**Лабораторная работа №5.**

Цель: обучиться работе с WinForms, получить навыки разработки ПО согласно спецификации.

Краткие теоретические сведения.

Со времени появления платформы .NET (примерно в 2001 г.) среди библиотек базовых классов появился API по имени Windows Forms, представленный в основном сборкой System.Windows.Forms.dll. Инструментальный набор Windows Forms предоставляет типы, необходимые для построения графических пользовательских интерфейсов для настольных компьютеров, создания специализированных элементов управления, управления ресурсами (например, строками и значками) и выполнения других задач, возникающих при программировании для пользовательских компьютеров. Имеется и дополнительный API по имени GDI+ (представленный сборкой System.Drawing.dll), который предоставляет дополнительные типы, позволяющие программисту генерировать двухмерную графику, взаимодействовать с сетевыми принтерами и обрабатывать графические данные.

Windows Forms (и GDI+) применяются в платформе .NET 4.0 и, видимо, будут существовать еще некоторое время (возможно, длительное) в составе библиотеки базовых классов. Правда, после выхода .NET 3.0 компания Microsoft выпустила совершенно новый инструментальный API под названием Windows Presentation Foundation (WPF).

**Пространства имен Windows Forms**

Windows Forms API состоит из сотен типов (классов, интерфейсов, структур, перечислений и делегатов), большинство из которых организованы в различные пространства имен сборки System.Windows.Forms.dll.

Несомненно, наиболее важным пространством имен Windows Forms является System.Windows.Forms. Типы из этого пространства имен можно разбить на следующие крупные категории:

* Базовая инфраструктура. Это типы, представляющие базовые операции программ, которые используют Windows Forms (Form и Application), и различные типы, предназначенные для взаимодействия с устаревшими элементами ActiveX, a также для взаимодействия с новыми специальными элементами управления WPF
* Элементы управления. Эти типы применяются для создания графических пользовательских интерфейсов (наподобие Button, MenuStrip, ProgressBar и DataGridView), все они являются производными от базового класса Control. Элементы управления допускают настройку на этапе проектирования и видимы (по умолчанию) во время выполнения.
* Компоненты. Это типы, которые не порождены от базового класса Control, но все-таки могут предоставлять программам Windows Forms визуальные возможности (например, ToolTip и ErrorProvider). Многие компоненты (к примеру, Timer и System.ComponentModel.BackgroundWorker) не видимы во время выполнения, но все-таки допускают настройку на этапе проектирования.
* Окна стандартных диалогов. В Windows Forms имеется несколько заготовленных диалоговых окон для распространенных операций (например, OpenFileDialog, PrintDialog и ColorDialog).

Общее количество типов в пространстве имен System.Windows.Forms существенно превышает 100, и дополнительную информацию по ним можно прочитать в документации .NET Framework 4.0 SDK.

**Шаблон проектов Windows Forms**

Применение средств проектирования Windows Forms обычно начинается с выбора шаблона для проекта приложения Windows Forms — с помощью пункта меню File\New Project (Файл\Создать проект).

|  |
| --- |
| [Шаблон проекта Windows Forms](http://it.fitib.altstu.ru/neud/cs/index.php?action=getimage&mode=1&imn=61) [[Увеличить изображение](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=61)](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=61)Шаблон проекта Windows Forms |

После создания нового проекта Windows Forms в Visual Studio появляется поверхность конструктора, на которую можно перетаскивать любое количество управляющих элементов. Этот же конструктор позволяет настраивать начальный размер окна, просто изменяя размеры самой формы с помощью специальных "рукояток".

|  |
| --- |
| [Визуальный конструктор форм](http://it.fitib.altstu.ru/neud/cs/index.php?action=getimage&mode=1&imn=62) [[Увеличить изображение](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=62)](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=62)Визуальный конструктор форм |

Если требуется настроить внешний вид окна (а также любого элемента на нем), это можно сделать с помощью окна Properties (Свойства). Как и в Windows Presentation Foundation, это окно позволяет назначать значения свойствам, а также устанавливать обработчики событий для элемента, выделенного в данный момент на конструкторе (конфигурация выбирается с помощью выпадающего списка в верхней части окна Properties).

|  |
| --- |
| [Окно Свойства для задания свойств и обработчиков событий](http://it.fitib.altstu.ru/neud/cs/index.php?action=getimage&mode=1&imn=63) [[Увеличить изображение](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=63)](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=63)Окно Свойства для задания свойств и обработчиков событий |

В окне Properties можно указать вывод содержимого по категориям или по алфавиту — для этого предназначены первые две кнопки под выпадающим списком.

|  |
| --- |
| [Окно обозревателя решений](http://it.fitib.altstu.ru/neud/cs/index.php?action=getimage&mode=1&imn=64) [[Увеличить изображение](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=64)](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=64)Окно обозревателя решений |

Следующий элемент конструктора, о котором стоит знать — окно обозревателя решений (Solution Explorer). Все проекты Visual Studio поддерживают это окно, но оно особенно удобно при создании приложений Windows Forms для быстрого изменения имени файла и соответствующего класса для любого окна и просмотра файла, который содержит сопровождаемый конструктором код . Если щелкнуть правой кнопкой мыши на значке Forml.cs и выбрать пункт Rename (Переименовать), то можно задать своё собственное имя файла для формы.

**Анатомия формы**

В мире Windows Forms тип Form представляет любое окно в приложении, включая главное окно самого верхнего уровня, дочерние окна приложений с многодокументным интерфейсом (multiple document interface — MDI), а также модальные и немодальные диалоговые окна. Тип Form содержит множество возможностей, унаследованных от классов-предков, а также из реализуемых им многочисленных интерфейсов.

Для полноценного порождения типа Form нужны и многие другие базовые классы и интерфейсы, но даже профессиональному разработчику Windows Forms совсем не обязательно знать роли всех членов всех классов или реализованных интерфейсов. Вообще-то большинство членов (а именно, свойств и событий) можно спокойно настраивать с помощью окна Properties (Свойства). Однако все-таки важно знать возможности, предоставляемые родительскими классами Control и Form.

**Возможности класса Control**

Класс System.Windows.Forms.Control задает общее поведение, необходимое для любого типа графического пользовательского интерфейса. Основные члены класса Control позволяют настраивать размер и позицию элемента, захватывать данные клавиатуры и мыши, получать или задавать фокус/видимость члена и т.д. Ниже приведены наиболее интересные свойства, которые сгруппированы по схожести функций.

