Лекция 13. Создание элементов управления Windows Forms

Элементы управления - это компоненты, имеющие видимый пользовательский интерфейс и способные к взаимодействию с пользователем во время выполнения. Microsoft .NET Framework предоставляет множество элементов управления, инкапсулирующих широкий спектр функциональности. Это, например, элементы управления `Button`, `TextBox` и `DateTimePicker`. В дополнение к уже существующим элементам управления можно разрабатывать собственные, чтобы обеспечить для своих приложений специализированную функциональность. Существует три вида разрабатываемых пользователем элементов управления: составные, которые создаются при объединении других элементов управления Windows Forms; специализированные, создаваемые с нуля и предоставляющие собственный код для прорисовки; и расширенные, которые добавляют функциональность к уже существующему элементу управления Windows Forms.

Создание составных элементов управления

Составные элементы управления — самая простая форма разрабатываемых пользователем элементов управления. Конструктор составного элемента управления содержит подобный форме графический интерфейс, позволяющий добавлять уже существующие элементы управления и компоненты, которые затем связываются вместе в один функциональный элемент.

Составные элементы управления (также известные как пользовательские элементы управления) в точности соответствуют своему названию: это элементы управления, составленные из других элементов управления. Составные элементы управления наследуются от класса `UserControl`. Он предоставляет базовый уровень функциональности, обеспечивающий добавление других элементов управления, а также свойств, методов и событий. Класс `UserControl` имеет собственный конструктор, позволяющий использовать в среде Visual Studio перетаскивание дополнительных элементов управления из Панели инструментов на поверхность конструктора и настраивать их.

Добавление функциональности к элементам управления

Составляющими элементами управления называют подчинённые элементы управления, включённые в составной. Составляющие элементы управления добавляются к своему составному элементу управления таким же образом, как добавляются элементы управления к форме — перетаскиванием их на поверхность конструктора из Панели инструментов. Настраиваются эти составляющие элементы тоже, как в форме, — можно присваивать значения их свойствам, изменять внешний вид и создавать методы, обрабатывающие события элемента управления. Когда составной элемент управления создан, в него будет встроена функциональность написанного вами кода.

В дополнение к добавлению составляющих элементов управления можно также для дополнительной функциональности к элементу управления в форме добавить методы, свойства и события.

Метод добавляется к элементу управления таким же образом, как он добавляется к форме или любому другому классу. Внутри объявления класса в окне кода нужно добавить объявление метода и его тело.

Добавление свойства похоже на добавление метода. Создаётся определение свойства и затем реализуется функциональность, требуемая для возвращения и присвоения значения, предоставляемого свойством. Обычно лежащее в основе свойства значение находится в закрытом поле.

К элементу управления можно добавить события, которые будут генерироваться для уведомления остальной части приложения о том, что случилось нечто интересное. Когда к классу или элементу управления добавлено событие, его можно вызвать в коде для отправления уведомления остальной части приложения. Язык C# требует явного наличия делегата для указания сигнатуры до использования ключевого слова `event` для создания нового события. Генерируется событие в коде простым вызовом события по типу метода в C#.

Предоставление свойств составляющих элементов управления

Когда составляющие элементы управления добавляются к составному элементу управления, им по умолчанию задается уровень доступа `private`. Составляющие элементы управления недоступны классам в других сборках. Если вы хотите позволить другим сборкам настраивать составляющие элементы управления, следует предоставить их свойства, обратив в объявление свойства составного элемента управления и затем создав в этом свойстве код для возвращения и присвоения значения свойства составляющего элемента

управления. Предположим, что вы хотите предоставить свойство BackColor составляющего элемента управления Button. Вы можете создать в составном элементе управления свойство с именем ButtonBackColor, через которое в методе get возвращается значение свойства BackColor, а в методе set значение присваивается:

```
public System.Drawing.Color ButtonBackColor
{
    get { return Button1.BackColor; }
    set { Button1.BackColor = value; }
}
```

Иногда может понадобиться сделать свой элемент управления невидимым во время выполнения. Невидимый элемент управления создается установкой свойства Visible в false. Невидимые элементы управления не могут взаимодействовать с пользователем через пользовательский интерфейс, но они продолжают взаимодействовать с приложением и другими элементами управления. Обратите внимание, что значение свойству Visible может быть присвоено только во время выполнения. Чтобы гарантировать невидимость элемента управления при запуске, устанавливайте свойство Visible в обработчике события Load этой формы.

