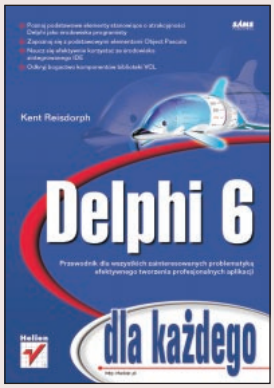
Delphi 6
dla każdego



Autor: Kent Reisdorph
Tłumaczenie: Andrzej Grażyński
ISBN: 83-7197-5465
Format: B5, stron: 864

Rynek narzędzi programistycznych typu RAD wzbogacił się niedawno o kolejną, szóstą już, edycję Delphi – potężne środowisko tworzenia aplikacji przeznaczonych dla Windows i (po raz pierwszy) dla platformy linuksowej. Treść niniejszej książki koncentruje się na praktycznych aspektach jego wykorzystania, stanowiąc przewodnik dla wszystkich zainteresowanych problematyką efektywnego tworzenia profesjonalnych aplikacji. Czytając ją:

- poznasz podstawowe elementy stanowiące o atrakcyjności Delphi jako środowiska programisty
- zaznajomisz się z podstawowymi elementami Object Pascala
- nauczysz się efektywnie korzystać ze środowiska zintegrowanego IDE
- odkryjesz bogactwo komponentów biblioteki VCL
- zobaczysz programistyczne oblicze Internetu

i być może stworzysz swą pierwszą aplikację dla Windows, własny komponent lub nową kontrolkę ActiveX. Zrozumiesz także, na czym polega łatwość tworzenia aplikacji za pomocą Delphi, analizując przykłady rozwiązań typowych zagadnień programistycznych – drukowania, rysowania grafiki, odtwarzania dźwięku i obrazu, tworzenia systemów pomocy, obsługi baz danych, korzystania z Rejestru itp.; poznasz także mechanizmy służące śledzeniu programów i usuwaniu błędów. Z pewnością pomogą Ci w tym ćwiczenia praktyczne towarzyszące poszczególnym rozdziałom, jak również dołączone do książki przykładowe aplikacje, stanowiące praktyczną ilustrację omawianych zagadnień.

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

Spis treści

Rozdział 8.

Tworzenie aplikacji w Delphi

Delphi udostępnia szeroki wybór narzędzi przeznaczonych do tworzenia formularzy, okien dialogowych i aplikacji. W niniejszym rozdziale zapoznasz się z następującymi zagadnieniami:

- Repozytorium
- Kreator dialogów
- Kreator aplikacji
- Dodawanie metod i pól do kodu
- Szablony komponentów
- Pakiety

Na początku poświęcę trochę czasu na omówienie repozytorium, w którym Delphi przechowuje wszelkie zbudowane wcześniej formularze, aplikacje lub inne obiekty w celu ponownego ich użycia. Następnie przyjdzie kolej na spotkanie z kreatorami. Kreatory udostępniają szereg okien dialogowych, które krok po kroku prowadzą Cię przez proces tworzenia formularza, aplikacji itp. — twoim zadaniem jest jedynie dostarczenie niezbędnych do tego celu informacji. W dalszej części rozdziału dowiesz się, jak korzystać z zasobów tworzonych w aplikacjach Delphi. Rozdział zakończę omówieniem pakietów Delphi.

Praca z repozytorium

Repozytorium (ang. *Object Repository*) umożliwia wybieranie predefiniowanych obiektów w celu wykorzystania ich w aplikacjach.

Dzięki repozytorium można:

- wybierać predefiniowane aplikacje, formularze lub okna dialogowe i implementować je w swoich projektach;
- dodawać do swoich aplikacji inne obiekty, takie jak dodatkowe pliki tekstowe czy moduły zawierające kod źródłowy;
- zarządzać modułami danych;
- tworzyć nowe komponenty;
- tworzyć nowe pakiety;
- tworzyć nowe kontrolki typu ActiveX lub formularze aktywne (*ActiveForms*);
- korzystać z pomocy kreatorów przy budowie okien dialogowych lub aplikacji.

Repozytorium nie stanowi zamkniętej całości, można bowiem:

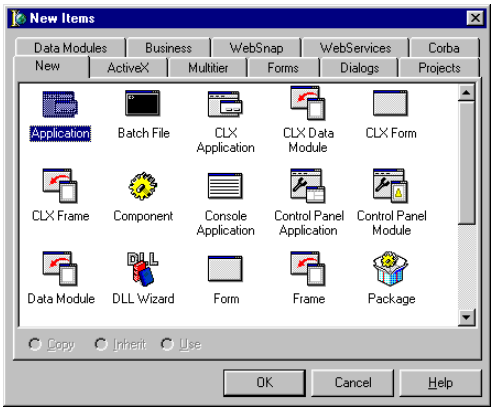
- dodawać do niego własne formularze, okna dialogowe i aplikacje.

Jest to tylko część możliwości oferowanych przez repozytorium — umożliwia ono tworzenie wielu innych obiektów nie wymienionych w powyższym zestawieniu.

Opcje i strony repozytorium

Repozytorium jest wyświetlane automatycznie za każdym razem, kiedy z głównego menu wybierasz polecenie *File|New|Other*. Rysunek 8.1 przedstawia okno repozytorium w jego pierwotnej postaci, po wybraniu opcji *File|New|Other* i przy braku otwartych projektów.

Rysunek 8.1.
Okno repozytorium



Nieco dziwnym może się wydać fakt, iż okno repozytorium nosi nazwę *New Items* („Nowe elementy"), a okno konfiguracji repozytorium nazwane zostało *Object Repository* („Repozytorium"). Nie da się ukryć, że jest to trochę mylące…

Repozytorium składa się z kilku stron, z których każda zawiera inne obiekty, możliwe do włączenia do Twojej aplikacji. Jak widzisz na rysunku 8.1, początkowo po otwarciu repozytorium wybraną zakładką jest *New*. Tabela 8.1 przedstawia strony repozytorium i opis elementów, jakie można znaleźć na każdej z nich; niektóre z wymienionych stron występują tylko w niektórych odmianach Delphi 6 — przykładowo strony `Multitier` i `WebSnap` obecne są jedynie w wersji `Enterprise`.

Tabela 8.1. *Strony repozytorium*

Strona/Zakładka	Opis
`New`	Pozwala na utworzenie nowej aplikacji, formularza lub modułu z przeznaczeniem do wykorzystania w Twojej aplikacji. Pozwala także tworzyć zaawansowane obiekty, np.: pakiety, biblioteki DLL, komponenty, aplikacje usługowe NT, aplikacje serwerów sieci i moduły danych.
`ActiveX`	Pozwala na stworzenie nowych kontrolek ActiveX, bibliotek typu, obiektów typu COM, formularzy aktywnych i innych obiektów ActiveX.
`Multitier`	Pozwala na stworzenie obiektów typu `CORBA` i `MTS`, a także modułów danych.
`Forms`	Pozwala na wykorzystanie formularzy wzorcowych, takich jak okno informacji o programie (*About*), podwójnych list, stron z zakładkami lub obiektów typu `QuickReports`.
`Dialogs`	Przedstawia wybór kilku podstawowych typów okien dialogowych. Zawiera także kreator dialogów.
`Projects`	Zawiera kompletne projekty wzorcowe, z których każdy może stanowić zaczątek tworzonej aplikacji. Zawiera również kreator aplikacji.
`Data Modules`	Pozwala na wybór modułów danych dla Twojej aplikacji.
`Business`	Zawiera kreatory przeznaczone dla formularzy baz danych, sieciowych aplikacji baz danych, raportów, wykresów, a także przykładową aplikację wykorzystującą komponent `Decision Cube`.
`WebSnap`	Zawiera kreatory aplikacji, modułu danych i modułu strony dla platformy WebSnap, służącej do tworzenia rozbudowanych aplikacji sieciowych.
`WebServices`	Zawiera kreatory serwerów sieciowych aplikacji usługowych (*Web Services*).
`Corba`	Zawiera kreatory tworzące aplikacje klienta i serwera zgodnie ze specyfikacją CORBA.

Jeżeli wywołasz *repozytorium* w chwili, kiedy będziesz miał już otwarty projekt, zobaczysz dodatkową zakładkę posiadającą nazwę Twojego projektu. Kliknięcie jej spowoduje wyświetlenie strony zawierającej wszystkie obiekty występujące aktualnie w projekcie. Pozwala to na szybkie, ponowne użycie formularza lub innego obiektu przez proste wybranie go z repozytorium.

U dołu okna znajdują się trzy przyciski opcji. Przyciski te, o nazwach `Copy`, `Inherit` i `Use` określają sposób implementowania wybranego obiektu. Czasem niektóre z tych przycisków są niedostępne — na przykład gdy wyświetlana jest strona `New`, nie jest dostępny żaden z nich, jedynym sposobem zaimplementowania obiektu z tej strony jest bowiem jego *skopiowanie* (odpowiadające opcji `Copy`), co Delphi wykonuje automatycznie.

Ze względu na swój charakter repozytorium nazywane jest niekiedy *galerią* (ang. *Gallery*).

Przycisk Copy

Po wybraniu przycisku opcji `Copy`, Delphi tworzy kopię wybranego obiektu i umieszcza ją w Twojej aplikacji. Od tego momentu możesz swobodnie modyfikować tę kopię na wszelkie możliwe sposoby; oryginał pozostaje w repozytorium nienaruszony.

Aby zilustrować tę sytuację załóżmy, że posiadasz często używany formularz („formularz" w sensie potocznym, nie w sensie Delphi) wydrukowany na papierze — np. harmonogram przydziału pracy. Powiedzmy, że chciałbyś wypełnić ten formularz wpisując pewne dane. Zapewne nie próbowałbyś wykorzystać oryginalnego formularza, ponieważ jego ponowne użycie stałoby się niemożliwe; zamiast tego zrobiłbyś jego kserokopię, oryginał zachował w bezpiecznym miejscu, a następnie wypełniłbyś kopię. Kopiowanie obiektów z repozytorium działa na identycznej zasadzie — jest to bodaj najbezpieczniejsza metoda ich używania.

Przycisk Inherit

Działanie opcji `Inherit` opiera się na mechanizmie dziedziczenia obiektów — na formularzu zostaje umieszczony obiekt dziedziczący wszystkie cechy oryginału z repozytorium. Przypomina to kopiowanie obiektu, chociaż tylko połowicznie — o ile bowiem wszelkie zmiany dokonane w obiekcie znajdującym się na formularzu pozostają bez wpływu na oryginał, to już wszelkie zmiany w ramach oryginału stają się skuteczne w stosunku do wszelkich obiektów wyprowadzonych z tego oryginału za pomocą opcji `Inherit`.

Aby wyobrazić sobie ten sposób implementacji obiektu, rozważ następujący scenariusz: autorzy sprawozdań często tworzą tabele za pomocą arkusza kalkulacyjnego, a następnie używają tych tabel w raportach tworzonych przy użyciu edytorów tekstu. Można oczywiście osadzić w dokumencie kopię tabeli (co jako żywo przypomina opcję `Copy`), jeżeli jednak chcemy zawsze dysponować jej aktualną wersją, należy umieścić w dokumencie jej łącznik. Mamy wówczas gwarancję, że wszelkie zmiany dokonane w tabeli widoczne będą we wszystkich dokumentach posiadających wspomniane łączniki.

Dziedziczenie obiektów z repozytorium jest szczególnie przydatne w przypadku formularzy. Jeżeli będziesz chciał użyć w swym projekcie kilku formularzy bazujących na wspólnym formularzu-przodku, powinieneś wybrać opcję `Inherit`, wówczas wszelkie zmiany w ramach formularza-przodka zostaną automatycznie odzwierciedlone w jego formularzach potomnych.

Przycisk Use

Opcja `Use` oznacza bezpośrednie włączenie do aplikacji oryginału znajdującego się w repozytorium. Wszelkie zmiany dokonywane w ramach obiektu znajdującego się

na formularzu przeprowadzane są *de facto* na jego *oryginale*, co powoduje ich odzwierciedlenie również we wszystkich projektach, które zaimplementowały ów obiekt za pomocą opcji `Use`.

Postępowanie takie ma sens np. w przypadku grupy projektów używających tego samego formularza — jeżeli podczas testowania jednego z tych projektów postanowimy wprowadzić do formularza pewne zmiany, nie będziemy musieli wprowadzać ich ponownie w pozostałych projektach.

Korzystanie z repozytorium

To, co dokładnie dzieje się w chwili, gdy wybierasz obiekt z repozytorium, zależy od kilku czynników: typu wybranego obiektu, stanu otwartego projektu, a także sposobu implementacji obiektu (`Copy`, `Inherit` albo `Use`).

I tak na przykład — jeżeli wybierzesz z repozytorium polecenie utworzenia nowej aplikacji, mając otwartą aplikację, to przed wyświetleniem nowego projektu zostaniesz poproszony o zapisanie ewentualnych zmian poczynionych w projekcie bieżącym.

Skrótem pozwalającym na utworzenie nowej aplikacji jest polecenie menu *File|New|Application*. Jest to odpowiednik polecenia polegającego na wybraniu opcji *File|New|Other* z głównego menu, a następnie wybrania obiektu `Application` z repozytorium. Analogicznie — polecenie *File|New|Form* w głównym menu jest odpowiednikiem analogicznego „polecenia" w repozytorium.

Sposób tworzenia nowego formularza na bazie repozytorium zależy od tego, czy w danej chwili otwarty jest jakikolwiek projekt. Jeżeli tak, to nowy formularz jest dodawany do tegoż projektu jako para formularz-moduł. Jeżeli nie jest otwarty żaden projekt, nowy formularz i moduł tworzone są jako obiekty samodzielne; taki „samodzielny" formularz może być np. dodany do repozytorium albo włączony (wraz z towarzyszącym mu modułem) do jakiegoś projektu (za pomocą opcji *File|Use Unit*).

