Урок 5

Объектно - Ориентированное Программирование

Windows Forms. Класс Control. Элементы Windows Forms. Приложение “Швейцарский нож”. RTF-формат. Привязка данных. Веб-службы. Десериализация и сериализация XML.

Windows Forms с нуля

Форма без наследования

Создадим пустой проект. Добавим класс MainWindow. Добавим ссылку на System.Windows.Forms.

```
using System;
using System.Windows.Forms;
namespace MainWindow
{
    class MainWindow
    {
        static int count = 0;
        static Button button = new Button();
        static void Main()
        {
            Form form = new Form();
            form.Show();
            form.Text = "Main Form";
            button.Text = count.ToString(); ;
            button.Left = 100;
            button.Top = 40;
            button.Parent = form; // Элемент, на котором отображается button
            button.Click += Button_Click;
            Application.Run(form);
        }
        private static void Button_Click(object sender, EventArgs e)
        {
             button.Text=(++count).ToString();
        }
    }
}
```

Делегат EventHandler

Представляет метод, который будет обрабатывать событие, не имеющее данных.

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public delegate void EventHandler(object sender,EventArgs e)
```

[SerializableAttribute] - указание, что объекты данного класса подлежат сериализации

Наследование

Наследуя класс формы от базового класса, мы имеем возможность расширять функциональность базового класса.

```
using System;
using System.Windows.Forms;
namespace MainWindow_Nasledovanie
{
    class MainWindowInheritor : Form
    {
        Button button = new Button();

        public MainWindowInheritor()
        {
            button.Text = "0";
            button.Parent = this;
        }
    }

    class Program
    {
        static void Main()
        {
            MainWindowInheritor form = new MainWindowInheritor();
            Application.Run(form);
        }
    }
}
```

Пример приложения, написанного с нуля, демонстрирующего программу под Windows Forms

```
using System;
using System.Windows.Forms;
namespace MainWindow_Nasledovanie
{
   class MainWindowInheritor:Form
   {
      int count = 0;
      Button button1;
      public MainWindowInheritor()
      {
         InitializeComponent();
      }
      private void InitializeComponent()
      {
         button1 = new Button();
         SuspendLayout();
         // button1
         button1.Location = new System.Drawing.Point(114, 34);
         button1.Name = "button1";
         button1.Size = new System.Drawing.Size(75, 23);
         button1.Text = count.ToString();
         button1.Click += Button1_Click;
         // MainWindowInheritor
         ClientSize = new System.Drawing.Size(284, 261);
         Controls.Add(this.button1);
         Name = "MainWindowInheritor";
         ResumeLayout(false);
      }
      private void Button1_Click(object sender, EventArgs e)
      {
         button1.Text = (++count).ToString();
      }
   }

   class Program
   {
      static void Main()
      {
         MainWindowInheritor form = new MainWindowInheritor();
         Application.Run(form);
      }
   }
}
```

Класс Control

Control — самый важный класс в Windows Forms — он один позволяет получить несчетное количество инструментов для создания приложений Windows Forms. Control не только базовый класс для таких элементов управления, как кнопки, деревья, панели инструментов и меню, но и для Form — класса, инкапсулирующего главное окно приложения Windows Forms и выполняющего другие функции в диалоговых окнах.

Практически всё на экране Microsoft Windows состоит из элементов управления. Действительно, в общем случае, элемент управления можно определить как визуальный объект. Обычно элементы управления занимают прямоугольную область экрана, хотя могут быть непрямоугольными или даже скрытыми и выполнять команды, поступающие с клавиатуры или мыши. Класс Control поддерживает множество событий (и соответствующих методов ввода) для обработки ввода информации, например, событий клавиатуры KeyDown, KeyUp и KeyPress и мыши — MouseDown, MouseUp и MouseMove. Также элементы управления должны «перерисовывать» себя на экране — для этого служит метод OnPaint.

Превращая информацию об операциях пользователя в простые события, элементы управления служат уровнем абстрагирования между пользователем и приложением. Например, для нормальной работы элемента Button достаточно определить отображаемый на кнопке текст и обработчик события Click. Об остальном позаботится элемент управления.

Родители и потомки

Одно из самых важных, определяемых классом Control, свойств — Parent; оно указывает на другой объект типа Control. В процессе исполнения приложения на экране отображаются только элементы управления с корректно определенным свойством Parent. Положение элемента всегда задается относительно «родителя» и отображается на поверхности последнего. Часть элемента управления, выступающая за края родителя, не видна.

Форма позволяет перечислить все элементы управления своего набора, используя цикл for:

```
for(inti=0;i<Controls.Count; i++)
{
  Control ctrl = Controls[i];
  MessageBox.Show(ctrl.ToString());
}
```

Шрифты и цвет

У всех элементов управления есть свойство Text, хотя некоторые из них (например, полоса прокрутки) не отображают никакого текста.

Элемент управления наследует исходные свойства Font, BackColor и ForeColor у своего родителя. При изменении свойств родителя свойства «дочек» также изменяются. Однако, если элементам управления явно задать другие свойства, они сохранят их и перестанут зависеть от родителей.

Некоторые элементы управления

Кнопки и двоичные переключатели

Элемент управления «кнопка» называется так не потому, что его можно щелкать, - в конечном итоге, все элементы по щелчку инициируют событие Click, а главным образом по той причине, что при щелчке он визуально ведет себя как кнопка. Все три типа кнопок происходят от абстрактного класса ButtonBase:

Control
  ButtonBase (абстрактный)
    Button
    CheckBox
    RadioButton

Обычно Button используется для инициирования действий, CheckBox — для установки и снятия флажков, а RadioButton обычно служит для выбора одного из нескольких взаимоисключающих вариантов.

Кнопка (Button)

По сравнению с ButtonBase класс Button привносит мало нового. В нем практически всегда определяется обработчик события Click.

Флажок (CheckBox)

Обычно элемент управления CheckBox — это небольшой прямоугольник, расположенный слева от текста. Однако, если свойству Appearance задать значение Appearance.Button (по умолчанию Appearance.Normal), элемент управления будет схож с кнопкой, за исключением того, что будет по щелчку «залипать» и «отлипать». Вид элемента управления CheckBox можно изменить, задав свойству CheckAlign значение из перечисления ContentAlignment. По умолчанию используется значение ContentAlignment.MiddleLeft, то есть флажок по вертикали выравнивается по оси текста и располагается слева от него. Булево свойство Checked указывает текущее состояние флажка — «установлен» или «сброшен». Событие CheckedChanged информирует об изменении состояния.

Флажок (RadioButton)

Этот элемент управления обычно представляет собой кружок, расположенный слева от текста. Как и у CheckBox, у класса RadioButton есть свойство Appearance, которому можно задать значение Appearance.Button, тогда элемент управления выглядит как кнопка, а также свойство CheckAlign, позволяющее менять положение кружка относительно текста.

Как и у CheckBox, у RadioButton есть булево свойство Checked, при изменении которого инициируется событие CheckedChanged.

Элементы управления с поддержкой редактирования текста

В Windows Forms есть несколько элементов управления, служащих для ввода или редактирования текста:

Control

- TextBoxBase (абстрактный)
  - MaskedTextBox
  - TextBox
    - DataGridTextBox
    - DataGridViewTextBoxEditingControl
  - RichTextBox

Самое важное свойство элемента с поддержкой редактирования текста, это, конечно же, Text, содержащее текст, отображаемый в элементе управления. Программа инициализирует текст в элементе управления простым определением значения Text, и получает текст, введённый или изменённый пользователем, обращением к свойству Text. Если txtbox объект типа TextBox, удалить из него текст просто:

```
txtbox.Text = "";
```

А добавляют строку в конец существующего текста так:

```
txtbox.Text += " добавленый текст"
```

Свойство Multiline определяет, может ли элемент управления получать и отображать многострочный текст. По умолчанию оно равно false в TextBox и MaskedTextBox и true — в RichTextBox. Свойство WordWrap (перенос по словам) знакомо пользователям блокнота (Notepad). Оно применимо только к многострочным элементам управления и по умолчанию равно true. Чтобы элемент управления отображал нередактируемый текст, свойству ReadOnly присваивают значение true.

Текстовое поле (TextBox)

Это простейший элемент управления с поддержкой редактирования текста, но вместе с тем он достаточно сложен, чтобы стать основой блокнота Windows.

TextBox добавляет всего несколько свойств к имеющимся в TextBoxBase. Наверное, самое важное из них — это ScrollBars, которому присваиваются значения из перечисления ScrollBars.Vertical, ScrollBars.Horizontal или ScrollBars.Both. Значение по умолчанию — ScrollBars.None, то есть, полосы прокрутки не отображаются, даже если этого требует длина текста. (Если значение свойства WordWrap равно true, горизонтальные полосы прокрутки не отображаются, независимо от значения ScrollBars).

Если TextBox используется для ввода пароля, нужно указать в PasswordChar символ, который будет отображаться при вводе. В этом отношении также полезно свойство CharacterCasing. Если присвоить этому свойству значение из перечисления CharacterCasing.Lower или CharacterCasing.Upper, вводимые символы будут приводиться в нижний или верхний регистр соответственно.

Список (ListBox)

Список содержит прокручиваемый набор объектов. По умолчанию пользователь в праве выбрать один (или несколько) элементов, используя клавиатуру или мышь. Выбранные объекты выделяются.

Отображаемые элементы списка задаются свойством Items — объектом типа

ListBox.ObjectCollection. В классе ListBox.ObjectCollection определён метод Add, позволяющий добавлять новые объекты в набор. В списке отображаются текстовые названия элементов, возвращаемые методом ToString каждого объекта. Свойство Sorted объекта ListBox обеспечивает автоматическую сортировку элементов. После этого можно использовать свойство Items объекта ListBox для обращения к отдельным объектам наподобие того, как это делают с массивом.

Индикатор хода процесса (ProgressBar)

Основные свойства ProgressBar — это целочисленные Minimum, Maximum и Value. По мере выполнения процесса свойству Value присваивают возрастающие значения в диапазоне от Minimum до Maximum.

Дата и время

В Windows Forms есть два элемента управления для работы с датами. Month Calendar предназначен для получения информации о дате от пользователя, а DateTimePicker позволяет получать ещё и время, но чаще всего используется только для ввода даты.

Календарь на месяц (MonthCalendar)

Элемент управления MonthCalendar отображает календарь с текущей датой, выделенной красным кружком и отображаемой внизу элемента управления. С помощью мыши или клавиатуры пользователь может выбрать до семи последовательных дней. Для этого надо выбрать первую, а затем, удерживая клавишу Shift, выделить последнюю дату диапазона. Для просмотра другого месяца нужно щелкнуть одну из стрелок, расположенных вверху элемента управления, или нажать клавишу Page Up или Page Down.

В программе узнать выбранный диапазон позволяют два свойства типа DateTime:SelectionStart и SelectionEnd. (В качестве альтернативы можно использовать свойство SelectionRange — объект типа SelectionRange со свойствами Start и End.)

Чтобы получать уведомление об изменении выбранных дат, устанавливают обработчик события DateChanged, основанный на DateRangeEventHandler. В сообщении о событии содержится объект типа DataRangeEventArgs с двумя свойствами типа DateTime — Start и End.

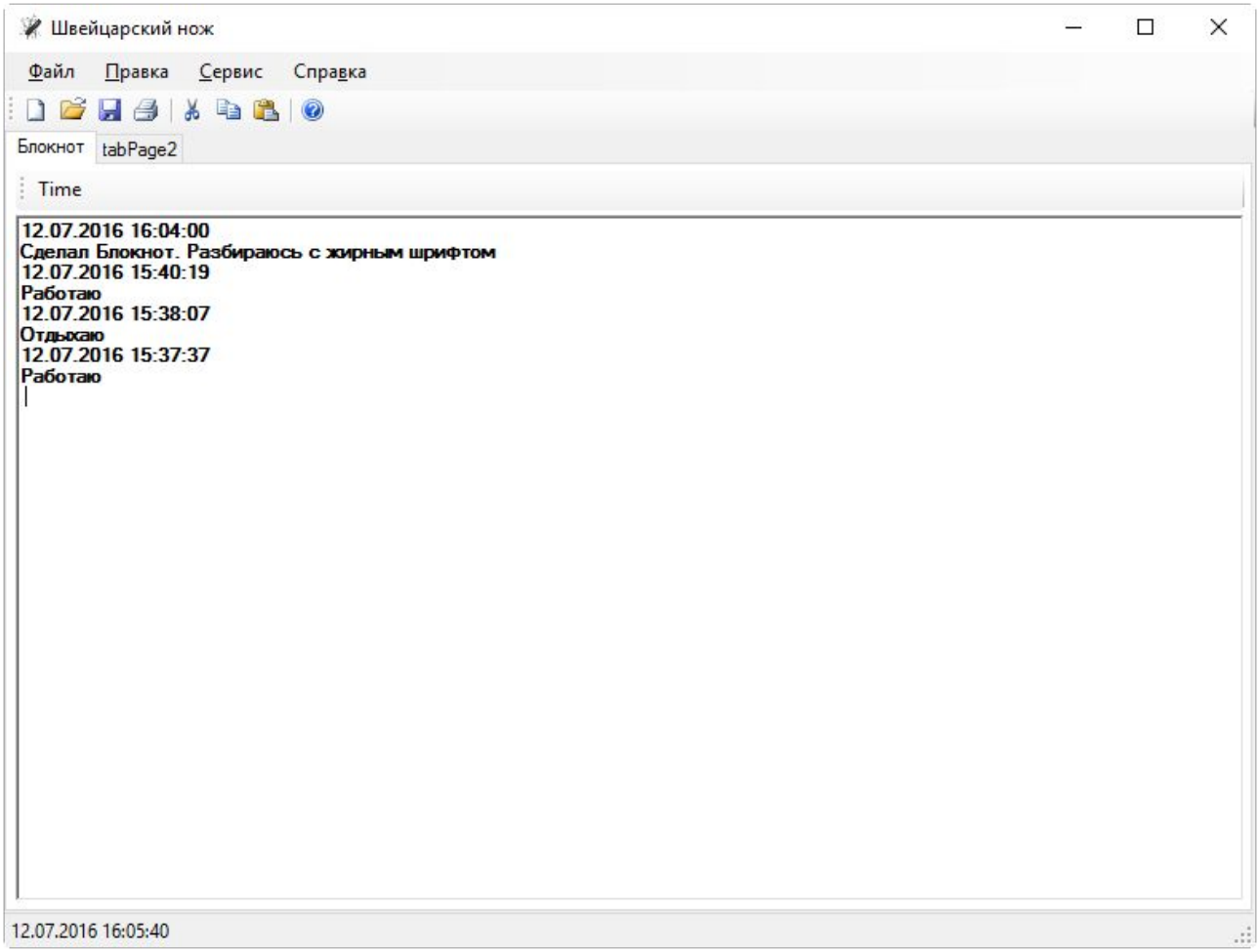
Элемент выбора даты и времени (DateTimePicker)

Хотя у DateTimePicker больше возможностей, чем у MonthCalendar, так как он позволяет выбирать дату и время, его обычно используют только для определения даты (впрочем, это и есть режим по умолчанию). В отличие от MonthCalendar, у DateTimee Picker нет возможности выбора диапазона дат.

С первого взгляда DateTimePicker походит на поле со списком с текстовым полем и стрелкой справа. При инициализации элемента управления из программы нужно задать свойству Value значение объекта типа DateTime, в противном случае ему будет присвоено значение DateTime.Now.

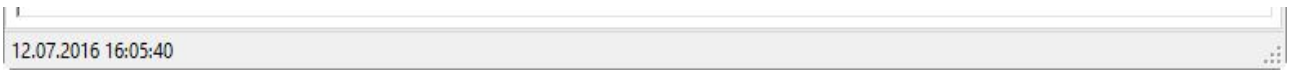
Приложение “Швейцарский нож”

Начнем разработку приложения, которое будет помогать “по хозяйству”. Мы на уроках добавим туда несколько мини-приложений.



Текущее время

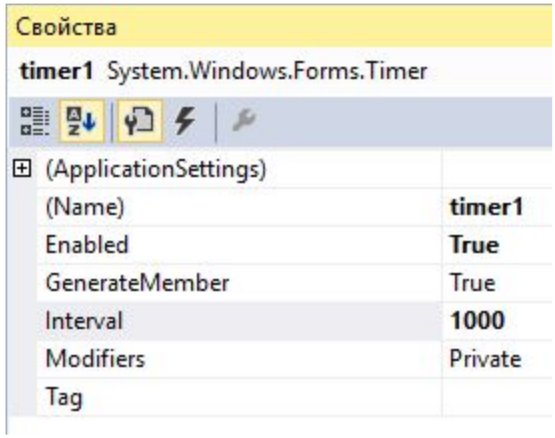
Начнем с простого. Сделаем, чтобы внизу в строке состояния (StatusStrip) отображалось текущее время.



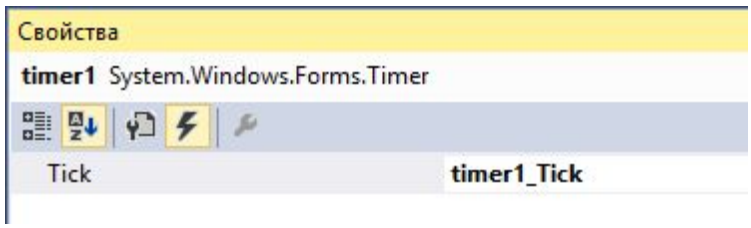
Для начала добавим время и дату в строку StatusStrip. Для этого разместим на форме элемент StatusStip, а на неё добавим элемент ToolStripStatusLabel под именем tsslCurrentTime.

Добавьте элемент Timer на форму.

Задайте его свойства как на рисунке:



Щелкните на обработчике событий от таймера и добавьте текст:




```
private void timer1_Tick(object sender, EventArgs e)
{
  tsslCurrentTime.Text = DateTime.Now.ToLongTimeString();
}
```

Запустите и убедитесь, что на форме показывается время.

Блокнот

Создадим блокнот с возможностью вставки текущей даты и времени с использованием RTF-формата.