* BackColor, ForeColor, Backgroundlmage, Font, Cursor - Определяют основные качества элемента (цвет, шрифт текста и вид курсора мыши, когда он находится над элементом)
* Anchor, Dock, AutoSize - Управляют положением элемента внутри контейнера
* Top, Left, Bottom, Right, Bounds, ClientRectangle, Height, Width - Задают текущие размеры элемента
* Enabled, Focused, Visible - Содержат логические значения, которые задают состояние элемента
* ModifierKeys - Статическое свойство, проверяет текущее состояние клавиш-модификаторов (Shift, Ctrl и Alt) и возвращает это состояние в типе Keys
* MouseButtons - Статическое свойство, проверяет текущее состояние кнопок мыши (левая, правая и средняя) и возвращает это состояние в типе MouseButtons
* Tablndex, TabStop - Позволяют настроить очередность переходов при нажатии клавиши < Таb >
* Opacity - Определяет прозрачность элемента 0.0 — полностью прозрачный, 1.0 — полностью непрозрачный)
* Text - Содержит строковые данные, связанные с данным элементом
* Controls - Предоставляет доступ к строго типизированной коллекции (например, ControlsCollection), которая содержит все дочерние элементы данного управляющего элемента

Конечно, класс Control определяет ряд событий, которые позволяют перехватывать действия, связанные, к примеру, с мышью, клавиатурой, отображением и перетаскиванием. Некоторые полезные события, сгруппированные по схожести функций, приведены ниже.

* Click, Doubleclick, MouseEnter, MouseLeave, MouseDown, MouseUp, MouseMove, MouseHover, MouseWheel - Позволяют взаимодействовать с мышью
* KeyPress, KeyUp, KeyDown - Позволяют взаимодействовать с клавиатурой
* DragDrop, DragEnter, DragLeave, DragOver - Позволяют отслеживать действия при перетаскивании

И, наконец, базовый класс Control определяет несколько методов, которые позволяют взаимодействовать с любым типом, порожденным от Control. Просматривая методы типа Control, обратите внимание, что имена многих из них содержат префикс On с последующим именем конкретного события (например, OnMouseMove, OnKeyUp и On Paint). Каждый из таких виртуальных On-методов представляет собой обработчик соответствующего события по умолчанию. Если вы переопределите любой из этих виртуальных методов, вы получите возможность выполнять всю необходимую пред- и постобработку события перед и после вызова реализации по умолчанию в родительском классе.

**Возможности класса Form**

Обычно (хотя и не обязательно) класс Form является непосредственным базовым классом для пользовательских типов Form. Кроме обширного набора членов, унаследованных от классов Control, ScrollableControl и ContainerControl, тип Form добавляет дополнительные возможности — в особенности для главных окон, дочерних MDI-окон и диалоговых окон.

* AcceptButton - Получает или назначает кнопку на форме, для которой имитируется щелчок при нажатии клавиши < Enter >
* ActiveMdiChild, IsMdiChild, IsMdiContainer - Используются в контексте MDI-приложений
* CancelButton - Получает или назначает кнопку на форме, для которой имитируется щелчок при нажатии клавиши < Esc >
* ControlBox - Получает или устанавливает значение, которое указывает, есть ли на форме блок управления (обычно значки, позволяющие свернуть, развернуть и закрыть форму в правом верхнем углу окна)
* FormBorderStyle - Получает или задает стиль формы. Используется в сочетании с перечислением FormBorderStyle
* Menu - Получает или устанавливает меню, расположенное на форме
* MaximizeBox, MinimizeBox - Определяют, активны ли на форме значки развертывания и свертывания
* ShowlnTaskbar - Определяет, будет ли отображаться форма на панели задач Windows
* StartPosition - Получает или определяет первоначальную позицию формы во время выполнения, заданную с помощью перечисления FormStartPosition
* WindowState - Задает отображение формы на этапе запуска. Используется в сочетании с перечислением FormWindowState

Кроме многочисленных стандартных On-обработчиков событий, тип Form определяет ряд основных методов, перечисленных ниже.

* Activate () - Делает данную форму активной и передает ей фокус ввода
* Close() - Закрывает текущую форму
* CenterToScreen() - Помещает форму в центр экрана
* LayoutMdi() - Размещает все дочерние формы (в соответствии с перечислением MdiLayout) в родительской форме
* Show () - Выводит форму как немодальное окно
* ShowDialog() Выводит форму как модальное диалоговое окно

И, наконец, класс Form определяет ряд событий, большинство из которых генерируются во время жизни формы :

* Activated - Возникает при активации формы, т.е. при получении фокуса на рабочем столе
* FormClosed - Возникают непосредственно перед закрытием формы и сразу после FormClosing закрытия
* Deactivate - Возникает при деактивации формы, т.е. при потере фокуса на рабочем столе
* Load - Возникает после размещения формы в памяти, но еще до отображения на экране
* MdiChildActive - Возникает при активации дочернего окна

**Жизненный цикл типа Form**

Жизнь формы начинается тогда, когда вызывается конструктор класса перед его передачей методу Application.Run(). Когда объект помещен в управляемую кучу, среда генерирует событие Load (Загружен). В обработчике события Load можно настроить внешний вид объекта Form, подготовить содержащиеся в нем дочерние элементы (наподобие ListBox и TreeView) или выделить ресурсы, необходимые для работы формы (например, подключение к базе данных и модули доступа к удаленным объектам). После события Load генерируется событие Activated (Активирован), но только тогда, когда форма получает фокус как активное окно на рабочем столе. Логической противоположностью событию Activated является (конечно же) событие Deactivate (Деактивирован), которое возникает, когда форма перестает быть активным окном.

События Activated и Deactivate могут возникать неоднократно за время жизни конкретного объекта Form, когда пользователь переключается между активными окнами и приложениями. Если пользователь решает закрыть данную форму, возникают два события: FormClosing (Форма закрывается) и FormClosed (Форма закрыта). Событие FormClosing генерируется первым и удобно для вывода конечному пользователю сообщения: "Вы уверены, что хотите закрыть данное приложение?" Этот шаг позволяет пользователю сохранить все нужные данные, прежде чем закрыть приложение.

Событие FormClosing работает в сочетании с делегатом FormClosingEventHandler. Если установить свойство FormClosingEventArgs.Cancel равным true, то окно не будет уничтожено и просто возвратится к нормальной работе. А если FormClosingEventArgs.Cancel равно false, то возникает событие FormClosed, и приложение Windows Forms завершает работу, выгружает AppDomain и завершает процесс.

**Создание диалоговых окон**

В программах с графическим пользовательским интерфейсом диалоговые окна являются одним из основных способов ввода пользовательской информации для использования в самом приложении. В отличие от других API-интерфейсов, в Windows Forms нет базового класса Dialog. Все диалоговые окна являются простыми типами, порожденными от класса Form.

Как правило, диалоговые окна не должны менять свой размер, поэтому свойству FormBorderStyle присваивается значение FormBorderStyle.FixedDialog. Кроме того, свойства MinimizeBox и MaximizeBox обычно устанавливаются равными false. В этом случае диалоговое окно имеет постоянный размер. А если установить равным false свойство ShowInTaskbar, то форма не будет отображаться на панели задач Windows.

Для кнопки ОК с помощью окна Properties можно установить свойство DialogResult равным ОК. Аналогично можно установить свойство DialogResult кнопки Cancel равным Cancel. Свойство DialogResult может быть довольно полезным, т.к. оно позволяет выбрать нужное действие, например выбираемое из списка: Abort, Cancel, Ignore, No, None, OK, Retry, Yes .

