abc

13

MECHANIZM DRAG & DROP

Tematem tego rozdziału jest przede wszystkim mechanizm przenoszenia elementów w obrębie aplikacji (z ang. *drag & drop*). Ale nie tylko. Pokażę w nim również, w jaki sposób tworzyć kod, który w jak największym stopniu wykorzystuje informacje przekazywane przez zdarzenia do metod zdarzeniowych.

Mechanizm *drag & drop* jest bardzo charakterystycznym elementem graficznego interfejsu użytkownika systemu Windows. Dzięki niemu możemy za pomocą myszy lub innego urządzenia wskazującego „złapać" niektóre elementy widoczne na ekranie, przenieść je w inne miejsce i na przykład umieścić w nowym „pojemniku" lub upuścić na innym elemencie. Przyjrzyjmy się, jak taki mechanizm zastosować w naszej aplikacji, analizując prosty przykład dwóch list. Będziemy przenosili pomiędzy nimi elementy. Nie jest to zadanie zbyt trudne, gdyż komponenty z biblioteki VCL w pełni wspierają mechanizm *drag & drop* w obrębie aplikacji. Inaczej sprawa wygląda w przypadku przeciągania elementów (np. plików) pomiędzy aplikacjami — niezbędne jest wówczas odwołanie się do mechanizmu komunikatów i WinAPI (zob. książka *Delphi 2005. 303 gotowe rozwiązania*, Helion 2005).

Operacja przeniesienia i upuszczenia składa się z trzech etapów:

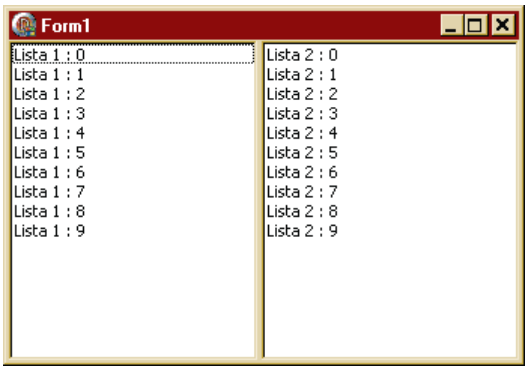
1. Rozpoczęcie przenoszenia. W zależności od ustawienia własności `DragMode` może się to odbywać automatycznie po złapaniu elementu lub poprzez wywołanie metody `BeginDrag` (ręczne wywołanie pozwala np. na filtrowanie elementów, które mogą być przenoszone).
2. Pozostałe elementy aplikacji mogą akceptować lub odrzucać możliwość upuszczenia przenoszonego elementu. Umożliwia to zdarzenie `OnDragOver`, którego metoda zdarzeniowa ma w sygnaturze referencję do zmiennej logicznej (domyślnie `Accept`). Jeżeli metoda zmieni wartość zmiennej `Accept` na `False`, to przenoszony obiekt nie będzie mógł być zrzucony na obiekt, którego dotyczy zdarzenie. Manifestuje się to charakterystycznym kształtem kursora myszy („zakaz wjazdu").
3. Ostatnim etapem jest reakcja na upuszczenie przenoszonego elementu na obiekt. W takiej sytuacji uruchamiana jest metoda związana ze zdarzeniem `OnDragDrop`. To w tej metodzie wykonywana jest operacja kryjąca się za przeciąganiem, np. kopiowanie pliku.

Tyle wprowadzenia. Teraz zabierzmy się za konkretny przykład. Liczba zastosowań mechanizmu *drag & drop* jest wielka, a jednocześnie w każdym przypadku sposób jego realizacji jest bardzo podobny. Dlatego mechanizm ten zilustrujemy na jednym modelowym przykładzie dwóch list, między którymi można przenosić elementy.

