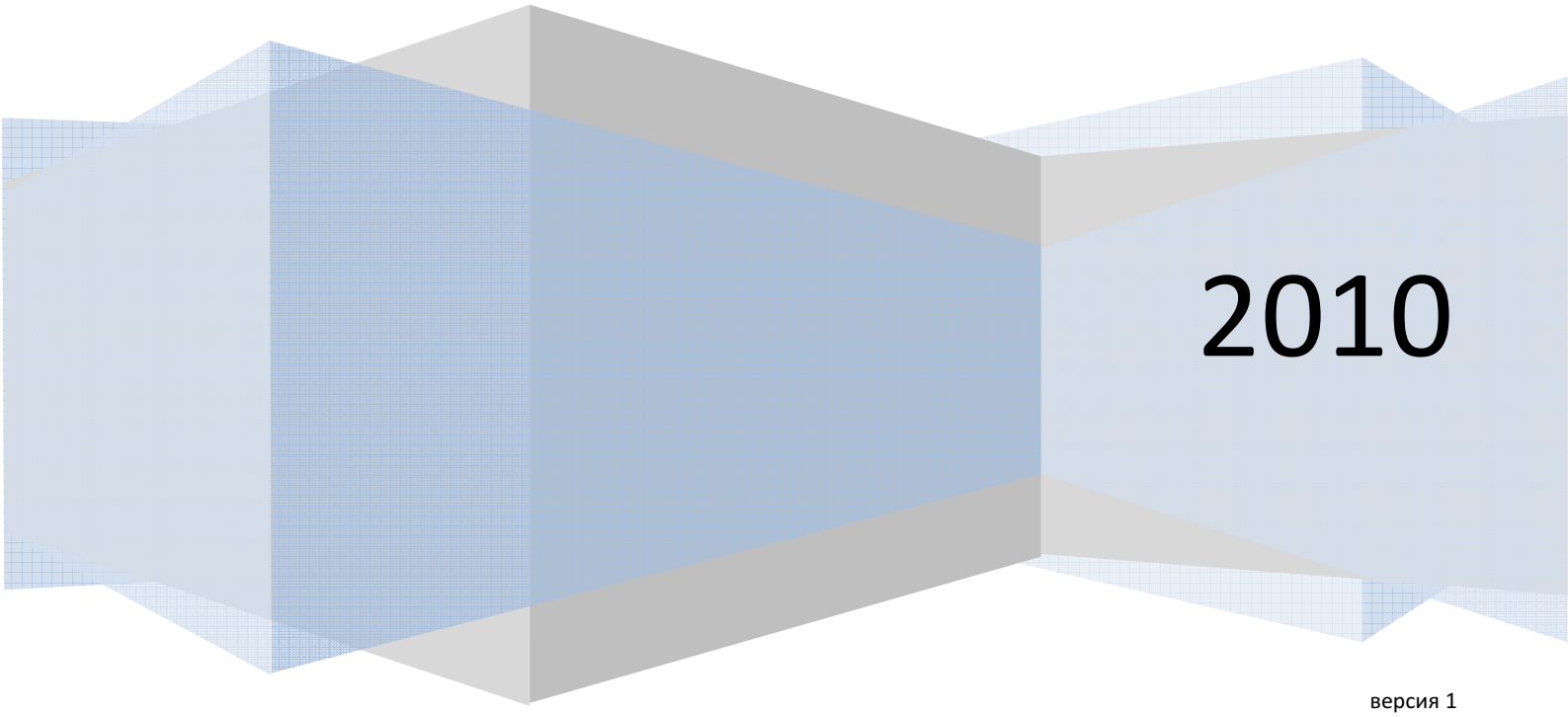
www.webdelphi.ru

Ribbon Controls в Delphi 2010

Визуальная разработка интерфейса приложения



2010

версия 1

Содержание

Введение

С момента выхода вместе с Microsoft Office 2007 Fluent UI (Ribbon) вызывает ажиотаж среди программистов и дизайнеров. На сегодняшний день существует огромное количество различных реализаций Ribbon, начиная от бесплатных и заканчивая профессионально выполненными платными. Новый пользовательский интерфейс, призванный заменить стандартные многоуровневые меню Windows все чаще и чаще встречается в приложениях. Использование Fluent UI способно в корне изменить внешний вид Вашего приложения.

Начиная с RAD Studio 2009, использование Ribbon Controls стало доступно и для программистов, использующих Delphi.

Однако прежде, чем мы приступим к изучению Ribbon Controls, следует отметить тот факт, что легальное использование этих компонентов возможно только после прохождения процедуры лицензирования в Microsoft. И именно с процедуры получения лицензии мы и начнем рассмотрение этого интересного вопроса – использование Ribbon Controls в Delphi.

Условные обозначения

Полужирный текст	Обозначение класса или компонента на форме
Полужирный зеленый текст	Свойство компонента
Полужирный синий текст	Значение свойства
Полужирный лиловый текст	Название окна IDE Delphi

Получение лицензии на использование Ribbon

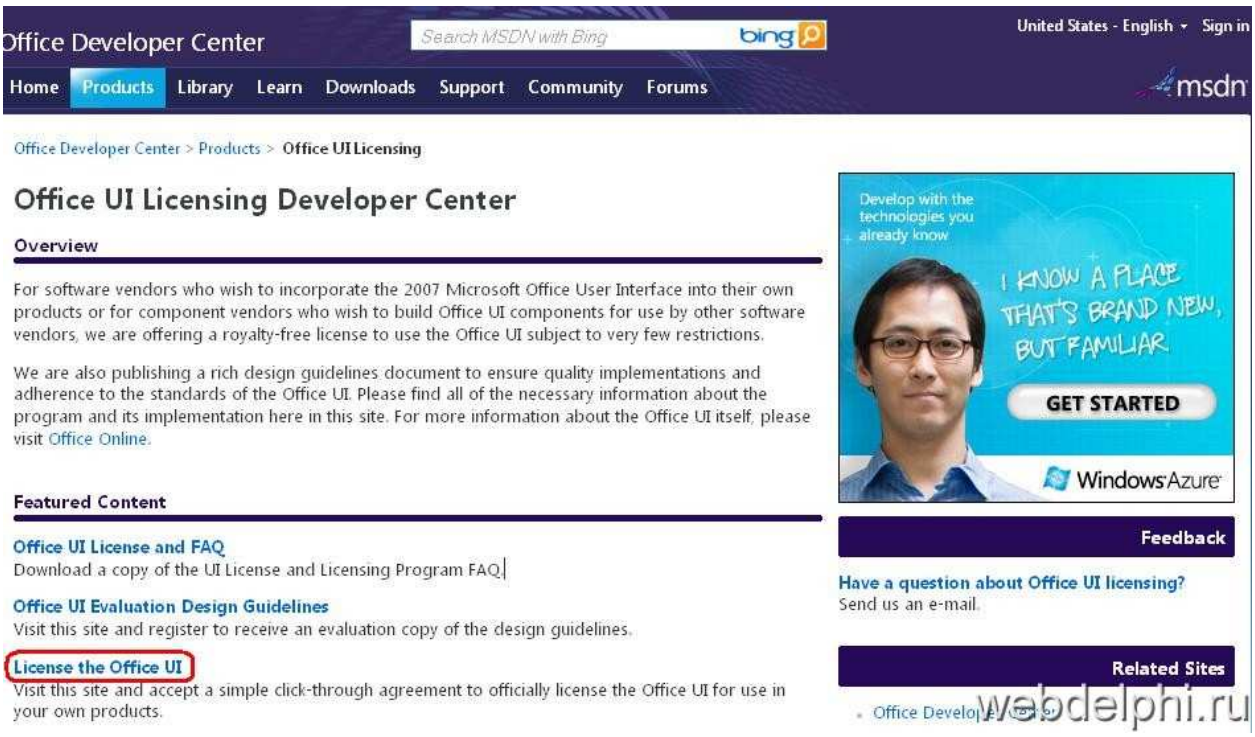
Во избежание нарушения первичной идеи дизайнеров, корпорация Microsoft ввела обязательное лицензирование использования Ribbon Controls в приложениях для всех разработчиков, получение лицензии бесплатно и проводится на сайте Microsoft по адресу:

http://msdn.microsoft.com/en-us/office/aa973809.aspx

*Начать процедуру лицензирования можно только после получения **Windows Live ID***

Рассмотрим весь процесс лицензирования по порядку.

1. Заходим на сайт Microsoft и переходим по ссылке «License the Office UI»



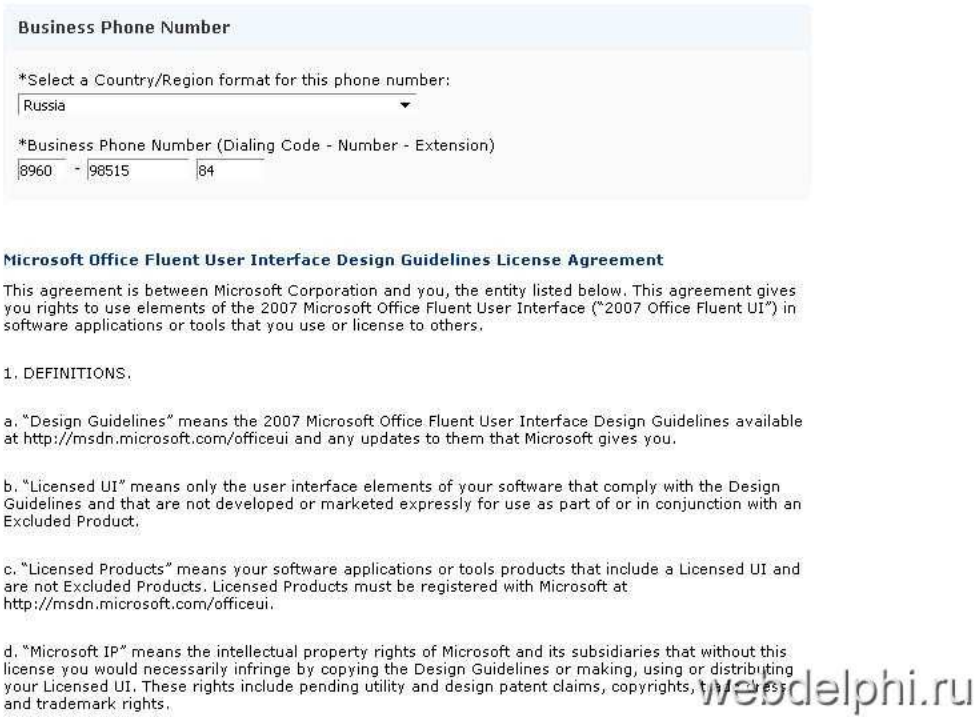
В открывшемся окне вводим регистрационные данные (Windows Live ID) и переходим к процессу лицензирования:

В первой форме необходимо обязательно заполнить поля:

- Имя (First Name)
- Фамилия (Last Name)

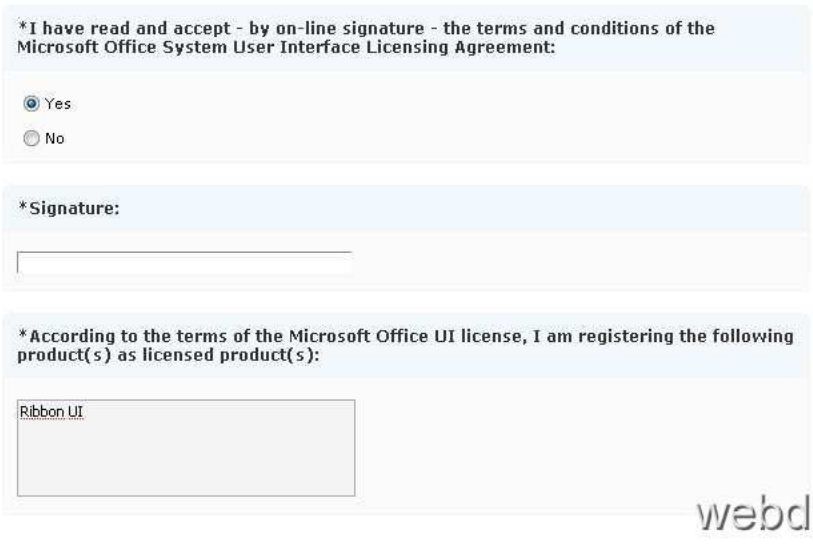
Обязательные к заполнению поля:

- Адрес электронной почты (E-mail Address)
- Название Вашей организации (Company/Organization Name)
- Страна (Country/Region)
- Адрес организации (Street Address)
- Город (City)
- Область/Штат (State)
- Почтовый индекс (Postal Code)

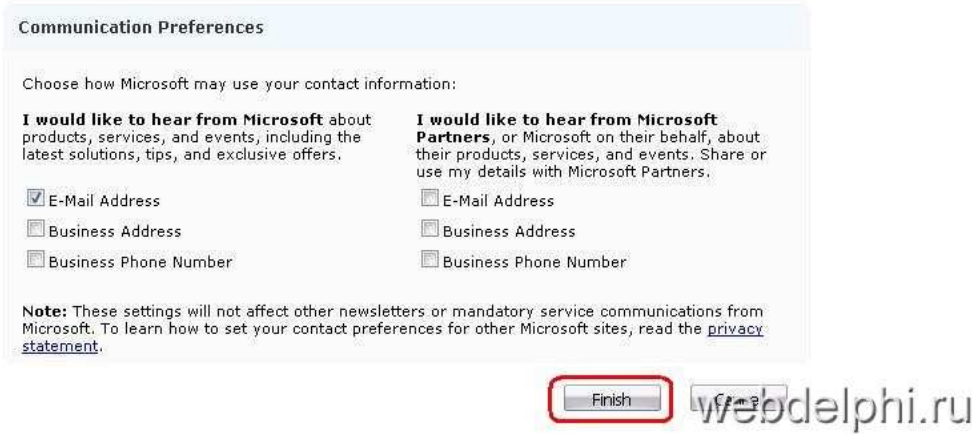


Теперь остается ввести номер телефона Вашей организации (Business Phone Number). Для этого выбираем сначала страну (Select a Country/Region format for this phone number) и вводим в поля формы телефонный номер согласно формату выбранной страны.

После того как все поля заполнены, нашему вниманию предоставляется текст лицензионного соглашения на английском языке.



Если Вы согласны с условиями лицензионного соглашения, то выбираете в следующей форме пункт «Yes», ставите свою подпись (фамилию и инициалы) в поле «Signature», а также указываете наименование того продукта на использование которого Вы получаете лицензию, т.е. Ribbon UI.



Далее Вы можете выбрать способ, которым Вы хотите получать новости от Microsoft или же сразу завершить процесс получения лицензии, нажав кнопку “Finish” после чего Вы попадёте на страницу для скачивания фалов Ribbon для WPF и полного текста лицензии на использование Fluent UI и правилами использования Ribbon.

На этом процесс получения регистрации можно считать законченным. Можно приступать к своим разработкам уже на вполне законных основаниях.

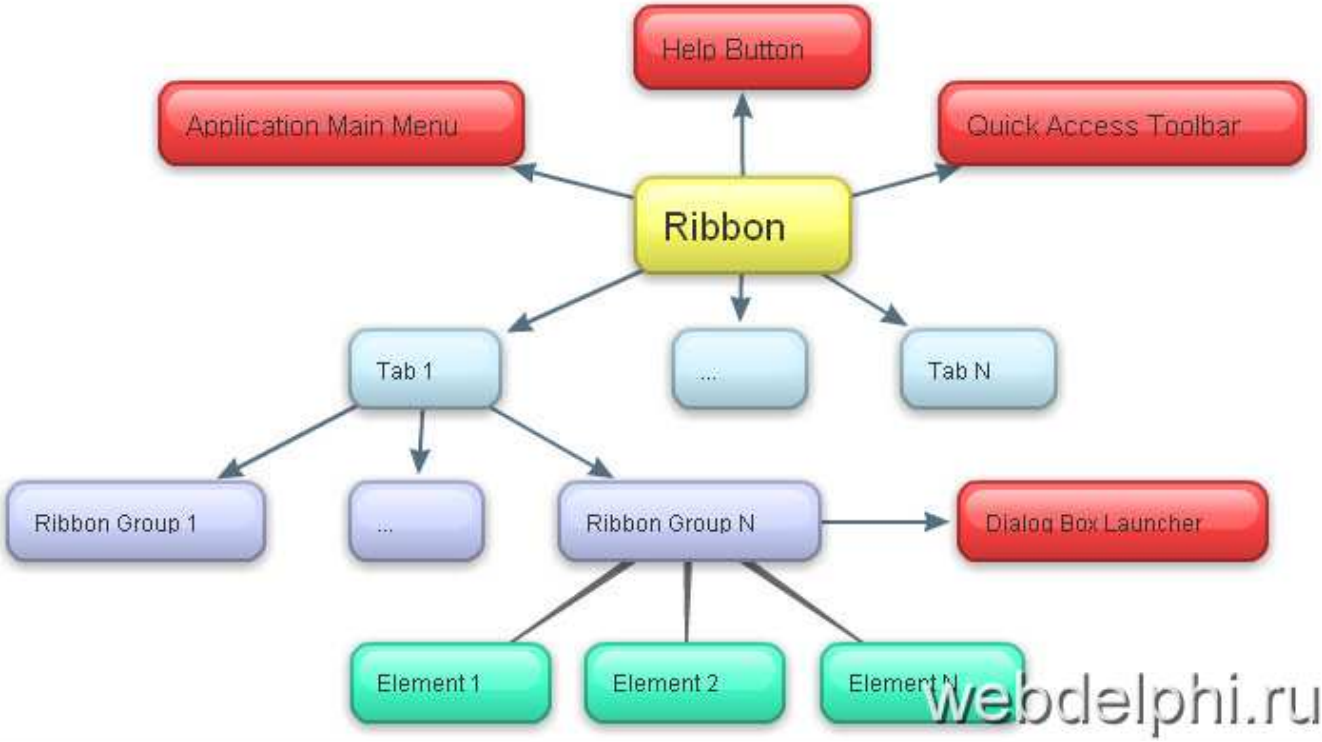
Устройство Ribbon

На рисунке показаны основные элементы интерфейса Ribbon, которые можно использовать в разработках с использованием Delphi 2009 – 2010.

В целом можно выделить группы элементов Ribbon:

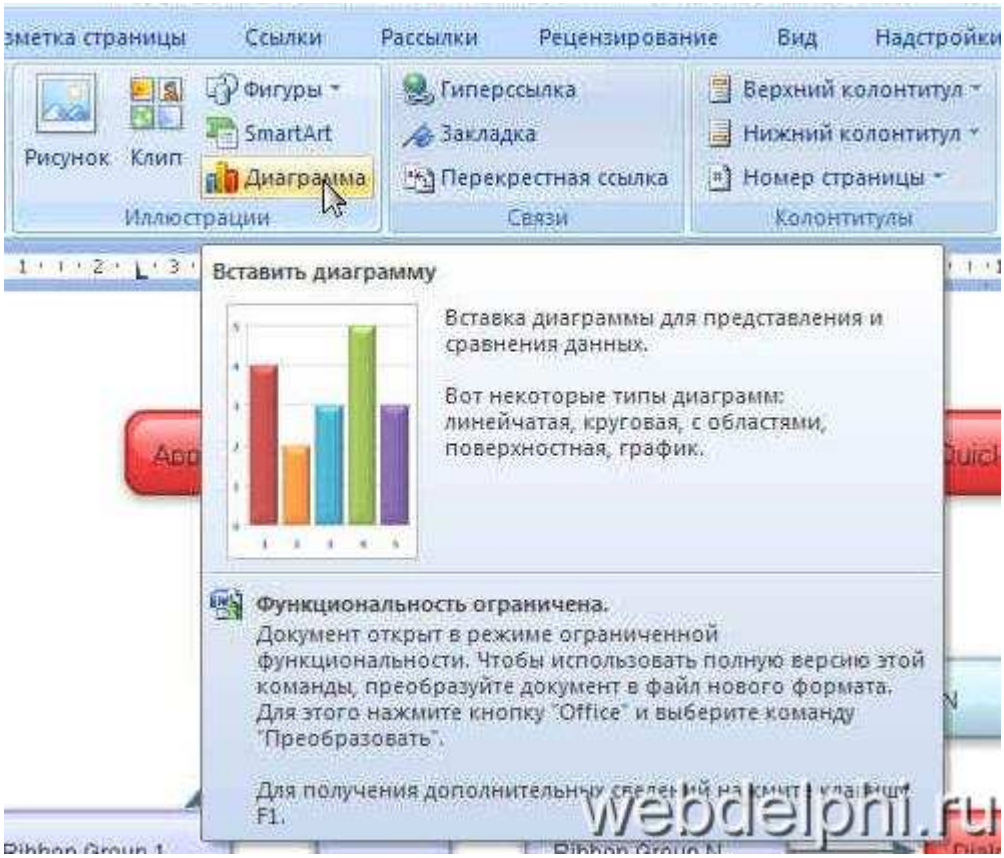
1. Вкладки
2. Группы
3. Кнопки, галереи и т.д.

