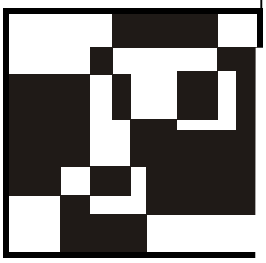
# ГЛАВА 8



# Работа с панелями инструментов и меню

Очень часто в приложении VBA вам потребуются свои наборы меню и панелей инструментов вместо стандартных, предусмотренных приложением. Работу с меню и панелями инструментов обеспечивает коллекция `CommandBars`, которая находится в объекте `Application` (об этом важном объекте мы будем говорить в следующих главах). Коллекция `CommandBars` содержит, как ясно из названия, набор объектов `CommandBar` (этот объект представляет как панели инструментов, так и меню), каждый из которых, в свою очередь, — коллекцию `CommandBarControls`, а эта коллекция представляет из себя хранилище элементов, из которых и состоит меню. Таких элементов может быть три:

- `CommandBarButton` — кнопка или элемент меню, который используется для выполнения программы или подпрограммы;
- `CommandBarComboBox` — сложный элемент меню или панели управления (это может быть поле ввода, раскрывающийся список, поле со списком);
- `CommandBarPopup` — меню или вложенное меню.

Пример создания собственной панели инструментов может выглядеть очень просто:

```
Dim CBar1 As CommandBar
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
CBar1.Enabled = True
CBar1.Visible = True
```

У нас появилась новая панель инструментов **Документы** (рис. 8.1), которую можно, к примеру, убрать через меню **Настройка | Панели инструментов**, однако пока она совершенно бесполезна: в ней нет ни одной кнопки. Для того чтобы на панели инструментов была кнопка, необходимо добавить новый элемент типа `CommandBarControl` (одного из трех типов, перечисленных ранее) в коллекцию `CommandBarControls` для этого меню.

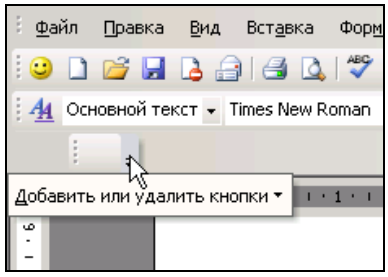



**Рис. 8.1.** Пустая панель инструментов **Документы**

Но вначале расскажем чуть подробнее о методе `Add()` коллекции `CommandBars`, при помощи которого мы создали новую панель инструментов. В нашем примере вызов этого метода выглядел так:

```
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
```

Первый параметр (`"Документы"`) — имя объекта (название панели). Второй параметр (`msoBarTop`) — либо положение пристыкованной панели (`msoBarTop`, `msoBarBottom`, `msoBarLeft`, `msoBarRight`), либо знак того, что панель не пристыкована (`msoBarFloating`), либо вообще указание на то, что это контекстное меню, которое до щелчка правой кнопкой мыши не видно (`msoBarPopup`). Свойство `Visible` для контекстного меню неприменимо.

Как будет выглядеть в итоге панель — как меню или как панель инструментов — зависит от того, какие элементы вы туда поместите.

Рассмотрим некоторые важные свойства и методы объекта `CommandBar`.

- `BuiltIn` — это свойство определяет, является ли данная панель/меню встроенной для этого приложения (т. е. предусмотренной в нем разработчиками приложения Office). Менять значение этого свойства нельзя. Его очень удобно использовать для того, чтобы убрать все стандартные меню или, наоборот, убрать все свои меню, оставив только стандартные.
- `Context` — определяет, где именно находится программный код для вашего меню (в Normal.dot, файле документа и т. п.). Можно использовать для проверок в случае потенциальной возможности вызова разных меню с одинаковыми именами.
- `Controls` — через это свойство можно получить коллекцию элементов управления `CommandBarControls`, которая нам, скорее всего, потребуется для работы с кнопками или элементами меню.
- `Enabled` — включение или отключение панели.
- `Height`, `Left`, `Top` и `Width` — очевидные свойства, относящиеся к расположению панели в окне приложения.

- `Index`, `Name` и `NameLocal` — эти свойства позволяют найти нужную нам панель в коллекции `CommandBars`. `Name` — это программное имя объекта, `NameLocal` — имя, которое будет видно пользователю, `Index` — номер данной панели в коллекции.

