PropertyGrid FAQ

Автор: *Алексей Кирюшкин*
The RSDN Group

Источник: *RSDN Magazine #3-2006*

Опубликовано: *06.12.2006*
Исправлено: *18.02.2007*
Версия текста: *1.6*

Тестовый проект (VS2005)

PropertyGrid – удобный компонент для визуального редактирования свойств объектов. Объект для редактирования задается в дизайнере WinForms, либо непосредственно в коде:

```
private PersonData _personData = new PersonData();
propertyGrid1.SelectedObject = _personData;
```

Хотя многие стандартные типы PropertyGrid редактировать умеет (см. рисунок 1), любое практическое применение требует все же ручной доводки.

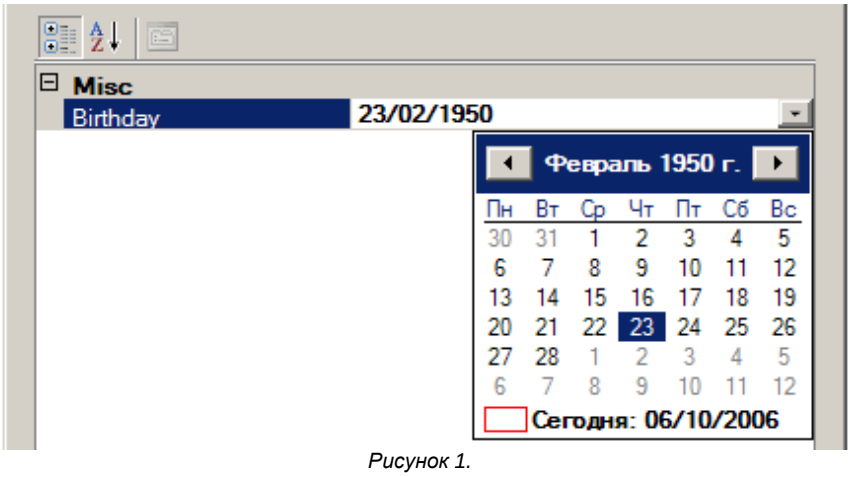

Рисунок 1.

В данном FAQ собраны ответы на некоторые вопросы, возникающие при использовании PropertyGrid.

Как заменить имя переменной в левой колонке "человеческим" именем свойства?

Для этого предназначен атрибут ***DisplayName***:

```
using System.ComponentModel;
...
      [DisplayName("День рождения")]
      public DateTime Birthday
      {
         get { return _birthday; }
         set { _birthday = value; }
      }
```

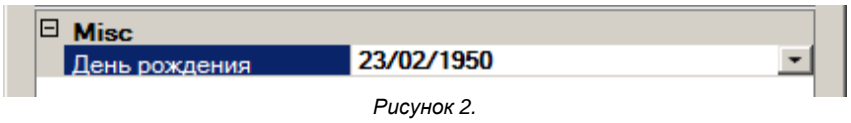

Рисунок 2.

Как отобразить расширенную подсказку по свойству в нижнем окне?

Для этого предназначен атрибут ***Description***:

```
      [DisplayName("День рождения")]
      [Description("День рождения он день рождения и есть")]
      public DateTime Birthday
      {
         get { return _birthday; }
         set { _birthday = value; }
      }
```

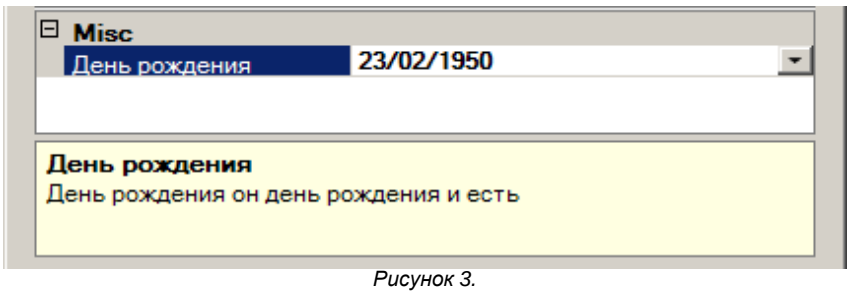

Рисунок 3.

Как сгруппировать свойства по категориям?

Используйте атрибут ***Category***:

```
[DisplayName("ФИО")]
[Description("Фамилия Имя Отчество")]
[Category("1. Идентификация")]
public string Name
{
  get { return _name; }
  set { _name = value; }
}

/// <summary>
/// День рождения
/// </summary>
[DisplayName("День рождения")]
[Description("День рождения он день рождения и есть")]
[Category("2. Общие")]
public DateTime Birthday
{
  get { return _birthday; }
  set { _birthday = value; }
}
```

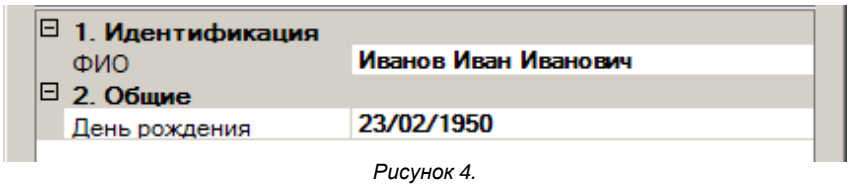

Рисунок 4.

Как отобразить свойство, недоступное для редактирования?

Можно либо само свойство сделать read-only (оставив только get), либо использовать атрибут ***ReadOnly***:

```
[DisplayName("ID")]
[Description("Идентификатор")]
[Category("1. Идентификация")]
[ReadOnly(true)]
public int Id
{
  get { return _id; }
  set { _id = value; }
}
```

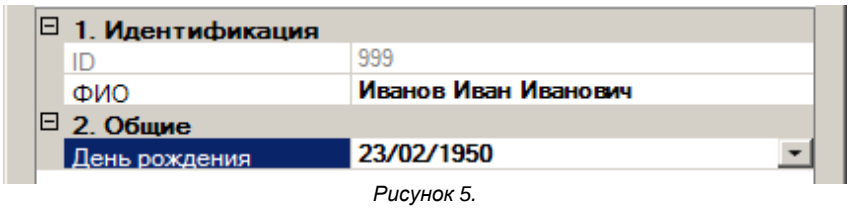

Рисунок 5.

Как заменить стандартные True/False в отображении свойств типа bool?