Также Ribbon может содержать панель быстрого запуска и главное меню приложения. Если рассматривать иерархию элементов Ribbon. то она выглядит следующим образом (см. рисунок):



лента (Ribbon) может содержать неограниченное количество элементов вкладок (Tab), в свою очередь каждая вкладка может содержать неограниченное количество групп (Ribbon Group), группа может содержать неограниченное количество кнопок, галерей и т.д.(Element) и одну кнопку для запуска диалога (Dialog Box Launcher). Отдельными элементами интерфейса являются: панель быстрого запуска (Quick Access Toolbar), главное меню приложения (Application Main Menu) и кнопка помощи (Help Button).

Одним из достоинств нового интерфейса являются информативные и красивые всплывающие подсказки:



По мимо того, что подсказка несет основную информацию по элементу, она также может содержать растровое изображение сравнительно большого размера, а также дополнительную информацию, например, как на рисунке – информацию об ограниченной функциональности. Если рассматривать подсказку (Ribbon ScreenTip) как отдельный элемент интерфейса, то можно выделить следующие части:

1. Заголовок (Header)
2. Основной текст (Description)
3. Изображение (Bitmap)

4. Подпись (Footer)

Footer также может содержать свое изображение для большей наглядности.

Отдельное внимание стоит уделить командным элементам (тем, которые отвечают за выполнение каких-либо действий). Можно выделить следующие виды таких элементов:

1. Кнопки (Buttons)

2. Раздвоенные кнопки (Split-Buttons)

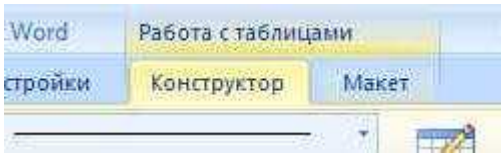
3. Галереи (Gallery)

4. Списки выбора (ComboBox)

5. Счётчики (Spin Edit)

В целом можно отметить, что имея в наличии только эти виды элементов можно создать интерфейс практически неограниченной функциональности и возможностей.

К сожалению, в настоящее время в Delphi реализованы не все функциональные возможности Ribbon. Так, например в Ribbon Delphi нет пока возможности реализовать контекстные вкладки такого вида:



Вполне вероятно, что эта и другие возможности Ribbon будут реализованы в более поздних версиях Delphi.

Ribbon Controls в Delphi.

В Delphi все компоненты для реализации интерфейса Ribbon располагаются на вкладке палитры компонентов “Ribbon Controls”

Всего в распоряжении разработчика имеется пять компонентов:

1. **Ribbon** (**TRibbon**) – основной элемент Ribbon;
2. **Ribbon ComboBox** (**TRibbonComboBox**)– список выбора для Ribbon;
3. **Ribbon SpinEdit** (**TRibbonSpinEdit**) – счётчик для Ribbon;
4. **ScreenTips Manager** (**TScreenTipsManager**) – менеджер подсказок Ribbon;
5. **ScreenTips Popup** (**TScreenTipsPopup**) – всплывающая подсказка Ribbon;

Следует отметить, что для работы с Ribbon Controls Вам необходимо также использовать в своих приложениях компоненты: **Action Manager**, **Action List** и другие, входящие в состав Delphi.

*Использование наряду с компонентами Ribbon Controls компонента **Action Manager** не является случайным или лишним. Дело в том, что этот компонент Delphi прекрасно подходит для реализации механизма команд (**Commands**), используемых в интерфейсах Ribbon. И разработчики Delphi не стали «городить огород» из нескольких близких по работе и структуре компонентов, а реализовали работу Ribbon Controls через уже знакомую систему взаимодействия с **Action Manager**.*

Также разработчики Ribbon Controls в Delphi практически не оставили программистам возможности изменить первоначальную идею дизайнеров Fluent UI, что является вполне обоснованным, т.к. одним из пунктов лицензионного соглашения является **сохранение первоначальной формы**. Так, при работе с Ribbon Controls следует учитывать следующие значения констант:

Константа	Значение	Описание
cRibbonHideWidth	300	Ширина родительского элемента при которой **Ribbon** должен автоматически скрываться
cRibbonHideHeight	250	Высота родительского элемента при которой **Ribbon** должен автоматически скрываться
cRibbonQuickAccessToolbarLeft	34	Левый отступ панели быстрого запуска
cRibbonHeight	117	Высота ленты
cRibbonQATHeight	26	Высота панели быстрого запуска
cRibbonUnthemedCaptionHeight	30	Высота заголовка ленты
cRibbonFirstTabOffSet	47	Отступ первой вкладки
cRibbonTabSpacing	6	Отступ между вкладками
cRibbonTabHeight	23	Высота вкладки

cRibbonMinimizedHeight	27	Высота вкладки в минимизированном состоянии
cRibbonTabScrollButtonWidth	12	Ширина скрола
cRibbonGroupCaptionHeight	16	Высота заголовка группы
cRibbonGroupHeight	86	Высота группы
cRibbonPageHeight	93	Высота вкладки
cRibbonMinimumCaptionWidth	50	Минимальная ширина заголовка ленты
UM_DISPLAYKEYTIPS	WM_USER + 1	Сообщение показа **KeyTips**
UM_CHECKSIZE	WM_USER + 2	Сообщение, посылаемое после **WMWindowPosChanged**

Программа-заготовка для работы с Ribbon Controls.

Прежде всего рассмотрим создание приложения-заготовки на примере которого и будем рассматривать все вопросы работы с Ribbon Controls. Создадим новый проект в Delphi 2010 и назовем его **RibbonApp**.

На главной форме разместим четыре компонента:

1.  **Ribbon** (вкладка **Ribbon Controls**)
2.  **ActionManager** (вкладка **Additional**)
3.  2 компонента **ImageList** (вкладка **Win32**)

Теперь настроим компоненты следующим образом.

У компонентов **ImageList**:

Имя свойства	***Значение***	
	ImageList 1	**ImageList 2**
ColorDepth	cd32Bit	
Name	il1	il2
Height	16	32
Width	16	32

У компонента **ActionManager:**

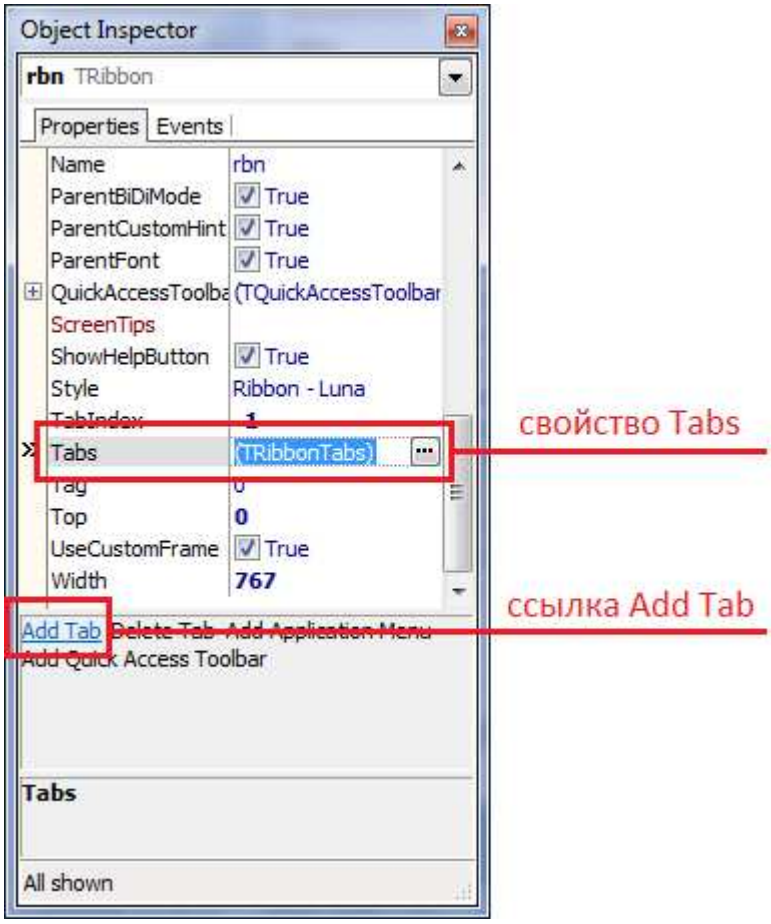
Имя свойства	***Значение***

Name	**actmgr**
Images	**il1**
LargeImages	**il2**

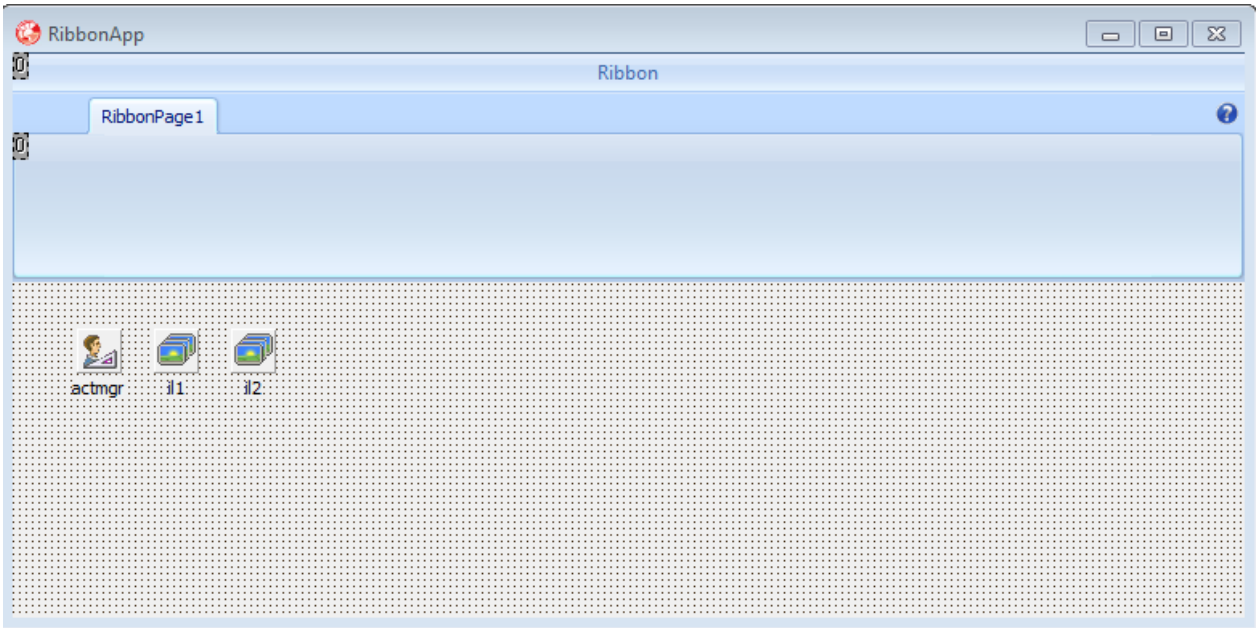
Свойства компонента **Ribbon**:

Имя свойства	***Значение***
Name	**rbn**
Caption	**Ribbon**
ActionManager	**actmgr**

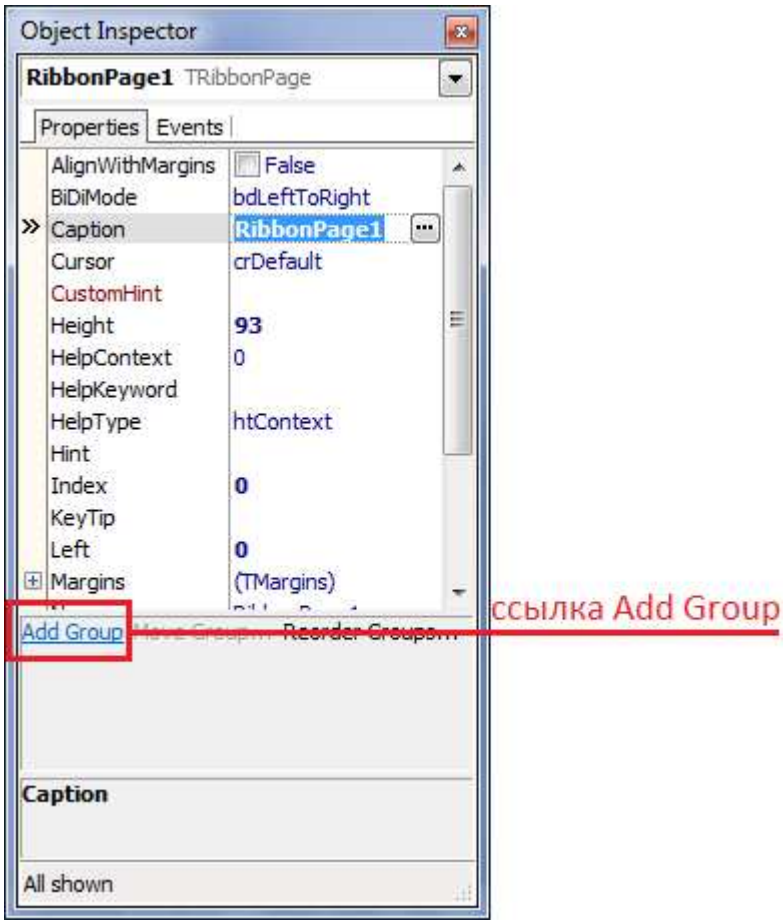
Теперь добавим на ленту **Ribbon** новую вкладку. Для этого необходимо нажать ссылку «*Add Tab*» в **Object Inspector** или воспользоваться свойством компонента **Ribbon** – **Tabs**:



На ленте нашего приложения должна появиться новая вкладка с названием ***RibbonPage1***:

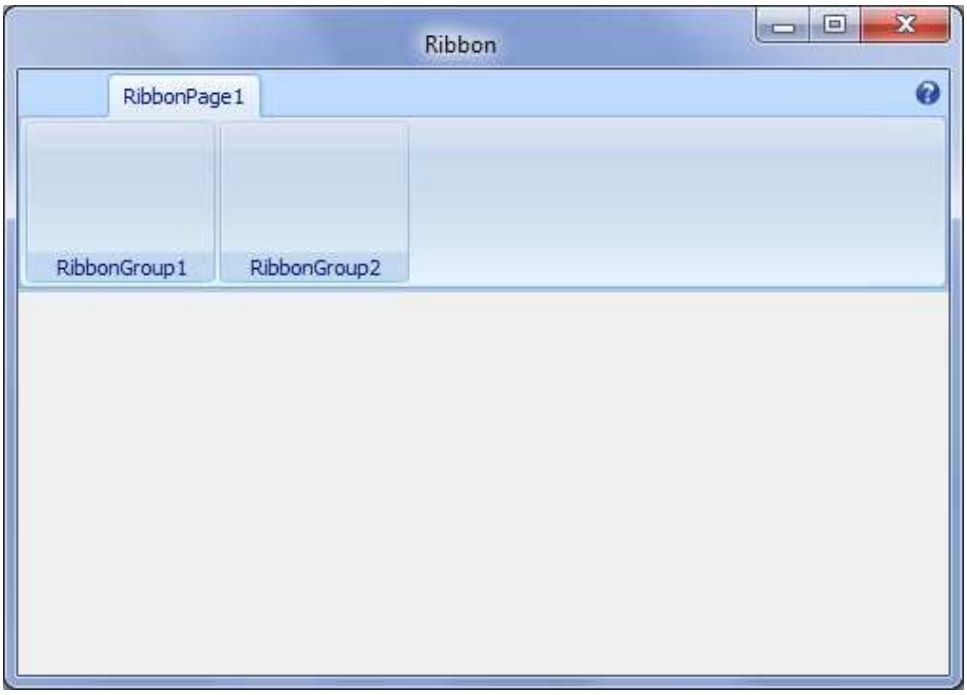


Теперь на вкладке ***RibbonPage1*** создадим две группы. Для этого необходимо выбрать в дизайнере вкладку и, как и в случае с вкладкой, воспользоваться ссылкой в **Object Inspector** “Add Group”:



*Обратите внимание, что у вкладок Ribbon в **Object Inspector** отсутствует свойство для хранения и доступа к коллекции групп **Groups**. Это свойство определено в секции **public** родителя **TRibbonPage** – **TCustomRibbonPage**.*

Теперь можно сказать, что у нас готова программа-заготовка. Вид приложения должен быть таким:



Элементы управления Ribbon Controls в Delphi 2010.

Использование кнопок на ленте Ribbon

В этом вопросе, прежде всего, стоит рассмотреть три важных момента использования кнопок на лентах Ribbon:

1. Какой размер кнопки следует использовать для конкретной команды?
2. Какие изображения следует использовать для кнопки?
3. Какого размера должны быть изображения?

Согласно концепции использования интерфейса Ribbon в приложениях (см. статью на MSDN), первоначальный размер кнопки подбирается исходя из того, как часто команда будет выполняться пользователем при работе с программой. То есть, в данном случае, **размер имеет значение**.

К примеру, в Microsoft Word в группе «*Буфер обмена*» все команды «*Вставить...*» объединены в split-кнопку большого размера, т.к. эта команда наиболее чаще используется пользователями, чем команда «*Формат по образцу*», кнопка которой сделана в группе маленького размера.

Что касается выбора изображений для кнопки, то здесь следует придерживаться следующих правил:

1. Изображение должно соответствовать команде. Например, иконка с изображением жирной буквы «**Ж**» во всех текстовых редакторах обозначает использование полужирного шрифта и будет

некорректным использование этой иконки для кнопки, команда которой будет выполняться иначе.

2. Изображение должно быть прямым. Например, такое изображение:

считается некорректным, т.к. выполнено в перспективе. Естественно, что никто не мешает Вам использовать и его, но правила хорошего тона всё-таки следует соблюдать.

Размер изображений. Очень важный момент для работы с Ribbon Controls. Все изображение кнопок **обязательно следует дублировать и делать в двух размерах** со следующими параметрами:

16x16 и **32x32** пикселя **96 dpi**.

Глубина цвета **32 бита на пиксель**.

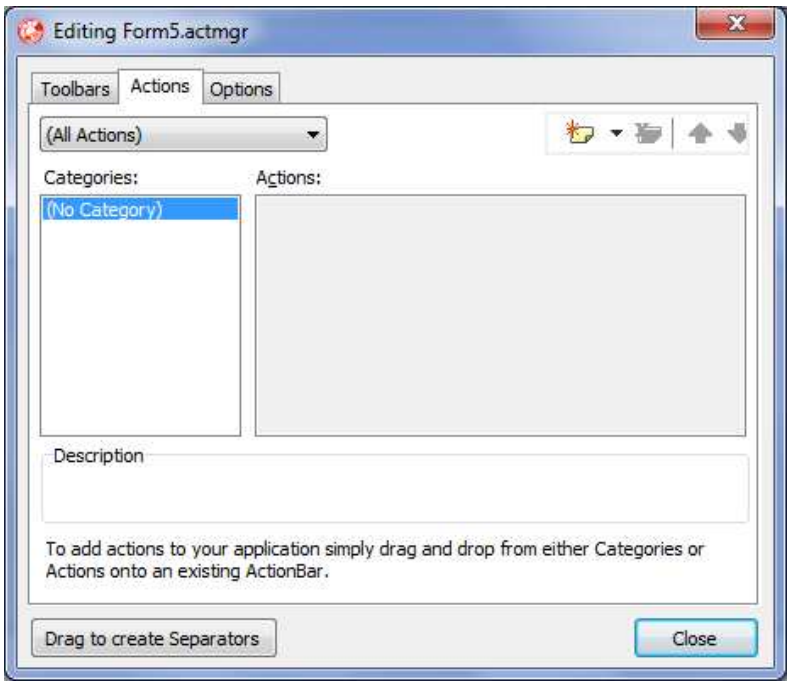
Именно по причине дублирования изображений мы в своем приложении использовали два компонента **ImageList** с предварительной настройкой свойства **ColorDepth** (глубина цвета).

