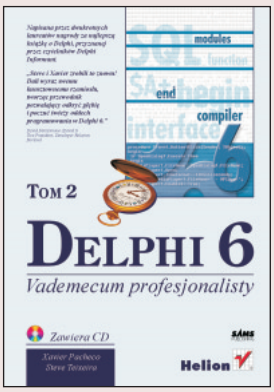
Delphi 6. Vademecum profesjonalisty. Tom II



Autorzy: Xavier Pacheco, Steve Teixeira
Tłumaczenie: Andrzej Grażyński
ISBN: 83-7197-745-X
Tytuł oryginału: Delphi 6 Developers Guide
Format: B5, stron: 496
oprawa twarda
Zawiera CD-ROM

Drugi tom „Delphi 6. Vademecum profesjonalisty", który oddajemy do rąk Czytelników, poświęcony jest zaawansowanym mechanizmom Delphi, związanym z programowaniem systemowym, nowoczesną obsługą baz danych i tworzeniem aplikacji internetowych. Jego treść wyraźnie oddaje specyficzną rolę Delphi 6 – jako świadectwa coraz ściślejszego integrowania się trzech najważniejszych filarów współczesnych zastosowań komputerów i zdobyczy informatyki: wizualnego projektowania aplikacji, efektywnego przetwarzania dużej ilości danych o coraz większej złożoności oraz Internetu.

Treść książki rozpoczyna się od omówienia podstawowych zasad tworzenia i wykorzystywania pakietów Delphi. Aplikacje dawno już przestały być pojedynczymi „programami" – ich postępująca złożoność wymusza raczej budowę modułową, której najważniejszymi konsekwencjami są: łatwość utrzymania i rozbudowy oraz sprawniejsza dystrybucja. Pakiety stanowią także podstawowy „budulec" IDE, będąc „nośnikami" jego kluczowych elementów, jak komponenty, edytory komponentów i właściwości oraz zintegrowane narzędzia rozszerzające.

Mimo, iż Delphi uwalnia programistę od mnóstwa nużących, niskopoziomowych szczegółów Windows, nie jest ono w stanie zapewnić elastyczności wystarczającej do wykorzystania wszystkich możliwości Windows; programista zmuszony jest więc niekiedy do bezpośredniego posługiwania się funkcjami Win32 API, w czym Delphi mu bynajmniej nie przeszkadza, a raczej oferuje dodatkowe mechanizmy wspomagające. Czytelnicy znajdą w niniejszej książce kilka praktycznych przykładów wykorzystania Win32 API: obsługę zasobnika, zarządzanie paskami aplikacji, tworzenie i obsługę łączników powłoki oraz aplikacje rozszerzające jej funkcjonalność.

Jedną z największych zalet IDE Delphi jest możliwość jego rozbudowy przez użytkownika. Oprócz (wynikającej z obiektowego modelu programowania) możliwości definiowania nowych komponentów, możliwe jest także tworzenie specjalizowanych edytorów współpracujących z projektantem formularzy; ich integrację ze środowiskiem umożliwia interfejs Open Tools API, którego unowocześnioną – bo bazującą już całkowicie na interfejsach COM – wersję opisujemy w jednym z rozdziałów.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 23.

WebSnap

Jedną z nowości Delphi 6 jest technologia WebSnap, łącząca wszelkie zalety błyskawicznego projektowania aplikacji (RAD) z mechanizmami charakterystycznymi dla aplikacji serwera WWW. W związku z jej wszechstronnością, przejawiającą się m.in. we współpracy z dowolnym niemal źródłem — bazami danych, grafiką, „surowymi" plikami itp. oraz w obsłudze języków skryptowych, programiści zyskali wspaniałą możliwość przeniesienia swego dorobku i doświadczeń na platformę internetową, co nadaje nową jakość procesowi tworzenia aplikacji serwerów WWW i programowaniu rozproszonemu w ogólności.

Możliwości WebSnap

WebSnap nie jest technologią całkowicie nową — jest ona wynikiem integracji dwóch istniejących technologii: WebBrokera i `InternetExpress`. Dzięki temu przenoszenie istniejących aplikacji na platformę WebSnap nie jest zbytnio skomplikowane. Technologia WebSnap daje wiele użytecznych możliwości, z których najważniejsze teraz w skrócie omówimy.

Wiele modułów danych

W poprzednich wersjach Delphi aplikacje WebBroker i InternetExpress posiadały pojedynczy moduł danych (`WebModule`). Użycie kilku modułów danych w ramach pojedynczej aplikacji było niemożliwe na etapie projektowania i wymagało dynamicznego tworzenia tychże w kodzie programu. WebSnap eliminuje to ograniczenie, dopuszczając dowolną liczbę modułów danych w ramach jednej aplikacji; każdy z tych modułów reprezentuje pojedynczą stronę WWW. Pozwala to na równoległą pracę kilku projektantów nad różnymi fragmentami aplikacji.

Techniki skryptowe

WebSnap umożliwia wykorzystanie technik skryptowych w tworzonych aplikacjach, co znacznie ułatwia tworzenie i konfigurowanie aplikacji i dokumentów HTML. Komponentem odpowiedzialnym za obsługę skryptów po „serwerowej" stronie aplikacji jest `TAdapter` (i jego komponenty pochodne).

Komponenty-adaptery

Komponenty wywodzące się z klasy `TAdapter` definiują interfejs pomiędzy skryptowymi mechanizmami serwera i klienta. Stanowią one jedyny środek udostępniania klientom skryptów serwera, co chroni te skrypty przed przypadkową lub zamierzoną modyfikacją i jednocześnie pozwala na ukrycie tych funkcji, które nie powinny być wyeksponowane dla aplikacji-klientów.

Definiując komponent pochodny do `TAdapter` zapewniamy sobie zarządzanie specyficzną zawartością, stanowiącą materiał źródłowy na wynikową stronę WWW; materiał ten dostępny jest także na etapie projektowania aplikacji. Taki komponent może zawierać dane oraz wykonywać określone akcje; na przykład komponent `TDataSetAdapter` może wyświetlać rekordy ze zbioru danych oraz wykonywać rozmaite czynności związane z tym zbiorem, jak przewijanie, dodawanie, aktualizacja i usuwanie rekordów.

Wielokierunkowe zarządzanie stronami

WebSnap oferuje kilka sposobów obsługi żądań HTTP. Zawartość strony WWW może być identyfikowana przez nazwę tej strony, przez akcję komponentu `TAdapter` (lub pochodnego) lub przez „zwykłą" akcję (na modłę WebBrokera). Umożliwia to elastyczną realizację wyboru strony do wyświetlenia — wyświetlenie takie może nastąpić wskutek naciśnięcia przycisku `Submit` lub wskutek wybrania łącznika (z menu łączników zbudowanego na podstawie zestawu dostępnych stron).

Komponenty-producenci

Integralną częścią aplikacji tworzonych z udziałem WebBrokera były komponenty–producenci stron, których zadaniem było generowanie zawartości strony wynikowej (w formacie HTML) na podstawie różnorodnych źródeł danych (na przykład zbioru danych lub wybranego rekordu). InternetExpress rozszerza tę koncepcję o komponent `TMidasPageProducer` (w Delphi 6 zastąpiony przez `InetXPageProducer` — por. rozdział 21.), a WebSnap idzie jeszcze dalej, oferując zestaw nowych komponentów wykorzystujących zawartość komponentu `TAdapter` oraz danych w formacie XSL/XML. Najważniejszym z tych komponentów jest `TAdapterPageProducer`, generujący zawartość strony wynikowej na podstawie akcji i pola komponentu `TAdapter` (lub pochodnego).

Zarządzanie sesjami

Aplikacje WebSnap posiadają wbudowany mechanizm zarządzania sesjami, umożliwiający śledzenie akcji danego użytkownika wśród wielu żądań HTTP. Ponieważ HTTP jest protokołem *bezstanowym* (ang. *stateless*), aplikacja musi zapewnić sobie we własnym zakresie środki do utrzymywania informacji o bieżącym stanie sesji każdego użytkownika; do środków tych należą m.in. *cookies*, odwołania do URL lub ukryte pola. WebSnap upraszcza to zadanie, udostępniając komponent `TSessionService`, sprawujący kontrolę nad wszystkimi żądaniami, niezależnie dla każdej sesji; informacja o sesji dostępna jest zarówno dla skryptu obecnego w serwerze, jak i dla kodu samej aplikacji.

Usługi logowania

Względy bezpieczeństwa aplikacji WebSnap wymagają logowania każdego użytkownika; WebSnap automatyzuje proces logowania, dostarczając w tym celu odpowiednich komponentów-adapterów (`TLoginFormAdapter` i `TLoginFormAdapterLoginAction`) realizujących dialog logowania i, w połączeniu z mechanizmami zarządzania sesjami, wykonujących uwierzytelnianie zgodne z wybranym modelem bezpieczeństwa aplikacji. Każdy użytkownik próbujący uzyskać dostęp do nieautoryzowanych stron odsyłany jest automatycznie na stronę logowania.

Zarządzanie informacją o użytkownikach

Integralną częścią zarządzania sesjami jest nadzór nad użytkownikami i ich uprawnieniami w stosunku do aplikacji (lub jej elementów). WebSnap umożliwia przechowywanie różnorodnych informacji o każdym z użytkowników (jak uprawnienia, preferencje itp.) i wykorzystywanie ich na potrzeby aplikacji.

Zarządzanie zasobami rozproszonymi

Strony HTML wykorzystywane, a także generowane, przez aplikację mogą być zależne od wielu różnych plików i zasobów, często generowanych w sposób dynamiczny. Aby ułatwić zarządzanie takimi „rozproszonymi" zasobami, WebSnap udostępnia tzw. usługi lokalizacyjne (ang. *location services*).

Usługi ładowania plików

Zarządzanie ładowaniem plików przez serwer (ang. *file uploading*) wymaga zazwyczaj dość dużej ilości specjalizowanego kodu. WebSnap uwalnia programistę od tego problemu, oferując odpowiedni komponent-adapter sprawujący kontrolę nad ładowanymi plikami i udostępniający związane z tym formularze.

Tworzenie aplikacji WebSnap

Najefektywniejszym i najbardziej pouczającym sposobem poznania możliwości oferowanych przez technologię WebSnap będzie z pewnością samodzielne stworzenie przykładowej aplikacji.

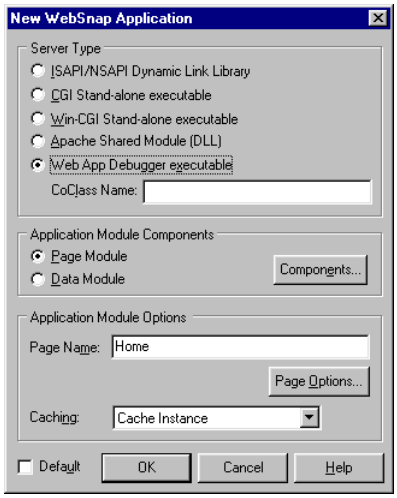
Projektowanie aplikacji

Tworzenie aplikacji WebSnap staje się prostsze, gdy używa się paska narzędziowego IDE o nazwie `Internet` (rys. 23.1); jeżeli nie jest on widoczny, kliknij prawym przyciskiem myszy którykolwiek z widocznych pasków i w wyświetlonym menu kontekstowym zaznacz opcję `Internet`.

Rysunek 23.1.
Pasek narzędziowy Internet

Kliknij następnie przycisk *New WebSnap Application* (ten z ręką trzymającą globus) — spowoduje to wyświetlenie okna przedstawionego na rysunku 23.2.

