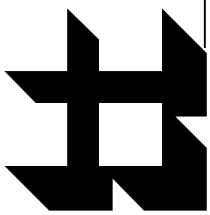
ГЛАВА 11



Создание пользовательских классов

Конструктор классов

В *главе 4* вы уже познакомились с Конструктором форм (Form Designer), который на самом деле является специализированной версией гораздо более мощного инструмента — Конструктора классов (Class Designer). Если Конструктор форм специализируется только на создании и редактировании форм, то в Конструкторе классов вы можете создавать новые классы, используя в качестве классов-родителей практически весь набор базовых классов Visual FoxPro, включая формы, за исключением классов `Empty`, `Header` и `Column`.

Конструктор классов хранит всю информацию в файлах с расширениями vcx и vct. Файл VCX представляет собой обычную DBF-таблицу, а файл VCT содержит memo-поля, в которых и хранятся основные данные. Но, в отличие от Конструктора форм, который для каждой формы или набора форм создает свой набор файлов, в одном VCX-файле можно хранить сведения о большом количестве классов, поэтому такие файлы принято называть *библиотеками классов*.

Классы и библиотеки классов

Вы можете создать библиотеку, классы которой будут использоваться только внутри конкретного проекта приложения. Файл такой библиотеки удобно хранить в папке проекта. Вы можете создать библиотеку универсальных классов и разместить ее в Галерее объектных компонентов Visual FoxPro, сделав классы доступными для использования во множестве приложений. В первом случае можно (и нужно) воспользоваться услугами Менеджера проектов, выделив в нем узел `Class Libraries` и нажав на кнопку **New**, во втором — воспользоваться главным меню Visual FoxPro, выбрав в нем пункты **File | New** или нажав на кнопку **New** стандартной панели инструментов; в появившемся диалоговом окне **New** выбрать **Class** и нажать кнопку **New File**. В результате любой из этих манипуляций на экране появится диалоговое окно **New Class** (рис. 11.1).

В этом окне нужно указать имя создаваемого класса (в поле **Class Name**) и имя класса-родителя (в поле **Based On**); родительским может быть как один из базовых классов Visual FoxPro, так и любой созданный вами класс, причем этот класс может располагаться как в этом, так и в любом другом библиотечном (или программном) файле. Если вы добавляете класс в уже существующую библиотеку классов, то ее имя (спецификация файла) выводится в поле **Store In**.

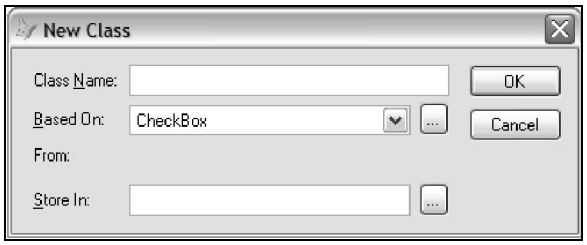


Рис. 11.1. Окно **New Class**

Начнем с уже знакомого вам, а именно — создадим в Конструкторе классов форму. Введите в поле **Class Name** имя создаваемого класса формы, например, **MyClassForm**. Откройте список базовых классов **Base On** и выберите в нем строку **Form**.

Для того чтобы использовать в качестве родительского любой другой созданный вами класс, достаточно нажать на расположенную правее поля **Based On** кнопку. Конструктор классов предложит вам выбрать нужный класс в диалоговом окне **Open** (рис. 11.2).

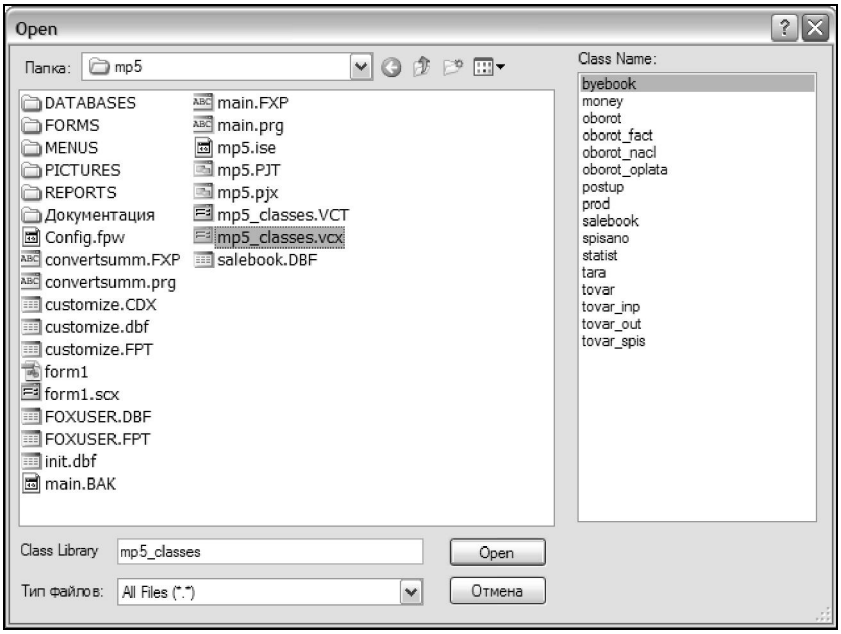


Рис. 11.2. Окно **Open** Конструктора классов

В этом окне вы можете выбрать библиотеку классов или программный файл, в котором находятся объявления классов (об объявлении классов в программном файле мы поговорим несколько позже). В расположенном справа списке вы видите список классов, включенных в выбранную библиотеку (или объявленных в программном файле). Чтобы использовать один из этих классов как родительский, установите на него указатель и нажмите на кнопку **Open**.

Вернемся к диалоговому окну **New Class**. Если библиотека классов еще не определена, то нажмите расположенную справа от поля **Store In** кнопку и в появившемся стандартном диалоговом окне **Save As** выберите папку и укажите имя файла сохраняемой библиотеки классов, например, `MyClasses`. Нажмите на кнопку **OK**. В указанной папке будут созданы файлы библиотеки классов, а Visual FoxPro загрузит Конструктор классов (рис. 11.3).

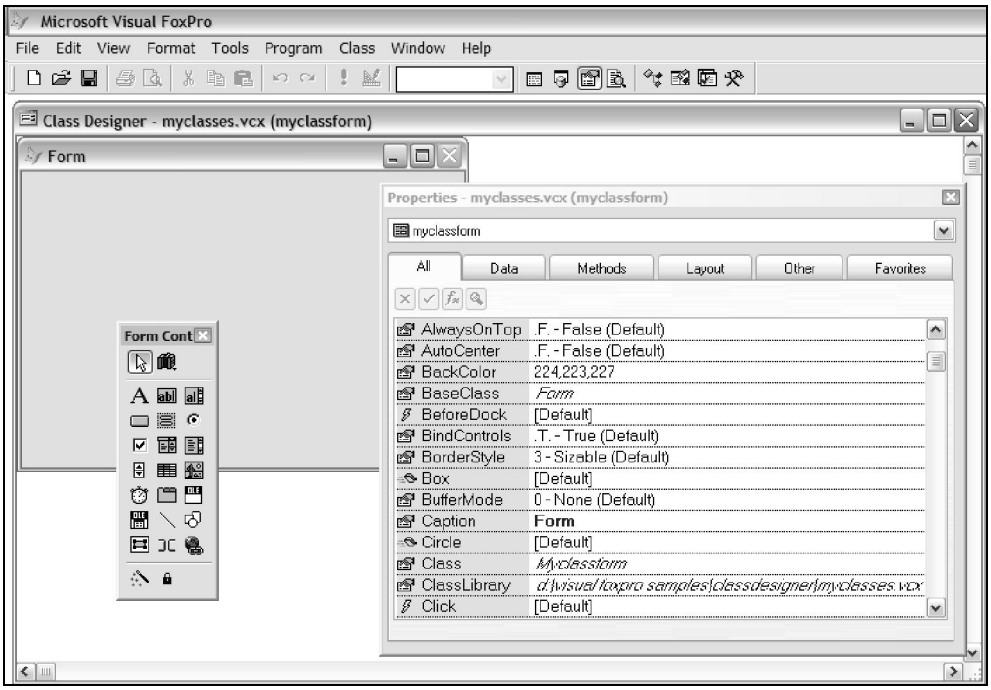


Рис. 11.3. Окно Конструктора классов

Конструктор классов в целом аналогичен Конструктору форм — он использует тот же набор инструментов и средств для визуального конструирования классов; основное отличие состоит в том, что в главном меню вместо пункта `Form` появился пункт `Class`. Обратите также внимание на информацию в окне свойств: в строке `Class` вы видите имя, данное вами классу формы, а в строке `ClassLibrary` — имя библиотеки классов. Строка `BaseClass` идентифицирует базовый класс Visual FoxPro; как видите, созданный класс является производным от базового класса `Form`.

Присвойте свойству `AutoCenter` формы значение `.T.` (истина). Разместите на форме кнопку (Command Button); в методе `Click` этой кнопки напишите следующий код:

```
Thisform.release
```

Закройте окно Конструктора классов.

Если вы создавали библиотеку классов из Менеджера проектов, то в результате ваших действий в узел **Class Libraries** будет добавлен новый вложенный узел, содержащий ссылку на созданную вами библиотеку классов; открыв этот узел, вы увидите в нем иконку только что созданной формы `myclassform` (рис. 11.4).

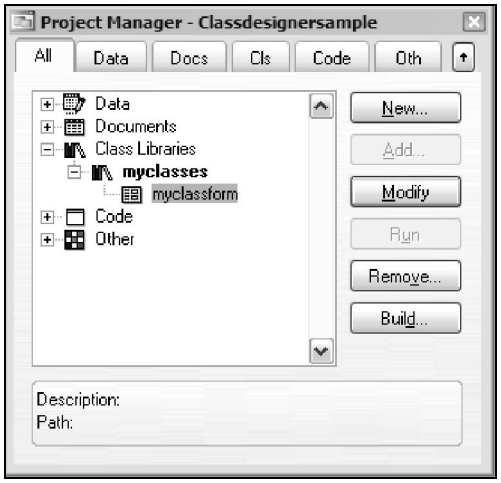


Рис. 11.4. Окно Менеджера проектов после добавления библиотеки классов

Создание объектов. Функции *CREATEOBJECT()* и *NEWOBJECT()*

Форму, созданную в Конструкторе форм, вы могли сразу запустить на выполнение, нажав на кнопку **Run** Менеджера проектов. Для формы, включенной в библиотеку классов, этого сделать нельзя. Следуя канонам объектно-ориентированного программирования, мы должны создать экземпляр класса — объект. В Visual FoxPro объект создается функциями `CREATEOBJECT()` или `NEWOBJECT()`.

Вот синтаксис объявления функции `CREATEOBJECT()`:

```
CREATEOBJECT(cClassName [, Parameters])
```

Параметры:

`cClassName`

имя класса из библиотеки классов или процедурного файла

`Parameters`

перечисленные через запятую необязательные параметры, которые будут переданы в метод `Init` объекта.

В листинге 11.1 показан код создания объекта формы из класса `MyClassForm` при помощи функции `CREATEOBJECT()`.

```
PUBLIC oForm
cPath = JUSTPATH(SYS(16))
SET DEFAULT TO (cPath)
SET CLASSLIB TO MyClasses
oForm = CREATEOBJECT('MyClassForm')
```

```
oForm.Show()
```

Первые три строки кода вам знакомы — это определение папки проекта для использования по умолчанию. Команда `SET CLASSLIB` указывает, что при поиске класса создаваемого объекта необходимо просмотреть библиотеку `MyClasses`. Если вы хотите использовать в проекте несколько различных библиотек классов, то для каждой следующей загружаемой библиотеки применяйте следующий формат команды:

```
SET CLASSLIB TO имя_файла_библиотеки_классов ADDITIVE
```

Функция `CREATEOBJECT()` создает объект — экземпляр класса `MyClassForm`; она так же возвращает объектную ссылку на созданный объект, которая записывается в переменную `oForm`.

Последнее действие, выполняемое в коде листинга, — это вызов метода `Show` созданной формы.

Сохраните этот код в программном файле; этот файл должен находиться в той же папке, в которой находится файл библиотеки классов. Запустите его на выполнение. Форма появится на экране. Нажмите кнопку на форме — форма закроется.

Функция `NEWOBJECT()` позволяет избежать использования команды `SET CLASSLIB`. Дело в том, что, выполняясь, эта команда загружает в память компьютера всю библиотеку классов; удалить библиотеку из памяти можно командой

```
RELEASE CLASSLIB имя_библиотеки_классов
```

Удалить из памяти все загруженные библиотеки можно командой `SET CLASSLIB TO` без параметров.

Применение функции `NEWOBJECT()` позволяет загрузить в память только нужный класс. Вот ее синтаксис:

```
NEWOBJECT(cClassName [, cModule [, cInApplication [, Parameters]]])
```

Параметры функции:

`cClassName`

имя класса из библиотеки классов или процедурного файла

`cModule`

необязательный параметр, определяющий имя файла (и, возможно, путь) библиотеки классов. Не используется, если необходимая библиотека загружена командой `SET CLASSLIB`.