Используйте атрибут ***TypeConverter***:

```
[DisplayName("Наличие страховки")]
[Description("Наличие страховки")]
[Category("3. Дополнительно")]
[TypeConverter(typeof(BooleanTypeConverter))]
public bool Insurance
{
  get { return _insurance; }
  set { _insurance = value; }
}
```

Реализация ***BooleanTypeConverter*** проста:

```
class BooleanTypeConverter : BooleanConverter
{
  public override object ConvertTo(ITypeDescriptorContext context,
    CultureInfo culture,
    object value,
    Type destType)
  {
    return (bool)value ?
      "Есть" : "Нет";
  }

  public override object ConvertFrom(ITypeDescriptorContext context,
    CultureInfo culture,
    object value)
  {
    return (string)value == "Есть";
  }
}
```

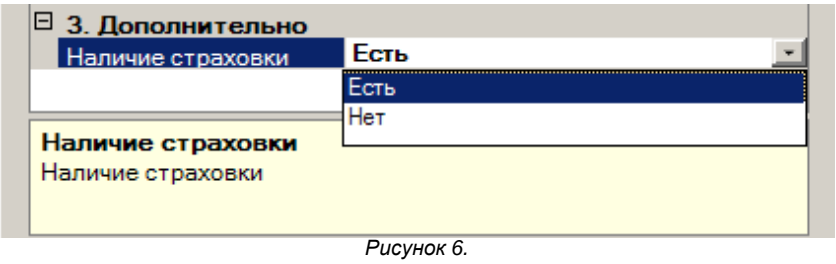

Рисунок 6.

Как заменить стандартное отображение имен членов перечисления?

Необходимо задать атрибут ***Description*** с нужным именем для каждого члена перечисления:

```
enum SEX
{
  [Description("Муж.")]
  Man,
  [Description("Жен.")]
  Woman,
  [Description("Неизв.")]
  Unknown
}
```

Реализовать ***EnumTypeConverter***, осуществляющий преобразование к строке с учетом атрибута ***Description***:

EnumTypeConverter

и задать атрибут ***TypeConverter*** для отображаемого свойства:

```
[DisplayName("Пол")]
[Description("Пол")]
[Category("2. Общие")]
[TypeConverter(typeof(EnumTypeConverter))]
public SEX Sex
{
  get {return _sex;}
  set {_sex = value;}
}
```

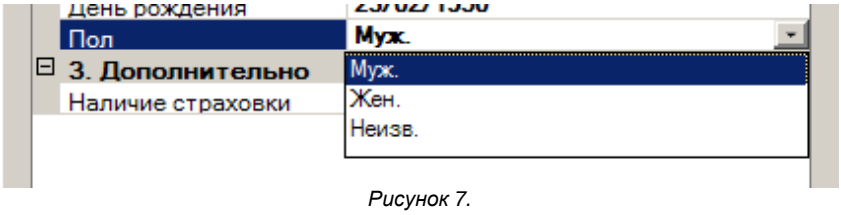

Рисунок 7.

Как показать свою картинку для каждого значения из перечисления?

Необходимо реализвать своего наследника от ***UITypeEditor*** с кодом отрисовки (в данном случае изображения хранятся в ресурсах с именами, соответствующими именам членов перечисления):

```
/// <summary>
/// Добавляет картинки, соответствующие каждому члену перечисления
/// </summary>
public class SexEditor : UITypeEditor
{
  public override bool GetPaintValueSupported(ITypeDescriptorContext context)
  {
    return true;
  }

  public override void PaintValue(PaintValueEventArgs e)
  {
    // картинки хранятся в ресурсах с именами, соответствующими
    // именам каждого члена перечисления SEX
    string resourcename = ((SEX)e.Value).ToString();

    // достаем картинку из ресурсов
    Bitmap sexImage =
      (Bitmap)Resources.ResourceManager.GetObject(resourcename);
    Rectangle destRect = e.Bounds;
    sexImage.MakeTransparent();

    // и отрисовываем
    e.Graphics.DrawImage(sexImage, destRect);
  }
}
```

и привязать его с помощью атрибута ***Editor*** к редактируемому свойству:

```
[DisplayName("Пол")]
[Description("Пол")]
[Category("2. Общие")]
[TypeConverter(typeof(EnumTypeConverter))]
[Editor(typeof(SexEditor), typeof(UITypeEditor))]
public SEX Sex
{
  get { return _sex; }
  set { _sex = value; }
}
```

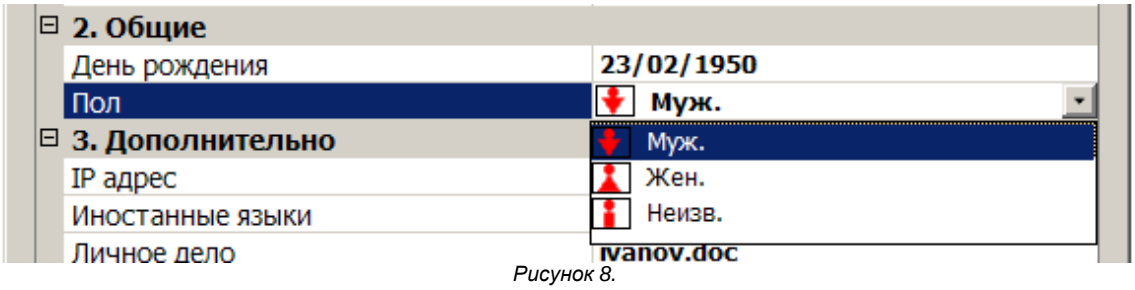

Рисунок 8.

Как организовать выбор значения из выпадающего списка, формируемого программно?

Необходимо реализовать ***TypeConverter***, предоставляющий список, из которого можно будет делать выбор:

```
/// <summary>
/// TypeConverter для списка должностей
/// </summary>
class PostTypeConverter : StringConverter
{
  /// <summary>
  /// Будем предоставлять выбор из списка
  /// </summary>
  public override bool GetStandardValuesSupported(
    ITypeDescriptorContext context)
  {
    return true;
  }

  /// <summary>
  /// ... и только из списка
  /// </summary>
  public override bool GetStandardValuesExclusive(
    ITypeDescriptorContext context)
  {
    // false - можно вводить вручную
    // true - только выбор из списка
    return true;
  }

  /// <summary>
  /// А вот и список
  /// </summary>
  public override StandardValuesCollection GetStandardValues(
    ITypeDescriptorContext context)
  {
    // возвращаем список строк из настроек программы
    // (базы данных, интернет и т.д.)
    return new StandardValuesCollection(Settings.Default.PostList);
  }
}
```

В данном случае возвращается список строк из настроек программы. Затем нужно задать этот класс в качестве параметра атрибута ***TypeConverter*** для редактируемого свойства:

```
/// <summary>
/// Должность
/// </summary>
[DisplayName("Должность")]
[Description("Занимаемая должность согласно штатного расписания")]
[Category("3. Дополнительно")]
[TypeConverter(typeof(PostTypeConverter))]
public string Post
{
  get { return _post; }
  set { _post = value; }
}
```

