

Smart tag – базовое применение.

Прочитать на сайте

Intro

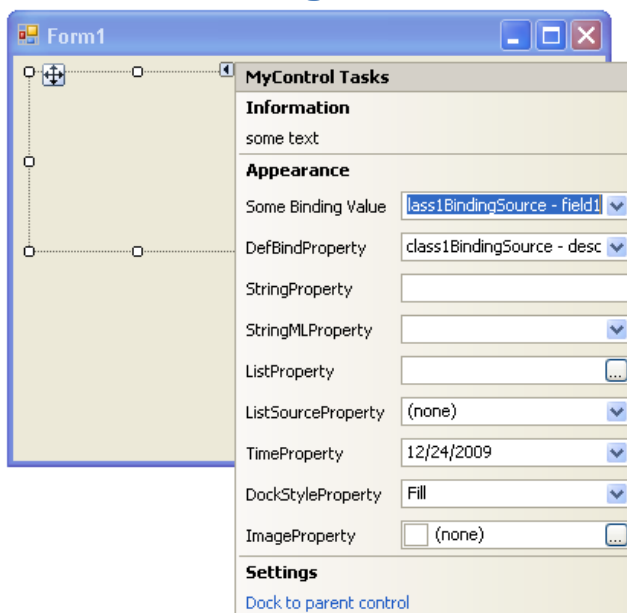
Долгое время я с удовольствием и радостью пользовался смарт тэгами. Вообще-то я долгое время даже не знал, как эта штука правильно называется, но это не мешало с пользой использовать смарт тэги. Некоторое время назад я начал разрабатывать собственные компоненты, но это не сильно сокращало время настройки пользовательского интерфейса. Очень много времени уходит на то, чтобы найти необходимое свойство в Property view, выставить нужное свойство. Это не проблема, если приложение небольшое и надо настроить только пару экранов, но превращается в головную боль, когда у тебя более 30 сложных экранов с самописными компонентами. Впрочем я решил найти, как же делаются смарт тэги (эти маленькие чудесные треугольнички) и настроить их для самописных компонентов и некоторых стандартных.

Результаты поиска увенчались успехом и, в результате некоторых тестов и проб пера, получилось то, что я хотел. И это действительно стало экономить кучу времени!

В серии статей про смарт тэги я бы хотел поделиться полученными знаниями и рассказать все с самого начала.

Для всего нижеописанного советую поставить ReSharper 5. В скобках горячие клавиши приведены для ReSharper.

Что такое smart tag?



Смарт тэги элементы пользовательского интерфейса похожие на быстрое меню с наиболее популярными общими действиями. Смарт тэги доступны в режиме дизайна форм. Большинство стандартных компонентов поставляемых с .Net Framework содержат смарт тэги или списки быстрых действий.

Смарт тэги состоят из трех основных вещей:

- Действия – выглядят как ссылки и по нажатию совершается какое-то заранее определенное действие по форматированию компонента.
- Поля редактирования – может быть много различных типов: редактор текста, картинок, дат, биндинг.
- Текст – просто текст, который можно использовать для вывода справочной информации. Не редактируемое поле.

Делая изменения в смарт тэге, вы автоматом меняете соответствующее свойство в компоненте.

Примеры применения и соображения как это можно использовать

Когда вы создаете свой собственный компонент, вы хотите использовать его с максимальной возможной простотой в настройке, аковыряние в окне свойств к этому явно не отнести. К примеру, я создал шаблон для шапки моего приложения и хочу, чтобы он прикреплялся (dock) к верху формы. Мне надо переключиться на экран свойств, найти свойство Dock и выбрать из выпадающего списка Top. Это долго даже если по полной использовать горячие клавиши. Но если сделать смарт тэг, все пройдет в разы быстрее! Мышка уже в районе компонента и очень легко добраться до смарт тэга и нажать на ссылку.

Смарт тэги есть у ComboBox, PictureBox, Panel и у многих других компонентов. Но, к сожалению, для Button или Label и еще некоторых общих компонентов нет смарт тэгов. На мой взгляд это было бы полезно создать смарт тэги для этих компонентов и задавать имена и подписи не переключаясь в окно свойств. А, как идея, хороша? ;)

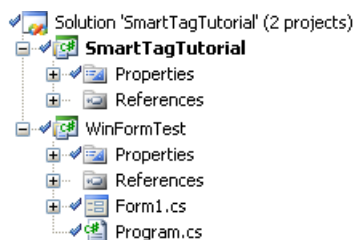
Ко всему прочему, в моем стиле программирования UI широко применение биндинга и я хочу сделать эту операцию быстрой. К примеру, забиндить Action<> к свойству Tag или к какому-нибудь новому свойству.