Настройка прозрачности фона элемента управления

Существует два типа прозрачности, которые можно настроить для фона вашего элемента управления. Прозрачность может быть такой, что сквозь фон элемента управления будет видна расположенная под ним форма. При настройке второго типа прозрачности элемент управления будет выглядеть, как прозрачное окно в форме, через которое видно все, что находится на рабочем столе под этой формой.

Чтобы создать элемент управления с прозрачным фоном, достаточно установить свойство BackColor в значение Transparent. Всё, что отображается в форме под этим элементом управления, будет показано сквозь его фон.

Создание прозрачного элемента управления, действующего как окно в форме, немного сложнее. Каждая форма имеет свойство с именем TransparencyKey, указывающее цвет, который будет прозрачным, когда он присутствует в форме. Установив свойство BackColor элемента управления в тот же цвет, что указан в свойстве TransparencyKey формы, можно создать в форме прозрачное окно.

Предоставление точечного рисунка элемента управления для Панели инструментов

После того как элемент управления создан, он автоматически появляется в Панели инструментов, если вы используете этот элемент управления в том же решении, которое его содержит. Элемент управления может быть добавлен в Панель инструментов и при создании в другом проекте.

Когда элемент управления добавляется в Панель инструментов, в нём отображается имя этого элемента управления и значок. Если не определён специальный значок, предоставляется универсальный. Значок, отображаемый рядом с именем вашего элемента управления, указывается с помощью класса ToolboxBitmapAttribute. Его экземпляр присоединяется к объявлению элемента управления и используется для указания точечного рисунка 16 на 16 пикселов, которым будет представляться элемент управления в Панели инструментов.

Точечный рисунок для можно указать тремя различными способами. Простейшим является указание пути к точечному рисунку, который вы хотите использовать:

```
[ToolboxBitmap(@"C:\myToolboxBitmap.bmp")]
class myControl : UserControl
{
}
```

Также можно использовать точечный рисунок уже существующего типа элемента управления. Например, с помощью следующего кода указывается использование того же ToolboxBitmap, что и для элемента управления Button:

```
[ToolboxBitmap(GetType(System.Windows.Forms.Button))]
class myControl : UserControl
```

```
{
}
```

Наконец, можно указать сборку и определённый в ней тип и затем загрузить ресурс значка, указанный строковым именем:

```
[ToolboxBitmap(GetType(myControl), "myControl.bmp")]
class myControl : UserControl
{
}
```

Создание специализированных элементов управления

Специализированные элементы управления обеспечивают самый высокий уровень конфигурируемости и настройки любого из элементов управления, но также требуют наибольшего времени для разработки. Для специализированных элементов управления не существует пользовательского интерфейса по умолчанию, и они должны предоставлять весь код, необходимый для отображения своего графического представления. Кроме того, конструктор поддерживает специализированные элементы управления ограниченно, давая вам возможность добавлять компоненты из Панели инструментов, но не обеспечивая никакого графического проектирования. Из-за этих проблем специализированные элементы управления могут быть самым трудным типом элементов управления для разработки, но являются также лучшим выбором, когда вы хотите создать элемент управления с особенно сложным визуальным представлением.

Специализированные элементы управления обеспечивают самый высокий уровень конфигурируемости при разработке элементов управления. Специализированный элемент управления разрабатывается для того, чтобы получить точно такое визуальное представление, которого вам необходимо, и закодировать любую требуемую функциональность для взаимодействия с пользователем.

Конструктор специализированного элемента управления значительно менее детализирован, чем пользовательского. У специализированных элементов управления нет никакого внешнего вида по умолчанию, поэтому их конструктор — просто пустое серое окно. Из Панели инструментов в этот конструктор можно перетаскивать компоненты и включать их функциональность в ваш элемент управления. Технически, существует возможность перетаскивать в конструктор также элементы управления и включать их в специализированный элемент управления, но они не будут отображаться как его часть. Если вы хотите включить уже существующие элементы управления в ваш, создайте пользовательский элемент управления, как описано выше.

