Wykład II – wykorzystanie w programie wielu okien (form)

Najprostszy program składa się z pojedynczego okna, w środowiskach programistycznych określanych często mianem *formatki* lub *formy*. W miarę rozbudowy programu konieczne często staje się wzbogacenie go o dodatkowe okna realizujące różnorakie funkcje. Zdarza się, że w programie pojawia się ostatecznie kilkadziesiąt okien – niektóre proste, z małą ilością umieszczonych na nich komponentów, inne skomplikowane. Każde okno, nawet to najprostsze, to twór złożony, pożerający zasoby systemu operacyjnego (i nie o samą pamięć komputera tu chodzi, ale o specjalne *zasoby systemu*, których nie ma aż tyle).
Jak powinno być czytelnikowi tego kursu wiadome – okno w systemie Windows powstaje w skomplikowany, wieloetapowy sposób. Najpierw tworzona jest klasa okna – wypełniany jest swoisty *formularz* definiujący zachowanie się i podstawowy wygląd okna. Z zachowaniem okna związane jest istnienie (utworzenie) tzw. *funkcji okienkowej* – specjalnej funkcji o precyzyjnie określonej budowie, do której przekazywany jest każdy komunikat systemu operacyjnego wymagający reakcji od strony okna. Funkcja okienkowa realizuje między innymi reakcję na zdarzenia formy i wszystkich komponentów na niej umieszczonych. Klasa ta musi zostać zarejestrowana w systemie operacyjnym, po czym można już utworzyć egzemplarz danego okna i wyświetlić go. Temat złożoności okna poruszony został, aby uzmysłowić stopień złożoności procesu powstawania okna i podkreślić obciążenie dla systemu, jakim jest okno. Nie należy więc mnożyć okien w nieskończoność, szafować nimi należy rozważnie, a zwłaszcza powstrzymać należy się od automatycznego tworzenia przy starcie programu wszystkich okien – tych potrzebnych *od zaraz*, jak i tych, których użytkownik pewnie na oczy nie zobaczy. W poprawnie zaprojektowanej aplikacji okna tworzone są i wyświetlane w miarę potrzeb.
Aby w aplikacji mogło zaistnieć dodatkowe okno, należy zdefiniować jego klasę (wywodzącą się z TForm) i utworzyć odpowiedni plik *.dfm skojarzony z unitem, w którym definiowana jest ta klasa. Obie te operacje najlepiej wykonać wybierając z menu glównego pozycję *File -> New -> Form*. W ten sposób dodane okno będzie tworzone automatycznie przy starcie programu ale początkowo niewidoczne. Uczynić je widocznym (automatycznie od momentu utworzenia) można ustawiając mu właściwość *Visible* na *true*.

Podkreślić należy, że automatyczne tworzenie i wyświetlanie dodatkowych (poza głównym) okien w programie to powinna być rzadkość.