Создание своего компонента

Есть два пути создания своего компонента. Первый – создать новый класс и наследовать его от Component; другой путь создать класс наследник UserControl. Легче всего это создается с помощью контекстного меню студии Add.

Когда мне нужен новый компонент как композиция уже существующих, я использую UserControl. Это наиболее общий способ создания новых компонентов в процессе написания прикладного ПО. Но если вы желаете подредактировать существующие компоненты, я советую использовать Component.

Создаем новый проект типа DLL Library и называем его в духе “SmartTagTutorial”. После этого должен появиться солюшен (не знаю как это адекватно перевести) с одним проектом. Для задач тестирования неплохо было бы создать и проект типа WinForms.



После всех манипуляций должно появиться что-то похожее.

Знакомо все пока что выглядит, а? ;)

Следующим шагом добавим новый UserControl. Вызываем контекстное меню проекта SmartTagTutorial и выбираем нужный пункт (Add > User Control). Далее может быть самая сложная часть – придумать, что же будет делать новый компонент, и что мы хотим в нем оптимизировать. Я остановился на создании шапки.

Итак, теперь у нас есть класс наследник UserControl. Настало время проявить свою фантазию и нарисовать, как будет выглядеть шапка. У меня получилось как на рисунке ниже.



Есть основной текст, дополнительный текст с описанием и заготовка для картинки. Думаю, будет достаточно. Наступило время писать код, начнем от простого к сложному.

Создание ссылок действий.

Действия, в контексте смарт тэгов, это методы без параметров. Визуально они представлены ссылками, и действие выполняется сразу при нажатии. В нашем случае будет логично сделать так, чтобы при нажатии наш компонент прикреплялся к верху родительского контейнера. Все ведь согласны с тем, что каждый раз выискивать Dock и выставлять нужное значение скучно?

Переключаемся в Solution Explorer (ctrl+alt+L), выбираем SmartTagTutorial, создаем новый класс (alt+ins) и называем его MyControlDesigner. Для того чтобы все заработало, наследуем его от класса ControlDesigner.

```
public class MyControlDesigner : ControlDesigner { ... }
```

Теперь необходимо переопределить свойство Verbs. Нажимаем в редакторе кода alt+ins, выбираем Override members и выделяем Verbs. Получаем заготовку перегрузки, где надо написать тело метода.

```
public override DesignerVerbCollection Verbs {  
    get { throw new NotImplementedException (); }  
}
```

Все «действия» являются имплементацией DesignerVerb и содержатся в DesignerVerbCollection. Предлагаю создать новый метод, который будет заполнять коллекцию .

```
private void SetVerb() {  
    verbs.Clear();  
    if (base.Control.Dock != DockStyle.None) {  
        verbs.Add(new DesignerVerb("Undock parent top ", OnVerb));  
    }  
    else {  
        verbs.Add(new DesignerVerb("Dock parent top ", OnVerb));  
    }  
}
```

Переменная verbs определена в теле класса как экземпляр DesignerVerbCollection. В приведенном куске кода видно как создаются «действия». Первый параметр это текст который будут видеть

конечные пользователи компонента; второй – EventHandler который среагирует на клик. Посмотрим поближе на метод OnVerb.

```
protected void OnVerb(object sender, EventArgs e) {
    (base.Control).Dock = (base.Control).Dock == DockStyle.None ?
    DockStyle.Top : DockStyle.None;
}
```

В нашем случае он достаточно прост. Но в целом тут можно писать все что душе угодно связанное с подконтрольным компонентом. Для этого есть ссылка base.Control.

Не следует, однако, увлекаться созданием «действий», потому что у смарт тэгов нет механизма скроллинга и все что выйдет за пределы экрана абсолютно недоступно для дальнейшего использования.

