor\Samples* jako plik z rozszerzeniem *.xml*.

Na rysunku 10.10 pokazano wygląd menu *Sample* w narzędziu Custom UI Editor po dodaniu kilku szablonów. Wybranie szablonu powoduje skopiowanie kodu szablonu XML na kartę *Custom UI*.

Dalsze pozycje do czytania

RibbonX: Customizing the Office 2007 Ribbon

Autorzy: Robert Martin, Ken Puls i Teresa Hennig
ISBN 978-0-470-191111-8



RYSUNEK 10.10. Własne szablony w narzędziu Custom UI Editor

Ponieważ Wstążka jest funkcją dostępną w całym pakiecie Office, więc nie ma książki poświęconej jedynie Wstążce w Excelu. Ta książka porusza temat dostosowania Wstążki do własnych potrzeb w aplikacjach Excel, Word i Access. Została napisana w prosty, łatwy do zrozumienia sposób i powinna być uznana za „biblię" RibbonX.

Portale internetowe

XML in Office Developer Portal

Portal XML in Office Developer na witrynie Microsoftu stanowi dobry punkt wyjścia do uzyskania większej ilości informacji na temat formatu Open XML. Warto odwiedzić stronę *http://msdn.microsoft.com/en-us/office/aa905545.aspx*.

OpenXMLDeveloper.org

Inna dobra witryna poświęcona Open XML to OpenXMLDeveloper.org dostępna pod adresem *http://openxmldeveloper.org/default.aspx*.

The Office Fluent User Interface Developer Portal

Portal The Office Fluent User Interface Developer na witrynie Microsoftu oferuje dostęp do wyczerpujących informacji na temat RibbonX. Warto odwiedzić stronę *http://msdn.microsoft.com/en-us/office/aa905530.aspx*.

Wnioski

W tym rozdziale przedstawiono niektóre z najważniejszych praktyk stosowanych podczas projektowania i programowania RibbonX. Zaprezentowane najlepsze praktyki będą ewoluowały wraz z zyskaniem coraz większego doświadczenia na tych polach. Na wiele sposobów RibbonX jest nową i interesującą technologią, choć obecnie ma pewne poważne ograniczenia podczas pracy z poziomu VBA. Prawdopodobnie najpoważniejszym ograniczeniem jest brak możliwości użycia podejścia bazującego na tabeli, zwłaszcza w aplikacjach dyktatorskich.