Лекция 12. Диалоговые окна. Расширенные возможности Windows Forms

Диалоговые окна

Диалоговые окна обычно используются для сбора сведений от пользователя приложения. Visual Studio, например, предоставляет предварительно подготовленные диалоговые окна, дающие возможность пользователю выбирать файл, шрифт или цвет. Для получения специализированных сведений от пользователя существует возможность создавать специализированные диалоговые окна. Например, вы можете создать диалоговое окно, собирающее информацию о пользователе и передающее ее в основную форму приложения.

Обычно диалоговое окно содержит кнопки ОК и Отмена, а также любые элементы управления, требуемые для сбора сведений от пользователя. Как правило, кнопка ОК принимает предоставленные пользователем сведения и закрывает форму, возвращая результат `DialogResult.OK`. Обычным поведением кнопки Отмена является отклонение пользовательского ввода, после чего она закрывает форму, возвращая результат `DialogResult.Cancel`.

Visual Studio предоставляет для диалоговых окон готовый шаблон. Значение свойству `DialogResult` для кнопок ОК и Отмена присваивается в окне свойств. В общем случае для кнопки ОК следует присвоить `OK`, а для кнопки Отмена — `Cancel`.

Диалоговые окна, (а также другие формы) бывают *модальными* и *немодальными*. Модальное диалоговое окно приостанавливает выполнение программы, пока оно не будет закрыто. Немодальное диалоговое окно, наоборот, позволяет приложению продолжить выполнение.

Используя метод `Form.Show`, диалоговое окно можно отобразить немодально:

```
DialogForm aDialog = new DialogForm();
aDialog.Show();
```

При немодальном отображении диалоговое окно не возвращает результат и не останавливает выполнение программы, но посредством проверки значений свойств формы остается возможность извлекать из неё сведения.

При вызове метода `Form.ShowDialog` диалоговое окно можно показать модально:

```
DialogForm aDialog = new DialogForm();
aDialog.ShowDialog();
```

Метод `ShowDialog` возвращает значение `DialogResult` кнопки, которая была щёлкнута для закрытия формы. Если свойство `DialogResult` кнопки не установлено, возвратится `DialogResult.None`. Следующий пример показывает, как возвратить `DialogResult` диалогового окна:

```
DialogForm aDialog = new DialogForm();
DialogResult aResult = aDialog. ShowDialog();
if (aResult == DialogResult.OK)
{
    // Действия с данными из диалогового окна
}
```

С помощью метода `ShowDialog` можно также настроить свойство `ParentForm` диалогового окна, указав родительскую форму в качестве параметра, как показано в следующем примере:

```
DialogForm aDialog = new DialogForm();
DialogResult aResult = aDialog. ShowDialog(this);
```

Когда родительская форма задана, из неё можно извлечь сведения посредством приведения типа свойства `ParentForm` к подходящему типу и дальнейшего чтения его свойств, как показано в следующем примере. Пример предполагает экземпляр диалогового окна с именем `DialogForm` и то, что свойство `ParentForm` установлено на экземпляр формы с именем `Form1`:

```
Form1 aForm = (Form1)DialogForm.ParentForm;
```

Когда диалоговое окно закрыто, оно больше невидимо для пользователя, но его экземпляр остаётся в памяти. Следовательно, из этого диалогового окна можно извлечь пользовательский ввод.

В общем случае, пользовательский ввод диалогового окна должен предоставляться через свойства этого диалогового окна. Например, диалоговое окно, дающее возможность пользователю вводить имя, должно предоставлять эти данные через свойство `FirstName`. Следующий пример показывает создание для диалогового окна свойства только для чтения с именем `FirstName`, принимающего пользовательский ввод из `TextBox` с именем `txtFirstName`:

```
public string FirstName
{
    get { return txtFirstName.Text; }
}
```

Теперь для извлечения имени пользователя можно прочитать свойство диалогового окна:

```
string name = DialogBox.FirstName;
```

Drag-and-Drop

Перетаскивание (*Drag-and-Drop*) означает возможность, нажав левую кнопку мыши, захватить данные, например строку или объект, переместить мышь с нажатой левой кнопкой к другому элементу управления, который может принять данные, а затем отпустить кнопку мыши для передачи данных. Такая функциональность реализуется главным образом при обработке событий.

В программировании Windows Forms функциональность перетаскивания является повсеместной. Она означает возможность для пользователя захватить мышью данные, например текст, изображение или другой объект, и перетащить их в другой элемент управления. Когда кнопка мыши над ним отпускается, перетаскиваемые данные помещаются в этот элемент управления, после чего могут быть выполнены различные действия.

Перетаскивание похоже на вырезание и вставку. Когда указатель мыши располагается над элементом управления и нажимается кнопка мыши, данные копируются из исходного элемента управления. Когда кнопка мыши отпускается, выполняется действие вставки. Программировать необходимо весь код для копирования данных из исходного элемента управления и любых действий в целевом элементе управления.