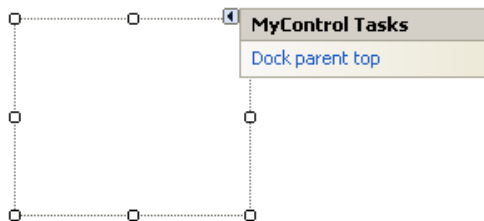
Когда у нас появился класс дизайнер, наступило время показать как он присоединяется к нужному компоненту. Открываем код для нашего компонента и добавляем следующий код:

```
[Designer(typeof(MyControlDesigner))]
```

Должно получиться примерно вот так:

```
[Designer(typeof(MyControlDesigner))]  
public partial class MyControl : UserControl { ... }
```

Пересобираем приложение и проверяем что получилось. Открываем форму (Form1) из проекта WinForms, выбираем окно инструментов (ctrl+alt+x) и перетаскиваем на компонент на форму. Выделяем компонент и видим знакомый треугольник.



Создание полнофункционального смарт тэга

О да, это гораздо интереснее и мощнее. Я думаю, что если правильно настроить смарт тэг, то эффективность работы с компонентом вырастет в разы! Особенно на ранних стадиях разработки. Я хочу чтобы компоненты настраивались по принципу «здесь и сейчас», без всего этого скроллинга по окну свойств. Бьюсь об заклад, что переименовать компонент, задать осмысленную подпись и настроить биндинг хочется сразу после создания компонента. Ну так мы это и сделаем!

Перво-наперво, создаем еще один класс и называем его MyControlDesignerEx (Ex значит Extended). Этот класс должен быть пронаследован от ParentControlDesigner. Для наших нужд будет достаточно переопределить свойство ActionList.

```
public override DesignerActionListCollection ActionLists { get { ... } }
```

В этом свойстве будет создан экземпляр класса в котором будет уже настоящая работа. И в качестве базового класса у него будет ControlActionListBase.

```
public class MyControlActionList : ControlActionListBase { ... }
```

На данный момент можно уже написать реализацию свойства ActionLists. План работ такой: создать экземпляр DesignerActionListCollection для возврата; создать DesignerActionList, в котором будет храниться только что созданная коллекция.

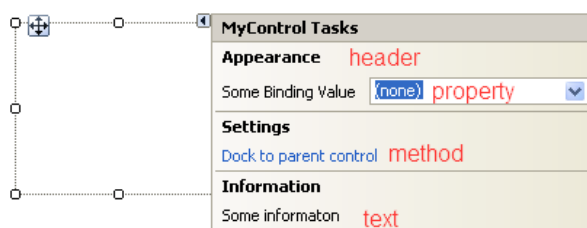
DesignerActionList принимает в конструктор 2 параметра: сам компонент и его дизайнер. У нас есть все необходимые ингредиенты и можно продолжать. Т.к. у нас дизайнер является наследником ParentControlDesigner, то возможно использование некоторых полезных родительских свойств, например такого как Control. Это свойство представляет собой ссылку на экземпляр компонента к которому применен дизайнер. Ладно, на практике все гораздо проще!

```
public override DesignerActionListCollection ActionLists {
    get {
        if (actionLists == null) {
            actionLists = new DesignerActionListCollection();
            actionLists.Add(new MyControlActionList((MyControl)
Control, this));
        }
        return actionLists;
    }
}
```

Я решил создать actionLists за пределами свойства, чтобы ускорить дальнейшее использование. В большинстве случаев этого кода будет достаточно, потому что смарт тэги как правило, не меняются в ответ на пользовательские действия. В этом куске кода инициализируется класс коллекции и создается список того, что увидит пользователь. Можно сказать, что основное действие будет запрограммировано в классе MyControlActionList.

В классе MyControlActionList необходимо переопределить метод **GetSortedActionItems**. Сделав это мы определим внешний вид и функционал смарт тэга. В этом методе могут быть объявлены свойства, заголовки, действия которые будут доступны через смарт тэг.

Типы элементов в смарт тэге.



На рисунке представлены все основные части.

Header – просто заголовок. Он просто представляет логическую категорию. Нет никаких специальных свойств, просто показывает текст жирным шрифтом и является якорем для группировки элементов. Добавить его очень просто:

```
items.Add(new DesignerActionHeaderItem("Information"));
items.Add(new DesignerActionHeaderItem("Appearance"));
```

где items экземпляр класса DesignerActionItemCollection.

Следующий, простой как лом в разрезе, объект это **Text**. Можно показывать статичный текст с разъяснениями, всякие копипасты и прочее. Создать его тоже легко и незатейливо.

```
items.Add(new DesignerActionTextItem("some text", "Information"));
```

Никаких дополнительных действий не надо. Дешево и сердито.