Многие пользователи привыкли перемещать фокус ввода с помощью клавиши < Таb >, если форма содержит несколько графических элементов. Настройка последовательности переходов требует знакомства с двумя свойствами: TabStop и Tablndex. Свойство TabStop может принимать значения true или false, в зависимости от того, хотите ли вы, чтобы данный элемент получал фокус с помощью клавиши < Таb >. Если свойство TabStop для какого-то элемента равно true, то для этого элемента можно установить свойство TabOrder, чтобы задать порядок активации в последовательности переходов (нумерация с нуля).

Во многих формах, требующих от пользователя ввода каких-то данных (особенно в диалоговых окнах) имеется специальная кнопка, которая срабатывает при нажатии клавиши . Допустим, что при нажатии пользователем клавиши < Enter > нужно вызывать обработчик события Click для кнопки btnOK. Для этого достаточно установить свойство AcceptButton в значение btnOK в программном коде или с помощью окна Properties.

При работе с диалоговыми окнами первым делом надо решить, открывать ли их в модальном или немодальном режиме. Модальные диалоговые окна должны быть закрыты пользователем, прежде чем он сможет вернуться в окно, из которого первоначально было открыто данное диалоговое окно. Примером модального окна может служить большинство окон "О программе". Чтобы открыть новое диалоговое окно, нужно вызвать метод ShowDialog() из объекта этого диалогового окна. А для вывода немодального диалогового окна предназначен метод Show(), который позволяет пользователю переключаться между диалогом и главным окном (как диалог "Найти/заменить"). Пример запуска формы:

**private void button1\_Click(object sender, EventArgs e)**

**{**

**Form2 f2 = new Form2();**

**f2.Show();//f2.ShowDialog();**

**button1.Text = "123";**

**}**

**Стандартные диалоговые окна**

Диалоговые окна используются для сбора сведений, вводимых пользователями. Можно создать свои собственные диалоговые окна или использовать встроенные, такие как FolderBrowserDialog и FontDialog. В этом разделе содержатся ссылки на разделы, посвященные созданию диалоговых окон и использованию стандартных встроенных диалоговых окон в приложениях.

Чтобы пользователи могли выбрать текстовый файл и загрузить его в элемент управления RichTextBox в форме Windows Forms, можно использовать компонент OpenFileDialog. В этом примере компонент OpenFileDialog создается во время выполнения. Пример:

**// Создаём объект класса OpenFileDialog.**

**// Если он вставлен в визуальном редакторе, то этого делать не нужно**

**OpenFileDialog openFile1 = new OpenFileDialog();**

**// Указываем OpenFileDialog фильтр для работы с текстовыми файлами.**

**openFile1.Filter = "Text Files|\*.txt";**

**// Проверяем был ли выбран какой0либо файл с помощью OpenFileDialog.**

**if(openFile1.ShowDialog() == System.Windows.Forms.DialogResult.OK)**

**// Загружаем выбранный файл в RichTextBox.**

**richTextBox1.LoadFile(openFile1.FileName,**

**RichTextBoxStreamType.PlainText);**

Компонент Windows Forms SaveFileDialog является стандартным диалоговым окном. Он аналогичен стандартному диалоговому окну Сохранение файла, используемому в операционных системах Windows. Он наследуется от класса CommonDialog.

Компонент используется в качестве простого решения для сохранения файлов пользователями вместо диалогового окна, настраиваемого самостоятельно. Использование стандартных диалоговых окон Windows помогает создавать приложения, основные функциональные возможности которых хорошо знакомы пользователям. Однако следует помнить, что при использовании компонента SaveFileDialog необходимо разработать собственный алгоритм сохранения файла.

Можно использовать метод ShowDialog для отображения диалогового окна во время выполнения. Используя метод OpenFile, можно открыть файл в режиме "чтение и запись". Добавленный в форму компонент SaveFileDialog появляется в нижней области конструктора Windows Forms.

В следующем примере кода показано создание SaveFileDialog, задание элементов, вызов диалогового окна с помощью метода ShowDialog и сохранение текущего файла. В примере предполагается, что используется форма, на которой размещена кнопка.

**Stream myStream ;**

**SaveFileDialog saveFileDialog1 = new SaveFileDialog();**

**saveFileDialog1.Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*" ;**

**saveFileDialog1.FilterIndex = 2 ;**

**saveFileDialog1.RestoreDirectory = true ;**

**if(saveFileDialog1.ShowDialog() == DialogResult.OK)**

**{**

**if((myStream = saveFileDialog1.OpenFile()) != null)**

**{**

**// Code to write the stream goes here.**

**myStream.Close();**

**}**

**}**

В создаваемых приложениях Windows пользователям часто приходится выбирать папку, обычно для сохранения набора файлов. С помощью компонента Windows Forms FolderBrowserDialog эта задача легко решается.

В процедуре проверьте свойство DialogResult компонента FolderBrowserDialog, чтобы узнать, как было закрыто диалоговое окно и получить значение свойства SelectedPath компонента FolderBrowserDialog.

Если необходимо задать папку самого верхнего уровня, которая будет отображаться в дереве этого диалогового окна, задайте свойство RootFolder, которое выбирает член перечисления SpecialFolder.

Кроме того, можно задать свойство Description, которое указывает строку текста, появляющуюся вверху дерева обозревателя папок.

В приведенном ниже примере компонент FolderBrowserDialog используется для выбора папки в случае, подобном тому, когда при создании проекта в Visual Studio пользователя просят выбрать папку для сохранения в ней проекта. В данном примере имя папки затем отображается в элементе управления TextBox этой формы. Прекрасное решение — поместить расположение в область, доступную для изменения, например элемент управления TextBox, чтобы пользователь имел возможность изменить свой выбор в случае ошибки или других проблем. В этом примере предполагается форма с компонентом FolderBrowserDialog и элементом управления TextBox.

**if (folderBrowserDialog1.ShowDialog() == DialogResult.OK)**

**{**

**textBox1.Text = folderBrowserDialog1.SelectedPath;**

**}**

Компонент Windows Forms FontDialog является стандартным диалоговым окном, аналогичным окну Windows Шрифт; он используется для предоставления шрифтов, установленных в операционной системе. Этот компонент используется в приложении Windows в качестве простого решения для выбора шрифтов вместо диалогового окна, настраиваемого самостоятельно.

По умолчанию в диалоговом окне отображаются списки "Шрифт", "Стиль" и "Размер"; поля флажков для таких эффектов, как зачёркивание и подчеркивание; раскрывающийся список "Набор символов"; поле, в котором показан образец шрифта. (Набор символов определяет, какие символы доступны для данного шрифта: например, иврит или японский.) Чтобы вывести диалоговое окно шрифтов, следует вызвать метод ShowDialog.