`cInApplication`

определяет приложение Visual FoxPro (файл EXE или APP) содержащее VCX-файл, имя которого указано в `cModule`. Вы должны указать расширение файла приложения. `cInApplication` игнорируется, если параметр `cModule` опущен или если значение `cInApplication` — пустая строка или ноль. Параметр `cInApplication` может использоваться только тогда, когда указан параметр `cModule`.

`Parameters`

перечисленные через запятую необязательные параметры, которые будут переданы в метод `Init` объекта.

В листинге 11.2 приведен код создания формы из класса `MyClassForm` при помощи функции `NEWOBJECT()`.

```
PUBLIC oForm
cPath = JUSTPATH(SYS(16))
SET DEFAULT TO (cPath)
oForm = NEWOBJECT('MyClassForm', 'MyClasses')
oForm.Show()
```

Полиморфизм в действии

Добавим в созданную нами библиотеку классов новый класс, тоже форму, но в качестве родительского будем использовать не базовый класс Visual FoxPro, а только что созданный класс `MyClassForm`.

Откройте библиотеку классов. Для этого в Менеджере проектов выделите любой узел с именем класса, включенного в эту библиотеку (но не узел с именем библиотеки) — в нашем случае это узел с именем `MyClassForm`, и нажмите кнопку **New**. На экране появится диалоговое окно **New Class** (см. рис. 11.1), в поле **Store In** которого уже указано имя файла библиотеки классов. Введите в поле **Class Name** имя нового класса, например, `MyTwoForm`. Нажмите на кнопку, расположенную правее поля **Based On**, в появившемся окне **Open** (см. рис. 11.2) найдите файл нашей библиотеки классов MyClasses.vcx и шелкните мышью по имени этого файла. В расположенном справа списке появится строка с именем класса `MyClassForm`. Выделите эту строку и нажмите на кнопку **Open**; окно **Open** закроется. Теперь окно **New Class** должно выглядеть так, как показано на рис. 11.5.

Убедитесь, что в поле **Based On** действительно находится имя созданного вами ранее класса формы. Если все правильно, то нажмите кнопку **OK**. Появится окно Конструктора классов, в котором вы увидите вашу форму, содержащую командную кнопку. Щелкните по этой кнопке, чтобы открылось окно для редактирования метода `Click` объекта. В этом окне ничего нет! Но ведь когда вы создавали форму `MyClassForm`, то записали в этот метод код thisform.release! Что же случилось? Да ничего особенного. Код метода класса-родителя инкапсулирован в дочернем классе. Выберите в окне свойств объект `Command1` (это имя нашей командной кнопки) и найдите строку с описанием метода `Click`. Там вы увидите следующее:

```
[Inherited MyClassForm имя_файла_библиотеки_классов]
```

То есть код этого метода унаследован от класса-родителя. А можно ли увидеть этот код? Да, можно. Для этого щелкните на кнопке **View Parent Code**, расположенной над областью ввода окна редактора кодов (рис. 11.6).

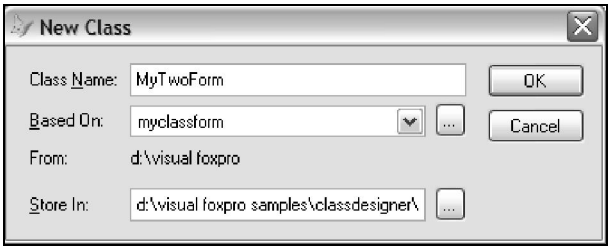


Рис. 11.5. Окно **New Class** с пользовательским классом-родителем

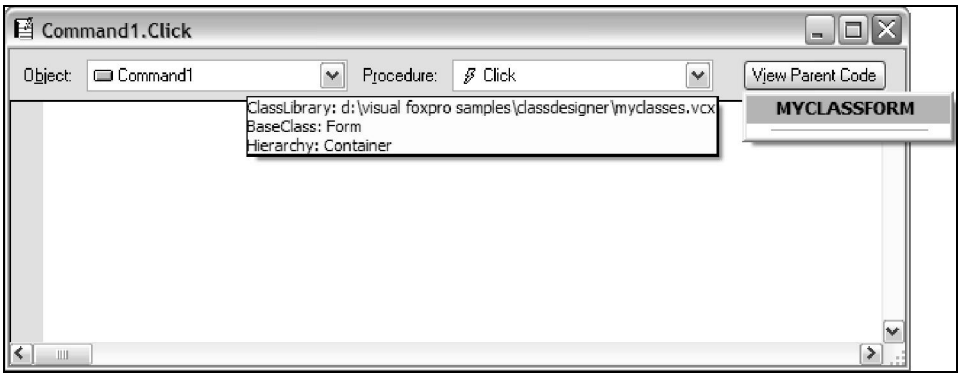


Рис. 11.6. Просмотр кода класса-родителя

В выпавшем списке выберите класс-родитель (этот список будет содержать имена всех предшествующих родительских классов). Откроется новое окно, в котором будет выведен код метода родительского класса. Вы не можете отредактировать этот код; но если вы все-таки хотите его изменить, то откройте в Конструкторе классов соответствующий класс из иерархии родителей. Но при этом вы должны помнить, что изменение коснется поведения всех классов — прямых наследников этого класса, а так же и всех их потомков! То есть, если этот метод не переопределен в одном из классов — потомков, то все объекты, созданные из этих классов, при вызове этого метода будут выполнять уже новый, измененный вами, код метода класса-родителя.

Продолжим эксперимент. Закройте Конструктор классов и выполните код из листинга 11.1, изменив имя параметра функции `CREATEOBJECT()` с `MyClassForm` на `MyTwoForm`. На появившейся форме нажмите на кнопку. Выполнится код класса-родителя, и форма закроется. Это — следствие того, что в Visual FoxPro используется так называемое *классическое наследование*, когда класс — наследник получает полный набор свойств и методов класса-родителя и всех включенных в него объектов. Чтобы окончательно понять сущность полиморфизма, снова откройте класс `MyTwoForm` и переопределите код метода `Click` командной кнопки, для чего введите в пустое окно редактора следующий код:

```
= MESSAGEBOX('Код метода класса-родителя переопределен!')
```

Закройте Конструктор классов и запустите модифицированный код из листинга 11.1 на выполнение. Нажмите на форме кнопку. Что произошло? Форма не закрылась, но появился стандартный диалог (рис. 11.7).

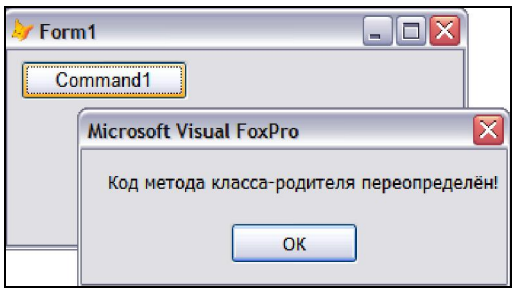


Рис. 11.7. Полиморфизм в действии

Как видите, код метода класса-родителя теперь не выполняется; вместо него выполняется код метода дочернего класса. Это и есть *полиморфизм при наследовании*. Если вы создадите еще один класс, используя в качестве родителя класс `MyTwoForm`, то в этом дочернем классе будет унаследован метод `Click` кнопки, выводящий диалоговое окно с сообщением **Код метода класса-родителя переопределен**!

Но существует ли возможность при перегрузке метода в дочернем классе выполнить код метода класса-родителя? Да, существует. Это — использование функции `DODEFAULT()`. Вызов этой функции в любом месте кода метода дочернего класса приводит к приостановке выполнения этого кода и выполнению кода метода класса-родителя. При вызове этой функции вы должны помнить о необходимости указать в ней все параметры, получаемые методом.

Изменим в последний раз код метода `Click` командной кнопки, используемой в классе `MyTwoForm`, добавив в него вызов функции `DODEFAULT()`:

```
= MESSAGEBOX('Код метода класса-родителя перегружен!')
DODEFAULT()
```

Что получится в результате этого изменения? При нажатии на кнопку формы по-прежнему будет появляться диалоговое окно с сообщением, но после закрытия этого окна форма так же закроется — т. е. выполнится метод `Click` кнопки класса — родителя `MyClassForm`.

Создание классов в Конструкторе классов

В *главе 5* вы познакомились с объектами — управляющими элементами, создаваемыми из базовых классов Visual FoxPro. Рассматриваемые в этом разделе классы `Container` и `Control` позволяют создавать новые визуальные управляющие элементы, которые вы так же можете использовать в своих приложениях наряду с базовыми.

Создание класса на основе базового класса Visual FoxPro

При конструировании формы в Конструкторе форм вам постоянно приходится изменять свойства размещаемых на ней управляющих элементов, например, тип шрифта и высоту символов в `TextBox`. И если вы применяете этот элемент с двумя-тремя измененными реквизитами во всех своих формах, то вам необходимо многократно выполнять настройку свойств для каждого такого размещаемого на форме элемента. Но можно поступить проще: создать пользовательский класс на базе класса `TextBox`, установив в нем необходимые значения свойств, и затем использовать этот класс в разрабатываемых формах. Конструктор классов предоставляет вам еще одну возможность, которая не была реализована в Конструкторе форм — это возможность включения новых свойств и методов в управляющий элемент на этапе разработки класса этого элемента.

Добавим в уже созданную вами библиотеку классов новый класс, родителем которого выберем класс `TextBox`. В Менеджере проектов выделите любой узел с именем класса библиотеки и нажмите на кнопку **New**. В появившемся окне **New Class** в поле **Class Name** введите `MyTextBox`, из списка базовых классов выберите `TextBox`. Нажмите кнопку **OK**. Загрузится Конструктор классов, в окне которого вы увидите окно **MyTextBox** (согласитесь, непривычно видеть управляющий элемент со стандартным оконным заголовком!) (рис. 11.8).

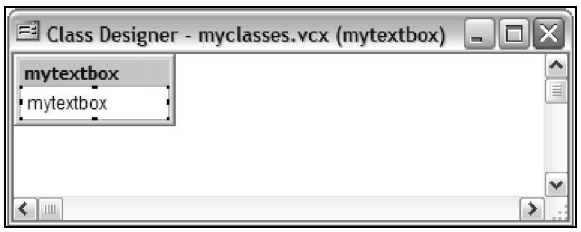


Рис. 11.8. Класс `MyTextBox` в окне Конструктора классов

Все ваши следующие действия — это установка новых значений свойств элемента в окне свойств.

После того как вы сохраните этот новый класс в библиотеке классов, в окне Менеджера проектов будет создан новый узел с его именем и иконкой (рис. 11.9).

Давайте ужесточим предъявляемые требования к функциональности класса `MyTextBox`. Мы добавим в класс новое свойство, которое позволит узнавать, изменялось ли содержимое текстового поля. Назовем это свойство `lChange`.

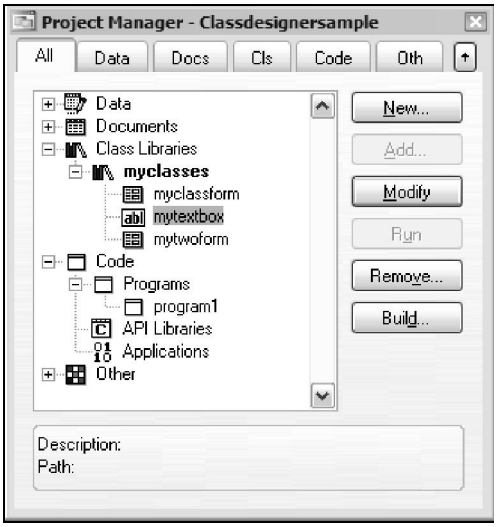


Рис. 11.9. Окно Менеджера проектов после добавления класса `MyTextBox`

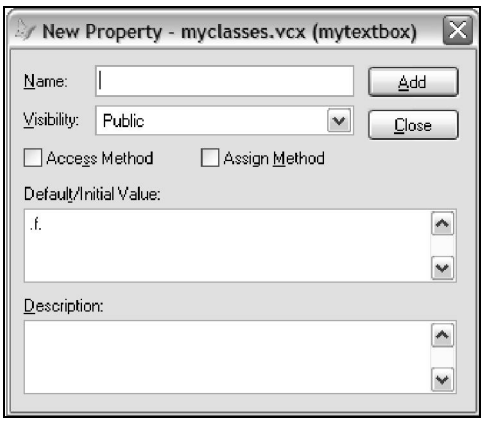


Рис. 11.10. Окно **New Property**

В Менеджере проектов выделите узел `mytextbox` и нажмите кнопку **Modify**. Загрузится Конструктор классов. Выберите в меню **Class** пункт **New Property...**. Конструктор выведет окно для добавления в класс нового свойства (рис. 11.10).

Это окно отличается от аналогичного окна Конструктора форм только наличием поля **Visibility**. В этом поле вы определяете уровень видимости добавляемого свойства:

- `Public` — свойство доступно извне класса объекта;
- `Protected` — свойство доступно только из методов класса; оно недоступно как извне класса, так и вложенным в класс-контейнер объектам. Свойство доступно при наследовании (доступно из методов дочерних классов);
- `Hidden` — свойство доступно только из методов класса; оно недоступно как извне класса, так и вложенным в класс-контейнер объектам. Свойство недоступно при наследовании.