**Примечание**

Просмотреть номера всех встроенных панелей и меню в Office можно при помощи кода:

```
Dim cBar As CommandBar
For Each cBar In CommandBars
    Debug.Print cBar.Index & vbTab & cBar.Name
Next
```

Если вы хотите поместить новый элемент в стандартное меню Office, то номер стандартного меню в коллекции `CommandBars` — 41.

- `Protection` — позволяет запретить пользователю убирать старые кнопки из этой панели или размещать на ней новые.
- `Type` — наверное, самое важное свойство. Определяет, чем будет данная панель (панелью инструментов, обычным меню или контекстным меню, открывающимся по щелчку правой кнопкой мыши). Однако это свойство доступно только для чтения.
- `Visible` — это свойство определяет, будет ли панель инструментов видимой. Для вновь созданных панелей инструментов по умолчанию используется значение `False`, т. е. панель инструментов не видна.

После того как создание панели завершено, необходимо разместить на ней элементы управления. Например, создание кнопки на панели инструментов может выглядеть так:

```
Dim But1 As CommandBarControl
Set But1 = CBar1.Controls.Add(msoControlButton)
But1.Caption = "Кнопка 1"
```

Вроде бы ничего не произошло — перед нами та же пустая панель. Однако если навести указатель мыши на начало панели, можно увидеть пустую кнопку, которая называется **Кнопка 1** (рис. 8.2).

Как видно из кода, мы использовали для создания элемента управления (кнопки в панели инструментов) метод `Add()` коллекции `Controls` объекта `CommandBar` (у нас он называется `CBar1`). Свойства и методы у этой коллекции стандартные, как у множества других коллекций.

- `Application` — возвращает ссылку на объект приложения (Word, Excel и т. п.).

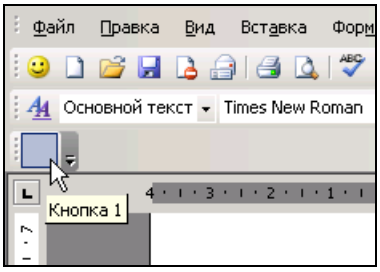



**Рис. 8.2.** На панели инструментов **Документы** появилась почти невидимая **Кнопка 1**

- `Count` — позволяет узнать, сколько всего элементов управления помещено в эту коллекцию.
- `Item` — позволяет по индексу (номеру) получить ссылку на объект элемента управления `CommandBarControl` в этой коллекции.
- `Add()` — позволяет добавить элемент управления в эту коллекцию (удаление элемента управления производится при помощи метода `Delete()` самого элемента управления).

Рассмотрим подробнее метод `Add()` коллекции `Controls`. Этот метод принимает пять необязательных параметров, из которых первые два параметра очень важны. Первый параметр `Type` определяет тип передаваемого в коллекцию элемента управления. Таких типов пять:

- `msoControlButton` — кнопка (т. е. в итоге получится панель инструментов);
- `msoControlEdit` — поле для ввода текста;
- `msoControlDropdown` — раскрывающийся список;
- `msoControlComboBox` — комбинированный список;
- `msoControlPopup` — пункт меню.

Например, чтобы вместо нашей кнопки был создан начальный пункт раскрывающегося меню (рис. 8.3), в нашем коде нужно изменить параметр метода `Add()`:

```
Dim But1 As CommandBarControl
Set But1 = CBar1.Controls.Add(msoControlPopup)
But1.Caption = "Меню 1"
```

Второй важный параметр метода `Add()` — параметр `Id`. Этот параметр позволяет привязать создаваемый элемент к уже имеющемуся в системе встроенному элементу управления. Например, чтобы добавить кнопку печати, этот параметр должен быть равен 4, а чтобы добавить кнопку предварительного просмотра документа, этот параметр должен быть равен 5. Если оставить этот

параметр пустым или указать для него значение 1, то будет создан пользовательский элемент управления, не привязанный ни к каким встроенным. К сожалению, значения этого параметра для встроенных элементов управления никак не документированы. Определить нужное значение можно или подбором, или просмотром значений свойства `Id` для имеющихся элементов управления (например, при помощи окна **Locals**).