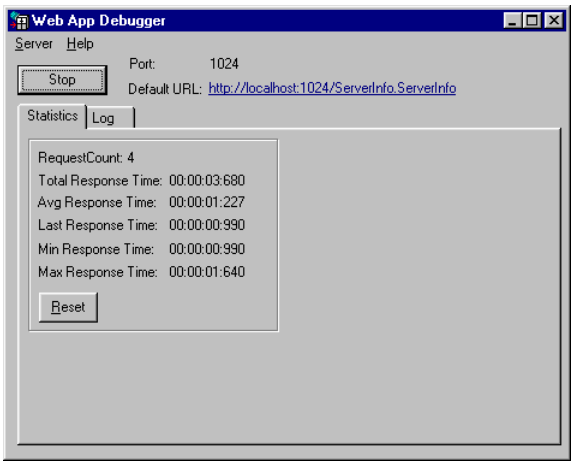
Rysunek 23.2.
Kreator nowej aplikacji WebSnap



Okno to zawiera opcje określające charakter tworzonej aplikacji WebSnap. Opcje w sekcji *Server Type* określają typ serwera, który będzie uruchamiany przez aplikację; ich znaczenie jest następujące:

- *ISAPI/NSAPI Dynamic Link Library* — wybranie tej opcji spowoduje utworzenie projektu aplikacji-rozszerzenia serwera WWW, przeznaczonej do uruchomienia pod kontrolą serwera ISS (lub innego serwera obsługującego aplikacje ISAPI, np. *Personal Web Servera* lub serwera Netscape z adapterem ISAPI). Wynikiem kompilacji projektu jest biblioteka DLL, wykonywana w przestrzeni adresowej serwera WWW.
- *CGI Stand-alone executable* — opcja ta reprezentuje niezależną aplikację konsolową CGI, dokonującą zapisu do standardowych portów wejścia-wyjścia i odczytu z nich, zgodnie ze specyfikacją CGI (*Common Gateway Interface*). Niemal wszystkie serwery WWW obsługują aplikacje CGI.
- *Win-CGI Stand-alone executable* — ta opcja związana jest z mało popularnym (i nie zalecanym) typem aplikacji CGI, przeznaczonej dla Windows i komunikującej się z serwerem za pośrednictwem tekstowych plików .INI.
- *Apache Shared Module (DLL)* — projekt inicjowany w wyniku wybrania tej opcji generuje bibliotekę DLL przeznaczoną dla serwera Apache; szczegółowe informacje na temat tego serwera możesz znaleźć na stronie *http://www.apache.org*.
- *Web App Debugger executable* — jeżeli wybierzesz tę opcję, utworzoną aplikację będzie można uruchomić w środowisku specjalnego debuggera Delphi o nazwie Web App Debugger (rys. 23.3); wynikowy moduł aplikacji będzie *zewnątrzprocesowym* serwerem COM, kontrolowanym i uruchamianym przez ten

debugger. Stwarza to okazję do pełnego wykorzystania Delphi w zakresie śledzenia aplikacji — programista nie jest już skazany na testowanie aplikacji metodą prób i błędów, wymuszające wielokrotne zatrzymywanie i ponowne uruchamianie serwera.



Rysunek 23.3. *Web App Debugger — debugger aplikacji WebSnap*

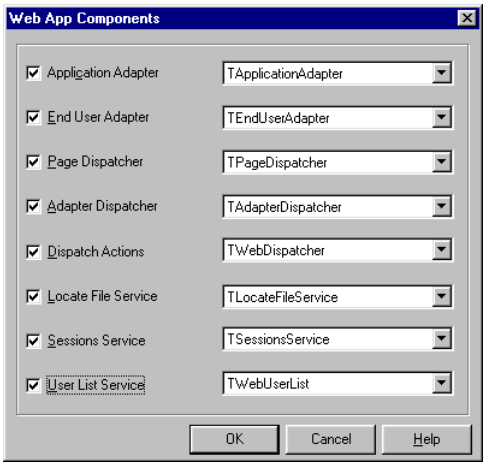
Aby nasza przykładowa aplikacja była jak najbardziej pouczająca, wybierzemy wariant *Web App Debugger executable*.

Web App Debugger uruchamiany jest za pomocą polecenia `Tools/Web App Debugger`. Aby mógł on funkcjonować prawidłowo, konieczne jest zarejestrowanie aplikacji `serverinfo.exe`, znajdującej się w podkatalogu *Bin* zainstalowanego Delphi 6 — należy po prostu uruchomić jednokrotnie tę aplikację np. z poziomu wiersza poleceń. Web App Debugger spełnia rolę serwera dla testowanej aplikacji, symuluje więc niejako rzeczywiste warunki jej pracy. Projekt stworzony w wyniku wybrania opcji *Web App Debugger executable* kreatora z rysunku 23.2 zawiera formularz i sieciowy moduł danych (ang. *Web module*). Formularz pełni rolę środowiska dla serwera COM, który zostaje automatycznie zarejestrowany przy pierwszym uruchomieniu aplikacji i staje się osiągalny za pośrednictwem przeglądarki WWW. Ponieważ testowana aplikacja jest wykonywalnym modułem .EXE, można ją załadować w zwykły sposób do IDE i ustawić punkty przerwań (ang. *breakpoints*) w interesujących miejscach kodu.

Aby uruchomić testowaną aplikację za pośrednictwem przeglądarki WWW, należy uruchomić Web App Debugger i kliknąć hiperłącze zatytułowane *Default URL*; spowoduje to uruchomienie programu udostępniającego listę zarejestrowanych w systemie zewnątrzprocesowych serwerów COM, z której można wybrać aplikację do uruchomienia. Za pomocą opcji *View Details* możemy poznać systemowe szczegóły wskazanej aplikacji, a także usunąć ją z rejestru, jeżeli nie będzie dłużej potrzebna. *Nie należy wyrejestrowywać aplikacji* `serverinfo.exe` — jeżeli dokonamy jej wyrejestrowania przez pomyłkę, trzeba będzie zarejestrować ją ponownie, przez uruchomienie jej z wiersza poleceń.

Kolejna sekcja opcji kreatora (*Application Module Components*) umożliwia określenie typu modułu danych oraz zawartych w nim komponentów. Wybranie opcji *Page Module* spowoduje utworzenie sieciowego modułu danych (*Web module*) reprezentującego stronę HTML; wybranie opcji Data Module spowoduje utworzenie tradycyjnego modułu danych (*data module*) funkcjonującego identycznie z modułem w zwykłej aplikacji klient-serwer. Dla naszej aplikacji wybierzemy sieciowy moduł danych (*Page Module*).

Po kliknięciu przycisku *Components* ujrzymy okno przedstawione na rysunku 23.4.



Rysunek 23.4. *Określenie zestawu komponentów umieszczanych automatycznie w module danych*

Kreator zaoferuje nam do wyboru następujące komponenty:

- *Application Adapter* — zarządza polami i akcjami obiektu skryptowego w serwerze; jego najbardziej interesującą właściwością jest `Title`.
- *End User Adapter* — zarządza informacjami o aktualnie zalogowanym użytkowniku: jego nazwą, przywilejami, identyfikatorem sesji itp. Może również zajmować się procesem logowania i wylogowania użytkownika.
- *Page Dispatcher* — zajmuje się obsługą żądań HTTP związanych ze stroną jako całością (inaczej mówiąc — z nazwą strony). Udostępnia on żądaną stronę na podstawie łącznika HREF lub odwołania z wnętrza akcji.
- *Adapter Dispatcher* — zarządza żądaniami wygenerowanymi przez akcje komponentów-adapterów. Najczęściej są to żądania wysyłki formularzy HTML.
- *Dispatcher Actions* — ta opcja związana jest z komponentem `TWebDispatcher`, dobrze znanym użytkownikom WebBrokera. Zajmuje się on obsługą żądań bazujących na adresach URL i może być użyty do definiowania dodatkowych akcji, na wzór WebBrokera.
- *Locate File Service* — zdarzenia tego komponentu generowane są każdorazowo, gdy sieciowy moduł danych wymaga wprowadzania danych HTML. Umożliwia to „przechwycenie" operacji wczytywania danych i zastąpienie lub uzupełnienie standardowych operacji, czyli w praktyce — wczytanie strumienia HTKL z dowolnego urządzenia. Komponent ten wykorzystywany jest bardzo często do tworzenia standardowych stron na podstawie zawartości innych stron i szablonów.
- *Session Service* — ten komponent zajmuje się zarządzaniem sesjami użytkownika, umożliwiając utrzymywanie informacji o stanie każdej sesji użytkownika pomiędzy kolejnymi żądaniami HTTP. Ma on również możliwość przerwania sesji użytkownika po przekroczeniu przez niego ustalonego okresu bezczynności. Możliwe jest również dodawanie specyficznej informacji związanej z daną sesją — odpowiedzialna za to jest właściwość `Session.Value`, będąca właściwością tablicową, indeksowaną

łańcuchami znaków[1]. Fizycznie, komponent `TSessionService` wykorzystuje *cookies* (na komputerze klienta) do przechowywania informacji o stanie sesji, możliwe jest jednak stworzenie klasy pochodnej wykorzystującej w tym celu inne środki, jak zmienne URL lub ukryte pola.

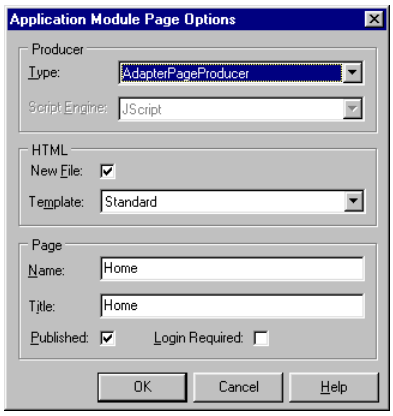
- *User List Service* — komponent ten utrzymuje listę użytkowników uprawnionych do logowania się do aplikacji oraz przechowuje niektóre informacje o tych użytkownikach.

Z każdą z opcji okna z rys. 23.4 związana jest rozwijana lista, oferująca do wyboru dostępne komponenty. Listę tę można poszerzyć o samodzielnie tworzone komponenty — należy je tylko zarejestrować w WebSnap. Możliwe jest na przykład stworzenie komponentu pochodnego do `TSessionService`, wykorzystującego inny niż *cookies* mechanizm do przechowywania informacji o stanie sesji.

By wybrać wszystkie dostępne kategorie komponentów, zaznacz wszystkie pola opcji. Następnie z listy towarzyszącej opcji *End User Adapter* wybierz komponent *TEndUserSessionAdapter*; komponentowi temu zostanie automatycznie przypisany identyfikator sesji użytkownika końcowego. Kliknij przycisk *OK*.

Kolejna opcja związana jest z generowaną stroną HTML i umożliwia określenie jej nazwy — w naszym przykładzie będzie to *Home*. Za pomocą przycisku *Page Options* można określić właściwości tworzonej strony — Delphi udostępnia w tym celu kreator, którego okno jest przedstawione na rysunku 23.5.

Rysunek 23.5.
Okno właściwości docelowej strony HTML



Opcje sekcji `Producer` umożliwiają wybranie odpowiedniego komponentu-producenta strony, stosownie do źródła danych, na podstawie którego generowany będzie strumień HTML stanowiący zawartość strony wynikowej. Wybierz komponent `TPageProducer`. W polu *Script Engine* można wybrać jeden z dwóch oferowanych przez Delphi 6 języków skryptowych: JScript i VBScript; pozostaw domyślną wartość *JScript*.

Opcje z grupy HTML umożliwiają określenie charakteru strony docelowej. Delphi 6 domyślnie oferuje standardową stronę z prostym menu nawigacyjnym (*Standard*) oraz stronę

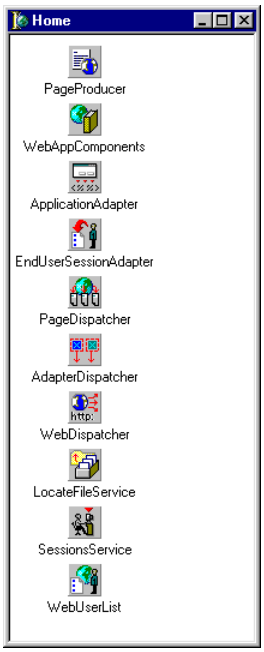
[1] To stwierdzenie jest mocno uproszczone: `Session.Value` jest tak naprawdę *kolekcją*, posiada bowiem licznik elementów `ValueCount` i ich tablicę `Values[]`; dla uproszczenia można jednak przyjąć, iż istotnie mamy do czynienia z tablicą — uproszczenie to nie dotyczy oczywiście kodu źródłowego aplikacji! — *przyp. tłum.*

całkowicie pustą (*Blank*), możliwe jest jednakże zarejestrowanie w WebSnap własnego szablonu strony i użycie go w tym miejscu. Dla naszej aplikacji użyjemy strony standardowej (*Standard*).

Pola *Name* i *Title* w sekcji *Page* powinny zawierać nazwę strony *Home* (nadaną jej w oknie głównym kreatora). Upewnij się ponadto, że zaznaczone jest pole *Published*, a pole *Login Required* pozostaje niezaznaczone.