Более подробную информацию о размерах и качестве изображений для лент Ribbon можно получить на сайте MSDN в статье «Specifying Ribbon Image Resources»

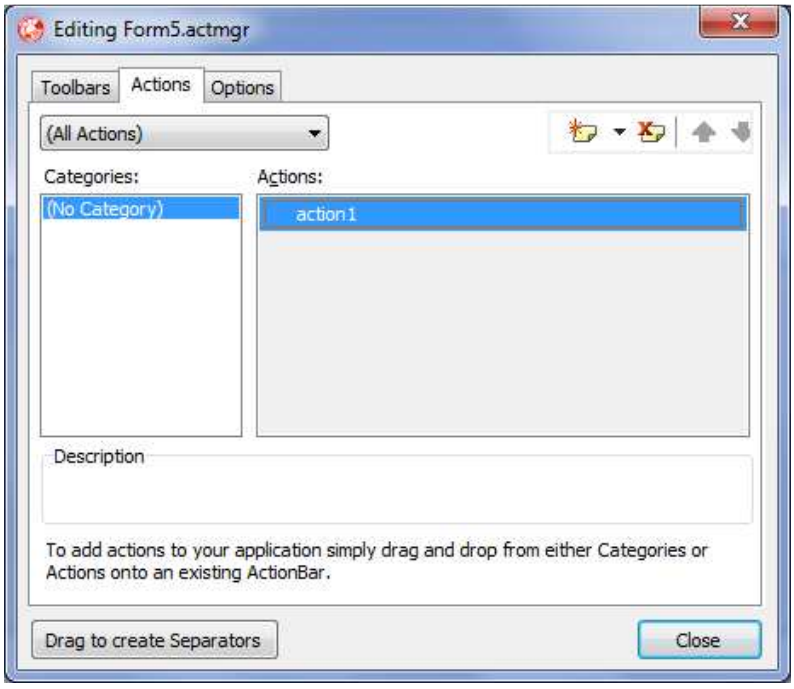
Как добавить новую кнопку в группу?

Вернемся к нашему приложению RibbonApp. Для того, чтобы добавить новую кнопку в группу необходимо выполнить следующие действия:

1. Вызвать окно редактора **ActionManager** и перейти на вкладку **Actions**. Для этого делаем двойной клик на компоненте или вызываем контекстное меню компонента и выбираем первый пункт «Customize…»:



2. Нажать кнопку «New Action..» или на клавиатуре Insert. В список **Actions** добавиться новое действие «*Action1*»:



Отредактируем свойства этого действия следующим образом:

Имя свойства	***Значение***
Caption	**Открыть URL**
Category	**sites**
Hint	**Открыть URL**

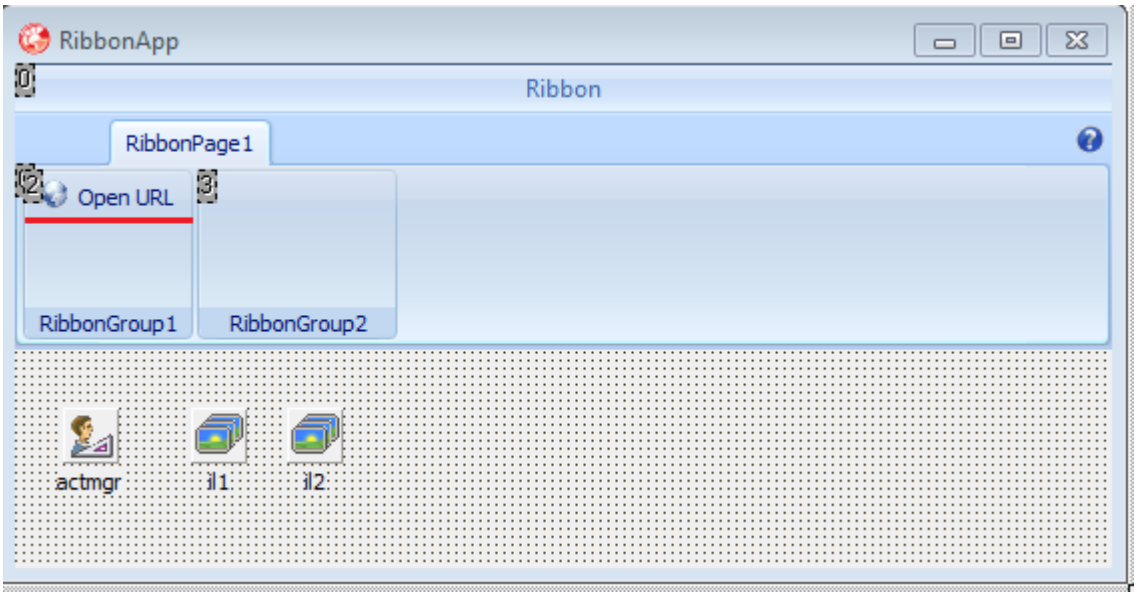
Name	cmd_openurl

Хотя свойство **Category** не является обязательным к заполнению, но его использование в значительной степени упрощает разработку программ, а также помогает пользователям более быстро настраивать ленту, т.к. все команды в этом случае разделяются по тематическим группам.

Теперь назначим изображение. Для этого загрузим в **il1** и **il2** два изображения. Я выбрал следующие:

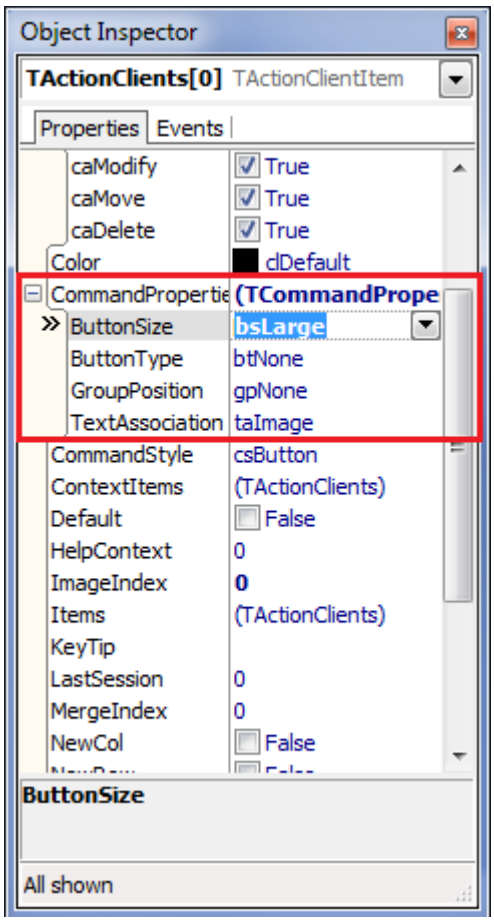
Теперь возвращаемся к команде **cmd_openurl** и в свойстве **ImageIndex** указываем **0**.

Теперь возьмите мышкой и перетащите на первую группу Ribbon эту команду. В группе должна появиться кнопка маленького размера:



Теперь выберите эту кнопку в дизайнере и обратите внимание на содержимое в **Object Inspector** – мы выбрали уже не объект **TAction**, а **TActionClientItem и** у этого объекта совершенно другие свойства**.** Давайте теперь настроим внешний вид нашей кнопки**.**

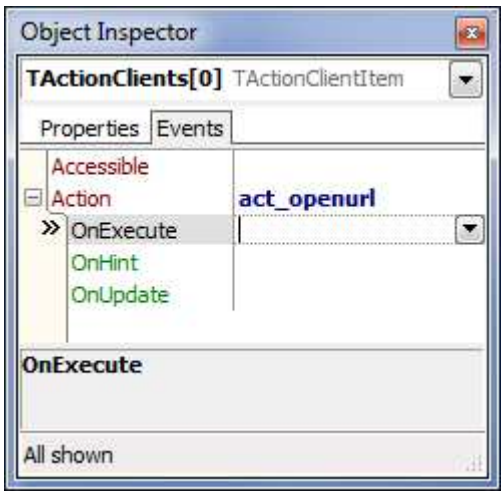
Выберите в Object Inspector вкладку свойств **CommandProperties** (Свойства команды) и измените свойство **ButtonSize** на ***bsLarge***:



Обратите внимание, что наша кнопка приняла размер по высоте равный высоте группы, а изображение автоматически изменилось – было изменено на изображение с тем же индексом из второго **ImageList.** Ширина же группы автоматически подогналась под ширину всех элементов, расположенных в ней, в нашем случае – под ширину одной большой кнопки.

Теперь запустите приложение и Вы увидите, что кнопка на панели Ribbon стала неактивна:

Чтобы кнопка была активной для ней необходимо создать обработчик действия **OnExecute**. Для этого выбираем в **Object Inspector** кнопку, переходим на вкладку **Events** и выбираем **Action → OnExecute**:



По двойному клику на пустом обработчике переходим в редактор кода и пишем обработчик для нашего действия **cmd_openurl**:

```
procedure TForm5.act_openurlExecute(Sender: TObject);
begin
 ShellExecute(Application.Handle,
  PChar('open'),
  PChar('www.webdelphi.ru'),
  PChar(0),
  nil,
 SW_NORMAL);
end;
```

*Метод **ShellExecute** открывает URL в браузере по умолчанию и находится в модуле **ShellAPI**.*

Теперь можете запустить приложение и убедиться, что кнопка стала активной и клик по ней открывает главную страницу блога webdelphi.ru.

Работа с раздваенными кнопками (Split-Buttons)

Split-кнопки имеет смысл использовать в том случае, если в программе предусмотрено несколько однотипных действий. Возвращаясь к примеру с группой «Буфер обмена» в MS Word, можно увидеть, что несколько однотипных действий, таких как «Вставить», «Специальная вставка» и «Вставить как гиперссылку» объединены в удобную Split-кнопку. Особенностью таких кнопок является то, что клик по верхней части выполняет действие по умолчанию, а нижняя часть содержит меню в котором содержаться все дополнительные действия и основное.

Давайте доработаем нашу кнопку, рассмотренную в предыдущем вопросе, и сделаем её split-кнопкой с двумя действиями:

1. **Открыть URL** . При этом будет выполняться уже написанный обработчик.

2. **Открыть в своем браузере**. При этом ссылка будет открываться в нашем собственном браузере в программе. Для этого переместите на главную форму компонент **TWebBrowser** с закладки Internet и установите у компонента свойство **Align** равным **alClient**.

Правильнее было бы второе действие сделать действием по умолчанию, но в целях сохранения порядка изложения приоритет действий установлен по порядку их добавления.

Добавьте в оба **ImageList**’а новые изображения для второго действия и создайте в **Action Manager** ещё одно действие со следующими свойствами:

Имя свойства	***Значение***
Caption	**Открыть в браузере программы**
Category	**sites**
Hint	**Открыть URL в браузере программы**
Name	**cmd_openurl_def**
ImageIndex	**1**

Теперь

1. Выберите в дизайнере формы или в окне ***Structure*** компонент ***TActionClientItem[0]*** (в Structure этот компонент должен иметь название «*Открыть &URL*»).
2. Установите свойство **CommandProperties →ButtonType** как **btSplit**.
3. В Object Inspector перейдите к свойству **Items** и откройте редактор свойств
4. Добавьте новый элемент в список.

Все эти действия продемонстрированы на рисунке ниже. Таким образом, мы добавили в список одно дополнительное действие для кнопки и сделали её Split-кнопкой.

Установите в Object Inspector свойство **Action** для **ActionClientItem0** равным **cmd_openurl_def**. В редакторе свойства **Items** элемент **ActionClientItem0** автоматически изменит своё свойство **Caption** на значение «*Открыть в браузере программы*».

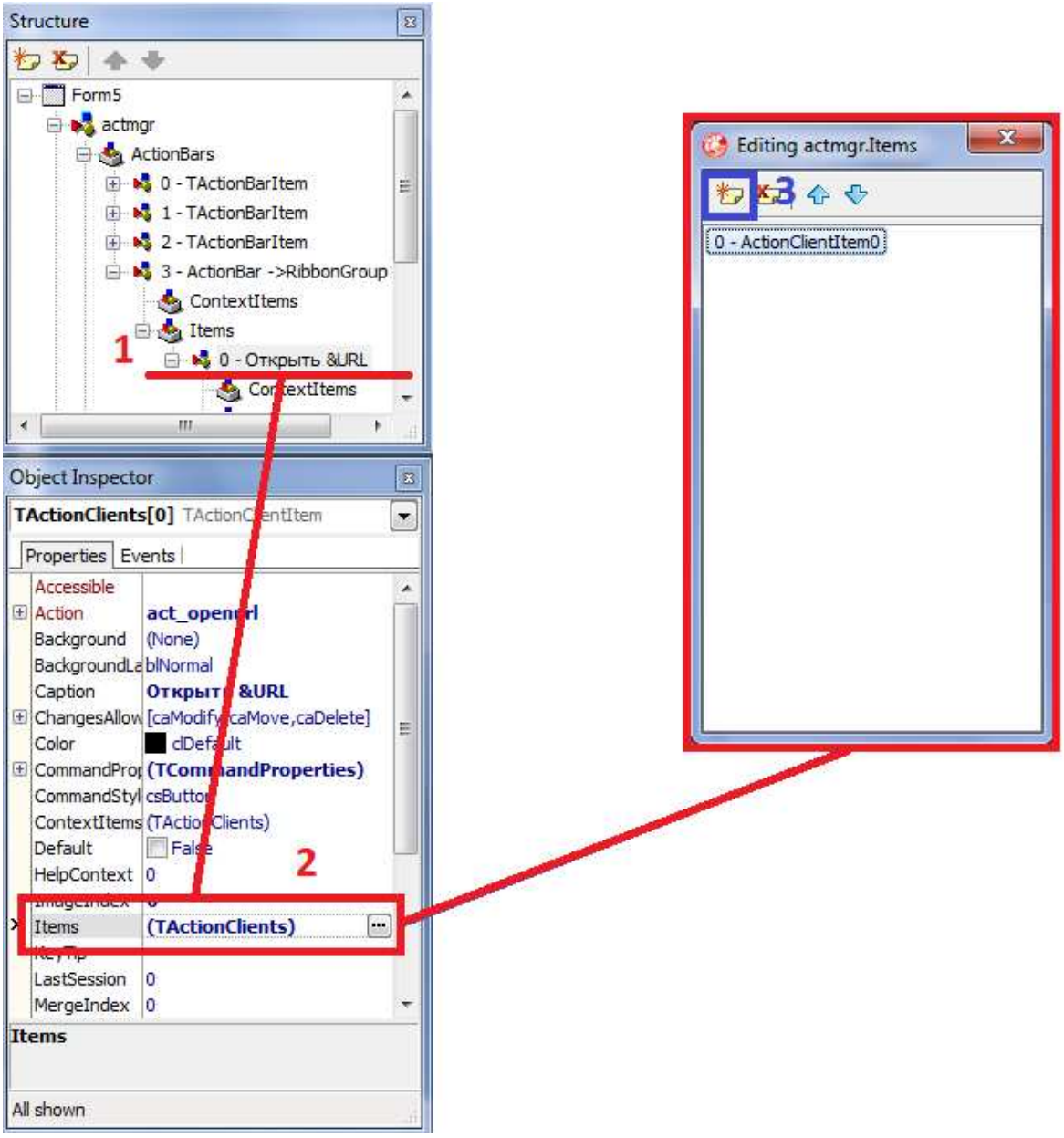
Обработчик **OnExecute** у **cmd_openurl_def** напишем следующий:

```
procedure TForm5.act_openurl_defExecute(Sender: TObject);
begin
  WebBrowser1.Navigate('www.webdelphi.ru');
end;
```

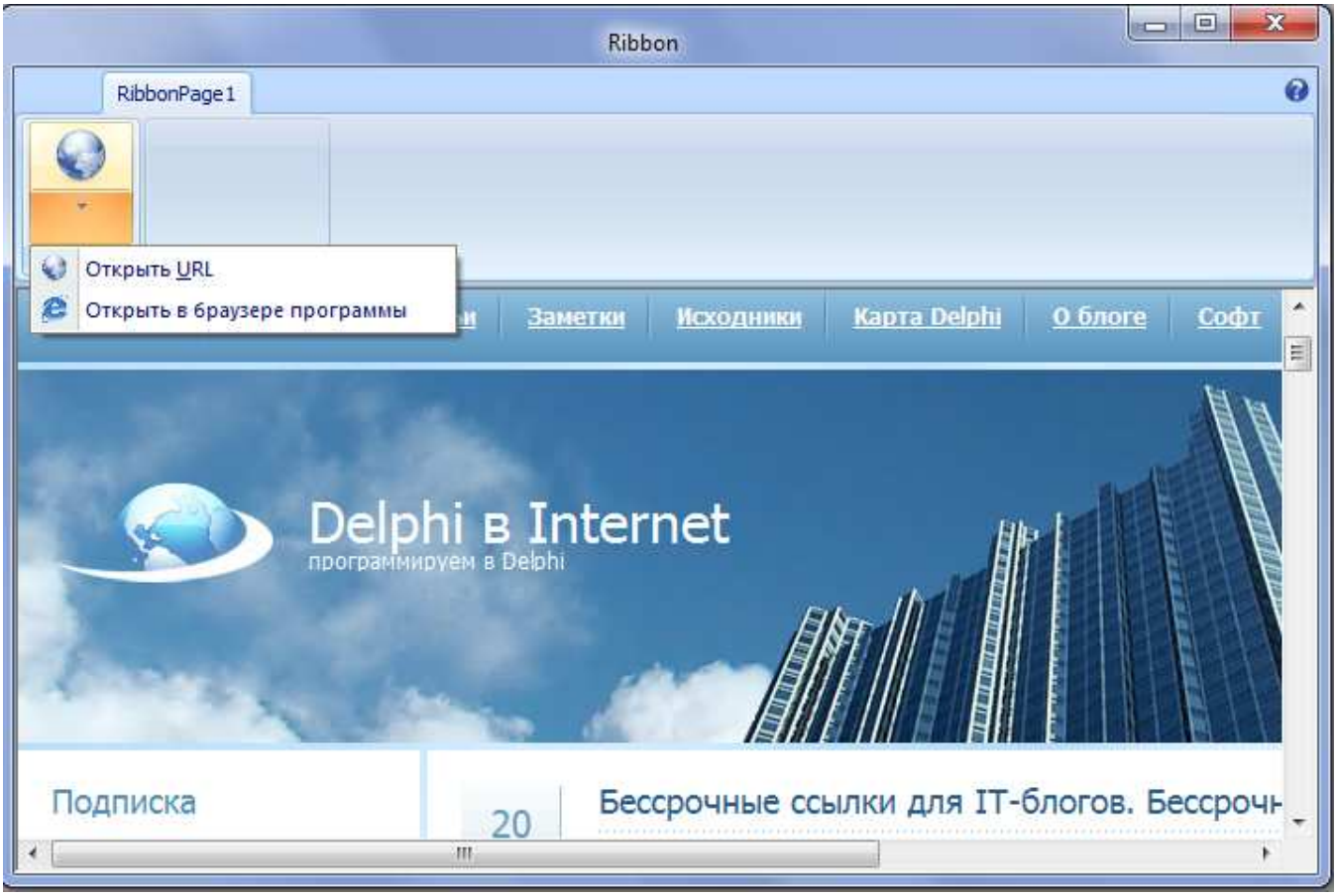
Аналогичным образом добавьте в список наше первое действие **cmd_openurl** и переместите его на первое место в списке.

Как Вы можете заметить, в дизайнере формы свойство **Caption** у split-кнопки закрывает собой нижнюю часть. В результате внешний вид приложения несколько портится. Чтобы этого избежать, необходимо установить у кнопки (***TActionClientItem[0]***) свойство **ShowCaption** в **False**.

*Никогда не стирайте свойство **Caption** у Ваших элементов управления. Во-первых, это не принесет необходимого результата – подпись кнопки будет всё равно видна при запуске приложения. Во-вторых, пустое значение **Caption** может вызвать некоторые затруднения при работе с Ribbon без использования визуальных средств разработки, например, в run-time.*



Теперь можете запустить приложение и убедиться, что split-кнопка работает как и полагается – клик по верхней части кнопки откроет URL в браузере по умолчанию, а клик по нижней части – воззовет список всех действий кнопки:



Работа с RibbonComboBox

TRibbonComboBox – это новый компонент, используемый при работе с Ribbon Controls. Несмотря на то, что этот компонент имеет очевидные сходства со стандартным **TComboBox**, его использование в интерфейсах с использованием Ribbon не только желательно, но и **необходимо**.

Вариант 1 - Использование списка Items:TStringList

Рассмотрим применение этого компонента на примере нашего приложения. Доработаем приложение таким образом, чтобы URL, введенный в **RibbonComboBox**, открывался в браузере по умолчанию или в браузере программы и при этом последний введенный URL сохранялся в списке.

