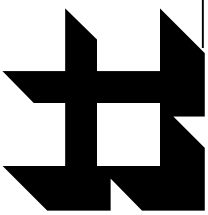

ГЛАВА 18

Компоненты ActiveX

При разработке приложений мы постоянно используем предоставляемые Visual FoxPro *базовые классы* для создания форм и различных управляющих элементов (Control). Но иногда возникает необходимость разместить на форме нечто такое, чего нет в базовых классах, например, управляющий элемент, отображающий список в виде дерева с узлами, которые можно открывать и закрывать.

Для решения этой проблемы и была разработана технология ActiveX, являющаяся одной из разновидностей (а некоторые считают — вершиной) технологии COM. Компоненты ActiveX, как и любые COM, представляют собой двоичные модули, которые можно использовать точно так же, как и управляющие элементы, созданные на основе базовых классов. Достоинство ActiveX — это простота и возможность применения в приложениях, написанных на любых языках, поддерживающих COM. В настоящее время разработано множество различных ActiveX, удовлетворяющих требованиям самых взыскательных пользователей. Как правило, они хранятся в файлах с расширением осх, хотя на самом деле это обычные DLL.

> ***Замечание***
>
> Некоторое время назад наблюдалась известная путаница в терминологии, связанной с описанием модели объектных компонентов. Но к настоящему времени сформировалась некоторая система, и теперь термином OLE обозначают те аспекты COM, которые выполняют функции "внедрения и связывания", а термином ActiveX, как правило, обозначают созданные по технологии COM управляющие элементы. Сам же термин COM применяется по отношению к COM-серверам (хотя любой COM является сервером).

Все ActiveX-компоненты, реализующие функции управляющих элементов, выполняются внутри создаваемого Visual FoxPro окна-контейнера, которое имеет свойство `HWnd`. Таким образом, вы можете посылать этому окну сообщения Windows, которые будут ретранслированы компоненту (подробнее о сообщениях Windows речь пойдет в *главе 23*).

В поставку Visual FoxPro включены компоненты ActiveX, разработанные и распространяемые Microsoft. Вы уже познакомились с одним из таких компонентов, `TreeView`, в первой части книги. В этой главе вы познакомитесь еще с некоторыми ActiveX, такими как `ProgressBar`, `Slider`, `RichText` и `Web Browser`.

Общие сведения об использовании компонентов

В этом разделе мы рассмотрим общие для всех ActiveX-компонентов моменты, связанные с особенностями их установки, лицензированием и интерактивной настройкой свойств. В следующих разделах описаны только те свойства и методы, которые характерны для рассматриваемого компонента.

Выбор и размещение компонента на форме

Компоненты ActiveX могут быть размещены на форме или в любом другом объекте — контейнере. Для выбора и размещения компонента необходимо выполнить следующие действия:

1. В Конструкторе форм (классов) создайте новую или откройте существующую форму.
2. В панели инструментов **Form Controls** щелкните мышью на кнопке **ActiveX Control (OleControl)**.
3. Выделите на форме область для размещения компонента или просто щелкните мышью в клиентской области окна.

На экране появится диалоговое окно **Insert Object**, содержащее список компонентов (рис. 18.1).

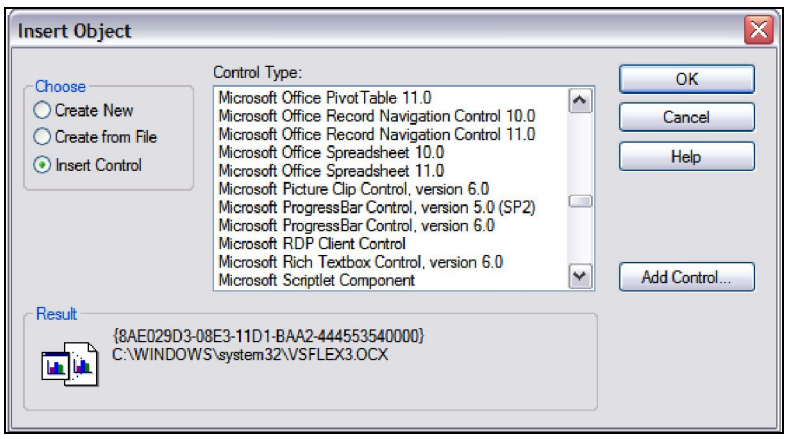

Рис. 18.1. Диалоговое окно **Insert Object**, содержащее список компонентов

Выберите в блоке **Choose** опцию **Insert Control**, затем найдите в списке **Control Type** наименование компонента. В области **Result** вы увидите ClsID компонента и имя файла, в котором этот компонент находится.

Щелкните на кнопке **OK** диалога. Компонент будет перенесен на форму.

На рис. 18.2 показано, как будет выглядеть ActiveX *Microsoft ProgressBar Control, version 5.0 (SP2)* после размещения на форме (рис. 18.2).

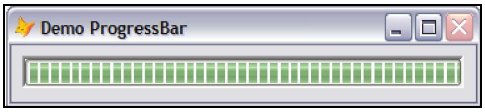

Рис. 18.2. Размещение на форме ActiveX-компонента **ProgressBar**

Замечание

По умолчанию для каждого добавляемого в форму компонента ActiveX Конструктор присваивает его свойству `Name` значение `OleControlN`, где "`N`" — порядковый номер этого элемента.

Когда в процессе конструирования вы делаете компонент активным, в окне **Properties** становятся доступными для редактирования все его свойства и методы (специфические для компонента свойства и методы выводятся синим цветом).

Вы можете так же включать ActiveX-компоненты и в классы, объявляемые в программных файлах (с расширением prg).

Лицензирование

Возможна ситуация, когда ActiveX "не хочет работать" из-за отсутствия лицензии. Попытка включить такой компонент в свое приложение без подтверждения права на его использование кончится неудачей.

Лицензирование — это один из способов защиты прав разработчиков программного обеспечения. Разработчик может продать вам приложение, использующее созданный им ActiveX, но при этом он не передает вам лицензионный файл этого компонента (с расширением lic). Так как компонент хранится во внешнем файле (с расширением ocx или dll), у вас может возникнуть искушение применить его в своих разработках. В соответствии с международными соглашениями о защите авторских прав, вы должны приобрести у разработчика этот ActiveX вместе с лицензионным файлом; а при распространении приобретенного компонента вы должно строго следовать требованиям лицензионного соглашения.