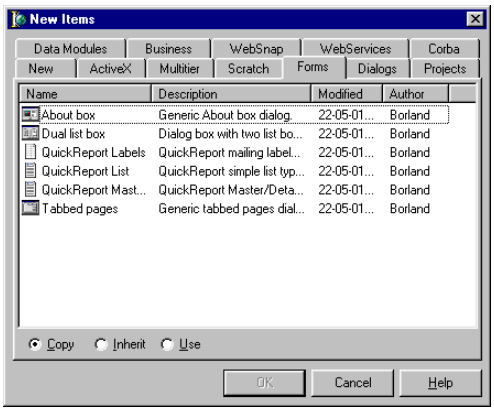
Jeżeli wybierzesz opcję utworzenia nowego modułu lub pliku tekstowego, zostanie on zwyczajnie utworzony w edytorze kodu (i — w przypadku modułu — dodany do bieżącego projektu). Powodów do stworzenia pliku tekstowego może być kilka. Załóżmy na przykład, że chcesz w swoich aplikacjach zaimplementować podobnie wyglądające pliki konfiguracyjne `.INI`; szkieletowa, wspólna dla wszystkich aplikacji, początkowa postać tego pliku jest idealnym kandydatem do umieszczenia w repozytorium. W podobny sposób możesz postąpić z często wykorzystywanymi modułami źródłowymi.

Wybranie opcji nowej biblioteki DLL spowoduje utworzenie nowego projektu z ustawieniami zorientowanymi na taką właśnie bibliotekę. Utworzenie nowego komponentu lub obiektu wątku spowoduje wyświetlenie okna dialogowego z prośbą o podanie szczegółowych informacji.

Widoki repozytorium

Rzeczywistym oknem repozytorium jest kontrolka Win32 widoku listy podobna do prawej strony Eksploratora Windows (tej jego części, w której wyświetlane są pliki). Okno takie może być wyświetlane w kilku wariantach: *Large Icons* (duże ikony), *Small Icons* (małe ikony), *List* (lista) i *Details* (szczegóły); domyślnie stosowany jest wariant *Large Icons*. Rysunek 8.2 przedstawia repozytorium z otwartą stroną formularzy (*Forms*) wyświetlaną w wariancie *Details* (zmianę wariantu wyświetlania umożliwia menu kontekstowe uruchamiane kliknięciem prawym przyciskiem myszy w obszarze okna).

Rysunek 8.2. *Repozytorium wyświetlane w wariancie Details*



Menu kontekstowe repozytorium zawiera również kilka opcji sortujących. Możliwe jest sortowanie według nazwy obiektu, opisu, daty lub autora.

Kiedy zawartość repozytorium wyświetlana jest w wariancie *Details*, kliknięcie nagłówka dowolnej kolumny spowoduje posortowanie wyświetlanej zawartości względem tejże kolumny.

Tworzenie nowych obiektów na bazie repozytorium

Z pewnością najbardziej podstawowym przeznaczeniem repozytorium jest tworzenie nowych obiektów przy użyciu pochodzących z niego obiektów wzorcowych. Aby to sobie wyobrazić, możesz stworzyć prostą aplikację z głównym formularzem — oknem informacji o programie (*About*) i drugim formularzem. Postępuj zgodnie z poniższymi instrukcjami:

1. Upewnij się, czy nie jest otwarty żaden projekt; w razie potrzeby zamknij aktywny projekt za pomocą opcji *File|Close All*. Wybierz polecenie menu *File|New|Other*. Wyświetlone zostanie repozytorium.

2. Kliknij ikonę `Application`, a następnie przycisk *OK*, aby utworzyć nową aplikację. Zostanie utworzony nowy projekt i wyświetlony pusty formularz.
3. Umieść na formularzu dwa przyciski. Zmień właściwość `Caption` przycisków na — odpowiednio — `O Programie...` i `Wyświetl Form2`. Jeżeli chcesz, możesz również zmienić właściwość `Name`.
4. Z głównego menu wybierz polecenie *File|New|Other*. Ponownie wyświetlone zostanie repozytorium. Kliknij zakładkę `Forms`.
5. Wybierz obiekt `About box`. Upewnij się, czy wybraną opcją jest `Copy`, a następnie kliknij przycisk *OK*, aby utworzyć nowy formularz. Wyświetlone zostanie okno formularza. Zmień dowolne ustawienia stosownie do własnych potrzeb.
6. Zmodyfikuj okno *About* (wpisz swoją własną informację, zmień ikonę, rozmiar, pozycję itp.)
7. Wybierz ponownie opcję *File|New|Other*. Repozytorium zostanie otwarte po raz trzeci.
8. Kliknij zakładkę `Forms`, a następnie wybierz obiekt o nazwie `Dual list box`. Kliknij przycisk *OK*, aby zamknąć repozytorium. Wyświetlony zostanie formularz podwójnej listy. (Nakazałem Ci wybrać ten obiekt tylko po to, abyś mógł go sobie obejrzeć.)
9. Napisz procedury obsługujące zdarzenia dla dwóch przycisków, które wyświetlają (odpowiednio) okno informacji o programie i drugi formularz. Nie zapomnij dodać nazw modułów obu formularzy do sekcji `uses` głównego formularza.
10. Skompiluj, uruchom i przetestuj program.

Oczywiście, ten program nie robi nic konkretnego, ale prezentuje pewien sposób wykorzystania repozytorium do szybkiego stworzenia prototypu aplikacji.

Dodawanie obiektów do repozytorium

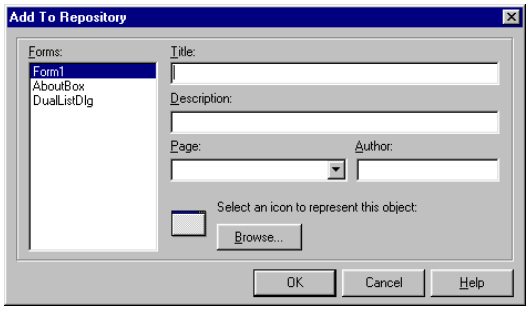
Repozytorium byłoby o wiele mniej użyteczne, gdybyś nie był w stanie dodawać do niego swoich własnych obiektów. Na szczęście możesz i powinieneś to robić. Dodawanie często używanych obiektów do repozytorium sprawi, że będziesz działał bardziej sprawnie, a przez to staniesz się bardziej cenionym programistą (nie ma przecież sensu ponowne odkrywanie Ameryki...).

Po stworzeniu aplikacji, formularza lub obiektu, zapisz go w repozytorium, dzięki czemu będziesz mógł ponownie z niego skorzystać w dowolnym momencie. Oczywiście nie powinieneś zapisywać w repozytorium *każdego* formularza, a tylko te, których będziesz używał najczęściej.

Możesz stworzyć obiekt z wyraźnym zamiarem dodania go do repozytorium lub dodać go w trakcie normalnego cyklu tworzenia aplikacji. (Termin *obiekt* jest pojęciem bardzo szerokim, dlatego użyję specyficznego przykładu, aby nadać sens powyższemu stwierdzeniu.) Powiedzmy, że w trakcie budowy aplikacji tworzysz formularz informacji o programie (*About*). Nagle świta Ci w głowie pomysł zapisania tego formularza, abyś mógł z niego korzystać we wszystkich przyszłych aplikacjach. W końcu znajduje się na nim nazwa Twojej firmy, Twoje logo, a także kompletna informacja na temat ochrony praw autorskich — i wszystko jest ułożone dokładnie według Twoich zamierzeń; szkoda byłoby więc odtwarzać to samo okno przy każdej nowej aplikacji, jaką będziesz tworzył. Właśnie w takich sytuacjach repozytorium okazuje się nieocenione.

Przed dodaniem formularza do repozytorium musisz go zapisać (jeżeli nie zapiszesz formularza, zostaniesz o to poproszony przed przystąpieniem do dalszych czynności). Następnie kliknij prawym przyciskiem myszy dowolny punkt należący do formularza i wybierz polecenie *Add To Repository* z otwartego menu kontekstowego projektanta formularzy. Kiedy to zrobisz, wyświetlone zostanie okno dialogowe dodawania do repozytorium (*Add To Repository*) — jak na rysunku 8.3.

Rysunek 8.3. *Okno dialogowe dodania obiektu do repozytorium*



Obszar listy o nazwie *Forms*, widoczny po lewej stronie okna, prezentuje bieżące formularze, jak również wszelkie inne obiekty występujące w aplikacji (np. moduły danych). Wybierz więc najpierw formularz, który chcesz dodać do repozytorium.

Formularz aktualnie aktywny w projektancie formularzy będzie od razu podświetlony na ww. liście.

Teraz wpisz nazwę obiektu (w pole *Title*). Nazwa ta pojawi się poniżej ikony w repozytorium. Pole *Description* umożliwia wprowadzenie dodatkowych informacji dotyczących obiektu. Opis ten jest wyświetlany, kiedy widok repozytorium jest ustawiony na wyświetlanie wszystkich szczegółów obiektów (wariant *Details* — jak na rysunku 8.2). W pole *Author* wpisz swoje dane osobowe — jesteś przecież autorem obiektu. Możesz tam wpisać imię i nazwisko, nazwę firmy lub jakąkolwiek inną nazwę identyfikującą.

Większość predefiniowanych obiektów w *repozytorium*, które dostarczone zostały razem z Delphi, zawiera w polu Author nazwę „Borland" — wyjątkiem są obiekty `QuickReport` i `TeeChart`.

Pole *Page* służy do wyboru strony w repozytorium, na której umieszczony zostanie dodawany obiekt. Możesz wybrać jedną z istniejących stron lub zwyczajnie wpisać nazwę nowej strony. Jeżeli strona o wpisanej nazwie nie istnieje, Delphi automatycznie ją utworzy. U dołu okna znajduje się przycisk o nazwie *Browse* służący do wyboru ikony, która reprezentować będzie obiekt.

Ikony możesz wybierać, między innymi, z katalogów `Borland Shared\Images\Icons` lub `Delphi6\Objrepos`. Ikony w katalogu `Delphi6\Objrepos` są używane przez Delphi dla obiektów znajdujących się w repozytorium.

Po wypełnieniu wszystkich pól i wybraniu stosownej ikony kliknij przycisk *OK*, aby ostatecznie dodać obiekt do repozytorium; obiekt zostanie dodany do strony, którą wybrałeś. Możesz odtąd używać tego obiektu na równi z pozostałymi. Jak sam widzisz, dodawanie obiektów do repozytorium jest tak proste, jak ich wykorzystywanie.

Przy dodawaniu obiektu do repozytorium Delphi tworzy (w specjalnym pliku) wpis zawierający informacje o obiekcie, przeważnie ścieżkę dostępu do plików kodu źródłowego i plików formularzy. Jeżeli przeniesiesz lub usuniesz plik formularza lub kodu źródłowego obiektu, nie będziesz mógł użyć go poprzez repozytorium.

Dodawanie projektów do repozytorium

Dodawanie projektów do repozytorium nie różni się zbytnio od dodawania do niego indywidualnych formularzy. Aby dodać projekt do repozytorium, wybierz opcję menu *Project|Add to Repository*. Wyświetlone zostanie okno dodania elementu do repozytorium (*Add To Repository*) — podobne do okna z rysunku 8.3, z tą jednak różnicą, że nie jest wyświetlana lista *Forms*. Wpisz wszelkie niezbędne informacje (nazwę, opis, dane autora), a następnie kliknij przycisk *OK*; efektem tego będzie dodanie projektu do repozytorium.

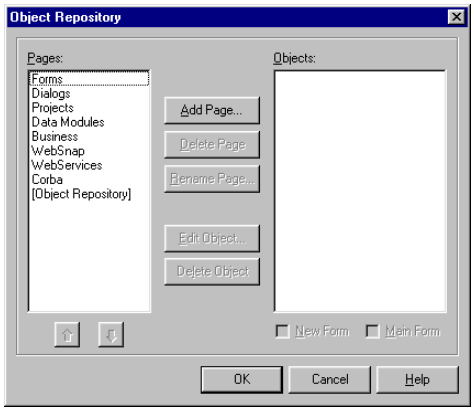
Kiedy już zaznajomisz się z Delphi, powinieneś utworzyć zarys aplikacji posiadający cechy, które najczęściej stosujesz w tworzonych przez siebie aplikacjach. Gdy będziesz rozpoczynał nową aplikację, za każdym razem pobierzesz kopię owego zarysu z repozytorium. W ten sposób w ciągu kilku sekund możesz mieć przygotowane np. wszystkie paski narzędzi i menu, okno informacji o programie i inne standardowe okna dialogowe. Po utworzeniu nowej aplikacji można w niej oczywiście dokonywać dowolnych modyfikacji, jak w przypadku każdego projektu.

Zarządzanie repozytorium

Istnieje możliwość zarządzania stronami i obiektami repozytorium poprzez użycie okna konfiguracji repozytorium.

Aby wyświetlić okno konfiguracji repozytorium, wybierz polecenie menu *Tools | Repository* lub — jeśli w danej chwili masz otwarte repozytorium — z jego menu kontekstowego wybierz polecenie *Properties*. Wygląd okna konfiguracji został przedstawiony na rysunku 8.4.

Rysunek 8.4.
Okno konfiguracji repozytorium



Okno to pozwala m.in. na usuwanie obiektów i stron z repozytorium, przenoszenie obiektów z jednej strony na inną, zmianę porządku stron itp. Wszystkie strony repozytorium znajdują się na liście o nazwie *Pages*, po lewej stronie okna. Wybranie (podświetlenie) jednej ze stron na tej liście spowoduje wyświetlenie wszystkich należących do niej obiektów — na liście o nazwie *Objects*, po prawej stronie okna.

Lista stron posiada dwie cechy godne uwagi. Po pierwsze, zauważ, że strona `New`, wyświetlana zawsze jako pierwsza po otwarciu repozytorium, nie znajduje się na liście (na liście nie ma również stron `ActiveX` i `Multitier`). Strona `New` jest bowiem z założenia niezmienna. Zauważ także, że na liście znajduje się element o nazwie *Object Repository*. Element ten symbolizuje ***wszystkie*** elementy zawarte w repozytorium.