Как и в Конструкторе форм, вы можете определить доступ к свойству посредством использования методов `Access` и `Assign`.

Введите в поле **Name** имя создаваемого свойства (`lChange`) и определите для него уровень видимости `Public`. Значение свойства по умолчанию оставьте без изменения. Нажмите кнопку **Add** для добавления свойства в класс и закройте окно.

В окне свойств выберите строку `InteractiveChange` и, дважды щелкнув по ней мышью, перейдите в режим редактирования этого метода. Введите в метод следующий код:

```
this.lChange = .T.
```

Событие `InteractiveChange` происходит при каждом изменении значения свойства `Value` объекта `TextBox` — вводите ли вы символ с клавиатуры или удаляете выделенный мышью фрагмент текста при помощи меню. Вот это событие и будет фиксироваться в свойстве `lChange`.

В метод `GotFocus` класса добавьте следующий код:

```
this.lChange = .F.
```

Теперь при каждом очередном получении элементом фокуса ввода значение свойства будет сбрасываться, и если пользователь ничего не введет, то свойство так и сохранит значение `False`.

Вот и все. После размещения объекта — экземпляра класса `MyTextBox` на форме в его методе `LostFocus` напишите код проверки значения свойства `lChange`:

```
IF this.lChange
    Выполняемый_код
ENDIF
```

Класс *Container*

Если в Конструкторе форм вы могли использовать управляющий элемент `Container` только для "наведения красоты", т. е. как обычный визуальный оформительский компонент, но никак не контейнер, то в Конструкторе классов вы можете создать на базе этого класса настоящий контейнер, типа **Command Group** или **Option Group**, разместив в нем различные управляющие элементы и другие объекты-контейнеры.

Добавьте в библиотеку классов новый класс с именем `MyContainer` и выберите для него базовый класс `Container`. На рис. 11.11 показано окно Конструктора классов с загруженным окном класса `Container`.

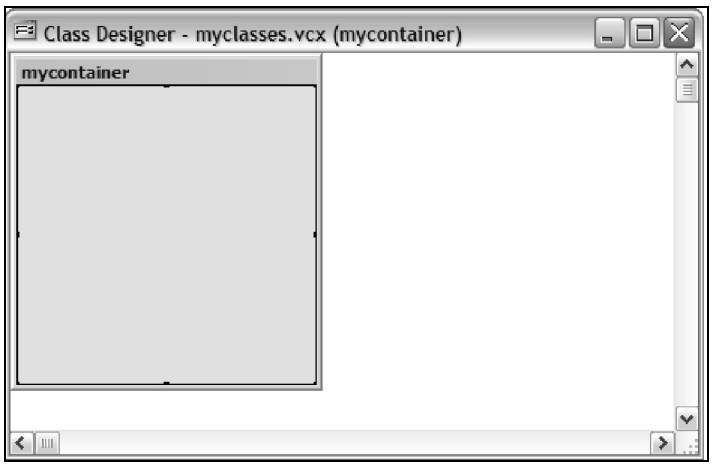


Рис. 11.11. Редактирование класса `MyContainer`

Точно так же, как вы размещали управляющие элементы на форме в Конструкторе форм, разместите в окне `mycontainer` четыре кнопки, метку и один `ComboBox`. Придайте окну вид, показанный на рис. 11.12.

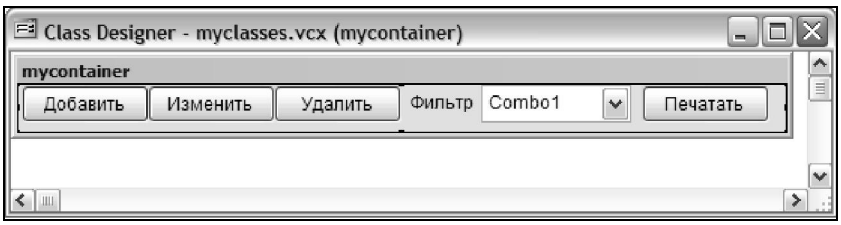


Рис. 11.12. Класс `MyContainer` с добавленными управляющими элементами

Получилось нечто похожее на панель инструментов, не так ли? Особенно, если на кнопках вместо надписей поместить пиктограммы?

Теперь осталось разместить созданный компонент на форме и убедиться, что он действительно ведет себя как контейнер.

Создайте проект и откройте в нем новую форму. В панели инструментов **Form Controls** нажмите кнопку **New Classes** (рис. 11.13).

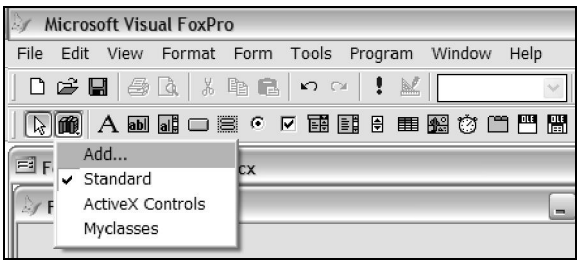


Рис. 11.13. Подключение библиотеки классов в Конструкторе форм

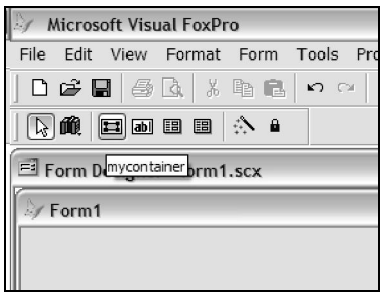


Рис. 11.14. Пользовательские классы в панели **Form Controls**

В появившемся контекстном меню выберите пункт **Add...**. Появится стандартный диалог для открытия файла, в котором вы должны будете найти и выбрать библиотеку классов, в которой находится наш класс `MyContainer`. Как только библиотека классов будет выбрана, панель инструментов **Form Controls** изменит свой вид, отображая классы из добавленной библиотеки (рис. 11.14).

Если вы наведете мышь на любую из иконок в панели инструментов **Form Controls**, Visual FoxPro выведет всплывающую подсказку с именем класса. Выберите класс `mycontainer` и положите его на форму. Увеличьте область формы для контейнера так, чтобы стали видны все размещенные на нем управляющие элементы. Подвигайте

контейнер по форме, чтобы убедиться, что это — один объект. Наконец, в окне свойств откройте список объектов формы. Вы увидите, что объект `Mycontainer1` (а именно такое имя дал новому объекту Конструктор форм) содержит четыре кнопки, метку и `ComboBox` (рис. 11.15).

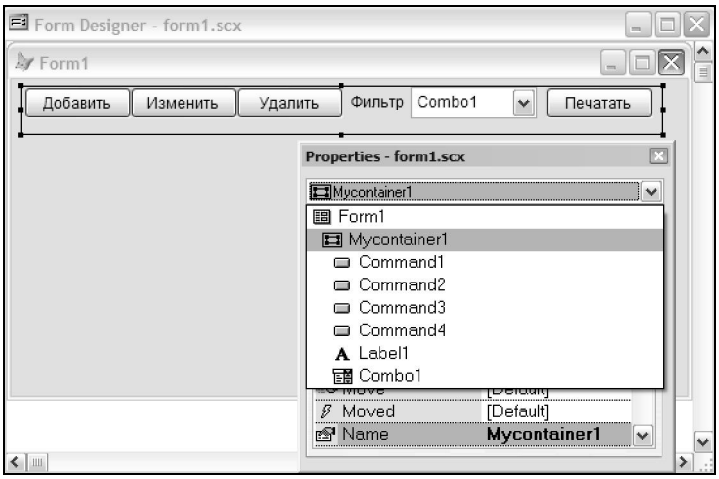


Рис. 11.15. Размещение контейнера на форме

Все методы и свойства вложенных в контейнер объектов доступны для программирования, как это и должно быть для объекта-контейнера.

Можно было бы на этом закончить описание класса `Container`, но мы, как обычно, усложним себе жизнь. Добавим в контейнер свойство `Value`, в котором будем запоминать номер нажатой кнопки, а также будем вызывать метод `Click` контейнера при нажатии на любую из размещенных на нем кнопок.

При добавлении в контейнер свойства `Value` (а такого свойства в базовом классе `Container` нет) установите для него уровень видимости `Public` и значение по умолчанию, равное нулю.

Номера кнопкам присвойте в соответствии с порядком их расположения в контейнере: слева направо. В методе `Click` каждой кнопки напишите следующий код:

```
this.Parent.value = номер_кнопки (1, 2 и т. д.)
this.Parent.Click()
```

Псевдоним `Parent` позволяет объекту обращаться к свойствам и методам объекта-контейнера, которому этот объект принадлежит. Так, свойство `Value` принадлежит контейнеру, поэтому кнопки используют псевдоним `Parent` для доступа к этому свойству (а также и к методу `Click` контейнера).

Теперь метод `Click` контейнера будет отрабатывать при нажатии на любую из вложенных в него кнопок; вы сможете проверять в этом методе значение свойства `Value`, чтобы определить, какая кнопка была нажата.

Класс *Control*

В отличие от класса `Container`, который разрешает доступ ко всем вложенным в него объектам, класс `Control`, также являясь контейнером, инкапсулирует в себе все свойства и методы включенных в него управляющих элементов, делая их недоступными извне класса. Таким образом, несмотря на свою контейнерную природу, объекты — экземпляры класса `Control` ведут себя как цельные компоненты, скрывая свое внутреннее строение.

Для демонстрации возможностей этого класса мы создадим управляющий элемент в виде окна с полосами прокрутки, в котором можно будет просматривать изображения. Создайте в библиотеке классов новый класс с именем `ScrollImage`, указав в качестве базового класс `Control`. В Конструкторе классов добавьте в созданный класс три компонента: `ImageBox` и два ActiveX-компонента `Microsoft Flat ScrollBar`. ActiveX-компоненты реализуют полосы прокрутки, которые, к сожалению, отсутствуют в наборе базовых классов Visual FoxPro. Присвойте свойству `Orientation` одной из полос прокрутки нулевое значение (вертикальная ориентация). Установите толщину полос прокрутки в 18 пикселов, горизонтально расположенной полосе дайте имя `ScrollX` (свойство `Name` элемента), вертикально расположенную полосу назовите `ScrollY`. Элемент `ImageBox` расположите таким образом, чтобы его верхний левый угол совпадал с левым верхним углом контейнера, т. е. свойства `Left` и `Top` элемента должны иметь нулевые значения. У вас должно получиться примерно то же, что показано на рис. 11.16.

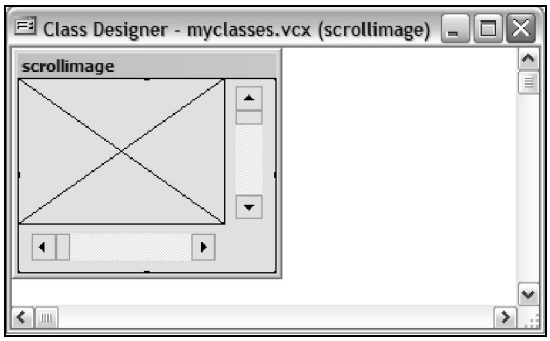


Рис. 11.16. Класс `ScrollImage` в Конструкторе классов

Пусть вас не смущает хаотическое расположение полос прокрутки. Их положение и размеры будут устанавливаться программно на этапе выполнения. К сожалению, Конструктор классов не поддерживает динамическое изменение размеров компонентов для таких композитных объектов, поэтому то, что мы видим при размещении это-

го элемента на форме, будет несколько отличаться от того, что мы увидим при выполнении этой формы.

Полосы прокрутки традиционно располагаются вдоль правой и нижней границы окна контейнера. Так как это размещение нужно делать программно, то в методе `Init` класса `ScrollImage` напишем следующий код (листинг 11.3).

```
WITH this.ScrollX
    .Left = 0
    .Width = this.Width — this.ScrollY.Width
    .Top = this.Height — .Height
ENDWITH
WITH this.ScrollY
    .Top = 0
    .Left = this.Width — this.ScrollX.Height
    .Height = this.Height
ENDWITH
this.Image1.Visible = .F.
```

Метод-обработчик события `Init` объекта-контейнера отрабатывает после того, как будут созданы все вложенные в него объекты. Таким образом, этот код располагает полосы прокрутки `ScrollX` и `ScrollY` соответственно по нижней и правой границам контейнера — а реальные размеры объекта-контейнера (и его границы) вы определите при конструировании формы.