Po kliknięciu przycisku *OK* powrócisz do okna głównego kreatora; po kliknięciu *jego* przycisku *OK* powinieneś zobaczyć wygenerowany moduł danych, przypominający wyglądem ten z rysunku 23.6.

Rysunek 23.6. *Sieciowy moduł danych aplikacji WebSnap*



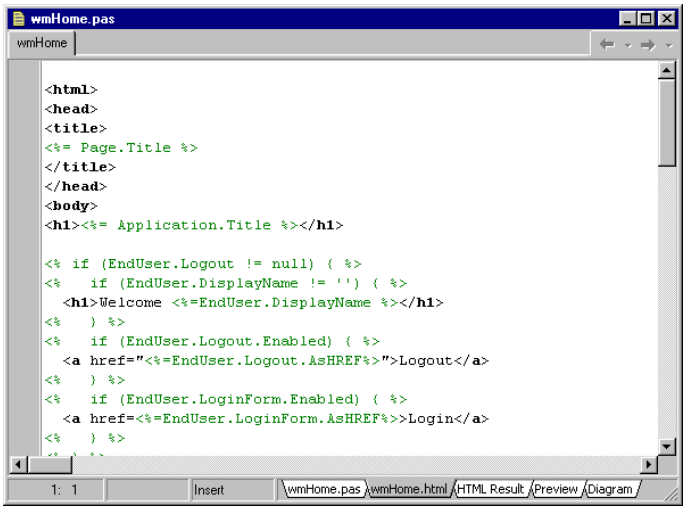
Nie powiedzieliśmy jeszcze ani słowa na temat komponentu `WebAppComponents`. Pełni on rolę *koordynatora* wiążącego ze sobą wszystkie pozostałe komponenty modułu danych, umożliwiając im wzajemne odwoływanie się do swych instancji oraz komunikowanie się ze sobą. Spojrzawszy w okno inspektora obiektów nietrudno zauważyć, iż większość jego właściwości to referencje do pozostałych komponentów.

Na tym etapie projekt nadaje się do zapisania; nadaj modułowi sieciowemu nazwę `wmHome` (zamiast proponowanej `Unit2`), formularzowi głównemu — `ServerForm` (zamiast proponowanej `Unit1`), a projektowi — `DDG6Demo`.

Zawartość okna edytora kodu dla modułu `wmHome` też wygląda cokolwiek niecodziennie (rys. 23.7). Po pierwsze, od razu rzucają się w oczy zakładki u dolnej krawędzi okna. Każdy sieciowy moduł danych — jako że reprezentuje stronę WWW — oprócz kodu „pascalowego" posiada także „skryptowy" odpowiednik w formacie HTML, dostępny za pośrednictwem drugiej (licząc od lewej) zakładki. Ponieważ w oknie edytora z rysunku 23.5 wybrałeś *standardowy* szablon dla strony WWW, wygenerowana strona zawiera podstawowe

menu nawigacyjne uwzględniające wszystkie *opublikowane* strony aplikacji; menu to będzie *automatycznie uzupełniane* w miarę dodawania nowych stron do aplikacji.

Rysunek 23.7. *Skrypt strony HTML związanej z sieciowym modułem danych*



Domyślny kod HTML, wygenerowany dla modułu `wmHome`, przedstawiamy na wydruku 23.1. Oprócz regularnych instrukcji HTML zawiera on także elementy JavaScriptu.

Wydruk 23.1. *Skrypt strony generowanej przez sieciowy moduł danych*

```
<html>
<head>
<title>
<%= Page.Title %>
</title>
</head>
<body>
<h1><%= Application.Title %></h1>

<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
<%   if (EndUser.Logout.Enabled) { %>
  <a href="<%=EndUser.Logout.AsHREF%>">Logout</a>
<%   } %>
<%   if (EndUser.LoginForm.Enabled) { %>
  <a href=<%=EndUser.LoginForm.AsHREF%>>Login</a>
<%   } %>
<% } %>

<h2><%= Page.Title %></h2>

<table cellspacing="0" cellpadding="0">
<td>
<%  e = new Enumerator(Pages)
     s = ''
     c = 0
     for (; !e.atEnd(); e.moveNext())
```

```
      {
        if (e.item().Published)
        {
          if (c>0) s += ' | '
          if (Page.Name != e.item().Name)
            s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
          else
            s += e.item().Title
          c++
        }
      }
      if (c>1) Response.Write(s)
%>
</td>
</table>
<P>
<FONT SIZE="+1" COLOR="Red">Witamy w aplikacji WebSnap!</FONT>
<P>
Aplikacja ta prezentuje nowe możliwości Delphi 6 w zakresie tworzenia aplikacji
interentowych WebSnap.
<P>

</body>
</html>
```

Zwróć uwagę, iż Delphi rozpoznaje składnię pliku HTML, wyróżniając elementy charakterystyczne dla języka skryptowego (szczegóły związane z wyróżnianiem składni przez edytor kodu konfigurowalne są z poziomu opcji edytora — *Tool/Editor Options* — na stronie *Color*). Możliwe jest także użycie własnego edytora stron HTML, na przykład HomeSite, i zintegrowanie go z IDE — integracji takiej dokonuje się za pośrednictwem karty *Internet* opcji środowiskowych (*Tools/Environment Options*). W oknie *Internet File Types* zaznacz pozycję *HTML* i kliknij przycisk *Edit*, a następnie w polu *Edit Action* (okna *Edit Type*) wpisz (lub wybierz z listy) odpowiednią akcję zarejestrowaną w systemie dla plików HTML[2]. W wyniku kliknięcia zakładki *HTML Result* pojawi się kod strony wynikowej w „czystym" HTML-u, natomiast kliknięcie zakładki *Preview* udostępnia ostateczną, *graficzną* postać strony.

Rozbudowa aplikacji

Przystąpimy teraz do rozbudowy projektu wygenerowanego przez kreator. Na początek przejdź do modułu danych *Home*, wybierz komponent *ApplicationAdapter* i zmień jego właściwość *ApplicationTitle* na *Delphi 6 Vademecum profesjonalisty — aplikacja demonstracyjna WebSnap*; gdy klikniesz zakładkę *Preview* (u dołu okna edytora kodu) zauważysz, iż zmiana ta odzwierciedlona została automatycznie w wyglądzie strony. Dzieje się tak za sprawą następującego elementu skryptowego:

```
<h1><%= Application.Title %><h1>
```

powodującego włączenie tytułu aplikacji (`ApplicationTitle`) do kodu HTML.

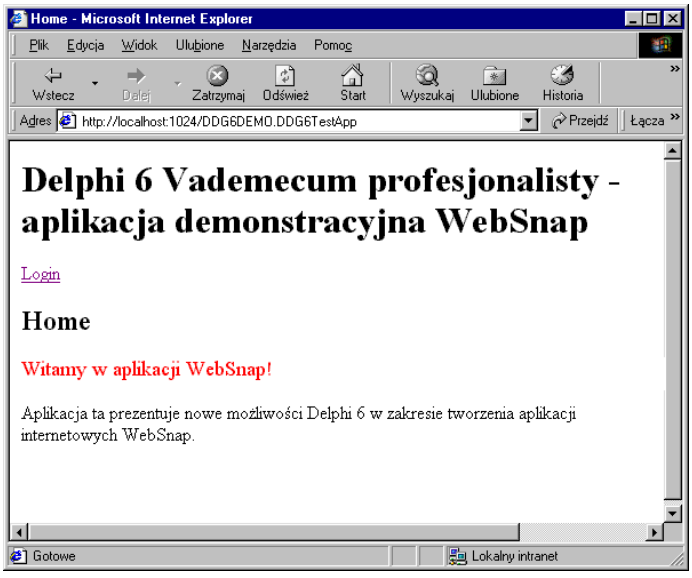
[2] Podobnie jak dla innych typów plików, dla plików *.HTM i *.HTML sterowanie dostępnymi akcjami odbywa się z poziomu okna *Typy plików* opcji folderów; należy wśród zarejestrowanych typów plików odnaleźć kategorię *HTML Document — przyp. tłum.*

Następnie kliknij zakładkę HTML, przejdź w pobliże końca pliku i po znaczniku `</table>` wstaw następującą sekcję kodu:

```
<P>
<FONT SIZE="+1" COLOR="Red">Witamy w aplikacji WebSnap!</FONT>
<P>
Aplikacja ta prezentuje nowe możliwości Delphi 6 w zakresie tworzenia
aplikacji internetowych WebSnap.
<P>
```

Przejdź następnie na zakładkę *Preview*, by zobaczyć nowy tekst w graficznym widoku strony.

Uruchom teraz projekt; ukaże się pusty formularz — to właśnie jest serwer COM aplikacji. Jeżeli nie jest on jeszcze zarejestrowany, uruchomienie to dokona jego rejestracji. Zamknij ten formularz (kończąc w ten sposób wykonywanie aplikacji) i uruchom Web App Debugger (za pomocą odpowiedniego polecenia z menu *Tools*). Kliknij jego hiperłącze *Default URL* (powinno wskazywać aplikację ServerInfo) i z wyświetlonej listy zarejestrowanych serwerów COM wybierz pozycję *DDG6DEMO.DDG6TestApp*, a następnie kliknij przycisk *Go*; w przeglądarce WWW powinna pojawić się wynikowa strona w formacie zbliżonym do prezentowanego na rysunku 23.8.



Rysunek 23.8. *Strona HTML wygenerowana przez aplikację WebSnap*

Menu nawigacyjne

Dodamy teraz do aplikacji następną stronę ilustrującą funkcjonowanie menu nawigacyjnego. Przejdź do paska menu głównego IDE i z paska narzędziowego Internet (por. rysunek 23.1) wybierz drugi przycisk (*New WebSnap Page Module*). Spowoduje to uruchomienie kreatora *New WebSnap Page Module* — pozostaw standardową zawartość wszystkich jego pól z wyjątkiem pola Name, w które powinieneś wpisać nazwę `Simple`. Spowoduje to utworzenie nowego modułu danych, zawierającego pojedynczy komponent `TPageProducer`.

Ustawianie opcji tworzenia i buforowania modułu danych

Kreator *New WebSnap Page Module* zawiera (w sekcji *Module Options*) dwa pola: *Create*, określające zasady tworzenia instancji modułu danych oraz *Caching*, określające, co stanie się z instancją modułu danych, gdy zakończy on obsługę żądania.

Instancje modułu danych mogą być tworzone *zawsze* (ang. *Always*) albo *na żądanie* (ang. *On Demand*), czyli wówczas, gdy pojawi się żądanie obsługiwane przez ten moduł. Tę drugą możliwość stosuje się do stron wyświetlanych bardzo rzadko, moduły odpowiadające stronom często wyświetlanym powinny korzystać z wariantu *Always*.

Gdy moduł zakończy obsługę żądania, jego instancja może być *zwolniona* (opcja *Delete Instance*) lub *przeniesiona do puli* tzw. instancji oczekujących (opcja *Cache Instance*). Gdy pojawi się nowe żądanie dotyczące danego modułu, a we wspomnianej puli znajduje się jego instancja, to ona właśnie zostanie użyta przez system (zamiast tworzenia nowej instancji); należy jedynie mieć na uwadze fakt, iż stan wszystkich pól modułu będzie taki, jak w momencie zakończenia jego pracy w poprzednim „wcieleniu".

Kod HTML tego modułu jest identyczny z tym, który widziałeś przed chwilą (dla pierwszej strony). Zapisz moduł pod nazwą *wmSimple*.

Dodaj teraz jakiś przykładowy tekst przy końcu modułu — na przykład taki:

```
<P>
To jest przykładowy tekst, który ilustruje konsekwencje pojawienia się
dodatkowej strony w aplikacji - strona ta zostanie dodana do menu
nawigacyjnego we wszystkich pozostałych stronach aplikacji.
<P>
```

Skompiluj i uruchom aplikację za pośrednictwem Web App Debuggera (jak poprzednio). W obrazie strony głównej pojawi się teraz menu zawierające tytuły wszystkich innych stron. To menu jest rezultatem wykonania następującego fragmentu skryptu:

```
<%  e = new Enumerator(Pages)
    s = ''
    c = 0
    for (; !e.atEnd(); e.moveNext())
    {
      if (e.item().Published)
      {
        if (c>0) s += ' | '
        if (Page.Name != e.item().Name)
          s += '<a href="' + e.item().HREF + '">' + e.item().Title + '</a>'
        else
          s += e.item().Title
        c++
      }
    }
    if (c>1) Response.Write(s)
%>
```

Fragment ten realizuje iterację po wszystkich stronach zarejestrowanych w obiekcie `Pages`, tworząc menu z ich nazw. Opcja menu zostaje uczyniona łącznikiem, jeżeli strona reprezentowana przez tę opcję nie jest stroną bieżącą (innymi słowy, żadna strona nie zawiera

łącznika do siebie samej). Nic oczywiście nie stoi na przeszkodzie, by to proste w gruncie rzeczy menu przystosować do bardziej specyficznych wymogów konkretnej aplikacji.