Zarządzanie obiektami

Zanim będziesz mógł zmodyfikować, usunąć lub przenieść obiekt, musisz go najpierw wybrać. Aby wybrać obiekt, kliknij go na liście *Objects*. Po wybraniu obiektu możesz przejść do jego edycji — klikając przycisk *Edit Object*. Edycja obiektu pozwala na zmianę jego nazwy, opisu i informacji o autorze, a także strony repozytorium, na której obiekt jest wyświetlany.

Aby szybko przejść do edycji obiektu, wystarczy go dwukrotnie kliknąć na liście *Objects*.

Możesz usunąć obiekt poprzez wybranie go, a następnie kliknięcie przycisku *Delete Object*. Przed usunięciem obiektu wyświetlane jest okno z prośbą o potwierdzenie.

Po usunięciu z repozytorium obiekt przestaje być widoczny na jakiejkolwiek stronie, niemniej jednak związane z tym obiektem pliki formularza i kodu źródłowego nie są usuwane.

Obiekt można w prosty sposób przenieść z jednej strony na inną przez przeciągnięcie go z listy obiektów (*Objects*) na listę stron (*Pages*). Upuść obiekt na stronie, na której chcesz go umieścić, a zostanie on tam przesunięty.

Zarządzanie stronami repozytorium

Poprzedni punkt dotyczył edytowania, usuwania i przesuwania poszczególnych obiektów. Okno konfiguracji repozytorium umożliwia również dodawanie, usuwanie i przemieszczanie całych stron. Zanim będziesz mógł usunąć stronę, musisz najpierw usunąć wszystkie znajdujące się na niej obiekty. Gdy strona jest już pusta zaznaczasz ją na liście, a następnie klikasz przycisk *Delete Page*. Delphi usuwa wówczas stronę z repozytorium — po wcześniejszym sprawdzeniu, czy jest ona rzeczywiście pusta.

Nową stronę można dodać poprzez kliknięcie przycisku *Add Page*. Wyświetlone zostaje okienko dialogowe z prośbą o wpisanie nazwy strony. Kliknięcie przycisku *OK* (po wpisaniu nazwy) spowoduje, iż nowa strona pojawi się na liście *Pages*.

Zmiana nazwy strony odbywa się w podobny sposób: po wybraniu strony należy kliknąć przycisk *Rename Page*, po czym wyświetlone zostanie okno dialogowe, w którym można wpisać nową nazwę strony.

Kolejność stron w repozytorium również może ulec zmianie. Aby zmienić pozycję danej strony, kliknij ją, co spowoduje jej podświetlenie, a następnie używając pionowych strzałek znajdujących się pod listą, przesuń ją na wyższą lub niższą pozycję. Możesz także po prostu przeciągnąć stronę na jej nową pozycję.

Ustawianie domyślnych formularzy i projektów

Okno konfiguracji repozytorium pozwala na ustawienie trzech domyślnych obiektów:

- Domyślnego formularza — wybieranego poleceniem menu *File|New|Form*.
- Domyślnego formularza — używanego jako formularz główny po wybraniu polecenia menu *File|New|Application*.
- Domyślnego projektu — wybieranego poleceniem menu *File|New|Application*.

Zauważ, że w zależności od tego, jaki obiekt wybierzesz, poniżej listy *Objects* pojawi się jedno lub dwa pola wyboru. Jeżeli wybierzesz formularz, pojawią się pola wyboru o nazwach `New Form` i `Main Form`. Wybranie projektu spowoduje natychmiastowe pojawienie się pola wyboru o nazwie `New Project`.

Ustawienie formularza lub projektu domyślnym jest bardzo proste. Załóżmy, że tworzysz główny formularz i chcesz, żeby był on domyślnym formularzem głównym dla każdej nowej aplikacji. Wybierz ten formularz na liście *Objects*, a następnie kliknij pole wyboru `Main Form` widoczne pod listą. Kiedy klikniesz przycisk *OK*, formularz ten stanie się formularzem domyślnym. Podobnie, jeżeli posiadasz projekt, który chcesz uczynić projektem domyślnym, najpierw zlokalizuj go w oknie konfiguracji repozytorium; potem kliknij go, a następnie uaktywnij pole wyboru `New Project`. Od tego momentu, kiedy wybierzesz polecenie menu *File|New|Application* pojawiać się będzie „na dzień dobry" właśnie ten projekt.

Jeżeli nie będziesz uważał, możesz przez przypadek wybrać dowolny formularz jako domyślny formularz dla nowej aplikacji. Jeżeli tak się stanie, sprawdź każdy formularz występujący w oknie konfiguracji repozytorium. Jeden z nich będzie posiadał aktywne pole wyboru `Main Form`. Wyczyść to pole, a wszystko wróci do normalnego stanu. Uwaga ta dotyczy również domyślnego projektu. Sprawdź stronę projektów (Projects) w poszukiwaniu elementu posiadającego aktywne pole `New Project`.

Budowanie formularzy i aplikacji za pomocą kreatorów

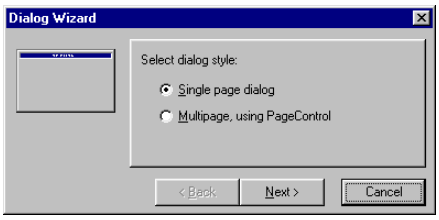
Delphi posiada dwa wbudowane kreatory przeznaczone do prowadzenia użytkownika przez proces tworzenia aplikacji. Kreator dialogów pomaga przy tworzeniu okien dialogowych, natomiast kreator aplikacji pomaga przy tworzeniu podstawowego układu aplikacji. Te dwa kreatory są omawiane w kolejnych punktach.

Korzystanie z kreatora dialogów

Mimo iż przy tworzeniu okna dialogowego z prawdziwego zdarzenia nie można raczej obejść się bez projektanta formularzy, to jednak podstawowe warianty okien dialogowych można w prosty sposób tworzyć za pomocą kreatora dialogów.

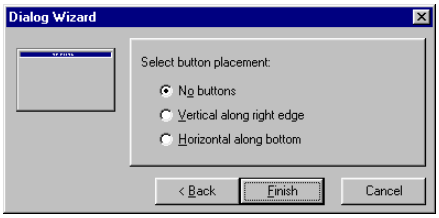
Kreator dialogów jest uruchamiany z wnętrza repozytorium — należy dwukrotnie kliknąć ikonę `Dialog Wizard` na stronie `Dialogs`, w wyniku czego pojawi się okno przedstawione na rysunku 8.5.

Rysunek 8.5.
Kreator dialogów



Możesz wybrać między utworzeniem jednostronicowego okna dialogowego (*Single page dialog*) i okna z zakładkami (wielostronicowego — *Multipage*). Ikona po lewej stronie okna prezentować będzie wygląd tworzonego okna dialogowego w kolejnych krokach. Jeżeli wybierzesz opcję utworzenia okna jednostronicowego, po kliknięciu przycisku *Next* zobaczysz następną stronę kreatora dialogów (rysunek 8.6).

Rysunek 8.6.
Druga strona kreatora dialogów

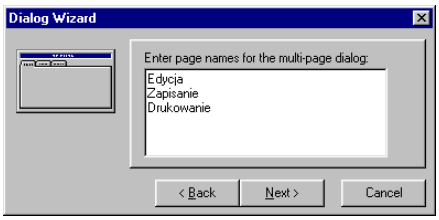


Strona ta umożliwia dokonanie wyboru zestawu obecnych w oknie przycisków poleceń, a także ich pozycji (po prawej stronie okna lub u jego dołu). Jest to ostatnia strona kreatora dialogów w przypadku okna jednostronicowego. Po wybraniu ułożenia przycisków kliknij przycisk *Finish*, a Delphi ostatecznie utworzy okno dialogowe.

Nowe okno dialogowe zostaje wyświetlone w projektancie formularzy razem ze wszystkimi cechami, jakie zostały mu nadane w kolejnych krokach kreatora dialogów. Posiada ono również właściwość `BorderStyle` ustawioną na wartość `bsDialog`, zwykle stosowaną w przypadku formularzy używanych jako okna dialogowe.

Jeżeli zdecydujesz się utworzyć okno z zakładkami, druga strona kreatora przyjmie postać przedstawioną na rysunku 8.7 (rysunek ten przedstawia okno dialogowe po dodaniu nazw stron.).

Rysunek 8.7.
Kreator dialogów w trakcie tworzenia okna dialogowego z zakładkami



Strona ta posiada obszar edycji, który służy do wpisywania nazw poszczególnych zakładek, jakie mają się pojawić w tworzonym oknie dialogowym. Wpisuj tekst każdej zakładki w pojedynczą linię, tak jak zostało to przedstawione na rysunku 8.7 Kiedy klikniesz przycisk *Next*, zobaczysz ostatnią stronę edytora dialogów, taką jak ta przedstawiona na rysunku 8.6. Wybierz umiejscowienie przycisków (lub opcję braku przycisków), a następnie kliknij przycisk *Finish*, co sprawi, że Delphi utworzy okno dialogowe z zakładkami.

Kreator dialogów jest najbardziej użytecznym narzędziem w przypadku tworzenia okien dialogowych z zakładkami. Tworząc jednostronicowe okna dialogowe łatwiej jest jednak wybrać z repozytorium jedno z predefiniowanych okien dialogowych.

Tworzenie aplikacji za pomocą kreatora aplikacji

Kreator aplikacji jest użytecznym narzędziem, które pomoże Ci szybko utworzyć powłokę aplikacji. Aby utworzyć nową aplikację z wykorzystaniem kreatora aplikacji, wybierz polecenie menu *File|New|Other*. Kiedy pojawi się repozytorium, kliknij zakładkę `Projects`, a następnie dwukrotnie kliknij ikonę kreatora aplikacji (*Application Wizard*).

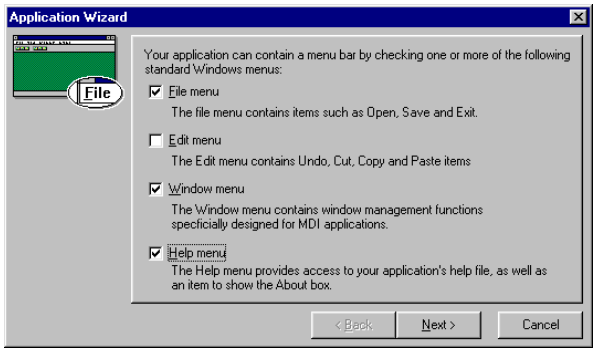
Polecenie menu *File|New|Application* tworzy nową aplikację na podstawie bieżących ustawień domyślnego projektu. Nie uruchamia ono — jak mogłoby się wydawać — kreatora aplikacji.

Przejdźmy kolejno przez wszystkie strony kreatora aplikacji.

Pierwsza strona: Wybór menu

Pierwsza strona, wyświetlana po uruchomieniu kreatora aplikacji, została przedstawiona na rysunku 8.8.

Rysunek 8.8.
Pierwsza strona kreatora aplikacji



Strona ta pozwala wybrać elementy, które chcesz umieścić w głównym menu swojej aplikacji. Możesz wybrać opcje dodające menu: *File* (Plik), *Edit* (Edycja), *Window* (Okno) i *Help* (Pomoc). Jeżeli chcesz, aby któreś z tych menu pojawiło się w Twojej aplikacji, zaznacz pole wyboru występujące obok jego opisu.

Menu *Window* (Okno) jest zwykle zarezerwowane dla aplikacji typu MDI. Nie ma potrzeby umieszczania go w przypadku aplikacji typu SDI, chyba że tworzysz aplikację specjalnego typu, która takiego menu wymaga.

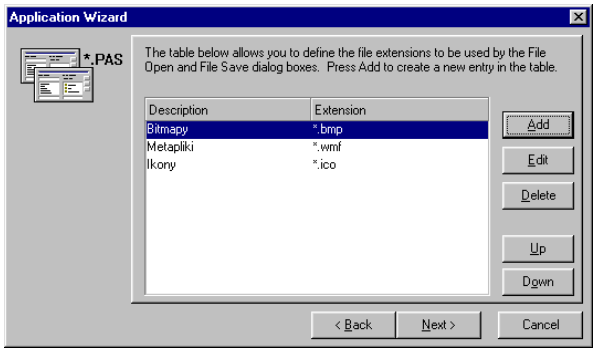
Menu kształtowane za pomocą kreatora aplikacji zawiera te elementy, które najczęściej występują w aplikacjach Windows. Pamiętaj, że zamierzeniem kreatora aplikacji jest stworzenie wygodnego punktu startowego dla nowej aplikacji; natomiast zadaniem programisty jest przekształcenie go w aplikację docelową.

Po wybraniu menu dla swojej aplikacji kliknij przycisk *Next*, aby przejść do następnej strony.

Strona druga: Ustawienie filtrów dla poleceń menu File

Jeżeli wybrałeś opcję dodania do aplikacji menu *File*, następna wyświetlona strona będzie wyglądać jak ta przedstawiona na rysunku 8.9.

Rysunek 8.9. *Ustawianie filtrów dla poleceń menu File*



Strona ta umożliwia ustawienie filtrów, które stosowane będą przez okna dialogowe *File|Open* (Otwórz plik) i *File|Save* (Zapisz plik) w Twojej aplikacji. (Rysunek 8.9 przedstawia okno dialogowe po dodaniu przykładowych filtrów.) Kliknij przycisk *Add*, aby dodać nowy filtr. Wyświetlone zostanie okno dialogowe z prośbą o wpisanie filtru i jego opisu. Wpisz filtry dokładnie w taki sam sposób, jak robisz to podczas ustawiania właściwości *Filter* „zwykłego" komponentu okna otwarcia pliku — wpisz opis tekstowy, a następnie rzeczywistą maskę plików (np. **.bmp*). W miarę potrzeby możesz użyć jednego z przycisków *Edit*, *Delete*, *Up* i *Down*, aby (odpowiednio) zmienić, usunąć lub przesunąć filtr na liście.