Добавим в класс новый метод, который будет загружать изображение в `ImageBox`. Назовем его `ShowImage`. Метод будет получать два параметра. Первый параметр, `tcSource` — это символьная строка, содержащая либо имя файла, либо имя переменной или поля таблицы, содержащей изображение. Второй параметр, `tlTyp`, который может быть опущен, определяет, что содержит первый параметр — имя файла (`tlTyp=.F.`) или данные (`tlTyp=.T.`). Код метода `ShowImage` показан в листинге 11.4.

```
LPARAMETERS tcSource, tlTyp
LOCAL lnWidth, lnHeight
IF tlTyp                && В tcSource переменная (поле таблицы)
    this.Image1.PictureVal = tcSource
ELSE                 && В tcSource — имя файла
    this.Image1.Picture = (tcSource)
ENDIF
lnWidth = this.Image1.Width    && Ширина изображения
lnHeight = this.Image1.Height  && Высота изображения
*
* Настраиваем параметры полос прокрутки
```

```
*
this.ScrollX.Max = lnWidth - this.Width - this.ScrollY.Width
this.ScrollY.Max = lnHeight - this.Height - this.ScrollX.Height
this.Image1.Visible = .T.
```

Чтобы добавить этот метод в класс `ScrollImage`, выберите в меню **Class** пункт **New Method...**. В появившемся окне **New Method** (рис. 11.17) в поле **Name** введите имя нового метода — в нашем случае это `ShowImage`. Уровень видимости метода определяется точно так же, как и уровень видимости свойства. Так как этот метод должен быть доступен извне объекта, определите этот уровень как `Public`. Нажмите на кнопку **Add**, затем на кнопку **Close**. В окне редактора кодов наберите код из листинга 11.4.

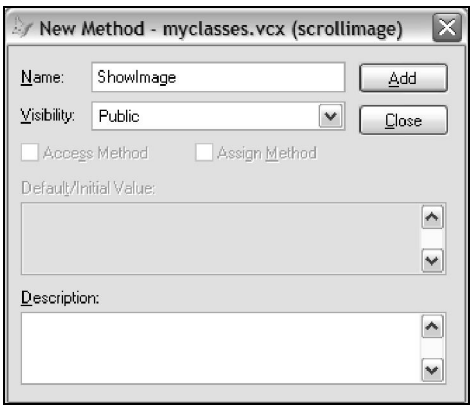


Рис. 11.17. Окно **New Method Конструктора классов**

Последнее, что осталось сделать, — это написать код методов объектов `ScrollX` и `ScrollY`. Этот код будет перемещать объект `ImageBox` внутри окна контейнера. ActiveX-объект `Microsoft Flat ScrollBar` имеет два метода, один из которых, `Change`, срабатывает, когда вы переместили мышью ползунок полосы прокрутки в новое положение и отпустили кнопку мыши. Второй метод, `Scroll`, срабатывает в процессе перемещения ползунка. Выбор используемого метода оставляю на ваше усмотрение, но считаю, что предпочтительнее использовать метод `Scroll`, т. к. в этом случае изображение внутри окна контейнера будет перемещаться плавно.

Вот код метода `Scroll` (или `Change`) для компонента `ScrollY`:

```
This.Parent.Image1.Top = -this.Value
```

Аналогичный код такого же метода для компонента `ScrollX`:

```
This.Parent.Image1.Left = -this.Value
```

Добавьте в проект форму, в Конструкторе форм разместите на ней объект — экземпляр класса `ScrollImage`. Добавьте на форму командную кнопку. Свойству `Caption` кнопки присвойте значение **Открыть**, а в методе `Click` напишите следующий код:

```
LOCAL lcFile
```

```
lcFile = GETFILE('JPG|GIF|PNG|BMP|TIF')
IF !EMPTY(lcFile)
    thisform.Scrollimage1.ShowImage(lcFile)
ENDIF
```

Как добавить в панель инструментов **Form Controls** Конструктора форм библиотеку классов, подробно рассмотрено в предыдущем разделе.

У вас должно получиться нечто, похожее на показанное на рис. 11.18.

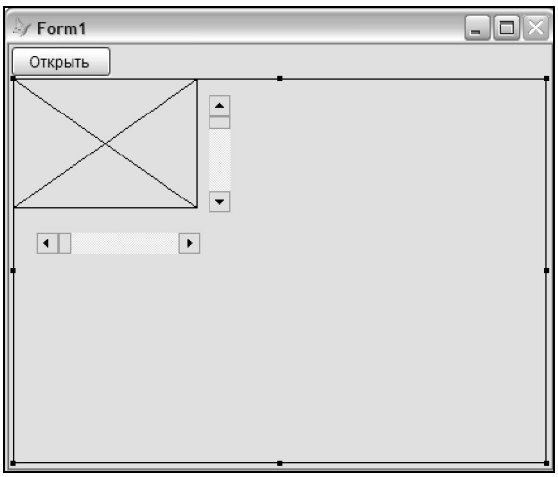


Рис. 11.18. Окно формы с объектом `ScrollImage1` в Конструкторе форм

Запустите форму на выполнение, нажмите на кнопку **Открыть**. В появившемся диалоге выберите файл, содержащий изображение одного из допустимых графических форматов. Вы должны увидеть примерно то, что показано на рис. 11.19.

Рис. 11.19. Окно формы с объектом `ScrollImage1` в процессе выполнения

Подвигайте ползунки на полосах прокрутки. Вы увидите, как изображение перемещается внутри окна управляющего элемента.

Последнее, что нам осталось сделать, — это убедиться в том, что объект `ScrollImage1` не является контейнером (конечно, только с точки зрения вашей программы). Откройте форму в Конструкторе форм, в окне свойств откройте список объектов (рис. 11.20).

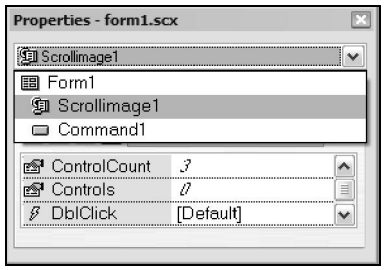


Рис. 11.20. Список объектов в окне свойств

Как видите, в списке нет никаких сведений об объектах `ImageBox` и `Microsoft Flat ScrollBar`. А в списке методов объекта `ScrollImage1` вы увидите метод `ShowImage`.

Класс *ToolBar*. Создание панели инструментов

Объекты — экземпляры этого класса представляют собой специализированные формы, на которых располагаются кнопки и (очень редко) другие управляющие элементы. В Конструкторе классов панель управления создается достаточно просто, хотя, на

мой взгляд, визуальная среда для конструирования могла бы быть более удобной. Впрочем, вы все оцените сами.

Добавить класс `ToolBar` в библиотеку классов так же просто, как и любой другой класс. В окне **New Class** (см. рис. 11.1) в поле **Class Name** введите имя панели, например, `MyPanel`. В списке **Base On** выберите класс `ToolBar`. Сохраните класс в библиотеке классов. В окне Конструктора классов появится окно панели инструментов. Все, что от вас требуется, — это разместить в этом окне командные кнопки и иные управляющие элементы.

Первое, что вы обнаружите, — это невозможность менять размеры окна панели инструментов в Конструкторе классов до тех пор, пока в нем не будет размещено несколько управляющих элементов. Второе неудобство — это невозможность произвольно размещать в этом окне добавляемые управляющие элементы; их взаимным расположением управляет Конструктор классов. И, тем не менее, приступим.

Добавьте в окно панели инструментов командную кнопку (**Command Button**). В окне свойств определите размеры кнопки: 24 пиксела по ширине и высоте (свойства `Width` и `Height`). Попробуйте справа от кнопки разместить еще одну кнопку, затем еще одну. У вас должно получиться то же, что показано на рис. 11.21.

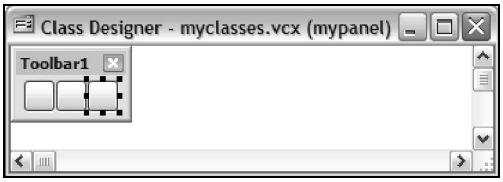


Рис. 11.21. Вид панели инструментов в Конструкторе классов

Обратите внимание, что кнопки панели не имеют названий. А посмотрев окно свойств, вы увидите, что свойство `ShowTips` панели установлено в `True` — т. е. по умолчанию разрешено использование всплывающих окон-подсказок.

Разместите на кнопках пиктограммы, в свойства `ToolTip Text` введите текст, который должен появляться при перемещении мыши над кнопкой. В методах `Click` напишите необходимый код, который будет выполняться при нажатии на кнопку. Обычно кнопки панелей инструментов связаны с пунктами меню, поэтому в методе `Click` вызывайте те же процедуры, которые вызываются из соответствующих пунктов меню (как создать свое меню, вы узнаете в следующей главе). Вот, пожалуй, и все. Панель инструментов готова.

Но несколько непривычно слишком близкое расположение кнопок. Все попытки както изменить расстояние между ними не приводят к успеху. Посмотрев в окно свойств, вы увидите, что свойства `Top` и `Left` кнопок действительно недоступны. Единственный способ изменить расстояние между кнопками — это вставить между ними специальный инструмент, который так и называется — Разделитель (Separator). Его место в панели инструментов **Form Controls** показано на рис. 11.22.

Рис. 11.22. Инструмент **Separator** в панели инструментов **Form Controls**

Вставьте этот компонент между кнопками на создаваемой панели инструментов. Если вы все сделаете правильно, то панель инструментов примет вид, представленный на рис. 11.23.

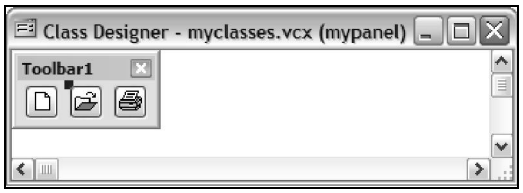


Рис. 11.23. Применение разделителей в панели инструментов

Когда в панели инструментов присутствует более одного управляющего элемента, вы можете изменять ее вид, размещая эти элементы по горизонтали, вертикали или в виде таблицы — т. е. так же, как вы управляете обычными панелями инструментов.

Что еще можно сделать? Например, установить для свойств `SpecialEffect` кнопок значение, равное 2 (`Hot Tracking`); тогда контур кнопки будет появляться только при наведении на нее указателя мыши.

Для создания объекта — экземпляра класса `ToolBar` воспользуйтесь уже известными вам функциями `CREATEOBJECT()` и `NEWOBJECT()`. Из всех методов этого объекта интерес представляет только метод `Dock()`. Этот метод определяет положение панели инструментов после создания. Вот его синтаксис:

```
ToolBarObject.Dock(nLocation [, X, Y])
```

Параметр `nLocation` определяет положение панели инструментов в окне формы верхнего уровня или главном окне Visual FoxPro. Его допустимые значения приведены в табл. 11.1.

Таблица 11.1. *Значения параметра* `nLocation` *метода* `Dock()`

nLocation	Описание
-1	Панель инструментов не пристыковывается
0	Пристыковывается к верхнему горизонтальному краю окна
1	Пристыковывается к левому краю окна
2	Пристыковывается к правому краю окна
3	Пристыковывается к нижнему краю окна

Параметры *X*, *Y* определяют позицию левого верхнего угла панели инструментов, если параметр `nLocation` равен –1.

В листинге 11.5 показан код, создающей панель инструментов. Предполагается, что класс панели расположен в библиотеке `MyClasses` и имеет имя `MyPanel`.

```
PUBLIC oTool
SET CLASSLIB TO myclasses
oTool = CREATEOBJECT('MyPanel')
oTool.Show()       && Панель запускается как немодальная форма
oTool.Dock(0)      && и пристыковывается к верхней границе окна
```

Если главной формой вашего приложения является форма верхнего уровня, то присвойте свойству `ShowWindow` класса панели инструментов значение 1 (In Top Level Form) для того, чтобы панель инструментов пристыковалась к вашей главной форме.

Совместное использование меню и панелей инструментов рассматривается в следующей главе.

Редактирование объявлений свойств и методов класса

Как и Конструктор форм, Конструктор классов позволяет вам выполнять редактирование объявлений свойств и методов класса. Вы можете изменить имя свойства или метода, определить уровень видимости, либо удалить метод из класса. Для перехода в режим редактирования выберите в главном меню пункт **Class**, и в выпавшем меню — пункт **Edit Property/Method...**. Другой способ — это использование окна свойств. Установите указатель на строку с именем метода или свойства и нажмите на правую кнопку мыши. В появившемся меню выберите пункт **Edit Property/Method...**. В результате этих манипуляций Конструктор классов выведет на экран диалоговое окно для редактирования объявлений свойств и методов (рис. 11.24).