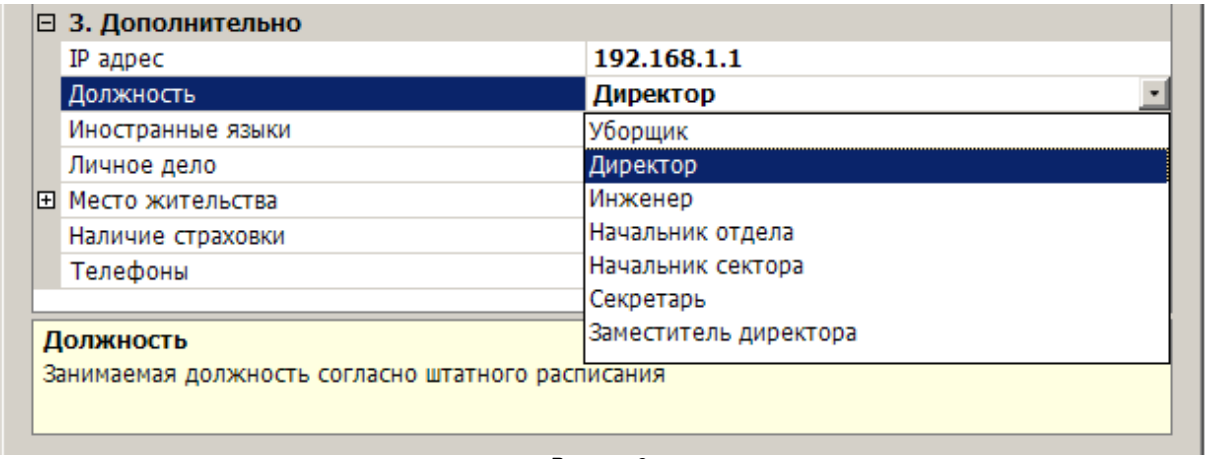

Рисунок 9.

Аналогично реализуется и выбор из списка фиксированных значений например double – см. класс ***PossibleValuesTypeConverter*** в тестовом проекте.

Как избавиться от ошибки "Конструктор для типа "System.String" не найден" при редактировании свойств типа StringCollection?

Свойства типа StringCollection

```
[DisplayName("Дети")]
[Description("Дети")]
[Category("2. Общие")]
[PropertyOrder(30)]
public StringCollection Children
{
   get { return _children; }
   set { _children = value; }
}
```

открываются для редактирования в PropertyGrid, но при попытке добавить строку к списку выдается сообщение об ошибке "***Конструктор для типа "System.String" не найден***" (***Constructor on type 'System.String' not found***):

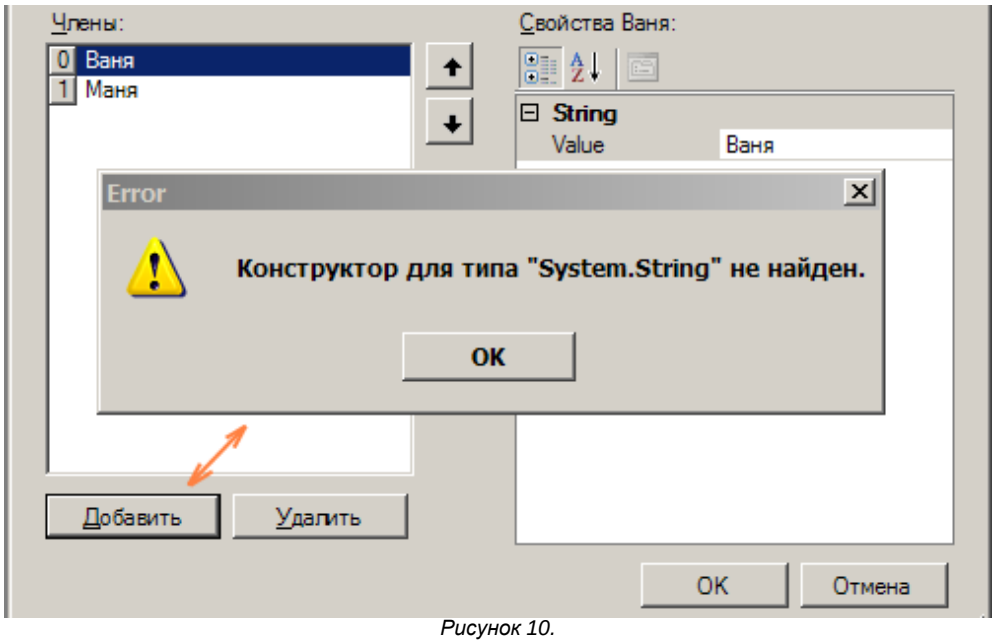

Рисунок 10.

Исправить положение можно добавив атрибут **Editor** со следующими параметрами:

```
[DisplayName("Дети")]
[Description("Дети")]
[Category("2. Общие")]
[PropertyOrder(30)]
[Editor(
  "System.Windows.Forms.Design.StringCollectionEditor, System.Design, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a",
  "System.Drawing.Design.UITypeEditor,System.Drawing, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
)]
public StringCollection Children
{
   get { return _children; }
   set { _children = value; }
}
```

Окно редактирования StringCollection примет при этом следующий вид:

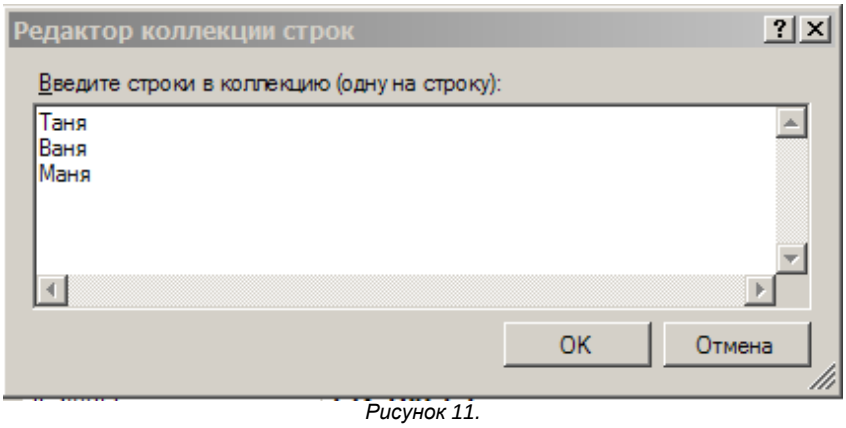

Рисунок 11.

Как реализовать отображение составного свойства?