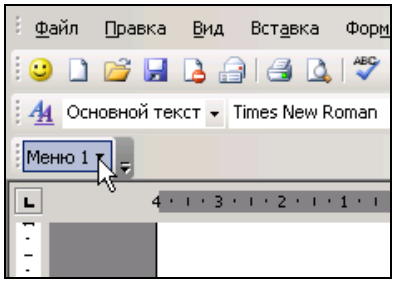



**Рис. 8.3.** Пункт меню на созданной нами панели инструментов

Остальные параметры этого метода относятся к созданию идентификатора элемента управления, его положению относительно других элементов управления и к тому, должен ли он быть постоянным или временным.

Конечно, работа с панелью управления/меню на этом не завершается. Нам потребуется еще донастроить свойства элементов управления. Например, для кнопки из первого примера нам, как минимум, нужно определить изображение (иконку), которое будет на кнопке, и процедуру, которая будет вызываться при нажатии на нее.

Важнейшие свойства и методы объекта `CommandBarButton` приводятся далее.

- `Caption` — надпись на элементе управления. Для кнопки выводится в виде всплывающей подсказки, а для пункта меню — название пункта.
- `Enabled` — определяет, включен или отключен данный элемент управления. Обычно используется для предупреждения ошибок пользователя.
- `FaceId` (только для кнопок) — позволяет использовать системную картинку для кнопки (не назначая ей соответствующей функции). Например, значение 4 присвоит кнопке изображение принтера. В Word и Excel встроено несколько тысяч иконок, и поэтому вместо создания новой иконки всегда есть возможность подобрать готовую. К сожалению, номера иконок тоже нигде не документированы, и вам придется выбирать подходящую методом перебора. Можно также создать для целей выбора иконки панели инструментов с несколькими сотнями кнопок, каждой из которых будет присвоено значение `FaceId`, увеличенное на 1 по отношению к предыдущей.

- `Id` — идентификатор встроенной функции, назначенной этой кнопке. Если вы назначили этой кнопке пользовательскую процедуру, то значение `Id` будет всегда равно 1.
- `Index` — номер элемента управления в коллекции `Controls`. Используется для выполнения служебных операций с элементами управления.
- `Mask` — позволяет наложить на рисунок объекта маску для показа только части рисунка. Маска выглядит как отдельный рисунок, прозрачные части которого должны быть белыми, а непрозрачные — черными.
- `OnAction` — самое важное свойство элемента управления. Его значение используется, когда элемент управления активизируется (щелчок по кнопке или пункту меню, завершение ввода текста в текстовом поле, выбор нового значения в списке). Предназначено для указания запускаемой процедуры или внешнего приложения (COM Add In). Наример:

  ```
  But1.OnAction = "MySub"
  ```

- `Parameter` — это свойство можно использовать для передачи параметров вызываемой подпроцедуре или просто для хранения своих данных. Работает с типом данных `String`.
- `Picture` — это свойство позволяет назначить рисунок (иконку) кнопке панели инструментов. Чаще используется не оно, а свойство `FaceId`. Если есть необходимость, то нужные иконки можно подобрать из большой коллекции, которая есть в Visual Studio, или воспользоваться одним из свободно доступных генераторов иконок.
- `ShortcutText` — текст с описанием клавиатурного сочетания, назначенного этой кнопке или пункту меню.
- `State` — вид кнопки (обычная, утопленная или обведенная рамкой).
- `Style` — еще одно свойство, влияющее на внешний вид. Позволяет определить, как будет выглядеть кнопка: будет показана только иконка, только надпись или и то, и другое вместе в разных вариантах.
- `ToolTip` — определяет текст всплывающей подсказки. По умолчанию "всплывает" значение свойства `Caption`.
- `Type` — возвращает тип элемента управления (только для чтения).
- `Visible` — определяет, будет этот элемент управления видимым или нет.
- `Delete()` — этот метод позволяет удалить кнопку из коллекции кнопок;
- `Execute()` — запускает на выполнение то, что определено при помощи свойства `OnAction`;
- `Reset()` — возвращает к исходным параметрам кнопки после внесенных изменений.

У объекта `CommandBarButton` есть единственное событие — `Click`. Оно также позволяет определить реакцию на нажатие кнопки, но работать с ним сложнее, чем со свойством `OnAction`.

В принципе, для работы с панелями инструментов этого вполне достаточно. Однако раскрывающиеся и контекстные меню устроены несколько сложнее. В раскрывающемся меню вам потребуется еще определить вложения элементов, а в контекстном меню — привязать это меню к какому-то объекту.

