Пишем простой многодокументный редактор текста RTF

В Visual Studio создадим приложение Windows Forms.
На форму приложения перетащим главное меню MenuStrip и стандартные диалоги для работы с файлами OpenFileDialog, SaveFileDialog. У диалогов достаточно настроить свойство Filter на нужный нам тип файлов, с которым будем работать, введём "`Файлы RTF|*.rtf`" (без кавычек).
В меню предусмотрим несколько пунктов, которые покажут разные возможности работы с файлами, текстом в дочерних окнах и самими окнами:

- Файл: Создать, Сохранить, Открыть, Выход
- Правка: Дата и время
- Окно: Каскад, Мозаика сверху вниз, Мозаика слева направо

Также не забудем установить родительской форме свойство IsMdiContainer = true.
Под меню добавим панель инструментов Toolstrip.

Затем добавим к приложению ещё одну форму Form2 (меню Проект - Добавить форму Windows Forms).
В окно Form2 добавим контекстное меню contextMenuStrip1, которое будет вызываться правой кнопкой мыши. Предусмотрим там 4 пункта (Вырезать, Копировать, Копировать все, Вставить). Это позволит приложению обмениваться кусками текста между окнами через системный Буфер Обмена. Сначала все пункты отключены (свойство Enabled = false). Указать горячие клавиши для меню можно, выбрав в Конструкторе нужный пункт меню и воспользовавшись свойством ShortcutKeys.
Также перетащим на форму Form2 многострочное текстовое поле RichTextBox и установим у него в окне Свойств размер во всю область дочернего окна (свойство Dock = Fill).
Созданное меню contextMenuStrip1 укажем в свойстве ContextMenuStrip компоненты richTextBox1.
Для простоты реализации Добавим в дочернее окно его собственный SaveFileDialog, чтобы можно было выбрать имя файла для сохранения, если мы закрываем дочернее окно с изменённым текстом. На диалог поставим такой же фильтр, как у родительской формы. В общем случае лучше было бы не плодить компоненты, а воспользоваться неким отдельным классом-наследником SaveFileDialog, который мы могли написать.

В классе главной формы пропишем свойство "Счётчик окон":

```
  private static int winCounter;
```

и собственный метод класса для создания нового окна:

```
  private void createNewWindow () { //Метод для создания нового окна
   Form2 newForm = new Form2 ();
   newForm.MdiParent = this; //Указали новой форме, кто её родитель
   newForm.Text = "Noname " + ++winCounter; //Номер нового окна - в заголовок
    //Так же при необходимости ставим любые другие свойства новой формы
   newForm.Show (); //Показываем новую форму
  }
```

Мы уже умеем создавать обработчики событий двойным кликом из списка событий в окне "Свойства", так что приведу только "внутренности" функций-обработчиков стандартных событий.
По выбору пункта меню "Создать" просто вызовем только что написанный метод:

```
   createNewWindow ();
```

Пункт "Выход" запрограммировать тоже просто, будем вызывать стандартный метод закрытия главной формы:

```
   this.Close ();
```

Для записи файла, имя которого было введено или выбрано в стандартном диалоге сохранения, напишем обработчик пункта меню "Сохранить...":

```
   Form activeChild = this.ActiveMdiChild;
```

```
    if (activeChild != null) {
     RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;
     if (activeChild.Text.Trim() != String.Empty)
      saveFileDialog1.FileName = activeChild.Text;
     if (saveFileDialog1.ShowDialog () == DialogResult.OK) {
      try {
       theBox.SaveFile (saveFileDialog1.FileName);
      }
      catch (Exception) {
       MessageBox.Show (String.Format ("Ошибка сохранения файла {0}",
        saveFileDialog1.FileName));
      }
      activeChild.Text = saveFileDialog1.FileName;
     }
    }
    else {
     MessageBox.Show ("Не выбрано ни одно окно");
    }
```

Здесь также видно, как обратиться к активному в настоящий момент дочернему окну или отследить ситуацию, при которой такого окна нет.
Готовый метод SaveFile есть у компоненты RichTextBox.