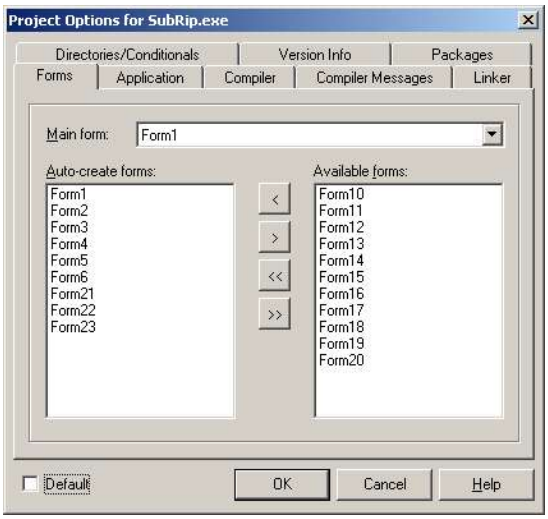
Dynamiczne tworzenie okna

Proces dynamicznego tworzenia okna przebiega dokładnie tak samo, jak dynamicznego tworzenia komponentu (patrz *Wykład I*) – nic dziwnego – i jedno i drugie to obiekty i do tego wywodzące sięod klasy *TComponent*.
Co prawda formy to nie klasyczne komponenty – nie można ich postawić na innej formie (no może nie w klasyczny sposób), ale ponieważ dziedziczą z *TComponent*, więc również mogą posiadać właściciela i rodzica.
Właścicielem komponentu zwykle była forma, na której był umieszczony. W przypadku formy musi być inaczej – jej właścicielem zwykle jest specjalny, tworzony automatycznie przy starcie aplikacji obiekt Application (klasy *TApplication*). Forma może nie mieć właściciela – na programiście spoczywa wówczas obowiązek jej zniszczenia.
Rodzic formy, to większy problem. Zwykle go nie ma. Jeśli się pojawia, to zwykle jest to sytuacja wyjątkowa – sztuczne położenie formy na innej formie.
Utworzyć możemy jedynie formę zdefiniowaną w aplikacji (w jakimś unicie znajduję się definicja jej klasy). Pamiętać należy o umieszczeniu odpowiedniego odniesienia do unitu z

definicją klasy formy w klauzuli *uses*. Środowisko programistyczne zaproponuje co prawda automatyczne zrobienie tej czynności po wykryciu odwołania do klasy formy, ale dobrym nawykiem jest ręczne zrobienie tego (lub poprzez wybranie pozycji *File -> Use Unit* z menu głównego).
Przykładowe ręczne tworzenie formy może więc wyglądać następująco:

```
NowaForma:=TForm2.Create(nil);          // bez właściciela – programista tworzy i niszczy własnoręcznie
```

Przed własnoręcznym, dynamicznym tworzeniem formy należy zadbać o to, aby nie była ona niepotrzebnie tworzona automatycznie przez uruchamiający się program. Zrobić to można na dwa sposoby. Oba działają tak samo i ściśle wiążą się ze sobą.
Pierwszy z nich polega na wejściu do opcji projektu (*Project -> Options* w menu głównym). Tam na zakładce Forms (rys. 1) przenieść formy tworzone tylko dynamicznie do sekcji *Available forms*.



Rys. 1. Opcje projektu – zarządzanie oknami projektu.

Drugim sposobem jest otwarcie głównego pliku projektu – z rozszerzeniem *dpr*, i usunięciu z niego jednej z linijek o postaci:

```
Application.CreateForm(TForm2, Form2);
```

Metody te mają skutek wzajemny – użycie jednej wpływa na drugą w odpowiedni sposób (przeniesienie formy w oknie projektu usuwa linijkę z pliku projektu i na odwrót).

Jeśli okno tworzone jest tylko dynamicznie, to warto również usunąć globalną zmienną o nazwie odpowiadającej nazwie klasy formy, która zadeklarowana jest automatycznie w unicie, w którym jest zdefiniowana klasa formy (zwykle bezpośrednio pod definicją klasy, w postaci var Form2:TForm2).

Główne okno programu

Pierwsze okno zdefiniowane w programie staje się automatycznie jego głównym oknem. Główne okno programu posiada jedną cechę odróżniającą je od innych okien – jego zamknięcie zamyka aplikację. W przypadku istnienia większej liczby okien w programie można w dowolnym momencie projektowania aplikacji zmienić okno główne. Można to

zrobić też na dwa sposoby – we wspomnianym oknie ustawień projektu – na zakładce z formami (combo *Main Form*) lub zmieniając kolejność linii w głównym pliku projektu (pamiętając o tym, że główne okno projektu, to po prostu to tworzone na początku).
Fakt, że główne okno tworzone jest pierwsze, jest często ignorowany przez programistów, umieszczających w jego zdarzeniu *OnCreate* odwołania do innych okien. Jest to oczywisty błąd w przypadku okien tworzonych automatycznie – one jeszcze wtedy nie istnieją. Postępować tak można tylko z oknami tworzonymi dynamicznie.
Przy zamykaniu aplikacji jest na odwrót – ostatnie zamykane i niszczone jest okno główne. Zamknięcie okna głównego – zamyka aplikację.

