Przeciągnij i upuść

Typowym zagadnieniem, które wiąże się z tworzeniem aplikacji z graficznym interfejsem użytkownika, jest korzystanie z możliwości przeciągania elementów pomiędzy różnymi komponentami-pojemnikami za pomocą myszy. Obsługę tego mechanizmu z poziomu kodu C# stworzymy na przykładzie aplikacji z dwoma listami.

Z punktu widzenia programisty operacja *drag & drop* (*przeciągnij i upuść*) może być zrealizowana dzięki obsłudze kilku zdarzeń, przede wszystkim MouseDown, DragEnter, DragOver i DragDrop. Operacja przeniesienia i upuszczenia składa się z trzech etapów:

1. Rozpoczęcia przenoszenia — wymaga wywołania metody DragDrop na rzecz komponentu, z którego przenoszony element jest zabierany.

2. Akceptacji — przeciągając element nad inny komponent, wywołujemy jego zdarzenie DragOver; jest to właściwy moment, żeby podjąć decyzję o tym, czy przenoszony element może być w ogóle upuszczony na docelowy komponent. Użytkownik aplikacji powinien zostać powiadomiony o decyzji za pomocą zmiany kształtu kursora myszy.

3. Reakcji na upuszczenie przenoszonego elementu — w takiej sytuacji uruchamiana jest metoda związana ze zdarzeniem DragDrop.

W naszym projekcie przygotujemy metody obsługujące przeciąganie i upuszczanie elementów. W miarę możliwości zrobimy to w taki sposób, żeby nie używać jawnie nazw pojemników. W zamian użyjemy argumentów przekazywanych przez metody zdarzeniowe. Dzięki temu metody będą elastyczniejsze — będzie ich można użyć dla wielu pojemników (w naszym przypadku list). A ponadto ułatwiać to będzie kopiowanie ich kodu do kolejnych projektów.

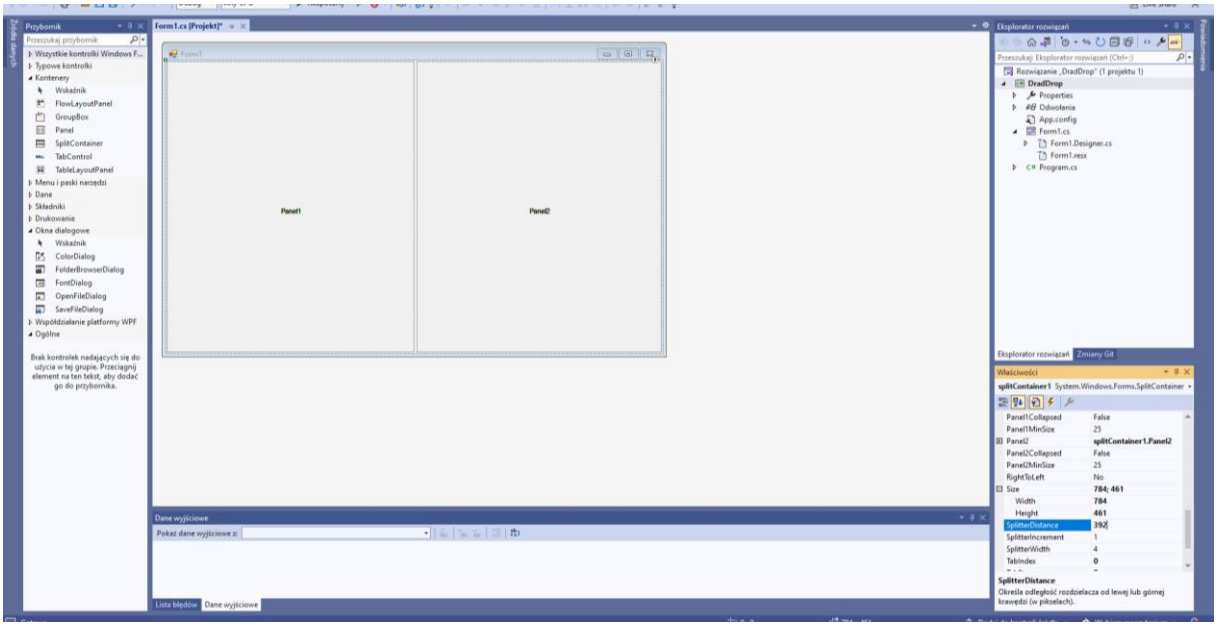
Interfejs aplikacji

Zacznijmy od przygotowania projektu graficznego aplikacji. W tym celu na formie umieścimy dwie listy wypełnione kilkoma pozycjami.