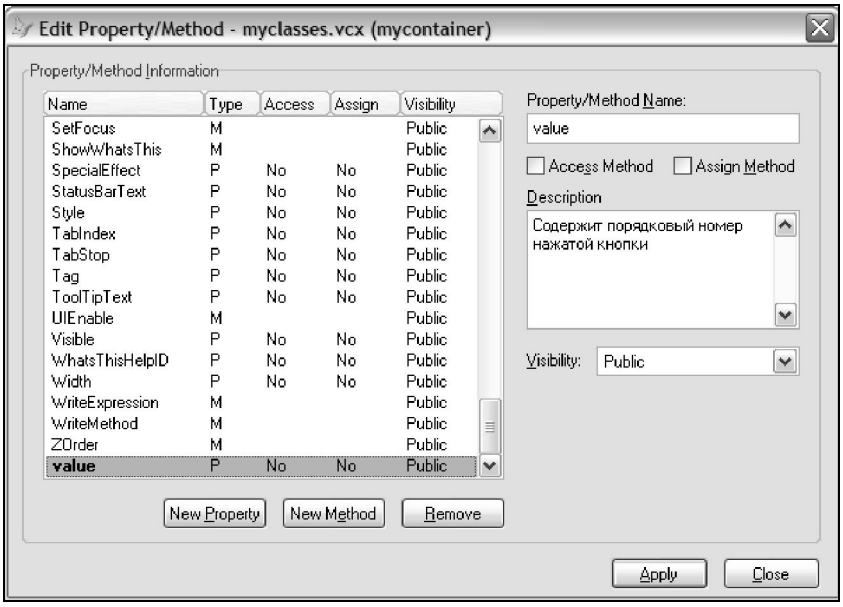


Рис. 11.24. Окно редактирования объявлений свойств и методов

Это окно отличается от аналогичного окна только возможностью изменения значения уровня видимости — для этого предназначен расположенный справа выпадающий список **Visibility**. Поэтому не будем останавливаться на рассмотрении кнопок **New Property**, **New Method** и **Remote** — с действиями, выполняемыми при нажатии на эти кнопки, вы познакомились, изучая Конструктор форм *(см. главу 4)*. Отмечу только, что вы можете установить уровень видимости для любого (в том числе и унаследованного от базового класса) свойства или метода.

Обозреватель классов

Вы вправе предположить, что все действия по созданию классов в Конструкторе классов заставляют Visual FoxPro генерировать код, который и будет исполняться при работе вашего приложения. Это действительно так. Посмотреть этот код, кроме того, и структуру класса, позволяет специальный инструмент, который называется Обозреватель классов (Class Browser).

Для вызова Обозревателя классов выберите в главном меню пункт **Tools**, и в выпавшем меню — пункт **Class Browser**. Альтернативный способ вызова Обозревателя классов — это щелчок по кнопке **Class Browser** стандартной панели инструментов Visual FoxPro. Расположение этого компонента на панели показано на рис. 11.25.

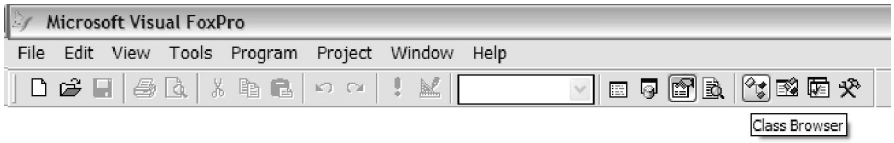


Рис. 11.25. Кнопка **Class Browser** на панели инструментов **Standard**

Окно Обозревателя классов показано на рис. 11.26.

В верхней части окна Обозревателя классов располагаются инструментальные кнопки, назначение которых приведено в табл. 11.2.

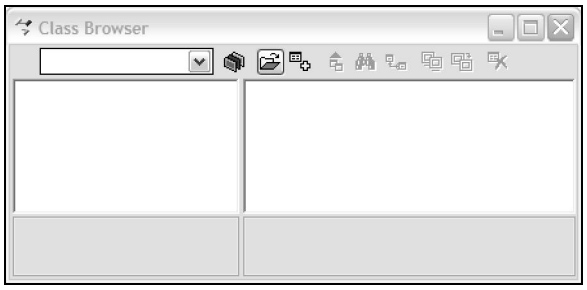


Рис. 11.26. Окно Обозревателя классов

Таблица 11.2. *Кнопки инструментальной панели Обозревателя классов*

Кнопка	Название	Выполняемое действие
	**Component Gallery**	Переключение между окном Обозревателя классов и окном Галереи компонентов
	**Open**	Открывает для просмотра и изменения новую библиотеку классов, форму, проект или программный файл, содержащий определения классов
	**View Additional File**	Добавляет существующую библиотеку классов или форму к списку классов
	**View Class Code**	Выводит окно просмотра кода класса
	**Find**	Поиск определенного текста в именах класса, описаниях класса, именах и описаниях реквизитов
	**New Class**	Создает класс-потомок для выбранного класса
	**Rename**	Изменяет имя выбранного класса, свойства или метода. **Предостережение:** Переименование класса может аннулировать форму или класс, который в настоящий момент не отображается в окне Обозревателя классов и который вложен в этот класс. Изменение имени свойства или метода может привести к потере кода

Таблица 11.2 *(окончание)*

Кнопка	Название	Выполняемое действие
	**Redefine**	Заменяет класс-родитель указанного класса
	**Clean Up Class Library**	Упаковывает библиотеку классов (выполняет команду `PACK MEMO` для таблиц Конструктора классов)

Нажмите кнопку **Open**. В появившемся стандартном диалоге открытия файла выберите интересующую вас библиотеку классов. Так, если вы в процессе чтения этой главы упорно создавали библиотеку классов `MyClasses`, то выберите файл myclasses.vcx. Обратите внимание на то, что вы можете открыть в Обозревателе классов не только библиотеку классов, но и форму (файлы типа SCX), проект (файлы типа PJX) и даже программный файл, содержащий определения классов. Действительно, и формы, и сам проект представляют собой классы Visual FoxPro.

Окно Обозревателя классов с загруженной библиотекой myclasses.vcx показано на рис. 11.27.

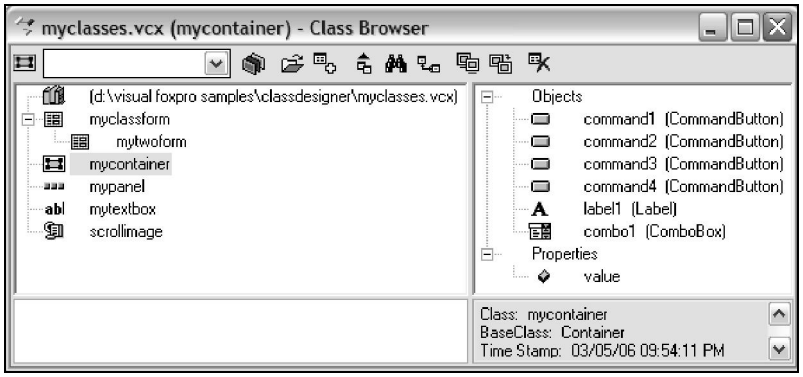


Рис. 11.27. Окно Обозревателя классов с загруженной библиотекой классов

В левой области окна Обозревателя отображаются классы, включенные в библиотеку классов; используемая здесь иерархическая структура представления данных позволяет отследить наследование, в частности, видно, что класс `mytwoform` является наследником класса `myclassform`. Перемещаясь по узлам с именами классов, в правой части окна вы увидите список объектов, включенных в этот класс, а также перечень добавленных вами свойств и методов. В нижней части окна выводится информация о классе: имя класса, базовый Visual FoxPro класс и дата последнего изменения.

Если вы дважды щелкнете мышью по имени класса, то Обозреватель вызовет Конструктор классов, в окно которого будет загружен выбранный класс.

При щелчке мышью по имени свойства в правой области окна выводится окно, в котором вы увидите значение по умолчанию для выбранного свойства; аналогично,

щелчок по наименованию метода загружает Конструктор классов и выводит окно редактора кода, содержащее код этого метода.

Наиболее интересной является кнопка **View Class Code**. При щелчке мышью по этой кнопке выводится окно, содержащее полный код класса. Вы можете копировать фрагменты этого кода; вы можете распечатать код на принтере; но вы не можете в этом окне редактировать код класса.

В листинге 11.6 показан сгенерированный Обозревателем классов код рассмотренного нами ранее класса `MyContainer`.

```
DEFINE CLASS mycontainer AS container
    Width = 513
    Height = 33
    value = 0
    Name = "mycontainer"
    ADD OBJECT command1 AS commandbutton WITH ;
        Top = 1, ;
        Left = 3, ;
        Height = 25, ;
        Width = 85, ;
        Caption = "Добавить", ;
        Name = "Command1"
    ADD OBJECT command2 AS commandbutton WITH ;
        Top = 1, ;
        Left = 87, ;
        Height = 25, ;
        Width = 85, ;
        Caption = "Изменить", ;
        Name = "Command2"
    ADD OBJECT command3 AS commandbutton WITH ;
        Top = 1, ;
        Left = 171, ;
        Height = 25, ;
        Width = 85, ;
        Caption = "Удалить", ;
        Name = "Command3"
    ADD OBJECT command4 AS commandbutton WITH ;
        Top = 1, ;
        Left = 416, ;
        Height = 25, ;
        Width = 85, ;
        Caption = "Печатать", ;
        Name = "Command4"
    ADD OBJECT label1 AS label WITH ;
        AutoSize = .T., ;
```

```
        BackStyle = 0, ;
        Caption = "Фильтр", ;
        Height = 17, ;
        Left = 262, ;
        Top = 5, ;
        Width = 44, ;
        Name = "Label1"
    ADD OBJECT combo1 AS combobox WITH ;
        Height = 25, ;
        Left = 309, ;
        Top = 1, ;
        Width = 100, ;
        Name = "Combo1"
    PROCEDURE command1.Click
        this.Parent.value = 1
        this.Parent.Click()
    ENDPROC
    PROCEDURE command2.Click
        this.Parent.value = 2
        this.Parent.Click()
    ENDPROC
    PROCEDURE command3.Click
        this.Parent.value = 3
        this.Parent.Click()
    ENDPROC
    PROCEDURE command4.Click
        this.Parent.value = 4
        this.Parent.Click()
    ENDPROC
ENDDEFINE
```

Замечание

Код, генерируемый Обозревателем классов, мягко говоря, не всегда соответствует действительности. Например, для формы, на которой размещен управляющий элемент `Grid`, будет сгенерирован неверный код. Это замечание относится ко всем управляющим элементам контейнерного типа (например, `PageFrame`). В общем случае вы должны иметь отдельные классы для объектов — контейнеров со всеми включенными в них управляющими элементами и при объявлении класса формы добавлять в нее уже эти классы.

Подробно с командами объявления класса вы познакомитесь в следующем разделе этой главы, а сейчас рассмотрим остальные элементы инструментальной панели Обозревателя классов.

На рис. 11.27 слева от выпадающего списка вы видите пиктограмму. Эта пиктограмма появляется, когда вы выделяете класс в иерархическом списке классов. Вы можете перетащить эту пиктограмму на форму (для этого форма должна быть открыта в окне

Конструктора форм или классов), и Visual FoxPro создаст на ней объект — экземпляр класса.

Выпадающий список содержит перечень базовых классов Visual FoxPro. Выбранное из списка значение будет использоваться как фильтр по базовому классу. Так, если вы выберете в этом списке **Form**, то в иерархическом списке останутся только классы, порожденные от базового класса `Form`.

Программное создание классов

В листинге 11.6 вы видели код класса, сгенерированный Обозревателем классов. В этом разделе вы научитесь создавать классы программно, используя набор команд, предоставляемых Visual FoxPro.

Программный способ создания классов является наиболее универсальным и не имеет никаких ограничений; более того, только программно доступен класс, базовый класс `Session`, который более подробно рассматривается в *главе 17*.

Определение класса

Команды `DEFINE CLASS` и `ENDDEFINE` образуют программный блок, весь код внутри которого относится к определению класса. Класс может быть определен в любом программном файле (типа PRG). Как до команд определения класса, так и после могут следовать определения процедур и функций, а так же исполняемые команды и выражения. Существует только одно ограничение: класс не может быть определен внутри цикла или условного оператора.

В листинге 11.7 приведен синтаксис определения класса (некоторые предложения и ключевые слова, определяющие структуру класса COM-объекта, здесь не приводятся).

```
DEFINE CLASS ClassName1 AS ParentClass [OF ClassLibrary]
   [[PROTECTED | HIDDEN] PropertyName1, PropertyName2 ...]
   [ADD OBJECT [PROTECTED] ObjectName AS ClassName2 [NOINIT] [WITH
                                                      cPropertylist]]
   [[PROTECTED | HIDDEN] FUNCTION | PROCEDURE Name[_ACCESS |_ASSIGN]
      [([ParamName | ArrayName[]) | PARAMETERS | LPARAMETERS ParamName]
          cStatements
   [ENDFUNC | ENDPROC]
ENDDEFINE
```

Вы должны соблюдать описанную в листинге 11.7 последовательность предложений класса. Так, сначала, после команды `DEFINE CLASS`, следуют объявления свойств, затем команды `ADD OBJECT` и, наконец, объявления методов.

Команда *DEFINE CLASS*

Команда объявляет, что все последующие строки кода, вплоть до команды ENDDEFINE, составляют тело класса.

```
DEFINE CLASS ClassName1 AS ParentClass [OF ClassLibrary]
```

Параметры:

ClassName1
имя создаваемого класса

ParentClass
имя базового класса Visual FoxPro или пользовательского класса.

ClassLibrary
полная спецификация файла библиотеки файлов или программного файла, в котором определен класс-родитель. Если родитель определен в том же самом программном файле, что и определяемый класс, параметр можно не использовать.