Интерактивная установка свойств компонента

Практически все созданные серьезными разработчиками ActiveX включают в себя специальное диалоговое окно, выполняющее функции интерактивной настройки свойств. Для вызова этого диалогового окна щелкните правой кнопкой мыши по области компонента и в появившемся меню выберите пункт ***Имя*Ctrl Properties**, где *Имя* — имя компонента. Например, для **ProgressBar** это пункт меню **ProgCtrl Properties** (рис. 18.3).

Окно **Свойства: ProgCtrl** для настройки свойств компонента **ProgressBar** показано на рис. 18.4.

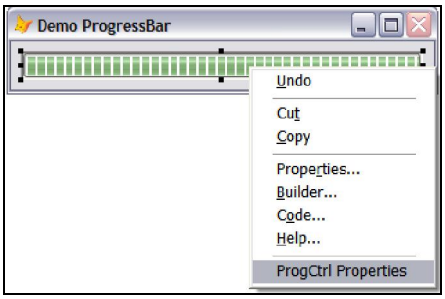

Рис. 18.3. Контекстное меню компонента

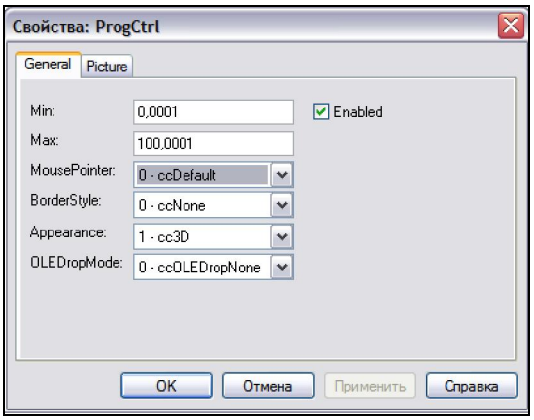

Рис. 18.4. Диалог для настройки свойств компонента **ProgressBar**

В этом окне вы можете установить минимальное и максимальное значения шкалы (поля **Min** и **Max**), определить вид указателя мыши в момент его перемещения над областью компонента (поле **MousePointer**), а также установить стиль бордюра (**BorderStyle**) и объемность (**Appearance**).

Компонент *ProgressBar*

Этот компонент реализует шкалу, динамически изменяющую свой размер. Вместе с Visual FoxPro поставляется две версии компонента.

Версия 5.0 находится в файле comctl32.ocx; ее особенностью является поддержка тем Windows XP. Имя компонента в окне **Insert Control**: *Microsoft ProgressBar Control, version 5.0 (SP2)*.

Версия 6.0 находится в файле mscomctrl.ocx. Версия не поддерживает тем Windows XP, но включает в себя дополнительные свойства. Имя компонента: *Microsoft ProgressBar Control, version 6.0*.

Компонент не имеет собственных методов; вы можете использовать методы окна-контейнера, в котором выполняется компонент. Это стандартные методы типа `Click`, `MouseDown` и т. д. Например, если вы поместите код в метод `Click`, то этот код будет выполняться, когда во время выполнения приложения вы щелкнете мышью по области компонента.

Свойства `Min` и `Max` позволяют установить начальное и конечное значение шкалы.

Свойство `Value` определяет длину шкалы; значение этого свойства должно находиться в интервале, определяемом свойствами `Min` и `Max`.

Свойства `Orientation` и `Scrolling` появились в шестой версии компонента. Вы можете расположить шкалу горизонтально, присвоив свойству `Orientation` значение 0 (используется по умолчанию), или вертикально, присвоив значение 1. Свойство `Scrolling` управляет видом полосы шкалы: стандартная (0) или сплошная (1).

Компонент **ProgressBar** удобно использовать для демонстрации хода выполнения длительного процесса, например, в циклах. При этом следует учитывать, что обращение к компоненту требует ресурсов, что может резко увеличить время выполнения процесса. Поэтому количество обращений к нему следует по возможности минимизировать.

В следующем фрагменте кода показано использование компонента для индикации процесса обработки записей в цикле `SCAN .. ENDSCAN` (предполагается, что имя компонента, записанное в свойстве `Name` — `Progress`).

```
SELECT mytable
nRecord = 0
nStep = 0.01 * RECCOUNT()
GO TOP
SCAN
   IF nRecord > nStep
      thisform.Progress.Value = RECNO()
      nRecord = 0
   ENDIF
   nRecord = nRecord + 1
* Здесь — код, выполняемый внутри цикла
ENDSCAN
```

Изменение состояния шкалы выполняется один раз на каждые сто просмотренных записей.

Компонент *Slider*

Компонент реализует шкалу, имеющую ползунок. Вы можете использовать этот компонент для интерактивного выбора значений в некотором заданном интервале.

Внешний вид ActiveX **Slider** показан на рис. 18.5.

Рис. 18.5. Внешний вид ActiveX-компонента **Slider**

Компонент так же поставляется в двух версиях.

Версия 5.0 находится в файле comctl32.ocx; ее особенностью является поддержка тем Windows XP. Имя компонента в окне **Insert Control**: *Microsoft Slider Control, version 5.0 (SP2)*.

Версия 6.0 находится в файле mscomctrl.ocx. Она не поддерживает тем Windows XP, но имеет дополнительные свойства. Имя компонента: *Microsoft Slider Control, version 6.0*.

Свойства

Свойства `Min` и `Max` определяют минимальное и максимальное значения шкалы.

Свойство `Value` определяет позицию ползунка на шкале.

Свойство `Orientation` управляет положением шкалы. Если значение свойства равно нулю, шкала расположена горизонтально, а если единице — то вертикально.

Свойство `SelectRange` изменяет вид шкалы; если значения свойства — "истина", то полоса шкалы становится равной высоте ползунка, как это показано на рис. 18.6.

Рис. 18.6. Вид компонента **Slider**, когда свойство `SelectRange = .T`

Свойство `TickStyle` определяет расположение отметок шкалы. Оно может принимать одно из следующих значений:

- 0 — отметки располагаются под шкалой (или справа при вертикальной ориентации);
- 1 — отметки располагаются над шкалой (или слева вертикальной ориентации);
- 2 — отметки располагаются по обеим сторонам шкалы; ползунок приобретает форму прямоугольника;
- 3 — отметки не выводятся.