Для этого разместим на **RibbonGroup2** компонент **RibbonComboBox1** с вкладки Ribbon Controls палитры компонентов и допишем обработчики **OnExecute** действий следующим образом:

```
procedure Add2History(const aURL:string);
var idx:integer;
```

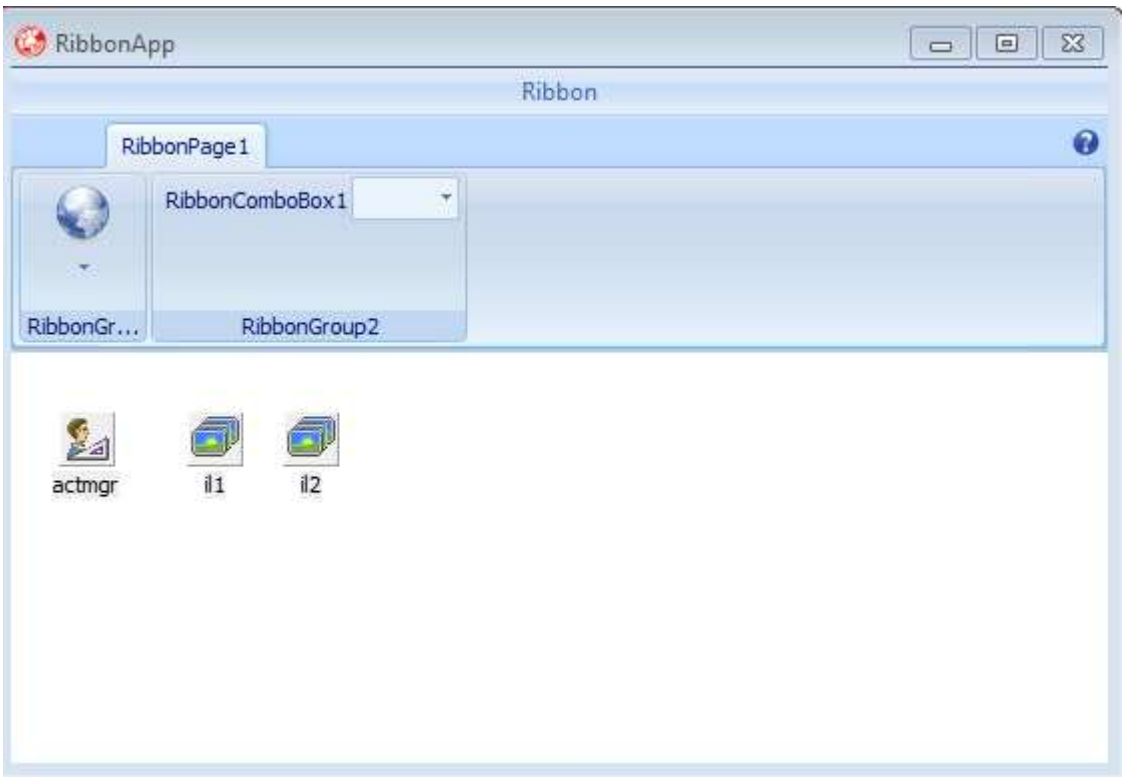
```
begin
  with Form5.RibbonComboBox1 do
    begin
      idx:=Items.IndexOf(aURL);
      case idx of
        -1:Items.Insert(0,aURL);
         0:exit;
      else
        Items.Move(idx,0);
      end;
    end;
end;

procedure TForm5.act_openurlExecute(Sender: TObject);
begin
  ShellExecute(Application.Handle,
    PChar('open'),
    PChar(RibbonComboBox1.Text),
    PChar(0),
    nil,
  SW_NORMAL);
  Add2History(RibbonComboBox1.Text);
end;

procedure TForm5.act_openurl_defExecute(Sender: TObject);
begin
  WebBrowser1.Navigate(RibbonComboBox1.Text);
  Add2History(RibbonComboBox1.Text);
end;
```

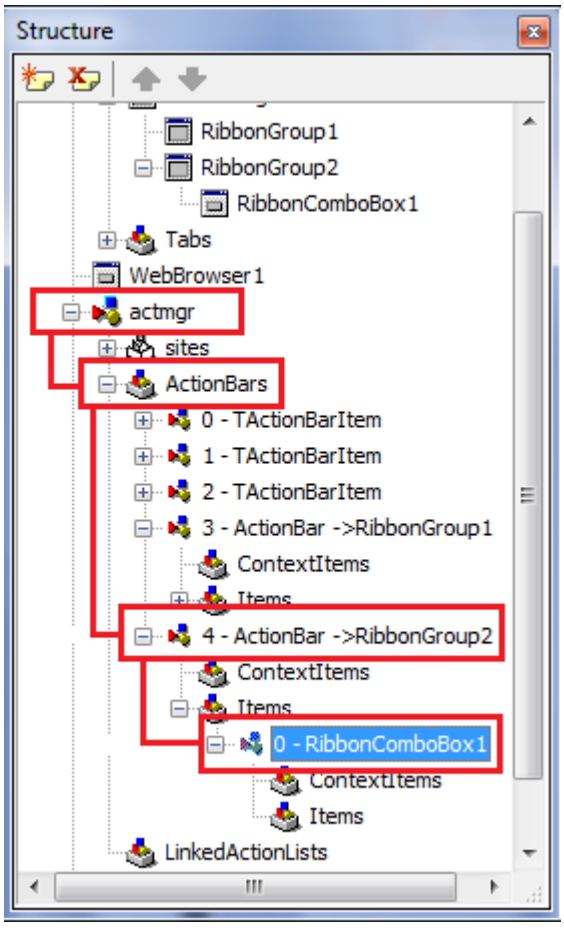
Здесь процедура **Add2History** проверяет наличие URL в списке **RibbonComboBox1** и, если такой URL найден, то перемещает его в верх списка, иначе – вставляет новый элемент в список.

На данные момент Ваше приложение должно иметь следующий вид:



Изменим внешний вид нового элемента управления. Для этого выбираем в окне Structure элемент:

actmgr →ActionBars →ActionBar->RibbonGroup2 →0-RibbonComboBox1



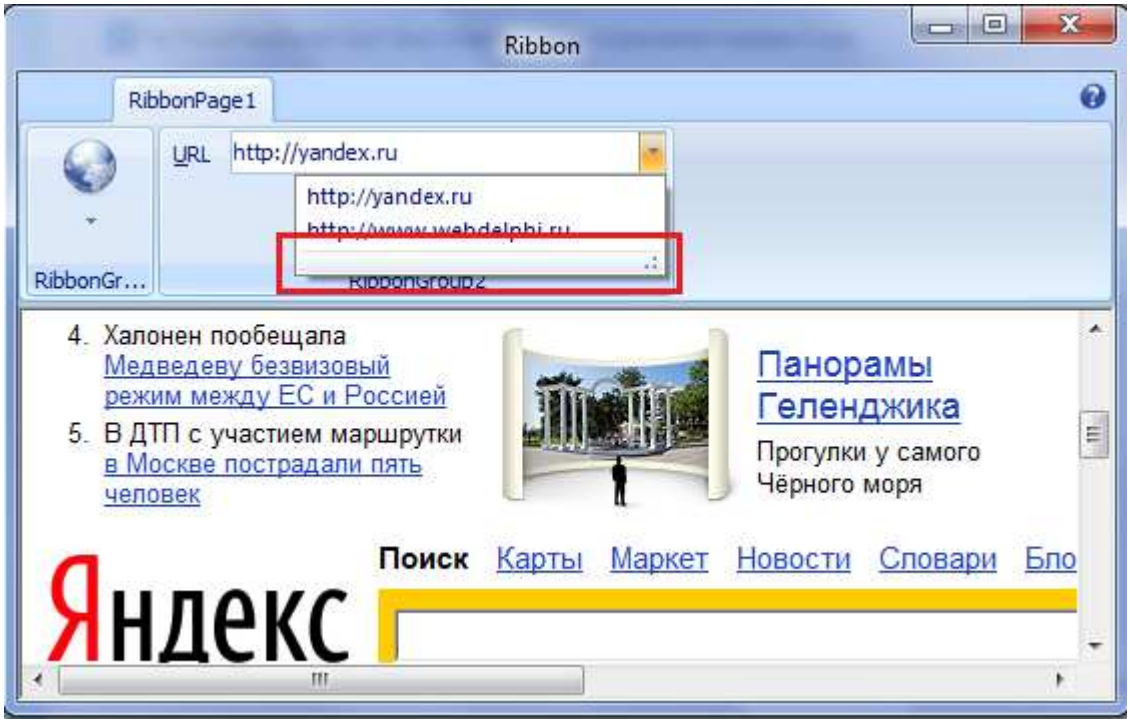
В Object Inspector изменяем свойства элемента следующим образом:

Имя свойства	*Значение*
Caption	**URL**
CommandProperties → Width	**250**
CommandProperties → AllowResize	**grBoth**

*Обратите внимание, что свойство **ShowCaption** не изменяет свое значение на **False**. Для того, чтобы скрыть подпись необходимо установить значение свойства **CommandProperties →LabelWidth** равным **0**. Значение **-1** означает, что подпись будет занимать всегда пространство по ширине равное ширине текста.*

Таким образом, мы изменили подпись компонента, увеличили ширину до 250 пикселов, а также разрешили изменение размеров выпадающего списка, как по ширине, так и по высоте.

Теперь запустите приложение и попробуйте загрузить несколько Web-страниц. Затем раскройте список **RibbonComboBox** – обратите внимание на вид выпадающего списка:



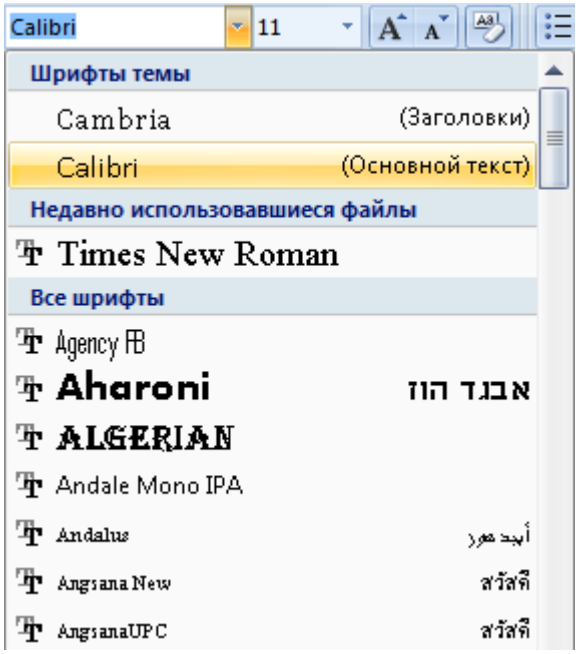
Вариант 2 - Использование действий (TAction) в RibbonComboBox.

Этот способ работы с компонентом **TRibbonComboBox** отличается от предыдущего тем, что:

1. Мы не обращаемся к свойству **Items** компонента – все элементы списка располагаются в **Action Manager**

2. Позволяет сделать интерфейс программы более удобным и красочным

Примером этого способа работы с **TRibbonComboBox** может служить список выбора шрифта в MS Word 2007:



Как видно на рисунке, все шрифты разделяются на три группы. Для этого в списке используется ещё один элемент пользовательского интерфейса – разделитель.

Для того, чтобы продемонстрировать создание такого списка в Delphi, добавим в **RibbonGroup2** ещё один **TRibbonComboBox**. Как и в первом случае выберем в окне Structure элемент:

actmgr →ActionBars →ActionBar->RibbonGroup2 →1-RibbonComboBox2

и установим для него следующие свойства:

Имя свойства	***Значение***
Caption	**&Поиск**
Width	**250**
NewRow	**True**

Теперь добавим в Action Manager новые действия. Я добавил два новых действия со следующими свойствами:

Имя свойства	***Значение***	
	1 TAction	**2 TAction**
Category	**Searches**	

Caption	yandex	google
ImageIndex	2	3

Обработчики действий следующие:

```
procedure TForm5.yandexExecute(Sender: TObject);
begin
 RibbonComboBox1.Text:='http://www.yandex.ru';
 RibbonComboBox2.Text:='Yandex';
 act_openurl_defExecute(self);
end;
procedure TForm5.googleExecute(Sender: TObject);
begin
  RibbonComboBox1.Text:='http://www.google.ru';
  RibbonComboBox2.Text:='Google';
  act_openurl_defExecute(self);
end;
```

Теперь снова возвращаемся к элементу:

actmgr →ActionBars →ActionBar->RibbonGroup2 →1-RibbonComboBox2

(теперь в окне **Structure** он должен значиться как ***&Поиск***), открываем редактор свойства **Items,** добавляем новый элемент в список и устанавливаем для этого элемента следующие свойства:

Имя свойства	*Значение*
Caption	&Поисковые системы
CommandStyle	csSeparator

Теперь добавляем в этот же список ещё два элемента и указываем у них в свойствах **Action** наши новые действия (**yandex** и **google**), не изменяя при этом другие свойства элемента.

Наш список готов к использованию. Запускаем приложение и убеждаемся, что при выборе действия в новом списке в первый **RibbonComboBox** выводится адрес главной страницы поисковой системы, а во второй – название.

Работа с CheckBox и RadioButton?

Для того, чтобы использовать на ленте такие элементы управления как **CheckBox** и **RadioButton** **ни в коем случае не следует использовать стандартные компоненты**. Несмотря на то, что на ленте Ribbon можно теоретически размещать любые компоненты из VCL, стандартные компоненты совершенно иначе перерисовываются при получении фокуса – в результате внешний вид приложения может резко снизиться.

Продемонстрируем использование элемента **CheckBox** в нашем приложении. Создайте в **Action Manager** новое действие со следующими свойствами:

Имя свойства	***Значение***
Caption	**CheckBox1**
Name	**Action1**
ImageIndex	**-1**

И создайте следующий обработчик события **OnExecute**:

```
procedure TForm5.Action1Execute(Sender: TObject);

begin

  Action1.Checked:=not Action1.Checked;

end;
```

Теперь поместите это действие на свободное место в **RibbonGroup2** и выберите в окне **Structure** следующий элемент:

actmgr →ActionBars →ActionBar->RibbonGroup2 →&CheckBox1

В Object Inspector установите следующие свойства элемента:

Имя свойства	*Значение*
Caption	**&CheckBox1**
CommandStyle	**csCheckBox**
Action	**Action1**
Width	**-1**

Теперь запустите приложение и убедитесь, что наш **CheckBox** работает как и стандартный, но при получении фокуса его вид полностью соответствует теме оформления ленты Ribbon.

Аналогичным образом используются и элементы управления **RadioButton**.

Работа с галереями (gallery) в Ribbon.

Использование галерей в Ribbon Controls обосновано в том случае, если:

1. Имеется чётко определенный набор связанных команд, которые пользователи будут использовать. Например, набор команд выбора цвета текста в MS Word выполнен в виде галереи – каждый выбор цвета – отдельная команда.
2. Действие той или иной команды лучше всего отобразить визуально.
3. Каждая команда будет выполняться в один клик, то есть не будет диалоговых окон с целью уточнить намерения пользователя.

В Ribbon есть два вида галерей:

In-ribbon Gallery: используются в том случаях если:

- содержат часто выполняемы е команды, например, галерея экспресс-стилей текста в Word (Заголовок 1, Заголовок 2 и т.д.).
- эскизы команд могут быть эффективно отображены на пиктограммах размером 48x48 пикселей.
- Есть возможность отобразить минимум три эскиза в пространстве ленты без ущерба для других элементов управления.

Drop-down gallery: представляют собой галерей эскизы которых отображаются в раскрывающемся списке (см. выбор стиля нумерованного списка в MS Word 2007).

В таких галереях могут содержаться пиктограммы со следующими размерами: **16x16**, **32x32**, **48x48**, **64x48**, **72x96**, **96x72**, **96x96** или **128x128** пикселей. В Delphi при использовании галерей можно использовать изображения с произвольными размерами, однако при работе следует строго придерживаться следующего правила: **Все элементы галерей должны иметь одинаковый размер.**

Рассмотрим пример создания нескольких drop-down галерей в Delphi с различными параметрами эскизов.

Drop-Down галерея с изображениями 16x16 пикселя.

Создайте на ленте Ribbon новую группу и присвойте свойству **Caption** значение «**Галереи**». Теперь добавьте в **Action Manager** новое действие со следующими свойствами:

Имя свойства	*Значение*
Category	**Gallery**
Caption	**GalleryButton**
Name	**GalleryAction**
ImageIndex	**0**

Создайте для этого действия любой обработчик события **OnExecute**, например, пустой:

```
procedure TForm5.GalleryActionExecute(Sender: TObject);
begin
//
end;
```

Теперь, разместите это действие на новой группе Ribbon и установите для элемента управления следующие свойства:

Имя свойства	*Значение*
Caption	**&GalleryButton**

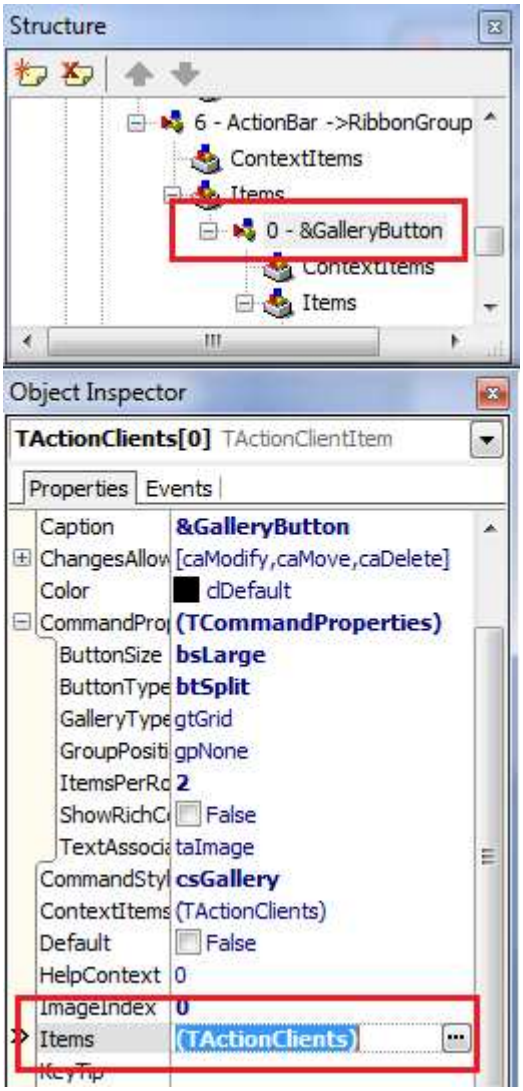
CommandStyle	**csGallery**
CommandProperties→ButtonSize	**bsLarge**
CommandProperties→ButtonType	**btSplit**
CommandProperties→GalleryType	**gtGrid**
CommandProperties→ItemsPerRow	**2**

В итоге мы создали новый элемент управления – drop-down галерею. Элементы галерей будут располагаться в табличной форме по 2 элемента на строку.

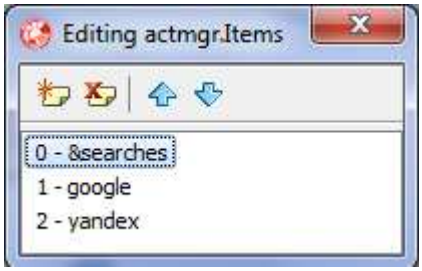
Вид группы Ribbon с размещенным на ней новым элементом управления представлен на рисунке:

Теперь разместим в галерее свои эскизы размером **32x32 пикселя**. Для этого воспользуемся уже имеющимися у нас действиями в группе Searches **Action Manager** (**yandex** и **google**).

Выберите в окне Structure элемент галереи и перейдите в редактор свойств **Items**:



Теперь, находясь в редакторе свойств, добавляем в список три новых элемента как показано на рисунке:



Характеристики элементов списка (по порядку их следования на рисунке):

0 – TActionClients[0]

Имя свойства	*Значение*
Caption	**searches**
CommandStyle	**csSeparator**

1 – TActionClients[1]

Имя свойства	*Значение*
Action	**google**
Caption	**google**

2 – TActionClients[2]

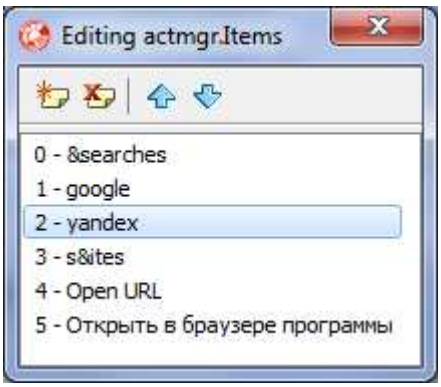
Имя свойства	*Значение*
Action	**yandex**
Caption	**yandex**

Остальные свойства элементов следует оставить без изменения.