Drag & Drop z biblioteką VCL

Przygotowanie interfejsu z dwoma listami

1. Tworzymy nowy projekt *VCL Form Application* dla Win32 lub dla .NET.
2. Na formie umieszczamy dwie listy `TListBox` (rysunek 13.1). Można między nimi umieścić komponent `TSplitter` w sposób omówiony w rozdziale 11.
3. Zapełniamy je ponumerowanymi elementami. W tym celu w inspektorze obiektów z rozwijanej listy wybieramy obiekt `Form1` i tworzymy metodę zdarzeniową do `OnCreate`, w której umieszczamy polecenia z listingu 13.1:



Rysunek 13.1. Listy zapełniamy przykładowymi elementami

Listing 13.1. Zawartość list nie jest szczególnie istotna w tym przykładzie

```
procedure TForm1.FormCreate(Sender: TObject);
var i :Integer;
begin
for i:=0 to 9 do
    begin
    ListBox1.Items.Add('Lista 1 : '+IntToStr(i));
    ListBox2.Items.Add('Lista 2 : '+IntToStr(i));
    end;
end;
```

W ten sposób będziemy mieli formę z dwoma listami. Każdy z nich ma 10 oznaczonych elementów tak, jak na rysunku 13.1.

Faza pierwsza: rozpoczęcie przenoszenia

Zwykle po naciśnięciu lewego klawisza myszy na komponencie listy rozpoczyna się operacja przenoszenia elementu znajdującego się pod myszką. To oznacza, że powinniśmy wykorzystać zdarzenie `OnMouseDown`. Ale tego nie zrobimy, bo komponent `TListBox` może nas w tym wyręczyć. Okazuje się, że wystarczy zmienić jego własność `DragMode` na `dmAutomatic`, a proces przenoszenia będzie inicjowany automatycznie.

Zróbmy tak. Zaznaczmy lewą listę (`ListBox1`) i za pomocą inspektora obiektów zmieńmy własność `DragMode` na `dmAutomatic`.

Po uruchomieniu aplikacji możemy zobaczyć, że kliknięcie lewego przycisku myszki na jednym z elementów i lekkie jej przesunięcie bez puszczania przycisku spowoduje zmianę kursora na charakterystyczny dla *drag & drop*.

Faza druga: Akceptacja upuszczenia

Jeżeli przenoszony element przesuniemy nad inny obiekt, wywoływana jest jego metoda zdarzeniowa związana ze zdarzeniem OnDragOver. Przygotujmy odpowiednią metodę zdarzeniową dla komponentu ListBox2 (listing 13.2).

Listing 13.2. Możliwość upuszczenia elementu zależy od jego źródła

```
procedure TForm1.ListBox2DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
Accept:=False;
if (Source=ListBox1) and (ListBox1.ItemIndex>=0) then
  Accept:=True;
end;
```

Metoda przyjmuje proste kryterium — jeżeli przenoszony element pochodzi z ListBox1 oraz w ListBox1 jest zaznaczony jakiś element, to element zostanie przez ListBox2 przyjęty. W pozostałych przypadkach zostanie odrzucony. Decyzję możemy poznać po kształcie kursora.

Faza trzecia: Upuszczenie przenoszonego elementu

W momencie upuszczenia elementu wywoływane jest zdarzenie OnDragDrop. Jeżeli z tym zdarzeniem komponentu ListBox2 związżemy metodę zdarzeniową, będzie to najwłaściwsze miejsce do zaprogramowania reakcji tego komponentu na upuszczenie na nim przenoszonego elementu. Może ona być zupełnie dowolna — zależy to jedynie od inwencji i potrzeb programisty. My po prostu dodamy przeciągany element na końcu ListBox2 i usuniemy go z ListBox1 (listing 13.3).

Listing 13.3. Po upuszczeniu elementu z listy ListBox1 jest on dopisywany na końcu listy ListBox2 i kasowany ze źródła

```
procedure TForm1.ListBox2DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
ListBox2.Items.Add(ListBox1.Items.Strings[ListBox1.ItemIndex]);
ListBox1.Items.Delete(ListBox1.ItemIndex);
end;
```

Odczytujemy numer przeciąganego elementu z własności ListBox1.ItemIndex i dodajemy jego etykietę do listy ListBox2 (na razie na jej końcu). Następnie usuwamy ten element z ListBox1.