Свойство `TickFrequency` определяет частоту появления отметок шкалы; количество отметок рассчитывается по формуле:

```
(Max – Min) / TickFrequency
```

где `Max`, `Min` и `TickFrequency` — значения свойств компонента.

Свойство `GetNumTicks`, доступное только для чтения, содержит количество выводимых отметок шкалы.

Компонент **Slider** шестой версии при нажатии мышью на ползунок шкалы выводит во всплывающем окне значение своего свойства `Value`. В нем появились два новых свойства: `Text` и `TextPosition`. Свойство `TextPosition` принимает значения 0 или 1,

определяющие, над или под ползунком выводится всплывающее окно. А в свойство `Text` вы можете ввести свой текст, который будет выводиться в этом окошке.

Методы

Метод `Change` обрабатывает событие `Change`, возникающее в момент отпускания мышью ползунка шкалы после его перемещения. Метод не имеет параметров.

Метод `Scroll` обрабатывает событие `Scroll`, возникающее в процессе перемещения ползунка шкалы. Метод также не имеет параметров.

Компонент *RichText*

Компонент подобен управляющему элементу **EditBox** Visual FoxPro и предназначен для отображения и редактирования текста в формате RTF. Находится он в файле richtx32.ocx.

Используя этот компонент, вы можете выводить в его окне не только форматированный текст (используя различные шрифты и цвета), но и изображения. Основным достоинством формата RTF является то, что созданные с его использованием документы можно хранить в memo-полях таблиц, недостатком — отсутствие поддержки национальных алфавитов, из-за чего символы кириллицы, например, записываются как esc-последовательности, предваряемые символами обратного слэша и амперсанда. Так, символ "А" русского алфавита в формате RTF будет записан как "\'c0".

Свойства

Рассмотрим основные свойства компонента в порядке их актуальности.

Свойство `FileName` позволяет указать имя файла, содержимое которого будет загружено в компонент в момент его инициализации (если вы выбираете файл в интерактивном режиме, то его содержимое становится видимым в окне компонента в процессе конструирования).

Свойство `MaxLength` позволяет ограничить максимальный размер текста, размещаемого в буфере компонента. Если значение свойства равно нулю, размер текста не ограничивается.

Свойство `MultiLine` (логического типа) определяет, как будет выводиться текст: как многострочный (`MiltiLine=.T.`) или в одной строке (`MiltiLine=.F.`).

Свойство `RightMargin` определяет размер области вывода текста по горизонтали (в пикселах). Если значение этого свойства больше, чем размер окна компонента, и строка выходит за границы окна, то, при соответствующем значении свойства `ScrollBars`, будет появляться горизонтальная полоса прокрутки.

Свойство `ScrollBars` разрешает или запрещает появление полос прокрутки в окне компонента. Оно может принимать следующие значения:

- 0 — **None**. Полосы прокрутки не появляются;
- 1 — **Horizontal**. Разрешено появление только горизонтальной полосы прокрутки;
- 2 — **Vertical**. Разрешено появление только вертикальной полосы прокрутки;
- 3 — **Both**. Разрешено появление вертикальной и горизонтальной полос прокрутки.

Свойство `Text` позволяет записывать в буфер компонента и считывать из него неформатированный текст (т. е. обычный текст, который потом можно открыть для редактирования в редакторе Notepad).

Свойство `TextRTF`, в отличие от свойства `Text`, используется для работы с форматированным текстом (формата RTF).

В следующем фрагменте кода показано, как загрузить в `RichText` RTF-файл:

```
thisform.Olecontrol1.textRTF = FILETOSTR(GETFILE("RTF"))
```

Свойство `Font` содержит ссылку на объект `Font`, реализующий стандартные свойства типа `Name`, `Size`, `Bold`, `Italic` и т. д. Особенность свойства заключается в том, что заданное в нем форматирование применяется *ко всему тексту*, отображаемому в `RichText`.

В следующем фрагменте кода показано, как установить единый формат шрифта:

```
WITH thisform.OLECONTROL1.Font && Для объекта Font
   .size = 12                  && высота шрифта
   .Name = "Times New Roman"   && наименование шрифта
   .Bold = .t.                 && полужирный
   .Italic = .t.               && наклонный
ENDWITH
```

Свойство `Locked` управляет редактированием текста в компоненте. Установка значения свойства в "истину" блокирует изменения.

Свойства типа *SelXXX*

Эта группа свойств предназначена для обработки выделенного фрагмента текста.

Свойство `SelLength` содержит количество символов в выделенном фрагменте текста. Для того чтобы программно отменить выделение, присвойте свойству значение нуль.

Свойство `SelStart` содержит номер первого символа выделенного фрагмента относительно начала текста. Вы можете программно выделить фрагмент текста, записав в свойство `SelStart` номер первого символа фрагмента, а в свойство `SelLength` — длину этого фрагмента. Свойство `SelStart` может быть так же использовано для позиционирования внутри текста: фрагмент текста, в котором находится указанная позиция, становится видимым.

Свойства `SelColor`, `SelFontName`, `SelItalic` и аналогичные позволяют применить форматирование к выделенному фрагменту текста. Если выделенного фрагмента не существует, то форматирование применяется к символам, вводимым после позиции указателя текста.

Последние свойства, представляющие для нас интерес, это свойства `SelRTF` и `SelText`. Свойство `SelRTF` содержит строку выделенного текста в формате RTF, а свойство `SelText` — в неформатированном виде.

Методы

Из всего набора методов компонента **RichText** мы рассмотрим следующие:

- `Find` — выполняет поиск подстроки в заданной области;
- `SaveFile` — сохраняет документ в формате RTF в указанном файле;
- `SelChange` — обрабатывает событие, возникающее при изменении текста;
- `SelPrint` — печатает документ на принтере.

Метод *Find*

Метод `Find` выполняет поиск подстроки внутри заданной области и возвращает номер первого символа подстроки в тексте или –1, если подстрока не найдена. Он получает следующие параметры:

- искомая подстрока;
- номер начальной позиции в тексте;
- номер конечной позиции в тексте;
- необязательный параметр, определяющий поведение компонента после завершения метода.

