Napiszemy prostą aplikację do prezentowania i edycji listy rzeczy do zrobienia. Nadajmy naszemu projektowi nadamy nazwę ToDoList. Zanim zaczniemy tworzyć interfejs użytkownika na naszym formularzu, zdefiniujemy klasę, która w naszej aplikacji będzie reprezentowała poszczególne zadania do wykonania. W tym celu dodajmy do projektu klasę o nazwie ToDoEntry

```
internal class ToDoEntry
{
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime DueDate { get; set; }

}
```

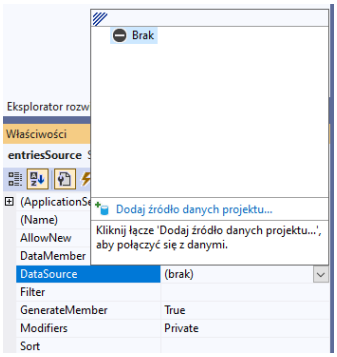
Po dodaniu powyższej klasy powinniśmy skompilować projekt. Będziemy bowiem korzystać z pewnych narzędzi projektowych udostępnianych przez Visual Studio, które muszą dysponować informacjami o naszych klasach, a to wymaga wcześniejszego zbudowania projektu.

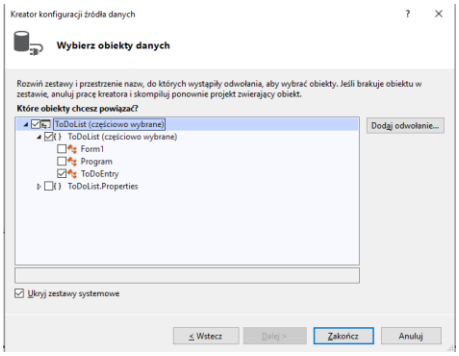
Teraz musimy się upewnić, że Windows Forms wie, iż będziemy używali tej klasy jako źródła danych. W tym celu utworzymy tak zwane *źródło wiązania* poprzez dodanie kontrolki BindingSource do naszego projektu (znajduje się w sekcji *Data*).

Klasa BindingSource nadzoruje sposób, w jaki interfejs użytkownika Windows Forms używa konkretnego źródła danych. W przypadku gdy dysponujemy kolekcją elementów takich jak wpisy na naszej liście rzeczy do zrobienia, klasa BindingSource śledzi, który z jej elementów jest w danej chwili wybrany, i jest w stanie koordynować operacje ich dodawania, usuwania oraz modyfikacji. Korzystanie z klasy BindingSource może także ułatwić prace w projektancie formularzy Visual Studio, gdyż dostarcza ona informacji dotyczących używanych danych, a to może pomóc w powiązaniu ich z kontrolkami.

Zmieńmy właściwość (Name) - przypiszmy jej nazwę entriesSource. Kolejną czynnością jest określenie, czego chcemy używać jako źródła danych. W panelu *Właściwości* można znaleźć właściwość DataSource, a kiedy rozwiniemy dostępną w niej listę, pojawi się okienko dialogowe prezentujące wszystkie źródła danych dostępne w projekcie. Aktualnie nie będzie dostępnych żadnych źródeł, dlatego też musimy kliknąć łącze *Dodaj źródło danych projektu* (*Add Project*

Data Source) umieszczone u dołu okna. Kliknięcie tego łącza spowoduje wyświetlenie okna dialogowego kreatora *Kreator konfiguracji źródła danych* (*Data Source Configuration Wizard).* Obsługuje ono kilka rodzajów źródeł danych. Jego postać może się nieco różnić w zależności od tego, z jakiej wersji Visual Studio korzystamy, niemniej jednak powinny w nim być dostępne takie opcje jak: *Baza danych (Database), Usługa (Service), Obiekt (Object).* W naszym projekcie będziemy korzystali z obiektów - w tym celu dodaliśmy do niego klasę ToDoEntry. Zaznaczamy opcję *Obiect* i klikamy Dalej. Następne okno dialogowe kreatora, pozwala określić typ obiektów, z którymi chcemy związać kontrolki. W naszym przypadku będzie to klasa ToDoEntry.