Процесс перетаскивания управляется главным образом событиями. В исходном элементе управления при перетаскивании происходят следующие события:

- `MouseDown` — происходит, когда нажимается кнопка мыши и указатель находится над этим элементом управления. Обычно в методе, обрабатывающем это событие, вызывается метод `DoDragDrop`;
- `GiveFeedBack` — предоставляет пользователю возможность установить специальный указатель мыши;
- `QueryContinueDrag` — позволяет источнику перетаскивания отменить событие перетаскивания.

А эти события происходят в целевом элементе управления:

- `DragEnter` — происходит, когда объект перетащен в границы элемента управления. Обработчик этого события получает объект `DragEventArgs`;
- `DragOver` — происходит, когда объект перетаскивается над целевым элементом управления. Обработчик этого события получает объект `DragEventArgs`;
- `DragDrop` — происходит, когда отпускается кнопка мыши над целевым элементом управления. Обработчик этого события получает объект `DragEventArgs`;
- `DragLeave` — происходит, когда объект перетаскивается за границы элемента управления.

Кроме того, для инициализации процесса перетаскивания требуется метод `DoDragDrop` исходного элемента управления, а у целевого элемента управления должно быть установлено в `true` свойство `AllowDrop`.

При перетаскивании события происходят в следующем порядке:

1. Инициализируется операция перетаскивания вызовом метода `DoDragDrop` исходного элемента управления. Обычно это делается в обработчике события `MouseDown`. `DoDragDrop` копирует требуемые дан-

ные из исходного элемента управления в новый экземпляр DataObject и устанавливает флажки, указывающие разрешенные с этими данными действия.

2. В этот момент происходят события GiveFeedBack и QueryContinueDrag. Обработчик события GiveFeedback может установить указатель мыши особого внешнего вида, а обработчик события QueryContinueDrag может использоваться для указания продолжения или отмены операции перетаскивания.
3. Указатель мыши перетаскивается к целевому элементу управления. Возможной целью является любой элемент управления, свойство AllowDrop которого установлено в true. Когда указатель мыши входит в границы такого элемента управления, для него генерируется событие DragEnter. Объект DragEventArgs, получаемый обработчиком этого события, проверяется на присутствие данных, подходящих целевому элементу управления. Если они есть, свойству Effect объекта DragEventArgs присваивается подходящее значение.
4. Когда пользователь отпускает кнопку мыши над допустимым целевым элементом управления, генерируется событие DragDrop. В коде обработчика этого события принимаются перетащенные данные и выполняются любые действия, подходящие целевому элементу управления.

Для выполнения операции перетаскивания действие, указанное в методе DoDragDrop, должно соответствовать значению параметра Effect объекта DragEventArgs, связанного с событием перетаскивания. Это значение обычно присваивается в событии DragDrop. Свойство Effect является экземпляром перечисления DragDropEffects со следующими значениями:

- All — данные копируются и удаляются из источника перетаскивания и вписываются в цель;
- Copy — данные копируются в цель;
- Link — данные связываются с целью;
- Move — данные перемещаются в цель;
- None — цель не принимает данные;
- Scroll — прокрутка подготавливается или происходит в цели.

Обратите внимание, что основной функцией параметра Effect является изменение указателя мыши, когда он находится над целевым элементом управления. Значение этого параметра фактически не влияет на выполняемое действие за исключением случая, когда Effect присвоено None, — тогда целевой элемент управления не будет принимать данные, так как не будет генерироваться событие DragDrop.

Операция перетаскивания инициализируется при вызове метода DoDragDrop исходного элемента управления. Этот метод имеет два параметра: объект, предоставляющий копируемые в DataObject данные, и экземпляр DragDropEffects, указывающий разрешённые с этими данными действия перетаскивания. Следующий пример показывает, как скопировать текст из поля и установить разрешенные действия Copy и Move:

```
private void textBox1_MouseDown(object sender, MouseEventArgs e)
{
    textBox1.DoDragDrop(textBox1.Text, DragDropEffects.Copy | DragDropEffects.Move);
}
```

Событие DragEnter должно быть обработано для каждого целевого элемента управления. Оно происходит, когда идет операция перетаскивания и указатель мыши входит в границы элемента управления. Это событие передает обрабатывающему его методу объект DragEventArgs, который используется для обращения к DataObject, связанному с операцией перетаскивания. Если данные соответствуют целевому элементу управления, свойству Effect можно присвоить подходящее для элемента управления значение. Следующий пример показывает, как проверить формат данных DataObject и установить свойство Effect.

```
private void textBox2_DragEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(DataFormats.Text))
        e.Effect = DragDropEffects.Copy;
}
```