Gdy przyjrzysz się uważnie ekranowi komputera w momencie przełączania stron, zobaczysz migoczący w tle formularz główny aplikacji. Jest to spowodowane traktowaniem aplikacji przez Web App Debugger: przy każdym nowym żądaniu uruchamia on aplikację zawierającą serwer COM, przekazuje jej żądanie, odbiera odpowiedź i zamyka aplikację.

Generowaną stronę WWW można uczynić dostępną jedynie dla *zalogowanych* użytkowników — należy w tym celu zaznaczyć opcję *Login Required* w oknie kreatora *New WebSnap Page Module*. Utwórz nową stronę, nadaj jej nazwę *LoggedIn* i zaznacz wspomnianą opcję. Zapisz wygenerowany moduł pod nazwą *wmLoggedIn*. Następnie dodaj fragment tekstu powitalnego dla zalogowanego użytkownika:

```
<P>
<FONT COLOR="Green"><B>Gratulacje! </B></FONT>
<BR>
Logowanie przeprowadzone pomyślnie. Tylko zarejestrowani użytkownicy
mają dostęp do tej strony - wszyscy pozostali zostaną odesłani
na stronę logowania.
<P>
```

Pomiędzy kodem pascalowym stron *Simple* i *LoggedIn* istnieje niewielka różnica w kodzie rejestracyjnym, znajdującym się w sekcji `initialization` modułu:

```
initialization

  if WebRequestHandler <> nil
  then
    WebRequestHandler.AddWebModuleFactory(
          TWebPageModuleFactory.Create(TLoggedIn,
          TWebPageInfo.Create([wpPublished,LoginRequired], '.html'),
          crOnDemand, caCache)           );
```

Kod ten dokonuje rejestracji stron w obiekcie `WebRequestHandler`, sprawującym nadzór nad wszystkimi stronami i udostępniającym ich kod HTML w razie potrzeby. Moduł ten zarządza także instancjami poszczególnych modułów danych. Parametrami konstruktora obiektu `TWebPageInfo` są opcje związane z publikowaniem strony (`wpPublished`) i wymogami logowania (`LoginRequired`); jeżeli któraś z tych opcji nie jest zaznaczona w oknie kreatora, odpowiadający jej atom jest w kodzie rejestracyjnym *wykomentowany*, jak na przykład w przypadku strony *Simple*:

```
initialization
  if WebRequestHandler <> nil
  then
    WebRequestHandler.AddWebModuleFactory(
          TWebPageModuleFactory.Create(TSimple,
          TWebPageInfo.Create([wpPublished {, wpLoginRequired}], '.html'),
          crOnDemand, caCache));
```

Ułatwia to późniejszą zmianę opcji strony *bezpośrednio w kodzie*, bez potrzeby uruchamiania kreatora.

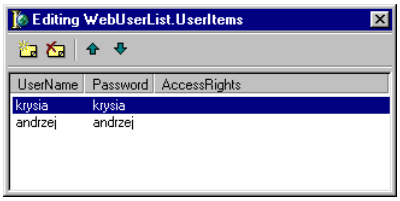
Logowanie

Logowanie użytkownika przeprowadzane będzie za pomocą kolejnej strony. W przeciwieństwie do dotychczas utworzonych — *Simple* i *LoggedIn* — nie powinna ona pojawiać się w menu nawigacyjnym, a więc *nie* powinna być *publikowana*; należy w związku z tym usunąć zaznaczenie opcji *Published* w oknie kreatora *New WebSnap Page Module*. Należy oczywiście pozostawić niezaznaczoną opcję *Login Required* — nie można przecież wymagać wcześniejszego zalogowania w celu… logowania się!

Nazwij nową stronę *Login* i wybierz *TAdapterPageProducer* jako klasę jej komponentu-producenta. Zachowaj ją pod nazwą *wmLogin*. Następnie umieść w jej module danych komponent `TLoginFormAdapter` (ze strony `WebSnap` palety komponentów).

Komponent `TAdapterPageProducer` jest specjalizowanym komponentem generującym kod HTML wynikowej strony WWW na podstawie zawartości komponentu `TAdapter` (lub pochodnego). W naszym przykładzie komponentem źródłowym będzie komponent `TLoginFormAdapter` zawierający „wszystko, co potrzeba" do pobrania nazwy i hasła użytkownika, bez potrzeby pisania jakiegokolwiek kodu.

Aby jednak logowanie w ogóle miało jakiś sens, należy wpierw zdefiniować użytkowników i ich hasła. W tym celu przejdź do modułu *Home* i kliknij dwukrotnie komponent *WebUserList*; w wyświetlonym oknie (rys. 23.9) masz możliwość dodawania i usuwania zdefiniowanych użytkowników, za pomocą klawiszy (odpowiednio *Ins* i *Del*). W wersji znajdującej się na załączonym CD-ROM-ie aplikacja rozpoznaje dwóch użytkowników: `krysia` i `andrzej` — ich hasła są tożsame z nazwami.



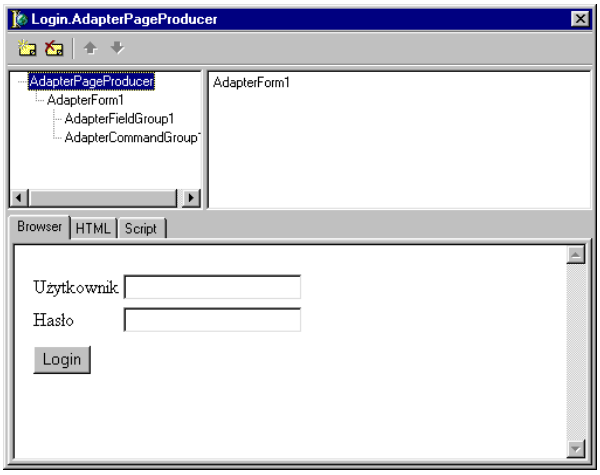
Rysunek 23.9. *Definiowanie użytkowników za pomocą komponentu WebUserList*

Przejdź teraz do komponentu `EndUserSessionAdapter` i ustaw jego właściwość `LoginPage` na `Login` (to nazwa strony logowania). Następnie przejdź do modułu *wmLogin* i kliknij dwukrotnie komponent `TAdapterPageProducer`. Spowoduje to otwarcie edytora *Web Surface Designer*, przedstawionego na rysunku 23.10.

Wybierz pozycję *AdapterPageProducer* w lewym górnym oknie i kliknij przycisk *New Item*. Wybierz *AdapterForm* i kliknij *OK*. Następnie wybierz (w lewym górnym oknie) pozycję *AdapterForm1* i ponownie kliknij przycisk *New Item* i wybierz pozycję *AdapterErrorList*; powtórz tę czynność dwukrotnie, dodając pozycje *AdapterFieldGroup* i *AdapterCommandGroup*. Następnie ustaw właściwość *Adapter* komponentu `AdapterFieldGroup1` na `LoginFormAdapter1`.

Wybierz następnie pozycję *AdapterFieldGroup1* i dodaj do niej dwie pozycje podrzędne typu *AdapterDisplayField*; ustaw ich właściwości *FieldName* na (odpowiednio) *UserName* i *Password*, zaś tytuły (właściwość `Caption`) na *Użytkownik* i *Hasło*.

Rysunek 23.10.
Web Surface Designer



Wybierz pozycję *AdapterCommandGroup* i ustaw jej właściwość *DisplayComponent* na *AdapterFieldGroup1*. Widok w oknie edytora powinien być podobny do tego z rysunku 23.10. Gdy otworzysz moduł `wmLogin` w oknie edytora kodu (na zakładce *Preview*), przekonasz się, iż na stronie pojawił się formularz logowania.

Uruchom teraz aplikację i pozostaw ją uruchomioną — ponieważ wykorzystujemy informację o sesjach użytkowników, aplikacja powinna pozostać w pamięci, by żądania poszczególnych użytkowników mogły być rejestrowane.

Za pomocą Web App Debuggera wywołaj aplikację i przejdź do strony *Simple*, wymagającej, jak wiesz, logowania. Zostaniesz skierowany na stronę logowania; wpisz poprawną nazwę i hasło użytkownika, kliknij przycisk *Login* — w rezultacie powinieneś ujrzeć żądaną stronę *Simple*. To samo będzie się dziać z każdą stroną wymagającą logowania — zauważ, iż osiągnęliśmy to wszystko bez napisania chociażby jednej linijki kodu pascalowego!

Możesz także sprawdzić tzw. idiotoodporność aplikacji przez podanie nazwy nieistniejącego użytkownika lub niepoprawnego hasła; obsługą błędów logowania zajmie się komponent `AdapterErrorList`, kolekcjonujący informację o występujących błędach i wyświetlający związane z nimi komunikaty.

Zwróć uwagę, iż aplikacja „pamięta" nazwę zalogowanego użytkownika i wyświetla jego nazwę w powitalnym zwrocie na każdej wyświetlanej stronie. Jest to wynikiem następującego fragmentu skryptu modułu danych:

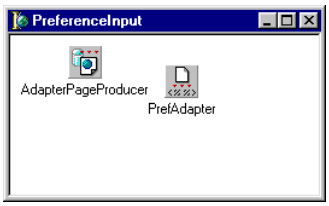
```
<% if (EndUser.Logout != null) { %>
<%   if (EndUser.DisplayName != '') { %>
  <h1>Welcome <%=EndUser.DisplayName %></h1>
<%   } %>
```

Zarządzanie informacją o preferencjach użytkowników

Kolejną funkcją naszej aplikacji będzie przechowywanie rozmaitych informacji na temat preferencji użytkowników. To dość powszechna funkcja dzisiejszych aplikacji biznesowych — sprzedawcy chcą o swych klientach wiedzieć jak najwięcej: od numeru karty

kredytowej, poprzez numer obuwia, do ulubionego koloru szczoteczki do zębów. WebSnap wychodzi naprzeciw potrzebom programistów tworzących aplikacje spełniające takie funkcje — tym razem konieczne będzie jednak napisanie kilku linijek kodu.

Stwórz kolejną stronę i nazwij ją *PreferenceInput*; zakwalifikuj ją jako wymagającą logowania. Zapisz ją pod nazwą *wmPreferenceInput*. Dodaj do nowego modułu danych komponent `TAdapter` i zmień jego właściwość *Name* na *PrefAdapter* (rys. 23.11).



Rysunek 23.11. *Moduł danych strony związanej z preferencjami użytkowników*

Kliknij następnie dwukrotnie komponent `PrefAdapter` i za pomocą wyświetlonego edytora dodaj do niego jedno pole `AdapterBooleanField` i dwa pola `AdapterField`. Nazwij dodane pola (odpowiednio) *LikesChocolate*, *FavoriteMovie* i *PasswordHint*, opatrując jednocześnie tytułami (*DisplayLabel*) *Czy lubisz czekoladę?*, *Ulubiony film* i *Ulubione powiedzonko*.

Komponent `PrefAdapter` będzie przechowywał zawartość wspomnianych pól, które jednocześnie dostępne będą dla innych stron. Komponenty klasy `TAdapter` są komponentami skryptowymi, zdolnymi do magazynowania i udostępniania informacji, lecz wymaga to stworzenia niezbędnego kodu pascalowego. Trzy pola, które dodaliśmy do komponentu–adaptera udostępniają swą zawartość w ramach zdarzenia `OnGetValue`, generowanego za każdym razem, gdy zawartość pola niezbędna jest przy przetwarzaniu skryptu. Ponieważ informacja stanowiąca zawartość pól nie może zaginąć pomiędzy kolejnymi żądaniami, jest przechowywana we właściwości `Session.Value` — jest to właściwość tablicowa, indeksowana łańcuchami znaków; nadaje się więc idealnie do przechowywania informacji w postaci „klucz-wartość”.