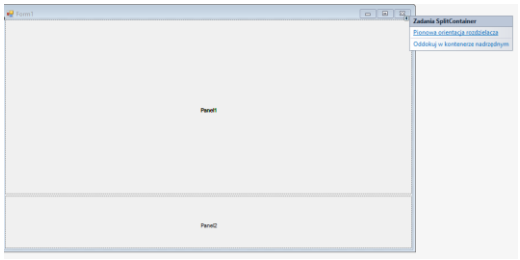


Po kliknięciu przycisku *Zakończ* kontrolka BindingSource będzie już wiedzieć, na jakich obiektach ma operować. Ostatnią czynnością będzie przekazanie jej konkretnych obiektów. Gdybyśmy dysponowali połączeniem z bazą danych, Visual Studio mogłoby przygotować wszystko, co niezbędne, by automatycznie pobrać dane z bazy, jednak ponieważ będziemy używali obiektów, musimy je przygotować i dostarczyć samodzielnie. Należy to zrobić w kodzie aplikacji.

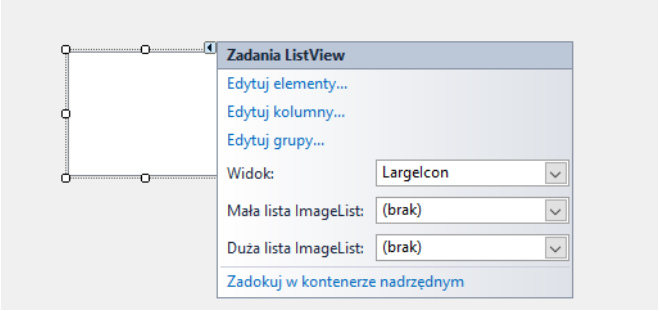
```
public partial class Form1 : Form
{
    private BindingList<ToDoEntry> entries = new BindingList<ToDoEntry>();
    public Form1()
    {
    InitializeComponent();
    entriesSource.DataSource = entries;
    }

}
```

Gdy zawartość obiektu BindingList<T> ulega zmianie, zgłasza on zdarzenia, które pozwalają systemowi wiązania danych technologii Windows Forms odpowiednio aktualizować stan interfejsu użytkownika podczas dodawania i usuwania elementów.

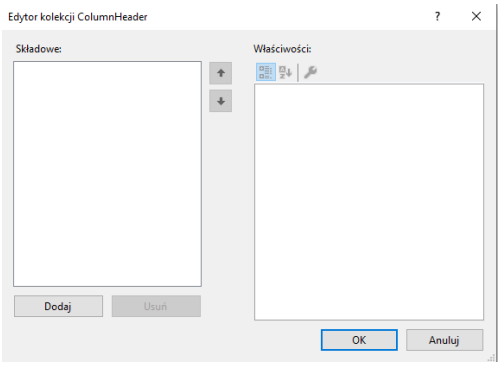
Nasza aplikacja będzie się składała z listy elementów wyświetlonej w górnej części okna oraz z kilku pól służących do edycji danych umieszczonych w jego dolnej części. Te dwie główne części interfejsu użytkownika utworzymy, korzystając z kontrolki SplitContainer. Domyślnie dzieli ona okno na dwie części w pionie — są one umieszczone po prawej oraz po lewej stronie paska. W prawym górnym wierzchołku kontrolki pojawia się niewielka strzałka, której kliknięcie powoduje wyświetlenie okienka z opcjami. Kliknięcie opcji *Pozioma orientacja rozdzielcza (Horizontal Splitter Orientation)* zmieni układ kontrolki na taki, jaki chcemy uzyskać.



W górnej części interfejsu użytkownika chcemy wyświetlić listę ze wszystkimi zadaniami do wykonania. Będziemy chcieli, by na liście były widoczne co najmniej dwie informacje o każdym z zadań - na przykład jego nazwa oraz termin, do którego należy je wykonać. W tym przypadku nie wystarczy nam zwyczajna kontrolka ListBox. Niezbędne możliwości obsługi większej liczby kolumn zapewnia kontrolka ListView. Umieśćmy kontrolkę ListView na formularzu i umieścimy nad jednym z paneli kontrolki SplitContainer.