Бывает так, что тип свойства является сложным объектом, также имеющим свойства. Хотелось бы иметь возможность редактировать свойства этого объекта в раскрывающемся списке. К классу, используемому в качестве типа составного свойства, необходимо применить атрибут ***TypeConverter*** с ***ExpandableObjectConverter*** в качестве параметра:

```
/// <summary>
/// Данные, входящие в адрес
/// </summary>
[TypeConverter(typeof(ExpandableObjectConverter))]
class AddressData
{
  /// <summary>
  /// Конструктор
  /// </summary>
  public AddressData(string town, string street, uint house)
  {
    _town = town;
    _street = street;
    _house = house;
  }

  private string _town;

  /// <summary>
  /// Город
  /// </summary>
  [DisplayName("Город")]
  [Description("Наименование населенного пункта")]
  public string Town
  {
    get { return _town; }
    set { _town = value; }
  }

  private string _street;

  /// <summary>
  /// Улица
  /// </summary>
  [DisplayName("Улица")]
  [Description("Название улицы")]
  public string Street
  {
    get { return _street; }
    set { _street = value; }
  }

  private uint _house;

  /// <summary>
  /// Номер дома
  /// </summary>
  [DisplayName("Дом")]
  [Description("Номер дома")]
  public uint House
  {
    get { return _house; }
```

```
    set { _house = value; }
  }

  /// <summary>
  /// Представление в виде строки
  /// </summary>
  public override string ToString()
  {
    return _town + ", " + _street + " - " + _house;
  }

}
```

Задавать дополнительные атрибуты для редактируемого свойства не нужно:

```
[DisplayName("Место жительства")]
[Description("Адрес")]
[Category("3. Дополнительно")]
public AddressData Address
{
   get { return _address; }
   set { _address = value; }
}
```

Рисунок 12.

Как организовать выбор файла с заданным расширением?

Задайте атрибут ***Editor***:

```
[DisplayName("Личное дело")]
[Description("Имя файла личного дела")]
[Category("3. Дополнительно")]
[Editor(typeof(DocFileEditor), typeof(UITypeEditor))]
public string PersonalFileName
{
  get { return _personalfilename; }
  set { _personalfilename = value; }
}
```

Собственно фильтр расширений задается в ***DocFileEditor***:

```
class DocFileEditor : FileNameEditor
{
  /// <summary>
  /// Настройка фильтра расширений
  /// </summary>
  protected override void InitializeDialog(OpenFileDialog ofd)
  {
    ofd.CheckFileExists = false;
    ofd.Filter = "Doc files (*.doc)|*.doc|All files (*.*)|*.*";
  }
}
```

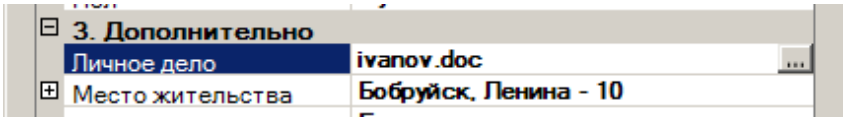

Рисунок 13.

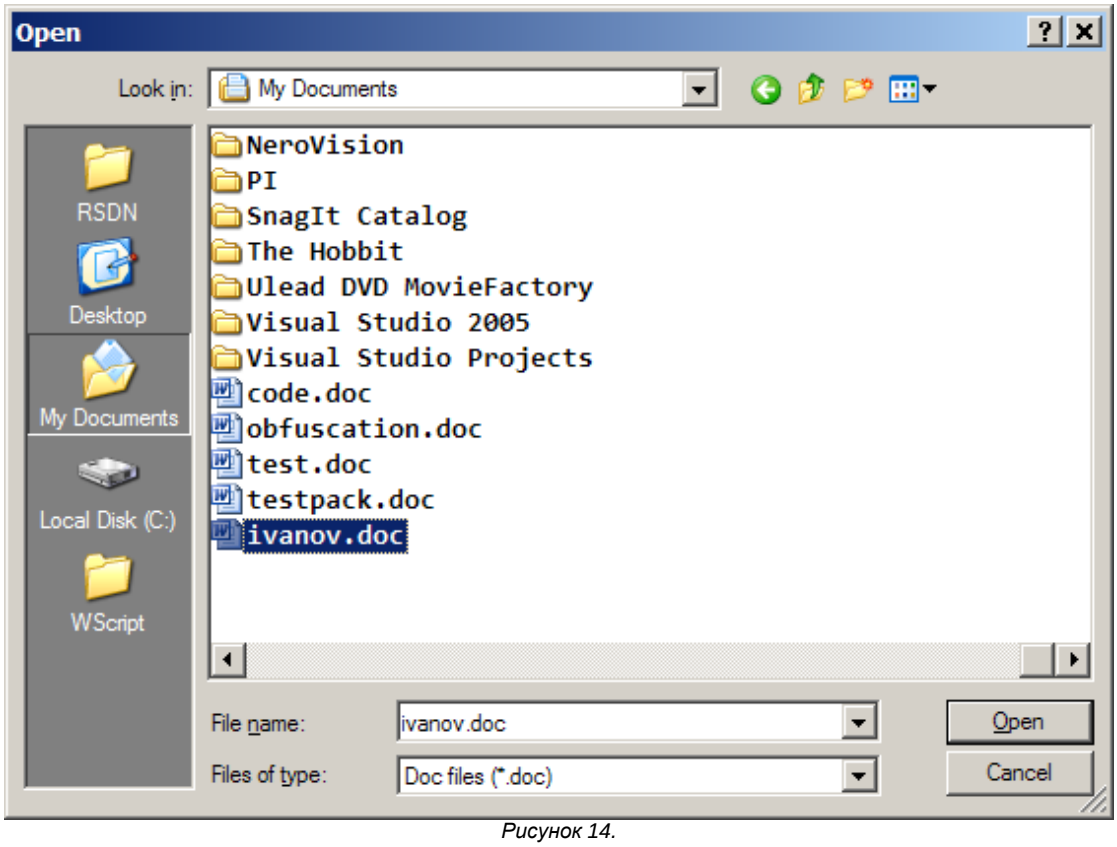

Рисунок 14.

Как организовать редактирование свойства в собственной форме?