Komponenty-adaptery umożliwiają także wykonywanie *akcji* na swoich danych, co najczęściej bywa wykorzystywane do wysyłania formularzy HTML. Wybierz komponent `PrefAdapter`, przejdź do inspektora obiektów i kliknij dwukrotnie właściwość *Actions*. Dodaj pojedynczą akcję i nazwij ją *SubmitAction* oraz opatrz tytułem *Wyślij informację*. Przejdź następnie do karty *Events* inspektora obiektów i dodaj procedurę obsługi zdarzenia `OnExecute` dla tej akcji, zgodnie z wydrukiem 23.2.

Wydruk 23.2. *Obsługa zdarzenia OnExecute akcji SubmitAction*

```
procedure TPreferenceInput.SubmitActionExecute(Sender: TObject;
  Params: TStrings);
var
  Value: IActionFieldValue;
begin

  Value := FavoriteMovieField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sFavoriteMovie] := Value.Values[0];
  end;
```

```
  Value := PasswordHintField.ActionValue;
  if Value.ValueCount > 0 then
  begin
    Session.Values[sPasswordHint] := Value.Values[0];
  end;

  Value := LikesChocolateField.ActionValue;
  if Value <> nil then
  begin
    if Value.ValueCount > 0 then
    begin
      Session.Values[sLikesChocolate] := Value.Values[0];
    end;
  end else
  begin
    Session.Values[sLikesChocolate] := 'false';
  end;;
end;
```

Powyższa procedura pobiera wartości pól formularza HTML i przypisuje je odpowiednim elementom tablicy `Sesssion.Value`; dzieje się to w odpowiedzi na kliknięcie przycisku *Submit* przez użytkownika. Oczywiście wartości te muszą być dostępne na każde żądanie komponentu skryptowego, stąd konieczność „oprogramowania" zdarzeń `OnGetValue` dla każdego pola komponentu `PrefAdapter` (patrz wydruk 23.3).

Wydruk 23.3. *Udostępnianie zawartości pól komponentu-adaptera*

```
…
  const
    sFavoriteMovie = 'FavoriteMovie';
    sPasswordHint = 'PasswordHint';
    sLikesChocolate = 'LikesChocolate';
    sIniFileName = 'DDG6Demo.ini';

…
…

procedure TPreferenceInput.LikesChocolateFieldGetValue(Sender: TObject;
  var Value: Boolean);
var
  S: string;
begin
  S := Session.Values[sLikesChocolate];
  Value := S = 'true';
end;

procedure TPreferenceInput.FavoriteMovieFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sFavoriteMovie];
end;

procedure TPreferenceInput.PasswordHintFieldGetValue(Sender: TObject;
  var Value: Variant);
begin
  Value := Session.Values[sPasswordHint];
end;
```

Prócz możliwości przechowywania i udostępniania preferencji użytkowników konieczne jest oczywiście ich *wprowadzanie* za pomocą odpowiedniego formularza. Wykorzystamy w tym celu komponent `TAdapterPageProducer`, podobnie jak uczyniliśmy to na stronie *Login*. Dodaj więc rzeczony komponent do modułu danych i kliknij go dwukrotnie w celu uruchomienia edytora Web Surface Designer. Dodaj do komponentu pozycję `AdapterForm`, a do niej — pozycje `AdapterFieldGroup` i `AdapterCommandGroup`. Ustaw właściwość `AdapterFieldGroup1.Adapter` na `PrefAdapter` oraz właściwość `AdapterCommandGroup1.DisplayComponent` na `AdapterFieldGroup1`.

Kliknij prawym przyciskiem myszy pozycję `AdapterFieldGroup1` i z jej menu kontekstowego wybierz opcję *Add All Fields*. Za pomocą inspektora obiektów ustaw odpowiednio właściwość `FieldName` każdego z pól (*LikesChocolateField*, *FavoriteMovieField* i *PasswordHintField*) oraz tytuły (*Czy lubisz czekoladę?*, *Ulubiony film* i *Ulubione powiedzonko*).

Wybierz pozycję `AdapterCommandGroup` i za pomocą kliknięcia prawym przyciskiem myszy wybierz z menu kontekstowego opcję *Add All Commands*. Ustaw nazwę (*ActionName*) jedynej akcji na `SubmitAction`, a następnie ustaw właściwość `AdapterActionButton1.PageName` na `PreferencesPage`.

Po uruchomieniu aplikacji i zalogowaniu się możesz obejrzeć wygląd nowego formularza; nie klikaj jednak przycisku *Wyślij*, ponieważ aplikacja nie posiada jeszcze możliwości *wyświetlania* przechowywanych danych.

W związku z tym stwórz kolejną stronę, nazwij ją *PreferencesPage* i zachowaj pod nazwą *wmPreferences*. Przejdź do skryptu strony (*wmPreferences.html*) i dodaj przy końcu następujący fragment tekstu:

```
<P>
<B>Ulubiony film:</B>
  <%= Modules.PreferenceInput.PrefAdapter.FavoriteMovieField.Value %>

<BR>

<B>Ulubione powiedzonko: </B>
  <%= Modules.PreferenceInput.PrefAdapter.PasswordHintField.Value %>

<BR>
<% s = ''
  if (Modules.PreferenceInput.PrefAdapter.LikesChocolateField.Value)
     s = 'Lubisz czekoladę'
   else
     s = 'Nie lubisz czekolady'
   s = '<B>' + s + '</B>';
   Response.Write(s);
%>
```

Po skompilowaniu i uruchomieniu aplikacji preferencje zalogowanego użytkownika będą mogły być wprowadzane i wyświetlane. Będą one jednocześnie dostępne dla skryptu w kodzie dowolnej strony aplikacji — za pośrednictwem komponentu-adaptera pobierającego je z informacji o sesji.

Każda ze stron aplikacji posiada swój odrębny plik HTML, który może być łatwo edytowany bez potrzeby ładowania aplikacji do IDE Delphi. Oznacza to możliwość łatwej modyfikacji aplikacji bez konieczności jej rekompilowania czy uruchamiania serwera. W dalszym ciągu rozdziału poznasz inne możliwości przechowywania stron HTML i uzyskiwania ich zawartości.

Przechowywanie informacji pomiędzy sesjami użytkowników

W związku z przechowywaniem danych użytkownika pozostaje do rozwiązania jeszcze jeden problem — są one przechowywane tylko w czasie sesji i zostają utracone w momencie wylogowania się użytkownika. Rozwiązaniem tego problemu jest zapis danych w pliku w momencie wylogowywania się użytkownika i ich odczyt w momencie logowania. W naszej aplikacji czynności te wykonywane są w ramach zdarzeń `OnLogin` komponentu `LoginFormAdapter` (logowanie) i `OnEndSession` komponentu `SessionService` (wylogowanie). Szczegóły obsługi tych zdarzeń przedstawiamy na wydruku 23.4.

Wydruk 23.4. *Zarządzanie preferencjami użytkownika w momencie logowania i wylogowania*

```
…

procedure TLogin.LoginFormAdapter1Login(Sender: TObject; UserID: Variant);
var
  IniFile: TIniFile;
  TempName: string;
begin
  // przechwyć dane sesji

  //WebContext.EndUser.DisplayName;
  TempName := Home.WebUserList.UserItems.FindUserID(UserId).UserName;


  Home.CurrentUserName := TempName;

  Lock.BeginRead;
  try
    IniFile := TIniFile.Create(IniFileName);
    try
      Session.Values[sFavoriteMovie] := IniFile.ReadString(TempName, sFavoriteMovie,
'');
      Session.Values[sPasswordHint] := IniFile.ReadString(TempName, sPasswordHint,
'');
      Session.Values[sLikesChocolate] := IniFile.ReadString(TempName, sLikesChocolate,
'false');
    finally
      IniFile.Free;
    end;
  finally
    Lock.EndRead;
  end;
end;
…

procedure THome.SessionsServiceEndSession(ASender: TObject;
  ASession: TAbstractWebSession; AReason: TEndSessionReason);
var
  IniFile: TIniFile;
```

```
begin
  //zapisz preferencje użytkownika
  Lock.BeginWrite;
  if FCurrentUserName <> '' then
  begin
    try
      IniFile := TIniFile.Create(IniFileName);
      try
        IniFile.WriteString(FCurrentUserName, sFavoriteMovie,
ASession.Values[sFavoriteMovie]);
        IniFile.WriteString(FCurrentUserName, sPassWordHint,
ASession.Values[sPasswordHint]);
        IniFile.WriteString(FCurrentUserName, sLikesChocolate,
ASession.Values[sLikesChocolate]);
      finally
        IniFile.Free;
      end;
    finally
      Lock.EndWrite
    end;
  end;
end;
```

Dla prostoty przykładu preferencje użytkowników przechowywane są w pliku .INI, nic jednak nie stoi na przeszkodzie, by przechowywać je w bazie danych lub innym trwałym medium.

Użyta w przykładzie zmienna `Lock` jest globalnym obiektem typu `TMultiReadExclusiveWriteSynchronizer`, a jego instancja tworzona jest w sekcji `initialization` modułu `Home`. Zapewnia on bezpieczeństwo wątkowe pliku w sytuacji, gdy kilka różnych sesji chciałoby równocześnie odczytywać zawartość pliku lub zapisywać ją.

Musisz więc dodać następującą deklarację do publicznej części modułu *wmHome.pas*:

```
var
  Lock: TMultiReadExclusiveWriteSynchronizer;
```

Musisz także utworzyć egzemplarz obiektu `Lock` w sekcji `initialization`

```
Lock := TMultiReadExclusiveWriteSynchronizer.Create;
```

i zwolnić go w sekcji `finalization`:

```
Lock.Free;
```

Potrzebna jest jeszcze funkcja zwracająca *nazwę* pliku:

```
const
  sIniFileName = 'DDG6Demo.ini';
…

function IniFileName: string;
begin
  Result := ExtractFilePath(GetModuleName(HInstance)) + sIniFileName;
end;
```

Jak łatwo zauważyć, plik .INI tworzony jest w katalogu, w którym znajduje się moduł wykonywalny aplikacji.

Przetwarzanie obrazów i obsługa grafiki

Nie sposób wyobrazić sobie nowoczesnej aplikacji bez możliwości wyświetlania grafiki, poprawiającej wygląd i funkcjonalność. Nie mogło więc zabraknąć w technologii WebSnap środków do przetwarzania obrazów — umożliwia ona obróbkę grafiki w różnej postaci: plików, zasobów, strumieni danych itp. Generalnie — jeżeli obraz może być zapisany do strumienia, może być przetwarzany przez WebSnap.

Dodaj do aplikacji kolejną stronę — *Images* — i zapisz ją pod nazwą *wmImages*. Jako jej komponentu-producenta użyj `TAdapterPageProducer`, opublikuj ją i zaznacz opcję wymuszającą logowanie. Dodaj do jej modułu danych komponent `TAdapter` i nazwij go `ImageAdapter`. Kliknij go dwukrotnie (by uruchomić edytor Web Surface Designer) i dodaj do niego dwa pola typu `TAdapterImageField` — każde z nich będzie wyświetlać grafikę w różny sposób.

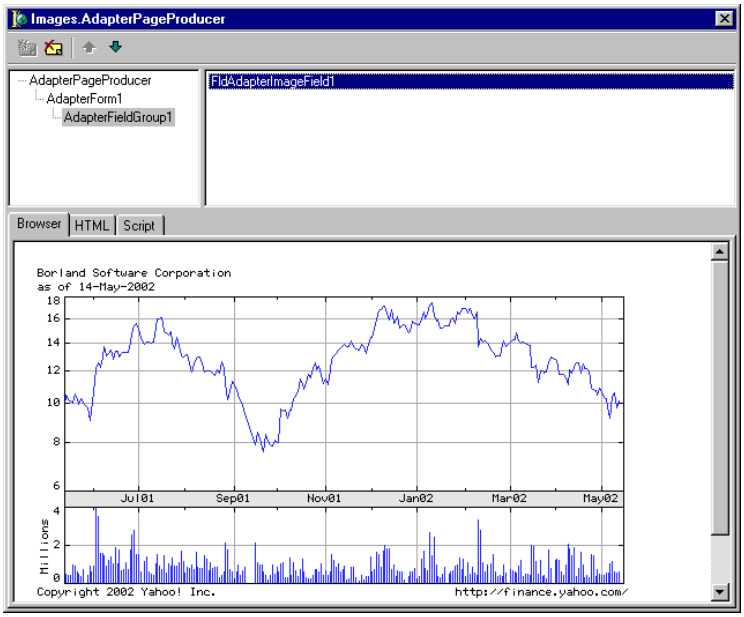
Jednym ze sposobów odwołania się do obrazka jest jego URL. Kliknij pierwsze z pól `AdapterImageField` i ustaw jego właściwość `HREF` na dowolny, kompletny URL; w naszym przykładzie zerkamy na wykres rocznych wahań cen akcji Borlanda, dostępny pod adresem *http://chart.yahoo.com/c/1y/b/borl.gif*.