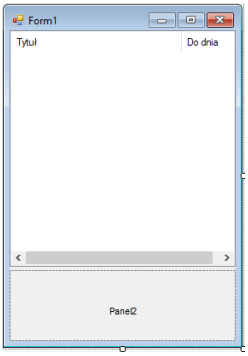


Chcemy, by lista zajmowała cały dostępny obszar górnego panelu – własność Dock. Oprócz tego będziemy chcieli zmienić domyślny widok używany przez listę - domyślnie wybrany jest widok LargeIcon, my zmienimy go na Details. I w końcu musimy też przekazać kontrolce informacje o kolumnach – „Edytuj kolumny”.



Kiedy je wybierzemy, na ekranie zostanie wyświetlone okno dialogowe *Edytor kolekcji ColumnHeader (ColumnHeader Collection Editor)*. Teraz należy dwa razy kliknąć przycisk *Dodaj*, by dodać dwie kolumny.

Korzystając z właściwości (Name), nadamy pierwszej kolumnie nazwę titleColumn, a drugiej - dueDateColumn. Oczywiście chcemy także wyświetlić w nagłówkach obu tych kolumn teksty, które będą nieco bardziej przydatne niż domyślny tekst ColumnHeader. W tym celu zmienimy wartość właściwości Text obu kolumn odpowiednio na *Tytuł* oraz *Do dnia*. Aby zapewnić, że obie kolumny właściwie wykorzystają początkowo dostępne dla nich miejsce, przypiszemy ich właściwościom Width następujące wartości: dla pierwszej kolumny 200, dla drugiej 70.



Sam komponent ListView nazwijmy entriesListView.

Należy starać się dobierać nazwy jak najkrótsze, które jednocześnie ułatwią nam zrozumienie kodu, gdy wrócimy do niego po dłuższym czasie.

Kontrolka ListView zapewnia możliwość wielokrotnego wyboru, jednak my chcemy, by w danej chwili mógł być zaznaczony tylko jeden element listy. Ponieważ wielokrotne zaznaczanie jest domyślnie wybrane, musimy przypisać właściwości MultiSelect wartość false.

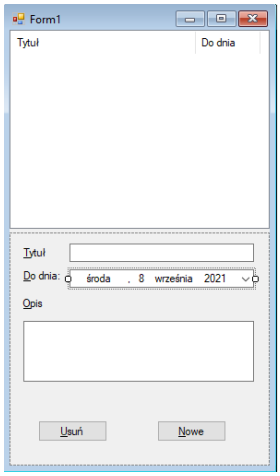
Kolejną rzeczą, jaką zrobimy, będzie dodanie pola TextBox, w którym użytkownik będzie mógł wpisać tytuł zadania, oraz towarzyszącej mu etykiety (kontrolki Label), która określi, do czego to pole służy. Właściwości Text kontrolki Label przypiszemy wartość &Tytuł:. Znak & określa klawisz skrótu.

Do formularza dodajmy także etykietę z tekstem *&Do dnia:*, a bezpośrednio za nią kontrolkę DateTimePicker oraz, dodatkowo, kolejną etykietę z tekstem *&Opis* oraz kolejnego pola TextBox (Warto zwrócić uwagę na uważne dobieranie klawiszy skrótów, tak by dla każdego pola były one inne — dla pola tytułu zadania będzie to *Ctrl+T*, dla daty *Ctrl+D*, a dla opisu *Ctrl+O*).

W przypadku podawania opisu chcielibyśmy, by użytkownik miał możliwość wpisania kilku wierszy tekstu. W tym celu zmieniamy właściwość AcceptsReturn na true (naciśnięcie klawisza *Enter* będzie obsługiwane przez pole tekstowe). Zmieńmy także właściwość Multiline na true.