Необходимо реализовать наследника ***UITypeEditor***, обеспечивающего вызов нужной формы (в данном случае ***IPAddressEditorForm***), передачу ей редактируемого значения и получение результата:

```
public class IPAddressEditor : UITypeEditor
{

  /// <summary>
  /// Реализация метода редактирования
  /// </summary>
  public override Object EditValue(
    ITypeDescriptorContext context,
    IServiceProvider provider,
    Object value)
  {
    if((context != null) && (provider != null))
    {
      IWindowsFormsEditorService svc =
        (IWindowsFormsEditorService)
        provider.GetService(typeof(IWindowsFormsEditorService));

      if(svc!= null)
      {
        using (IPAddressEditorForm ipfrm =
          new IPAddressEditorForm((IPAddress)value))
        {
          if (svc.ShowDialog(ipfrm) == DialogResult.OK)
          {
            value = ipfrm.IP;
          }
        }
      }
    }

    return base.EditValue(context, provider, value);
  }
```

```
  /// <summary>
  /// Возвращаем стиль редактора - модальное окно
  /// </summary>
  public override UITypeEditorEditStyle GetEditStyle(
    ITypeDescriptorContext context)
  {
    if (context != null)
      return UITypeEditorEditStyle.Modal;
    else
      return base.GetEditStyle(context);
  }

}
```

а затем привязать его к редактируемому свойству при помощи атрибута ***Editor***:

```
[DisplayName("IP адрес")]
[Description("IP адрес компьютера рабочего места")]
[Category("3. Дополнительно")]
[Editor(typeof(IPAddressEditor), typeof(UITypeEditor))]
public IPAddress IPaddress
{
  get { return _ipAddress; }
  set { _ipAddress = value; }
}
```

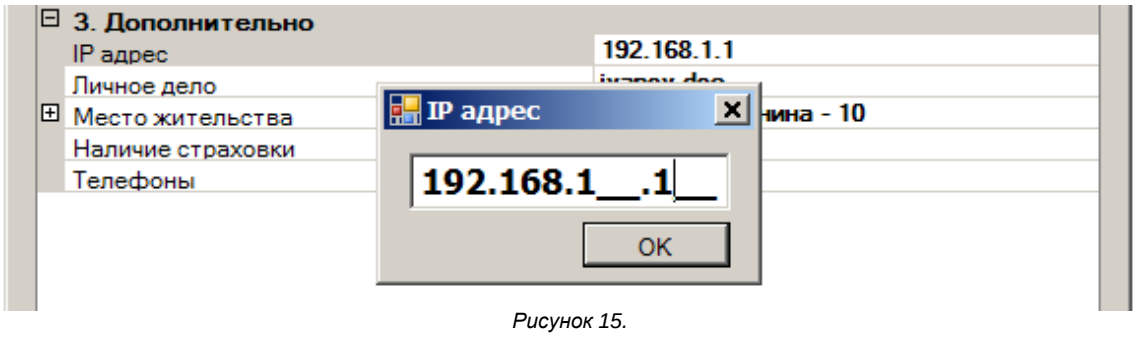

Рисунок 15.

Как организовать редактирование свойства в выпадающем списке?

Опять же, реализовать своего наследника ***UITypeEditor***, отображающего выпадающий список с нужным control-ом, передать ему исходное значение редактируемого свойства и принять результат по окончании редактирования (***ForeignLangsControl*** – составной UserControl с элементами, необходимыми для редактирования свойства):

```
public class ForeignLangsDropDownEditor : UITypeEditor
{

  /// <summary>
  /// Реализация метода редактирования
  /// </summary>
  public override Object EditValue(
    ITypeDescriptorContext context,
    IServiceProvider provider,
    Object value)
  {
    if((context != null) && (provider != null))
    {
      IWindowsFormsEditorService svc =
        (IWindowsFormsEditorService)
        provider.GetService(typeof(IWindowsFormsEditorService));

      if(svc!= null)
      {
        ForeignLangsControl flctrl =
          new ForeignLangsControl((ForeignLangs)value);
        flctrl.Tag = svc;

        svc.DropDownControl(flctrl);
```

```
          value = flctrl.Foreignlangs;
        }
      }

      return base.EditValue(context, provider, value);
    }

    /// <summary>
    /// Возвращаем стиль редактора - выпадающее окно
    /// </summary>
    public override UITypeEditorEditStyle GetEditStyle(
      ITypeDescriptorContext context)
    {
      if (context != null)
        return UITypeEditorEditStyle.DropDown;
      else
        return base.GetEditStyle(context);
    }

  }
```

и, соответственно, привязать полученный редактор к редактируемому свойству:

```
[DisplayName("Иностанные языки")]
[Description("Какими иностранными языками владеет")]
[Category("3. Дополнительно")]
[Editor(typeof(ForeignLangsDropDownEditor), typeof(UITypeEditor))]
public ForeignLangs Foreignlangs
{
  get { return _fl; }
  set { _fl = value; }
}
```

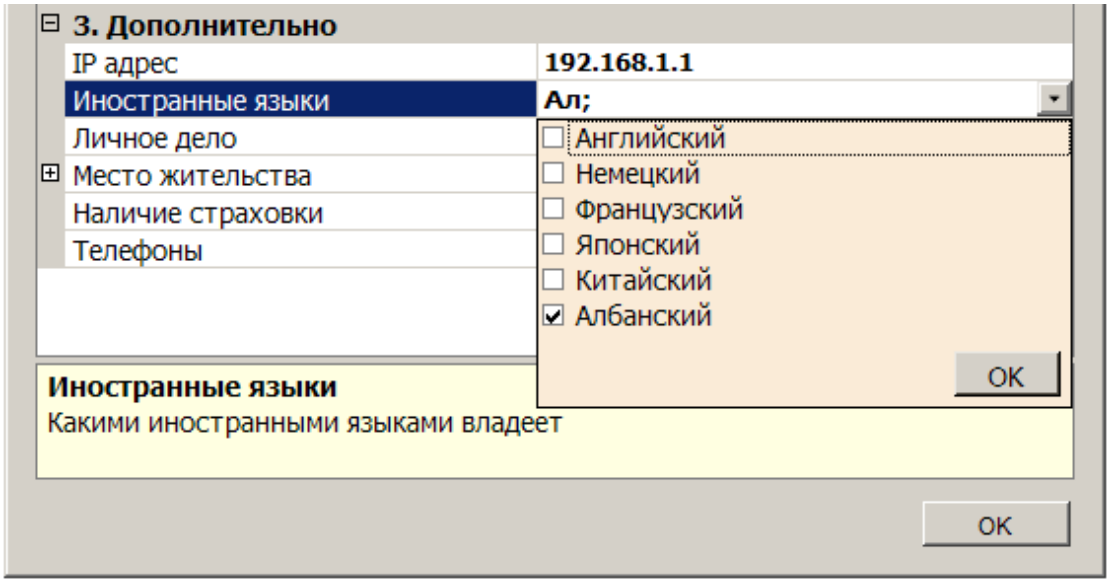

Рисунок 16.

Как скрыть пароль при редактировании?