Для работы с вложенными меню используется точно та же коллекция `Controls`, которой мы уже пользовались. Единственное отличие в том, что эта коллекция `Controls` принадлежит не объекту `CommandBar`, а объекту `CommandBarPopup`, т. е. другому меню. В нашем примере вложение будет выглядеть так:

```
'Создаем стандартный объект CommandBar
Dim CBar1 As CommandBar
Set CBar1 = CommandBars.Add("Документы", msoBarTop)
CBar1.Enabled = True
CBar1.Visible = True

Dim Menu1 As CommandBarPopup
Dim SubMenu1 As CommandBarPopup
Dim SubMenu1Item As CommandBarButton

'Создаем верхнее меню
Set Menu1 = CBar1.Controls.Add(msoControlPopup)
Menu1.Caption = "Меню 1"

'Создаем вложенное меню
Set SubMenu1 = Menu1.Controls.Add(msoControlPopup)
SubMenu1.Caption = "Подменю 1"

'Создаем элемент во вложенном подменю и назначаем ему процедуру Proc1
Set SubMenu1Item = SubMenu1.Controls.Add(msoControlButton)
SubMenu1Item.FaceId = 5
SubMenu1Item.Caption = "Элемент подменю"
SubMenu1Item.OnAction = "Proc1"
```

У вас должно получиться то, что изображено на рис. 8.4.

Конечно, можно добавлять элементы не только в свои меню, но и во встроенные. Добавление происходит точно так же, а найти нужное встроенное меню можно при помощи цикла `For Each` и проверки значения свойства `Name`.

Контекстные меню (в справке VBA *shortcut menus*) — это меню, которые открываются по щелчку правой кнопкой мыши. Работа с ними выглядит так:

```
Set CBar1 = CommandBars.Add("Мое контекстное меню", msoBarPopup, , _
            True)
Set MenuItem1 = CBar1.Controls.Add
MenuItem1.FaceId = 3
MenuItem1.Caption = "Элемент меню 1"
Set MenuItem2 = CBar1.Controls.Add
MenuItem2.FaceId = 5
MenuItem2.Caption = "Элемент меню 2"
```

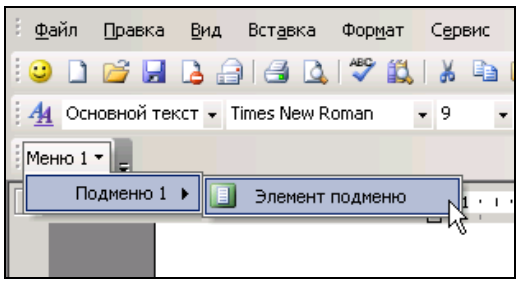



**Рис. 8.4.** Пример вложенного меню

Как мы видим, все очень просто и стандартно. Однако если выполнить этот код, то никакого контекстного меню не появится: необходимо еще добавить вызов метода `ShowPopup()`:

```
CBar1.ShowPopup
```

Тогда контекстное меню возникнет в том месте, где сейчас находится указатель мыши (можно передать этому методу координаты места появления). Конечно, этот метод нужно вызывать из обработчика события, связанного с правой кнопкой мыши, а его как раз и нет для многих объектов. Например, в Excel для листа есть событие `BeforeRightClick`, а для документа Word такого события нет. Но в этом случае у нас останется возможность добавить свои пункты в стандартное контекстное меню.

# Задание для самостоятельной работы 8: Работа с панелями инструментов, меню и помощником

**ЗАДАНИЕ:**

1. Создайте и сделайте видимой в Word новую панель инструментов с двумя кнопками: **Показать помощника** и **Отключить помощника**. Она может выглядеть, например, так, как представлено на рис. 8.5.

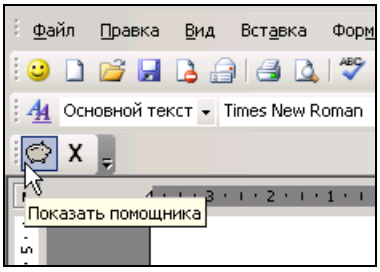



**Рис. 8.5.** Новая панель инструментов Word

При нажатии на кнопку **Показать помощника** должен появляться помощник, а при нажатии на кнопку **Отключить помощника** он должен отключаться.

2. Сделайте так, чтобы при нажатии на кнопку **Показать помощника** появлялось дополнительное меню, аналогичное представленному на рис. 8.6.