1. Utwórz nowy projekt typu Windows Forms Application o nazwie *DragAndDrop*.

2. W widoku projektowania powiększ formę i umieść na niej komponent SplitContainer z zakładki *Containers* (z ang. *pojemniki*).

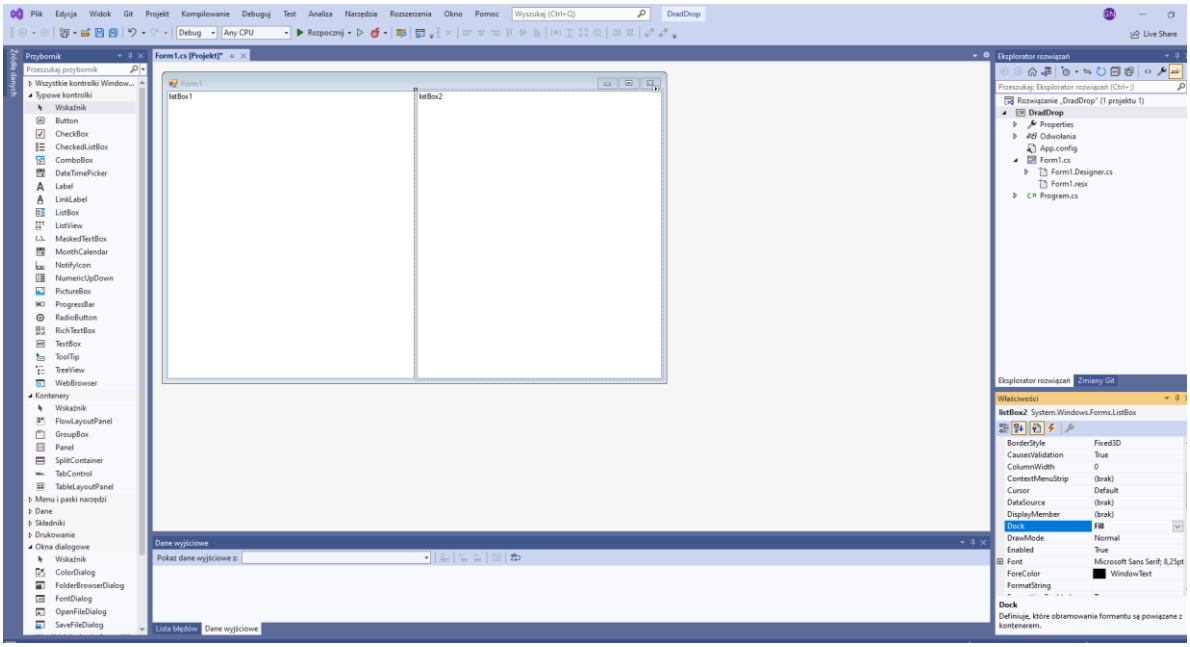
3. Ustaw identyczną szerokość jego paneli (własność SplitterDistance).



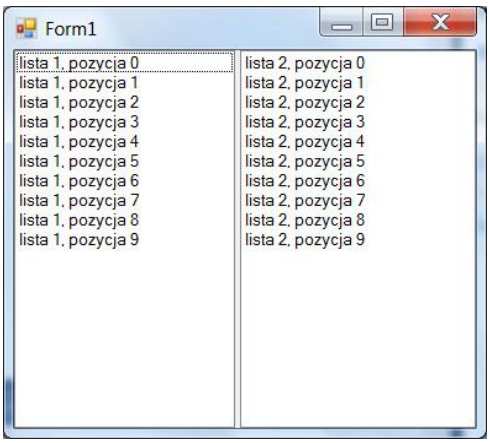
4. W jego lewym panelu umieść listę ListBox.