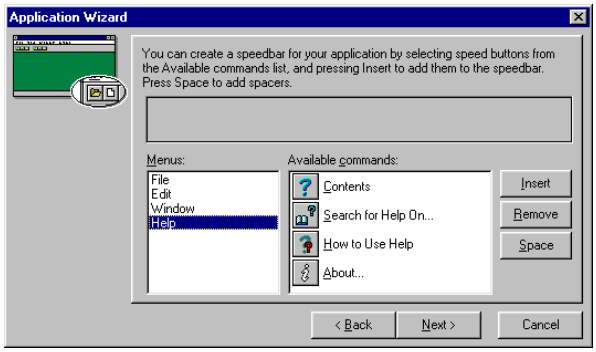
Strony druga i trzecia będą wyświetlone tylko w takim przypadku, gdy uprzednio wybrałeś jakiekolwiek menu na pierwszej stronie kreatora aplikacji. W szczególności, strona druga będzie wyświetlona tylko wtedy, gdy w pierwszym kroku wybrałeś menu *File*.

Strona trzecia: Ustawienie paska narzędzi

Trzecia strona kreatora aplikacji pomaga ustawić pasek narzędzi (zwany również paskiem „przyspieszającym" — ang. *speedbar*) dla tworzonej aplikacji; jest to prawdopodobnie jedna z najbardziej użytecznych cech kreatora aplikacji. Rysunek 8.10 przedstawia trzecią stronę kreatora aplikacji, po utworzeniu paska narzędzi.

Obszar listy po lewej stronie okna — o nazwie *Menus* — pokazuje cztery menu, do których możesz dodać przyciski. Kiedy wybierzesz jedno z nich, związane z nim przyciski zostaną wyświetlone na liście *Available Commands* po prawej stronie okna. Aby dodać przycisk paska narzędzi, najpierw go kliknij — w obszarze listy *Available Commands* — a następnie kliknij przycisk *Insert*. Wybrany przycisk zostanie dodany do przykładowego paska, widocznego u góry okna.

Rysunek 8.10.
Ustawianie paska narzędzi



Przycisk *Space* umożliwia wstawienie separatora; dodanie separatora rozróżnia wizualnie grupy przycisków. Dodawaj kolejne przyciski i separatory według własnego uznania, aż do momentu kiedy stwierdzisz, że pasek narzędzi jest kompletny. Jeżeli zechcesz usunąć któryś przycisk, kliknij go na odnośnym pasku, a następnie użyj przycisku *Remove*.

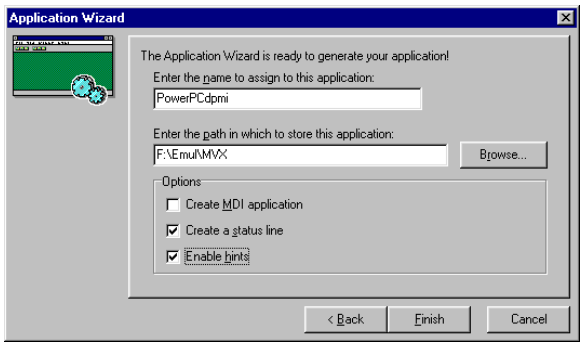
Jeżeli postanowiłeś nie dodawać określonego menu do swojej aplikacji, przyciski z nim związane nie zostaną pokazane. Przykładowo, jeżeli nie dodałeś menu ***Window***, po wybraniu go na liście ***Menus***, obszar listy ***Available Commands*** będzie pusty.

Niektóre specjalizowane aplikacje posiadają tylko pasek narzędzi — przy zupełnym braku menu. Kreator aplikacji tworzy pasek narzędzi tylko pod warunkiem wcześniejszego utworzenia menu. Aby obejść to ograniczenie, nakaż kreatorowi aplikacji utworzenie menu, a następnie zbuduj pasek narzędzi. Kiedy aplikacja zostanie wygenerowana, możesz usunąć z niej komponent `MainMenu` i tym samym pozbyć się niepotrzebnego menu głównego.

Strona czwarta: Wybór ustawień końcowych

Czwarta i ostatnia strona kreatora aplikacji pozwala na zdefiniowanie takich elementów, jak nazwa programu; ścieżka dostępu do katalogu, w którym projekt powinien być zapisany; model aplikacji (SDI albo MDI) oraz kilku innych ustawień. Rysunek 8.11 przedstawia ostatnią stronę kreatora aplikacji.

Rysunek 8.11.
Końcowe ustawienia kreatora aplikacji



Pierwsze pole na tej stronie służy do określenia nazwy aplikacji. Nie jest to nazwa, która pojawia się w oknie dialogowym ustawień projektu, ale nazwa pliku, jaką Delphi użyje do zapisania projektu (nazwę aplikacji można określić w oknie dialogowym ustawień projektu (*Project Options*). Drugie pole określa katalog, w którym projekt powinien zostać zapisany. Jeżeli nie znasz dokładnej ścieżki, kliknij przycisk *Browse* (znajdujący się po prawej stronie tego pola) i wybierz odpowiednią ścieżkę za pomocą okna dialogowego wyboru katalogu (*Select Directory*).

Okno dialogowe wyboru katalogu (*Select Directory*) służy zarówno do wyboru katalogu, jak i jego utworzenia. Kliknij przycisk *Browse*, aby otworzyć okno *Select Directory*. Wpisz ścieżkę dostępu do katalogu, który chcesz utworzyć, a następnie wybierz przycisk *OK* lub naciśnij *Enter*. Delphi wyświetli okno z prośbą o potwierdzenie utworzenia nowego katalogu (jeżeli katalog o wpisanej nazwie nie istnieje).

Dolna część ostatniej strony kreatora prezentuje trzy dodatkowe opcje. Jeżeli tworzysz aplikację typu MDI, kliknij pole wyboru o nazwie *Create MDI Application*. (Aplikacje typu MDI były omawiane w rozdziale 4. „Środowisko zintegrowane (IDE)"). Pozostałe dwa pola umożliwiają zaimplementowanie paska stanu i pomocy kontekstowej dla Twoich komponentów.

Po dokonaniu wszystkich niezbędnych wyborów kliknij przycisk *Finish*. Delphi, bazując na określonych przez Ciebie ustawieniach, utworzy aplikację. Delphi tworzy dla aplikacji maksymalną możliwą ilość kodu źródłowego. Nie oznacza to wniesienia do programu dużej ilości kodu, niemniej jednak pewne podstawowe elementy oprogramowania są już z góry napisane. Na przykład, jeżeli wybierzesz menu *File*, utworzona zostanie funkcja obsługująca zdarzenie kliknięcia opcji `File|Open`:

```
procedure TMainForm.FileOpen(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin

    …
    // kod realizujący dialog otwarcia pliku
    …

  end;
end;
```

Szkielet kodu aktywującego okno dialogowe otwarcia pliku (*File Open*) jest już na swoim miejscu. Twoim zadaniem jest napisanie kodu, który zajmie się obsługą nazwy pliku zwróconej przez to okno.

Po utworzeniu projektu za pomocą kreatora aplikacji możesz zapisać go w repozytorium, dzięki czemu w przyszłości zaoszczędzisz sobie kłopotu i ponownego wykonywania wszystkich kroków kreatora w celu stworzenia podstawowej struktury aplikacji.

Korzystanie z kreatorów cechuje się łatwością i szybkością pracy. Nie zwalniają one jednak użytkownika z obowiązku napisania właściwego programu, niemniej jednak Delphi daje możliwość szybkiego rozpoczęcia pracy nad programem, oszczędzając programiście niewdzięcznej roboty, związanej z tworzeniem podstawowych elementów aplikacji. Delphi jest typem przyjaznego oprogramowania służącego do szybkiej rozbudowy aplikacji (RAD), a kreatory jawią się jako narzędzia jeszcze bardziej ten proces upraszczające. Można powiedzieć, że kreatory Delphi same w sobie są narzędziami typu RAD.

Oprócz kreatorów aplikacji i dialogów, Delphi udostępnia również inne narzędzia tego typu. W rozdziale 15. przedstawiony został kreator wspomagający tworzenie kontrolek ActiveX, zaś w rozdziale 17. — kreator formularzy baz danych.

Dodawanie metod i pól do kodu

Jak już się przekonałeś, Delphi jest doskonałym narzędziem służącym do szybkiej budowy wizualnej części aplikacji związanej z interfejsem użytkownika (ang. *user interface*). Delphi samodzielnie buduje funkcje obsługujące zdarzenia, następnie pozwala programiście uzupełnić je stosownym kodem.

Tworzenie złożonej aplikacji oznacza jednak także samodzielną rozbudowę kodu źródłowego. Częściowo oznacza to dodawanie własnych pól danych i metod do kodu, który został stworzony przez Delphi. Typowa aplikacja może na przykład zawierać dwa tuziny procedur zdarzeniowych, których szkielety wygenerowane zostały automatycznie, niezależnie jednak od tego zawiera ona najprawdopodobniej kolejne dwa tuziny metod stworzonych niezależnie od generowania automatycznego.

Dodawanie własnych metod i pól do kodu wygenerowanego przez Delphi nie jest zadaniem trudnym pod warunkiem, że odbywa się z zachowaniem pewnych zasad; w przeciwnym razie można narobić sobie niemało kłopotów.

W jaki sposób Delphi zarządza deklaracjami klas?

Jak już wiesz, w chwili, kiedy tworzysz nowy formularz w projektancie formularzy, Delphi automatycznie tworzy plik modułu z kodem źródłowym. Kiedy Delphi tworzy deklarację klasy, w rzeczywistości tworzy dwie sekcje. Pierwsza sekcja jest częścią deklaracji klasy zarządzaną przez Delphi, druga dostępna jest dla Twoich ingerencji.

W rozdziale 6. „Praca z projektantem formularzy i projektantem menu", stworzyłeś program `Scratch`. Jeżeli wykonałeś ćwiczenia zamieszczone na końcu rozdziału, zbudowałeś również okno informacji o programie (*About*) i dodałeś kilka dodatkowych przycisków. Wydruk 8.1 przedstawia deklarację klasy głównego formularza, po uwzględnieniu wszystkich wspomnianych elementów.

Pamiętaj, że deklaracje indywidualnych komponentów pojawiają się w kolejności zgodnej z kolejnością umieszczania ich na formularzu. Twoja klasa powinna więc zawierać takie same deklaracje komponentów, jak te przedstawione na wydruku 8.1, ale mogą one występować w innej kolejności.

Wydruk 8.1. *Deklaracja klasy głównego formularza programu Scratch*

```
type
  TMainForm = class(TForm)
    StatusBar: TStatusBar;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    Memo: TMemo;
    MainMenu: TMainMenu;
    FileMenu: TMenuItem;
    FileNew: TMenuItem;
    FileOpen: TMenuItem;
    FileSave: TMenuItem;
    FileSaveAs: TMenuItem;
    N1: TMenuItem;
    FilePrint: TMenuItem;
    FilePrintSetup: TMenuItem;
    N2: TMenuItem;
    FileExit: TMenuItem;
    Edit1: TMenuItem;
    EditReplace: TMenuItem;
    EditFind: TMenuItem;
    N4: TMenuItem;
    EditPaste: TMenuItem;
    EditCopy: TMenuItem;
    EditCut: TMenuItem;
    N5: TMenuItem;
    EditUndo: TMenuItem;
    Help1: TMenuItem;
    HelpAbout: TMenuItem;
    HelpContents: TMenuItem;
    EditSelectAll: TMenuItem;
    N3: TMenuItem;
    EditWordWrap: TMenuItem;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    MemoPopup: TPopupMenu;
    PopupCut: TMenuItem;
    PopupCopy: TMenuItem;
    PopupPaste: TMenuItem;
    procedure FileExitClick(Sender: TObject);
    procedure EditCutClick(Sender: TObject);
    procedure EditCopyClick(Sender: TObject);
    procedure EditPasteClick(Sender: TObject);
    procedure FileNewClick(Sender: TObject);
    procedure FileSaveClick(Sender: TObject);
    procedure FileOpenClick(Sender: TObject);
    procedure FileSaveAsClick(Sender: TObject);
    procedure EditUndoClick(Sender: TObject);
    procedure EditSelectAllClick(Sender: TObject);
```

```
  procedure EditWordWrapClick(Sender: TObject);
  procedure HelpAboutClick(Sender: TObject);
  procedure FormCreate(Sender: TObject);
private
  { Private declarations }
  procedure MyOnHint(Sender : TObject);
public
  { Public declarations }
end;
```

Zapamiętaj, że sekcja pomiędzy pierwszą linią deklaracji klasy a słowem kluczowym `private` powinna być traktowana jako „strefa zakazana" — jest ona kontrolowana wyłącznie przez Delphi i nie wolno jej modyfikować. Złamanie tej zasady może spowodować, iż program przestanie się kompilować, a w skrajnym wypadku może ulec zniszczeniu plik formularza..

Deklaracje swoich własnych pól i metod możesz bezpiecznie umieszczać w sekcjach `private` i `public` deklaracji klasy. Oczywiście możesz również dodać sekcję `protected` i tam umieścić swoje pola i metody.

Słowo o paskach stanu i pomocy kontekstowej

Za chwilę zmodyfikujesz program `Scratch`, dodając do niego funkcje wyświetlania pomocy kontekstowej na pasku statusu. Zanim jednak przystąpisz do pracy, powinieneś przejść krótkie szkolenie dotyczące sposobu, w jaki realizowany jest mechanizm pomocy kontekstowej.

Kiedy właściwość obiektu `Application` o nazwie `ShowHint` jest ustawiona na wartość `True` (domyślnie), a kursor myszy zostanie umieszczony nad komponentem, którego właściwość `ShowHint` jest także ustawiona na wartość `True`, generowane jest zdarzenie `OnHint` obiektu `Application`. Właściwość `Hint` tego obiektu zawiera wtedy tekst pomocy kontekstowej obiektu, który zdarzenie to wygenerował. Aplikacja może więc użyć zdarzenia `OnHint` do wyświetlenia tekstu pomocy na pasku stanu.

Problem jednak w tym, że nie można w sposób bezpośredni uzyskać dostępu do zdarzenia `OnHint` obiektu `Application`. To, co możesz zrobić, to przypisanie zdarzeniu `OnHint` wskazania na jedną z Twoich metod. Kiedy pojawi się zdarzenie pomocy kontekstowej, zostanie ono przekierowane do Twojej własnej metody obsługującej zdarzenie `OnHint`. Aby tak się stało, musisz napisać swoją własną procedurę obsługi zdarzenia `OnHint`. Tym zajmiemy się w następnej sekcji.