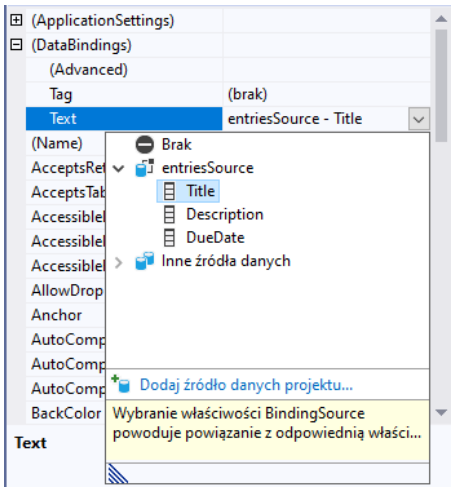
Oprócz tego będziemy potrzebowali dwóch przycisków: jednego do dodawania nowych zadań oraz drugiego do ich usuwania. Właściwościom Text tych przycisków nadamy odpowiednio wartości *&Nowe* oraz *&Usuń*.

Zmieńmy jeszcze nazwy odpowiednich kontrolek: titleText, dueDatePicker, descriptionText, newButton oraz deleteButton.



Zmieńmy w odpowiedni sposób właściwość Anchor komponentów umieszczonych na formie, tak żeby w przypadku zmiany wielkości formularza nie pojawiały się problemy wizualne.

Kolejnym krokiem będzie powiązanie naszych kontrolek z danymi. Zaznaczmy pole tekstowe służące do podawania tytułu zadania i we właściwościach wybierzmy rozwijalny element *DataBindings*. Po jego kliknięciu pojawia się lista właściwości, które możemy powiązać. Kiedy rozwiniemy listę dostępną we właściwości *Text*, zobaczymy listę dostępnych źródeł (w naszym przypadku jest dostępne tylko jedno - zdefiniowane w formularzu źródło entriesSource), które można dalej rozwinąć, by wybrać odpowiednią właściwość.



W naszej aplikacji powiążemy oba pola tekstowe formularza oraz kontrolkę do wyboru daty (w tym przypadku do zdefiniowania powiązania wykorzystamy właściwość Value) z odpowiednimi właściwościami źródła danych.

Aby sprawdzić, czy utworzone powiązanie działa, będziemy potrzebowali jakichś danych - utworzona wcześniej lista zadań jest aktualnie pusta. Dlatego napiszemy metodę pomocniczą służącą do tworzenia nowego zadania oraz dodawania go do listy i użyjemy jej zarówno do obsługi kliknięcia przycisku *Nowe*, jak i w celu utworzenia kilku przykładowych zadań podczas uruchamiania aplikacji:

```
private void CreateNewItem()
{
    ToDoEntry newEntry = (ToDoEntry)entriesSource.AddNew();
    newEntry.Title = "Nowe zadanie";
    newEntry.DueDate = DateTime.Now;
    entriesSource.ResetCurrentItem();

}
```

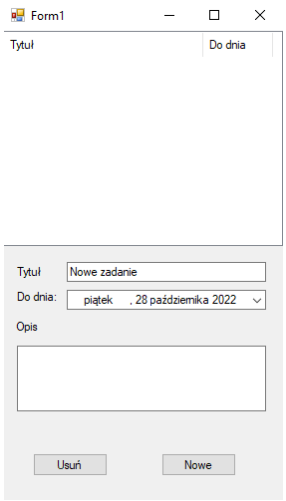
Korzystamy tu z metody AddNew źródła danych. Oznacza to, że system wiązania danych będzie świadom faktu utworzenia nowego elementu i jeśli inne kontrolki zostaną powiązane z tym źródłem, to będą poinformowane o wprowadzonej zmianie. W dalszej części kodu określamy wartości dwóch właściwości zadania.

Nasza klasa ToDoEntry nie udostępnia zdarzeń powiadamiających o zmianie właściwości, zatem sami musimy poinformować źródło danych o konieczności odświeżenia wszelkich powiązanych kontrolek. W tym celu należy wywołać metodę ResetCurrentItem.

Wywołanie naszej nowej metody musimy umieścić w konstruktorze formularza, tak by w momencie jego wyświetlania zawierał on jeden nowy element:

```
public Form1()
{
    InitializeComponent();
    entriesSource.DataSource = entries;
    CreateNewItem();

}
```

Dzięki wprowadzonym zmianom po wyświetleniu formularza zobaczymy w nim napis *Nowe zadanie* utworzony przez metodę CreateNewItem i wyświetlony w polu *Tytuł*.