В следующем фрагменте кода показано, как выполнить поиск подстроки и выделить ее красным цветом и полужирным начертанием (предполагается, что объекту `RichText` присвоено имя `oRTF`):

```
nStart = 0
nFullLen = LEN(thisform.oRTF.text)   && Длина всего текста в компоненте
cString = "Этот текст"                          && Подстрока поиска
nStringLen = LEN(cString)                       && Ее длина
nPos = thisform.oRTF.Find(cString, 0, nFullLen) && Поиск...
IF nPos > 0
   thisform.oRTF.SelStart = nPos          && Отмечаем начало фрагмента
   thisform.oRTF.SelLength = nStringLen   && и его длину
   thisform.oRTf.SelBold = .t.            && Устанавливаем полужирный
   thisform.oRTF.SelColor = RGB(255,0,0)  && шрифт и красный шрифт
ENDIF
```

Если вы хотите выделить все вхождения подстроки в тексте, то выполняйте вызов метода `Find` в цикле, пока он не вернет –1. При каждой очередной итерации цикла увеличивайте значение свойства `SelStart` на единицу, чтобы избежать зацикливания.

Метод *SaveFile*

Метод сохраняет файл в формате RTF. Он получает два параметра: первый содержит имя файла или дескриптор потока ввода-вывода, а второй (необязательный) — флаг; значение этого параметра при выводе в файл (если он указан) должно быть равно нулю.

Следующий фрагмент кода демонстрирует использование метода:

```
lcFile = PUTFILE("Имя RTF файла:","","RTF")
IF !EMPTY(lcFile)
   thisform.oRTF.SaveFile(lcFile)
ENDIF
```

Вы можете использовать альтернативные способы сохранения файла при помощи встроенной функции `STRTOFILE()`:

```
* Сохранение файла в формате RTF
STRTOFILE(thisform.oRTF.textRTF, "MyText.rtf")
* Сохранение неформатированного текста
STRTOFILE(thisform.oRTF.text, "MyText.txt")
```

Метод *SelChange*

Метод является обработчиком события `Change`, которое возникает при всяком изменении текста. Метод не имеет параметров.

Метод *SelPrint*

Метод печатает текст на принтере. Он получает указатель на графический контекст принтера.

О том, что такое графический контекст принтера, мы подробно поговорим в *главе 23*, а сейчас приведем фрагмент кода, необходимый для правильного выполнения метода:

```
DECLARE Long CreateDC IN WIN32API String, String, String, String
HDC = CreateDC(Null, SET("Printer",2), Null, Null)
thisform.oRTF.SelPrint(HDC)
```

В методе вызывается Windows API-функция `CreateDC`, которая возвращает графический контекст принтера, используемого по умолчанию.

Для правильной работы тегов форматирования, управляющих выравниванием строк, определите ширину документа, присвоив необходимое значение свойству `RightMargin` компонента.

Контекстное меню *RichText*

Если в процессе выполнения вы щелкнете правой кнопкой мыши по области компонента, то появится контекстное меню (рис. 18.7).

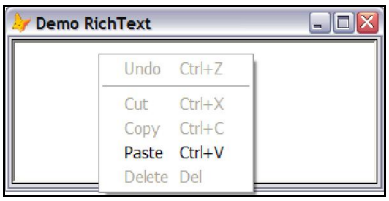

Рис. 18.7. Контекстное меню ActiveX-компонента **RichText**

Используя это меню, вы можете "откатываться назад", удалять, вырезать, вставлять и копировать фрагменты текста в буфер обмена Windows.

Работа с буфером обмена

Вы можете копировать выделенные фрагменты текста в буфер обмена Windows, а так же вставлять в документ данные из буфера программно. Для этого необходимо воспользоваться Windows API-функцией `SendMessage` и тем, что ActiveX-компонент, в отличие от обычных элементов управления Visual FoxPro, предоставляет свойство `HWnd`, которое передается в API-функцию как параметр.

> ***Замечание***
> Подробно о применении функций Windows API рассказывается в *главе 19*.

Для копирования выделенного фрагмента текста в буфер обмена Windows используйте следующий код:

```
#DEFINE WM_COPY 0x0301
DECLARE Long SendMessage IN WIN32API Long, long, long, string
SendMessage(thisform.oRTF.hwnd, WM_COPY, 0, 0)
```

А в следующем фрагменте кода показано, как вставить текст из буфера обмена Windows:

```
#DEFINE WM_PASTE 0x0302
DECLARE Long SendMessage IN WIN32API Long, long, long, string
SendMessage(thisform.oRTF.hwnd, WM_PASTE, 0, 0)
```

И в том и в другом примере подразумевается, что компонент **RichText** имеет имя `oRTF`.

Форматирование RTF-документа

Нет, вы не увидите здесь описания всех тегов форматирования, которые входят в стандарт RTF. Вы найдете здесь только совет, где и как получить необходимую информацию.

Во-первых, конечно же, это Интернет. Существует достаточно большое количество сайтов, посвященных RTF.

Во-вторых, это редактор WordPad, который входит в состав Windows. В отличие от Microsoft Word, он генерирует достаточно простой RTF-код. Наберите в WordPad текст, отформатируйте его и сохраните в формате RTF. Как правило, я именно таким способом нахожу нужные теги и изучаю порядок их применения.

Единственное, о чем нельзя забывать, — это то, что символы кириллицы в RTF заменяются на esc-последовательности. Поэтому при изучении формата RTF лучше пользоваться латинским алфавитом...

Компонент *Web Browser*

Если у вас установлена русскоязычная версия Windows, то этот компонент в окне **Insert Control** называется *Обозреватель веб-страниц (Microsoft)*. Он находится в файле shdocvw.dll.

Как следует из названия, компонент предназначен для просмотра страниц в сети Интернет. Но этим его возможности далеко не ограничиваются: он может показывать содержимое HTML-документа как страницу сайта; он может использоваться как файловый менеджер, позволяя просматривать папки и запускать на выполнение различные приложения.

Здесь мы рассмотрим два аспекта применения компонента **Web Browser**: как средства для создания справочной системы, основанной на HTML-файлах, и как инструмент для создания интерактивных отчетов.