5. Jej własność Dock zmień na Fill. W efekcie lista wypełni cały lewy panel.

6. W podobny sposób umieść drugą listę w prawym panelu i ją również zadokuj.



7. W listach umieść po dziesięć pozycji (własność Items), które będziesz mógł następnie przenosić z jednej listy do drugiej



8. Dla wygody można również zmienić pozycję okna po uruchomieniu aplikacji. W widoku projektowania zmienić własność StartPosition formy na CenterScreen.

Inicjacja procesu przeciągania

Aby można było uruchomić procedurę *drag & drop*, konieczne jest wywołanie metody DragDrop na rzecz komponentu, z którego przenosimy element (w naszym przypadku jednej z list). Uruchommy zatem metodę DragDrop listy w przypadku wciśnięcia na jej obszarze lewego przycisku myszy.

1. W edytorze kodu zdefiniuj prywatne pole klasy Form1 o nazwie dragDropSource typu object, które będzie przechowywać referencję do komponentu, z jakiego zabraliśmy przeciągany element. Nowa deklaracja musi być umieszczona w obrębie klasy Form1, ale poza jej metodami.

```
public partial class Form1 : Form
    {

        object dragDropSource = null;

       . . .
```

2. Następnie przejdźmy do widoku projektowania i przytrzymując klawisz *Ctrl*, zaznaczmy jednocześnie komponenty listBox1 i listBox2.

3. Utworzmy metodę zdarzeniową związaną ze zdarzeniem MouseDown obu komponentów. W niej

```
        private void listBox1_MouseDown(object sender, MouseEventArgs e)
        {
            ListBox lbSender = sender as ListBox;
            int indeks = lbSender.IndexFromPoint(e.X, e.Y);
            dragDropSource = sender; // przechowanie referencji dla DragOver
            if (e.Button == MouseButtons.Left && indeks != -1)
            {
                lbSender.DoDragDrop(lbSender.Items[indeks],DragDropEffects.Copy |
DragDropEffects.Move);
            }

        }
```

Wywołanie metody DragDrop obwarowaliśmy kilkoma warunkami: sprawdzamy, czy użyty przycisk myszy to przycisk lewy i czy rzeczywiście w momencie jego przyciśnięcia kursor myszy znajdował się nad jednym z elementów listy. Dopiero wtedy, kiedy oba warunki są spełnione, wywołujemy metodę DragDrop.

Metoda ta posiada dwa parametry: data typu object (czyli dane które chcemy przenieść) oraz typ wyliczeniowy DragDropEffects (zbiór operacji, jakie można na nim wykonać). W typie wyliczeniowym możemy znaleźć takie opcje jak:

None – kontrolka docelowa nie pozwala na kopiowanie

Copy – powoduje skopiowanie danych z kontrolki źródłowej do docelowej

Move – przenosi dane z kontrolki źródłowej do docelowej

Link – zostaje utworzone połączenie (logiczne) z elementem docelowym

Scroll – pozwala na przewijanie kontrolki docelowej podczas przenoszenia

All – połączenie efektów Copy, Move i Scroll

W przypadku list sensowne operacje ograniczają się w zasadzie do DragDropEffects.Move oznaczającej przenoszenie oraz DragDropEffects.Copy oznaczającej kopiowanie. Zezwalamy na obydwie.

Działanie metody DragDrop kończy się dopiero wtedy, gdy przenoszony obiekt zostanie upuszczony (przycisk myszy zostanie zwolniony). Z tego wynika, że zdarzenia DragOver i DragDrop muszą być wykonane w osobnym wątku, bo metoda, w której uruchomiona zostanie metoda DragDrop, nie zakończy się przed ich wywołaniem.

Uwaga. Nie należy po wywołaniu metody DragDrop umieszczać żadnych poleceń, które miałyby należeć do pierwszej fazy procesu *drag & drop*. Można tam umieścić polecenia należące do trzeciej fazy procesu, a szczególnie ewentualne operacje dotyczące komponentu źródłowego, np. usuwanie przenoszonego elementu.