W naszym programie kontrolka ListView nie pokazuje jeszcze żadnych danych. Dzieje się tak, ponieważ, nie obsługuje ona technologii wiązania danych. Kod musimy napisać samemu. Zaznaczmy komponent entriesSource, a następnie w okienku zdarzenia wybierzmy ListChanged. W naszej aplikacji będą występowały trzy rodzaje zmian listy zadań: dodanie nowego zadania oraz modyfikacja lub usunięcie istniejącego. Napiszemy zatem trzy metody obsługujące te trzy operacje. Argument zdarzenia przekazywany do procedury jego obsługi zawiera informację o rodzaju zmiany. Skorzystamy z tego i na jego podstawie wywołamy odpowiednią metodę

```
private void entriesSource_ListChanged(object sender, ListChangedEventArgs e)
    {
        switch (e.ListChangedType)
        {
            case ListChangedType.ItemAdded:
                MakeListViewItemForNewEntry(e.NewIndex);
                break;
            case ListChangedType.ItemDeleted:
                RemoveListViewItem(e.NewIndex);
                break;
            case ListChangedType.ItemChanged:
                UpdateListViewItem(e.NewIndex);
                break;
        }

    }
```

Zacznijmy od kodu obsługującego operację dodania nowego zadania. Musimy w nim utworzyć nowy obiekt ListViewItem, który następnie dodamy do listy, i upewnić się przy tym, że zawiera on dwie kolumny. Ponieważ nowy obiekt ListViewItem domyślnie zawiera jedną kolumnę, drugą musimy dodać sami. Następnie wystarczy dodać element, umieszczając go w dowolnym miejscu wskazanym przez źródło wiązania

```
private void MakeListViewItemForNewEntry(int newItemIndex)
{
    ListViewItem item = new ListViewItem();
    item.SubItems.Add("");
    entriesListView.Items.Insert(newItemIndex, item);
}
```

Umieszczając kod aktualizujący widoczną zawartość listy w procedurze obsługi zdarzenia informującego o zmianie, obsługujemy dwie sytuacje: dodanie nowego elementu oraz modyfikację istniejącego.

```
private void UpdateListViewItem(int itemIndex)
      {
          ListViewItem item = entriesListView.Items[itemIndex];
          ToDoEntry entry = entries[itemIndex];
          item.SubItems[0].Text = entry.Title;
          item.SubItems[1].Text = entry.DueDate.ToShortDateString();

      }
```

Kolejny listing przedstawia kod służący do obsługi usuniętych elementów.

```
private void RemoveListViewItem(int deletedItemIndex)
{
    entriesListView.Items.RemoveAt(deletedItemIndex);

}
```

Teraz uruchomienie aplikacji spowoduje natychmiastowe wyświetlenie tytułu oraz daty zakończenia nowo utworzonego zadania. Dodatkowo jakakolwiek modyfikacja tytułu lub daty spowoduje odpowiednią aktualizację listy.

Domyślnie mechanizm wiązania danych nie przeprowadza aktualizacji aż do momentu przejścia do innej kontrolki. W przypadku kontrolki kalendarza daje on dziwne efekty — wybór dnia wiąże się z kliknięciem, a kiedy to zrobimy, kalendarz zniknie. Konieczność wybrania jakiejś innej kontrolki, by zmiany zostały pokazane na liście, jest dosyć dziwna. Możemy zmienić ten sposób działania, konfigurując powiązania samodzielnie, gdyż dzięki temu będziemy mieli możliwość precyzyjnego określenia, kiedy dane będą przekazywane.

W tym celu w pierwszej kolejności musimy usunąć powiązania skonfigurowane przez Visual Studio - jeśli mamy zamiar utworzyć powiązania samodzielnie, nie będą nam potrzebne te przygotowane wcześniej. Aby to zrobić, należy przejść do sekcji *(DataBindings)* panelu *Właściwości*, kliknąć prawym przyciskiem myszy odpowiednią właściwość i wybrać opcję *Resetuj* z wyświetlonego menu kontekstowego. Powiązania musimy usunąć wyłącznie dla pola tytułu oraz daty zakończenia. Opis zadania jest pokazywany wyłącznie w polu tekstowym, co oznacza, że standardowy sposób jego aktualizacji w zupełności nam wystarczy, w związku z czym możemy go zostawić tak, jak jest. Następnie w konstruktorze formularza powinniśmy dodać fragment bezpośrednio za wywołaniem metody InitializeComponent.

```
public Form1()
{
    InitializeComponent();
    titleText.DataBindings.Add("Text", entriesSource, "Title", true,
                               DataSourceUpdateMode.OnPropertyChanged);
    dueDatePicker.DataBindings.Add("Value", entriesSource, "DueDate", true,
                               DataSourceUpdateMode.OnPropertyChanged);
    entriesSource.DataSource = entries;
    CreateNewItem();

}
```