Этот компонент обладает рядом свойств, определяющих его внешний вид. Свойствами, задающими параметры диалогового окна, являются Font и Color. СвойствоFontзадает шрифт, стиль, размер, набор символов и эффекты, например: Arial, 10pt, style=Italic, Strikeout

В следующем примере кода диалоговое окно ShowDialog используется отображения диалогового окна FontDialog. Для этого кода необходимо, чтобы до его вызова была создана форма Form с размещенными на ней текстовым полем TextBox и кнопкой. Также требуется, чтобы было создано диалоговое окно fontDialog1. Объект Font содержит сведения о размере, но не содержит сведений о цвете.

**fontDialog1.ShowColor = true;**

**fontDialog1.Font = textBox1.Font;**

**fontDialog1.Color = textBox1.ForeColor;**

**if(fontDialog1.ShowDialog() != DialogResult.Cancel )**

**{**

**textBox1.Font = fontDialog1.Font ;**

**textBox1.ForeColor = fontDialog1.Color;**

**}**

Компонент Windows Forms ColorDialog является стандартным диалоговым окном, в котором пользователь может выбрать цвет из палитры, а также добавить в палитру дополнительные цвета. Это окно аналогично диалоговому окну для выбора цветов в других приложениях Windows. Этот компонент используется в приложении Windows в качестве простого решения вместо диалогового окна, настраиваемого самостоятельно.

Цвет, выбранный в диалоговом окне, возвращается в свойстве Color. Если для свойства AllowFullOpen задано значение false, кнопка "Определить цвет" недоступна и пользователь может работать только со стандартными цветами палитры. Если для свойства SolidColorOnly задано значение true, пользователь не может выбирать полутона. Для отображения диалогового окна вызывается метод ShowDialog.

Внешний вид компонента Windows Forms ColorDialog можно настроить с помощью набора его свойств. В этом диалоговом окне два раздела: в первом представлены основные цвета, во втором — средства для настройки цветов пользователем.

Большинство свойств ограничивают набор цветов, которые пользователь может выбрать в этом диалоговом окне. Если для свойства AllowFullOpen задано значение true, пользователь может задавать собственные цвета. Для свойства FullOpen задано значение true, если диалоговое окно развертывается для определения пользовательских цветов; в противном случае пользователь должен нажать кнопку "Определить цвет". Если для свойства AnyColor задано значение true, в диалоговом окне отображаются все доступные цвета в наборе основных цветов. Если для свойства SolidColorOnly задано значение true, пользователь не может выбрать полутона; для выбора доступны только чистые цвета.

Если для свойства ShowHelp задано значение true, в диалоговом окне появляется кнопка "Справка". При нажатии кнопки "Справка" возникает событие HelpRequest компонента ColorDialog

В следующем примере иллюстрируется создание нового диалогового окна ColorDialog. В этом примере предполагается, что данный метод вызывается из существующей формы, в которой имеются поле TextBox и кнопка Button.

**ColorDialog MyDialog = new ColorDialog();**

**// Keeps the user from selecting a custom color.**

**MyDialog.AllowFullOpen = false ;**

**// Allows the user to get help. (The default is false.)**

**MyDialog.ShowHelp = true ;**

**// Sets the initial color select to the current text color.**

**MyDialog.Color = textBox1.ForeColor ;**

**// Update the text box color if the user clicks OK**

**if (MyDialog.ShowDialog() == DialogResult.OK)**

**textBox1.ForeColor = MyDialog.Color;**

**Приложения с интерфейсом MDI**

Приложения с многооконным интерфейсом (MDI) позволяют отображать несколько документов сразу, при этом каждый документ отображается в отдельном окне. Приложения с MDI-интерфейсом часто содержат элементы меню Window с вложенным меню для переключения между окнами или документами.

В Windows Forms существуют определенные различия в работе MDI-форм и окон с однодокументным интерфейсом (SDI). Свойство Opacity не влияет на внешний вид дочерних форм MDI. Кроме того, метод *CenterToParent* не влияет на поведение дочерних форм MDI.

**Создание родительских MDI-форм**

Основой приложения с MDI-интерфейсом является родительская MDI-форма. Это форма содержит дочерние MDI-окна, являющиеся "вложенными окнами", в которых пользователи взаимодействуют с MDI-приложением. Создать родительскую MDI-форму несложно как в конструкторе Windows Forms, так и программными средствами.

Чтобы создать родительскую MDI-форму в режиме разработки, выполните следующие действия. Создайте проект "Приложение Windows". В окне Свойства установите свойству*IsMDIContainer* значение *true*. При этом форма назначается MDI-контейнером для дочерних окон.

При задании свойств в окне Свойства можно также установить для свойства *WindowState* значение *Maximized*, поскольку легче всего управлять дочерним MDI-окном, когда родительская форма развернута. Кроме того, следует помнить, что край родительской MDI-формы примет системный цвет (заданный в системной панели управления Windows), а не черный цвет, заданный с помощью свойства *Control.BackColor*.

|  |
| --- |
| [Создание главного меню](http://it.fitib.altstu.ru/neud/cs/index.php?action=getimage&mode=1&imn=65) [[Увеличить изображение](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=65)](http://it.fitib.altstu.ru/neud/cs/index.php?action=viewimage&imn=65)Создание главного меню |

Из панели элементов перетащите компонент MenuStrip в форму. Создайте пункт меню верхнего уровня, свойству *Text* которого присвоено значение *&Файл*, с пунктами вложенного меню *&Создать* и *&Закрыть*. Создайте также пункт меню верхнего уровня *&Окно*. Первое меню будет создавать и скрывать пункты меню во время выполнения, а второе меню будет контролировать открытые дочерние MDI-окна. С этого момента родительское MDI-окно создано.

**Создание дочерних MDI-форм**

Дочерние MDI-формы являются основным элементом Приложения с интерфейсом MDI, поскольку они находятся в центре взаимодействия с пользователями.

Чтобы создать дочерние MDI-формы, выполните следующие действия. Создайте родительскую MDI-форму, структура меню которой содержит пункты меню верхнего уровня Файл и Окно и пункты меню Создать и Закрыть. Из раскрывающегося списка в верхней части окна Свойства выберите пункт меню, соответствующий пункту меню *&Окно*, и задайте свойству*MdiList* значение *true*. Это позволит меню *Окно* поддерживать список открытых дочерних MDI-окон с галочкой рядом с активным дочерним окном.

В Обозревателе решений щелкните правой кнопкой мыши проект, выберите команду Добавить, затем Добавить новый элемент. Эта форма будет шаблоном для создаваемой дочерней MDI-формы. Дочерняя MDI-форма, создаваемая на этом этапе, является стандартной формой Windows Forms. В этом качестве у нее есть свойство *Opacity*, позволяющее управлять прозрачностью формы. Однако свойство *Opacity* разработано для окон верхнего уровня. Его не следует использовать в дочерних MDI-формах, иначе могут возникнуть проблемы с рисованием.