Dodawanie metod do kodu

Aby zilustrować dodawanie metod do aplikacji, zaimplementujmy pomoc kontekstową dla programu `Scratch`, który wcześniej stworzyłeś.

Najpierw ponownie otwórz ten program, by w kolejnych krokach dodać teksty pomocy do każdego z przycisków paska narzędzi, a także przygotować pasek stanu na wyświetlanie

tekstów pomocy. Pamiętaj, że przyciski paska narzędzi, które dodałeś w rozdziale szóstym, aktualnie są tylko atrapami; nie przeszkadza to jednak w dodaniu do nich pomocy kontekstowej. Postępuj według poniższego scenariusza:

1. Upewnij się, czy widoczny jest główny formularz programu. Kliknij pierwszy przycisk jego paska narzędzi.
2. Korzystając z inspektora obiektów, zlokalizuj właściwość *Hint* i wpisz następujący tekst pomocy kontekstowej:

   ```
   Otwórz|Otwórz istniejący plik
   ```

3. Zmień wartość właściwości `ShowHint` na `True`.
4. Powtórz kroki 2 i 3 dla pozostałych przycisków na pasku narzędzi, dodając tekst pomocy, który uznasz za właściwy.
5. Kliknij pasek statusu, znajdujący się u dołu głównego formularza. Zmień jego właściwość `SimplePanel` na wartość `True`. Dzięki temu cały pasek stanu będzie mógł wyświetlać łańcuchy tekstowe poprzez swą właściwość `SimpleText`.

Wszystko zostało przygotowane, nadszedł więc czas na utworzenie własnej funkcji obsługującej zdarzenie `OnHint` – nazwijmy ją `MojeOnHint`:

1. Przejdź do edytora kodu i upewnij się, że widoczny jest plik *SPMain.pas*.
2. Przejdź na dół przez deklarację klasy `TScratchPad`, aż znajdziesz sekcję `private`. Dodaj następującą linię kodu, poniżej słowa kluczowego `private`:

   ```
   procedure MojeOnHint(Sender: TObject);
   ```

   Abyś lepiej mógł orientować się w pliku, poniżej zostało przedstawionych kilka końcowych linii deklaracji klasy:

   ```
   private
      { Deklaracje prywatne }
       procedure MojeOnHint(Sender: TObject);
   public
      {Deklaracje publiczne }
   end;
   ```

Teraz należy wpisać implementację metody do sekcji `implementation`, a następnie przypisać tę metodę do zdarzenia `OnHint`:

1. Przejdź na koniec sekcji `implementation`.
2. Wpisz poniższy fragment kodu źródłowego (powyżej końcowego słowa kluczowego `end`):

   ```
   procedure TMainForm.MojeOnHint(Sender: TObject);
   begin
     StatusBar.SimpleText:=Application.Hint;
   end;
   ```

3. Przejdź do inspektora obiektów. Z listy wyboru obiektów wybierz formularz główny.

4. W inspektorze obiektów przejdź do strony `Events` i kliknij dwukrotnie kolumnę `Value` obok zdarzenia `OnCreate`. Otwarty zostanie edytor kodu, gotowy do wpisywania nowego kodu źródłowego.

5. Uzupełnij następująco metodę `FormCreate`:

```
procedure TMainForm.FormCreate(Sender : TObject);
begin
  Application.OnHint:=MojeOnHint;
end;
```

6. Skompiluj i uruchom program. Kiedy umieścisz kursor myszy nad dowolnym przyciskiem paska narzędzi, na pasku stanu pojawi się długi tekst pomocy kontekstowej. Krótki tekst pomocy kontekstowej zostanie wyświetlony jako podpowiedź, jeżeli na chwilę zatrzymasz kursor myszy nad przyciskiem.

W drugim kroku tekst pomocy kontekstowej (pochodzący z właściwości `Hint` obiektu `Application`) jest przypisywany właściwości `SimpleText` komponentu `StatusBar`. W kroku piątym metoda, którą stworzyłeś w drugim kroku, jest przypisywana zdarzeniu `OnHint` klasy `TApplication`. Od tego momentu, za każdym razem, kiedy pojawi się zdarzenie `OnHint`, wywoływana będzie metoda `MojeOnHint` powodująca wyświetlenie tekstu pomocy kontekstowej na pasku stanu.

Wyświetlanie tekstu długiej pomocy kontekstowej na pasku statusu można także zrealizować w prostszy sposób, mianowicie, ustawiając na `True` jego właściwość `AutoHint`.

Dodawanie pól danych do klas

Dodawanie pól do klas wygenerowanych przez Delphi odbywa się w identyczny sposób. Musisz jedynie upewnić się — podobnie jak w przypadku własnych metod — że dodajesz pole do publicznej (`public`) lub prywatnej (`private`) sekcji deklaracji klasy — bądź do sekcji chronionej (`protected`), o ile taką stworzyłeś.

Usuwanie kodu wygenerowanego przez Delphi

Może się okazać, że będziesz musiał usunąć fragment kodu stworzonego przez Delphi. Na przykład w formularzu może występować przycisk, który z pewnych względów nie jest już dłużej potrzebny. Oczywiście, wszystko co trzeba zrobić, aby usunąć przycisk z formularza, to wybrać go w projektancie formularzy i nacisnąć przycisk *Delete* na klawiaturze. Przycisk przestaje istnieć jako komponent, ale skojarzona z nim procedura obsługująca zdarzenie `OnClick` pozostaje w kodzie.

Delphi „wie", że przycisk skojarzony ze wspomnianą procedurą obsługi zdarzenia `OnClick` nie istnieje, ale mimo to nie usuwa jej, ponieważ istnieje prawdopodobieństwo, że z tej samej procedury korzystają również inne komponenty. Od Ciebie więc zależy, czy chcesz usunąć tę procedurę z kodu.

Być może spotkałeś się ze stwierdzeniem, iż w przypadku niepewności, czy dana funkcja jest używana przez inne komponenty, najlepiej pozostawić ją w spokoju. Nie podzielam tego poglądu — moim zdaniem programista musi być odpowiedzialny za to, co znajduje się kodzie źródłowym i pozbywać się wszystkich nie używanych metod. Chociaż nie używany kod nie stanowi żadnego zagrożenia, powoduje on powstawanie obszerniejszych plików wykonywalnych (*.exe*). W niektórych przypadkach nadmiar takiego kodu może prowadzić do spadku wydajności programu. Dbaj o to, aby w Twoich aplikacjach nie pozostawał nie używany lub nieefektywny kod.

Tworzenie szablonów komponentów

Szablonem komponentu nazywany jest komponent (lub grupa komponentów), modyfikowany według własnych potrzeb, a następnie zapisywany w celu późniejszego wykorzystania.

Szablony komponentów pozwalają tworzyć, zapisywać i ponownie wykorzystywać grupy komponentów. W rzeczywistości szablon komponentu wcale nie musi być grupą komponentów — może on być pojedynczym komponentem. Za chwilę pokażemy na prostym przykładzie, jak użyteczne mogą być szablony komponentów.

Standardowa kontrolka Windows — jednoliniowe pole edycyjne, jak wszystkie kontrolki Windows, cechuje się pewnymi predefiniowanymi sposobami zachowania. Jedno z tych zachowań dotyczy sposobu reagowania na naciśnięcie klawisza *Enter* — Windows poszukuje wówczas na formularzu domyślnego przycisku (tj. przycisku z ustawioną na `True` właściwością `Default`) i jeżeli go znajdzie, symuluje jego naciśnięcie (wywołując procedurę obsługi zdarzenia `OnClick`). Jeżeli będzie to np. przycisk OK, służący do zamknięcia wyświetlonego modalnie formularza, formularz ten istotnie zostanie zamknięty.

Mimo że jest to standardowe zachowanie Windows, wielu użytkowników traktuje je jako irytujące i mylące. W sytuacji, gdy na formularzu znajduje się kilka kontrolek edycyjnych, użytkownik oczekiwałby raczej, iż naciśnięcie klawisza *Enter* przeniesie aktywność na następną kontrolkę.

Rozwiązanie tego problemu jest naprawdę proste. Wszystko, co trzeba zrobić, to zmiana standardowej obsługi zdarzenia `OnKeyPress`:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key : Char);
begin
  if Key = Char(VK_RETURN) then
  begin
    Key:=#0;
    PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
  end;
end;
```

Powyższa procedura zdarzeniowa sprawdza, czy naciśniętym klawiszem był *Enter* (jego wirtualnym kodem jest `VK_RETURN`). Jeżeli tak, zmienna zawierająca kod klawisza

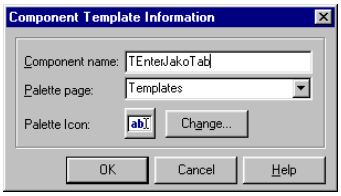
zostaje przestawiona na wartość #0, co powoduje, iż naciśnięcie klawisza uznaje się za niebyłe. Dzięki temu eliminowane jest niepożądane zamykanie formularza (w najmniej oczekiwanym momencie). Instrukcja w następnej linii powoduje wysłanie do formularza komunikatu `WM_NEXTDLGCTL`, który powoduje przeniesienie aktywności na kontrolkę następną w kolejności „cyklu Tab".

Tak zmodyfikowaną kontrolkę edycyjną można zapisać jako szablon komponentu. W tym celu postępuj zgodnie z poniższym scenariuszem:

1. W pustym formularzu umieść komponent `Edit`. Zmień jego właściwość `Name` na wartość `EnterJakoTab` i wyczyść właściwość `Text`.
2. W inspektorze obiektów przejdź na stronę zdarzeń (`Events`) i stwórz procedurę obsługującą zdarzenie `OnKeyPress`, poprzez wpisanie do jej wnętrza następującego kodu:

```
if Key = Char(VK_RETURN) then
begin
  Key:=#0;
  PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
end;
```

3. Upewnij się, że wybranym komponentem jest `EnterJakoTab`, a następnie wybierz z menu głównego polecenie *Component | Create Component Template*. Wyświetlone zostanie okno dialogowe informacji o szablonach komponentów (*Component Template Information*).
4. W pole nazwy komponentu (*Component Name*) wpisz `TEnterJakoTab`. Okno dialogowe powinno w tym miejscu wyglądać tak, jak zostało to przedstawione na rysunku 8.12.



Rysunek 8.12. *Okno informacji o szablonie komponentu*

5. Kliknij przycisk *OK*, aby zapisać szablon.

Od teraz paleta komponentów będzie posiadać stronę o nazwie Templates. Przejdź do niej, wybierz swój nowy komponent i umieść go na formularzu. Przekonasz się, że komponent umieszczony na formularzu zawiera funkcję obsługującą zdarzenie `OnKeyPress`.

Dzięki szablonom komponentów możesz mieć w każdej chwili dostęp do zbioru komponentów zmodyfikowanych według własnych potrzeb: okienek dialogowych z predefiniowanymi filtrami, nazwami i klawiszami-akceleratorami, list lub obiektów *combo*, automatycznie ładujących swoje elementy z pliku itp.

Jeżeli na formularzu umieścisz kilka komponentów typu `EnterJakoTab`, kod procedury obsługującej zdarzenie `OnKeyPress` zostanie powtórzony dla każdego z nich (w kodzie źródłowym znajdzie się kilka procedur o identycznej treści). Aby uniknąć duplikowania kodu, wystarczy na formularzu umieścić tylko jeden komponent `EnterJakoTab`. Pozostałe komponenty edycyjne mogą być standardowymi polami edycji (`TEdit`), którym zmieniono jedynie obsługę zdarzenia `OnKeyPress`, przypisując mu procedurę obsługującą to zdarzenie w komponencie `TEnterJakoTab`.

Chociaż koncepcja szablonów komponentów znajduje swoje zastosowanie w przypadku pojedynczego komponentu, ma ona jeszcze większy sens w przypadku wielu komponentów. Jeżeli często zdarza Ci się umieszczać tę samą grupę komponentów na formularzu, możesz stworzyć z nich szablon. Po utworzeniu takiego szablonu komponentu, całą grupę można sprowadzić na formularz za pomocą pojedynczego kliknięcia.

Z pewnością zachodzi pewne podobieństwo pomiędzy szablonami komponentów, a zapisywaniem formularzy w repozytorium. Używaj szablonów w przypadku grup komponentów, które zazwyczaj stosujesz jako część większego formularza. Natomiast w repozytorium zapisuj kompletne formularze, które masz nadzieję wykorzystać ponownie w przyszłości.

W rozdziale 4. opisane zostały ramki funkcjonujące jako szablony komponentów.

Korzystanie z plików zasobów

Każdy program Windows korzysta z zasobów. *Zasoby* są tymi elementami programu, które wspomagają go, ale nie są kodem wykonywalnym.

Do typowych zasobów programów Windows należą:

- Klawisze-akceleratory
- Bitmapy
- Kursory
- Okna dialogowe
- Ikony
- Menu
- Tablice danych
- Tablice łańcuchów
- Informacja o wersji
- Specjalne zasoby definiowane przez użytkownika (np. pliki dźwiękowe i wideo)

Dodanie do projektu informacji o wersji programu umożliwia zakładka *Version Info*, widoczna w oknie opcji projektu (*Project Options*). Okno to zostanie dokładnie omówione w następnym rozdziale.

Zasoby są przechowywane zazwyczaj w *pliku skryptowym zasobów* (pliku tekstowym z rozszerzeniem *.rc*). Plik skryptowy zasobów (ang. *resource script file*) podlega procesowi kompilacji, po czym dodawany jest do pliku typu *.exe* w fazie konsolidacji.

Zwykle zasoby traktowane są jako dane związane z plikiem wykonywalnym. Niektóre zasoby, takie jak bitmapy, tablice łańcuchów i pliki dźwiękowe, mogą być umieszczane w plikach zewnętrznych (*.bmp, .txt* i *.wav*) lub mogą stanowić fragment pliku *.exe*. Możesz zdecydować się na jedną z powyższych opcji. Przechowywanie zasobów w pliku *.exe* niesie ze sobą dwie główne korzyści:

- Dostęp do zasobów jest szybszy, ponieważ zlokalizowanie zasobu w pliku wykonywalnym zajmuje mniej czasu, niż załadowanie go z dysku.
- Plik programu i zasobów może być przechowywany w postaci pojedynczego modułu (pliku *.exe*), bez dodatkowych plików wspomagających.

Konsekwencją takiego podejścia będzie jednakże zwiększenie rozmiaru pliku *.exe*. Rozmiar pliku programu nie będzie co prawda większy od sumarycznego rozmiaru zewnętrznych plików zasobów i pliku wykonywalnego, ale znaczny wzrost tego rozmiaru może w efekcie wydłużyć czas ładowania programu. Wybór konkretnego rozwiązania — odrębne pliki zasobów czy komasacja ich z plikiem wykonywalnym — jest każdorazowo sprawą konkretnego programu; w szczególności nie wyklucza się stosowania obydwu tych metod.

Zasoby w Delphi

W większości przypadków tradycyjny program Windows składa się przynajmniej z jednego okna dialogowego i ikony. W przypadku aplikacji Delphi sprawa wygląda jednak odrobinę inaczej. Po pierwsze, w aplikacjach Delphi nie występują prawdziwe okna dialogowe, a więc nie występują również ich zasoby (binarny plik *.DFM* jest plikiem zasobowym, ale jest to zasób typu `RCDATA`, a nie zasób okna dialogowego).

Typowa aplikacja Delphi posiada natomiast zasób zawierający co najmniej jej ikonę — zasób ten tworzony jest automatycznie przy tworzeniu nowego projektu. Podobnie, kiedy wybierasz bitmapy dla przycisków `SpeedButton`, komponentów typu `Image` lub `BitBtn`, Delphi dołącza plik bitmapy jako część zasobów formularza. Formularz i zasoby są następnie dołączane do pliku programu w trakcie kompilowania i konsolidowania aplikacji. Większość z tych czynności jest wykonywana automatycznie.

Może się jednak zdarzyć, że będziesz chciał zaimplementować jakiś zasób niezależnie od automatycznych działań Delphi. Na przykład, aby stworzyć animację, będziesz musiał posiadać serię bitmap, które będzie można ładować jak najszybciej; musisz wówczas samodzielnie dołączyć je do pliku wykonywalnego.

Proces włączania pliku zasobów do pliku wykonywalnego jest bardzo prosty, w rzeczywistości o wiele trudniejsze okazuje się tworzenie samych zasobów. Tworzenie podstawowych zasobów, takich jak bitmapy, ikony, i kursory nie jest trudne, jeżeli dysponujesz dobrym edytorem zasobów, niemniej jednak tworzenie bitmap i kursorów 3D o profesjonalnym wyglądzie jest sztuką (ile razy zdarzyło Ci się spotkać w miarę przyzwoite programy z naprawdę okropnymi bitmapami przycisków?). Tworzenie ikon, bitmap i kursorów umożliwia jeden z programów narzędziowych Delphi — *Image Editor*. (Edytor ten jest omówiony w rozdziale 11. „Narzędzia i opcje Delphi").

Jeżeli masz zamiar utworzyć zasoby w postaci łańcuchów, danych użytkownika, plików dźwiękowych lub innych specjalistycznych zasobów, prawdopodobnie będziesz musiał skorzystać z zewnętrznego edytora.

Jeżeli posiadasz gdzieś kopię Borland Pascala, możesz skorzystać z jego edytora zasobów (Resource Workshop) umożliwiającego tworzenie zasobów bardziej zaawansowanych. Utworzone zasoby musisz następnie zapisać w pliku z rozszerzeniem *.rc*, który będziesz mógł skompilować do postaci pliku *.res* za pomocą kompilatora zasobów Borlanda (Borland Resource Compiler — *BRCC32.EXE*) wchodzącego w skład Delphi. Pliki .rc można oczywiście tworzyć za pomocą dowolnego edytora tekstu, w rzeczywistości jednak o wiele łatwiej jest skorzystać z edytora zasobów.

Kompilowanie plików zasobów

Po utworzeniu pliku zasobów, należy go skompilować. Możesz to zrobić na dwa różne sposoby:

- Ręcznie skompilować plik zasobów, bezpośrednio z wiersza poleceń.
- Dodać plik wsadowy do grupy projektów.

Każdy z tych sposobów daje w efekcie plik z rozszerzeniem *.res*, który następnie powinien być włączony do aplikacji w procesie konsolidacji — będzie o tym mowa już za chwilę (grupy projektów będą omawiane szczegółowo w następnym rozdziale).

Kompilowanie plików zasobów z poziomu wiersza poleceń

Aby skompilować plik zasobów z poziomu wiersza poleceń, wystarczy otworzyć okno wiersza poleceń w Windows i wpisać linię tekstu podobną do następującej:

```
brcc32 jjres.rc
```

Oczywiście polecenie takie ma sens przy założeniu, że katalog zawierający plik *BRCC32.EXE* znajduje się w systemowej ścieżce dostępu; jeśli tak nie jest, musisz w linii poleceń podać pełną ścieżkę dostępu.

Korzystanie z projektu pliku wsadowego

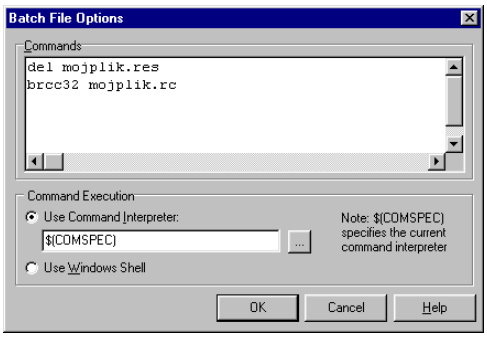
Dodanie projektu pliku wsadowego do grupy projektów jest tak proste, jak kompilowanie z poziomu wiersza poleceń, a ponadto posiada dodatkową zaletę — mianowicie daje gwarancję, że plik zasobów jest zawsze aktualny. Aby zobaczyć, w jaki sposób funkcjonują projekty plików wsadowych, wykonaj poniższe kroki:

1. Stwórz nową aplikację.
2. Wybierz polecenie menu *View | Project Manager*, aby otworzyć menedżer projektów.
3. Kliknij przycisk *New*, widoczny na pasku narzędzi menedżera projektów. Wyświetlone zostanie repozytorium.
4. Kliknij dwukrotnie ikonę o nazwie `Batch File` (plik wsadowy), aby utworzyć nowy projekt pliku wsadowego. Projekt pliku wsadowego zostanie dodany do menedżera projektów pod nazwą `Project2`.
5. Kliknij prawym przyciskiem myszy węzeł pliku wsadowego i wybierz polecenie zapisania (*Save*). Zapisz plik pod nazwą *test.bat*.
6. Ponownie kliknij prawym przyciskiem myszy węzeł pliku wsadowego i wybierz polecenie *Edit/Options*. Wyświetlone zostanie okno dialogowe opcji pliku wsadowego (*Batch File Options*).
7. Wpisz następujący tekst w obszarze edycji okna:

```
del mojplik.res
brcc32 mojplik.rc
```

Rysunek 8.13 przedstawia okno dialogowe opcji pliku wsadowego po wykonaniu tej operacji.

Rysunek 8.13. *Okno dialogowe opcji pliku wsadowego*



8. Kliknij przycisk *OK*, aby zamknąć okno.

Przed chwilą stworzyłeś plik wsadowy, który będzie uruchamiany za każdym razem, gdy będzie kompilowana grupa projektów. Polecenia wpisane w punkcie siódmym usuwają plik o nazwie *mojplik.res* i wywołują kompilator zasobów Delphi w celu skompilowania pliku *mojplik.rc*, czego wynikiem jest plik *mojplik.res* — z którego przypuszczalnie korzysta następny projekt w grupie.

Być może zastanawiasz się, dlaczego w ogóle usuwany jest plik *mojplik.res*. Otóż, usuwając plik wynikowy, można być pewnym, że kompilator zasobów zbuduje go od nowa. W przypadku niepowodzenia plik wynikowy nie zostanie utworzony, a korzystające z niego następne projekty w grupie nie skompilują się — błędna sytuacja będzie dla użytkownika oczywistym faktem. Gdyby pozostawić „stary" plik wynikowy, kompilator w razie niepowodzenia nie podmieni go na nową wersję; w efekcie, aplikacja korzystać będzie ze starej wersji — co oczywiście też jest błędem, lecz łatwym do przeoczenia.

Opisaną koncepcję ilustruje jeden z przykładowych projektów dołączonych do książki.

Łączenie plików zasobów z modułem wykonywalnym

W rezultacie skompilowania pliku skryptowego zasobów powstaje plik wynikowy, który należy połączyć z wykonywalnym plikiem programu. Do tego celu służy dyrektywa kompilatora `$R`. Przykładowo, aby do programu włączyć zasoby przechowywane w pliku *mojplik.res*, należy dołączyć poniższą linię kodu do modułu głównego formularza (w jego górnej części):

```
{$R mojplik.res}
```

Spowoduje to automatyczne dołączenie — podczas konsolidacji — pliku zasobowego *mojplik.res* do modułu wykonywalnego.

Przykładowy program korzystający z zasobów

Wydruk 8.2 zawiera kod źródłowy głównego formularza programu o nazwie `Jumping Jack`. Program ten pokazuje prostą animację, której towarzyszą efekty dźwiękowe. Główny formularz zawiera tylko dwa przyciski — komponent typu `Image` i komponent typu `Label`. Program `Jumping Jack` przedstawia kilka aspektów wykorzystywania zasobów w aplikacjach Delphi. W szczególności pokazuje, w jaki sposób ładowana jest bitmapa pamiętana w postaci zasobu, jak ładowany jest i wyświetlany łańcuch, a także w jaki sposób następuje odtworzenie pliku audio będącego zasobem. Wydruk 8.3 przedstawia fragment pliku zasobów wykorzystany w programie `Jumping Jack`. Prześledź te programy, a ja potem omówię działanie samego programu.

Wydruk 8.2. *JmpJackU.pas*

```
unit JmpJackU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, MMSystem;

{$R JJRES.RES}
```

```
const
  IDS_UP = 101;
  IDS_DOWN = 102;

type
  TMainForm = class(TForm)
    Image: TImage;
    Label1: TLabel;
    Start: TButton;
    Stop: TButton;
    procedure FormCreate(Sender: TObject);
    procedure StartClick(Sender: TObject);
    procedure StopClick(Sender: TObject);
  private
    { Deklaracje prywatne }
    Done : Boolean;
    procedure DrawImage(var Name: string);
  public
    { Deklaracje publiczne }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Image.Picture.Bitmap.
    LoadFromResourceName(HInstance, 'ID_BITMAP1');
end;

procedure TMainForm.StartClick(Sender: TObject);
var
  S       : string;
  ResName : string;
  I       : Integer;
  Buff    : array [0..9] of Char;
begin

  { Po kliknięciu przycisku Start, animacja           }
  { rozpoczyna swoją pracę. Nazwy zasobów bitmap      }
  { noszą nazwy od ID_BITMAP1 do ID_BITMAP5,          }
  { dlatego na początku zaczniemy od łańcucha         }
  { 'ID_BITMAP', do którego, w miarę potrzeby         }
  { dodawane będą kolejne cyfry.                      }
  S := 'ID_BITMAP';
  { Znacznik informujący o zakończeniu animacji. }
  Done := False;

  { Rozpoczęcie pętli i kontynuowanie jej }
  { aż do momentu naciśnięcia przycisku 'Stop'.    }

  while not Done do begin
    { Pętla wykonywana kolejno dla wszystkich nazw bitmap }
    { od 1 do 5.}
```

```
    for I := 1 to 5 do begin
      { Dołączenie wartości "I" do końca łańcucha. }
      { aby skonstruować łańcuch będący nazwą zasobu.       }
      ResName := S + IntToStr(I);
      { Wywołanie metody klasy w celu     }
      { wyświetlenia bitmapy.                       }
      DrawImage(ResName);
    end;
    { Załadowanie zasobu łańcucha o wartości 'Up' za pomocą          }
    { funkcji WinAPI - LoadString, wyświetlenie łańcucha             }
    { i przekazanie systemowi Windows polecenia ponownego
    { narysowania etykiety.                                      }
    LoadString(HInstance, IDS_UP, Buff, SizeOf(Buff));
    Label1.Caption := Buff;
    Label1.Refresh;
    { Odtworzenie dźwięku 'up' przy użyciu funkcji Windows API       }
    { PlaySound. Asynchroniczne odtworzenie dźwięku.                 }
    PlaySound('ID_WAVEUP',
      HInstance, SND_ASYNC or SND_RESOURCE);
    { Zatrzymanie animacji na chwilę w momencie wyskoku. }
    Sleep(200);
    { Powtórzenie całego procesu, ale w odwrotnym kierunku.}
    for I := 5 downto 1 do begin
      ResName := S + IntToStr(I);
      DrawImage(ResName);
    end;
    PlaySound('ID_WAVEDOWN',
      HInstance, SND_ASYNC or SND_RESOURCE);
    LoadString(HInstance, IDS_DOWN, Buff, SizeOf(Buff));
    Label1.Caption := Buff;
    Label1.Refresh;
    Sleep(200);
  end;
end;

procedure TMainForm.StopClick(Sender: TObject);
begin
  { Naciśnięto przycisk Stop, przekazanie polecenia }
  { zakończenia pętli. }
  Done := True;
end;

procedure TMainForm.DrawImage(var Name : string);
begin
  { Załadowanie bitmapy na podstawie }
  { nazwy przekazanej w nagłówku. }
  Image.Picture.Bitmap.
    LoadFromResourceName(HInstance, name);
  { Pchnięcie pętli komunikatów, aby Windows }
  { miał szansę wyświetlić bitmapę.                          }
   Application.ProcessMessages;