Пункт верхнего меню "Открыть..." будет выполнять довольно похожую работу - определять, есть ли активное дочернее окно, получать для него имя файла и выводить стандартный диалог открытия, после чего попытается сделать непосредственно загрузку из файла содержимого:

```
    createNewWindow (); //Открываем всегда в новом окне
    Form activeChild = this.ActiveMdiChild;
    if (activeChild != null) {
     RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;
     if (activeChild.Text.Trim () != String.Empty)
      openFileDialog1.FileName = activeChild.Text;
     if (openFileDialog1.ShowDialog () == DialogResult.OK) {
      if (openFileDialog1.FileName == null) return;
      try {
       theBox.LoadFile (openFileDialog1.FileName);
       theBox.Modified = false;
      }
      catch (Exception ex) { // Отчет об ошибках
       MessageBox.Show (ex.Message, "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
      }
      activeChild.Text = openFileDialog1.FileName;
      }
     }
     else {
      MessageBox.Show ("Не выбрано ни одно окно");
     }
```

Удобство MDI-приложения в том, что при закрытии главного окна обработчики события FormClosing дочерних окон будут вызваны автоматически, так что каждому экземпляру дочернего окна достаточно будет позаботиться о сохранности только своего текста из своего RichTextBox.

При этом, отказ от сохранения какого-либо из текстов в дочерних окнах не будет означать отказа от закрытия приложения.
Ниже приведён обработчик события FormClosing главной формы. Комментарием показано, как можно было бы обойти в цикле все дочерние окна.

```
/*
for (int x = 0; x < this.MdiChildren.Length; x++) {
 //Так можно перебрать потомков
 Form tempChild = (Form)this.MdiChildren [x];
 MessageBox.Show (tempChild.Text);
}
*/
//Для MDI-приложений метод сам вызовет FormClosing потомков
e.Cancel = false;
```

Выбор пункта меню "Дата и время" вставит в конец текста активного окна текущие дату и время, примерно как в стандартном Блокноте Windows. Эта функция добавлена просто для иллюстрации:

```
Form activeChild = this.ActiveMdiChild;
if (activeChild != null) {
 try {
  RichTextBox theBox = (RichTextBox)activeChild.ActiveControl;
  if (theBox != null) {
   DateTime d = new DateTime (DateTime.Now.Year,
    DateTime.Now.Month, DateTime.Now.Day,
    DateTime.Now.Hour, DateTime.Now.Minute, DateTime.Now.Second);
    theBox.AppendText (String.Format ("\n{0}, {1}\n",
     d.ToLongDateString (), d.ToLongTimeString ()));
  }
 }
 catch (Exception) {
  MessageBox.Show ("Ошибка вставки даты и времени");
 }
}
else {
 MessageBox.Show ("Не выбрано ни одно окно");
}
```

Пункт "Каскад" уложит каскадом имеющиеся дочерние окна:

```
this.LayoutMdi (MdiLayout.Cascade); //Уложить дочерние окна каскадом
```

По аналогии легко сделать остальные стандартные "Укладки":

```
this.LayoutMdi (MdiLayout.TileHorizontal); //Мозаика сверху вниз

this.LayoutMdi (MdiLayout.TileVertical); //Мозаика слева направо
```

Напишем немного кода в файле `Form2.cs`, программируя функционал дочерних окон.
Во-первых, позаботимся о том, чтобы пункты контекстного меню были доступны только тогда, когда они нужны.
Собственный метод класса checkForClipboardFormat будет проверять, что находится в Буфере Обмена и разрешать вставлять только RTF:

```
private void checkForClipboardFormat () {
 if (Clipboard.GetDataObject ().GetDataPresent (DataFormats.Rtf))
  contextMenuStrip1.Items[3].Enabled = true; //Включить пункт Вставить
 else contextMenuStrip1.Items [3].Enabled = false;
}
```