W trzecim etapie nie powinniśmy już sprawdzać, czy dany obiekt może być upuszczony. To powinna robić metoda związana ze zdarzeniem OnDragOver w drugiej fazie. Inaczej moglibyśmy doprowadzić do sytuacji, w której pojawia się akceptujący kursor myszy, a elementu nie można upuścić.

Usprawnienia

Umieszczanie elementu w miejscu upuszczenia

Ostatnią metodę można nieco zmodyfikować, tak aby element był dodawany nie na końcu listy, a w miejscu upuszczenia. Wystarczy zamiast ListBox2.Items.Add wykorzystać metody ListBox2.Items.Insert oraz ListBox2.ItemAtPos zgodnie ze wzorem z listingu 13.4.

Listing 13.4. Wersja pozwalająca na wstawienie elementu między elementy listy ListBox2

```
procedure TForm1.ListBox2DragDrop(Sender, Source: TObject; X, Y: Integer);
var punkt :TPoint;
begin
punkt.X:=X;
punkt.Y:=Y;
ListBox2.Items.Insert(ListBox2.ItemAtPos(punkt,False),ListBox1.Items.Strings
[ListBox1.ItemIndex]);
ListBox1.Items.Delete(ListBox1.ItemIndex);
end;
```

Na razie możliwości przenoszenia są dość skromne. Można przenosić tylko po jednym elemencie z lewej listy na prawą. W kolejnych paragrafach uelastycznimy powyższe metody w taki sposób, aby można je było związać z dowolnym obiektem formy. Wszelkie informacje będziemy przekazywać poprzez argumenty metod zdarzeniowych bez jawnego korzystania z nazw obiektów, pomiędzy którymi przenosimy wybrany element. Wbrew pozorom to uprości, a nie skomplikuje kod metod. Poza tym umożliwimy przenoszenie większej liczby elementów jednocześnie.

Uelastycznianie kodu. Wykorzystanie referencji Sender

Pierwszym argumentem wszystkich metod zdarzeniowych VCL jest referencja Sender przechowująca adres obiektu, którego zdarzenie dotyczy. Sender jest zmienną typu TObject, a więc tego, z którego dziedziczą wszystkie obiekty VCL. W przypadku zdarzeń OnDragOver i OnDragDrop przesyłana jest również druga referencja Source, wskazująca na komponent, z którego podjęte zostały elementy.

Zmieńmy najpierw metodę TForm1.ListBox2DragOver i zamiast sprawdzać, czy źródło z którego zabrany został element jest komponentem ListBox1, sprawdzimy po prostu, czy jest to inny komponent niż ten, na który element jest upuszczany. W metodzie należy wprowadzić zdarzenia z listingu 13.5:

Listing 13.5. Wpuszczeni będą tylko obcy

```
procedure TForm1.ListBox2DragOver(Sender, Source: TObject; X, Y: Integer;
  State: TDragState; var Accept: Boolean);
begin
Accept:=False;
if (Source <> Sender) then Accept:=True;
end;
```

Dzięki tej drobnej zmianie metoda nie jest w ogóle zależna od konkretnych nazw komponentów. Może więc być podpięta do lewej listy i działać na jej rzecz bez żadnych modyfikacji. Zróbmy to. To znaczy zaznaczmy komponent ListBox1 w widoku projektowania, w inspektorze obiektów zmieńmy zakładkę na *Events* i z rozwijanej listy przy zdarzeniu OnDragOver wybierzmy metodę ListBox2DragOver.

Poza tym umożliwmy przenoszenie elementów z prawej listy przełączając własność DragMode komponentu ListBox2 na dmAutomatic.