  { Chwilowe uśpienie procesu, aby animacja }
  { nie odtwarzała się zbyt szybko. }
  Sleep(20);
end;

end.
```

Wydruk 8.3. *JJRes.rc*

```
STRINGTABLE
BEGIN
 101, "W górę"
 102, "W dół"
END

ID_WAVEUP   WAVE "up.wav"
ID_WAVEDOWN WAVE "down.wav"

ID_BITMAP1 BITMAP LOADONCALL MOVEABLE DISCARDABLE IMPURE
BEGIN
 '42 4D 76 02 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 20 00 00 00 20 00 00 00 01 00 04 00 00 00'
 '00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00'

        …tutaj następuje dalsza część zasobów…
```

W deklaracji klasy głównego formularza znajduje się pole `Done` typu `Boolean` określające, kiedy należy zatrzymać animację. Metoda `DrawImage` służy do wyświetlania bitmapy przez komponent typu `Image`.

Zauważ, że na wydruku 8.2 wykorzystane zostały dwie funkcje API Windows: jedna z nich posłużyła do załadowania łańcucha, a druga do załadowania pliku dźwiękowego. We wnętrzu metody `StartClick` funkcja `LoadString` ładuje łańcuch do bufora tekstowego, na podstawie identyfikatora numerycznego zasobu (zajrzyj do wydruku 8.3, aby przekonać się, w jaki sposób tworzone są łańcuchy). Załadowany łańcuch jest następnie przypisywany właściwości `Caption` obiektu `Label`.

Funkcja `PlaySound` służy do odtworzenia pliku dźwiękowego pamiętanego w postaci zasobu. Znacznik `SND_ASYNC`, użyty jako parametr wywołania, nakazuje systemowi Windows rozpoczęcie odtwarzania dźwięku i bezzwłoczne zwrócenie sterowania do programu (odtwarzanie dźwięków jest szczegółowo omówione w rozdziale dwunastym „Programowanie grafiki i multimediów"). Dzięki temu animacja może być kontynuowana w trakcie odtwarzania dźwięku. Znacznik `SND_RESOURCE` informuje Windows o tym, że dźwięk jest zapisany w postaci zasobu, a nie jako plik dyskowy. Obydwie funkcje — `LoadString` i `PlaySound` — korzystają ze zmiennej globalnej `HInstance`, aby poinformować Windows, iż zasobów należy szukać we wnętrzu pliku wykonywalnego. Zasoby będące bitmapami są ładowane za pomocą metody VCL o nazwie `LoadFromResourceName`.

Pierwsze pięć linii wydruku 8.3 przedstawia budowę tablicy łańcuchów w pliku skryptowym zasobów (tablice łańcuchów można z łatwością tworzyć za pomocą dowolnego edytora tekstowego). Następnie tworzone są dwa elementy typu `WAVE`, dla każdego z plików dźwiękowych, które zostały nagrane wcześniej i umieszczone w katalogu projektu. Kiedy kompilator zasobów rozpozna deklarację typu `WAVE`, czyta każdy indywidualny plik dźwiękowy, po czym kompiluje go do postaci binarnego pliku zasobów.

Patrząc na wydruk 8.3, przekonałeś się, iż niektóre typy zasobów można z łatwością utworzyć za pomocą dowolnego edytora tekstowego. W przypadku bitmap i plików dźwiękowych, zapisanych w oddzielnych plikach, można dołączyć je do pliku *.rc*, który następnie zostanie przetworzony na postać binarną przez kompilator zasobów. Następnie binarny plik zasobów zostanie włączony do wykonywalnego pliku aplikacji.

Wydruk 8.3 przedstawia tylko fragment kodu. Bitmapy tworzone za pomocą tradycyjnego edytora zasobów są często pamiętane w pliku zasobów jako dane numeryczne. Opisy zasobów dla bitmap mogą być nieraz bardzo długie — opuszczona część opisu zasobów bitmap dla programu `Jumping Jack` wymaga około 200 linii kodu, dlatego opis ten nie został tutaj przedstawiony w całości. Rysunek 8.14 przedstawia program `Jumping Jack` w działaniu.

Rysunek 8.14.
Jumping Jack w akcji

Tworzenie dodatkowych zasobów dla aplikacji nie jest zagadnieniem skomplikowanym, ale nie jest to również zadanie banalne. Trochę czasu upłynie, zanim zrozumiesz, jak składać te wszystkie klocki w jedną całość. Być może okaże się, że nigdy nie będziesz musiał tworzyć dodatkowych zasobów dla swoich aplikacji — nie zaszkodzi jednak zaznajomić się z ideą całego tego mechanizmu.

Bitmapy, ikony i kursory występujące w innych programach są zwykle chronione prawem autorskim. Nie korzystaj więc — bez wyraźnego zezwolenia — z zasobów należących do innych programów. Ponadto zawsze zakładaj, iż programy są chronione, chyba że one same informują o swoim statusie prawnym jako *freeware*. W swoich aplikacjach możesz swobodnie stosować bitmapy, ikony i kursory dostarczone razem z Delphi (znajdziesz je w katalogu *Borland Shared\Images*).

Pakiety

Po zakończeniu prac nad aplikacją możesz rozpocząć jej dystrybucję na dwa różne sposoby (przez dystrybucję rozumiemy tu proces dostarczania produktu użytkownikom). Możesz rozprowadzać swoją aplikację wśród ogółu użytkowników lub ewentualnie wśród użytkowników należących do Twojej firmy. W obu przypadkach musisz określić strategię rozwoju programu. W szczególności powinieneś wybrać pomiędzy konsolidacją statyczną, a dynamiczną z wykorzystaniem pakietów. Opcje te zostaną omówione w niniejszym podrozdziale, abyś mógł w przyszłości podjąć świadomy wybór sposobu dystrybucji swojej aplikacji. Nasze rozważania zaczniemy od dyskusji na temat pakietów, po czym przejdziemy do opisu możliwości dystrybucyjnych.

Czym jest pakiet?

Zanim przejdziemy do szczegółów, należy zdefiniować pojęcie pakietu.

Pakiet jest fragmentem skompilowanego kodu, rezydującym w pliku o rozszerzeniu *.BPL*.

Prawdopodobnie definicja ta nie stanowi dla Ciebie dostatecznego wyjaśnienia, więc ją rozwińmy. Po bliższym przyjrzeniu się pakietowi, okazuje się on być zwykłą biblioteką DLL, noszącą zamiast rozszerzenia *.dll* rozszerzenie *.bpl*. (nie jest to dokładne wyjaśnienie, ale dla naszych celów jest ono wystarczające). W Delphi istnieją dwa typy pakietów: pakiety wykonywalne i pakiety projektowe.

Pakiety wykonywalne

Pakiet wykonywalny zawiera kod niezbędny do pracy aplikacji. Chociaż Delphi dostarcza wiele różnych pakietów, wśród nich znajduje się jeden podstawowy o nazwie *vcl60.bpl*. W pakiecie tym znajduje się zasadniczy kod biblioteki VCL. Jeżeli zdecydujesz się na wyodrębnienie pakietów ze swojej aplikacji, załaduje ona plik o nazwie *vcl60.bpl* i, w miarę potrzeby, będzie z niego wywoływać procedury. Jeżeli aplikacja będzie bazą danych, załaduje również pakiet *vcldb60.bpl*, aby korzystać z procedur tam zapisanych. Oprócz wspomnianych wyżej dwóch pakietów Delphi posiada również wiele innych.

Może się okazać, że budowana przez Ciebie aplikacja, oprócz pakietów VCL, wymaga również innych modułów. Dzieje się tak w sytuacji, gdy w aplikacji wykorzystywane są komponenty pochodzące spoza Delphi, tworzone przez niezależnych autorów (być może również przez Ciebie).

Pakiety środowiskowe

Pakiety środowiskowe mają ścisły związek z obsługą komponentów przez projektanta formularzy. Z większością komponentów tworzonych za pomocą Delphi związane są dwa pakiety — wykonywalny i środowiskowy. Pakiet wykonywalny zawiera cały kod niezbędny do funkcjonowania komponentu w uruchomionym programie, natomiast kod zawarty w pakiecie środowiskowym niezbędny jest do operowania komponentem na etapie projektowania, włączając w to (ewentualne) edytory właściwości i specjalizowany edytor komponentu.

Pakiet środowiskowy zawiera listę *Requires*, zawierającą nazwy innych pakietów niezbędnych do jego funkcjonowania. Pakiet środowiskowy prawie zawsze wymaga kodu pochodzącego z pakietu wykonywalnego, a często również kodu pochodzącego z jednego lub wielu pakietów VCL. Należy sobie zdawać sprawę z faktu, iż jeden pakiet (zarówno wykonywalny, jak i środowiskowy) może zawierać kod dla kilku komponentów — nie jest wymagane istnienie oddzielnego pakietu dla każdego z komponentów.

Ponieważ pakiety środowiskowe zawierają tylko kod niezbędny do funkcjonowania komponentu na etapie projektowania, są one zazwyczaj o wiele mniejsze niż ich wykonywalne odpowiedniki.

Konsolidacja statyczna kontra konsolidacja dynamiczna

Skoro posiadasz już pewną wiedzę na temat pakietów, możesz dowiedzieć się trochę na temat dwóch możliwych trybów konsolidowania.

Konsolidowanie statyczne

W przypadku statycznej konsolidacji aplikacji pakiety nie są w ogóle wykorzystywane. Cały kod niezbędny do prawidłowej pracy aplikacji jest dołączany bezpośrednio do jej pliku wykonywalnego. Aplikacja w tym przypadku jest standardowym programem i nie potrzebuje żadnych plików wspomagających (pakietów czy bibliotek DLL).

Od tej reguły istnieje jednak wyjątek. Stwierdzenie, że statycznie skonsolidowana aplikacja nie wymaga żadnych dodatkowych bibliotek DLL opiera się na dwóch założeniach. Po pierwsze — aplikacja nie jest bazą danych. Aplikacja bazy danych stworzona przez Delphi wymaga do pracy mechanizmu o nazwie Borland Database Engine (BDE). BDE jest zbiorem bibliotek DLL, dlatego aplikacja bazy danych będzie ich wymagała, mimo iż została skonsolidowana statycznie. Drugie założenie opiera się na stwierdzeniu, że aplikacja nie korzysta z żadnych kontrolek ActiveX. Kontrolki ActiveX w rzeczywistości są pewną formą bibliotek DLL, dlatego aplikacja, która z nich korzysta, nie jest aplikacją samodzielną.

Chociaż Delphi daje możliwość wyboru opcji konsolidowania, domyślnie wykonywana jest konsolidacja statyczna. Posiada ona dwie cechy dające jej przewagę nad konsolidacją dynamiczną:

- Nie musisz martwić się o dołączanie jakichkolwiek dodatkowych plików do aplikacji. Sama aplikacja zawiera cały niezbędny kod i nie wymaga dodatkowych bibliotek.
- Ogólnie rzecz biorąc, całkowity rozmiar aplikacji skonsolidowanej statycznie jest mniejszy od sumarycznego rozmiaru aplikacji wymagającej pakietów. Wspomnę o tym za chwilę, kiedy będą omawiane zalety i wady konsolidacji dynamicznej.

Konsolidację statyczną cechuje jedna podstawowa wada, ujawniająca się jedynie w przypadku aplikacji korzystających z bibliotek DLL zdefiniowanych przez użytkownika. Jest nią duplikowanie się kodu VCL i RTL w każdym module — w głównym module wykonywalnym i w każdej bibliotece DLL. Oznacza to niepotrzebne powielanie kodu.

Załóżmy na przykład, że każdy moduł wymaga minimum 200 kB podstawowego kodu VCL i RTL. Dodatkowo załóżmy, że posiadasz aplikację główną i dziesięć wspierających

ją bibliotek DLL. Wynika stąd, że 2200 kB kodu wykonuje czynności, które mogłyby być wykonane przez 200 kB. Jedenastokrotne powielenie kodu jest jednak konieczne, ponieważ statycznie konsolidowane moduły nie mogą korzystać wspólnie z jednego fragmentu kodu VCL I RTL.

Konsolidowanie dynamiczne

Konsolidowanie dynamiczne odnosi się do scenariusza, w którym aplikacja dynamicznie ładuje kod bibliotek w trakcie swojej pracy. W przypadku aplikacji stworzonej w Delphi oznacza to, że każdy wymagany pakiet jest ładowany w trakcie jej pracy. W skład wymaganych pakietów z całą pewnością wejdzie przynajmniej jeden pakiet VCL, a być może również dodatkowe pakiety zewnętrzne.

Ładowanie pakietów przez aplikację odbywa się automatycznie i nie wymaga pisania dodatkowego kodu. Zmiana konsolidowania aplikacji — ze statycznej na dynamiczną lub odwrotnie — również nie wymaga ingerencji w kod źródłowy. Zmienia się natomiast sposób dystrybucji aplikacji — o czym będzie mowa za chwilę.

Konsolidowanie dynamiczne ma tę podstawową przewagę nad konsolidowaniem statycznym, iż kilka modułów jest w stanie korzystać wspólnie z pojedynczego fragmentu kodu (pakietu). Pamiętasz przytoczony przed chwilą przykład aplikacji i dziesięciu wspierających ją bibliotek DLL? W przypadku konsolidacji dynamicznej aplikacja i wszystkie biblioteki DLL mogą wspólnie korzystać z kodu zawartego w pakiecie VCL. Każdy z tych modułów będzie mniejszy o około 200 kB, ponieważ podstawowy kod będzie zawarty w bibliotekach DLL biorących udział w pracy aplikacji. Zaleta ta staje się wyraźnie odczuwalna w aplikacjach składających się z kilkudziesięciu bibliotek DLL.

Z konsolidacją dynamiczną łączy się jednak kilka problemów. Po pierwsze — aplikacja skonsolidowana z podziałem na pakiety wymaga do swej pracy obecności pakietu *vcl60.bpl* o rozmiarze ok. 1,3 MB, co niekiedy może zniwelować oszczędności wynikające ze współdzielenia kodu. Drugi problem jest znacznie poważniejszy i dotyczy, ogólnie rzecz biorąc, stosowania różnych wersji oprogramowania. Zobrazuję to zagadnienie na przykładowym scenariuszu. Powiedzmy, że tworzysz aplikację wykorzystującą pakiet Delphi 6.02 (zakładamy hipotetycznie istnienie kilku wersji Delphi 6) i korzystasz z konsolidacji dynamicznej, która wymaga od Ciebie dołączenia do programu pakietów VCL i RTL. Klient, któremu sprzedałeś aplikację, instaluje oprogramowanie na swoim komputerze i wszystko działa poprawnie.

W tym czasie, ja buduję aplikację, korzystając z pakietu Delphi 6.0 (powiedzmy, że jestem zbyt skąpy, aby zapłacić za przesłanie uaktualnienia), i również stosuję konsolidację dynamiczną. Twój klient kupuje i instaluje moją aplikację. Mój program instalacyjny jest nieprofesjonalny i działa niezgodnie z przyjętymi zasadami, powodując zastąpienie moimi pakietami i bibliotekami tych pakietów i bibliotek, które zainstalowała Twoja aplikacja. W efekcie Twoja aplikacja odmawia pracy, niezdolna do wykorzystania nieodpowiednich (starych) modułów.

Mając to na uwadze, firmy produkujące oprogramowanie komercyjne (takie, jak np. Borland), zapobiegają powstawaniu takich problemów i stosują inne nazwy pakietów i bibliotek DLL w każdej kolejnej wersji produktu, a także dołączają do bibliotek i pakietów informacje o ich wersji. Dobry program instalacyjny na początku sprawdza numer wersji i instaluje nowe pakiety tylko wtedy, gdy te istniejące w systemie użytkownika są starsze od instalowanych. Nie można jednak zakładać, że wszyscy autorzy komponentów VCL są autorami odpowiedzialnymi, zwłaszcza wobec szerokiej dostępności oprogramowania przez Internet; należy więc zachować ostrożność przy instalowaniu (niedrogich lub darmowych) komponentów pochodzących z nieznanych źródeł.

Który sposób jest lepszy?

Już słyszę jak się zastanawiasz: „Co więc powinienem stosować — konsolidację statyczną, czy też dynamiczną?". Odpowiedź na to pytanie zależy od programu, jaki tworzysz. Generalnie, jeżeli budujesz pojedynczą aplikację małego lub średniego rozmiaru, powinieneś zastosować konsolidację statyczną. W przypadku, gdy tworzysz bardzo obszerne aplikacje lub aplikacje z dużą liczbą bibliotek DLL, korzystniejsze może się okazać zastosowanie konsolidacji dynamicznej.

Rozpatrzenie prostego przykładu może pomóc w lepszym zrozumieniu tego zagadnienia. W rozdziale szóstym stworzyłeś program `Scratch`. Program ten, po skompilowaniu i statycznym skonsolidowaniu przyjmuje rozmiar (w przybliżeniu) 460 kB. Jeżeli skonsolidujesz go z rozbiciem na pakiety, otrzymasz plik wykonywalny (*.exe*) o rozmiarze 25 kB, ale do programu musisz dołączyć pakiet *vcl60.bpl*, który ma ok. 1,3 MB. Jak widzisz, konsolidacja dynamiczna okazała się w tym przypadku nieefektywna.

Stosowanie pakietów wykonywalnych we własnych aplikacjach

Dynamiczna konsolidacja wymaga zmiany tylko jednego ustawienia w opcjach projektu — postępuj według poniższych kroków:

1. Aby otworzyć okno dialogowe opcji projektu, wybierz polecenie menu *Project | Options*. Wybierz zakładkę *Packages* i kliknij opcję *Build with runtime packages*, widoczną u dołu okna (górną część okna, dotyczącą pakietów środowiskowych, możesz zignorować).
2. Kliknij przycisk *OK*, aby zamknąć okno opcji projektu.
3. Ponownie zbuduj projekt (za pomocą opcji *Project|Build...*).

To wszystko. Pamiętaj, że konsolidacja dynamiczna nie wymaga żadnych zmian w kodzie programu.

Dystrybucja aplikacji z wykorzystaniem pakietów

Dystrybucja aplikacji skonsolidowanej dynamicznie wymaga od Ciebie rozeznania co do wykorzystywanych przez nią pakietów. Jeżeli dokładnie prześledziłeś poprzednią sekcję, możesz być pewny, że zawsze będziesz potrzebował (co najmniej) pliku *vcl60.bpl*. W zależności od komponentów, jakie zastosowałeś w swojej aplikacji, niezbędne mogą się okazać również inne pakiety.

Żeby uzyskać stuprocentową pewność co do wymaganych pakietów, możesz skorzystać z programu *TDUMP.EXE*, aby sprawdzić, do jakich modułów odwołuje się plik wykonywalny (program ten znajdziesz w podkatalogu *Bin*). Aby uruchomić program *TDUMP*, wywołaj okno wiersza poleceń i przejdź do katalogu, w którym znajduje się Twoja aplikacja. Następnie (zakładając, że w systemowej ścieżce dostępu umieszczony został podkatalog *Bin*) wpisz poniższe polecenie:

```
tdump scratch.exe
```

Przygotuj się do naciśnięcia przycisku *Pause*, ponieważ *TDUMP* produkuje informacje w bardzo szybkim tempie. Gdzieś pośród generowanych informacji powinieneś zobaczyć linie podobne do przedstawionych poniżej:

```
Imports from vcl60.bpl
                __fastcall Forms::initialization()
                __fastcall Forms::Finalization()
                __fastcall Forms::TApplication::MessageBox(const char *, const
                    char *, int)
                __fastcall Forms::TApplication::Run()
                __fastcall Forms::TApplication::CreateForm(System::TMetaClass *,
                    void *)
                __fastcall Forms::TApplication::Initialize()
                __fastcall Forms::TApplication::SetTitle(const System::AnsiString)
                __stdcall Forms::TCustomForm::QueryInterface(const _GUID&, void *)
                __fastcall Forms::TCustomForm::UpdateActions()
                __fastcall Forms::TCustomForm::ShowModal()
                __fastcall Forms::TCustomForm::SetFocus()
                __fastcall Forms::TCustomForm::CloseQuery()

....
```

Zestaw taki może powtórzyć się kilka razy. Będziesz musiał obserwować ekran w poszukiwaniu plików z rozszerzeniem *.BPL* i notować ich nazwy. Kiedy skończysz, będziesz miał kompletną listę pakietów, które muszą towarzyszyć Twojej aplikacji.

Aby ułatwić sobie przeglądanie danych generowanych przez program TDUMP, można przekierować je do pliku tekstowego:

```
Tdump mojprojekt.exe > dump.txt
```

Teraz możesz otworzyć plik *DUMP.TXT* w edytorze kodu Delphi i przejrzeć jego zawartość.

Możesz zaoszczędzić sobie dużo czasu i kłopotów, sprowadzając dobry program instalacyjny. Program, który mógłbyś wykorzystać — *InstallShield Express* — jest rozprowadzany z dwiema wersjami Delphi 6: Professional i Enterprise. Równie dobry jest produkt firmy Great Lakes Business Solutions o nazwie *Wise Install*. Dobre programy instalacyjne rozpoznają, jakich pakietów wymaga aplikacja i automatycznie uwzględniają je w procesie instalacji. Niezależnie od istniejącej sytuacji nie polecam pisania własnych programów instalacyjnych — ze względu na mnogość czynników, które należy wówczas uwzględnić.

Warsztat

Warsztat składa się z pytań kontrolnych oraz ćwiczeń utrwalających i pogłębiających zdobytą wiedzę. Odpowiedzi na pytania quizu możesz znaleźć w dodatku A.

Pytania i odpowiedzi

P W jakiej sytuacji powinienem stosować opcję Use, dostępną w repozytorium?

O Opcję tę powinieneś stosować, jeżeli zajdzie potrzeba uaktualnienia obiektu zapisanego wcześniej w repozytorium.

P Czy istnieje ograniczenie liczby obiektów, które można umieścić w repozytorium?

O Technicznie takie ograniczenie nie istnieje. Pamiętaj jednak, że przeznaczeniem repozytorium jest pomoc w lokalizowaniu i ponownym wykorzystaniu formularzy, okien dialogowych i innych obiektów. Jeżeli w repozytorium umieścisz zbyt dużo rzadko używanych obiektów, jego zastosowanie stanie się nieefektywne, ponieważ odnalezienie wymaganego komponentu będzie zabierało zbyt dużo czasu. Spadnie również szybkość, z jaką repozytorium będzie ładować i wyświetlać wszystkie umieszczone w nim obiekty.

P W repozytorium znajduje się grupa obiektów, których od dawna nie używam. W jaki sposób mogę się ich pozbyć?

O Wybierz polecenie menu *Tools | Repository*. Wyświetlone zostanie okno dialogowe konfiguracji repozytorium. Aby usunąć obiekt, wybierz go z listy obiektów, a następnie kliknij przycisk *Delete Object*.

P Zapisałem obiekt w repozytorium. Teraz, kiedy próbuję z niego skorzystać, wyświetlone zostaje okno z następującym komunikatem: `Unable to find both a form and a source file` (Nie można znaleźć pliku formularza i pliku kodu źródłowego). W czym tkwi problem?

O Musiałeś wcześniej usunąć lub przemieścić plik kodu źródłowego lub formularza określonego obiektu. Repozytorium przechowuje dane dotyczące katalogu, w którym dany obiekt został zapisany. Po przesunięciu lub usunięciu obiektu repozytorium nie jest w stanie go znaleźć i z tego powodu zgłasza błąd.

P Czy mogę dodawać obiekty do strony *New repozytorium*?

O Nie, zawartość strony *New* jest z góry ustalona i nie można jej usunąć ani zmodyfikować. Swoje obiekty musisz umieszczać na innych stronach.

P Dodałem metodę do klasy głównego formularza i od tego momentu proces kompilacji kończy się błędem. W czym tkwi problem?

O Prawdopodobnie przez przypadek dodałeś deklarację metody do sekcji zarządzanej przez Delphi. Musisz upewnić się, iż deklaracja metody, którą dodałeś, znajduje się w sekcji `public`, `private` lub `protected`.

P Dysponuję edytorem zasobów, dzięki któremu jestem w stanie dekompilować zasoby zapisane w innych programach. To z kolei pozwala mi „pożyczać" bitmapy i inne zasoby z tych programów. Czy to jest w porządku?

O Krótka odpowiedź brzmi: „Nie" — zawsze należy z góry zakładać, że materiały należące do innych programów są prawnie chronione i nie mogą być swobodnie wykorzystywane. Więcej informacji na ten temat uzyskasz, gdy skonsultujesz się z prawnikiem.

P Moja aplikacja wymaga wielu plików dźwiękowych i bitmap. Czy można umieścić je wszystkie w pliku nie będącym samym plikiem wykonywalnym aplikacji?

O Tak. Zasoby można zapisać w bibliotece dołączanej dynamicznie (DLL).

Quiz

1. W jakiej sytuacji należy korzystać z opcji `Inherit` w repozytorium?
2. W jaki sposób przebiega procedura zapisywania projektu w repozytorium?
3. Co dzieje się z formularzami potomnymi po wprowadzeniu zmian do formularza podstawowego?
4. Gdzie w deklaracji klasy głównego formularza można umieszczać własne metody?
5. Gdzie należy umieścić definicję metody, gdy dodaje się ją do kodu Delphi?
6. W jaki sposób — w repozytorium — można zidentyfikować autora danego obiektu?
7. W jaki sposób można usunąć istniejące lub dodać nowe strony do repozytorium?

8. Czy łatwiej jest stworzyć aplikację „od zera", czy też z pomocą kreatora aplikacji?
9. Która opcja jest lepsza w przypadku małej aplikacji: konsolidacja statyczna czy konsolidacja dynamiczna z użyciem pakietów?
10. Czy przy użyciu edytora tekstowego jesteś w stanie utworzyć plik skryptowy zasobów, zawierający tablicę łańcuchów?

Ćwiczenia

1. Stwórz nowy formularz. Dodaj do niego kilka komponentów według własnego uznania. Zapisz formularz na stronie `Forms` w repozytorium, nadając mu nazwę `FormularzPodstawowy`.
2. Zainicjuj nową aplikację. Wybierz polecenie *File*|*New*|*Other*, aby wyświetlić repozytorium. Przejdź do strony `Forms`. Kliknij przycisk opcji `Inherit`. Wybierz obiekt `FormularzPodstawowy`, który utworzyłeś w ćwiczeniu pierwszym, i dodaj go do aplikacji. (Upewnij się, że wybrałeś opcję `Inherit`.) Zapisz projekt i zamknij go.
3. Otwórz obiekt `FormularzPodstawowy` stworzony w ćwiczeniu pierwszym. Usuń wszystkie komponenty występujące w formularzu, a następnie zapisz go.
4. Ponownie otwórz projekt utworzony w ćwiczeniu drugim. Wyświetl formularz utworzony wcześniej. Zauważ, że wszystkie komponenty widoczne poprzednio w formularzu przestały istnieć (stało się tak, ponieważ obiekt ten dziedziczy właściwości po obiekcie podstawowym, który uległ zmianie.)
5. Z głównego menu wybierz polecenie *Tools*|*Repository*. Usuń utworzony niedawno obiekt `FormularzPodstawowy`.
6. Utwórz projekt za pomocą kreatora aplikacji. Użyj wszystkich dostępnych opcji menu oraz utwórz aplikację jako aplikację typu MDI.
7. Do aplikacji, utworzonej w ćwiczeniu szóstym, dodaj okno dialogowe z wieloma zakładkami. Skorzystaj z kreatora dialogów.
8. Korzystając z repozytorium, dodaj okno informacji o programie (*About*) do aplikacji, którą stworzyłeś w ćwiczeniu szóstym.
9. Utwórz prosty program, a następnie go skompiluj. Uruchom Eksploratora Windows i sprawdź, jakiego rozmiaru jest plik *.EXE* utworzony przez Delphi. Teraz ustaw w projekcie opcję wykorzystania pakietów wykonywalnych. Ponownie zbuduj program i sprawdź rozmiar powstałego pliku *.EXE*. Jaka różnica występuje w rozmiarach obu plików?