Обратите внимание, на свойство ***Caption*** *первого элемента в списке галереи (****csSeparator****) – это имя* ***обязательно должно соответствовать категории, в которой находятся следующие за разделителем элементы****. В нашем случае действия* ***yandex*** *и* ***google*** *находятся в категории* ***searches*** *– метка разделителя тоже* ***searches****. Допускается использовать в имени разделителя символ* ***&****, но, тем не менее, следующая за символом строка должна соответствовать названию категории.*

Теперь запустите приложение и убедитесь, что галерея содержит два элемента, каждый из которых представлен эскизом 16x16 пикселей:

Добавим в эту галерею ещё два элемента, находящиеся в группе **sites**. Выполняемые действия, аналогичны приведенным выше – добавляем три новых элемента, первый из которых – разделитель с названием категории:



Вид галереи после добавления новых элементов:

Вы можете составлять галереи из элементов, находящихся в различных категория, но при этом должны соблюдать следующие правила:

1. Метка (**Caption**) разделителя должна соответствовать названию категории, из которой добавляются элементы.
2. По одним разделителем должны находиться элементы из одной категории
3. Эскизы изображений должны иметь один размер.

Несоблюдение этих трех простых правил может повлечь за собой искажение внешнего вида программы, например, как показано на рисунках ниже:

Эскиз	***Описание ошибки***

	Выделенный элемент был добавлен под разделитель **Searches**, но в запущенном приложении был автоматически перемещен под свой разделитель - **sites**
	Метка первого разделителя в галерее была неверной – вместо названия категории (**searches**) в **Caption** была записана строка «**searchEngines**» В результате все элементы из категории наложились друг на друга, а разделитель оказался «снизу»

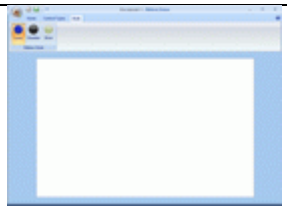
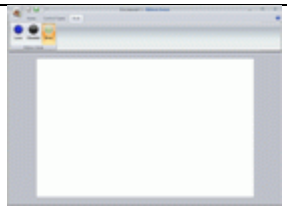
Drop-Down галерея с изображениями произвольного размера.

В представленном выше примере работы с галереями мы использовали действия, находящиеся в **Action Manager**. В результате эскизы изображений для галереи были взяты из связанных с **Action Manager** компонентов **ImageList** (см. «Программа-заготовка для работы с Ribbon Controls.»).

В случае если Вам необходимо использовать в эскизах галереи изображения других размеров этот способ работы не подойдет, т.к. с **Action Manager** могут быть связаны **ImageList** с двумя разновидностями размеров изображений (согласно основной идее дизайна Ribbon - 16x16 и 32x32 пикселя).

Рассмотрим пример создания галереи Ribbon, содержащей эскизы изображений с произвольными размерами. Пусть наша галерея будет содержать стандартные темы оформления Ribbon и при клике на элементе галереи наше приложение будет менять свой внешний вид.

Три эскиза для галереи представлены ниже:

Luna	**Obsidian**	**Silver**
		
Размеры изображений	**137x100 пикселей**	

Теперь разместите на форме приложения два новых компонента:

1.  **ActionList** (вкладка **Standard**)
2.  **ImageList** (вкладка **Win32**)

Установите для компонентов следующие свойства:

ImageList:

Имя свойства	*Значение*
Name	**il3**
ColorDepth	**cd32Bit**
Height	**100**
Width	**137**

ActionList:

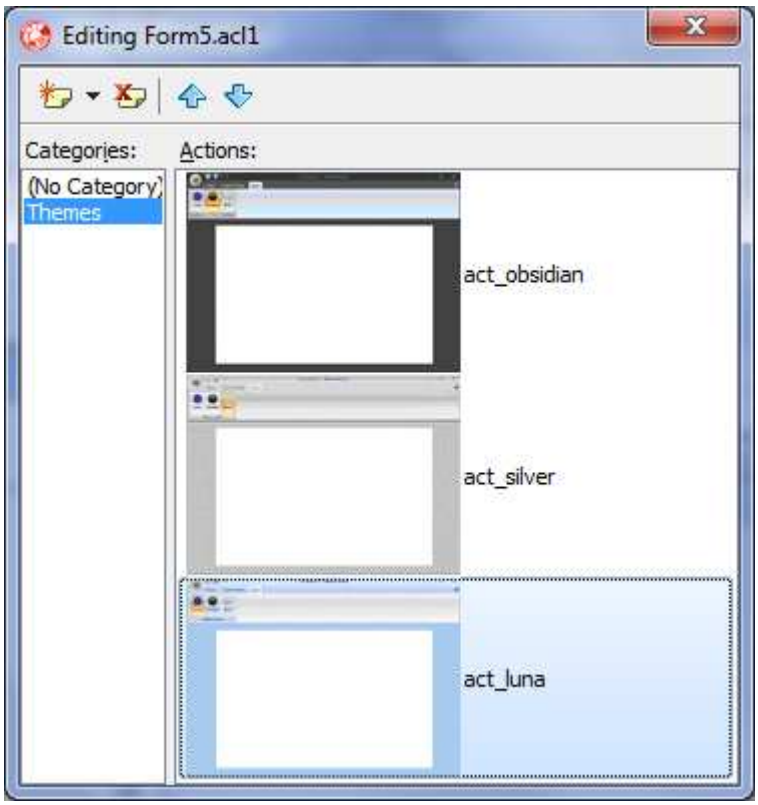
Имя свойства	*Значение*
Name	**acl1**
Images	**il3**

Загрузите в **ImageList il3** эскизы изображений.

Теперь двойным щелчком мыши на **Action List** откройте его редактор действий и добавьте 3 новых действия со следующими свойствами:

Имя свойства	*Значение*		
	Action 1	*Action 2*	*Action 3*
Category	**Themes**		
Caption	**Obsidian**	**Silver**	**Luna**
ImageIndex	**0**	**1**	**2**
Name	**act_obsidian**	**act_silver**	**act_luna**

Вид редактора **Action List** после добавления новых действий должен быть таким:

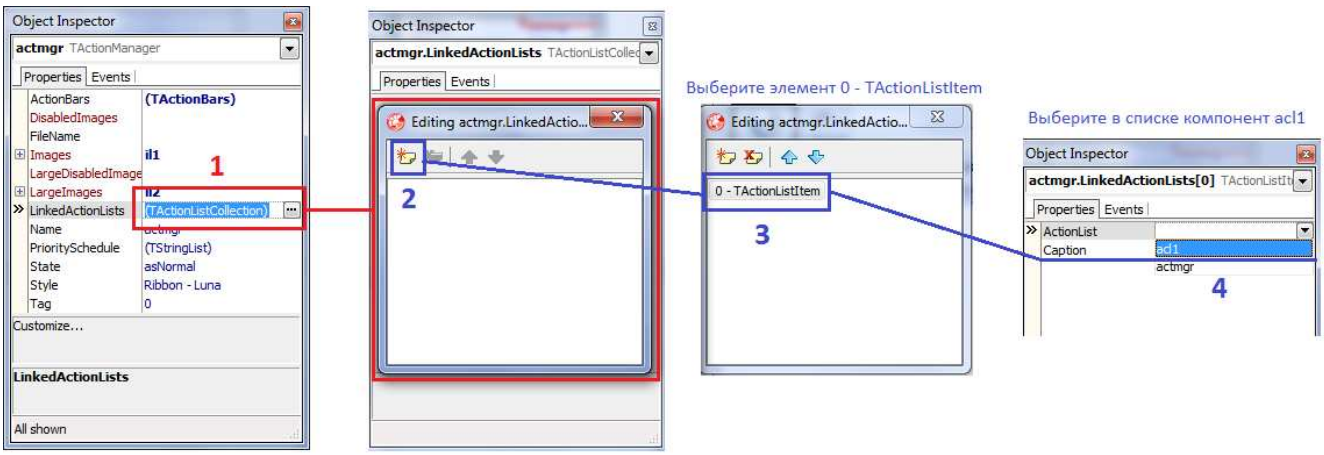


Обработчики событий **OnExceute** для этих действий будут такими:

```
procedure TForm5.act_obsidianExecute(Sender: TObject);
begin
  rbn.Style := RibbonObsidianStyle;
end;
procedure TForm5.act_silverExecute(Sender: TObject);
begin
  rbn.Style := RibbonSilverStyle
end;
procedure TForm5.act_lunaExecute(Sender: TObject);
begin
  rbn.Style := RibbonLunaStyle;
end;
```

Для того, чтобы задействовать в приложении использование тем Ribbon в раздел uses главного модуля приложения подключите три модуля: **RibbonLunaStyleActnCtrls**, **RibbonObsidianStyleActnCtrls** и **RibbonSilverStyleActnCtrls.**

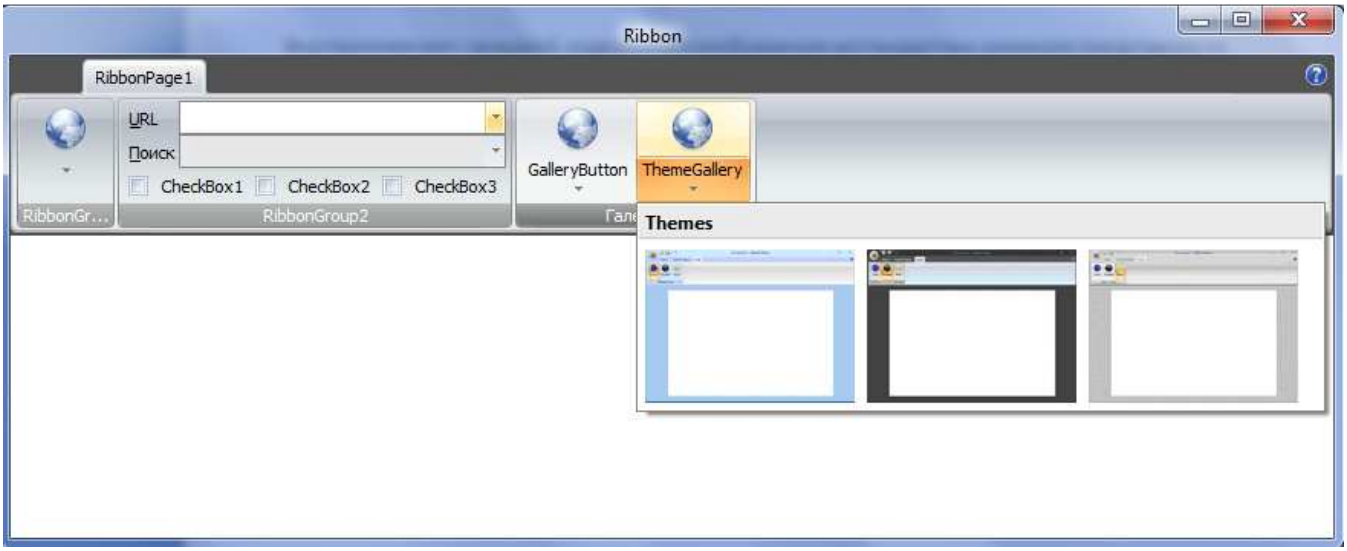
Теперь выберите в дизайнере формы или в окне Structure компонент **Action Manager** (**actmgr**) и отредактируйте его список **LinkedActionLists**, добавив в него компонент **acl1**:



В **Action Manager** теперь будут содержаться действия из **acl1** с нестандартными размерами изображений 137x100 пикселей.

Теперь можете добавить эти элементы в новую галерею таким же способом, как и в первом случае, не забыв при этом установить свойство **Caption** разделителя как «**Themes**».

Вид приложения с галереей, содержащей изображения нестандартных размеров представлен на рисунке:

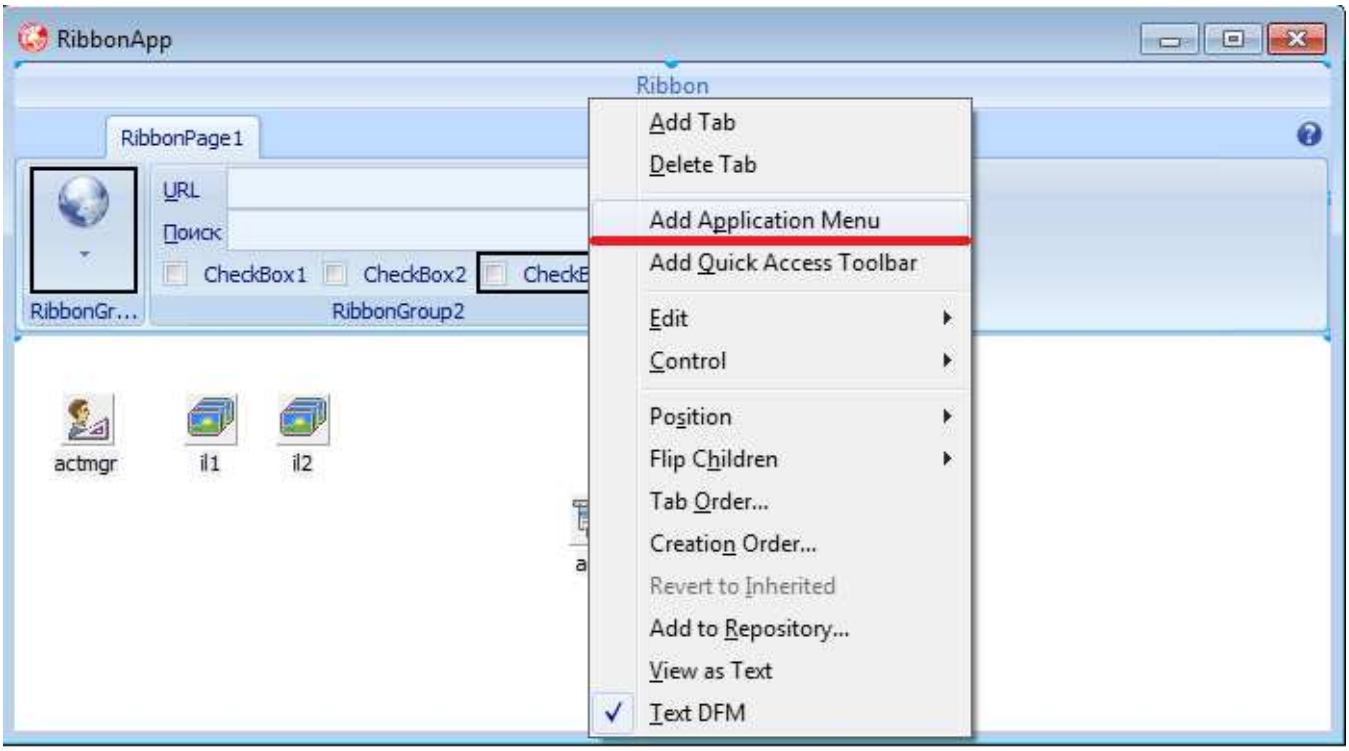


Работа с главным меню приложения.

В отличие от стандартных линейных меню приложений, к которым мы так все привыкли, в Ribbon все элементы главного меню сосредоточены всего лишь в одной кнопке, которая называется «*Application Button*» и располагается в левом верхнем углу главной формы приложения.

В Delphi главное меню Ribbon представлено объектом **TRibbonApplicationMenuBar**.

Для того, чтобы добавить кнопку «*Application Button*» необходимо выделить компонент **TRibbon** и воспользоваться ссылкой в **Object Inspector** “*Add Application Menu*” или выбрать в контекстном меню компонента одноименную функцию:

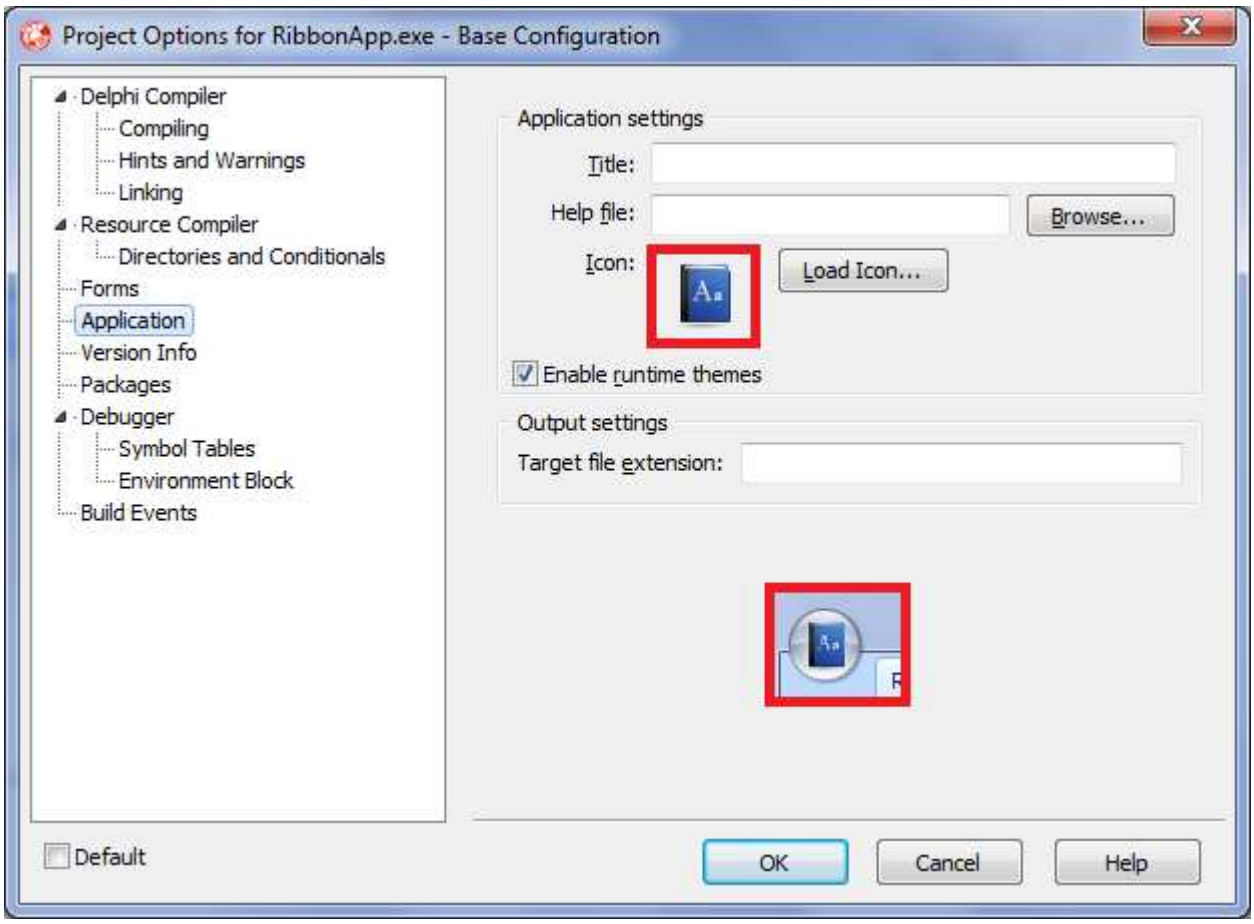


По умолчанию на кнопке главного меню приложения отображается вашего приложения, т.е. если Вы не изменяли свойства проекта, то в запущенном приложении кнопка главного меню будет иметь следующий вид:

Для того, чтобы сменить иконку кнопки вам необходимо зайти в меню Delphi:

Project→Options→Application

и загрузить новую иконку приложения. Иконка должна соответствовать требованиям Ribbon (96 dpi, глубина цвета 32 бита):

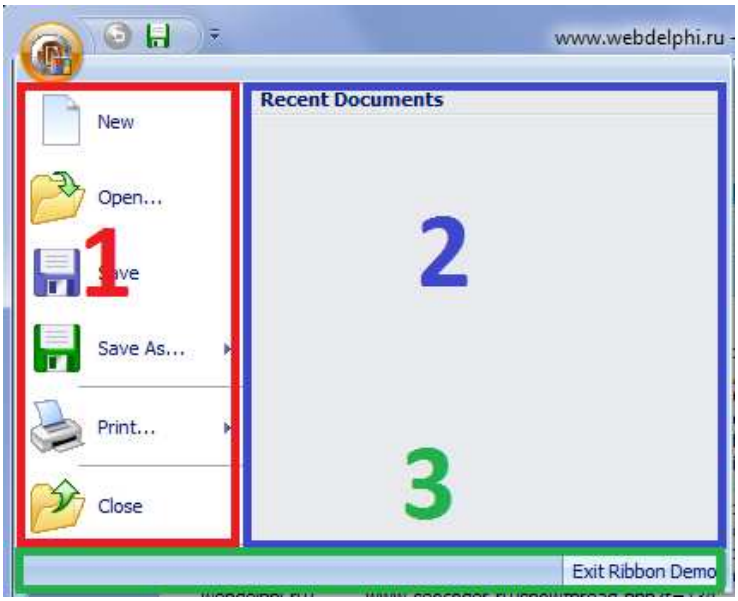


Второй вариант смены иконки у кнопки главного меню заключается в редактировании свойств компонента **TRibbon**:

TRibbon → **ApplicationMenu**→**Icon** (иконка меню)

TRibbon → **ApplicationMenu**→**IconSize** (размер иконки из ряда: ***[isSmall, isMedium, isLarge]***)

Теперь рассмотрим более подробно состав главного меню приложения. Меню состоит из трех областей:



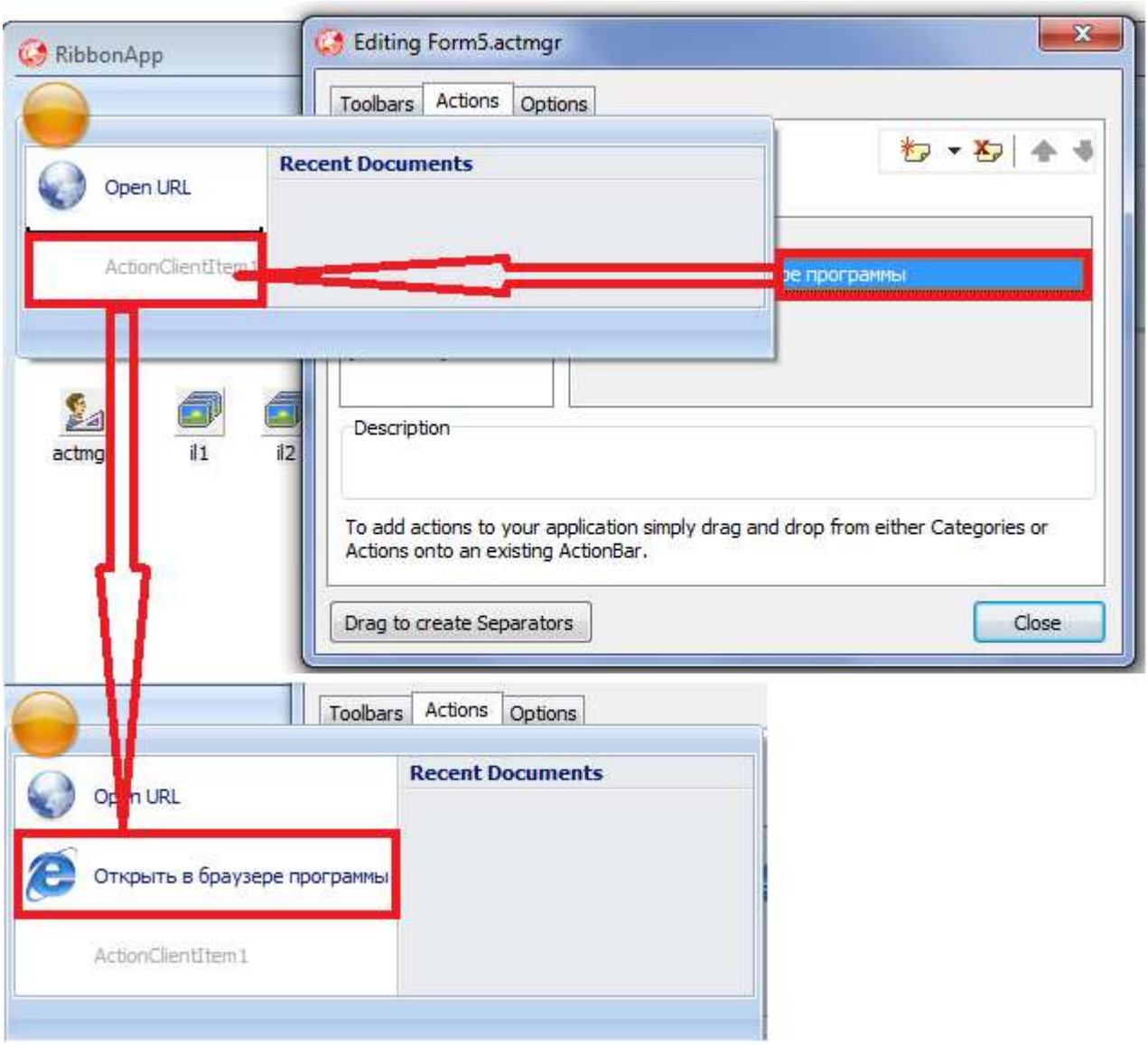
Область 1 – здесь располагаются все элементы меню.

Область 2 – в основном эта область предназначена для отображения ссылок на последние открытые документы, но также может использоваться для отображения дополнительных элементов меню.

Область 3 – чаще всего эта область предназначается для расположения в ней кнопок для выхода из приложения, но также как и **область 2** эта панель может использоваться для расположения на ней других команд меню. Рассмотрим работу с главным меню по порядку, начиная с первой области.

Работа с пунктами меню в Ribbon

Для того, чтобы добавить новый пункт меню достаточно открыть список действий в **Action Manager** и перетащить необходимое действие на область меню:

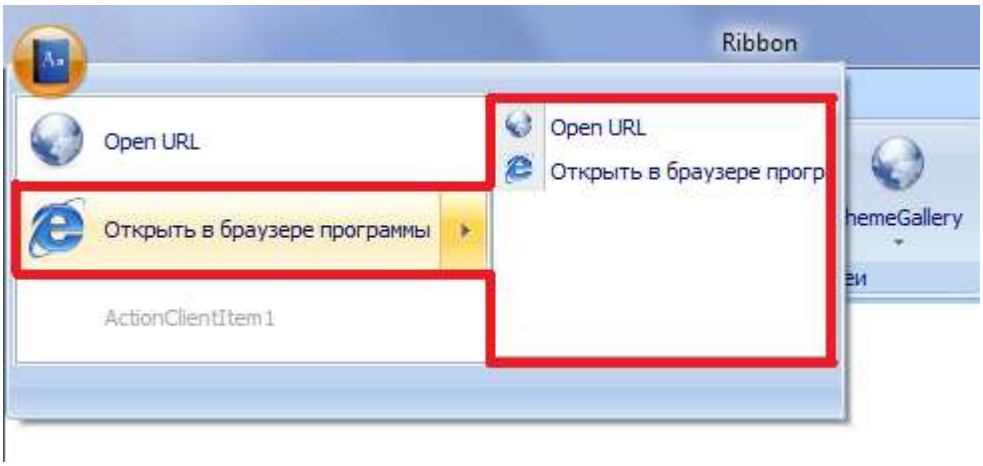


Для того, чтобы создать вложенные пункты в меню (submenu) необходимо:

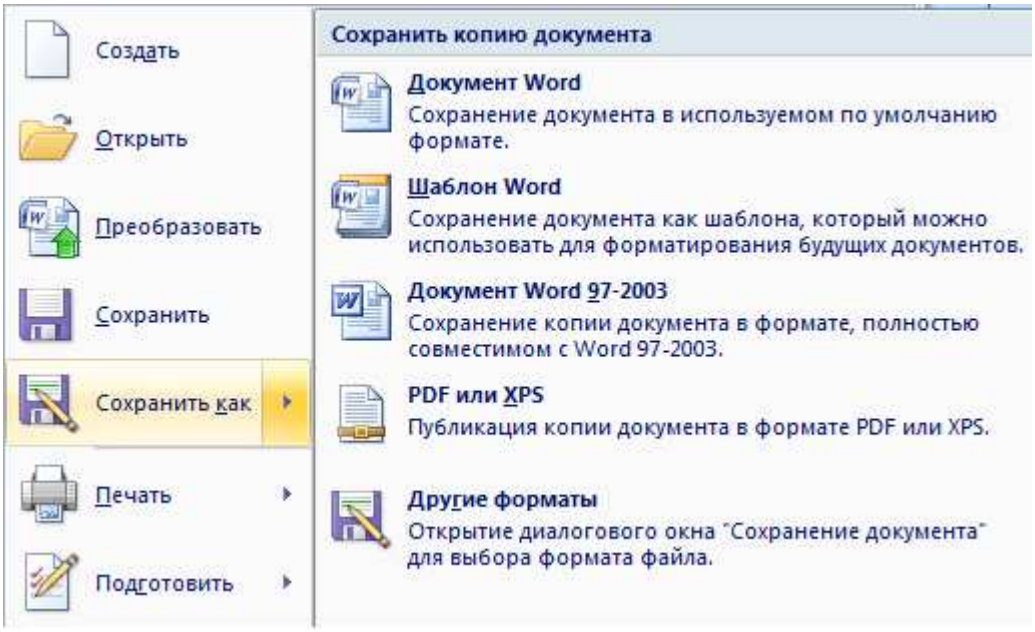
1. Выбрать пункт меню и в **Object Inspector** установить свойство **CommandStyle** = **csMenu**

2. Заполнить коллекцию **Items** необходимыми элементами – теми, которые будут находиться в подменю.

В этом случае главное меню приложения будет выглядеть следующим образом:



Если Вы хотите создать пункты подменю так как это сделано, например, в MS Word:



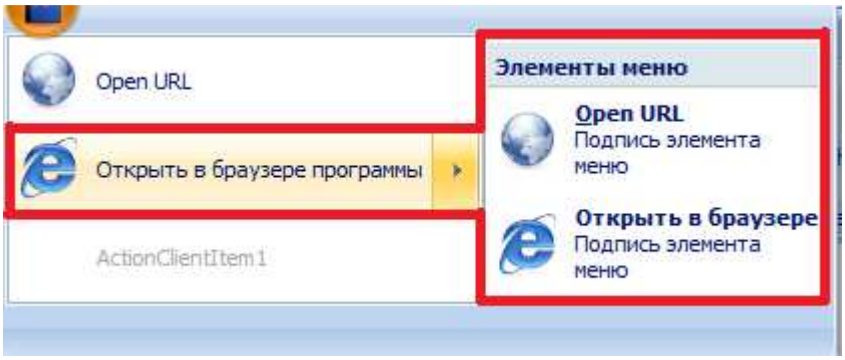
То последовательность действие будет следующей:

1. Добавляем в свойство Items новый элемент со свойством **CommandStyle=csSeparator**

2. Добавляем в Items новые пункты подменю устанавливая для каждого из них следующие свойства:

Имя свойства	*Значение*
CommandStyle	**csMenu**
CommandProperties→Content	**“Подпись для элемента меню"**
CommandProperties→ShowRichContent	**true**

В итоге Вы получите меню, как показано на рисунке:



Работа с разделом Recent Documents в главном меню Ribbon

Раздел меню Recent Documents (правая панель) может использоваться для:

1. хранения ссылок на ранее открытые файлы
2. хранения каких-либо дополнительных команд меню

Для работы с этим разделом меню у объекта **TRibbonApplicationMenuBar** предусмотрена отдельная коллекция элементов **RecentItems: TOptionItems**.

Использование раздела Recent Documents для хранения ссылок на файлы

Рассмотрим пример того как можно использовать правую панель меню в качестве хранения ссылок на открытые ранее файлы.

Вариант 1 – автоматическое создание обработчика OnExecute. Метод AddRecentItem

В том случае, если Вы работаете не с URL, а с файлами на жестком диске, например с текстовыми файлами, документами Word и т.д., то можно избежать лишней работы с исходным кодом и для добавления нового элемента в список **RecentItems** воспользоваться методом:

TRibbon→ AddRecentItem(const FileName: string)

Этот метод добавляет новый элемент в список и помещает его в самый верх, если при добавлении будет найдена ранее добавленная аналогичная ссылка, то эта ссылка будет удалена из списка.

При этом каждый новый элемент списка получит свой обработчик события **OnExecute** – **RecentActionHandler**, который, в случае клика по элементу списка в меню, передаст в событие:

TRibbon→ OnRecentItemClick

Данные о файле и индекс элемента списка по которому произведен клик:

```
TRecentItemClickEvent = procedure(Sender: TObject; FileName: string; Index: Integer) of object;
```

Рассмотрим пример работы со ссылками на файлы, используя предложенный вариант работы.

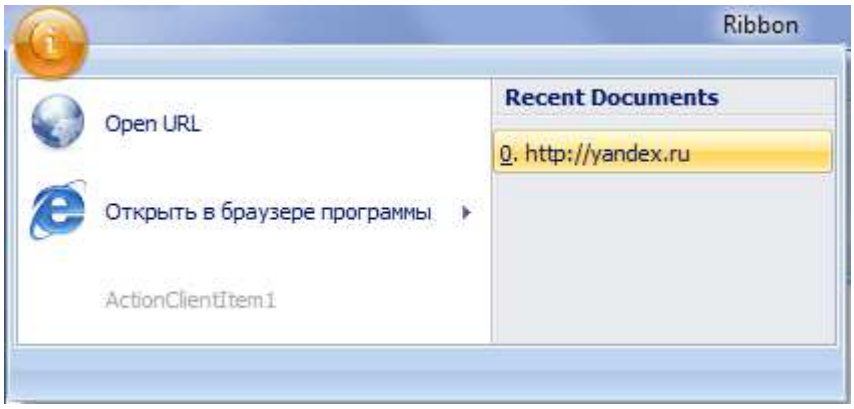
У нас уже есть действие **act_openurl** обработчик события **OnExecute** которого открывает URL в браузере по умолчанию. Доработаем этот обработчик таким образом, чтобы каждый вновь открываемый URL автоматически попадал в список **RecentItems.**

```
procedure TForm5.act_openurlExecute(Sender: TObject);
begin
 […]
 rbn.AddRecentItem(RibbonComboBox1.Text)//добавляем новую ссылку в список
end;
```

Теперь необходимо обеспечить, чтобы клик по новой ссылке открывал URL, причём также как это сделано в обработчике **OnExecute** – в браузере по умолчанию. Создадим обработчик события **OnRecentItemClick** у компонента **TRibbon**:

```
procedure TForm5.rbnRecentItemClick(Sender: TObject; FileName: string;
  Index: Integer);
begin
 ShellExecute(Application.Handle,
   PChar('open'),
   PChar(FileName),
   PChar(0),
   nil,
  SW_NORMAL);
end;
```

Запустите приложение, наберите в **RibbonComboBox1** любой URL и откройте его в браузере по умолчанию. Убедитесь, что в списке «*Recent Documents*» появился новый элемент:



Клик по новому элементу в списке откроет этот URL в окне браузера.

Как видно из рисунка выше, список «*Recent Documents*» нумеруется, начиная с нуля, а элементы списка не содержат каких-либо иконок.

Если Вам необходимо, чтобы напротив ссылки, красовалась иконка, то можно воспользоваться вторым вариантом работы.

Вариант 2 – свой обработчик OnExecute и ручное помещение ссылки в список

На данный момент наше приложение может открывать какие-либо URL в собственном браузере. Пусть все открываемые ссылки будут храниться в разделе **Recent Documents**.

До этого момента все наши элементы меню (**TAction**) выполняли однозначно определенное действие, которое мы записывали в обработчик события **OnExecute**.

В случае работы со списком ранее открытых ссылок/файлов мы не можем заранее предугадать, какая ссылка будет открыта и, тем более, не можем для каждого элемента списка **RecentItems** в запущенном приложении создавать новый обработчик **OnExcute**. Поэтому сейчас нам придётся немного поработать в редакторе кода.

Первое, что нам необходимо сделать – это написать универсальный обработчик события **OnExecute** для всех элементов, находящихся в списке **RecentItems.**

Можно предложить такой вариант реализации:

```
procedure TForm5.OpenRecentURL(Sender: TObject);
begin
  WebBrowser1.Navigate((Sender as TAction).Hint);
end;
```

В приведенном методе мы приводим объект Sender к **TAction** (которым **Sender** и является) и используем в качестве URL текст, находящийся в подсказке этого объекта (в свойстве **Hint**). При этом мы не указываем явно, какой **TAction** мы используем, следовательно, процедуру **OpenRecentURL** мы можем использовать в качестве обработчика **OnExecute** для любого действия, что нам и требовалось.

Следующим шагом будет написание процедуры, которая будет добавлять новый элемент в список **RecentItems.**

```
procedure TForm5.Add2Recent(const aURL: string);
var OptionItem: TOptionItem;
    Action: TAction;
begin
 OptionItem:=rbn.ApplicationMenu.Menu.RecentItems.Add; //создали новый элемент в списке
 Action:=TAction.Create(actmgr); //создали новый обработчик действия
 Action.ImageIndex:=0; //выбрали иконку для элемента
 Action.OnExecute:=OpenRecentURL; //назначили обработчик
 Action.Caption:=aURL; //определили метку
 Action.Hint:=aURL; //определили подсказку
 OptionItem.Action:=Action; //назначили обработчик действия элементу коллекции
end;
```

Обратите внимание, на создание объекта TAction:

Action:=TAction.Create(actmgr);

мы создали объект, используя коллекцию действий в Action Manager и, поэтому в дальнейшем без проблем смогли воспользоваться свойством ImageIndex:

Action.ImageIndex:=0;

назначив изображение из Image List, связанного с Action Manager через его свойство Images.

При других способах создания объекта TAction, например при таком:

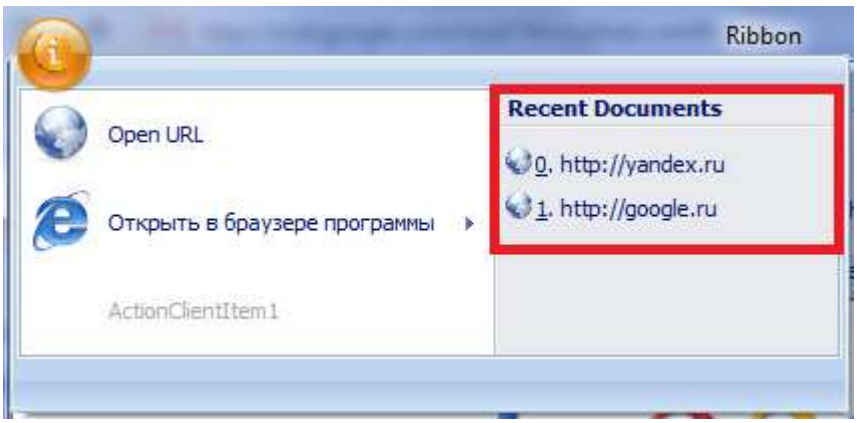
Action:=TAction.Create(self);

процедура будет выполняться корректно, но изображение для Action вы уже не назначите

Теперь остается только чуть-чуть видоизменить обработчик события **OnExecute** у объекта **act_openurl_def** (открытие URL в браузере программы):

```
procedure TForm5.act_openurl_defExecute(Sender: TObject);
begin
 …
 Add2Recent(RibbonComboBox1.Text);
end;
```

Теперь запускаем приложение, для верности загружаем пару страниц в браузере программы и смотрим состав главного меню приложения:



Использование раздела Recent Documents для дополнительных команд

Для того, чтобы список **RecentItems** главного меню приложения использовался для хранения дополнительных команд необходимо изменить свойство компонента **TRibbon**:

TRibbon→ApplicationMenu→CommandType

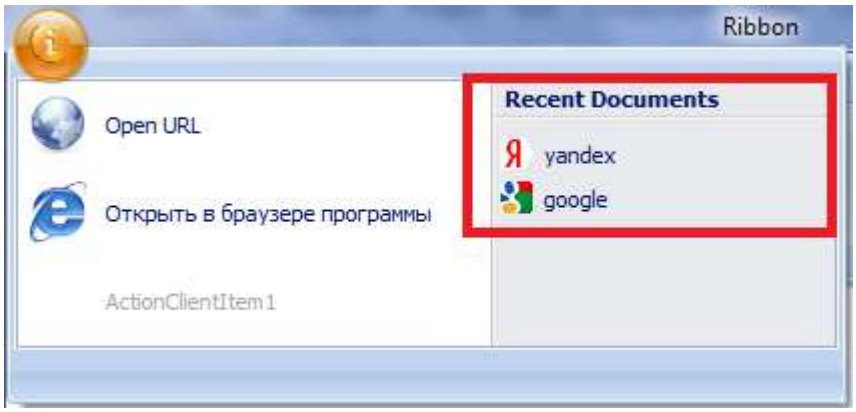
установив его в значение **ctCommands**.