Мы вызовем его в обработчике события Activated дочерней формы:

```
  checkForClipboardFormat ();
```

Также напишем для richTextBox1 обработчики событий SelectionChanged (изменение выбора текста):

```
   if (richTextBox1.SelectionLength == 0)
    for (int i = 0; i < 2; i++) contextMenuStrip1.Items[i].Enabled = false;
   else
    for (int i = 0; i < 2; i++) contextMenuStrip1.Items[i].Enabled = true;
    //Включить Вырезать, Копировать
   checkForClipboardFormat ();
```

и TextChanged (изменение самого текста), имеющий схожие цели:

```
   if (richTextBox1.Text.Length < 1) {
    richTextBox1.Modified = false; //Пустой текст не будем предлагать сохранить
    for (int i = 1; i < 3; i++) contextMenuStrip1.Items [2].Enabled = false;
    //и копировать
   }
   else //Включить Копировать, Копировать все
    for (int i = 1; i < 3; i++) contextMenuStrip1.Items [2].Enabled = true;
   checkForClipboardFormat ();
```

Пункт меню "Вырезать" будет запрограммировать просто:

```
  if (richTextBox1.SelectionLength > 0) richTextBox1.Cut ();
```

"Копировать" тоже:

```
  if (richTextBox1.SelectionLength > 0) richTextBox1.Copy ();
```

"Копировать все" тоже:

```
   richTextBox1.SelectAll ();
   richTextBox1.Copy ();
   richTextBox1.DeselectAll ();
```

А вот обработчик пункта "Вставить" будет дополнительно проверять, есть ли в системном Буфере данные подходящего формата и не выделено ли что-то в тексте из richTextBox1:

```
   if (Clipboard.GetDataObject().GetDataPresent (DataFormats.Rtf)) {
    //Есть Rtf в буфере
    if (richTextBox1.SelectionLength > 0) { //И что-то выделено,
     //спросим, как вставлять - поверх выделенного или в конец?
     if (MessageBox.Show ("Вставить поверх выделения?", "Сообщение",
      MessageBoxButtons.YesNo) == DialogResult.No)
      richTextBox1.SelectionStart = richTextBox1.Text.Length;
    }
    richTextBox1.Paste ();
   }
```

Перед закрытием дочерней формы в обработчике события FormClosing нам придётся запросить у пользователя сохранение файла, если оно требуется, и дать ему возможность сохранить изменения или отказаться от этого:

```
   if (this.richTextBox1.Modified == false) return;
   saveFileDialog1.FileName = this.Text;
   var MBox = MessageBox.Show (
     String.Format ("Текст в файле {0} был изменен.\nСохранить изменения?",
     saveFileDialog1.FileName), "RichTextEditor",
     MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);
   if (MBox == DialogResult.No) return;
   if (MBox == DialogResult.Yes) {
    if (saveFileDialog1.ShowDialog () == DialogResult.OK) {
     try {
      richTextBox1.SaveFile (saveFileDialog1.FileName);
      richTextBox1.Modified = false;
```

```
      }
      catch (Exception) {
       MessageBox.Show ("Ошибка сохранения файла");
      }
      return;
     }
    }
```

Проект собирается. Теперь можно добавить в главное окно, например, инструменты панели ToolStrip - кнопки для дублирования пунктов меню и операций над текстом активного окна, таких как изменение цвета, размера или начертания шрифта и т.п., что обычно бывает в текстовых редакторах.