В диалоговом окне Добавить новый элемент выберите Форма Windows Forms из области Шаблоны. В поле Имя введите имя формы Form2. Нажмите кнопку Открыть, чтобы добавить форму в проект. Откроется конструктор Windows Forms, отображающий форму Form2.

Создайте обработчик событий *Click* для элемента меню *Создать*. Вставьте код, аналогичный следующему, чтобы создать новую дочернюю MDI-форму при нажатии пользователем пункта меню Создать.

В следующем примере обработчик событий обрабатывает событие Click для MenuItem2. Следует помнить, что, в зависимости от особенностей архитектуры приложения, созданный пункт меню Создать может не являться MenuItem2.

**protected void MDIChildNew\_Click(object sender, System.EventArgs e){**

**Form2 newMDIChild = new Form2();**

**// Set the Parent Form of the Child window.**

**newMDIChild.MdiParent = this;**

**// Display the new form.**

**newMDIChild.Show();**

**}**

Обратите внимание, что при выборе команды *Создать* в меню *Файл* можно создавать новые дочерние MDI-формы, управление которыми осуществляется в меню *Окно*. Следует помнить, что когда в дочерней MDI-форме есть компонент *MenuStrip* (обычно обладающий структурой пунктов меню), и этот компонент открыт внутри родительской MDI-формы, также имеющей компонент *MenuStrip* (обычно обладающий структурой пунктов меню), пункты меню будут объединены автоматически, если задано свойство *MergeAction* (и, по выбору, свойство *MergeIndex*. Установите для свойства *MergeAction* главного элемента *MenuStrip* значение *MatchOnly*. Кроме того, установите свойство *MergeIndex* таким образом, чтобы пункты обоих меню появлялись в установленном порядке, а свойство *Visible* установите в значение *false* . Необходимо отметить, что при закрытии родительской MDI-формы каждая из дочерних MDI-форм создает событие *Closing* до создания события *Closing* для родительской MDI-формы. Отмена события *Closing* дочерней MDI-формы не отменяет события *Closing*родительской MDI-формы; тем не менее, для аргумента *CancelEventArgs* для события *Closing* родительской MDI-формы не устанавливается значение *true*. Можно добиться того, чтобы родительская и все дочерние MDI-формы закрылись, задав аргументу *CancelEventArgs* значение *false*.

**Определение активной дочерней MDI-формы**

В некоторых случаях требуется иметь команду, которая бы работала с элементом управления, находящимся в фокусе в текущей активной дочерней форме. Предположим, что требуется скопировать выделенный текст из текстового поля дочерней формы в буфер обмена. Необходимо создать процедуру, которая копирует выделенный текст в буфер обмена, используя событие Click команды "Копировать" в стандартном меню "Правка".

Так как в приложении с MDI-интерфейсом могут существовать несколько экземпляров одной и той же дочерней формы, процедура должна знать, какую форму использовать. Чтобы указать требуемую форму, используйте свойство ActiveMdiChild, которое возвращает дочернюю форму, которая находится в фокусе или использовалась самой последней.

Если в форме существуют несколько элементов управления, требуется также указать активный элемент. Подобно свойству ActiveMdiChild свойство ActiveControl возвращает элемент управления с фокусом на активной дочерней форме. Следующая процедура служит примером процедуры копирования, которую можно вызвать из меню дочерней формы, из меню формы с MDI-интерфейсом или с помощью кнопки в панели инструментов.

В этом примере предполагается, что существует родительская форма MDI (Form1), которая имеет одну или несколько дочерних MDI-окон, содержащих элемент управления RichTextBox.

**protected void mniCopy\_Click (object sender, System.EventArgs e)**

**{**

**// Determine the active child form.**

**Form activeChild = this.ActiveMdiChild;**

**// If there is an active child form, find the active control, which**

**// in this example should be a RichTextBox.**

**if (activeChild != null)**

**{**

**try**

**{**

**RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;**

**if (theBox != null)**

**{**

**// Put the selected text on the Clipboard.**

**Clipboard.SetDataObject(theBox.SelectedText);**

**}**

**}**

**catch**

**{**

**MessageBox.Show("You need to select a RichTextBox.");**

**}**

**}**

**}**

**Отправка данных в активную дочернюю MDI-форму**

Часто в контексте приложений с многодокументным интерфейсом (MDI) требуется отправить данные в активное дочернее окно. Например, это требуется в том случае, если пользователь вставляет данные из буфера обмена в приложение с MDI-интерфейсом.

Чтобы отправить данные в активную дочернюю MDI-форму из буфера обмена. В этом примере предполагается, что существует родительская форма MDI (Form1), которая имеет одну или несколько дочерних MDI-окон, содержащих элемент управления RichTextBox.

**protected void mniPaste\_Click (object sender, System.EventArgs e)**

**{**

**// Determine the active child form.**

**Form activeChild = this.ParentForm.ActiveMdiChild;**

**// If there is an active child form, find the active control, which**

**// in this example should be a RichTextBox.**

**if (activeChild != null)**

**{**

**try**

**{**

**RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;**

**if (theBox != null)**

**{**

**// Create a new instance of the DataObject interface.**

**IDataObject data = Clipboard.GetDataObject();**

**// If the data is text, then set the text of the**

**// RichTextBox to the text in the clipboard.**

**if (data.GetDataPresent(DataFormats.Text))**

**{**

**theBox.SelectedText = data.GetData(DataFormats.Text).ToString();**

**}**

**}**

**}**

**catch**

**{**

**MessageBox.Show("You need to select a RichTextBox.");**

**}**

**}**

**}**

**Упорядочение дочерних форм интерфейса MDI**

Приложения могут иметь команды меню, такие как "Мозаика", "Каскадом" и "Расположить", которые относятся к открытым дочерним MDI-формам. Можно также использовать метод LayoutMdi с перечислением MdiLayout для изменения порядка дочерних форм в родительской форме MDI.

В методе MdiLayout можно использовать одно из четырех различных значений перечисления LayoutMdi. С помощью значений перечисления можно представить дочерние формы в виде каскада, вертикально или горизонтально без перекрытия или в виде значков форм, расположенных вдоль нижней части MDI-формы.

Часто эти методы используются в качестве обработчиков событий, вызываемых событием Click пункта меню. Таким образом, пункт меню с текстом "Cascade Windows" может оказывать желаемое воздействие на дочерние MDI-формы.

Чтобы упорядочить дочерние формы в методе используйте метод LayoutMdi для задания перечисления MdiLayout для родительской MDI-формы. В следующем примере для дочерних окон родительской MDI-формы (Form1) используется параметр Cascade перечисления MdiLayout. Перечисление используется в коде во время обработки события Click пункта меню Cascade Windows.

**protected void CascadeWindows\_Click(object sender, System.EventArgs e){**

**this.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade);**

**}**

**Операции перетаскивания и поддержка буфера обмена**

Пользовательские операции перетаскивания в приложении Windows можно включить, обрабатывая последовательность событий, которая, как правило, включает события DragEnter, DragLeave и DragDrop. Можно также реализовать поддержку вырезания/копирования/вставки и передачу данных пользователя в буфер обмена в Windows-приложения с помощью вызова простых методов.