Специализированные элементы управления наследуются от класса `Control`. Этот класс обеспечивает функциональность, требуемую для элемента управления. Он предоставляет базовую функциональность, необходимую для взаимодействия элемента управления с остальной частью приложения. Например, класс `Control` позволяет элементу управления обнаружить присутствие мыши и предоставляет общие события, такие как `Click`. Этот класс включает также свойства, полезные для задания пользовательского интерфейса, например `ForeColor`, `BackColor`, `Visible`, `Location`. Однако класс `Control` не предоставляет никакой функциональности, специфической для элемента управления.

Ключевой задачей при разработке специализированного элемента управления является реализация видимого пользовательского интерфейса. Его можно создать реализацией метода `OnPaint`, вызываемого каждый раз, когда элемент управления отображается на экране. Она должна содержать код, требуемый для прорисовки элемента управления. Для реализации метода `OnPaint` необходимо использовать графические классы .NET Framework.

Введение в пространства имён `System.Drawing`

Пространства имён `System.Drawing` предоставляют значительное количество графической функциональности. Все графические классы разделены на несколько пространств имён:

- `System.Drawing` — большинство классов, участвующих в отображении графического содержимого на экране;
- `System.Drawing.Design` — классы, обеспечивающие дополнительную функциональность для графических операций времени разработки;

- System.Drawing.Drawing2D — классы, предназначенные для отображения двумерных эффектов и сложных фигур;
- System.Drawing.Imaging — классы, облегчающие управление изображениями и их отображение;
- System.Drawing.Printing — классы, участвующие в печати содержимого;
- System.Drawing.Text — классы, облегчающие управление шрифтами.

Большинство классов, которые вы будете использовать для отображения графики элемента управления, предоставляются пространствами имён System.Drawing и System.Drawing.Drawing2D.

Класс Graphics — это основной класс, участвующий в отображении графики. Экземпляр класса Graphics предоставляет поверхность для рисования визуального элемента, например формы или элемента управления. Этот класс инкапсулирует интерфейс между .NET Framework и системой визуализации графики и используется для отображения всей графики, представляющей визуальный элемент.

Непосредственно создать объект Graphics невозможно, поскольку он должен быть связан с визуальным элементом. Вместо этого следует получить ссылку на объект Graphics от визуального элемента, который им владеет. Классы, унаследованные от Control (включая Form и созданные вами специализированные элементы управления) предоставляют метод CreateGraphics, возвращающий ссылку на связанный с этим элементом управления объект Graphics.

Класс Graphics предоставляет несколько методов, которые используются для отображения графики на его поверхности рисования. Эти методы делятся на используемые для рисования контуров и залитых фигур. Каждый из этих методов имеет различный набор параметров, определяющих точки координат и положение рисуемых фигур. Каждый метод для выполнения отображения требует объект. Для контуров это Pen, для залитых фигур — Brush.

Объект Pen, по сути, является пером и требуется для рисования контуров. Как и реальное перо, объект Pen позволяет рисовать линию указанной ширины и цвета. Если ширина не указана, она будет составлять один пиксел:

```
Pen myPen1 = new Pen(Color.Tomato);
Pen myPen3 = new Pen(Color.Tomato, 3);
```

Объекты Brush является кистью и используется для заливки фигур и текста. Объект Brush требуется для любого из методов класса Graphics, рисующих залитые фигуры. Хотя существует несколько различных классов Brush, которые могут использоваться для отображения сложных визуальных эффектов, чаще всего вы будете использовать SolidBrush для рисования залитых фигур чистого цвета:

```
SolidBrush myBrush = new SolidBrush(Color.Lime);
```