Dynamiczne wyświetlanie i zamykanie okna – formy modalne i niemodalne

Każde okno, nie tylko to tworzone dynamicznie, można wyświetlać *na żądanie*, a nie tylko w momencie startu aplikacji.
Takie dynamiczne wyświetlanie daje większe możliwości – pozwala wyświetlić okno na dwa sposoby – niemodalny i modalny. Pierwszy sposób daje efekt taki sam, jak ustawienie właściwości *Visible* na *true* – w programie pojawia się okno równorzędne pod każdym względem poza efektem zamykania z głównym oknem programu. Drugi sposób (modalny) daje możliwość przejęcia przez okno kontroli nad aplikacją – będzie aktywne tylko okno wyświetlone modalne.

Podkreślić należy, że choć potocznie używa się określenia okno modalne i niemodalne, to jednak okna są takie same – zmienia się tylko sposób ich wyświetlania.

Wyświetlanie niemodalne odbywa się poprzez wywołanie metody ***Show*** okna, które chcemy wyświetlić. Jest to **procedura**, która wyświetli okno, ustawi jego właściwość *Visible* na *true* (można korzystać z tej właściwości chcąc dowiedzieć się, czy okno jest widoczne w danym momencie na ekranie) i natychmiast skończy swoje działanie. Zamknąć tak wyświetlone okno można wywołując jego metodę ***Close***.

Zamknięcie okna nie powoduje jego zniszczenia. Zniszczenie, tak jak w przypadku innych obiektów powoduje metoda Free.

Wyświetlanie modalne odbywa się poprzez wywołanie metody ***ShowModal*** okna, które chcemy wyświetlić. Jest to dla odmiany funkcja, która wyświetli modalnie okno, ustawi jego właściwość *Visible* na *true*, ale swoje działanie i wynik zwróci dopiero po zamknięciu wyświetlonego nią okna. Wynik zwracany przez tą funkcję, to wartość właściwości **ModalResult** wyświetlanego okna. W połączeniu z dodatkowym sposobem zamykania okna wyświetlanego modalnie (o tym za moment) pozwala on na proste powiadomienie programu o sposobie (a właściwie powodzie) zamknięcie wyświetlonego modalnie okna i podjęcia przez program adekwatnych działań (rekacji np. na zamknięcie okna modalnego poprzez wciśnięcie przycisku *Anuluj* lub *Zatwierdź*). Okno modalne zamknięte może zostać tak jak zwykłe okno myszką (kliknięcie na **X**) lub skrótem klawiszowym, może być zamknięte poleceniem Close (zwraca wówczas wynik 2 mający interpretację jako *Cancel*), może być też zamknięte w dodatkowy sposób – poprzez ustawienie jego właściwości *ModalResult* na wartość różną od zera. Ta wartość będzie oczywiście wówczas zwrócona jako wynik.
Pewne wartości przypisywane właściwości *ModalResult* maja zwyczajową interpretację i powiązane są z odpowiednią predefiniowaną stałą (tak jak kolory). Nazwy stałych związanych z ModalResult zaczynają się od liter ***mr*** i są to:
MrNone, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrYes, mrNo, mrAll, mrNoToAll, mrYesToAll.

Niestandartowy przycisk *TBitBtn* (zakładka *Additional*) poprzez swoją właściwość *Kind* ułatwia wykorzystanie możliwości zamykania formy poprzez ustawianie jej właściwości *ModalResult*.
Różnice między metodami *Show* i *ShowModal* można więc określić tak, jak to podano w tabeli 1.

Show	**ShowModal**
Wyświetla okno niemodalnie	Wyświetla okno modalnie
Jest procedurą	Jest funkcją – zwraca wartość właściwości *ModalResult* wyświetlanej formy
Kończy natychmiast swoje działanie	Czeka na zamknięcie formy

Uwaga! Wyswietlić modalnie można tylko okno w danym momencie niewidoczne (*Visible = false*).