**Выполнение операции перетаскивания в Windows Forms**

Для выполнения операций перетаскивания в приложениях Windows необходимо обрабатывать последовательность событий, главным образом события *DragEnter*, *DragLeave* и*DragDrop*. Перетаскивание можно с легкостью организовать при работе со сведениями, доступными через аргументы этих событий.

Все операции перетаскивания начинаются с переноса данных. Функциональные возможности, позволяющие собрать данные при начале перетаскивания, реализуются в методе DoDragDrop. В следующем примере событие MouseDown используется для начала операции перетаскивания, так как это первое, что делает интуитивно пользователь (большинство операций перетаскивания начинаются с отпущенной кнопкой мыши). Однако следует учесть, что для инициализации процедуры перетаскивания может использоваться любое событие.

Некоторые элементы управления имеют специальные события, относящиеся к перетаскиванию. Например, элементы управления ListView и TreeView имеют событие ItemDrag.

Чтобы начать операцию переноса, выполните следующие действия. В событии MouseDown для элемента управления, с которого начнется операция переноса, используйте метод DoDragDrop, чтобы определить данные для переноса и действия, разрешенные при переносе.

В следующем примере показан способ инициализации операции переноса. Элементом управления, в котором начинается перетаскивание, является *Button*, перетаскиваемыми данными является строка, представляющая свойство *Text* элемента управления *Button*, а в качестве разрешенных действий устанавливаются копирование и перемещение.

**private void button1\_MouseDown(object sender,**

**System.Windows.Forms.MouseEventArgs e)**

**{**

**button1.DoDragDrop(button1.Text, DragDropEffects.Copy |**

**DragDropEffects.Move);**

**}**

В качестве параметра метода *DoDragDrop* можно использовать любые данные. В предыдущем примере использовалось свойство *Text* элемента управления *Button* (вместо того чтобы жестко запрограммировать значение или получить значение из набора данных), поскольку это свойство было связано с местом расположения данных, которые требовалось перенести (элемент управления *Button*). Помните об этом при реализации операций перетаскивания в приложениях *Windows*.

Во время выполнения операции перетаскивания можно обрабатывать событие *QueryContinueDrag*, которое "запрашивает разрешение" системы на продолжение операции перетаскивания. При обработке этого метода удобно также вызывать методы, влияющие на операцию перетаскивания, таких как развертывание объекта *TreeNode* в элементе управления *TreeView*, когда курсор находится над ним

После начала операции переноса данных из определенной области в форме Windows Form или из элемента управления, необходимо эти данные куда-либо поместить. При прохождении курсора через область формы или элемент управления, которые правильно настроены для освобождения данных, вид курсора изменится. Любая область внутри формы Windows Form или элемента управления может быть настроена для приема перетащенных данных путем установки свойства *AllowDrop* и обработки событий *DragEnter* и *DragDrop*.

Чтобы осуществить освобождение данных, выполните следующие действия. Задайте значение "true" свойству *AllowDrop*. Убедитесь, что перетаскиваемые данные являются допустимым типом для события *DragEnter* элемента управления, в который осуществляется перетаскивание (в данном случае этим типом является *Text*). Затем в коде для эффекта, возникающего после освобождения данных, задайте значение перечисления *DragDropEffects*.

**private void textBox1\_DragEnter(object sender,**

**System.Windows.Forms.DragEventArgs e)**

**{**

**if (e.Data.GetDataPresent(DataFormats.Text))**

**e.Effect = DragDropEffects.Copy;**

**else**

**e.Effect = DragDropEffects.None;**

**}**

Можно определить собственные форматы *DataFormats*, указав собственный объект в качестве параметра *Object* метода *SetData*. Убедитесь, что при выполнении этого действия указанный объект можно сериализовать.

В событии *DragDrop* для элемента управления, в котором будут размещены данные, используйте метод *GetData* для получения перетаскиваемых данных. В показанном ниже примере элемент управления *TextBox* является элементом управления, в который переносятся данные (в котором будет выполнено освобождение данных). В этом коде свойство *Text*элемента управления *TextBox* устанавливается равным перетаскиваемым данным.

**private void textBox1\_DragDrop(object sender,**

**System.Windows.Forms.DragEventArgs e)**

**{**

**textBox1.Text = e.Data.GetData(DataFormats.Text).ToString();**

**}**

Кроме того, можно работать со свойством *KeyState*: в зависимости от клавиш, нажатых во время операции перетаскивания, происходят определенные действия (например, при нажатии клавиши CTRL обычно происходит копирование переносимых данных).

**Выполнение операции перетаскивания между приложениями**

Выполнение операций перетаскивания между приложениями не отличается от выполнения этого действия в одном приложении, пока поведение участвующих приложений соответствует "контракту", установленному между свойствами *AllowedEffect* и *Effect*.

Ниже на примере созданного пользователем приложения Windows и встроенного в операционную систему Windows текстового редактора WordPad показана операция перетаскивания между приложениями. В приложении WordPad имеется определенный набор допустимых действий с перетаскиваемым текстом; приложение Windows, которое создается с помощью кода, будет использовать эти действия для успешного завершения операций перетаскивания.

Чтобы осуществить процедуру перетаскивания между приложениями, выполните следующие действия.

* Создайте новое приложение Windows.
* Добавьте элемент управления TextBox в форму.
* Настройте элемент управления TextBox для получения перетащенных данных.
* Запустите приложение Windows и редактор WordPad. WordPad — это текстовый редактор операционной системы Windows, который поддерживает операции перетаскивания. Чтобы его запустить, можно нажать кнопку Пуск, выбрать Выполнить, ввести WordPad в текстовое поле диалогового окна Запуск программы и нажать кнопку OK.
* Открыв WordPad, наберите в нем произвольную строку текста.
* Используя мышь, выделите текст и перетащите его в элемент управления TextBox приложения Windows.

Обратите внимание, что, когда указатель мыши находится над элементом управления TextBox (и, следовательно, инициируется событие DragEnter), курсор изменяется и можно освободить выделенный текст в элемент управления TextBox. Кроме того, можно настроить элемент управления TextBox таким образом, чтобы разрешить перетаскивание текста в приложение WordPad.

**Добавление данных в буфер обмена**

Класс *Clipboard* предоставляет методы, которые можно использовать для взаимодействия с функцией буфера обмена операционной системы Windows. Многие приложения используют буфер обмена в качестве временного хранилища данных. Например, текстовые процессоры используют буфер обмена во время операций вырезания и вставки. Буфер обмена также полезен для передачи данных из одного приложения в другое.

При добавлении данных в буфер обмена можно указать формат данных таким образом, чтобы другие приложения могли распознать данные, если они могут использовать этот формат. Можно также добавить данные в буфер обмена в нескольких различных форматах, чтобы увеличить количество других приложений, которые потенциально могут использовать данные.