W metodzie listBox1_MouseDown, podobnie jak i w kolejnych, unikamy jawnego używania nazw komponentów. Zamiast tego wykorzystaliśmy argument sender metody zdarzeniowej przesyłający referencję do komponentu, z powodu którego wywołano zdarzenie. W naszym przypadku należy go oczywiście wcześniej zrzutować na typ ListBox (zmienna lbSender). Przygotowana w taki sposób metoda może być używana do wszystkich list, które umieścimy na formie.

Ponieważ do metody zdarzeniowej związanej ze zdarzeniem DragOver, która będzie wykorzystywana w kolejnej fazie procesu (pozwolenia upuszczenia przenoszonego elementu w zależności od tego skąd pochodzi), nie jest przekazywana żadna informacja o komponencie, z którego pochodzi przenoszony obiekt należy zdefiniować referencję do komponentu, z którego pobieramy element.

```
dragDropSource = sender; // przechowanie referencji dla DragOver
```

Akceptacja upuszczenia elementu

Jeżeli przenoszony element przesuniemy nad docelowy komponent (np. drugą listę), uruchomione zostanie jego zdarzenie DragOver. Daje to możliwość nadania kursorowi myszy kształtu, który informuje użytkownika o tym, czy upuszczenie przenoszonego obiektu na danym komponencie jest możliwe. Aby jednak to zdarzenie było w ogóle wywoływane, własność AllowDrop komponentu docelowego musi być ustawiona na true.

Przejdźmy do kontrolki docelowej. Aby możliwe było upuszczenie na docelową kontrolkę jakiegoś elementu, kontrolka musi obsłużyć dwa zdarzenia: DragOver oraz DragDrop. Zdarzenia te otrzymują parametr DragEventArgs, który zawiera informacje wymagane do wykonania procedury przenoszenia.

Elementy składające się na DragEventArgs:

AllowedEffect – efekt który jest wspierany przez kontrolkę źródłową,

Data – zwraca obiekt typu IDataObject w którym przechowywane są dane z kontrolki źródłowej,

Effect – ustala efekt dla kontrolki docelowej,

X,Y – współrzędne myszy,

KeyState – zwraca stan klawiszy i myszy jako integer.

Zaznacz obie listy na podglądzie formy. Zmień ich własność AllowDrop na True. Utwórz dla nich obu metodę zdarzeniową związaną ze zdarzeniem DragOver, w której umieścisz instrukcje określające kształt kursora po przeniesieniu obiektu nad te komponenty (własność Effect – ustala efekt dla kontrolki docelowej)

```
private void listBox1_DragOver(object sender, DragEventArgs e)

{
    if (sender == dragDropSource)
        e.Effect = DragDropEffects.None;
    else
        if ((e.KeyState & 8) == 8)
            e.Effect = DragDropEffects.Copy; // z CTRL
        else
            e.Effect = DragDropEffects.Move;


}
```

Kształt kursora wyznaczony jest w powyższej metodzie przez rodzaj operacji, jaką chcemy wykonać na przenoszonym elemencie. Należy pamiętać przy tym, że wartość przypisana do decydującej o tym własności e.Effect musi być jedną z wartości wskazanych w drugim argumencie metody DragDrop. My wskazaliśmy wówczas dwie: kopiowanie i przenoszenie. Rodzaj wykonywanej operacji zależy od tego, czy został naciśnięty klawisz *Ctrl*. Jeżeli tak, element zostanie skopiowany, jeżeli nie — przeniesiony.

KeyState – zwraca stan klawiszy i myszy jako integer:

```
if ((e.KeyState & 32) == 32)  // ALT wciśnięty
if ((e.KeyState & 8) == 8)  // CTRL wciśnięty
 if ((e.KeyState & 4) == 4)  // SHIFT wciśnięty
```

Dodatkowym warunkiem, wyrażonym w pierwszej linii kodu metody listBox1_DragOver, jest to, żeby pojemnik zgłaszający zdarzenie (komponent, nad którym jest kursor myszy przenoszący element) nie był tym samym komponentem, z jakiego obiekt został pobrany. W takim przypadku możliwość upuszczenia jest blokowana. Właśnie do tego typu warunków, i tylko dla nich, konieczne było umieszczenie w zmiennej dragDropSource referencji do komponentu, z którego przenoszony jest obiekt.