Когда в ходе операции перетаскивания нажатая кнопка мыши отпускается над целевым элементом управления, генерируется событие DragDrop. В методе, обрабатывающем событие DragDrop, можно использовать метод GetData в DataObject для извлечения из DataObject скопированных данных и выполнить любые подходящие целевому элементу управления действия. Следующий пример показывает, как поместить строку в TextBox.

```
private void textBox2_DragDrop(object sender, DragEventArgs e)
{
    textBox2.Text = (string)e.Data.GetData(DataFormats.Text);
}
```

Поддержка операций перетаскивания между приложениями .NET Framework встроена в систему. Чтобы совершать подобные операции, не нужно предпринимать никаких дополнительных действий, достаточно обеспечить выполнение следующих условий:

- целевой элемент управления должен разрешать одно из действий перетаскивания, указанных при вызове метода DoDragDrop;
- целевой элемент управления должен принять данные в формате, установленном при вызове метода DoDragDrop.

Реализация форм MDI

Приложения MDI организуют дочерние формы в одной родительской форме. В отличие от приложений однодокументного интерфейса, в которых можно работать только над одним документом за раз, приложения MDI позволяют открывать, организовывать и обрабатывать несколько документов одновременно.

Приложения MDI придерживаются модели «родительская форма/дочерняя форма». Обычно это приложение имеет одну родительскую форму (хотя для него возможно наличие нескольких родительских форм), которая содержит и организует несколько дочерних форм. В Microsoft Office Excel — примере приложения MDI — в границах родительской формы можно открыть несколько документов и по отдельности работать с ними. Родительская форма организует и упорядочивает все дочерние документы, открытые в настоящий момент.

Родительская форма — основная для любого приложения MDI. Она содержит все дочерние формы, с которыми взаимодействует пользователь, а также управляет компоновкой и организацией дочерних форм. Для указания того, что форма будет являться родительской, необходимо установить её свойство IsMDIContainer в true.

Дочерние формы в приложениях MDI являются центром пользовательского взаимодействия. Они предоставляют пользователю данные и обычно содержат самостоятельные документы. Дочерние формы находятся внутри родительской формы, управляются ею и создаются посредством своего свойства MdiParent. Следующий код создаёт дочернюю форму по отношению к текущей:

```
ChildForm aChildForm = new ChildForm();
aChildForm.MdiParent = this;
aChildForm.Show();
```

Во время работы многодокументного приложения возникает необходимость идентифицировать активную дочернюю форму. Например, общей чертой приложений MDI является наличие в главном меню родительской формы команд, воздействующих на дочернюю форму, имеющую фокус. Для получения ссылки на форму, использованную последней, применяется свойство ActiveMDIChild родительской формы. Следующий пример кода показывает, как получить ссылку на активную дочернюю форму:

```
Form aForm;
aForm = this.ActiveMDIChild;
```

Когда активная форма MDI идентифицирована, её свойства можно использовать для передачи данных из буфера обмена активному элементу управления формы. Эту функциональность можно использовать для

реализации команды меню Paste. В следующем примере кода проверяется, является ли активный элемент управления полем, и в этом случае в него вставляется текст из буфера обмена.

```
Form activeForm = this.ActiveMDIChild;
if (activeForm != null)
    if (this.ActiveControl.GetType() is TextBox)
     {
        TextBox aTextBox = (TextBox)this.ActiveControl;
        IDataObject data = Clipboard GetDataOb]ect();
        if (data.GetDataPresent(DataFormats.Text))
            aTextBox.Text = data.GetData(DataFormats.Text).ToString();
     }
```

Родительская форма MDI также может упорядочить содержащиеся в ней дочерние формы посредством вызова метода `LayoutMdi`. Этот метод имеет параметр, являющийся членом перечисления `MdiLayout`. Метод `LayoutMdi` заставляет формы, содержащиеся в родительской, упорядочиваться указанным в этом параметре способом. Перечисление `MdiLayout` содержит следующие значения:

- `ArrangeIcons` — все дочерние значки MDI упорядочиваются в пределах клиентской области родительской формы MDI;
- `Cascade` — все дочерние окна MDI упорядочиваются каскадом в пределах клиентской области родительской формы MDI;
- `TileHorizontal` — все дочерние окна MDI упорядочиваются сверху вниз в пределах клиентской области родительской формы MDI;
- `TileVertical` — все дочерние окна MDI упорядочиваются слева направо в пределах клиентской области родительской формы MDI.

Приложения MDI часто содержат меню списка всех окон приложения в настоящий момент. Пользователи могут выбрать подходящее окно из списка, которое будет активизировано в родительской форме MDI. Visual Studio делает реализацию меню списка окон для приложения MDI простой задачей. Сначала нужно создать для меню списка окон элемент меню верхнего уровня. Например, можно создать элемент с именем `WindowToolStripMenuItem`. Затем в конструкторе выбрать компонент `MenuStrip` и указать в свойстве `MdiWindowListltem` пункт меню, созданный для списка окон. Меню автоматически заполнится списком дочерних форм, а при выборе строки из этого списка будет активизироваться соответствующая дочерняя форма.