Метод *Navigate2*

Этот основной метод компонента, который выполняет переход на новую страницу по ее адресу в формате URL. Метод получает следующие параметры:

- *url* — адрес в формате URL, по которому находится загружаемая страница (файл);
- *flags* (необязательный) — определяет, добавить ли ресурс к списку хронологии, читать ли записи из кэша, отобразить ли ресурс в новом окне. Вычисляется как сумма значений, перечисленных в табл. 18.1;
- *targetframename* (необязательный) — ссылка на структуру, содержащую имя фрейма для отображения ресурса или NULL;
- *postdata* (необязательный) — данные, посылаемые из метода POST HTML-формы;
- *headers* (необязательный) — ссылка на структуру, которая содержит дополнительные HTTP-заголовки, посылаемые серверу.

Таблица 18.1. *Битовые маски параметра* `flags` *метода* `Navigate2`

Обозначение в MSDN	Значение	Описание
navOpenInNewWindow	1	Открывает загружаемую страницу в новом окне

navNoHistory	2	Не добавляет URL страницы в список просмотренных страниц; новая страница заменяет текущую страницу в списке
navAllowAutosearch	16	Если происходит сбой при переходе на новый адрес, то выполняется автопоиск в общих корневых доменах типа .com, .edu и т. д.

Обычно при вызове метода достаточно указать только первый параметр — URL страницы. Можно не указывать префиксы типа *http:*// (или *ftp:*///, или *file:*///); Web Browser сам пытается определить местонахождение страницы, что, например, при загрузке файла с диска потребует гораздо больше времени, чем в случае, когда явно указан префикс *file:*///:

```
thisform.WebBrowserControl.Navigate2("file:///d:\MyWebs\Index.html")
```

Окно справки, использующее Web Browser

Как правило, на что у программиста никогда не хватает времени — так это на написание справочной документации. Мои коллеги, услышав предложение "задокументировать" продукт, досадно морщатся и всячески намекают на то, как много у них других дел.

Использование компонента **Web Browser**, как ни странно, позволяет упростить процесс создания справочной документации. Во время создания очередной формы вашего приложения вы можете "нарисовать" небольшой HTML-файл, содержащий необходимую информацию о работе с этой формой. Затем, когда создание приложения подойдет к концу, вы можете при помощи **Help Work Shop** после незначительной доработки скомпоновать эти файлы в справочный файл формата CHM. А компонент **Web Browser**? Он будет использоваться как средство для просмотра справочной информации в процессе отладки приложения (а может, и после).

Создайте в вашем проекте папку, предназначенную для хранения HTML-файлов, содержащих справочную информацию. Было бы неплохо, чтобы каждый HTML-файл имел имя, указанное в свойстве `Tag` описываемой им формы (надо же когда-нибудь начать использовать это свойство) — это облегчит нам процедуру вызова нужного файла. При необходимости создайте гиперссылки — **Web Browser** позволит вам перемещаться по HTML-файлам как по Web-страницам.

Создайте в проекте форму и разместите на ней компонент так, чтобы он полностью был вписан в клиентскую область окна. Установите свойство `Anchor` компонента равным 15, чтобы он изменял свои размеры при изменении размеров окна, всегда занимая всю клиентскую область. Дайте компоненту имя oBrowser. В свойстве `Caption` формы напишите что-то типа "Оперативная справка". Желательно, чтобы форма была модальной, или, по крайней мере, всегда находилась поверх остальных окон.

Откройте для редактирования метод `Init` формы и введите в него следующий код:

```
LPARAMETERS tcURL
```

```
this.oBrowser.navigate2("file:///" + tcUrl)
```

При инициализации формы этот метод получает строку, содержащую имя HTML-файла и полный путь к нему; в нем вызывается метод `Navigate2` объекта — экземпляра класса **Web Browser**, в который имя файла передается уже в формате URL, после чего в окне объекта будет отображено содержимое файла.

Для вызова этой формы мы должны перехватывать в главной форме нажатие на клавишу <F1>. Проще всего это сделать при использовании в главном меню приложения пункта вызова справки, связанного с этой "горячей" клавишей; тогда вы сможете получать справочную информацию как при выборе этого пункта меню, так и при нажатии на клавишу <F1>.

Предположим, что при выборе пункта **Помощь** главного меню, связанного с "горячей" клавишей <F1>, вызывается процедура `MyHelp`. В этой процедуре должен быть примерно такой код:

```
PROCEDURE MyHelp
LOCAL lcHTMLFile
lcHTMLFile = _VFP.ActiveForm.tag    && Достаем имя файла из свойства Tag
IF LEN(lcHTMLFile) > 0
   lcHTMLFile = qcPath + "\" + lcHTMLFile + ".htm"
   IF FILE(lcHTMLFile)
      DO FORM Help WITH lcHTMLFile
   ENDIF
ENDIF
```

Предполагается, что в глобальной переменной `qcPath` указан полный путь к папке с HTML-файлами (без завершающего обратного слэша); например, это может быть путь, возвращаемый после выполнения команды

```
PUBLIC qcPath
qcPath = JUSTPATH(SYS(16))
```

в корневом PRG-файле приложения. В этом случае HTML-файлы должны располагаться в той же папке, из которой стартует ваше приложение.

Вот и все. После отладки очередной формы проекта создайте HTML-файл, внесите в него текст справки, а в свойстве `Tag` формы укажите имя этого файла. Так постепенно, по мере создания приложения, будет сформирована и справочная документация для него!

Интерактивные отчеты в *Web Browser*

Вы можете использовать **Web Browser** как средство просмотра отчетов. Для этого ваше приложение должно сформировать HTML-документ и сохранить его в файле, а затем передать имя этого файла в метод `Navigate2` компонента.

Если загруженный в компонент файл содержит гиперссылки, то вы можете переходить к просмотру других файлов и даже можете выйти в Интернет. Особенностью

Web Browser является способность перехвата команд перехода по гиперссылкам или данных, посылаемых методом `POST` HTML-формы. Перед тем как переход по указанному адресу будет выполнен, происходит событие, которое вызывает для обработки метод `BeforeNavigate2`. В этом методе вы можете проанализировать передаваемые данные и решить, нужно ли перейти на новую страницу или необходимо вызвать процедуру Visual FoxPro для обработки этих данных.