Покажем работу с кнопками ToolStrip на примере типового форматирования текста.
Пользуясь раскрывающимся списком на элементе toolStrip1, добавим на него три кнопки (Button). Для простоты очистим им свойство Image и правой кнопкой мыши в Конструкторе выберем для всех кнопок значение DisplayStyle = Text. Затем поставим свойство Text кнопок в значения "Ж", "К", "Ч" (жирный, курсив, подчёркивание) с соответствующими подсказками в свойствах ToolTipText. Также у всех кнопок установлено свойство CheckOnClick = True.
По клику на кнопках запрограммируем код для переключения начертаний шрифта контрола активного окна:

```
  private void cnangeStyle (FontStyle style) {
   Form2 activeChild = this.ActiveMdiChild as Form2;
   if (activeChild != null) {
    activeChild.changeStyle (style);
   }
  }

  private void toolStripButton1_Click (object sender, EventArgs e) {
   cnangeStyle (FontStyle.Bold);
  }

  private void toolStripButton2_Click (object sender, EventArgs e) {
   cnangeStyle (FontStyle.Italic);
  }

  private void toolStripButton3_Click (object sender, EventArgs e) {
   cnangeStyle (FontStyle.Underline);
  }
```

Публичный метод changeButtons будет вызываться из дочерней формы и управлять переключением состояний кнопок:

```
  public void changeButtons(int [] states) {
   toolStripButton1.Checked = ( states [0] != 0 ? true : false );
   toolStripButton2.Checked = ( states [1] != 0 ? true : false );
   toolStripButton3.Checked = ( states [2] != 0 ? true : false );
  }
```

В дочерней форме Form2.cs нетрудно увидеть, что функция `checkForClipboardFormat` вызывается всегда, когда нужно контролировать состояние наших кнопок - то есть, при активации формы (`Form2_Activated`), а также при изменении текущего выбора текста (`richTextBox1_SelectionChanged`) и изменении самого текста (`richTextBox1_TextChanged`). В конец этой функции перед закрывающей фигурной скобкой добавим блок, который соберёт информацию о

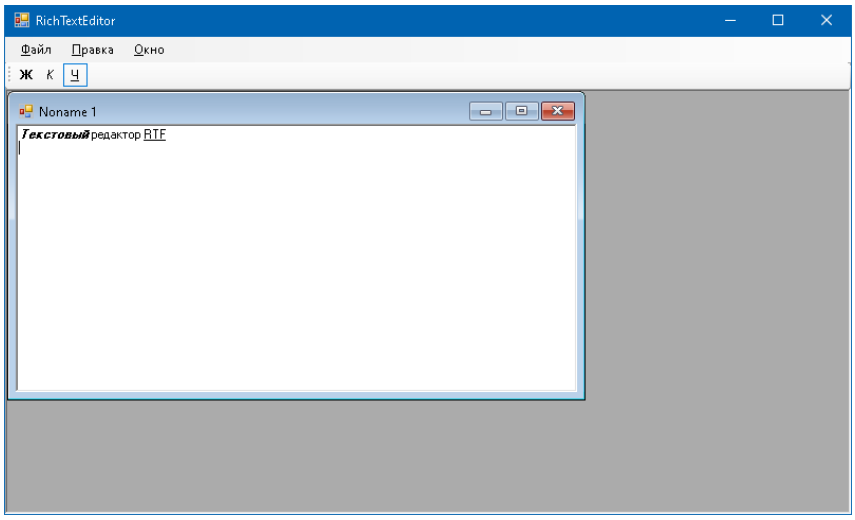
текущем состоянии текста и передаст её методу changeButtons родительской формы:

```
Form1 parent = this.MdiParent as Form1;
if (parent != null) {
 int [] states = new int [] {
  (int)richTextBox1.SelectionFont.Style & (int)FontStyle.Bold,
  (int)richTextBox1.SelectionFont.Style & (int)FontStyle.Italic,
  (int)richTextBox1.SelectionFont.Style & (int)FontStyle.Underline
 };
 parent.changeButtons (states);
}
```

Осталось не забыть реализовать метод changeStyle , который мы вызывали из главной формы и который будет непосредственно менять начертание шрифта:

```
public void changeStyle(FontStyle style) {
 Font f = new Font (richTextBox1.SelectionFont,
  style ^ richTextBox1.SelectionFont.Style); //переключить шрифт
 richTextBox1.SelectionFont = f; //установить новый шрифт
 checkForClipboardFormat ();
}
```

Вот что получилось:



По этому образцу вы сможете добавить и другие инструменты в многодокументное приложение.