W momencie upuszczenia elementu aktywowane zostaje zdarzenie DragDrop komponentu, nad którym aktualnie znajduje się kursor myszy. Chcemy, aby w reakcji na to zdarzenie przenoszony element był dodany do listy, na którą został upuszczony. Kod metody zdarzeniowej związanej ze zdarzeniem DragDrop (należy ją związać ze zdarzeniem obu list).

```
        private void listBox1_DragDrop(object sender, DragEventArgs e)
        {
            ListBox lbSender = sender as ListBox;
            int indeks = lbSender.IndexFromPoint(lbSender.PointToClient(new
Point(e.X, e.Y)));
            if (indeks == -1) indeks = lbSender.Items.Count;
            lbSender.Items.Insert(indeks,
e.Data.GetData(DataFormats.Text).ToString());

        }
```

Na tym etapie nie powinniśmy już sprawdzać, czy dany obiekt może zostać upuszczony. To należało do zadań metody związanej ze zdarzeniem DragOver. Inaczej mogłoby dojść do sytuacji, w której pojawiłby się kursor myszy świadczący o akceptacji upuszczenia, a przenoszonego elementu wcale nie dałoby się upuścić.

W drugiej linii powyższej metody określamy pozycję elementu listy, nad którym znajdował się upuszczony element w momencie zwolnienia przycisku myszy.

```
int indeks = lbSender.IndexFromPoint(lbSender.PointToClient(new Point(e.X, e.Y)));
```

Pewnym utrudnieniem jest to, że tym razem położenie myszy przekazywane jest w obiekcie DragEventArgs we współrzędnych ekranu, a nie formy. Pozornie jest to szczegół, ale nieświadomy tego faktu programista może osiwieć, próbując znaleźć powód, dla którego kod nie chce pracować, szczególnie że w obiekcie MouseEventArgs, używanym wcześniej w metodzie zdarzeniowej związanej z MouseDown, położenie myszy podawane było we współrzędnych okna.

Jeżeli miejsce zwolnienia przycisku myszy i upuszczenia obiektu nie znajduje się nad żadnym elementem listy, to zwracany indeks równy jest –1.

```
if (indeks == -1) indeks = lbSender.Items.Count;
```

Wówczas zmieniamy jego wartość na równą liczbie elementów w liście, co wskazuje na miejsce za ostatnim elementem. Pozwoli to na użycie także i tu metody ListBox.Items.Insert, która w tym przypadku zadziała tak samo, jakbyśmy użyli metody ListBox.Items.Add.

Czynności wykonywane po zakończeniu procesu przenoszenia i upuszczania

W metodzie listBox1_DragDrop brakuje poleceń usuwających przenoszony element z komponentu źródłowego. Można je tam umieścić, korzystając z referencji do pojemnika źródła dragDropSource zrzutowanej na typ ListBox, ale wygodniej umieścić je w metodzie listBox1_MouseDown po wywołaniu (i zakończeniu) metody DragDrop, oczywiście jeżeli zrealizowany proces był przenoszeniem, a nie kopiowaniem.

```
        private void listBox1_MouseDown(object sender, MouseEventArgs e)
        {
            ListBox lbSender = sender as ListBox;
            int indeks = lbSender.IndexFromPoint(e.X, e.Y);
            dragDropSource = sender; // przechowanie referencji dla DragOver
            if (e.Button == MouseButtons.Left && indeks != -1)
            {
                DragDropEffects operacja =
                lbSender.DoDragDrop(lbSender.Items[indeks], DragDropEffects.Copy
| DragDropEffects.Move);

                if (operacja == DragDropEffects.Move)
                    lbSender.Items.RemoveAt(indeks);
            }

            dragDropSource = null;          }
```

Sprawdzenia, czy użytkownik zdecydował się na kopiowanie, czy na przenoszenie, dokonujemy, korzystając z wartości typu DragDropEffects zwracanej przez metodę DragDrop. Jest to wartość równa tej, jaką przypisaliśmy w metodzie listBox1_DragOver do e.Effect. Na tej podstawie podejmowana jest decyzja o tym, czy usunąć, czy pozostawić przenoszony element w pojemniku - źródle. Na końcu tej metody dodaliśmy także polecenie „czyszczące” pole dragDropSource.