Используйте атрибут ***PasswordPropertyText***:

```
      [DisplayName("Пароль")]
      [Description("Пароль для доступа на сервер компании")]
      [Category("3. Дополнительно")]
      [PropertyOrder(80)]
      [PasswordPropertyText(true)]
      public string Password
      {
         get { return _password; }
         set { _password = value; }
      }
```

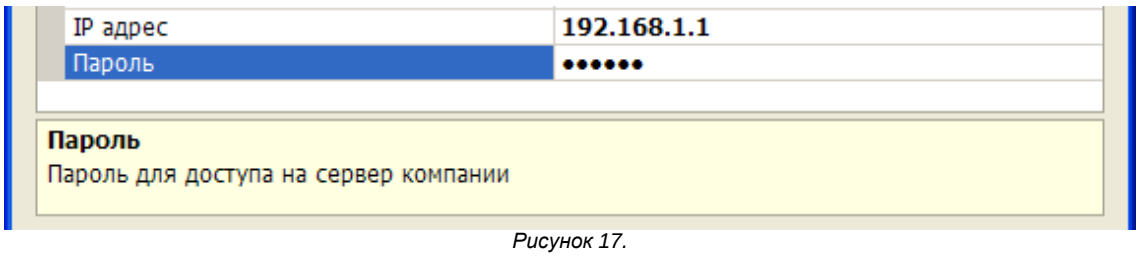

Рисунок 17.

Как управлять видимостью свойства в зависимости от значения другого свойства?

Стандартный атрибут ***Browsable*** позволяет задавать видимость свойства в PropertyGrid только на этапе написания кода. Чтобы управлять видимостью свойства в зависимости от значения другого свойства настраиваемого объекта, понадобятся новый атрибут – ***DynamicPropertyFilter*** и базовый класс – ***FilterablePropertyBase***:

DynamicPropertyFilter

Указываем базовый класс для класса настраиваемого объекта:

```
/// <summary>
/// Данные для редактирования в PropertyGrid
/// </summary>
class PersonData : FilterablePropertyBase
{
  public PersonData()
...
```

ПРИМЕЧАНИЕ

Решение не очень красивое, так как делает данные зависимыми от средств отображения и редактирования. – прим. ред.

А для свойства, видимость которого зависит от другого свойства – атрибут ***DynamicPropertyFilter***:

```
/// <summary>
/// Должность
/// </summary>
[DisplayName("Должность")]
[Description("Занимаемая должность согласно штатному расписанию")]
[Category("3. Дополнительно")]
[TypeConverter(typeof(PostTypeConverter))]
public string Post
{
  get { return _post; }
  set { _post = value; }
}

/// <summary>
/// Имя файла личного дела
/// </summary>
[DisplayName("Личное дело")]
[Description("Имя файла личного дела")]
[Category("3. Дополнительно")]
[Editor(typeof(DocFileEditor), typeof(UITypeEditor))]
[DynamicPropertyFilter("Post", "Уборщик, Инженер, Начальник отдела, Начальник сектора, Секретарь")]
public string PersonalFileName
{
  get { return _personalfilename; }
```

```
  set { _personalfilename = value; }
}
```

Также нужно добавить обработчик события PropertyGrid – ***PropertyValueChanged***:

```
private void propertyGrid1_PropertyValueChanged(
  object s, PropertyValueChangedEventArgs e)
{
  propertyGrid1.Refresh();
}
```

В данном случае свойство ***PersonalFileName*** (*Имя файла личного дела*) будет показано в PropertyGrid только тогда, когда свойство ***Post*** (**Должность**) будет иметь любое из указанных в атрибуте значений:

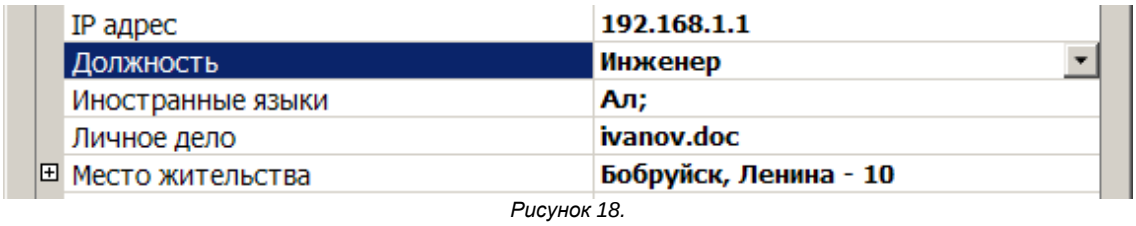

Рисунок 18.

Если переключить свойство ***Должность*** в значение, отсутствующее в параметре атрибута ***DynamicPropertyFilter***, свойство ***Личное дело*** исчезнет из списка отображаемых свойств:

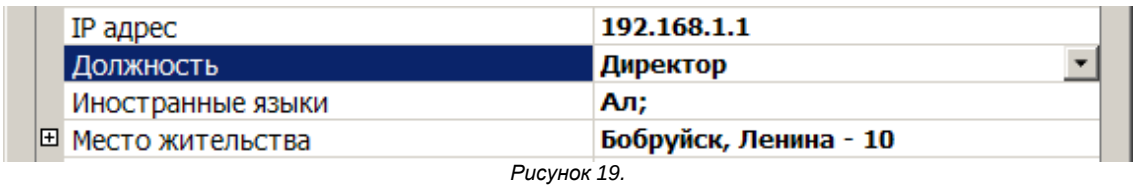

Рисунок 19.

Аналогично, вторым параметром атрибута ***DynamicPropertyFilter*** можно передавать значения параметров других типов, например, перечисления:

```
[DynamicPropertyFilter("Sex", "Woman,Unknown")]
```

или bool:

```
[DynamicPropertyFilter("Insurance", "True")]
```

Как избавиться от стандартного "(Collection)" в правой колонке для свойств-коллекций?

Используйте атрибут ***TypeConverter:***

```
[DisplayName("Телефоны")]
[Description("Список номеров телефонов")]
[Category("3. Дополнительно")]
[TypeConverter(typeof(CollectionTypeConverter))]
public List<PhoneNumber> Phones
{
  get { return _phones; }
  set { _phones = value; }
}
```

Реализация ***CollectionTypeConverter*** проста:

```
class CollectionTypeConverter : TypeConverter
{
  /// <summary>
  /// Только в строку
  /// </summary>
  public override bool CanConvertTo(
    ITypeDescriptorContext context, Type destType)
  {
    return destType == typeof (string);
  }

  /// <summary>
  /// И только так
  /// </summary>
  public override object ConvertTo(
    ITypeDescriptorContext context, CultureInfo culture,
    object value, Type destType)
  {
    return "< Список... >";
  }
}
```

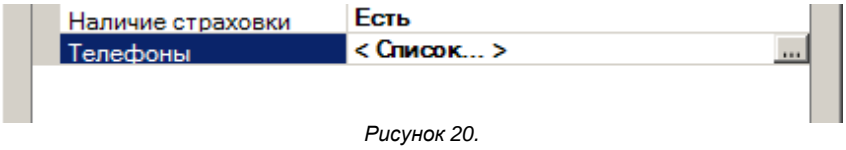

Рисунок 20.