Pierwsze trzy argumenty dwóch wywołań metody Add określają odpowiednio właściwość kontrolki, źródło danych oraz właściwość źródła danych, czyli wszystkie informacje potrzebne mechanizmowi wiązania. Kolejny, czwarty argument, którym w obu przypadkach jest wartość true, określa, że w razie konieczności mechanizm wiązania może odpowiednio formatować dane. Wszystkie te argumenty zapewniają dokładnie to samo, co wcześniej Visual Studio zrobiło dla nas

automatycznie. Ostatni argument metody Add. informuje, że chcemy, by powiązanie było aktualizowane za każdym razem, gdy zmieni się wartość właściwości, a nie, jak to jest domyślnie, że należy poczekać aż do usunięcia kursora z kontrolki lub zajścia innego zdarzenia, które wymusi aktualizację. Po dodaniu powyższego fragmentu kodu zmiany tytułu lub daty zakończenia zadania będą się od razu pojawiały na liście.

Aby obsłużyć zdarzenia Click naszych dwóch przycisków, należy je dwukrotnie kliknąć w projektancie formularzy. W efekcie Visual Studio doda metody obsługi zdarzeń o odpowiednich nazwach i sygnaturach, a ponadto w wygenerowanej części klasy formularza wstawi także kod służący do obsługi tych zdarzeń. Sposób obsługi kliknięcia przycisku *Nowe* jest bardzo prosty, gdyż już wcześniej napisaliśmy metodę dodającą nowy element listy:

```
private void newButton_Click(object sender, EventArgs e)
{
    CreateNewItem();

}
```

Obsługa usuwania elementów:

```
private void deleteButton_Click(object sender, EventArgs e)
{
    if (entriesListView.SelectedIndices.Count != 0)
    {
        int entryIndex = entriesListView.SelectedIndices[0];
        entriesSource.RemoveAt(entryIndex);
    }

}
```

Kontrolka ListView jest w stanie obsługiwać zaznaczanie większej liczby elementów listy jednocześnie. Początkowo wyłączyliśmy tę możliwość, wciąż jednak musimy korzystać z API zaprojektowanego pod jej kątem. Udostępnia ono kolekcję SelectedIndices zawierającą indeksy wszystkich zaznaczonych elementów. Upewniamy się, że kolekcja ta nie jest pusta, a następnie pobieramy jej pierwszy element. Następnie usuwamy obiekt za pośrednictwem źródła wiązania, dzięki czemu mechanizm wiązania danych będzie wiedział o wykonywanej operacji.

Teraz możemy już dodawać do listy kolejne elementy. Przy okazji ujawnił się kolejny brak w naszym programie — nie zrobiliśmy jeszcze nic, by zapewnić, że po wybraniu jednego z elementów listy wartości jego właściwości zostaną wyświetlone w odpowiednich polach formularza. W tym celu musimy dodać metodę obsługi zdarzenia SelectedIndexChanged kontrolki listy. Jest to jej zdarzenie domyślne, a zatem by dodać metodę jego obsługi, wystarczy dwukrotnie kliknąć listę w projektancie formularzy. Jedyną operacją, jaką musimy wykonać w tej metodzie, jest odpowiednie ustawienie właściwości Position źródła danych:

```
private void entriesListView_SelectedIndexChanged(object sender, EventArgs e)
  {
      if (entriesListView.SelectedIndices.Count != 0)
      {
          int entryIndex = entriesListView.SelectedIndices[0];
          entriesSource.Position = entryIndex;
      }

  }
```

Proszę uzupełnić aplikację o możliwość zapisania listy do pliku oraz odczytania listy danych z pliku.