Aby sprawdzić poprawność i ogólność przygotowanych metod, możemy wykonać prosty test. Dodajmy do formy trzecią listę ListBox, ustawmy jej własność AllowDrop na true i podepnijmy ją do wszystkich trzech metod zdarzeniowych (odpowiednio do zdarzeń MouseDown, DragOver i DragDrop). W ten sposób włączymy ją do układu kontrolek, między którymi możliwe jest przenoszenie elementów.

Przenoszenie wielu elementów

Komponent ListBox pozwala na zaznaczanie wielu elementów jednocześnie. Aby tę możliwość odblokować, należy ustawić własność SelectionMode na MultiSimple lub MultiExtended. W pierwszym trybie lista działa podobnie do zbioru pól opcji, tzn. kliknięcie dowolnego elementu powoduje jego zaznaczenie niezależnie od innych, a ponowne kliknięcie usuwa zaznaczenie. W trybie MultiExtended do zaznaczania w ten sposób pojedynczych elementów bez usuwania zaznaczania innych należy wykorzystać klawisz *Ctrl*. Poza tym można także używać klawisza *Shift*, który pozwala na zaznaczanie grupy elementów od obecnie zaznaczonego do klikanego myszą.

Zaznacz obie listy i ustaw ich własności SelectionMode na MultiExtended.

Zmiany rozpoczniemy od metody listBox1_MouseDown. Traci w niej zasadność definiowanie zmiennej lokalnej indeks, która do tej pory przechowywała numer przenoszonego elementu. Zamiast tego należy odczytywać listę zaznaczonych elementów z własności ListBox.SelectedItems.

```
private void listBox1_MouseDown(object sender, MouseEventArgs e)
{
    ListBox lbSender = sender as ListBox;
    // int indeks = lbSender.IndexFromPoint(e.X, e.Y);
     dragDropSource = sender; // przechowanie referencji dla DragOver
   //if (e.Button == MouseButtons.Left && indeks != -1)
   if (e.Button == MouseButtons.Left && (lbSender.SelectedIndices.Count > 0))
     {
       DragDropEffects operacja =
        //lbSender.DoDragDrop(lbSender.Items[indeks], DragDropEffects.Copy |
DragDropEffects.Move);
       lbSender.DoDragDrop(lbSender.SelectedItems, DragDropEffects.Copy |
       DragDropEffects.Move);
   if (operacja == DragDropEffects.Move)
       // foreach (int indeks in lbSender.SelectedIndices)
       //lbSender.Items.RemoveAt(indeks)
       for (int i = lbSender.SelectedItems.Count - 1; i >= 0; i--)
            lbSender.Items.Remove(lbSender.SelectedItems[i]);
       }
   dragDropSource = null;

}
```

Teraz zmieńmy metodę listBox1_DragDrop, w której elementy zaznaczone w jednej z list są dodawane do drugiej.

```
        private void listBox1_DragDrop(object sender, DragEventArgs e)
        {
            ListBox lbSender = sender as ListBox;
            ListBox lbSource = dragDropSource as ListBox;
            int indeks = lbSender.IndexFromPoint(lbSender.PointToClient(new
Point(e.X, e.Y)));
            if (indeks == -1) indeks = lbSender.Items.Count;
           // lbSender.Items.Insert(indeks,
e.Data.GetData(DataFormats.Text).ToString());
            for (int i = lbSource.SelectedItems.Count - 1; i >= 0; i--)
                lbSender.Items.Insert(indeks,
lbSource.Items[lbSource.SelectedIndices[i]]);

        }
```

W metodzie listBox1_MouseDown argumentem metody DragDrop jest teraz kolekcja zaznaczonych elementów. Mimo wszystko nie skorzystamy z tego przesyłanego zbioru w metodzie listBox1_DragDrop (dzięki temu zmiany będą drobniejsze).

Kolekcję zaznaczonych elementów pobierzemy z własności SelectedItems odczytanej z referencji dragDropSource (po zrzutowaniu na typ ListBox).