При переходе к редактированию коллекции отобразится стандартное окно Collection Editor с данными редактируемой коллекции:

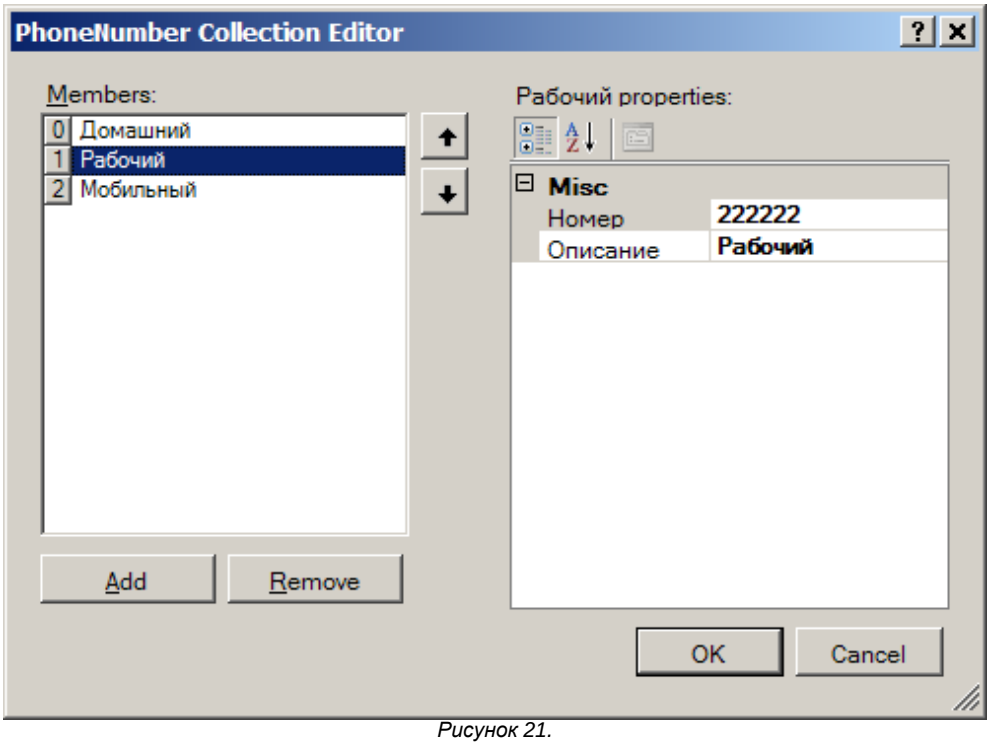

Рисунок 21.

Как заменить стандартные подписи (Members, properties) в окне Collection Editor?

"Members", "properties", "Add" и "Remove" можно заменить русскими аналогами (при наличии установленного .NET Framework 2.0 Russian Language Pack) переключением ***CurrentUICulture***, как это сделано в методе Main() тестовой программы:

```
Thread.CurrentThread.CurrentUICulture = new CultureInfo("ru-RU", false);
```

Однако, если надпись "Члены" над списком телефонов вас тоже не устраивает, можно применить и более

радикальный способ. Заодно решим и следующую проблему.

Как добавить в Collection Editor окно с расширенной подсказкой по редактируемым свойствам?

Добавим к свойству-коллекции атрибут ***Editor*** со специализированной версией CollectionEditor:

```
[DisplayName("Телефоны")]
[Description("Список номеров телефонов")]
[Category("3. Дополнительно")]
[TypeConverter(typeof(CollectionTypeConverter))]
[Editor(typeof(PhoneNumbersCollectionEditor), typeof(UITypeEditor))]
public List<PhoneNumber> Phones
{
  get { return _phones; }
  set { _phones = value; }
}
```

Новая реализация CollectionEditor, ***PhoneNumbersCollectionEditor***, умеет запоминать положение и размеры своего окна, меняет стандартные подписи на соответствующие редактируемым данным и добавляет окно с расширенной подсказкой по свойствам:

PhoneNumbersCollectionEditor

Результат что называется, налицо:

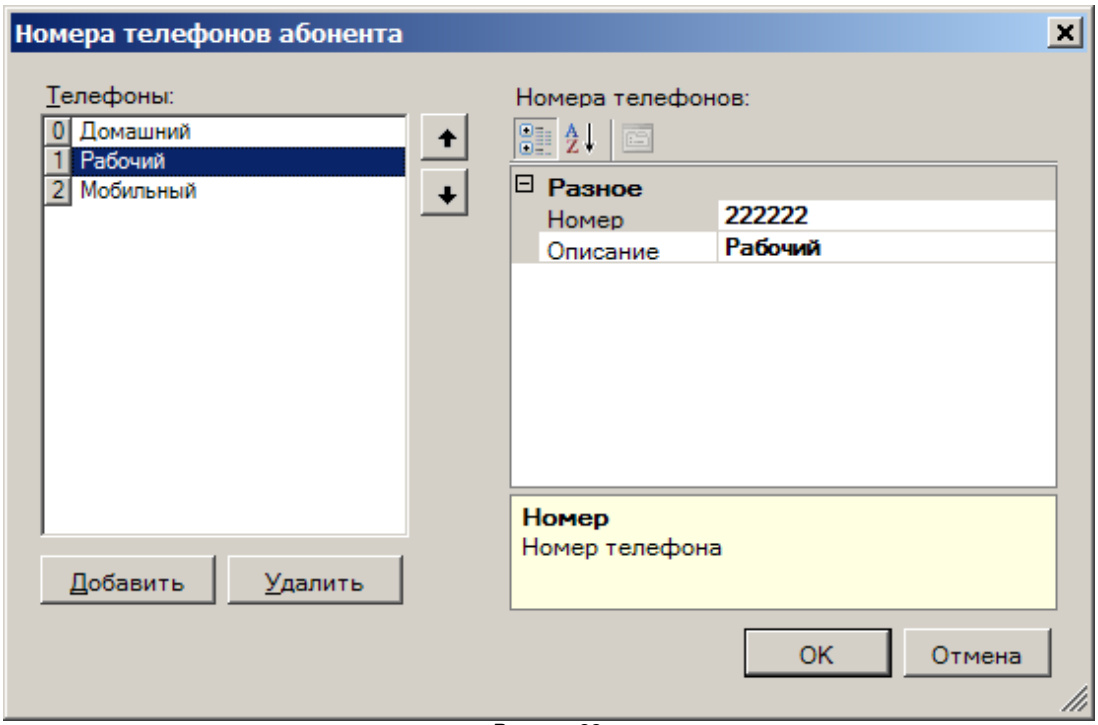

Рисунок 22.

Как задать отличный от алфавитного порядок следования свойств внутри категории?