Метод *BeforeNavigate2*

Метод получает следующие параметры:

- *`pdisp`* — ссылка на интерфейс `IDispath` объекта, представляющего собой окно или фрейм, в который загружена страница;
- *`url`* — адрес страницы в формате URL, которую нужно загрузить;
- *`flags`* — в настоящее время не используется;
- *`targetframename`* (необязательный) — ссылка на структуру, содержащую имя фрейма для отображения ресурса или NULL;
- *`postdata`* — строка, содержащая список имен и их значений, переданных из метода `POST` HTML-формы;
- *`headers`* (необязательный) — ссылка на структуру, которая содержит дополнительные HTTP-заголовки, посылаемые серверу. Заголовки могут определять информацию типа требуемого действия сервера, тип данных, передаваемых к серверу, и код состояния;
- *`cancel`* — значение логического типа, определяющее действие компонента. Если вы присвоите этому параметру значение "ложь" (это — значение по умолчанию), то Web Browser загрузит страницу, адрес которой указан в *`url`*; в противном случае страница не будет загружена.

Из всего многообразия параметров метода для принятия решения нам достаточно проанализировать только два: *`url`* и *`postdata`*.

Обработка гиперссылок

Адрес страницы, указанной в гиперссылке, содержится в передаваемом методу `BeforeNavigate2` параметре *`url`*. Обычно адреса в URL начинаются с префикса *http:*// или (при загрузке файлов) — *file:*///. Но ведь никто не запрещает нам придумать и использовать свои префиксы, например, *vfp:*///.

В листинге 18.1 приведен код метода `BeforeNavigate2`, в котором анализируется префикс URL и, в зависимости от его значения, загружается указанный файл либо вызывается процедура Visual FoxPro.

```
LPARAMETERS pdisp, url, flags, targetframename, postdata, headers, cancel   IF
UPPER(LEFT(url,4)) == "VFP:" && Если префикс — Vfp:
   DO ProcedureName             && то вызываем процедуру Visual FoxPro
   CANCEL = .t.                 && и запрещаем загрузку новой страницы
ENDIF
```

Скорее всего, простой вызов процедуры вас не устроит: неплохо было бы передать этой процедуре какие-нибудь параметры!

Для демонстрации решения этой проблемы нам потребуется небольшой проект.

Создайте в проекте форму, разместите на ней **Web Browser**, свойству `Name` присвойте значение `oBrowser`. Установите значения свойства `Anchor` равным 15.

Разместите на форме командную кнопку. У вас должно получиться примерно то, что показано на рис. 18.8.

В метод `Click` кнопки введите следующий код:

```
LOCAL lcFile
lcFile = GETFILE("HTM|HTML")
IF !EMPTY(lcFile)
   thisform.oBrowser.Navigate2("file:///" + lcFile)
ENDIF
```

При нажатии на кнопку будет появляться диалоговое окно **Open**, в котором вы должны будете выбрать HTML-файл. Этот файл будет загружен в окно компонента.

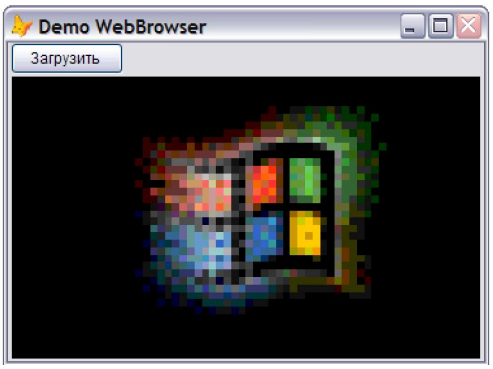

Рис. 18.8. Вид формы в Конструкторе форм

В редакторе Notepad введите код демонстрационного HTML-файла (листинг 18.2).

```
<html>
<head>
<meta http-equiv="Content-Language" content="ru">
```

```
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
<title>Демонстрационный пример</title>
</head>
<body>
<p>Проверка перехвата сообщений</p>
<p><a href="http://microsoft.com">Перейти на сайт Microsoft.</a></p>
<p><a href="vfp:///thisform.mymethod(16,'Текстовая строка')">
Вызвать метод формы</a></p>
</body>
</html>
```

Сохраните код в файле с любым именем и расширением htm.

> ***Замечание***
> Конечно, для создания HTML-файлов лучше использовать специально разработанные приложения, например, Front Page из Microsoft Office.

В этом HTML-файле реализованы две гиперссылки:

```
<a href="http://microsoft.com">Перейти на сайт Microsoft.</a>
```

и

```
<a href="vfp:///thisform.mymethod(16,'Текстовая строка')">
Вызвать метод формы</a>
```

При щелчке мышью по первой гиперссылке загрузится главная страница сайта Microsoft, а что произойдет при щелчке по второй ссылке? Да, вместо вызова метода формы `mymethod` и передачи ему двух параметров вы получите сообщение об ошибке. Но ситуацию можно поправить, написав в методе `BeforeNavigate2` следующий код (листинг 18.3).

```
LPARAMETERS pdisp, url, flags, targetframename, postdata, headers, cancel
LOCAL lcCmd
IF UPPER(LEFT(url, 4)) == "VFP:"  && Префикс URL: Vfp:?
   lcCmd = SUBSTR(url, 8)         && да, выделяем из строки вызов метода
   &lcCmd                         && Вызываем метод формы
   CANCEL = .t.                   && запрещаем загрузку новой страницы
ENDIF
```

Как видите, решение проблемы — в использовании макроопределения.

> ***Замечание***
> При формировании URL после префикса *vfp:* всегда должно следовать *три* обратных слэша: "Vfp:///".

Но не все так просто. Работая с Internet Explorer, вы наверняка обращали внимание на вид формируемых адресов ссылок. Вот и при выполнении кода из листинга 18.3 в переменной *lcCMD* будет записана следующая строка:

```
thisform.mymethod(16,'Текстовая%20строка')
```

Появившаяся в строке esc-последовательность *%20* всего лишь обозначает пробел. Но дело в том, что строка URL может содержать множество таких esc-последовательностей. Естественно, такая команда не может быть выполнена в Visual FoxPro.