Формат буфера обмена является строкой, которая определяет формат таким образом, чтобы приложение, использующее этот формат, могло получать связанные данные. Класс*DataFormats* предоставляет предварительно определенные имена форматов для использования. Можно также использовать собственные имена форматов или использовать тип объекта в качестве его формата.

Чтобы добавить данные в буфер обмена в одном или нескольких форматах, используйте метод *SetDataObject*. Этому методу можно передать любой объект, но для добавления данных в нескольких форматах необходимо сначала добавить данные в отдельный объект, предназначенный для работы с несколькими форматами данных. Обычно данные добавляются к объекту *DataObject*, но можно использовать любой тип, реализующий интерфейс *IDataObject*.

В .NET Framework, версия 2.0 можно добавить данные непосредственно в буфер обмена с помощью новых методов, позволяющих упростить основные задачи при работе с буфером обмена. Используйте эти методы при работе с данными в одном общем формате, такими как текст. Все приложения Windows используют один буфер обмена. Поэтому содержимое буфера обмена может изменяться при переходе к другому приложению.

Класс *Clipboard* может использоваться только в потоках в режиме однопотокового подразделения. Чтобы использовать этот класс, убедитесь, что используемый метод *Main*помечен атрибутом *STAThreadAttribute*.

Для помещения в буфер обмена объект должен быть сериализуемым. Чтобы сделать тип сериализуемым, его необходимо пометить атрибутом *SerializableAttribute*. Если методу буфера обмена передается не сериализуемый объект, метод завершится неудачей без создания исключения.

Чтобы добавить данные в буфер обмена в одном общем формате, выполните следующие действия.

Используйте метод SetAudio, SetFileDropList, SetImage или SetText. Эти методы доступны только в .NET Framework, версия 2.0

**// Demonstrates SetAudio, ContainsAudio, and GetAudioStream.**

**public System.IO.Stream SwapClipboardAudio(**

**System.IO.Stream replacementAudioStream)**

**{**

**System.IO.Stream returnAudioStream = null;**

**if (Clipboard.ContainsAudio())**

**{**

**returnAudioStream = Clipboard.GetAudioStream();**

**Clipboard.SetAudio(replacementAudioStream);**

**}**

**return returnAudioStream;**

**}**

**// Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList**

**public System.Collections.Specialized.StringCollection**

**SwapClipboardFileDropList(**

**System.Collections.Specialized.StringCollection replacementList)**

**{**

**System.Collections.Specialized.StringCollection returnList = null;**

**if (Clipboard.ContainsFileDropList())**

**{**

**returnList = Clipboard.GetFileDropList();**

**Clipboard.SetFileDropList(replacementList);**

**}**

**return returnList;**

**}**

**// Demonstrates SetImage, ContainsImage, and GetImage.**

**public System.Drawing.Image SwapClipboardImage(**

**System.Drawing.Image replacementImage)**

**{**

**System.Drawing.Image returnImage = null;**

**if (Clipboard.ContainsImage())**

**{**

**returnImage = Clipboard.GetImage();**

**Clipboard.SetImage(replacementImage);**

**}**

**return returnImage;**

**}**

**// Demonstrates SetText, ContainsText, and GetText.**

**public String SwapClipboardHtmlText(String replacementHtmlText)**

**{**

**String returnHtmlText = null;**

**if (Clipboard.ContainsText(TextDataFormat.Html))**

**{**

**returnHtmlText = Clipboard.GetText(TextDataFormat.Html);**

**Clipboard.SetText(replacementHtmlText, TextDataFormat.Html);**

**}**

**return returnHtmlText;**

**}**

Чтобы добавить данные в буфер обмена в пользовательском формате, выполните следующие действия.

Используйте метод *SetData* с именем пользовательского формата. Этот метод доступен только в .NET Framework, версия 2.0. Можно также использовать предварительно определенные имена форматов с помощью метода *SetData*.

**// Demonstrates SetData, ContainsData, and GetData**

**// using a custom format name and a business object.**

**public Customer TestCustomFormat**

**{**

**get**

**{**

**Clipboard.SetData("CustomerFormat", new Customer("Customer Name"));**

**if (Clipboard.ContainsData("CustomerFormat"))**

**{**

**return Clipboard.GetData("CustomerFormat") as Customer;**

**}**

**return null;**

**}**

**}**

**...**

**[Serializable]**

**public class Customer**

**{**

**private string nameValue = string.Empty;**

**public Customer(String name)**

**{**

**nameValue = name;**

**}**

**public string Name**

**{**

**get { return nameValue; }**

**set { nameValue = value; }**

**}**

**}**

Чтобы добавить данные в буфер обмена в нескольких форматах, выполните следующие действия. Используйте метод *SetDataObject* и передайте ему объект *DataObject*, содержащий данные. Необходимо использовать этот метод для добавления данных в буфер обмена для более ранних версий, чем .NET Framework 2.0.

**// Demonstrates how to use a DataObject to add**

**// data to the Clipboard in multiple formats.**

**public void TestClipboardMultipleFormats()**

**{**

**DataObject data = new DataObject();**

**// Add a Customer object using the type as the format.**

**data.SetData(new Customer("Customer as Customer object"));**

**// Add a ListViewItem object using a custom format name.**

**data.SetData("CustomFormat",**

**new ListViewItem("Customer as ListViewItem"));**

**Clipboard.SetDataObject(data);**

**DataObject retrievedData = (DataObject)Clipboard.GetDataObject();**

**if (retrievedData.GetDataPresent("CustomFormat"))**

**{**

**ListViewItem item =**

**retrievedData.GetData("CustomFormat") as ListViewItem;**

**if (item != null)**

**{**

**MessageBox.Show(item.Text);**

**}**

**}**

**if (retrievedData.GetDataPresent(typeof(Customer)))**

**{**

**Customer customer =**

**retrievedData.GetData(typeof(Customer)) as Customer;**

**if (customer != null)**

**{**

**MessageBox.Show(customer.Name);**

**}**

**}**

**}**

**...**

**[Serializable]**

**public class Customer**

**{**

**private string nameValue = string.Empty;**

**public Customer(String name)**

**{**

**nameValue = name;**

**}**

**public string Name**

**{**

**get { return nameValue; }**

**set { nameValue = value; }**

**}**

**}**

### Извлечение данных из буфера обмена

Класс *Clipboard*  предоставляет методы, которые можно использовать для взаимодействия с функцией буфера обмена операционной системы Windows. Многие приложения используют буфер обмена в качестве временного хранилища данных. Например, текстовые процессоры используют буфер обмена во время операций вырезания и вставки. Буфер обмена также полезен для передачи данных из одного приложения в другое.