Для добавления новых команд на панель необходимо воспользоваться редактором свойств списка **RecentItems**:

TRibbonApplicationMenuBar→ RecentItems

*Элементы размещаемые в правой панели меню не имеют свойств **CommandStyle** и **CommandProperties**, т.к. относятся к другому классу объектов. Поэтому Вы не сможете организовать в этой панели меню с разделителями, большими иконками, подпунктами и т.д. Лучше всего использовать эту панель по её прямому назначению, т.е. для хранения ссылок на ранее открытые документы.*

В отличие от ссылок на ранее открытые документы, команды в правой панели не нумеруются:

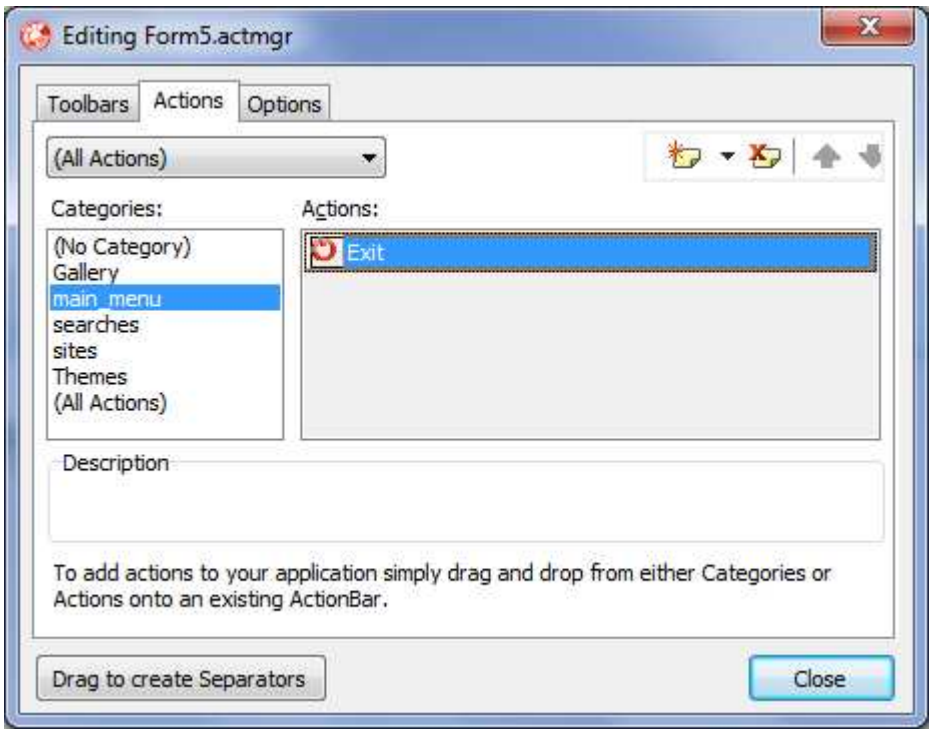


Помещаем кнопку «Выход» в главное меню программы

Рассмотрим работы с областью 3 главного меню (нижняя панель). Для добавления новых команд на эту панель необходимо воспользоваться редактором свойств списка **OptionItems**:

TRibbonApplicationMenuBar→ OptionItems

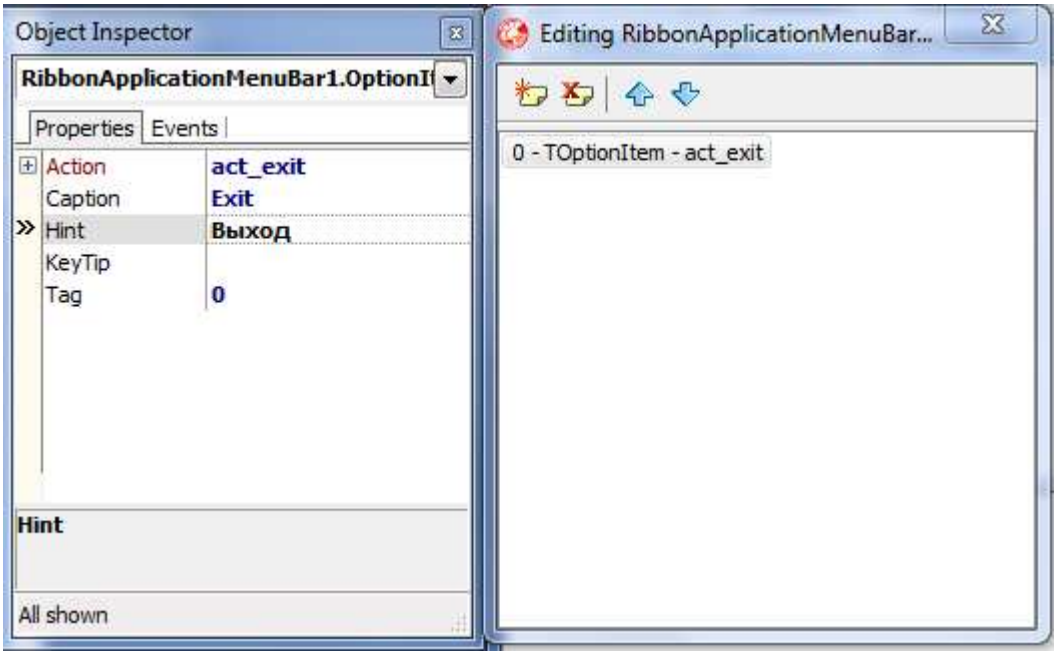
Добавим кнопку выхода из приложения в главное меню программы. Для этого создадим в **Action Manager** новое действие:



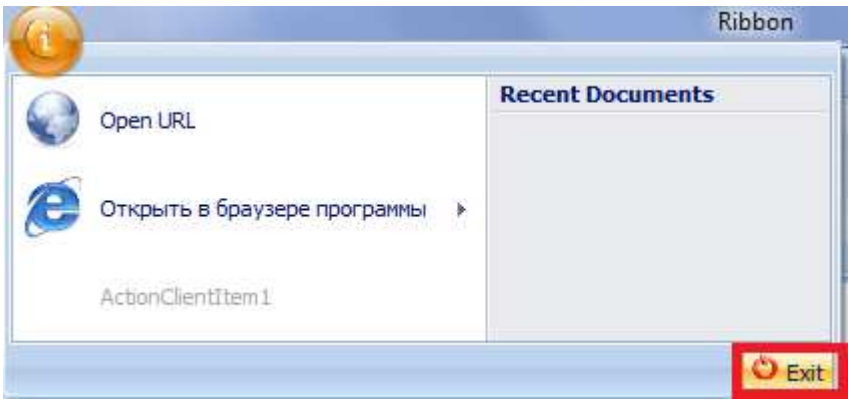
Обработчик **OnExecute**:

```
procedure TForm5.act_exitExecute(Sender: TObject);
begin
Application.Terminate
end;
```

Теперь выберите в **Object Inspector** элемент **RibbonApplicationMenuBar1,** откройте редактор свойства **OptionItems** и добавьте новый элемент в список как показано на рисунке:



Теперь запустите приложение и убедитесь, что кнопка «**Exit**» работает:

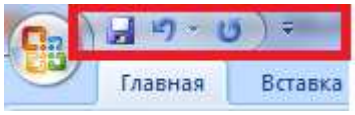


Аналогичным образом добавляются и другие кнопки.

Для выхода из программы в Ribbon Controls по умолчанию можно использовать двойной клик по кнопку главного меню.

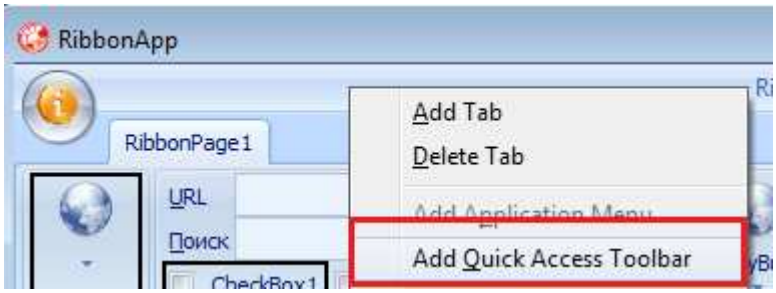
Работа с панелью быстрого доступа (Quick Access ToolBar)

В Ribbon появилась возможность создания настраиваемой панели быстрого доступа (*Quick Access ToolBar*), которая размещается справа от кнопки главного меню приложения:



В панели быстрого доступа рекомендуется размещать команды, которые Вы не можете отнести к какой-либо группе на ленте. Согласно основной концепции разработки приложений с использованием Ribbon необходимо **всегда** использовать панель быстрого доступа в своих приложениях. Даже в том случае, если Ваше приложение содержит всего одну вкладку, на которой сгруппированы все возможные команды и первоначально панель быстрого доступа остается пустой – всё равно её необходимо создать, т.к. она может обеспечить, так называемую, согласованность различных приложений. Если пользователь, например, привык, что на этой панели находятся команды сохранения или печати документа – он сможет самостоятельно настроить панель и тем самым избежать «привыкания» к работе с Вашим приложением.

Для того, чтобы разместить панель быстрого доступа на ленте необходимо вызвать контекстное меню компонента **TRibbon** и выбрать пункт меню «*Add Quick Access Toolbar*»:



По умолчанию панель появляется в верхней части ленты слева от кнопки главного меню. В Delphi при визуальной разработке интерфейса нет возможности в design-time добавить новые команды на панель быстрого доступа при помощи простого перетаскивания команд из **Action Manager** как это делалось для групп. Подобная настройка используется в уже запущенном приложении, когда пользователь вызывает контекстном меню панели и выбирает пункт «*More Commands*»:

В этом случае пользователю будет предложено самостоятельно перенести на панель необходимые ему команды и настроить панель.

Работа с PopUp-меню приложения.

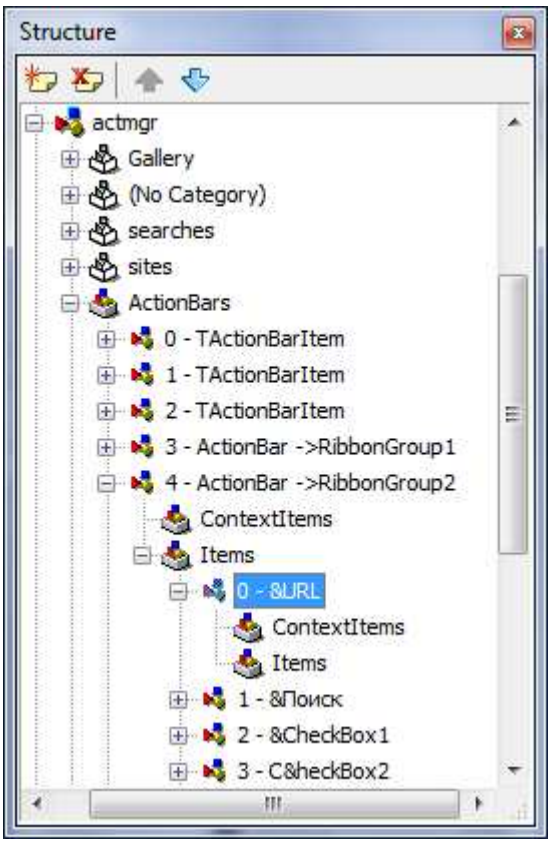
Pop-up меню элементов управления, расположенных на ленте Ribbon

Для того, чтобы создать контекстное меню для любого элемента управления, расположенного на ленте Ribbon необходимо:

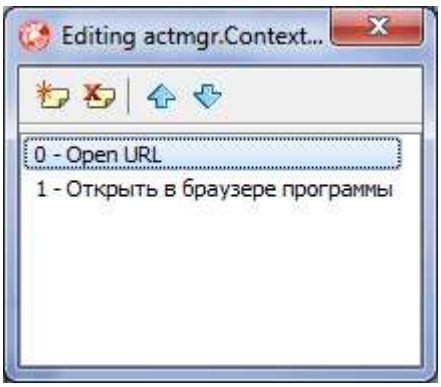
1. Выбрать элемент управления на ленте
2. Внести в список **ContextItems** элемента управления любое количество действие, расположенных в **Action Manager**.

Для примера, создадим контекстное меню для элемента управления **RibbonComboBox1** (URL), который уже расположен на ленте нашего приложения. Выбираем в окне Structure элемент:

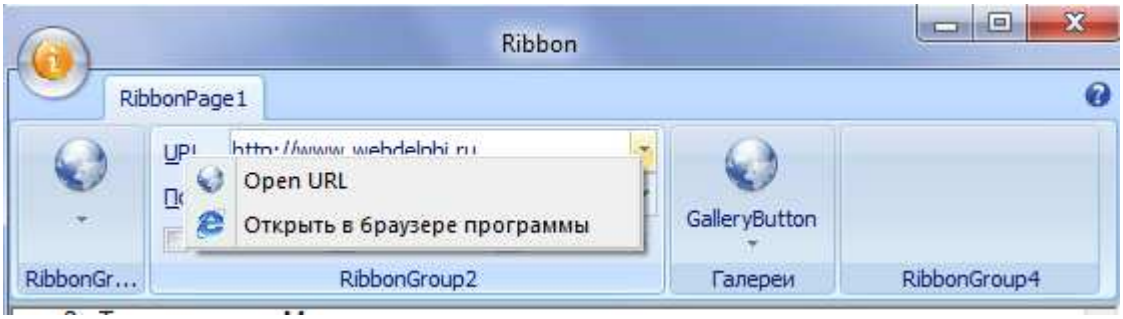
TActionManager→ActionBars→4-ActionBar->RibbonGroup2→Items→&URL



Переходим к его свойству **ContextItems** и в редакторе свойства добавляем несколько действий, как показано на рисунке:



Теперь запустите приложение и убедитесь, что элемент управления теперь имеет свое контекстное меню:



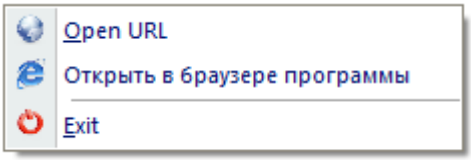
Pop-up меню в стиле Ribbon для стандартных элементов управления

Для того, чтобы использовать pop-up меню с темой оформления Ribbon для стандартных элементов управления, расположенных на форме, необходимо воспользоваться компонентом


TPopupActionBar

В отличие от стандартного компонента **TPopupMenu** компонент **TPopupActionBar** имеет свойство **Style**, которое используется для установки темы оформления меню.


*Для того, чтобы вставить в меню разделитель (горизонтальную линию) достаточно создать новый элемент меню и свойству **Caption** установить значение «-» (1 символ) после чего в меню появится разделительная горизонтальная линия:*



Подсказки в Ribbon Controls

Использование расширенных подсказок в Delphi

Рекомендации по использованию Enhanced tooltips

В Ribbon Controls появился новый тип подсказок - Enhanced tooltips (расширенные подсказки). В общем случае такая подсказка может содержать четыре блока:

Расширенные подсказки позволяют более детально объяснить пользователю назначение той или иной команды, а также, в случае необходимости, предоставить результат действия команды в виде небольшого поясняющего изображения.

Существует ряд рекомендации по использованию расширенных подсказок:

1. Не следует использовать в заголовках подсказок знаки пунктуации
2. Не следует делать слишком длинное описание подсказки, если команда является общеизвестной, например, вставка текста и т.д.
3. Описание команды должно начинаться с глагола или существительного. Не стоит использовать в описании, конструкции вида «Вставляет таблицу…», правильнее будет написать «Вставка таблицы…»
4. Для Split-кнопок желательно использовать две подсказки – по одной подсказке на каждую часть кнопки.

В Delphi для использования расширенных подсказок используются два компонента:

1. **TScreenTipsManager** – менеджер расширенных подсказок
2. **TScreenTipsPopup** – всплывающая расширенная подсказка

Работа с компонентом TScreenTipsManager

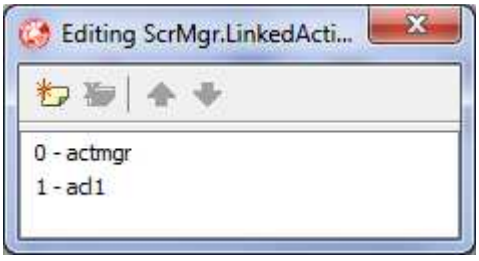
Для того, чтобы воспользоваться возможностями компонента **TScreenTipsManager** необходимо:

1. Разместить компонента на форме
2. Для компонента **TRibbon** в свойстве **ScreenTips** указать размещенный компонент менеджера подсказок
3. У **TScreenTipsManager** добавить в список **LinkedActionLists** все компоненты **Action Manager**, **Action List** и т.д. для действий которых необходимы расширенные подсказки.
4. Выбрать в контекстном меню компонента **TScreenTipsManager** пункт «Generate ScreenTips»
5. Отредактировать внешний вид и содержание подсказок в редакторе.
6. Установить у главной формы приложения свойство **ShowHint** в **True**.

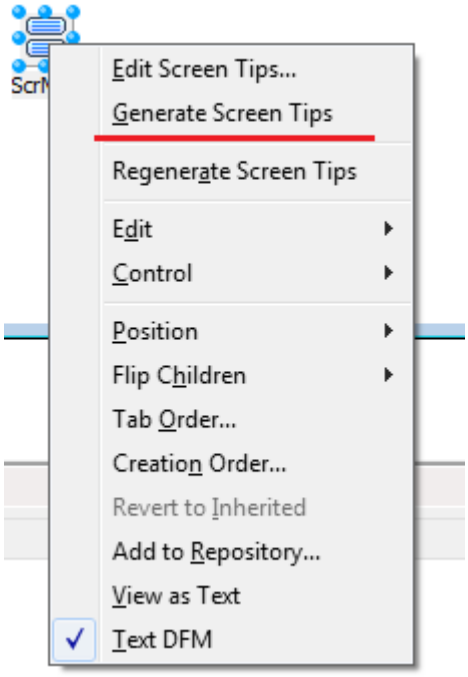
Рассмотрим процесс создания расширенных подсказок на примере нашего приложения.

Итак, разместите на форме компонент **TScreenTipsManager** и измените его свойство Name на «**ScrMgr**».