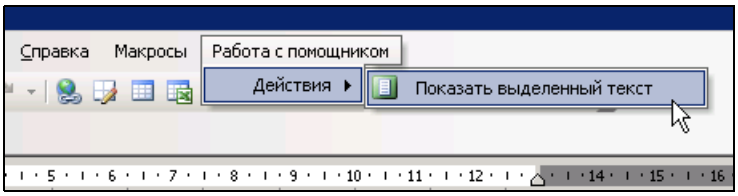



**Рис. 8.6.** Созданное программным образом новое меню

При выборе пункта меню **Показать выделенный текст** в окне помощника должен отображаться текст, который выделен в документе (рис. 8.7)


**Рис. 8.7.** Вызванный макросом помощник в действии

## Ответ к заданию 8

Набор процедур, которые бы выполняли поставленные условия, может выглядеть так, как представлено далее (для создания панели инструментов нуж-

но запустить процедуру `ShowNewToolbar()`). Если возможности этих процедур нужны во всех документах Word, то лучше их поместить в модуль проекта Normal.dot.

```
Public Sub ShowNewToolbar()
'Исходная процедура, которая создает новую панель инструментов Assistant

    'Проверка — если панель с таким именем уже есть, создавать ее не надо
    Dim cBar As CommandBar
    For Each cBar In CommandBars
        If cBar.Name = "Assistant" Then
            cBar.Visible = True
            Exit Sub
        End If
    Next

    'Создаем панель управления
    Dim CBar1 As CommandBar
    Set CBar1 = CommandBars.Add("Assistant", msoBarTop)
    CBar1.Enabled = True
    CBar1.Visible = True

    'Помещаем на нее первую кнопку
    Dim But1 As CommandBarControl
    Set But1 = CBar1.Controls.Add(msoControlButton)
    But1.Caption = "Показать помощника"
    But1.FaceId = 52
    But1.OnAction = "ShowAssistant"

    'Помещаем на нее вторую кнопку
    Dim But2 As CommandBarControl
    Set But2 = CBar1.Controls.Add(msoControlButton)
    But2.Caption = "Отключить помощника"
    But2.FaceId = 103
    But2.OnAction = "DisableAssistant"
End Sub

Public Sub ShowAssistant()
'Процедура, которая запускается по нажатию кнопки "Показать помощника"

    'Включение помощника
    Assistant.On = True
    Assistant.Visible = True

    'Начинается код для создания нового меню

    'Проверяем, есть ли уже такой пункт в стандартном меню
    Dim cBarCont As CommandBarControl
```

```
    For Each cBarCont In CommandBars(41).Controls
        If cBarCont.Caption = "Работа с помощником" Then
            Exit Sub
        End If
    Next

    'Создаем меню
    Dim CBar1 As CommandBar
    Set CBar1 = CommandBars(41)
    CBar1.Enabled = True
    CBar1.Visible = True

    Dim Menu1 As CommandBarPopup
    Dim SubMenu1 As CommandBarPopup
    Dim SubMenu1Item As CommandBarButton

    'Создаем верхнее меню
    Set Menu1 = CBar1.Controls.Add(msoControlPopup)
    Menu1.Caption = "Работа с помощником"

    'Создаем вложенное меню
    Set SubMenu1 = Menu1.Controls.Add(msoControlPopup)
    SubMenu1.Caption = "Действия"

    'Создаем элемент во вложенном подменю и назначаем ему
    'процедуру TextToAssistant
    Set SubMenu1Item = SubMenu1.Controls.Add(msoControlButton)
    SubMenu1Item.FaceId = 5
    SubMenu1Item.Caption = "Показать выделенный текст"
    SubMenu1Item.OnAction = "TextToAssistant"
End Sub

Public Sub DisableAssistant()
'Процедура, которая отключает помощника и созданное меню

    'Этот код можно использовать для кнопки "Отключить помощника"
    Assistant.On = False

    Dim cBarCont As CommandBarControl
    For Each cBarCont In CommandBars(41).Controls
        If cBarCont.Caption = "Работа с помощником" Then
           cBarCont.Delete
            Exit Sub
        End If
    Next
End Sub
```

```
Public Sub TextToAssistant()
'Процедура, которая выводит в помощнике выделенный в документе текст

    Dim oBal1 As Balloon
    Set oBal1 = Assistant.NewBalloon
    oBal1.Text = Selection.Text
    oBal1.Show
End Sub
```