Некоторые приложения хранят данные в буфере обмена в нескольких форматах, чтобы увеличить количество других приложений, которые потенциально могут использовать данные. Формат буфера обмена является строкой, которая определяет формат. Приложение, использующее указанный формат, может извлечь связанные данные из буфера обмена. Класс *DataFormats* предоставляет предварительно определенные имена форматов для использования. Можно также использовать собственные имена форматов или использовать тип объекта в качестве его формата.

Чтобы определить, содержит ли буфер обмена данные в определенном формате, используйте один из методов *ContainsFormat* или метод *GetData*. Чтобы извлечь данные из буфера обмена, воспользуйтесь одним из методов *GetFormat* или методом *GetData*. Эти методы являются новыми в .NET Framework, версия 2.0.

Для доступа к данным из буфера обмена с помощью более ранних версий, чем .NET Framework 2.0, используйте метод *GetDataObjec*t и вызывайте методы возвращаемого интерфейса *IDataObject*. Чтобы определить, является ли доступным определенный формат в возвращаемом объекте, вызовите, например, метод *GetDataPresent*.

Все приложения Windows используют один системный буфер обмена. Поэтому содержимое буфера обмена может изменяться при переходе к другому приложению. Класс*Clipboard* может использоваться только в потоках в режиме однопотокового подразделения. Чтобы использовать этот класс, убедитесь, что используемый метод *Main* помечен атрибутом *STAThreadAttribute*.

Для получения данных из буфера обмена в одном стандартном формате, выполните следующие действия.

Используйте метод *GetAudioStream*, *GetFileDropList*, *GetImage* или *GetText*. При необходимости используйте сначала соответствующие методы *ContainsFormat*, чтобы определить, доступны ли данные в определенном формате. Эти методы доступны только в .NET Framework, версия 2.0.

**// Demonstrates SetAudio, ContainsAudio, and GetAudioStream.**

**public System.IO.Stream SwapClipboardAudio(**

**System.IO.Stream replacementAudioStream)**

**{**

**System.IO.Stream returnAudioStream = null;**

**if (Clipboard.ContainsAudio())**

**{**

**returnAudioStream = Clipboard.GetAudioStream();**

**Clipboard.SetAudio(replacementAudioStream);**

**}**

**return returnAudioStream;**

**}**

**// Demonstrates SetFileDropList, ContainsFileDroList, and GetFileDropList**

**public System.Collections.Specialized.StringCollection**

**SwapClipboardFileDropList(**

**System.Collections.Specialized.StringCollection replacementList)**

**{**

**System.Collections.Specialized.StringCollection returnList = null;**

**if (Clipboard.ContainsFileDropList())**

**{**

**returnList = Clipboard.GetFileDropList();**

**Clipboard.SetFileDropList(replacementList);**

**}**

**return returnList;**

**}**

**// Demonstrates SetImage, ContainsImage, and GetImage.**

**public System.Drawing.Image SwapClipboardImage(**

**System.Drawing.Image replacementImage)**

**{**

**System.Drawing.Image returnImage = null;**

**if (Clipboard.ContainsImage())**

**{**

**returnImage = Clipboard.GetImage();**

**Clipboard.SetImage(replacementImage);**

**}**

**return returnImage;**

**}**

**// Demonstrates SetText, ContainsText, and GetText.**

**public String SwapClipboardHtmlText(String replacementHtmlText)**

**{**

**String returnHtmlText = null;**

**if (Clipboard.ContainsText(TextDataFormat.Html))**

**{**

**returnHtmlText = Clipboard.GetText(TextDataFormat.Html);**

**Clipboard.SetText(replacementHtmlText, TextDataFormat.Html);**

**}**

**return returnHtmlText;**

**}**

Для извлечения данных из буфера обмена в пользовательском формате, выполните следующие действия. Используйте метод *GetData* с именем пользовательского формата. Этот метод доступен только в .NET Framework, версия 2.0. Можно также использовать предварительно определенные имена форматов с помощью метода *SetData*. Дополнительные сведения см. в разделе *DataFormats*.

**// Demonstrates SetData, ContainsData, and GetData**

**// using a custom format name and a business object.**

**public Customer TestCustomFormat**

**{**

**get**

**{**

**Clipboard.SetData("CustomerFormat", new Customer("Customer Name"));**

**if (Clipboard.ContainsData("CustomerFormat"))**

**{**

**return Clipboard.GetData("CustomerFormat") as Customer;**

**}**

**return null;**

**}**

**}**

**...**

**[Serializable]**

**public class Customer**

**{**

**private string nameValue = string.Empty;**

**public Customer(String name)**

**{**

**nameValue = name;**

**}**

**public string Name**

**{**

**get { return nameValue; }**

**set { nameValue = value; }**

**}**

**}**

Для извлечения данных из буфера обмена в нескольких форматах, выполните следующие действия. Используйте метод *GetDataObject*. Этот метод необходимо использовать для извлечения данных из буфера обмена для более ранних версий, чем .NET Framework 2.0.

**// Demonstrates how to use a DataObject to add**

**// data to the Clipboard in multiple formats.**

**public void TestClipboardMultipleFormats()**

**{**

**DataObject data = new DataObject();**

**// Add a Customer object using the type as the format.**

**data.SetData(new Customer("Customer as Customer object"));**

**// Add a ListViewItem object using a custom format name.**

**data.SetData("CustomFormat",**

**new ListViewItem("Customer as ListViewItem"));**

**Clipboard.SetDataObject(data);**

**DataObject retrievedData = (DataObject)Clipboard.GetDataObject();**

**if (retrievedData.GetDataPresent("CustomFormat"))**

**{**

**ListViewItem item =**

**retrievedData.GetData("CustomFormat") as ListViewItem;**

**if (item != null)**

**{**

**MessageBox.Show(item.Text);**

**}**

**}**

**if (retrievedData.GetDataPresent(typeof(Customer)))**

**{**

**Customer customer =**

**retrievedData.GetData(typeof(Customer)) as Customer;**

**if (customer != null)**

**{**

**MessageBox.Show(customer.Name);**

**}**

**}**

**}**

**...**

**[Serializable]**

**public class Customer**

**{**

**private string nameValue = string.Empty;**

**public Customer(String name)**

**{**

**nameValue = name;**

**}**

**public string Name**

**{**

**get { return nameValue; }**

**set { nameValue = value; }**

**}**

**}**

Задание на лабораторную работу:

* 1. Создать приложение согласно спецификациям, созданным в лабораторных работах 1-4.
  2. Написать отчет о проделанной работе.

Содержание отчета:

1. Титульный лист.
2. Содержание.
3. Обзор предметной области.
4. Описание элементов WinForms использованные в программе (оформить в виде таблицы с колонками: название, описание действий)
5. Диаграммы соответствующие спецификациям должны быть вынесены в качестве приложений [Приложения нумеруются большими буквами русского алфавита А, Б, В, …]. **(Внимание диаграммы должны соответствовать разработанному программному обеспечению. Если диаграммы не соответствуют, их необходимо доработать).**

**ВНИМАНИЕ!!!** На защиту приносится распечатанный отчет.