```
using System;
using System.Windows.Forms;
namespace SwissKnife
{
    public partial class Form1 : Form
    {
        string notepadFilename = "notepad.rtf";
        public Form1()
        {
            InitializeComponent();
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            tsslCurrentTimer.Text = DateTime.Now.ToString();
        }
        private void tsbCurrentTimeInsert_Click(object sender, EventArgs e)
        {
// Вставляем текст в формате rtf
            string rtftext = @"{\rtf1\b " +DateTime.Now.ToString() + @"\b0\par\line}}";
// Устанавливаем фокус ввода
```

```
            rtbNotepad.Focus();
 // Устанавливаем курсор в начало текстового поля
            rtbNotepad.Select(0, 0);
 // Копируем в буфер обмена данные с указанием типа данных
            Clipboard.SetData(DataFormats.Rtf, (object)rtftext);
// Вставляем данные из буфера обмена
            rtbNotepad.Paste();
// Устанавливаем курсор для ввода данных
            rtbNotepad.Select(rtbNotepad.SelectionStart - 1, 0);
        }
        private void tsmiExit_Click(object sender, EventArgs e)
        {
            Close();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            if (System.IO.File.Exists(notepadFilename)) rtbNotepad.LoadFile(notepadFilename);
        }
        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            rtbNotepad.SaveFile(notepadFilename);
        }
        private void tsmiSaveAs_Click(object sender, EventArgs e)
        {
            SaveFileDialog dlg = new SaveFileDialog();
            dlg.Filter = "Файл rtf|*.rtf";
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                notepadFilename = dlg.FileName;
                rtbNotepad.SaveFile(notepadFilename);
            }
        }
        private void tsmiSave_Click(object sender, EventArgs e)
        {
            rtbNotepad.SaveFile(notepadFilename);
        }
        private void tsmiLoad_Click(object sender, EventArgs e)
        {
            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Filter = "Файл rtf|*.rtf";
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                notepadFilename = dlg.FileName;
                rtbNotepad.LoadFile(notepadFilename);
            }
        }
    }
}
```

Формат Rich Text (RTF)

Данный формат был определён фирмой Microsoft, как стандартный формат для обмена текстовыми документами. Следовательно, по назначению этот формат подобен SYLK-формату для электронных таблиц. RTF поддерживается многими продуктами фирмы Microsoft. Так, например, начиная с версии 2.0, он введён в Windows в качестве Clipboard-формата, благодаря чему возможен обмен данными между различными прикладными программами Windows. Кроме того, RTF-формат поддерживается программами WORD для Macintosh, начиная с версии 3.X, и WORD для PC, начиная с версии 4. X.

Более подробно про RTF можно почитать в Интернете (см. раздел Дополнительно)

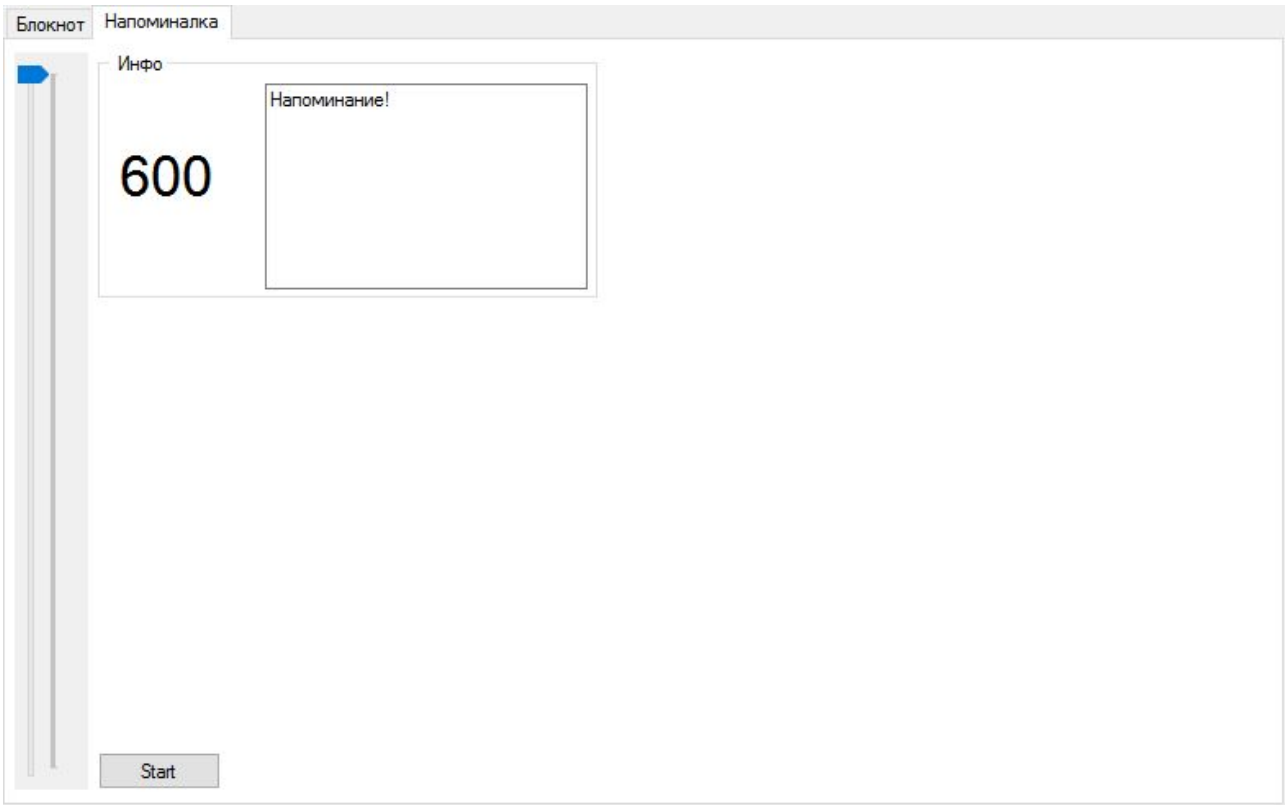
Советы по дальнейшей разработке программы

Для изучения rtf формата можно добавить на какую-нибудь свободную кнопку обработчик события с кодом, который будет выводить содержимое rtf файла, например, так:

```
MessageBox.Show(rtbNotepad.Rtf);
```

Напоминалка

Сделаем себе напоминалку-таймер. На его примере покажем связывание данных. Для этого разместите на вкладке элементы TrackBar, Button, Label, Timer и TextBox.



Для кнопки сделайте обработчик события со следующим кодом:

```
private void btnStartAlert_Click(object sender, EventArgs e)
{
    timer2.Enabled = !timer2.Enabled;
}
```

В обработчик события Tick таймера следующий код:

```
private void timer2_Tick(object sender, EventArgs e)
{
    if (trackBar1.Value > 0) trackBar1.Value--;
    else
    {
        timer2.Stop();
        System.Media.SystemSounds.Exclamation.Play();
        MessageBox.Show(tbAlertMessage.Text);
    }
}
```

Теперь кнопка будет работать и запускать и останавливать таймер, а с помощью TrackBar можно будет устанавливать количество секунд.

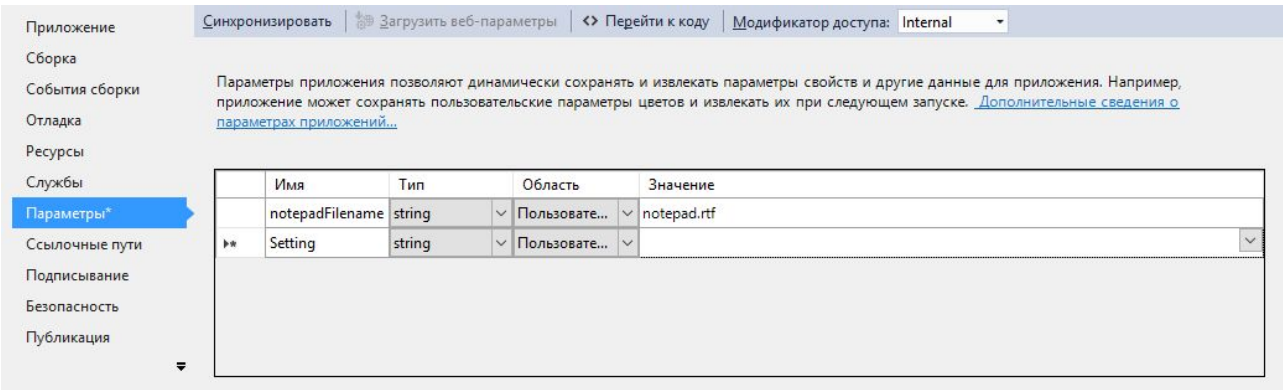
Теперь свяжем свойство Text метки со свойством Value TrackBar'а. Для этого добавьте в конструктор формы следующий код:

```
public Form1()
{
    InitializeComponent();
    lblTimerAlert.DataBindings.Add("Text", trackBar1, "Value");
}
```

С помощью DataBindings мы легко можем связывать свойства элементов Windows Forms с другими свойствами элементов Windows Forms.

Сохранение настроек приложения

В нашей программе есть параметр с названием файла блокнота notepadFilename. Давайте научимся сохранять этот параметр в параметрах приложения. Для этого выберите в Обозревателе свойства вашего проекта и на вкладке Параметры создайте параметр notepadFilename, как показано на рисунке. **Сохраните параметры.** При этом создастся класс Properties.Settings, и мы воспользуемся им для хранения этого поля.



В обработчики событий OnLoad и OnClosed добавьте следующий код:

```
private void Form1_Load(object sender, EventArgs e)
{
   notepadFilename = Properties.Settings.Default.notepadFilename;
   if (System.IO.File.Exists(notepadFilename)) rtbNotepad.LoadFile(notepadFilename);
}