Przenoszenie z prawej na lewą nie jest jeszcze możliwe — działa etap pierwszy i drugi, ale do tego, żeby zadziałał i trzeci, musimy uelastycznić metodę ListBox2DragDrop. Potem będziemy mogli związać ją ze zdarzeniem OnDragDrop lewej listy.

Rzutowanie referencji Sender

Modyfikujemy zatem metodę ListBox2DragDrop w taki sposób, żeby nie zawierała nazw obiektów. Tym razem będziemy musieli zrzutować oba argumenty (Sender i Source) na typ TListBox, żeby móc skorzystać z metod komponentów TListBox pozwalających na dodawanie i usuwanie elementów z listy. Dla ułatwienia, żeby nie powtarzać rzutowania, zdefiniujemy lokalną zmienną SenderLB typu TListBox i przypiszemy jej adres przechowywany przez zmienną Sender za pomocą polecenia SenderLB:=Sender as TListBox;.

Niektórzy Czytelnicy mogą dziwić się, że „wskaźnik do TObject przechowuje te same informacje, co TListBox, a przecież deklaracja tej klasy jest znacznie uboższa". Ale wskaźnik nie przechowuje żadnych informacji poza adresem obiektu, który dla wszystkich typów zmiennych i obiektów jest tego samego rozmiaru. Twierdzenie to nie dotyczy wskaźników własności i metod, które poza swoim bezpośrednim adresem muszą też przechowywać adres obiektu — są więc dwukrotnie większe.

Konieczne zmiany zostały przedstawione na listingu 13.6. Po ich wprowadzeniu możemy związać tę metodę z lewą listą. W tym celu przechodzimy do widoku projektowania i zaznaczamy komponent ListBox1. W inspektorze obiektów, na zakładce *Events* z rozwijanej listy przy zdarzeniu OnDragDrop wybieramy ListBox2DragDrop.

Listing 13.6. Przy upuszczaniu należy rozpoznać zarówno komponent, z którego pochodzi upuszczany element, jak i jego źródło

```
procedure TForm1.ListBox2DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  SenderLB,SourceLB :TListBox;
  punkt :TPoint;
begin
SenderLB:=Sender as TListBox;
SourceLB:=Source as TListBox;
punkt.X:=X;
punkt.Y:=Y;
SenderLB.Items.Insert(SenderLB.ItemAtPos(punkt,False),SourceLB.Items.Strings
[SourceLB.ItemIndex]);
SourceLB.Items.Delete(SourceLB.ItemIndex);
end;
```

Jak przenosić wiele elementów?

Aby umożliwić zaznaczanie wielu elementów, należy włączyć własność MultiSelect w obu listach. W przypadku zaznaczonej większej ilości elementów własność ItemIndex przestaje być użyteczna. Zaznaczone elementy są identyfikowane na podstawie tablicy Selected, która ma tę samą długość co Items i przechowuje wartości logiczne (True, jeżeli odpowiedni element jest zaznaczony, i False w przeciwnym przypadku).

Komponent ListBox1 ma własność ExtendedSelect, na którą także warto zwrócić teraz uwagę. Odpowiada ona m.in. za sposób, w jaki zaznaczane jest wiele elementów. Jeżeli jest ona włączona, a domyślnie tak jest, to po włączeniu MultiSelect możliwe jest zaznaczanie myszką grupy elementów przez przeciągnięcie po nich myszką z naciśniętym lewym klawiszem. Można również zaznaczyć jeden element, i przyciskając klawisz *Shift* inny, a zaznaczone zostaną wszystkie elementy pomiędzy nimi. Z kolei klawisz *Ctrl* służy do zaznaczania lub „odznaczania" poszczególnych elementów. Wadą tego trybu jest jednak to, że lewym klawiszem myszy bez naciskania dodatkowych klawiszy klawiatury nie można po prostu zaznaczać elementów. Jest to jednak możliwe, jeżeli własność ExtendedSelect przełączymy na False. Wówczas klikane myszką elementy w ListBox1 są zaznaczane tak, jakbyśmy trzymali klawisz *Ctrl* w trybie rozszerzonym. W tym przypadku nie jest jednak możliwe, przeciągając myszką lub z pomocą klawisza *Shift,*