Теперь перейдите в редактор его свойства **LinkedActionLists** и добавьте в список, имеющиеся на форме **Action List** (**acl1**) и **Action Manager** (**actmgr**):

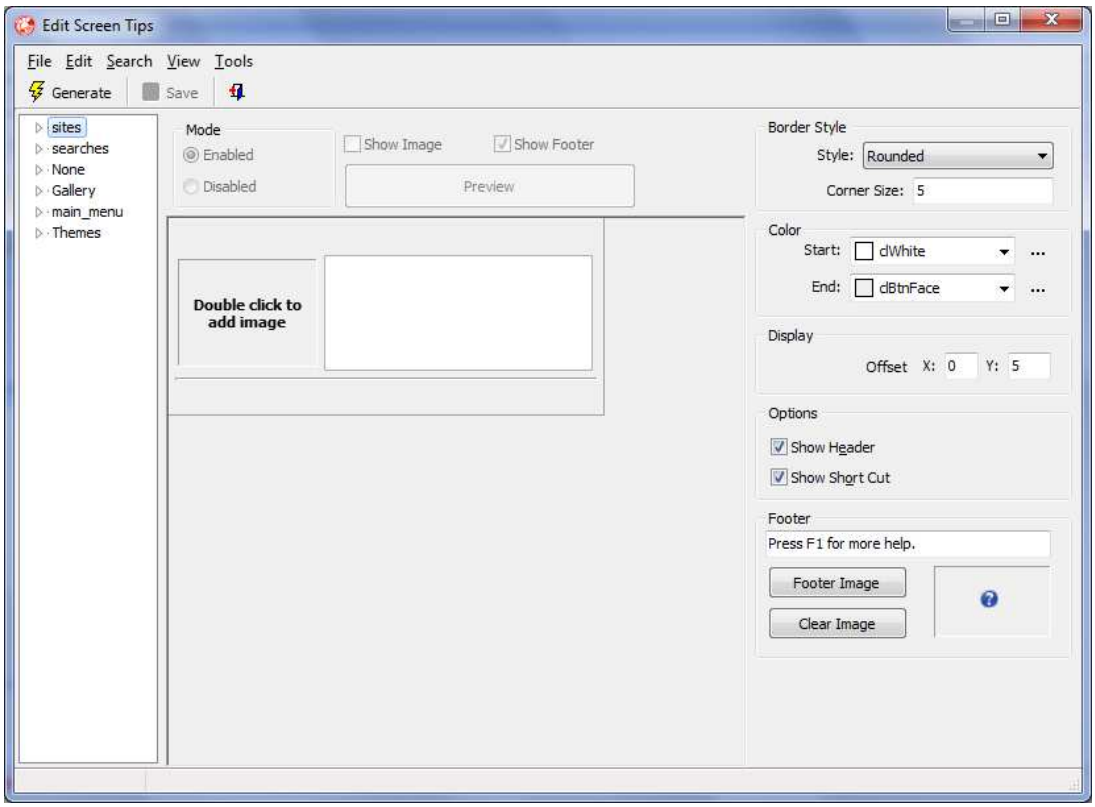


Выберите в компонент **TRibbon** и в его свойстве **ScreenTips** укажите «**ScrMgr**». Теперь кликните правой кнопкой мыши по компоненту **ScrMgr** и в меню выберите «**Generate ScreenTips**»:



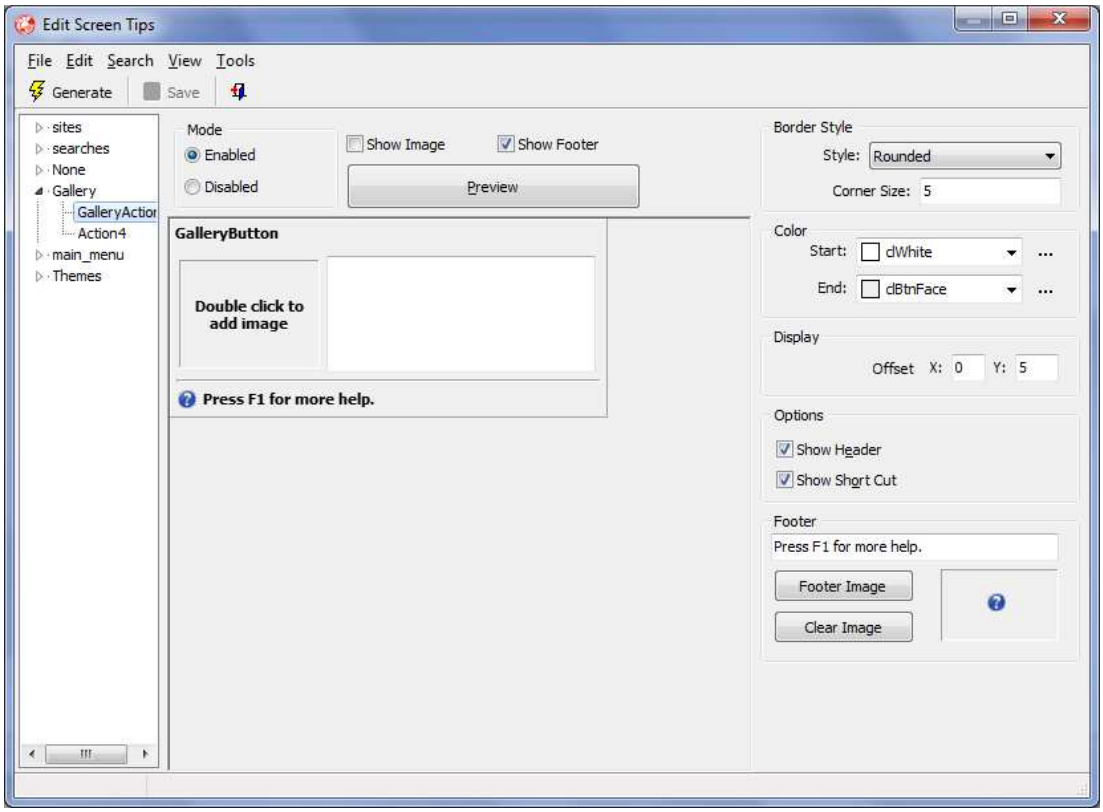
Если все сделано правильно, то Вы увидите на экране сообщение, свидетельствующее о том, что все подсказки были созданы:

Теперь снова вызовите контекстное меню компонента и выберите пункт «*Edit Screen Tips…*» перед Вами откроется окно редактора расширенных подсказок:



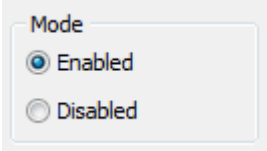
Воспользуемся редактором и создадим подсказку для галереи с темами оформления Ribbon (Кнопка **GalleryButton**).

Для этого в списке слева на находим и выбираем действие «**GalleryAction**». В редакторе появится вся информация, которая была сгенерирована автоматически:



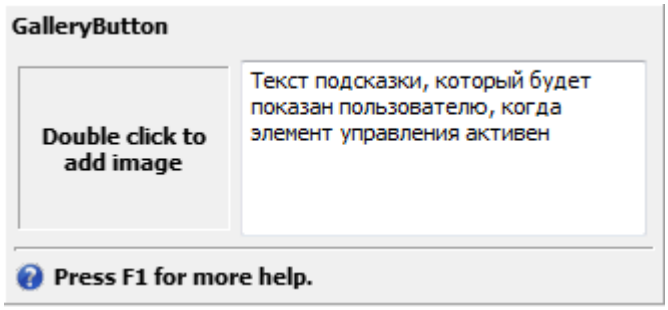
Рассмотрим все элементы редактора по мере создания новой подсказки.

Блок выбора режима расширенной подсказки



Все расширенные подсказки могут создаваться для двух режимов элемента управления. **Mode=Enabled** используется для создания расширенной подсказки, которая будет показана пользователю, когда элемент управления активен.

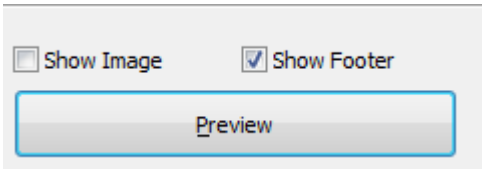
Редактор текста подсказки



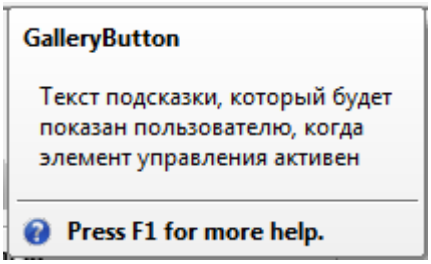
Здесь можно отредактировать текст, который будет показан пользователю, а также загрузить изображение для подсказки.

Для изображений в подсказках используются только bmp-файлы

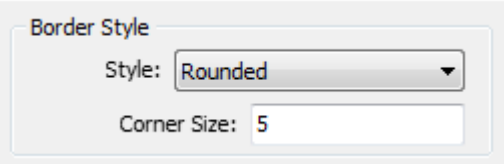
Блок настройки частей расширенной подсказки



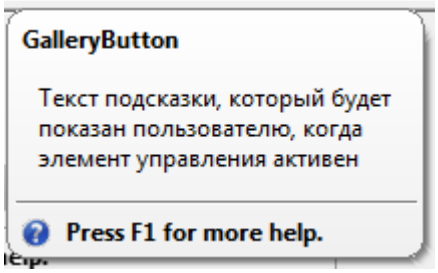
Здесь можно определить какие элементы расширенной подсказки будут задействованы в запущенном приложении, а также сделать предпросмотр подсказки (для этого необходимо навести указатель мыши на кнопку «*Preview*»). В случае, показанном на рисунке, расширенная подсказка будет показана без изображения, т.е. иметь следующий вид:



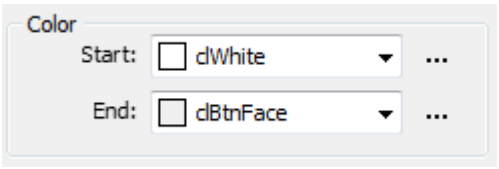
Блок настройки углов подсказки



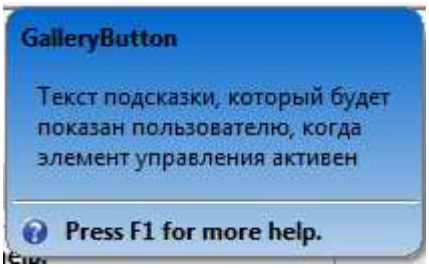
Подсказка может иметь скругленные углы (**Style** = **Rounded**) и прямые (**Style** = **Normal**). Если выбран показ подсказки со скругленными углами, то в параметре «**Corner Size**» указывается радиус скругления углов. Например, на рисунке ниже, показана подсказка с радиусом скругления углов 20:



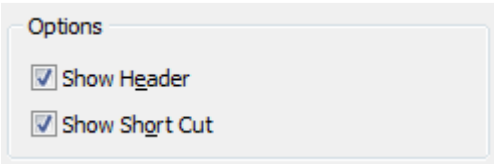
Блок настройки цвета подсказки



Расширенные подсказки имеют градиентную заливку, поэтому необходимо указывать начальный и конечный цвет для градиента. Манипулируя этими свойствами, можно создать расширенную подсказку, например, с такой заливкой:

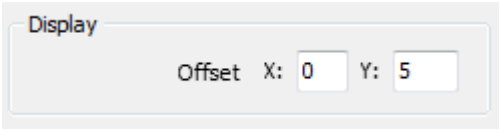


Блок опций расширенной подсказки



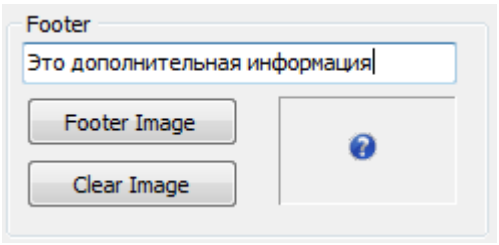
Здесь указывается будет ли расширенная подсказка иметь свой заголовок и будет ли в заголовке указываться комбинация горячих клавиш для выполнения команды (если она определена в настройках для **TAction**).

Блок настройки отступов



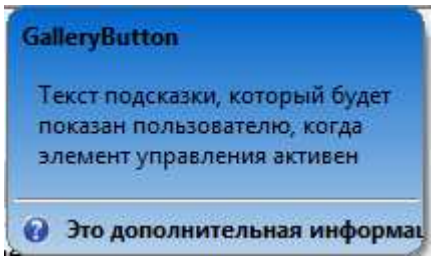
Свойства **Offset.X** и **Offset.Y** используются для указания отступа верхней границы подсказки от нижнего левого угла элемента управления.

Блок настройки подвала (нижней части) подсказки



Наиболее часто подвал используется для указания информации о том, какая клавиша или комбинация клавиш должна быть нажата пользователем для показа более детальной информации о команде. Также Вы можете загрузить дополнительное изображение (16x16 пикселей)

В результате рассмотрения частей редактора расширенных подсказок у нас получилась такая подсказка:



Теперь установите у главной формы приложения свойство

ShowHint = True

запустите приложение и убедитесь, что расширенная подсказка отображается при наведении курсора мыши на кнопку «**Gallery Button**»:

*Для того, чтобы упростить себе задачу редактирования и настройки расширенных подсказок при создании новых элементов управления старайтесь давать корректное описание команд в их свойствах **Hint** – именно это свойство используется для создания первоначального текста расширенной подсказки.*

Работа с компонентом TScreenTipsPopup

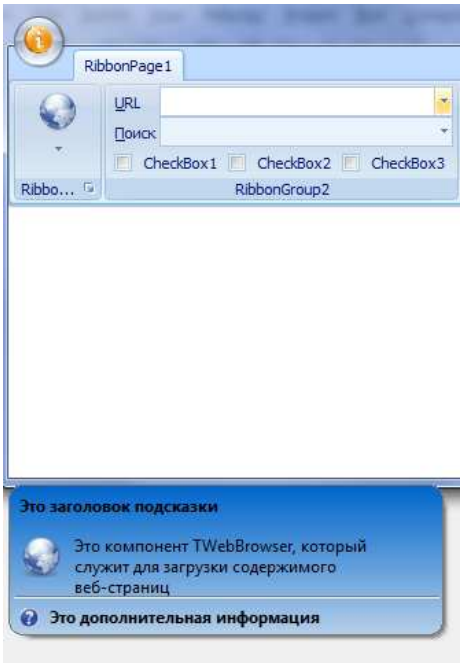
TScreenTipsPopup в общем случае имеет смысл использовать тогда, когда необходимо отобразить красивую расширенную подсказку для какого-либо стандартного компонента на форме, либо, когда необходимо выделить на форме элемент расширенной подсказки.

Рассмотрим процесс работы с компонентом **TScreenTipsPopup** на примере нашего приложения – создадим расширенную подсказку для компонента **TWebBrowser**.

Разместите на форме приложения компонент **TScreenTipsPopup** и установите для него следующие свойства:

Имя свойства	*Значение*
Associate	**WebBrowser1**
ScreenTipManager	**ScrMgr (TScreenTipsManager)**
ScreenTip→Description	**«Любая строка, которая будет служить описанием»**
ScreenTip→Header	**«Любая строка, которая будет служить заголовком»**
ScreenTip→Image	**Загрузите изображение для подсказки**
ScreenTip→ShowHeader	**True**
ScreenTip→ShowImage	**true**
Visible	**False**

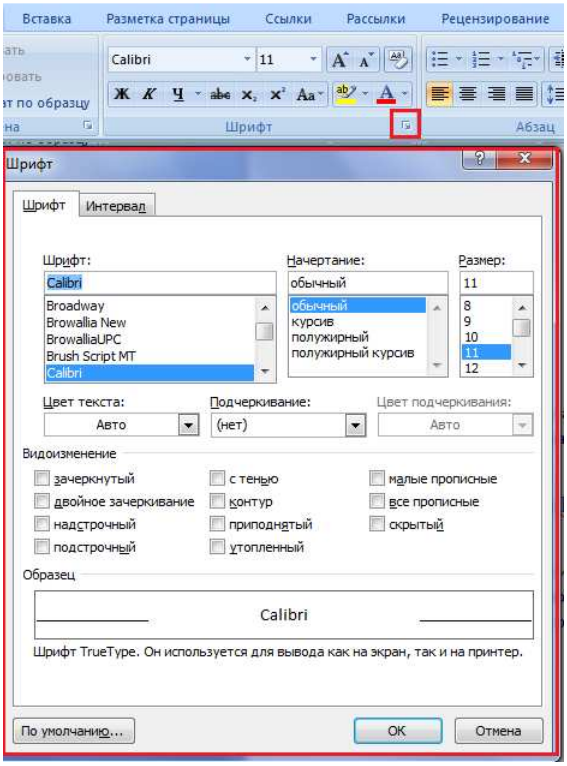
Теперь запустите приложение и наведите курсор мыши на **WebBrowser**. Т.к. мы использовали настройки **ScreenTipsManager** сделанные в предыдущем пункте, то в итоге получилась подсказка как показано на рисунке:



Обратите внимание на то, что как бы вы не перемещали курсор – подсказка всегда появляется в месте, которое определено в свойствах **Display →Offset** у **ScreenTipsManager**.

Работа с диалогами в Ribbon

Для запуска диалогов в Ribbon используются специальные элементы управления «*Dialog box launchers*», которые располагаются в правой части заголовка группы. Например, в MS Word кнопка запуска диалога в группе «*Шрифт*» откроет диалог настройки шрифтов:



Чтобы использовать кнопки запуска диалогов в своем приложении Вам необходимо:

1. Вы брать группу Ribbon
2. В свойстве **DialogAction** указать действие, которое должно выполнятся после клика по кнопке.

Если Вы планируете использовать стандартные диалоги, например, настройки шрифта или печати, то Вам нет никакой необходимости создавать новые действия и обработчики событий – просто выберите необходимое действие из раздела ***New Standard Actions →Dialogs****.*

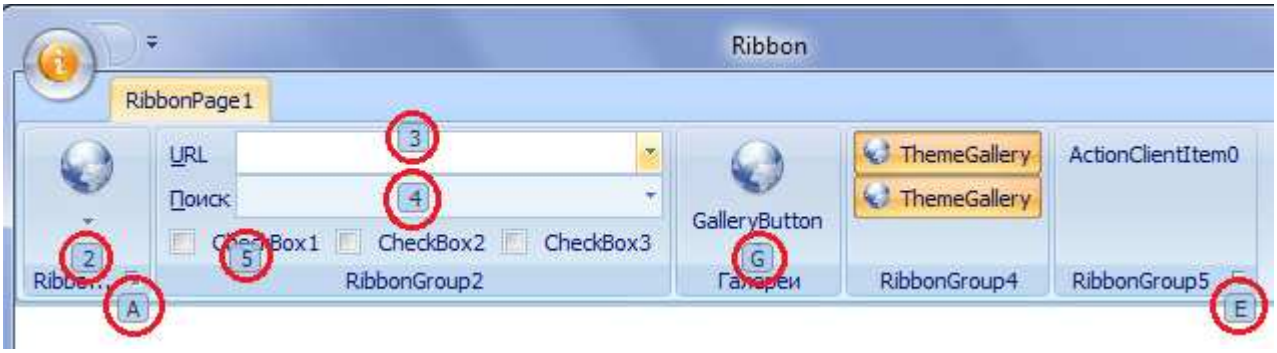
Настройка клавиш быстрого доступа к элементам Ribbon

Для обеспечения легко доступа к любому элементу ленты в Ribbon был введен новый тип горячих клавиш – **KetTips**. Это свойство имеется у всех элементов ленты, включая вкладки и группы. Назначение **KeyTips** для элементов ленты рекомендуется производить следующим образом:

- Для кнопки главного меню приложения назначается клавиша **F**
- Для элементов управления, расположенных на панели быстрого доступа рекомендуется назначать числовые клавиши **1, 2, 3** и т.д.
- Для вкладок: **H** – для домашней вкладки, для всех остальных рекомендуется назначать клавиши, соответствующие первой букве в названии, начиная с самых часто используемых вкладок.
- Для команд также рекомендуется использовать буквенные клавиши. При этом **KeyTips** назначаются в начале для часто используемых команд по первой букве команды, затем для менее используемых команд – по первым буквам, либо использовать доступные согласные, для самых редко используемых команд и кнопок запуска диалогов можно использовать комбинации из двух символов, если такие клавиши Вам необходимы в приложении.

KeyTips *необходимо назначать в сквозном порядке для всех элементов ленты, начиная с вкладок. Если вкладке не будет назначен свой* ***KeyTip*** *– Вы не сможете, используя только клавиатуру, добраться до групп и элементов управления на этой вкладке.*

Что бы воспользоваться возможностями быстрого доступа к элементам ленты Ribbon достаточно нажать на клавиатуре клавишу **[Alt]** и на ленте отобразятся все клавиши доступа в виде всплывающих подсказок:



Доступ к элементу управления пользователем осуществляется в следующей последовательности:

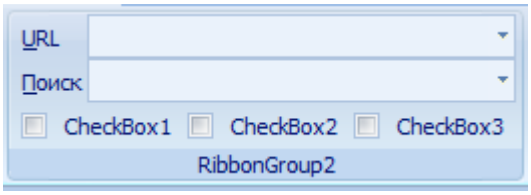
- Пользователь нажимает клавишу **[Alt]** – на ленте появляются **KeyTips** для первого уровня доступа: главное меню, панель быстрого доступа и вкладки.
- Пользователь нажимает необходимую клавишу, например, клавишу доступа к первой вкладке – открывается вкладка и на ней появляются **KeyTips** для диалогов и команд.
- Пользователь нажимает необходимую клавишу и выполняется запрошенная команда.

Выравнивание элементов управления в группах Ribbon

Для выравнивание элементов управления в группах есть два способа, определенных в свойстве **TRibbonGroup→GroupAlign**.

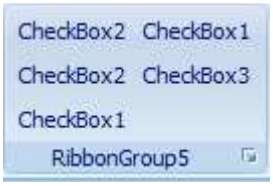
Значение gaHorizontal – означает, что элементы управления будут выравниваться построчно слева на право. Строка считается завершенной в том случае, когда элемент управления имеет свойство **NewRow** = **True**. При этом свойство **NewCol** игнорируется.

Пример такого способа выравнивания представлен на рисунке ниже:



Два верхних элемента располагаются по 1 в строке, нижние **3 CheckBox** выровнены горизонтально.

Значение gaVertical– означает, что элементы управления будут выравниваться сверху вниз по столбцам. Максимальное количество элементов в столбце **3**. Это выравнивание применяется в группе по умолчанию. Результат вертикального выравнивания элементов управления представлен на рисунке:



Список использованных источников информации

1. Официальный сайт MSDN http://msdn.microsoft.com/en-us/library/cc872782.aspx
2. Документация Embarcadero по RAD Studio 2010 http://docs.embarcadero.com/products/rad_studio/
3. Программная документация CodeGear RAD Studio 2009 http://docs.codegear.com/products/rad_studio/delphiAndcpp2009/HelpUpdate2/EN/html/delphivclwin32/idx.html
4. Блог WebDelphi.ru http://www.webdelphi.ru