Kliknij dwukrotnie komponent `TAdapterPageProducer`, dodaj do niego pozycję `AdapterForm`, a do niej pozycję `AdapterFieldGroup`. Ustaw właściwość `AdapterFieldGroup1.Adapter` na `ImageAdapter`. Kliknij prawym przyciskiem myszy pozycję `AdapterFieldGroup1` i wybierz z jej menu kontekstowego opcję *Select All Fields*. Ustaw na `True` właściwość `AdapterImageField.ReadOnly` (jeżeli właściwość ta ma wartość `True`, na stronie wyświetlany będzie *obrazek*; w przeciwnym razie zamiast obrazka ujrzymy *kontrolkę edycyjną* do wpisania nazwy pliku i towarzyszący jej przycisk *Przeglądaj*). Gdy ujrzymy obrazek po raz pierwszy, będzie mu towarzyszył zbędny, szpecący tytuł; aby go usunąć, musimy ustawić właściwość `DisplayLabel` pola `AdapterImageField` na *pojedynczą spację* — usunięcie wszystkich znaków spowoduje bowiem automatyczne przyjęcie tytułu domyślnego. Po wykonaniu opisanych czynności okno edytora Web Surface Designer powinno wyglądać podobnie do tego na rysunku 23.12.

Aby na etapie projektowania zapewnić wyświetlenie obrazków identyfikowanych przez *ścieżki względne*, należy dodać do ścieżki `Search Path` w opcjach projektu katalog, w którym rezyduje `Web App Debugger`.

Oprócz obrazków identyfikowanych przez URL, WebSnap potrafi także obsługiwać grafikę przechowywaną w strumieniu (*stream*). Wybierz drugie z pól `AdapterImageField` (komponentu `ImageAdapter`) i za pomocą inspektora obiektów wygeneruj szkielet procedury obsługi zdarzenia `OnGetImage`. Umieść jakiś obrazek w tym samym katalogu, w którym znajduje się projekt aplikacji (wersja na załączonym CD-ROM-ie wykorzystuje obrazek *athena.jpg*) i uzupełnij następująco wspomnianą procedurę zdarzeniową:

```
procedure TImages.AdapterImageField2GetImage(Sender: TObject;
  Params: TStrings; var MimeType: String; var Image: TStream;
  var Owned: Boolean);
begin
  MimeType := 'image\jpg';
  Image := TFileStream.Create('athena.jpg', fmOpenRead);
end;
```



Rysunek 23.12. *Wyświetlanie obrazka identyfikowanego przez URL*

Wszystko wygląda tu niesłychanie prosto: binarny strumień składający się na zawartość obrazka przypisywany jest do parametru `Image`. Ponieważ aplikacja nie posiada żadnych informacji na temat *typu* zawartości tego strumienia, należy jej ten typ wskazać — do tego służy parametr `MimeType`. Strumień plikowy `TFileStream` nie jest co prawda niczym nadzwyczajnym, jeśli chodzi o przechowywanie grafiki, jednak WebSnap potrafi z równą łatwością „dobierać się" do obrazków przechowywanych w *dowolnym* strumieniu, jak np. `TBlobStream`, czy nawet strumień pamięciowy o zawartości tworzonej *ad hoc*.

Po uruchomieniu aplikacji powinieneś na stronie `Images` zobaczyć, poniżej prezentowanego już wykresu, portret (znanej z Delphi 2) bogini Ateny.

Wyświetlanie zawartości bazy danych

Wśród bogatych możliwości oferowanych przez WebSnap nie mogło oczywiście zabraknąć obsługi baz danych. Zawartość *dowolnego* zbioru danych może być wyświetlana na stronie WWW zarówno w formie tabelarycznej, jak i w postaci oddzielnych rekordów. Zawartość każdego rekordu może być łatwo modyfikowana, można także usuwać rekordy i dodawać nowe.

Wyświetlanie kolejnych rekordów zbioru danych (po jednym na stronie) jest bardzo łatwe. Dodaj do aplikacji nowy moduł danych — kliknij *trzeci* przycisk na pasku *Internet* i zaakceptuj domyślne wartości ustawień. Dodaj do utworzonego modułu danych komponent `TDataSetAdapter` (ze strony `WebSnap` palety komponentów) i komponent `TTable` (ze strony BDE). Skojarz komponent `TTable` z tabelą `BioLife` bazy danych `DBDEMOS`, a następnie przypisz sam komponent do właściwości `DataSet` komponentu `TDataSetAdapter`. Otwórz tabelę bazy danych przez ustawienie na `True` właściwość `Active` komponentu `TTable`. Na koniec nadaj nowemu modułowi nazwę `BioLifeData` i zapisz go pod nazwą `wdmBioLife`.

W naszym przykładzie wykorzystaliśmy pojedynczą tabelę bazy danych Paradox, jednak komponent `TDataSetAdapter` może współpracować z dowolnym komponentem wywodzącym się z klasy `TDataSet`. Ponadto trzeba przyznać, iż wykorzystanie tabeli `TTable` w aplikacji `WebSnap` *bez jawnego obsługiwania sesji* nie jest dobrym posunięciem, zważywszy na warunki pracy wielowątkowej (por. rozdział 5.) — zdecydowaliśmy się na takie posunięcie, by nie komplikować przykładu.

Dla odmiany — otwórz okno Object TreeView (za pomocą kombinacji klawiszy *Shift+Alt+F11*). Kliknij prawym przyciskiem myszy pozycję *Actions* komponentu `DataSetProducer1` i z menu kontekstowego wybierz opcję *Add All Actions*; w podobny sposób wybierz opcję *Add All Fields* z menu kontekstowego pozycji *Fields*. Przejdź następnie do komponentu `Table1` i za pomocą edytora pól dodaj do modułu danych wszystkie jego pola (*Add All Fields*).

Ponieważ WebSnap operuje na zbiorach danych w trybie *bezstanowym*, konieczne jest istnienie klucza głównego dla zbioru danych, by nawigacja wśród jego rekordów i modyfikacja jego danych w ogóle były możliwe. WebSnap oferuje łatwe rozwiązanie tego problemu: w oknie *Object TreeView* wybierz pozycję *Fields* (podporządkowaną komponentowi `Table1`) i dla pola `Species_No` ustaw opcję `pfInKey` we właściwości `ProviderFlags`.

Dodaj do aplikacji nową stronę, zakwalifikuj ją jako wymagającą logowania, nadaj jej nazwę *BioLife* i zapisz pod nazwą *wmBiolife*. Ponieważ jej moduł źródłowy będzie się odwoływać do modułu `wdmBiolife`, umieść nazwę tego ostatniego na liście *uses*.

Przejdź do okna *Object TreeView* i z menu kontekstowego pozycji *WebPageItems* wybierz polecenie *New Component*, po czym dodaj komponent `AdapterForm`. W analogiczny sposób dodaj do pozycji `AdapterForm1` komponenty `AdapterErrorList` i `AdapterGrid`; ustaw właściwość `Adapter` tych komponentów na `DataSetAdapter1`. Za pomocą menu kontekstowego komponentu `AdapterGrid1` dodaj wszystkie jego kolumny (*Add All Columns*).

Przejdź do modułu `BioLifeData` i z menu kontekstowego pozycji *Actions* (podporządkowanej komponentowi `DatasetAdapter1`) wybierz polecenie *Add All Actions*; w ten sam sposób wybierz polecenie *Add All Fields* dla pozycji *Fields*.

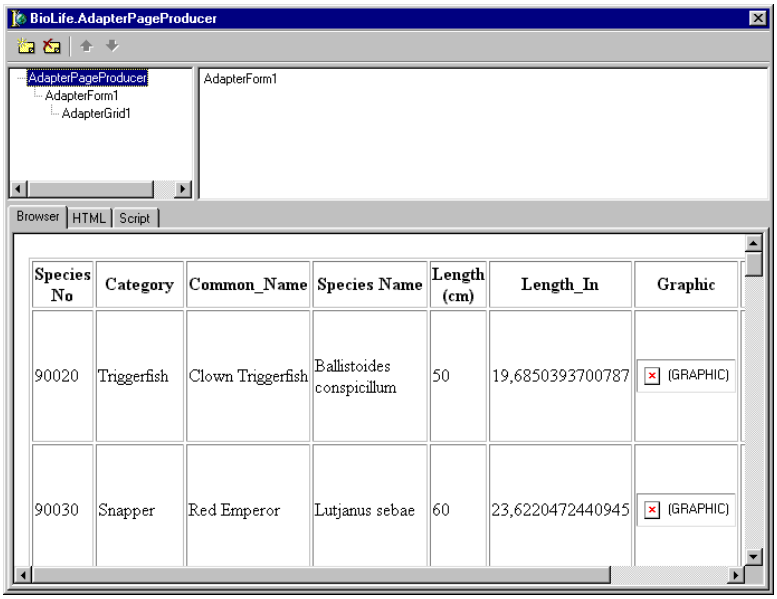
Powróć do modułu `BioLife` i kliknij dwukrotnie komponent `AdapterPageProducer`; powinieneś zobaczyć okno edytora Web Surface Designer z uwidocznioną zawartością rekordów tabeli (rys. 23.13); jeżeli rekordy się nie ukażą, prawdopodobnie zbiór nie został otwarty — zmień wówczas na `True` właściwość `Active` komponentu `Table1`.

Ponieważ pole `Notes` „spycha” na prawo następne pola, przejdź do komponentu `AdapterGrid1` (w oknie *Object TreeView*) i usuń pozycję `ColNotes`.

Web Surface Designer nie wyświetla zawartości pól graficznych, będą one jednak widoczne na stronie wynikowej — o czym można się przekonać, gdy skompiluje się i uruchomi ponownie aplikację. Zauważ, że znowu wszystko odbyło się bez napisania jakiegokolwiek kodu pascalowego.

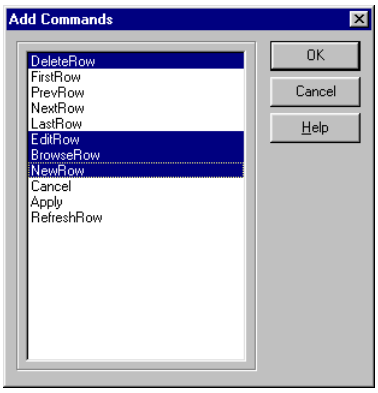
Oczywiście obsługa baz danych nie byłaby w pełni użyteczna, gdyby sprowadzała się jedynie do *przeglądania* danych; WebSnap umożliwia więc dodawanie, przeglądanie, edytowanie i usuwanie poszczególnych rekordów tabeli.