zaznaczanie grupy elementów. Ponadto, jeżeli własność DragMode przełączona jest na dmAutomatic, zaznaczanie wielu elementów przy wyłączonej własności ExtendedSelect nie działa właściwie. Nie działa wówczas również zaznaczanie wielu elementów przez przeciąganie myszką, gdy ExtendedSelect jest włączona.

Jedyna zmiana, jaką musimy wprowadzić w kodzie, dotyczy metody ListBox2DragDrop, w której należy wykonać pętle po wszystkich elementach listy będącej źródłem przeciąganych elementów i dla tych spośród nich, które są zaznaczone, wykonać operację przenoszenia (listing 13.7).

Listing 13.7. Jeżeli dodawanie do SenderLB i usuwanie z SourceLB byłoby wykonywane w jednej pętli, przenoszony byłby jedynie co drugi element z powodu zmiany numeracji elementów

```
procedure TForm1.ListBox2DragDrop(Sender, Source: TObject; X, Y: Integer);
var
  SenderLB,SourceLB :TListBox;
  punkt :TPoint;
  i :Integer;
begin
SenderLB:=Sender as TListBox;
SourceLB:=Source as TListBox;
punkt.X:=X;
punkt.Y:=Y;

for i:=SourceLB.Items.Count-1 downto 0 do
  if (SourceLB.Selected[i]) then
    begin
    SenderLB.Items.Insert(SenderLB.ItemAtPos(punkt,False),SourceLB.Items.
    Strings[i]);
    SourceLB.Items.Delete(i);
    end;
end;
```

Założę się, że zwrócił uwagę Czytelnika fakt, iż dodawanie i usuwanie elementów zrealizowane jest w pętli indeksowanej malejąco (słowo kluczowe downto zamiast typowego to w instrukcji pętli for). Dlaczego? Po to, aby ominąć problemy ze zmianą numeracji przenoszonych elementów. Załóżmy na przykład, że zaznaczyliśmy elementy od czwartego do siódmego w pierwszej liście i przenosimy je do drugiej listy. Jeżeli pętla byłaby indeksowana w typowy sposób, to pierwszą wartością i, dla której własność Select zwróciłaby True jest 4. Zostałaby ona wstawiona w odpowiednie miejsce drugiej listy i usunięta z pierwszej. Ale wówczas dzieje się jeszcze coś, czego powinniśmy być świadomi. Po usunięciu elementu następuje automatyczne porządkowanie numeracji na liście. Element piąty spada na czwartą pozycję, szósty na piątą i tak dalej. Jednak nasza pętla szłaby dalej i sprawdzałaby własność Select pozycji o numerze 5. W ten sposób element, który przed usunięciem czwartej pozycji był na miejscu piątym, a potem spadł na miejsce czwarte, zostałby pominięty.

Indeksowanie w dół pomaga także w dodawaniu elementów do drugiej listy na określonej pozycji. Jeżeli indeksowalibyśmy „normalnie", czyli rosnąco, konieczne byłoby także zwiększanie pozycji, na której wstawiamy elementy. Natomiast jeżeli wstawiamy elementy od ostatniego do pierwszego, to możemy wstawiać każdy dokładnie w miejsce obliczone na podstawie pozycji myszki. Elementy o wyższym indeksie zostaną automatycznie przesunięte.

No i nasza aplikacja uzyskała wreszcie pełną funkcjonalność mechanizmu *drag & drop*. Żeby przenosić wiele elementów, wystarczy je zaznaczyć, korzystając z klawisza *Shift* lub *Ctrl*, i (nadal trzymając ten klawisz, aby nie zmienić zaznaczenia) przeciągnąć je na drugą listę.