private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
   rtbNotepad.SaveFile(notepadFilename);
   Properties.Settings.Default.notepadFilename = notepadFilename;
   Properties.Settings.Default.Save();
}
```

Веб-службы

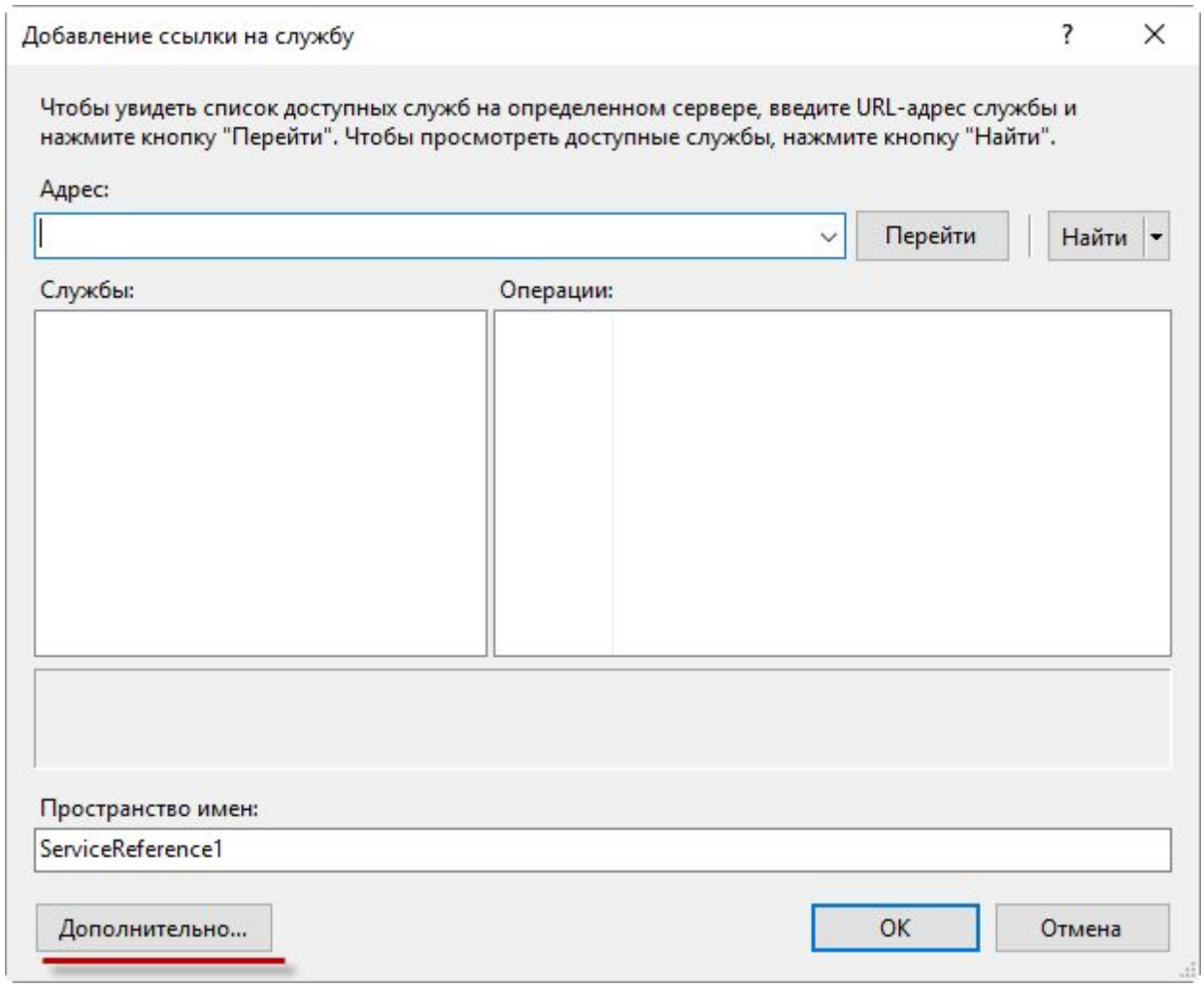
Веб-служба - это программная система (более конкретно - откомпилированная библиотека динамической компоновки, т.е. файл формата DLL в папке bin приложения), расположенная на удалённом сервере, к которой можно обращаться со своего клиентского компьютера. При этом возможности удалённой веб-службы будут реализованы в вашем Windows или веб-приложении, а пользователь, при достаточной скорости обмена данными, может даже не заметить обращения вашей программы к удалённому серверу.

Получение погоды

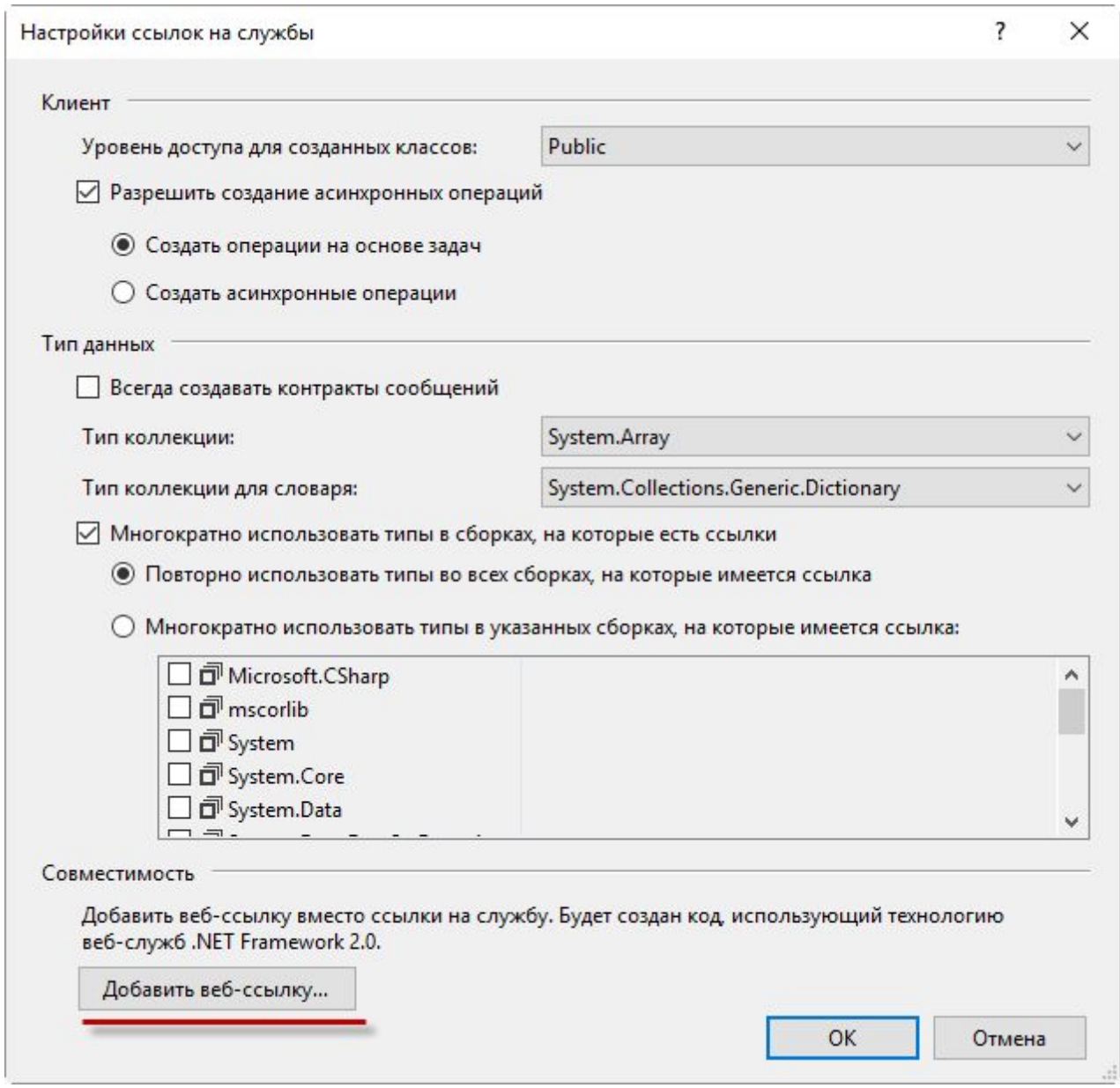