Rysunek 23.13. *Zawartość tabeli BioLife wyświetlana w oknie Web Surface Designera uruchomionego przez komponent TDataSetAdapter*



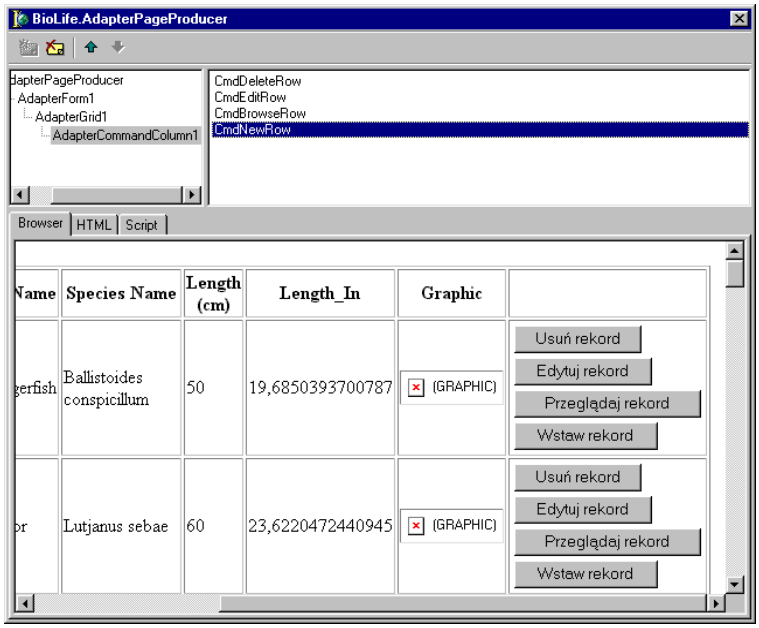
W oknie Web Surface Designera przejdź do pozycji *AdapterGrid* i dodaj pozycję podrzędną *AdapterCommandColumn*; za pomocą polecenia `Add Commands` jej menu kontekstowego dodaj cztery pozycje `CmdDeleteRow`, `CmdEditRow`, `CmdBrowseRow` i `CmdNewRow`, posługując się oknem przedstawionym na rysunku 23.14 (zaznaczając drugą i kolejne opcje, trzymaj naciśnięty klawisz *Ctrl*).

Rysunek 23.14. *Dodawanie poleceń do komponentu AdapterCommandColumn*



Po zamknięciu okna *Add Commands* przejdź do inspektora obiektów i zmień standardowe tytuły (`Caption`) utworzonych przycisków na (odpowiednio) *Usuń rekord*, *Edytuj rekord*, *Przeglądaj rekord* i *Wstaw rekord*. Wymuś pionowe ułożenie przycisków przez ustawienie na `1` właściwości `DisplayColumns` komponentu `AdapterCommandColumn1`[3]. Gdy przewiniesz w poziomie zawartość dolnego okna Web Surface Designera, zobaczysz ostatnią kolumnę tabeli zawierającą dodane przed chwilą przyciski (rys. 23.15).

[3] Właściwość ta określa, w ilu kolumnach mają być wyświetlane wygenerowane obiekty — *przyp. tłum.*



Rysunek 23.15. *Przyciski poleceń TAdapterActionButton w oknie Web Surface Designera*

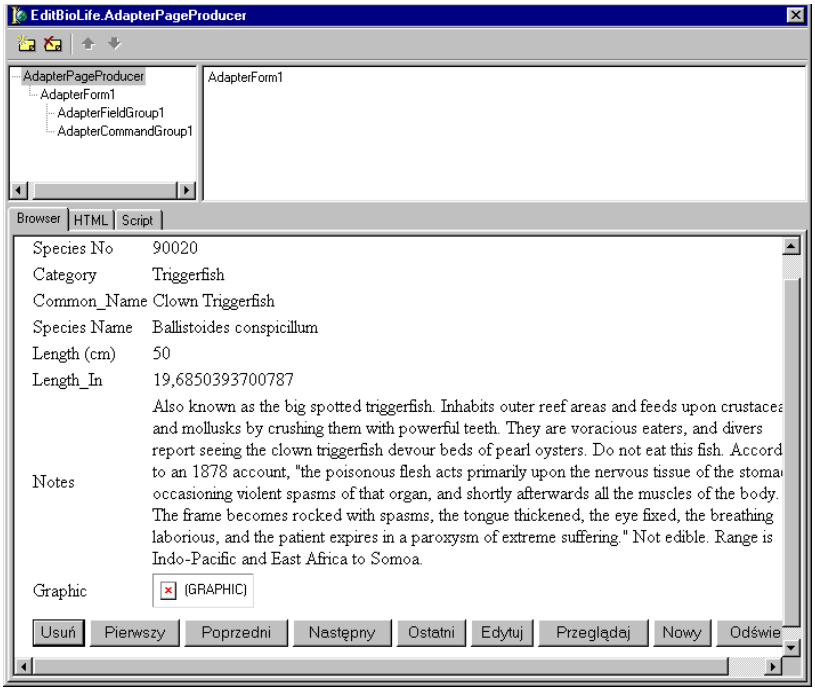
Same przyciski jednak nie wystarczą — aby możliwe było przeglądanie, edycja i dodanie rekordu, potrzebna jest dodatkowa strona wyświetlająca „formatkę" rekordu, czyli czytelny układ jego zawartości. Dodaj więc do projektu kolejną stronę o nazwie *EditBioLife* i zapisz ją pod nazwą *wdmBioLife*. Umieść na niej komponent `AdapterPageProducer`, dodaj do niego podporządkowaną pozycję `AdapterForm`, do tego ostatniego dodaj komponenty `AdapterErrorList`, `AdapterFieldGroup` i `AdapterCommandGroup`. Za pomocą menu kontekstowych dodaj wszystkie pola (*Add All Fields*) komponentu `AdapterFieldGroup` i wszystkie polecenia (*Add All Commands*) komponentu `AdapterCommandGroup`. W oknie Web Surface Designera powinna pojawić się zawartość konkretnego rekordu, jak na rysunku 23.16.

Musisz jeszcze dokonać powiązania przycisków widocznych w ostatniej kolumnie tabeli (na stronie `BioLife`) ze stroną *EditBioLife*. Zaznacz przyciski *CmdEditRow*, *CmdBrowseRow* i *CmdNewRow* (trzymając naciśnięty klawisz *Ctrl*) i za pomocą inspektora obiektów przypisz ich właściwości `PageName` wartość `EditBiolife`.

Kiedy zażądasz przeglądania rekordu, wyświetlona zostanie strona *EditBioLife* prezentująca dane rekordu w „płaskim" układzie tekstowym; kiedy zażądasz edycji lub wstawienia rekordu, wyświetlona zostanie ta sama strona, jednak dane rekordu widoczne będą w edytowalnych polach; pole związane z grafiką pozwala nawet na zmianę obrazka w rekordzie przez wskazanie odpowiedniego pliku.

Ponownie — wszystko odbyło się bez pisania jakiegokolwiek kodu.

Pole edycyjne związane z polem `Notes` rekordu jest zwykłą kontrolką edycyjną, w związku z czym edytowany tekst jest w dużej części niewidoczny. Można zmienić ten stan rzeczy przez odpowiednie ustawienie właściwości `TextAreaWrap`, `DisplayRows` i `DisplayWidth` komponentu `FldNotes`.



Rysunek 23.16. *Zawartość rekordu tabeli w oknie Web Surface Designera*

Konwertowanie aplikacji do postaci ISAPI DLL

Nasza aplikacja została zaprojektowana specjalnie dla celów jej śledzenia przez Web App Debugger; nie da się jej więc zastosować jako rozszerzenie ISAPI serwera WWW, bowiem rozszerzenia takie mają postać bibliotek DLL. Na szczęście problem ten stosunkowo łatwo da się rozwiązać.

Mianowicie, należy stworzyć nowy projekt aplikacji ISAPI, usunąć z niego wszystkie moduły (oczywiście z wyjątkiem pliku *.*dpr*) i dodać do niego wszystkie moduły znajdujące się w źródłowej aplikacji .EXE, *z wyjątkiem formularza głównego*. Po zapisaniu i skompilowaniu, utworzona biblioteka .DLL będzie jak najbardziej poprawnym modułem ISAPI.

Koncepcję tę można rozszerzyć, posługując się dwiema wersjami aplikacji: do śledzenia przez Web App Debugger (*.EXE) oraz do docelowego zastosowania w serwerze WWW (ISAPI DLL) — obydwie wersje korzystałyby oczywiście z tych samym modułów. Na załączonym CD-ROM-ie, w podkatalogu przeznaczonym dla niniejszego rozdziału, znajdują się dwa pliki projektu — *DDG6Demo.dpr* i *DDG6DEmoISAPI.dpr*, co jest wyrazem tej właśnie koncepcji.

Podczas uruchamiania aplikacji ISAPI DLL nie należy tylko zapomnieć o jednej istotnej rzeczy: wszelkie wykorzystywane przez aplikację pliki HTML powinny znajdować się w tym samym katalogu, w którym znajduje się jej biblioteka DLL.

Zagadnienia zaawansowane

Przedstawione dotychczas przykłady ilustrujące m.in. zarządzanie informacjami o użytkownikach i ich preferencjach, nadzorowanie sesji użytkowników, wyświetlanie i manipulowanie danymi itp. są dość intuicyjne i stanowią wymowne świadectwo możliwości technologii WebSnap. WebSnap „potrafi" jednak jeszcze więcej, dając użytkownikom jeszcze większą kontrolę nad tworzonymi aplikacjami.

Komponent LocateFileService

Tworzenie aplikacji WebSnap wiąże się zazwyczaj z operowaniem wieloma plikami i zasobami — jak pliki HTML, skrypty serwera, moduły pascalowe, zbiory danych i komponenty sterowników, grafika, itp.; wszystko to musi oczywiście być w logiczny sposób powiązane ze sobą, by aplikacja mogła funkcjonować poprawnie. Najczęściej czynnikiem „koordynującym" są same strony HTML, w które wbudowuje się powiązane zasoby; WebSnap umożliwia wprawdzie przechowywanie stron HTML niezależnie od skompilowanego modułu wynikowego, jednak muszą one znajdować się w tym samym katalogu, w którym jest ów moduł wynikowy. Nie zawsze jest to możliwe i nie zawsze wygodne — pożądana byłaby możliwość przechowywania stron HTML np. w zasobach czy bazach danych.

Na taką okazję WebSnap oferuje komponent `LocateFileService`, umożliwiający pobieranie stron HTML z dowolnego źródła, które może być reprezentowane w formie strumienia danych (*stream*). Źródłem takim mógłby być na przykład zasób binarny, skonsolidowany z modułem wynikowym. Zaprezentujemy to rozwiązanie — umieścimy we wspomnianym zasobie kopię strony *wmLogin.html*, nazwaną *embed.html* i uzupełnioną stosownym tekstem wyjaśniającym, z którą stroną mamy faktycznie do czynienia.

Najprostszym środkiem implementowania zasobów są w Delphi pliki **.RC*. Umieszczone w projekcie, podlegają automatycznej kompilacji i konsolidacji z wynikowym modułem wykonywalnym. Plik .RC reprezentujący zasób-stronę HTML w naszym przykładzie ma następującą zawartość:

```
#define HTML 23 // identyfikator zasobu
EMBEDDEDHTML HTML EMBED.HTML
```

Po skompilowaniu aplikacji strona *EMBED.HTML* zostanie dołączona do modułu wykonywalnego jako zasób `EMBEDDEDHTML`.

Dodaj do naszej aplikacji kolejną stronę, nazwij ją *Embedded* i zapisz pod nazwą *wmEmbedded*. Przejdź do strony *Home*, wybierz komponent `LocateFileService` i korzystając z inspektora obiektów stwórz następującą procedurę obsługi zdarzenia `FindStream`:

```
procedure THome.LocateFileServiceFindStream(ASender: TObject;
  AComponent: TComponent; const AFileName: String;
  var AFoundStream: TStream; var AOwned, AHandled: Boolean);
begin
  // interesują nas tylko pliki zawierające w nazwie "wmembedded"
  if Pos('WMEMBEDDED', UpperCase(AFileName)) > 0 then
  begin
    AFoundStream := TResourceStream.Create(hInstance, 'EMBEDDEDHTML', 'HTML');
    AHandled := True; // plik został znaleziony
  end;
end;
```

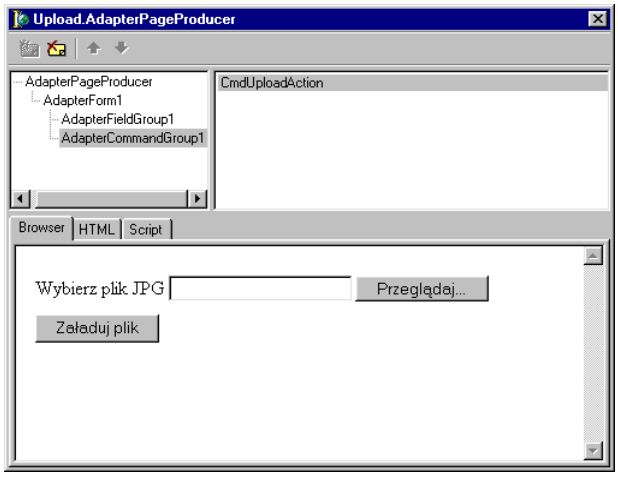
Parametr `AFileName` zawiera *niekwalifikowaną* nazwę poszukiwanej strony HTML, zaś parametrowi `AFoundStream` przypisywany jest strumień utworzony na podstawie zasobu `EMBEDDEDHTML`. Zwróć uwagę, iż zasób ten udostępniany będzie na *każde* żądanie, które dotyczy strony posiadającej wbudowany w nazwę człon „*wmembedded*" (niezależnie od wielkości liter); komponent `LocateFileService` zapewnia więc pewien stopień uniwersalizmu, polegającego na możliwości zastosowania tego samego pliku lub zasobu dla różnych żądań. Wynikowa wartość `True` parametru `AHandled` oznacza, iż żądanie zostało obsłużone i parametr `AFoundStream` reprezentuje strumień wynikowy.