Объявление свойств класса

Объявления свойств должны следовать непосредственно за командой объявления класса.

```
[[PROTECTED | HIDDEN] PropertyName1, PropertyName2 ...]
```

Ключевые слова PROTECTED и HIDDEN определяют уровень видимости свойства. PROTECTED (защищенный) запрещает доступ для считывания или изменения значений свойства извне определения класса. Методы и события, определенные в классе или дочернем классе, могут обращаться к защищенным свойствам. HIDDEN (скрытый) так же запрещает доступ и изменение свойства извне определения класса. Свойство недоступно из дочерних классов.

Если вы не используете ни одно из этих ключевых слов, то уровень видимости свойства — PUBLIC.

Вы можете объявить произвольное количество новых свойств, а также изменить уровень видимости свойств, наследуемых от класса-родителя.

Свойства с уровнем видимости PUBLIC можно связать с методами _ASSIGN и _ACCESS. В этом случае при попытке записи данных в свойство будет вызываться метод _ASSIGN, а при попытке чтения — метод _ACCESS, т. е. непосредственный доступ к свойству извне класса будет невозможен.

> ***Замечание***
>
> Методы _ASSIGN и _ACCESS могут быть связаны с любым свойством класса, в том числе и имеющим уровень видимости PROTECTED и HIDDEN, но их использование для этих уровней не актуально, т. к. основная масса ошибок возникает именно при обращении к свойствам извне класса.

Если вы хотите присвоить свойству значение по умолчанию, то делайте это в отдельной строке кода, например:

```
PROTECTED Alpha
Alpha = 10
```

Вы можете объявить массив как свойство класса. В отличие от обычного свойства, вы не можете инициализировать массив иначе, чем присвоив значения его элементам в одном из методов класса, например, в методе `Init`. Попытка присвоить значения в секции объявления свойств класса приведет к ошибке.

Объявление методов класса

Объявления методов должны следовать после всех других объявлений класса. Метод объявляется либо как процедура, либо как функция, хотя любой метод может возвращать значение, т. е. по сути является функцией, независимо от использованного при его объявлении ключевого слова.

```
[[PROTECTED | HIDDEN] FUNCTION | PROCEDURE Name[_ACCESS |_ASSIGN]
   [([ParamName | ArrayName[]) | PARAMETERS | LPARAMETERS ParamName ]
      cStatements
[ENDFUNC | ENDPROC]
```

Ключевые слова `PROTECTED` и `HIDDEN` определяют уровень видимости метода *(см. выше)*. Если вы не используете ни одно из этих ключевых слов, то уровень видимости метода — `PUBLIC`.

Ключевые слова `PROCEDURE` или `FUNCTION` определяют начало кода метода, а команды `ENDPROC` или `ENDFUNC` — его окончание. Как и в случае с обычными процедурами и функциями, вы можете не использовать команды `ENDPROC` и `ENDFUNC`, т. к. Visual FoxPro будет считать концом метода либо следующую команду `PROCEDURE` или `FUNCTION`, либо команду `ENDDEFINE`.

Параметр `Name` — это имя метода. Если используются ключевые слова `_ACCESS` или `_ASSIGN`, то `Name` — это имя свойства класса, запись и чтение которого контролируются соответствующими методами. Например, если вы хотите объявить метод `_ASSIGN` для свойства `MyValue`, то сделать это нужно так:

```
PROCEDURE MyValue_ASSIGN
```

Для указания списка получаемых методом параметров может использоваться один из следующих форматов:

```
PROCEDURE MyMethod(Param1, Param2, ...)
```

или

```
PROCEDURE MyMethod
PARAMETERS Param1, Param2, ...
```

Первый формат равнозначен применению команды `LPARAMETERS`, т. е. все получаемые параметры имеют локальную область видимости. При использовании второго формата получаемые параметры имеют область видимости `PRIVATE`.

Массивы передаются в методы по ссылке. Не забывайте использовать команду EXTERNAL ARRAY для объявления параметра как массива, как, например, в следующем фрагменте кода:

```
PROCEDURE Alpha(taMyArray)
EXTERNAL ARRAY taMyArray
```

Если вы перегружаете метод, унаследованный от класса-родителя, то в объявлении такого метода нужно перечислить все получаемые им параметры в том же порядке, в каком они перечисляются в методе класса-родителя. Например, метод-обработчик события базового класса Form KeyPress получает два параметра: код нажатой клавиши и флаг, определяющий, была при этом нажата одна из клавиш <Ctrl>, <Shift> или <Alt>. При перегрузке этого метода в своем классе вы должны объявлять его следующим образом:

```
PROCEDURE KeyPress(nKeyCode, nShiftAltCtrl)
```

Значения nKeyCode и nShiftAltCtrl будут переданы в метод при его вызове.

Добавление объектов в класс-контейнер

Для добавления объектов (управляющих элементов) в класс-контейнер применяется команда ADD OBJECT.

```
[ADD OBJECT [PROTECTED] ObjectName AS ClassName2 [NOINIT] [WITH
                                                    cPropertylist]]
```

Параметры:

ADD OBJECT [PROTECTED] ObjectName AS ClassName2

Определяет объект ObjectName, который наследуется от класса ObjectName. В качестве класса-родителя может использоваться как базовый класс Visual FoxPro, так и пользовательский класс или ActiveX-компонент. Необязательное ключевое слово PROTECTED предотвращает доступ для изменения свойств объекта извне определяемого класса или подкласса.

NOINIT

Необязательное ключевое указывает, что метод Init добавляемого объекта не выполняется.

WITH

Определяет список свойств и их значений для объекта, которые вы добавляете в определение класса.

cPropertyList

Перечисленные через запятую операторы присваивания, в которых свойствам объектов присваиваются значения "по умолчанию".

В листинге 11.8 показан пример создания класса формы и размещения на ней командной кнопки.

```
DEFINE CLASS MyForm AS Form
    Width = 300
    Height = 80
    ADD OBJECT Command1 as CommandButton WITH ;
        Left = 200, Top = 52, Width = 95, Height = 25, ;
        Caption = 'Закрыть'
ENDDEFINE
```

Вы можете устанавливать значения только для свойств, объявленных (существующих) в добавляемом объекте. Visual FoxPro не контролирует правильность ввода наименований свойств, поэтому если вы ошибетесь в имени какого-либо свойства, то это ошибка проявится только на этапе выполнения приложения.

Для перегрузки метода добавляемого объекта вы должны после ключевого слова, определяющего тип метода (`PROCEDURE` или `FUNCTION`), указать имя объекта и, через точку, наименование метода этого объекта. Так, для того чтобы перегрузить метод `Click` добавленной в класс `MyForm` командной кнопки, добавьте в класс следующий код:

```
PROCEDURE Command1.Click
    Строки_кода_метода
ENDPROC
```

В листинге 11.9 показан пример программного создания формы.

```
PUBLIC oForm
oForm = CREATEOBJECT('MyForm')  && Создать объект Форма
oForm.Show()                    && Показать форму на экране
*
* Объявление класса формы
*
DEFINE CLASS MyForm AS Form
    Width = 300
    Height = 80
    ADD OBJECT Command1 as CommandButton WITH ;
        Left = 200, Top = 52, Width = 95, Height = 25, ;
        Caption = 'Закрыть'
    PROCEDURE Click()
        = MESSAGEBOX('Вы щелкнули по мне!')
    ENDPROC
    PROCEDURE Command1.Click()
        thisform.release
    ENDPROC
ENDDEFINE
```

Первые три строки кода — это команды, которые создают форму и выводят ее на экран.

Из объявления класса видим, что форма будет иметь размеры 300 на 80 пикселов; т. к. значения для свойств `Top` и `Left` формы не устанавливаются, то используются их значения по умолчанию.

В форму добавляется объект — управляющий элемент `Command1`, создаваемый из базового класса `CommandButton`. Определяются заголовок, положение и размеры командной кнопки.

Перегружается метод `Click` формы. Теперь при щелчке мышью по форме будет выводиться окно с сообщением: "Вы щелкнули по мне!". Далее перегружается метод `Click` объекта `Command1` — обратите внимание на то, что используется именно то имя объекта, которое вы дали ему в команде `ADD OBJECT`.

Сохраните этот код как программный файл. Запустите его на выполнение. Убедитесь, что форма ведет себя именно так, как описано в ее классе.

> ***ЗАМЕЧАНИЕ***
>
> Командой `ADD OBJECT` можно добавлять в класс-контейнер только объекты — управляющие элементы, наследуемые как от базовых классов Visual FoxPro, так и от компонентов ActiveX. Программно создание класса-потомка ActiveX-компонента, а также добавление в контейнер ActiveX-компонентов рассматривается в *главе 18*.

Методы *Init*, *Destroy* и *Error*

Все пользовательские классы наследуют свойства и методы класса-родителя. Здесь вы познакомитесь с назначением и применением наследуемых методов `Init`, `Destroy` и `Error`, которые вы также можете перегружать.

Метод *Init*

Метод `Init` выполняется после инициализации объекта — экземпляра класса. К моменту выполнения этого метода объект создан, а также созданы все размещаемые на нем командой `ADD OBJECT` управляющие элементы. Особенностью метода `Init` является возможность получать параметры, передаваемые ему функциями `CREATEOBJECT()` и `NEWOBJECT()`, а для динамически создаваемых управляющих элементов — методами `AddObject` и `NewObject` объектов-контейнеров. Список получаемых параметров вы можете перечислить как в строке с именем метода, так и используя команду `LPARAMETERS`:

```
PROCEDURE Init(Parameter1 [, Parameter2, ets... ])
```

или

```
PROCEDURE Init
LPARAMETER Parameter1 [, Parameter2, ets... ]
```

Если вы хотите использовать переданные в метод параметры, то вы должны сохранить их в свойствах класса.

Если вы используете для завершения выполнения кода метода команду

```
RETURN = .F.
```

то объект разрушается, объектная ссылка не создается.

После отработки метода `Init` объект переходит в состояние ожидания событий.

Метод *Destroy*

Этот метод обрабатывает событие `Destroy` и всегда выполняется перед разрушением объекта. На момент возникновения этого события все вложенные объекты и свойства существуют.

Метод *Error*

Метод `Error` вызывается, когда при выполнении кода в одном из методов класса возникает ошибка. Метод получает следующие параметры:

`nError`

номер ошибки Visual FoxPro

`cMethod`

имя метода, в котором произошла ошибка (или имя вызванной из метода процедуры, при выполнении которой произошла ошибка)

`nLine`

строка кода, вызвавшая ошибку.

Если вы не перегружаете этот метод, то при возникновении ошибки во время выполнения одного из методов класса будет выведено окно со следующим сообщением (рис. 11.28).

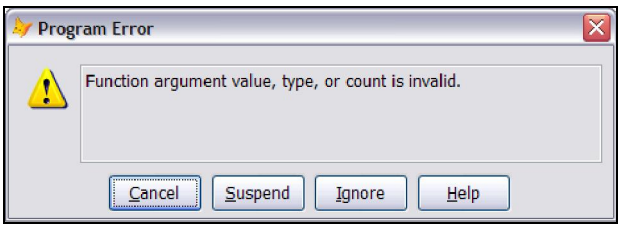


Рис. 11.28. Системное сообщение об ошибке выполнения кода метода

При переопределении метода `Error` в простейшем случае вы можете вызвать из него функцию `MESSAGEBOX()`, которая выведет диалоговое окно с подробной информацией об ошибке. Пример перегрузки метода показан в листинге 11.10.

```
PROCEDURE Error(nError, cMethod, nLine)
    = MESSAGEBOX('Ошибка ' + LTRIM(STR(nError)) + ' в методе ' + ;
```

```
                cMethod + ' в строке ' + LTRIM(STR(nLine)))
ENDPROC
```

В результате при возникновении ошибки во время выполнения одного из методов класса будет выведено окно со следующим сообщением (рис. 11.29).

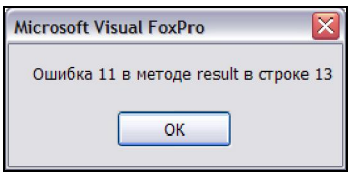


Рис. 11.29. Результат выполнения метода `Error` класса

Использование метода `Error` позволяет получить подробную информацию о месте, причине и характере ошибки. Все зависит только от написанного вами кода.

Подключение программных модулей к приложению

Если вы объявляете классы в программных модулях (файлах типа PRG), то вы должны использовать команду

```
SET PROCEDURE TO имя_файла [ ADDITIVE ]
```

для указания Visual FoxPro на необходимость просматривать эти файлы при выполнении функции (метода), создающей объект.

Ключевое слово `ADDITIVE` используется в том случае, если вы подключаете несколько программных модулей.

Класс *Custom*

Класс `Custom` — это один из "легких" базовых классов Visual FoxPro, позволяющий создавать не визуальные пользовательские классы. Он имеет ограниченный по сравнению с визуальными классами набор свойств и методов, с которыми вы можете познакомиться в справочной документации. На его основе вы можете создавать классы, решающие различные задачи, не требующие визуализации.

Вы не можете в объявлении этого класса использовать предложение `ADD OBJECT`; тем не менее класс содержит методы `AddObject`, `NewObject` и `RemoveObject`, что позволяет динамически добавлять в него и удалять невизуальные объекты, например таймер.