Нужно реализовать новый атрибут для задания порядка сортировки – ***PropertyOrderAttribute,*** и наследника ExpandableObjectConverter (***PropertySorter***), возвращающего список свойств, упорядоченный согласно значениям, заданным для них в атрибуте ***PropertyOrder:***

PropertySorter

Задаем атрибут ***TypeConverter*** с параметром ***PropertySorter*** для всего класса с настраиваемыми свойствами:

```
/// <summary>
/// Данные для редактирования в PropertyGrid
/// </summary>
[TypeConverter(typeof(PropertySorter))]
class PersonData : FilterablePropertyBase
{
```

и указываем атрибут ***PropertyOrder*** для упорядочиваемых свойств:

```
[DisplayName("Место жительства")]
[Description("Адрес")]
[PropertyOrder(30)]
[Category("3. Дополнительно")]
public AddressData Address
...

[DisplayName("Наличие страховки")]
[Description("Наличие страховки")]
[PropertyOrder(40)]
[Category("3. Дополнительно")]
[TypeConverter(typeof(BooleanTypeConverter))]
public bool Insurance
...

[DisplayName("Телефоны")]
[Description("Список номеров телефонов")]
[PropertyOrder(50)]
[Category("3. Дополнительно")]
[TypeConverter(typeof(CollectionTypeConverter))]
[Editor(typeof(PhoneNumbersCollectionEditor), typeof(UITypeEditor))]
public List<PhoneNumber> Phones
...

[DisplayName("Личное дело")]
[Description("Имя файла личного дела")]
[Category("3. Дополнительно")]
[PropertyOrder(20)]
[Editor(typeof(DocFileEditor), typeof(UITypeEditor))]
[DynamicPropertyFilter("Post",
  "Уборщик, Инженер, Начальник отдела, Начальник сектора, Секретарь")]
public string PersonalFileName
...

[DisplayName("IP-адрес")]
[Description("IP-адрес компьютера рабочего места")]
[PropertyOrder(70)]
[Category("3. Дополнительно")]
[Editor(typeof(IPAddressEditor), typeof(UITypeEditor))]
public IPAddress IPaddress
...

[DisplayName("Иностранные языки")]
[Description("Какими иностранными языками владеет")]
[PropertyOrder(60)]
[Category("3. Дополнительно")]
[Editor(typeof(ForeignLangsDropDownEditor), typeof(UITypeEditor))]
public ForeignLangs Foreignlangs
...

[DisplayName("Должность")]
[Description("Занимаемая должность согласно штатного расписания")]
[Category("3. Дополнительно")]
[PropertyOrder(10)]
[TypeConverter(typeof(PostTypeConverter))]
public string Post
...
```

результат – свойства в заданном порядке:

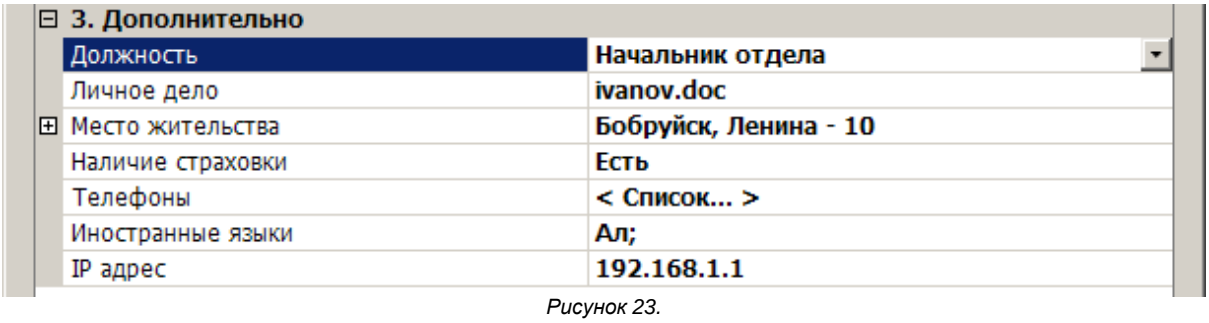

Рисунок 23.

Как запомнить и восстановить положение разделителя колонок в PropertyGrid?

Это можно сделать при помощи следующих функций:

```
/// <summary>
/// Сохранение положения разделителя в гриде
/// </summary>
private void SaveGridSplitterPos()
{
  Type type = propertyGrid1.GetType();
  FieldInfo field = type.GetField("gridView",
    BindingFlags.NonPublic | BindingFlags.Instance);

  object valGrid = field.GetValue(propertyGrid1);
  Type gridType = valGrid.GetType();
  Settings.Default.GridSplitterPos = (int)gridType.InvokeMember(
    "GetLabelWidth",
    BindingFlags.Public | BindingFlags.InvokeMethod | BindingFlags.Instance,
    null,
    valGrid, new object[] { });

  Trace.WriteLine("SaveGridSplitterPos(): "
    + Settings.Default.GridSplitterPos);
}

/// <summary>
/// Восстановление положения разделителя в гриде
/// </summary>
private void RestoreGridSplitterPos()
{
  try
  {
    Type type = propertyGrid1.GetType();
    FieldInfo field = type.GetField("gridView",
      BindingFlags.NonPublic | BindingFlags.Instance);

    object valGrid = field.GetValue(propertyGrid1);
    Type gridType = valGrid.GetType();
    gridType.InvokeMember("MoveSplitterTo",
      BindingFlags.NonPublic | BindingFlags.InvokeMethod
        | BindingFlags.Instance,
      null,
      valGrid, new object[] { Settings.Default.GridSplitterPos });

    Trace.WriteLine("RestoreGridSplitterPos(): "
      + Settings.Default.GridSplitterPos);
  }
  catch
  {
    Trace.WriteLine("MainForm::RestoreGridSplitterPos() exception");
  }
}
```

Вызываются они соответственно перед закрытием и перед загрузкой окна формы, содержащей PropertyGrid:

```
private void MainForm_Load(object sender, EventArgs e)
{
  Trace.WriteLine("MainForm_Load()");

  //устанавливаем редактируемый объект
  propertyGrid1.SelectedObject = _personData;

  // восстанавливаем положение окна
  RestorePos();

  // и разделителя колонок в гриде
  RestoreGridSplitterPos();
}

private void MainForm_FormClosing(object sender, FormClosingEventArgs e)
{
  Trace.WriteLine("MainForm_FormClosing()");

  // запоминаем положение окна
  SavePos();

  // и разделителя в гриде
  SaveGridSplitterPos();

  // сохраним возможно измененные значения параметров
  Settings.Default.Save();
}
```

p.s.

ПРЕДУПРЕЖДЕНИЕ

Все имена, фамилии и ip-адреса вымышлены. Любые совпадения случайны. В ходе экспериментов ни один Иван Иванович из г.Бобруйска не пострадал.

---

*Эта статья опубликована в журнале RSDN Magazine #3-2006. Информацию о журнале можно найти **здесь***