Перья и кисти представляют цвета, которые используются системой для отображения пользовательского интерфейса. Они создаются с помощью обращения к перечислениям SystemPens и SystemBrushes. Эти перечисления могут быть полезны при использовании системных настроек, а также при проектировании для специальных потребностей, чтобы гарантировать возможность использования режима высокой контрастности. Следующий пример кода показывает получение ссылки на SystemPens и SystemBrushes:

```
Pen myPen = SystemPens.Control;
Brush myBrush = SystemBrushes.Control;
```

Методы, предоставляемые объектом Graphics, можно использовать для отображения множества простых фигур. Все методы, рисующие контуры фигур, требуют объект Pen. Аналогично все методы, рисующие залитые фигуры, требуют объект Brush. Кроме того, необходимо предоставлять любые другие требуемые методом параметры, например координаты или другие объекты. Указываемые координаты находятся в системе координат поверхности рисования, представляемой объектом Graphics. Например, если используемый объект Graphics является элементом управления, координата (0, 0) представляет левый верхний угол этого элемента управления. Обратите внимание, что эти координаты не зависят от положения элемента управления,

представленного в свойстве Location, поскольку оно определяет положение левого верхнего угла элемента управления в системе координат его контейнера. Следующий пример показывает отображение залитого эллипса с использованием объекта Graphics формы:

```
SolidBrush myBrush = new SolidBrush(Color.PapayaWhip);
Graphics g = this.CreateGraphics();
Rectangle myRectangle = new Rectangle(0, 0, 10, 30);
g.FillEllipse(myBrush, myRectangle);
g.Dispose();
myBrush.Dispose();
```

Имейте в виду, что всегда следует вызывать Dispose для ваших объектов Pen, Brush и Graphics, потому что они используют системные ресурсы, и если от этих объектов быстро не избавиться, производительность ухудшится.

Объект Graphics предоставляет метод с именем DrawString, который может использоваться для отображения текста. Для этого нужно указать шрифт, а также положение левого верхнего угла текста и объект Brush. Следующий пример показывает отображение текста в форме с использованием метода DrawString:

```
Graphics g = this.CreateGraphics();
string myString = "Hello World!";
Font myFont = new Font("Times New Roman", 36, FontStyle.Regular);
// Последние два параметра являются координатами X и Y
// левого верхнего угла отображаемой строки
g.DrawString(myString, myFont, SystemBrushes.Highlight, 20, 20);
g.Dispose();
```

Отображение специализированных элементов управления с помощью переопределения метода OnPaint

Визуальный интерфейс специализированного элемента управления можно отобразить с помощью переопределения метода OnPaint. Внутри этого метода генерируется событие Paint и содержится весь код, требуемый для отображения визуального представления элемента управления.

Метод OnPaint имеет один параметр — экземпляр PaintEventArgs, который содержит два важных члена. Свойство ClipRectangle указывает прямоугольник, в котором будет происходить прорисовка. Свойство Graphics содержит экземпляр класса Graphics, предоставляющего поверхность рисования для отображаемого элемента управления.

Когда элемент управления рисуется или обновляется, это делается только для той части элемента управления, которая в этом нуждается. Если должен быть обновлен весь элемент управления, объект ClipRectangle предоставит размер всего элемента управления. Однако если должна быть обновлена лишь часть элемента управления, ClipRectangle предоставит только регион, который должен быть перерисован. В основном вы, как разработчик, не должны использовать свойство ClipRectangle — оно используется объектом Graphics автоматически.

Объект Graphics предоставляет поверхность рисования для элемента управления. Используя методы, описанные в предыдущем разделе, можно отобразить визуальное представление элемента управления. Все методы отображения графики требуют координат ее положения. Левый верхний угол элемента управления имеет координаты (0, 0). Ограничивается элемент управления в соответствии со свойствами Control.Width и Control.Height. Следующий пример показывает переопределение метода OnPaint и отображение залитого синим цветом прямоугольника, заполняющего весь элемент управления:

```
protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e); Graphics g = e.Graphics;
    g.FillRectangle(Brushes.Blue, 0, 0, this.Width, this.Height);
}
```

Обратите внимание на вызов `base.OnPaint`. При переопределении метода для вызова любой базовой реализации обычно следует вызывать метод базового класса в унаследованном классе.