В листинге 11.11 показан код класса `ConvertAmount`, который предназначен для создания объектов, выполняющих преобразование значения суммы (числа) в наименование, например, число `1286.55` будет преобразовано в строку `Одна тысяча двести восемьдесят шесть рублей 55 коп.` В качестве базового класса используется класс `Custom`.

```
DEFINE CLASS ConvertAmount as Custom
PROTECTED aHundreds[9], aTens[9], aUnits[19]
* Массив наименований сотен
aHundreds[1] = 'сто'
aHundreds[2] = 'двести'
aHundreds[3] = 'триста'
aHundreds[4] = 'четыреста'
aHundreds[5] = 'пятьсот'
aHundreds[6] = 'шестьсот'
aHundreds[7] = 'семьсот'
aHundreds[8] = 'восемьсот'
aHundreds[9] = 'девятьсот'
* Массив наименований десятков
aTens[2] = 'двадцать'
aTens[3] = 'тридцать'
aTens[4] = 'сорок'
aTens[5] = 'пятьдесят'
aTens[6] = 'шестьдесят'
aTens[7] = 'семьдесят'
aTens[8] = 'восемьдесят'
aTens[9] = 'девяносто'
* Массив наименований единиц
aUnits[3] = 'три'
aUnits[4] = 'четыре'
aUnits[5] = 'пять'
aUnits[6] = 'шесть'
aUnits[7] = 'семь'
aUnits[8] = 'восемь'
aUnits[9] = 'девять'
aUnits[10] = 'десять'
aUnits[11] = 'одиннадцать'
aUnits[12] = 'двенадцать'
aUnits[13] = 'тринадцать'
aUnits[14] = 'четырнадцать'
aUnits[15] = 'пятнадцать'
aUnits[16] = 'шестнадцать'
aUnits[17] = 'семнадцать'
aUnits[18] = 'восемнадцать'
aUnits[19] = 'девятнадцать'

FUNCTION Convert(tnValue)
   LOCAL lcSumma, lcResult, lnDec
   IF tnValue > 2147483647 .or. tnValue <= 0
      RETURN "Число вне допустимых пределов"
```

```
   ENDIF
   lcSumma = STR(INT(tnValue),12)
   lcResult = this.NumToString(SUBSTR(lcSumma,1,3),"миллиард",3)
   lcResult = lcResult+this.NumToString(SUBSTR(lcSumma,4,3),"миллион",3)
   lcResult = lcResult+this.NumToString(SUBSTR(lcSumma,7,3),"тысяч",2)
   lcResult = lcResult+this.NumToString(SUBSTR(lcSumma,10,3),"рубл",1)
   lnDec = 100 * (tnValue — INT(tnValue))
   RETURN UPPER(LEFT(lcResult,1)) + SUBSTR(lcResult,2) + ;
          TRANSFORM(lnDec, "@L 99") + " коп"
ENDFUNC
PROTECTED FUNCTION NumToString(tcValue, tcName, tnFlag)
   LOCAL lcResult, lnHundreds, lnTens, lnUnits, lcName, lcExt
   IF tcValue = "   "
      IF tnFlag = 1
         RETURN "Ноль рублей "
      ENDIF
      RETURN ""
   ENDIF
   IF tnFlag = 2
      this.aUnits[1] = "одна"
      this.aUnits[2] = "две"
   ELSE
      this.aUnits[1] = "один"
      this.aUnits[2] = "два"
   ENDIF
   lcResult = ""
   lnHundreds = VAL(LEFT(tcValue,1))  && Выделяем количество сотен
   lnTens = VAL(SUBSTR(tcValue,2,1))  && ... десятки
   lnUnits = VAL(SUBSTR(tcValue,3,1)) && ... единицы (от 1 до 19)
   IF lnHundreds > 0
      lcResult = lcResult + this.aHundreds[lnHundreds] + " "
   ENDIF
   IF lnTens > 1
      lcResult = lcResult + this.aTens[lnTens] + " " + ;
                 this.aUnits[lnUnits] + " "
   ELSE
      lcResult = lcResult + this.aUnits[10 * lnTens + lnUnits] + " "
   ENDIF
   lcResult = lcResult + tcName
*
* Формирование окончаний
*
   lcExt = ""
   DO CASE
      CASE tnFlag = 1  && Рубли (единицы)
         lcExt = "ей"  && По умолчанию "рублей"
         IF lnTens != 1
```

```
            DO CASE
               CASE lnUnits = 1 && 1; 21; 31 ...
                  lcExt = "ь" && "рубль"
               CASE lnUnits > 1 .and. lnUnits < 5
                  lcExt = "я"
            ENDCASE
         ENDIF
      CASE tnFlag = 2 && Тысячи
         IF lnTens != 1
            DO CASE
               CASE lnUnits = 1
                  lcExt = "а"
               CASE lnUnits > 1 .and. lnUnits < 5
                  lcExt = "и"
            ENDCASE
         ENDIF
      CASE tnFlag = 3 && Миллионы, миллиарды
         lcExt = "ов" && По умолчанию "миллионов"
         IF lnTens != 1
            DO CASE
               CASE lnUnits = 1
                  lcExt = ""
               CASE lnUnits > 1 .and. lnUnits < 5
                  lcExt = "а"
            ENDCASE
         ENDIF
   ENDCASE
   RETURN lcResult + lcExt + " "
ENDFUNC
ENDDEFINE
```

В классе объявляются защищенные массивы, содержащие наименования чисел, и два метода — открытый метод `Convert`, которому передается преобразуемое число, и защищенный `NumToString`. Как выполняется преобразование числа в строку, понятно из кода метода.

Для использования класса нужно создать объект и вызвать его метод `Convert`:

```
oCnv = CREATEOBJECT("ConvertAmount")
cString = oCnv.Convert(1286.55)
```

Управление временем жизни объектов

Каждый объект — экземпляр класса связан либо с переменной, выполняющей роль объектной ссылки (указателя на объект), либо с объектом-контейнером.

Если вы создаете объект, используя функции `CREATEOBJECT()` или `NEWOBJECT()`, то время жизни объекта определяется временем жизни переменной, которой вы присваива-

ете ссылку на объект. Если переменная выходит из области видимости либо уничтожается явно (например, командой RELEASE), то уничтожается и связанный с ней объект. Но если существует несколько ссылок на объект, то объект будет уничтожен только после уничтожения последней ссылки:

```
oObj1 = CREATEOCJECT('MyClass')
oObj2 = oObj1
RELEASE oObj1      && После уничтожения этой переменной объект еще
                   && существует
RELEASE oObj2      && Теперь объект уничтожен
```

Если существует внешняя ссылка на вложенный объект объекта-контейнера, то, если при уничтожении объекта эта ссылка будет существовать, это приведет к тому, что объект не сможет уничтожить вложенный объект, и, следовательно, и он, и не уничтоженный вложенный объект останутся в памяти, что приведет к утечке памяти и, возможно, ошибкам времени выполнения.

Если в качестве ссылки вы используете свойство существующего объекта, то создаваемый объект будет уничтожен при уничтожении объекта, содержащего это свойство. Уничтожение объекта происходит также при присваивании переменной (или свойству), содержащей указатель на объект, нового значения:

```
Obj = CREATEOBJECT('ClassName')
Obj = NULL    && или, например, Obj = .F.
```

Вторая строка кода уничтожает объект — экземпляр класса *ClassName*.

Объект также будет уничтожен, если в переменную, содержащую объектную ссылку, будет записана ссылка на другой объект:

```
Obj = CREATEOBJECT('ClassName1')
Obj = CREATEOBJECT('ClassName2')
```

Вторая строка кода уничтожает объект — экземпляр класса ClassName1.

Размещаемые в объектах-контейнерах объекты — управляющие элементы уничтожаются после выполнения метода Destroy контейнера, т. е. на момент вызова этого метода все управляющие элементы еще существуют. Для принудительного удаления объекта из объекта-контейнера используется метод RemoveObject.

Перед уничтожением все объекты генерируют событие Destroy, метод обработки которого в своих классах вы можете перегрузить.

Класс *Collection*

Что такое *коллекции*? Коллекция — это инструмент для объединения связанных объектов в группы. Например, объект Form поддерживает коллекцию объектов — управляющих элементов Controls и имеет соответствующие методы для добавления новых элементов в коллекцию или удаления их из нее. Вы можете обратиться к любому элементу в коллекции Controls по его индексу, например:

```
thisform.Controls[nIndex].Name
```

Свойство `ControlCount` формы содержит значение, равное количеству объектов в коллекции. При удалении объекта из коллекции, независимо от его положения внутри коллекции, значение свойства `ControlCount` уменьшается на единицу, а индексы всех объектов, следующих в коллекции за удаляемым объектом, также уменьшаются на единицу.

Коллекции позволяют организовать объектно-ориентированный подход к хранению данных. Кроме того, в отличие от массивов, коллекции не требуют повторного задания размеров в случае добавления или удаления элементов.

В Visual FoxPro для создания коллекций используется базовый класс `Collection`. В объекте — экземпляре этого класса вы можете хранить не только объекты, но и данные любых других типов: числовые, символьные, логические, и даже массивы. К каждому элементу коллекции можно обратиться как по его индексу, так и по ключу — строке, как и индекс, однозначно идентифицирующей этот элемент. В отличие от массивов, вы можете добавлять новые элементы в коллекцию, размещая их между любыми уже существующими элементами.

Создание

Для создания объекта — экземпляра класса `Collection` можно использовать функции `CREATEOBJECT()` или `NEWOBJECT()`, например:

```
oCollection = CREATEOBJECT('Collection')
```

Вы можете добавлять объекты `Collection` в свои классы, используя команду `ADD OBJECT`, либо динамически во время выполнения в уже существующие объекты-контейнеры при помощи методов `AddObject` или `NewObject`.

Вы также можете использовать класс `Collection` как класс-родитель при объявлении собственных классов.

Свойства

Свойство `Count` содержит количество элементов коллекции. Оно доступно только для чтения:

```
nCount = oCollection.Count
```

Свойство `KeySort` определяет, в какой последовательности Visual FoxPro перечисляет элементы коллекции в цикле `FOR EACH` (табл. 11.3).

Таблица 11.3. *Значения свойства `KeySort` класса `Collection`*

KeySort	Описание
0	Сортировка по возрастанию значения индекса (используется по умолчанию)

1	Сортировка по убыванию значения индекса
2	Сортировка по возрастанию значения ключа
3	Сортировка по убыванию значения ключа

В листинге 11.12 показано применение свойства `KeySort`.

```
oCollection.KeySort = 1
FOR EACH oItem IN oCollection
    ? IIF(VARTYPE(oItem) = 'O', oItem.Name, oItem)
ENDFOR
```

Остальные свойства класса `Collection` — общие для всех базовых классов Visual FoxPro; при необходимости вы найдете их описание в справочной документации.

Методы

Из всех методов класса мы рассмотрим здесь методы `Add`, `Item`, `GetKey` и `Remove`. Описание остальных методов класса вы можете найти в справочной документации.

Метод *Add*

Метод добавляет в коллекцию новый элемент. Вот его синтаксис:

```
Collection.Add( eItem [, cKey [, [eBefore |, eAfter ]]] )
```

Параметр *eItem* представляет собой выражение любого типа, которое добавляет элемент в коллекцию.

Необязательный параметр *cKey* определяет ключ — символьную строку, которая будет идентифицировать объект. Ключи должны быть уникальны, т. е. не допускается использование двух одинаковых символьных строк как ключей для нескольких объектов коллекции.

Необязательные параметры *eBefore* и *aAfter* определяют положение добавляемого элемента в коллекции. Параметр *eBefore* указывает ключ, *перед которым* должен быть помещен добавляемый элемент, а параметр *aAfter* — ключ, *после которого* должен быть помещен добавляемый элемент. Вы можете указать одновременно оба параметра либо один из них; если вы опускаете параметр *eBefore*, то его место должно быть обозначено в списке параметров:

```
oCollection.Add('Орхидея', 'Цветок1', , 'Цветок3')
```

В листинге 11.13 показано применение метода `Add`.

```
LOCAL oItems AS Collection
oItems = NEWOBJECT("Collection")
oItems.Add("Орхидея", "Цветок2")
oItems.Add("Роза", "Цветок1", "Цветок2")
oItems.Add("Колокольчик", "Цветок3")
? oItems.Item[1],' ',oItems.GetKey(1)
? oItems.Item[2],' ',oItems.GetKey(2)
? oItems.Item[3],' ',oItems.GetKey(3)
```

В результате выполнения этого кода вы получите следующий список:

```
Роза Цветок1
Орхидея Цветок2
Колокольчик Цветок3
```

Почему вывод произведен именно в указанной последовательности? Действительно, после добавления в коллекцию строки `Орхидея` индекс этого элемента получил значение, равное единице. Но при добавлении следующего элемента было указано, что он должен быть размещен перед элементом с ключом `Цветок2`. Таким образом, элемент `Роза` получил значение индекса, равное единице, а индекс элемента `Орхидея` стал равен двум. При добавлении следующего элемента (строки `Колокольчик`) его положение явно не указано, поэтому этот элемент был добавлен в конец коллекции.