Ładowanie plików

Jednym z najtrudniejszych zadań w aplikacjach internetowych jest obsługa przesyłania pliku od klienta do serwera, polegająca na pieczołowitej kontroli każdego nadchodzącego bajtu. WebSnap znacznie upraszcza to zadanie, powierzając większość jego skomplikowanych szczegółów komponentowi-adapterowi, a użytkownikowi pozostaje jedynie umieszczenie wspomnianego pliku w strumieniu.

Stwórz nową stronę o nazwie *Upload* i zapisz ją pod nazwą *wmUpload*. Umieść w jej module danych komponent `TAdapter` i dodaj do niego jedno pole `AdapterFileField`, które będzie służyć do wybierania przez klienta plików do załadowania. Dodaj także pojedynczą akcję i nazwij ją `UploadAction`.

Dodaj do komponentu `TAdapter` komponent `AdapterForm`, a do tego ostatniego — trzy komponenty: `AdapterErrorList`, `AdapterFieldGroup` i `AdapterCommandGroup`. Połącz dwa pierwsze z komponentem `Adapter1` (przez właściwość `Adapter`), natomiast `AdapterCommandGroup` połącz z `AdapterFieldGroup` (przez właściwość `DisplayComponent`). Następnie dodaj wszystkie pola komponentu `AdapterFieldGroup1` i wszystkie akcje komponentu `AdapterCommandGroup`. Zmień tytuł przycisku na *Załaduj plik*. Rysunek 23.17 przedstawia obraz strony w edytorze Web Surface Designer.



Rysunek 23.17. *Wygląd strony dokonującej załadowania pliku*

Konieczne jest ponadto dodanie kodu w dwóch miejscach modułu. Pierwszym jest procedura obsługi zdarzenia `OnFileUpload` komponentu `Adapter1.AdapterFileField` — przedstawia ją wydruk 23.5.

Wydruk 23.5. *Obsługa zdarzenia OnFileUpload*

```
procedure TUpload.AdapterFileField1UploadFiles(Sender: TObject;
  Files: TUpdateFileList);
var
  i: integer;
  CurrentDir: string;
  Filename: string;
  FS: TFileStream;
begin
  // załaduj plik
    if Files.Count <= 0 then
    begin
      Adapter1.Errors.AddError('Nie wybrano pliku do załadowania');
      Exit;
    end;
    for i := 0 to Files.Count - 1 do
    begin
    // Upewnij się, że plik ma rozszerzenie .jpg lub .jpeg
    if (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpg') <> 0)
        and (CompareText(ExtractFileExt(Files.Files[I].FileName), '.jpeg') <> 0)
    then
    begin
      Adapter1.Errors.AddError('Musisz wybrać plik JPG albo JPEG');
    end else
    begin
      CurrentDir := ExtractFilePath(GetModuleName(HInstance)) + 'JPEGFiles';
      ForceDirectories(CurrentDir);
      FileName := CurrentDir + '\' + ExtractFileName(Files.Files[I].FileName);
      FS := TFileStream.Create(Filename, fmCreate or fmShareDenyWrite);
      try
        FS.CopyFrom(Files.Files[I].Stream, 0); // skopiuj cały plik
      finally
        FS.Free;
      end;
    end;
  end;
end;
```

Poniższy kod dokonuje sprawdzenia, czy wybrano jakiś plik i czy plik ten ma rozszerzenie .jpg albo .jpeg, po czym sprawdza, czy katalog docelowy istnieje, czy też należy go utworzyć (funkcja `ForceDirectories`). Następnie tworzony jest strumień plikowy i do tego strumienia kopiowane są kolejno zawartości wyspecyfikowanych plików. Faktyczne działania związane z ładowaniem plików nie są widoczne na wydruku, skrywają się bowiem wewnątrz klasy `TUpdateFileList`.

Drugim zdarzeniem wymagającym oprogramowania jest zdarzenie `OnExecute` akcji `UploadAction` komponentu `Adapter1`:

```
procedure TUpload.UploadActionExecute(Sender: TObject; Params: TStrings);
begin
  Adapter1.UpdateRecords;
end;
```

Zdarzenie to wymusza odświeżenie rekordów komponentu `Adapter1` i związane z tym pobranie żądanych plików.

Wykorzystanie specyficznych szablonów

Kreatory WebSnap oferują użytkownikowi tylko dwa szablony określające początkową postać strony HTML: *Blank* i *Standard*. Jest to w zupełności wystarczające dla aplikacji ilustrujących ogólne zasady działania WebSnap (jak ta opisywana w niniejszym rozdziale), jednak w przypadku zadań bardziej skomplikowanych same szablony też są na ogół bardziej skomplikowane i przede wszystkim specyficzne dla danego zastosowania.

Standardowy zestaw szablonów HTML dla WebSnap można poszerzyć przez zdefiniowanie klasy pochodnej do `TProducerTemplateList` i zarejestrowanie jej w module umieszczonym w pakiecie środowiskowym. Szczegóły tej operacji możesz poznać studiując przykładową aplikację znajdującą się w podkatalogu *Demos\WebSnap\ Producer Template* zainstalowanego Delphi. Aby zarejestrować własne szablony, należy dodać reprezentujące je pozycje do pliku .RC, skompilować projekt i zainstalować w IDE wygenerowany pakiet. Do skompilowania tego projektu będzie jednak potrzebny inny pakiet — `TemplateRes` — którego pliki źródłowe znajdują się w podkatalogu *Demos\WebSnap\Util*.

Współpraca komponentu TAdapterPageProducer z komponentami tworzonymi przez użytkowników

W niniejszym rozdziale większość pracy związanej z wyświetlaniem stron HTML wykonywana była przez komponent `TAdapterPageProducer` i komponenty jemu podporządkowane. Zestaw tych ostatnich nie jest bynajmniej zamknięty — WebSnap umożliwia tworzenie własnych, na potrzeby tworzonych aplikacji. Każdy komponent przeznaczony do współpracy pod nadzorem komponentu `TAdapterPageProducer` musi wywodzić się z klasy `TWebContainingComponent` i implementować interfejs `IWebContent`. Stanowi to dobrą okazję do stworzenia klasy bazowej spełniającej te wymagania (patrz wydruk. 23.6).

Wydruk 23.6. *Klasa bazowa dla komponentów pracujących pod nadzorem TAdapterPageProducer*

```
type

  Tddg6BaseWebSnapComponent = class(TWebContainedComponent, IWebContent)
  protected
    { IWebContent }
    function Content(Options: TWebContentOptions; ParentLayout: TLayout): string;
    function GetHTML: string; virtual; abstract;
  end;

…

…

function Tddg6BaseWebSnapComponent.Content(Options: TWebContentOptions;
  ParentLayout: TLayout): string;
var
  Intf: ILayoutWebContent;
begin
  if Supports(ParentLayout, ILayoutWebContent, Intf) then
    Result := Intf.LayoutField(GetHTML, nil)
  else
    Result := GetHTML;
end;
```

Klasa Tddg6BaseWebSnapComponent implementuje tylko jedną metodę interfejsu — Content. Metoda ta sprawdza, czy komponent nadrzędny jest zgodny z klasą LayoutGroup; jeśli tak, to traktowany jest zgodnie z regułami tej klasy, w przeciwnym razie metoda pobiera swą wartość z metody GetHTML. Ta ostatnia jest *abstrakcyjna*, a więc każdy komponent pochodny musi ją implementować. Na załączonym CD-ROM-ie znajduje się definicja dwóch klas, które dostarczają komponentowi TAdapterPageProducer zawartość strony HTML w postaci (odpowiednio) pliku i listy łańcuchów. Definicję drugiej z nich przedstawiamy na wydruku 23.7.

Wydruk 23.7. *Komponent dostarczający zawartość strony HTML w postaci listy łańcuchów*

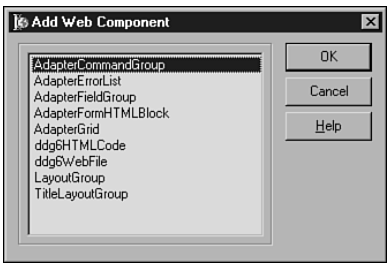
```
type
    Tddg6HTMLCode = class(Tddg6BaseWebSnapComponent)
      private
        FHTML: TStrings;
        procedure SetHTML(const Value: TStrings);
      protected
        function GetHTML: string; override;
      public
        constructor Create(AOwner: TComponent); override;
        destructor Destroy; override;
      published
        property HTML: TStrings read FHTML write SetHTML;
      end;


  …
  …

  constructor Tddg6HTMLCode.Create(AOwner: TComponent);
  begin
    inherited;
    FHTML := TStringList.Create;
  end;

  destructor Tddg6HTMLCode.Destroy;
  begin
    FHTML.Free;
    inherited;
  end;

  function Tddg6HTMLCode.GetHTML: string;
  begin
    Result := FHTML.Text;
  end;

  procedure Tddg6HTMLCode.SetHTML(const Value: TStrings);
  begin
    FHTML.Assign(Value);
  end;
```

Klasa Tddg6HTMLCode posiada właściwość HTML, przechowującą listę łańcuchową. Zawartość tej listy udostępniana jest (przez metodę GetHTML) komponentowi TAdapterPageProducer w postaci *łańcucha*, stanowiącego konkatenację łańcuchów składowych poprzedzielanych znakami nowego wiersza (#13#10). W ten sposób można przekazać każdą zawartość,

która da się skonwertować do postaci łańcucha — obrazek, plik itp. Aby jednak nowo zdefiniowany komponent mógł być zauważony przez Web Surface Designer, musi zostać zarejestrowany przez procedurę `RegisterWebComponents`, umieszczoną wewnątrz procedury `Register`, jak w module *WebSnapComps.pas* na załączonym CD-ROM-ie:

```
procedure Register;
begin
  RegisterWebComponents([Tddg6WebFile, Tddg6HTMLCode], ddg6Helper);
end;
```

Po zarejestrowaniu i zainstalowaniu, nowe komponenty stają się dostępne w oknie dialogowym opcji *New Component* (w edytorze Web Surface Designer — rys. 23.18).



Rysunek 23.18. *Zestaw nowych komponentów do wyboru w oknie dialogowym opcji New Component*

Podsumowanie

W niniejszym rozdziale przedstawiliśmy podstawy technologii WebSnap, efektywnie łączącej zalety szybkiego tworzenia aplikacji (RAD) z mechanizmami serwera WWW. Przedstawiliśmy przykładową aplikację, prezentującą rozmaite sposoby generowania wynikowych stron HTML, zajęliśmy się także kilkoma bardziej zaawansowanymi zagadnieniami związanymi z wykorzystaniem technologii WebSnap. Przy okazji zademonstrowaliśmy zastosowanie Web App Debuggera do śledzenia aplikacji i uruchamiania ich pod jego nadzorem, omówiliśmy także krótko problem konwersji aplikacji z „testowej" wersji .EXE do wymaganej przez rzeczywisty serwer WWW postaci ISAPI DLL.

WebSnap stwarza fascynujące możliwości w zakresie tworzenia aplikacji internetowych, lecz — jak miałeś okazję się przekonać, studiując zakończony właśnie rozdział — jest to technologia charakteryzująca się pewnym stopniem komplikacji i wymaga zrozumienia oraz pewnej wprawy. Pożyteczne uzupełnienie lektury niniejszego rozdziału stanowić będą z pewnością przykładowe projekty znajdujące się w podkatalogu *Demos\WebSnap* zainstalowanego Delphi 6.