Решить эту проблему можно при помощи Windows API-функции `UrlUnescape`. Эта функция заменяет все esc-последовательности на соответствующие им символы.

Модифицируем код метода `BeforeNavigate2`, добавив в него команды перекодировки строки (листинг 18.4).

```
LPARAMETERS pdisp, url, flags, targetframename, postdata, headers, cancel
LOCAL lcCmd, lcOutStr
IF UPPER(LEFT(url, 4)) == "VFP:"
   DECLARE Long UrlUnescape IN ShlWapi.dll string, string @, long @, long
   lcCmd = SUBSTR(url, 8)
   lcOutStr = SPACE(LEN(lcCmd))
   lnLenOutStr = LEN(lcOutStr)
   = UrlUnescape(lcCmd, @lcOutStr, @lnLenOutStr, 0)
   lcCMD = LEFT(lcOutStr, lnLenOutStr)
   &lcCMD
   CANCEL = .t.
ENDIF
```

Для того чтобы проверить, действительно ли при обработке второй гиперссылки нашего HTML-файла будет вызван метод формы, добавим в форму этот метод. Как следует из гиперссылки, его имя — `mymethod`. Вот его код:

```
LPARAMETERS tnValue, tcText
= MESSAGEBOX(STR(tnValue) + CHR(13) + tcText)
```

Ничего сложного. Метод получает два параметра и выводит их значения в диалоговом окне **MESSAGEBOX**.

Запустите форму на выполнение. Нажмите на кнопку **Загрузить**. В диалоговом окне **Open** выберите созданный на основе листинга 18.2 HTML-файл. Щелкните мышью по гиперссылке *Перейти на сайт Microsoft*. Загрузится страница корпорации Microsoft. Для того чтобы вернуться назад, щелкните правой кнопкой мыши по области компонента **Web Browser** и в появившемся контекстном меню выберите пункт **Назад**.

Щелкните по гиперссылке *Вызвать метод формы*. Метод будет вызван, и на экране появится диалоговое окно **MESSAGEBOX**, как это показано на рис. 18.9.

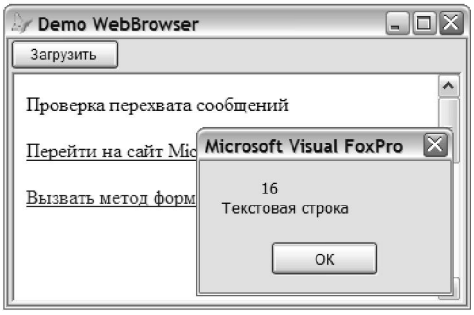

Рис. 18.9. Вызов метода формы Visual FoxPro из Web Browser

Подготовка данных для отправки методом *POST* HTML-формы

Данные могут быть посланы методом `POST` HTML-формы. Возможно, на первый взгляд, для приложений Visual FoxPro это выглядит несерьезно, но, вполне возможно, что ваши пользователи захотят иметь в отчетах, формируемых как HTML-документы, формы, заполнив которые они могут отправить данные на обработку вашему приложению.

Для демонстрации механизма получения данных от HTML-формы создайте в редакторе Notepad новый HTML-файл, код которого показан в листинге 18.5.

```
<html>
<head>
<meta http-equiv="Content-Language" content="ru">
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
<title>Демонстрационный пример</title>
</head>
<body>
<p>Проверка посылки сообщений методом POST</p>
<form action="Vfp:///" method="post">
<table border="0" width="300" cellspacing="3" cellpadding="3">
   <tr>
      <td width="90">Ваше имя:</td>
      <td><input type="text" name="txt1" size="30"></td>
   </tr>
   <tr>
      <td>Пол:</td>
      <td><select size="1" name="D1">
```

```
          <option value="0" selected>Мужской</option>
          <option value="1">Женский</option>
          </select></td>
   </tr>
   <tr>
      <td>Образование:</td>
      <td><select size="1" name="D2">
          <option value="0">Начальное</option>
          <option value="1">Среднее</option>
          <option value="2">Среднее специальное</option>
          <option value="3">Незаконченное высшее</option>
          <option value="4" selected>Высшее техническое</option>
          </select></td>
   </tr>
   <tr>
      <td> </td>
      <td><input type="submit" value="Отправить"></td>
   </tr>
</table>
</form>
</body>
</html>
```

HTML-файл содержит форму, для которой указан тег `Action`, содержащий параметр *Vfp:///*; именно эта строка будет передана методу `BeforeNavigate2` в параметре *`url`*. В теге `method` формы указан параметр *`post`*, т. е. данные элементов формы будут передаваться методом `POST`, и метод получит их в параметре *`postdata`*.

В форме созданы три объекта для ввода данных: текстовый бокс с именем *txt1* и два раскрывающихся списка с именами *D1* и *D2*. Данные посылаются при нажатии на кнопку "Отправить" типа *`submit`*.

Вид HTML-страницы в Internet Explorer показан на рис. 18.10.

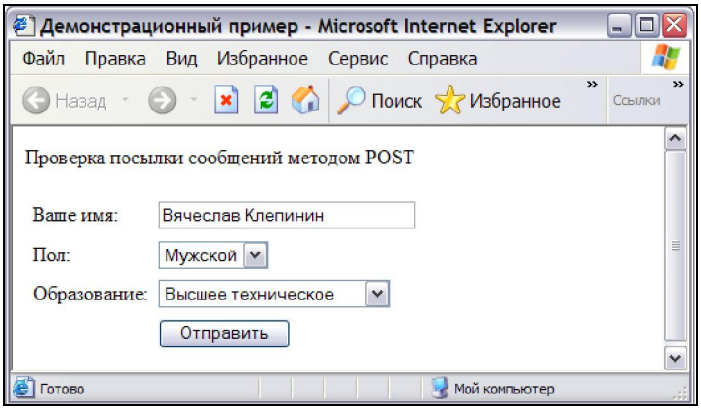

Рис. 18.10. Вид HTML-файла из листинга 18.5 в окне Internet Explorer

Получение данных из метода *POST* HTML-формы

Для того чтобы при возникновении события `BeforeNavigate` узнать, есть ли данные, посланные методом `POST`, нужно проверить тип данных в параметре *`postdata`* метода `BeforeNavigate2` при помощи встроенной функции `VARTYPE`. Если данные есть, то `VARTYPE` вернет "`Q`" (двоичные данные); иначе функция вернет "*X*".