> ***ЗАМЕЧАНИЕ***
>
> Если вы хотите использовать параметр *cKey* хотя бы для одного элемента коллекции, то тогда вы должны использовать этот параметр для всех элементов коллекции; при нарушении этого требования возникает ошибка времени выполнения.

Метод *Item*

Метод позволяет обратиться к элементу коллекции по его индексу. Вот его синтаксис:

```
oCollection.Item(eIndex)
```

где *eIndex* может быть либо числовым значением индекса элемента, либо наименованием ключа. Пример применения метода показан в листинге 11.14.

```
LOCAL oItems AS Collection, oForm AS Form
oForm = NEWOBJECT("Form")
oForm.Name = "MyForm"
oCollection = NEWOBJECT("Collection")
oCollection.Add(oForm, "Это форма")
? oCollection.Item(1).Name            && Обращение по индексу
? oCollection.Item("Это форма").Name  && Обращение по ключу
? oCollection(1).Name                 && Использование метода Item
? oCollection("Это форма").Name       && по умолчанию
```

Последние четыре строки кода демонстрируют варианты обращения к элементу коллекции. Все они возвращают одинаковый результат.

Метод *GetKey*

Метод позволяет получить значение ключа по индексу элемента коллекции либо значение индекса по ключу. Если элемента с указанным индексом (или ключом) нет в коллекции, метод возвращает ноль. Вот синтаксис этого метода:

```
oCollection.GetKey(eIndex)
```

В табл. 11.4 перечислены возвращаемые методом значения.

Таблица 11.4. *Значения, возвращаемые методом* `GetKey`

Возвращаемое значение	**Значение параметра `eIndex`**	**Описание**
Ключ (строка)	Индекс (число)	Возвращает значение ключа элемента по его индексу
Индекс (число)	Ключ (строка)	Возвращает индекс элемента по значению его ключа

Таблица 11.4 *(окончание)*

Возвращаемое значение	**Значение параметра `eIndex`**	**Описание**
Пустая строка	Индекс (число)	Если указанный индекс не существует, или если элемент не имеет ключа
0	Ключ (строка)	Если указанный индекс не существует

Метод *Remove*

Метод удаляет элемент из коллекции. Вот его синтаксис:

```
oCollection.Remove(eIndex)
```

Параметр *eIndex* может быть либо числовым значением индекса элемента, либо наименованием ключа. При удалении элемента из коллекции значение свойства `Count` уменьшается на единицу; так же на единицу уменьшаются индексы для всех элементов, расположенных в коллекции после удаляемого элемента.

Применение

Применение коллекций позволит вам решить множество проблем, которые возникают при хранении данных в массивах. Простота использования и объектно-ориентированный подход являются определяющими факторами в пользу применения коллекций.

В листинге 11.15 приведен демонстрационный пример кода из справочной документации, в котором показано применение коллекции для хранения ссылок на объекты — управляющие элементы формы.

```
LOCAL loForm, loItem, lnTop
loForm = CREATEOBJECT("myForm")
lnTop=0
FOR EACH loItem IN loForm.myCollection
   TRY
      loItem.Top = lnTop
      lnTop=lnTop+20
      ? loItem.Name
   CATCH
   ENDTRY
ENDFOR
loForm.Show(1)
DEFINE CLASS myForm AS Form
   AllowOutput=.F.
   AutoCenter=.T.
   ADD OBJECT myTextBox1 AS TextBox
   ADD OBJECT myTextBox2 AS TextBox
   ADD OBJECT myButton1 AS CommandButton
   ADD OBJECT myButton2 AS CommandButton
   ADD OBJECT myCollection AS col1
ENDDEFINE
DEFINE CLASS col1 AS Collection
   PROCEDURE Init
      FOR i = 1 TO THISFORM.Objects.Count
         THIS.Add(THISFORM.Objects(m.i))
      ENDFOR
   ENDPROC
ENDDEFINE
```

Класс *Empty*

Класс `Empty` используется для создания объектов, предназначенных для хранения данных. В отличие от ранее рассмотренных классов, этот класс не может использоваться как родительский при создании пользовательских классов. Он не содержит никаких свойств и методов; вы можете добавлять новые свойства в объекты этого класса, используя функцию `ADDPROPERTY()`. Удалить добавленное в этот объект свойство можно, используя функцию `REMOVEPROPERTY()`. Объект — экземпляр класса `Empty` может быть создан командой `SCATTER ... NAME`:

```
SELECT MySessionTable
SCATTER FIELDS CookieText, SessionId NAME oCustomer ADDITIVE
```

```
? oCustomer.CookieText
? oCustomer. SessionId
```

Функции для работы с классами и объектами

Знакомство с темой создания пользовательских классов было бы не полным без рассмотрения некоторых из встроенных функций Visual FoxPro, ориентированных на работу с классами и объектами.

Функции *ADDPROPERTY()* и *REMOVEPROPERTY()*

Функция ADDPROPERTY() добавляет новое свойство или изменяет значение существующего свойства объекта Visual FoxPro во время выполнения приложения.

Синтаксис функции:

```
ADDPROPERTY(oObjectName, cPropertyName, [, eNewValue])
```

Параметры:

OObjectName

определяет имя объекта, в который добавляется свойство. Если *oObjectName* не является допустимым объектом Visual FoxPro (например, если это ActiveX), то генерируется соответствующее сообщение об ошибке.

cPropertyName

определяет имя нового свойства. Если свойство с таким именем не существует, то оно будет создано. Если свойство уже существует, и указан параметр *eNewValue*, то свойству будет присвоено значение, переданное в *eNewValue*. Если это новое свойство, и параметр eNewValue не указан, то свойству присваивается значение False.

eNewValue

изменяет значение свойства. Если выполняется попытка изменить значения защищенного (PROTECTED) или скрытого (HIDDEN) свойства, то генерируется сообщение об ошибке.

В следующем фрагменте кода показано, как добавить свойство в объект — экземпляр класса Empty:

```
oEmpty = CREATEOBJECT("Empty")
ADDPROPERTY(oEmpty, "NewProp", "Новое свойство")
```

Функция ADDPROPERTY() возвращает True в случае успешного выполнения и False при возникновении ошибки.

Функция REMOVEPROPERTY() удаляет свойства из объекта во время выполнения приложения.

Синтаксис функции:

```
REMOVEPROPERTY(oObjectName, cPropertyName)
```

Параметры:

OObjectName

определяет объект, свойство которого требуется удалить.

cPropertyName

определяет имя существующего свойства.

Удаляемые свойства должны иметь уровень видимости PUBLIC.

Функция *ACLASS()*

Функция позволяет получить сведения о именах всех классов-родителей объекта.

Синтаксис функции:

```
ACLASS(ArrayName, oRefObject)
```

Параметры:

ArrayName

одномерный массив, в который будет помещена информация.

oRefObject

имя переменной, содержащей ссылку на объект.

Если массив *ArrayName* не существует, то он будет создан. Если массив существует, то его размеры будут изменены до требуемой величины.

Функция возвращает число, содержащее количество классов-родителей (количество элементов массива *ArrayName*). Функция возвращает ноль, если ей не удается создать массив.

Имена классов-родителей располагаются в массиве в порядке старшинства, т. е. первый элемент массива будет содержать имя родительского класса объекта, а последний элемент — имя базового класса Visual FoxPro.

Функция *AMEMBERS()*

Функция помещает имена свойств, методов объекта, а также имена вложенных в него объектов в массив. Вот ее синтаксис:

```
AMEMBERS(ArrayName, oObject | cClass [, nArrayContentsID] [, cFlags])
```

Параметры:

ArrayName

определяет массив, в который будут помещены имена свойств объекта oObjectName. Если массив не существует, то он будет создан

oObject

определяет объект, информация о свойствах которого будет помещена в массив, определенный в ArrayName

cClass

определяет класс Visual FoxPro, информация о свойствах которого будет помещена в массив, определенный в ArrayName

nArrayContentsID

необязательный параметр; определяет контекст формируемого массива. Возможные значения параметра приведены в табл. 11.5.

Таблица 11.5. *Значения параметра nArrayContentsID функции AMEMBERS()*

Значение	Описание контекста
0	Формируется одномерный массив, содержащий имена свойств. Если параметр nArrayContentsID опущен, то этот контекст используется по умолчанию
1	Формируется двумерный массив, содержащий два столбца; в первый столбец заносятся наименования реквизитов объекта и имена вложенных в контейнер объектов, а во второй — одно из следующих ключевых слов: Property (свойство), Event (метод, обрабатывающий событие), Method (метод) или Object (имя вложенного объекта)
2	Формируется одномерный массив, в строки которого заносятся имена вложенных в контейнер объектов (только для объектов, добавленных в класс командой ADD OBJECT)
3	Формируется массив, содержащий четыре столбца. В первом столбце содержится имя свойства или метода, во втором — тип реквизита (Property или Method), в третьем — список получаемых методом параметров, в четвертом — справочная информация, связанная с реквизитом

CFlags

необязательный параметр; устанавливает фильтр, используемый при формировании массива *ArrayName*. Допустимые значения флагов перечислены в табл. 11.6.

Таблица 11.6. *Значения параметра cFlags функции AMEMBERS()*

cFlags	Описание
P	Защищенные (PROTECTED) свойства и методы
H	Скрытые (HIDDEN) свойства и методы
G	Открытые (PUBLIC) свойства и методы
N	Собственные свойства и методы
U	Определенные пользователем свойства и методы
C	Измененные свойства (т. е. те свойства, для которых было изменено значение по умолчанию)
I	Унаследованные свойства и методы
B	Свойства и методы базового класса
R	Свойства, доступные только для чтения

Замечание 1

Значение параметра *nArrayContentsID*, равное 3, нельзя использовать при обращении к функции из APP- или EXE-модулей.

Замечание 2

При значении параметра *nArrayContentsID*, равном 3, в качестве *oObject* можно использовать ссылку на COM-объект.

Функция *AMEMBERS()* возвращает количество свойств, методов и наименований вложенных объектов. Функция возвращает ноль, если ей не удалось создать массив *ArrayName*.

Функция *GETPEM()*

Функция возвращает текущее значение свойства или код перегруженного метода объекта. Вот ее синтаксис:

```
GETPEM(oObjectName | cClassName, cProperty | cEvent | cMethod)
```

Параметры:

OObjectName

определяет объект, значение свойства или код метода которого будут возвращены.

CClassName

определяет класс, значение свойства или код метода которого будут возвращены.

CProperty

определяет собственность (реквизит), чье значение возвращено.

CEvent

определяет событие, для которого будет возвращен код метода — обработчика этого события.

CMethod

определяет метод, код которого будет возвращен.

Следующий фрагмент кода позволяет получить код метода `Click` объекта `Text1` формы:

```
cCode = GETPEM(thisform.text1, "Click")
```

Функция *PEMSTATUS()*

Функция извлекает состояние (статус) заданного свойства, метода или вложенного объекта. Вот ее синтаксис:

```
PEMSTATUS(oObjectName, cProperty | cMethod | cObject, nAttribute)
```

Параметры:

OObjectName

определяет объект, состояние свойства, метода или вложенного объекта которого необходимо узнать

cProperty

указывает свойство, состояние которого нужно узнать

cMethod

указывает метод, состояние которого нужно узнать

cObject

указывает на вложенный объект. Например, вы можете добавить объект в объект-контейнер, используя метод `AddObject`, и затем вызвать функцию `PEMSTATUS()` для получения информации о его состоянии

`nAttribute`

число, определяющее вид операции и тип возвращаемого значения. Возможные значения параметра перечислены в табл. 11.7.

Таблица 11.7. *Значения параметра* `nAttribute` *функции* `PENSTATUS()`

Значение	Описание
0	Функция возвращает `True`, если значение свойства по умолчанию было изменено
1	Функция возвращает `True`, если свойство доступно только для чтения (ReadOnly)

Таблица 11.7 *(окончание)*

Значение	Описание
2	Функция возвращает `True`, если свойство или метод защищены (имеют уровень видимости `PROTECTED`)
3	Функция возвращает символьную строку, содержащую тип реквизита: `Property`, `Event`, `Method`, `Object`
4	Функция возвращает `True`, если свойство или метод добавлены пользователем (при помощи методов `AddProperty` или `AddMethod`, или функции `ADDPROPERTY`)
5	Функция возвращает `True`, если указанное свойство, метод или объект существуют
6	Функция возвращает `True`, если реквизит унаследован от класса-родителя

Следующий фрагмент кода позволяет получить тип реквизита объекта:

```
cType = PEMSTATUS(thisform.Text1,"Value",3) && Возвращает слово Property
```

Заключение

В этой главе вы познакомились с возможностями Visual FoxPro по созданию пользовательских классов, как в интерактивном режиме при помощи Конструктора классов, так и программно. Вы познакомились со специфическими классами `Collection` и `Empty`, а также с некоторыми встроенными функциями Visual FoxPro. Если вы еще не пытались создавать свои собственные классы, то надеюсь, что изложенный в этой главе материал поможет вам сделать это.

Коды всех рассмотренных в главе классов вы найдете на прилагаемом к книге компакт-диске.