Для создания специализированного элемента управления необходимо создать класс, унаследованный от Control, переопределить его метод OnPaint, отвечающий за отображение элемента, и добавить к элементу необходимую функциональность.

Создание расширенных элементов управления

Элементы управления могут расширять функциональность других элементов управления. К уже существующим элементам управления можно добавлять свойства и методы, а в некоторых случаях даже обеспечивать для стандартного элемента управления другое визуальное представление (например, круглый элемент управления `Button`).

Расширенные элементы управления — это разработанные пользователем элементы управления, расширяющие уже существующий элемент управления .NET Framework. Таким образом сохраняется вся функциональность существующих элементов управления, но при этом добавляются свойства и методы, а при необходимости изменяется и внешнее представление элемента управления.

Расширенный элемент управления создается посредством создания класса, наследующего расширяемый элемент управления. Следующий пример показывает создание элемента управления, наследующего класс `Button`:

```
public class ExtendedButton : System.Windows.Forms.Button
{
}
```

Класс `ExtendedButton`, созданный в этом примере, имеет тот же внешний вид, поведение и свойства, что и класс `Button`, но теперь эту функциональность можно расширить, добавляя специализированные свойства или методы. Так, далее показано добавление свойства с именем `ButtonValue`, возвращающего целое число:

```
public class ExtendedButton : System.Windows.Forms.Button
{
    public int ButtonValue { get; set; }
}
```

Помимо добавления новых методов и свойств к вашему элементу управления можно также предоставить новую реализацию для существующих методов с помощью их переопределения. Это позволит вам заменить основную реализацию метода своей собственной или добавить дополнительную функциональность к уже существующей. Далее показано переопределение метода `OnClick` в классе, наследуемом от `Button`. Новая реализация инкрементирует переменную с именем `Clicks` и затем вызывает основную реализацию `OnClick`.

```
protected override void OnClick(System.EventArgs e)
{
    Clicks++;
    base.OnClick(e);
}
```

Для некоторых элементов управления изменить визуальное представление можно при переопределении метода `OnPaint`. Это позволяет или добавлять, или заменять отображающую логику элемента управления. Чтобы добавить для элемента управления отображение по умолчанию, следует вызвать метод `base.OnPaint` для выполнения кода отображения из базового класса в дополнение к вашему собственному. Для обеспечения элементу управления специализированного внешнего вида нужно опустить вызов метода `OnPaint` базового класса. Следующий пример показывает создание простой эллиптической кнопки. Обратите внимание, однако, что изменяется только фигура элемента управления и не затрагиваются более тонкие задачи отображения, например оконтуривание:

```
protected override void OnPaint(System.Windows.Forms.PaintEventArgs pevent)
{
    System.Drawing.Drawing2D.GraphicsPath x = new System.Drawing.Drawing2D.GraphicsPath();
    x.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new Region(x);
    base.OnPaint(pevent);
}
```

Расширенные элементы управления сторонних разработчиков

Кроме стандартных элементов управления, которые входят в состав .NET Framework, существует огромное количество элементов управления сторонних разработчиков. Эти элементы имеют дополнительную функциональность, которая может пригодиться для создания современных приложений.

Дополнительные элементы и описание принципов их использования можно найти в Интернете. Среди многочисленных библиотек элементов можно выделить два коммерческих продукта, содержащих большое количество разнообразных элементов, которые используются во многих существующих приложениях.

Компания **Developer Express** занимается исследованиями в области программных технологий, разработкой компонент и дополнений для Delphi (CBuilder), VB/VC++ и платформы .Net. Набор компонентов **DevExpress** содержит более 120 элементов Windows Forms, а также компоненты для других платформ. Ознакомиться с компонентами и примерами их использования можно на сайте http://www.devexpress.com.

Также большую популярность получил набор компонентов **Component One Studio**. Как и DevExpress, Component One Studio даёт возможности для быстрого создания мощнейших таблиц-сеток, всевозможных графических отчётов и диаграмм, обработки пользовательских данных и улучшения пользовательского интерфейса приложений на базе широкого спектра технологий. Официальный сайт: http://componentone.com.