Веб-служба, обсуждаемая в этом примере, возвращает прогноз погоды в городе, который пользователь задает в запросе к службе. Эта веб-служба поддерживает два метода: GetCitiesByCountry и GetWeather. На вход первой функции подают название страны, где хотят получить прогноз погоды, а на выходе функции получают перечисление городов этой страны, для которых веб-служба готова сделать прогноз погоды.

На вход второй функции GetWeather подают названия города и страны, а на выходе функции получают XML-строку, содержащую прогнозируемые параметры погоды.

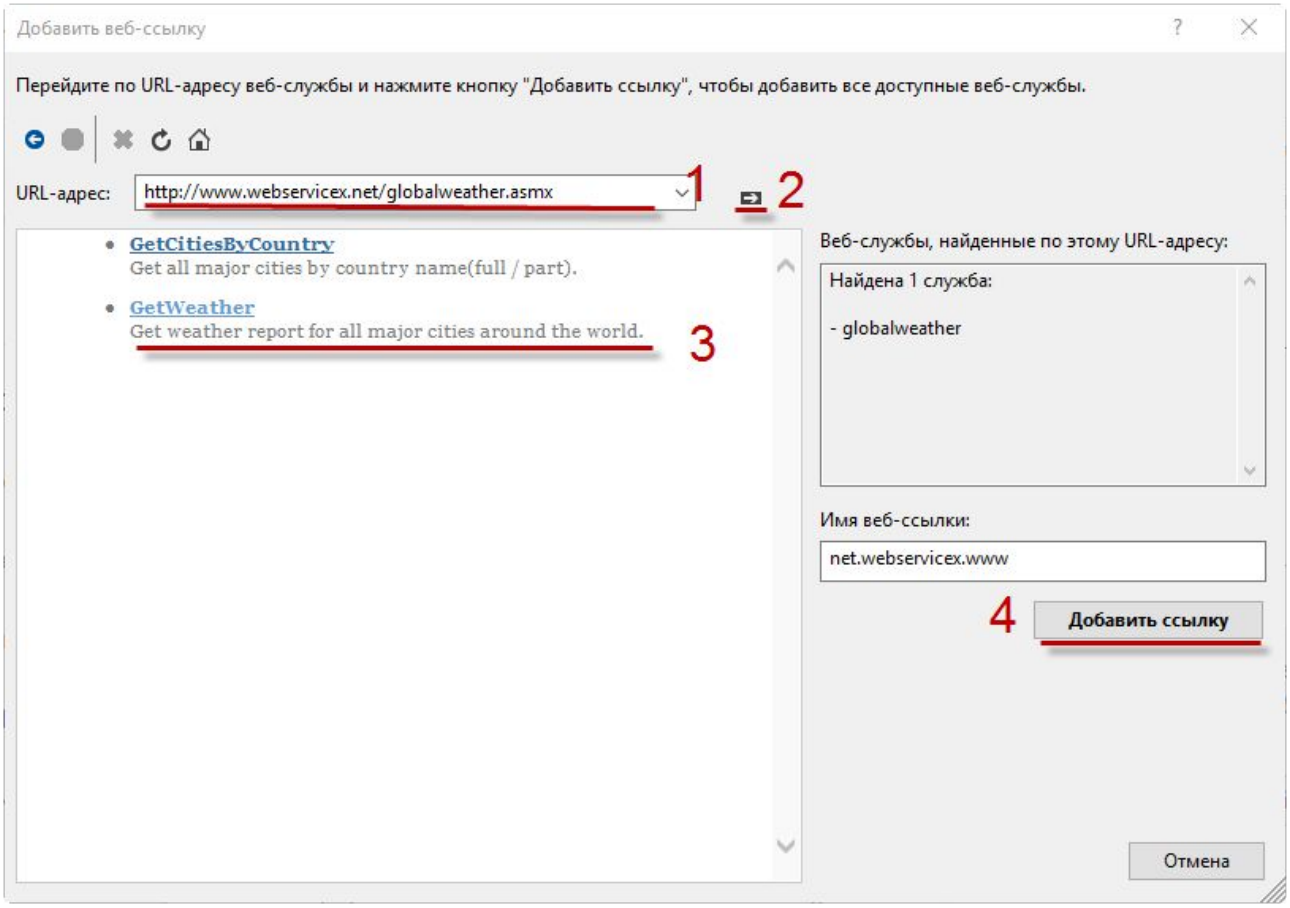
Для демонстрации работы веб-службы создадим консольное приложение. Далее выберите ссылки и нажмите Добавление ссылки на службу. Нажмите далее.



Выберите Дополнительно...



Нажмите “Добавить веб-ссылку...”



Введите в URL-адрес ссылку http://www.webservicex.net/globalweather.asmx и нажмите добавить ссылку. Веб-служба подключена.