Method – выглядит и работает как «быстрые действия». Поскольку `MyControlActionList` класс предоставляет достаточно богатую функциональность, то и создание этого элемента будет посложнее.

```
items.Add(new DesignerActionMethodItem( this,
                                         "Dock",
                                         "Dock to parent control",
                                         "Settings",
                                         "Dock top",
                                         true));
```

- Вторым параметром идет имя метода который будет вызван на щелчек мыши;
- Третьим – название для пользователя;
- Четвертым – категория, в которой отобразить «действие»;
- Пятым – всплывающая подсказка;
- Последний параметр для того, чтобы указать показывать в окне Свойства или нет.

Теперь вы обязаны создать метод с таким же именем (с учетом регистра), которое вы написали во втором параметре. Метод ничего не принимает и не возвращает.

```
private void Dock() {
    linkedControl.Dock = linkedControl.Dock == DockStyle.None ?
    DockStyle.Top : DockStyle.None;
}
```

Если категория, указанная в четвертом параметре не существует, то метод будет показан после всех категорий.

Property – последний в списке, но не по значению. Я думаю это вообще главная побудительная причина для написания смарт тэгов. Для создания свойства надо написать следующий код:

```
items.Add(new DesignerActionPropertyItem(
    "Caption",
    "Header Main Caption"
    "Appearance",
    "Sets the support header text."));
```

- Первый параметр это точное имя свойства, которое обрабатывает значения, передаваемые в компонент смарт тэга. Свойство объявляется в текущем классе;
- Второй – текст для пользователя;
- Четвертый – в какую категорию отнести элемент;
- Последний – для всплывающей подсказки.

НЕ ПЫТАЙТЕСЬ проверить смарт тэг в действии до того, как вы реализуете все свойства и методы объявленные в описанных элементах. Студия просто упадет, если не будет свойства `Caption` или метода `Dock`.

В соответствии со всем вышесказанным, в классе `MyControlActionList` надо определить свойство `Caption`.

```
public string Caption {
    get { return
    (string)GetPropertyByName("MainText").GetValue(linkedControl); }
    set { GetPropertyByName("MainText").SetValue(linkedControl, value); }
```

```

}

internal PropertyDescriptor GetPropertyByName(string propName) {
    var prop = TypeDescriptor.GetProperties(LinkedControl)[propName];
    if (null == prop) {
        throw new ArgumentException("Matching property not found.", propName);
    }

    return prop;
}

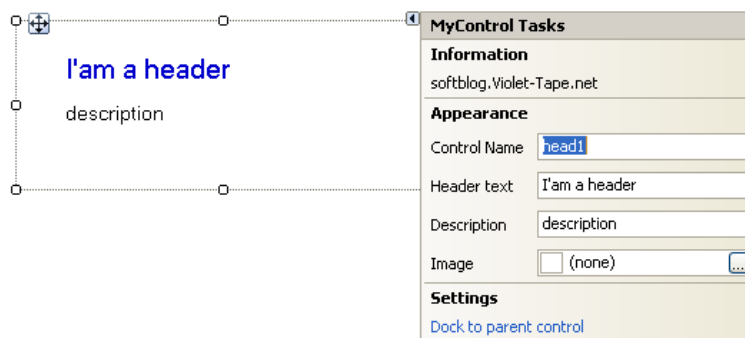
```

Да, более сложное задание свойств, но это потому что будет генерироваться код в бэграунде формы. Без этого, например, нельзя будет сменить имя созданного компонента.

Для большинства базовых типов которые используются для свойств, в студии есть базовые графические редакторы значений. Для строки это будет EditText, для изображений Image Set Dialog, ну и так далее. Но иногда требуется явно задать какой редактор использовать. Предположим, что мы хотим для Caption использовать редактор который позволяет вводить текст в несколько строк. Тогда надо к свойству Caption добавить атрибут:

```
[Editor(typeof (MultilineStringEditor), typeof (UITypeEditor))]
```

В итоге у меня получился такой вот смарт тэг:



Outro

Пересоберите солюшен, откройте форму и перенесите новый компонент для того, чтобы увидеть ваш смарт-тег. Попробуйте добавить новые свойства и действия. Поэкспериментируйте с типами свойств, например DateTime, DockStyle.

В следующей статье цикла я расскажу о том, как создавать методы для биндинга и как дебажить смарт тэги.

Hard'n'heavy!