Так как в параметре `postdata` находятся двоичные данные, то предварительно нужно преобразовать их в символьный формат при помощи встроенной функции `CAST`:

```
IF VARTYPE(postdata) = "Q"
   cPostData = CAST(postdata AS C(LEN(postdata))
ENDIF
```

Вот как будут выглядеть данные, посланные из HTML-формы (рис. 18.10), после конвертирования в символьный формат:

```
txt1=%C2%FF%F7%E5%F1%EB%E0%E2+%CA%EB%E5%EF%E8%ED%E8%ED&D1=0&D2=4
```

Как видите, получено три группы параметров типа *параметр=значение*, разделенных амперсандом (&). После обработки полученной строки функцией `UrlUnescape` получим:

```
txt1=Вячеслав+Клепинин&D1=0&D2=4
```

Здесь реализовано требование к данным, посылаемым методом `POST`: они не должны содержать пробелов, поэтому функция `UrlUnescape` заменяет пробелы на символы "плюс". Заменить "плюсы" на пробелы вы сможете при помощи встроенной функции `CHRTRANS()`.

В листинге 18.6 показан еще один вариант кода метода `BeforeNavigate2`, обрабатывающий данные, получаемые в параметрах *`url`* и *`postdata`*.

```
LPARAMETERS pdisp, url, flags, targetframename, postdata, headers, cancel
LOCAL lcCmd, lcOutStr
IF UPPER(LEFT(url, 4)) == "VFP:"
   DECLARE Long UrlUnescape IN ShlWapi.dll string, string @, long @, long
   lcCmd = SUBSTR(url, 8)
   IF LEN(lcCMD) > 0
      lcOutStr = SPACE(LEN(lcCmd))
      lnLenOutStr = LEN(lcOutStr)
      = UrlUnescape(lcCmd, @lcOutStr, @lnLenOutStr, 0)
      lcCMD = LEFT(lcOutStr, lnLenOutStr)
      &lcCMD
   ENDIF
   IF VARTYPE(postdata) = "Q"
      lcCmd = CAST(postdata as C(LEN(postdata)))
      lcOutStr = SPACE(LEN(lcCmd))
      lnLenOutStr = LEN(lcOutStr)
      = UrlUnescape(lcCmd, @lcOutStr, @lnLenOutStr, 0)
      lcCMD = CHRTRAN(LEFT(lcOutStr, lnLenOutStr),"+"," ")
      thisform.GetPostData(lcCMD)
   ENDIF
   CANCEL = .t.
ENDIF
```

Метод предполагает, что в форме существует пользовательский метод `GetPostData`, который получает и обрабатывает строку, полученную в параметре *`postdata`*.

Другие методы компонента *Web Browser*

Думается, у вас еще не возникло мысли создать полноценный Обозреватель Интернета на базе компонента **Web Browser**. Перед тем как приступить к его созданию, подумайте: а стоит ли?

Тем не менее коротко рассмотрим основные методы компонента, позволяющие выполнять позиционирование по страницам.

Метод `DocumentComplete` обрабатывает событие, извещающее о том, что страница полностью загружена. Он получает два параметра:

- `pdisp` — ссылка на интерфейс IDispath объекта, представляющего собой окно или фрейм, в который загружена страница;
- `url` — URL загруженной страницы (все esc-последовательности в строке заменены на "нормальные" символы.

Вы можете запоминать эти URL в массиве или объекте — экземпляре класса `Collection` для последующего просмотра ранее загруженных страниц (как это реализовано в Internet Explorer).

Методы `GoBack`, `GoForward`, `GoHome` и `GoSearch` не имеют параметров. Они используются для перехода на предыдущую, следующую или домашнюю страницы, либо для выполнения поиска.

Метод `NavigateError` вызывается при возникновении ошибки при переходе на новую страницу. Он получает следующие параметры:

- *`pdisp`* — ссылка на интерфейс IDispath объекта, представляющего собой окно или фрейм, в который загружена страница;
- *`url`* — URL страницы, попытка перехода на которую привела к возникновению ошибки;
- *`frame`* — ссылка на структуру, содержащую имя фрейма для отображения ресурса;
- *`statuscode`* — содержит код ошибки;
- *`cancel`* — логический параметр, указывающий, нужно ли формировать страницу с сообщением об ошибке в виде HTML-кода. Если значение параметра — "истина", то такая страница не формируется.

Web Browser так же предоставляет большое количество методов-обработчиков событий типа закрытия окна. Их изучение целесообразно только в том случае, если вы собираетесь создавать полноценный Обозреватель Web-страниц.

Функция *SYS(2333)*

В Visual FoxPro по умолчанию не поддерживается дуальный интерфейс для ActiveX, поэтому если компонент поддерживает раннее связывание (vtable-интерфейс), то у вас есть возможность значительно увеличить его быстродействие. Управлять поддержкой раннего связывания вы можете при помощи встроенной функции `SYS(2333)`. Вот ее синтаксис:

```
SYS(2333 [, 0 | 1 | 2])
```

Значения параметра

- 0 — отключает поддержку дуального интерфейса для ActiveX. Это значение по умолчанию для Visual FoxPro 6.0 и последующих версий;
- 1 — разрешает поддержку дуального интерфейса для ActiveX;
- 2 — возвращает текущее значение параметра (0 или 1).

Заключение

Мир ActiveX отличается чрезвычайным разнообразием; например, не нужно особых усилий для того, чтобы создать такой компонент в Visual Basic (хотя на C++ они получаются "более качественными"). Также обычно не возникает никаких проблем при использовании ActiveX в ваших приложениях. Конечно, невозможно в рамках одной

главы рассмотреть даже те ActiveX, которые поставляются с дистрибутивом Visual FoxPro, не говоря уже об ActiveX, установленных в вашей системе — посчитайте, сколько строк появляется в области **Control Type** диалогового окна **Insert Control**!

Информация, которую вы получили в этой главе, безусловно, будет полезной при освоении новых ActiveX, а применение компонентов **ProgressBar**, **Slider**, **RichText** и **Web Browser** в ваших приложениях позволит сделать их интерфейс более удобным и привлекательным!