```
using System;
namespace PrognozPogody2
{
   class Program
   {
      static void Main(string[] args)
      {
         var weather = new net.webservicex.www.GlobalWeather();
         var str_xml = weather.GetWeather("Moscow", "Russia");
         Console.WriteLine(str_xml.ToString());
         Console.ReadKey();
      }
   }
}
```

Данная программа выведет нечто подобное:

```
<?xml version="1.0" encoding="utf-16"?>
<CurrentWeather>
  <Location>Moscow / Vnukovo , Russia (UUWW) 55-39N 037-16E</Location>
  <Time>Jul 14, 2016 - 08:30 AM EDT / 2016.07.14 1230 UTC</Time>
  <Wind> from the NW (320 degrees) at 4 MPH (4 KT) (direction variable):0</Wind>
  <Visibility> greater than 7 mile(s):0</Visibility>
  <SkyConditions> mostly cloudy</SkyConditions>
  <Temperature> 71 F (22 C)</Temperature>
  <DewPoint> 60 F (16 C)</DewPoint>
  <RelativeHumidity> 68%</RelativeHumidity>
  <Pressure> 29.94 in. Hg (1014 hPa)</Pressure>
  <Status>Success</Status>
</CurrentWeather>
```

Это данные в так называемом XML-формате. Теперь давайте посмотрим, как мы можем разобрать этот формат.

Десериализуем полученные данные

Для десериализации изучим полученный XML-файл и создадим класс с автоматическими свойствами, соответствующими названиям полей в XML-файле. Учтите, что такая простая десериализация была возможна из-за простоты XML-файла, и такой способ подойдет далеко не всегда.

```
using System;
using System.Xml.Serialization;
using System.IO;
namespace PrognozPogody
{
   [Serializable]
   public class CurrentWeather
   {
      public string Location { get; set; }
      public string Time { get; set; }
      public string Wind { get; set; }
      public string Visibility { get; set; }
      public string SkyConditions { get; set; }
      public string Temperature { get; set; }
      public string DewPoint { get; set; }
      public string RelativeHumidity { get; set; }
      public string Pressure { get; set; }
      public string Status { get; set; }
      public static void LoadFromXmlFormat(ref CurrentWeather obj, string stringXML)
      {
         XmlSerializer xmlFormat = new XmlSerializer(typeof(CurrentWeather));
         TextReader sr = new StringReader(stringXML);
         obj = (xmlFormat.Deserialize(sr) as CurrentWeather);
         sr.Close();
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         var погода=new net.webservicex.www.GlobalWeather();
         var строка_xml = погода.GetWeather("Moscow", "Russia");
         CurrentWeather cw = new CurrentWeather();
         CurrentWeather.LoadFromXmlFormat(ref cw,строка_xml);
         Console.WriteLine(cw.Location);
         Console.WriteLine(cw.Temperature);
         Console.WriteLine(cw.Time);
         Console.ReadKey();
      }
   }
}
```

Парсируем полученные данные

Еще один способ разобрать XML это прочитать XML файл с помощью класса XMLReader. Вот пример Windows приложения, в котором получается прогноз погоды с помощью разбора XML-файла:

```
using System;
using System.Windows.Forms;
// Приложение, потребляющее сервис удаленной веб-службы прогноза погоды. Приложение в
// текстовом поле TextBox демонстрирует XML-строку с параметрами погоды для города, указанного //
во входных параметрах при обращении к веб-службе. Также выводит в текстовую метку значение
// температуры в этом городе
// На основе примера из книги:"Зиборов В.В. - Visual C# 2012 на примерах"
namespace WeatherXMLParse
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }
      private void btnParseXML_Click(object sender, EventArgs e)
      {
// Создаём клиентское приложение веб-службы:
// http://www.webservicex.net/globalweather.asmx
// Эта веб-служба часто бывает перегружена, и поэтому может выдать сообщение: "Server is too busy"
// Создание экземпляра прокси-класса:
         var weather = new net.webservicex.www.GlobalWeather();
         var str_XML = weather.GetWeather("Moscow", "Russia");
         textBox2.Text = str_XML;
         var Документ = new System.Xml.XmlDocument();
// Загрузка строки XML в XML-документ
         Документ.LoadXml(str_XML);
         var reader = new System.Xml.XmlNodeReader(Документ);
         var name = String.Empty;
         var temp = String.Empty;
// Цикл по узлам XML-документа:
         while (reader.Read() == true)
         {
// Читаем последовательно каждый узел, выясняя тип узла:
            if (reader.NodeType == System.Xml.
                  XmlNodeType.Element) name = reader.Name;
// Каждый раз запоминаем имя узла
            if (reader.NodeType != System.Xml.
                  XmlNodeType.Text) continue;
            if (name == "Temperature")
            {
               temp = reader.Value;
               break;
            }
         }
 // Выход из цикла, когда прочитали данные узла "Temperature"
         textBox5.Text = "Температура воздуха в Москве: " + temp;
      }
   }
```

Домашнее задание

1. Напишите программу созданную на уроке со своими собственными именами элементов разработав собственную систему именований (lblStatus, lStatus или labelStatuc и т.д)
2. Добавить в “Швейцарский нож” утилиту расчёта длительности жизни с использованием двух DateTimePicker и Label
3. Добавить в “Швейцарский нож” утилиту получения курса валют с использованием службы http://www.cbr.ru/DailyInfoWebServ/DailyInfo.asmx
   а) вывести в формате XML;
   б)**произвести парсинг или десериализацию XML-файла.

Дополнительные материалы

1. SerializableAttribute
2. Веб-служба
3. Описание rtf формата

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Петцольд Ч. “Программирование с использованием Windows Forms”, 2006 г.
2. “Зиборов В.В. - Visual C# 2012 на примерах”, БВХ-Петербург, 2013
3. “Язык программирования C# 5.0 и платформа .NET 4.5”. Эндрю Троелсен, Питер 2013 г.
4. MSDN